Deep Learning for Functional Data Analysis with Adaptive Basis Layers

Junwen Yao [1] **Jonas Mueller** [2] **Jane-Ling Wang** [1]

Abstract

Despite their widespread success, the application of deep neural networks to functional data remains scarce today. The infinite dimensionality of functional data means standard learning algorithms can be applied only after appropriate dimension reduction, typically achieved via basis expansions. Currently, these bases are chosen a priori without the information for the task at hand and thus may not be effective for the designated task. We instead propose to adaptively learn these bases in an end-to-end fashion. We introduce neural networks that employ a new Basis Layer whose hidden units are each basis functions themselves implemented as a micro neural network. Our architecture learns to apply parsimonious dimension reduction to functional inputs that focuses only on information relevant to the target rather than irrelevant variation in the input function. Across numerous classification/regression tasks with functional data, our method empirically outperforms other types of neural networks, and we prove that our approach is statistically consistent with low generalization error. Code is available at: https://github.com/jwyyy/AdaFNN

1. Introduction

Deep learning has revolutionized data analysis and predictive modeling as its learned input representations capture more relevant aspects of a problem than representations based on manually selected features of the data. While the powerful capabilities of neural networks (NN) have been clearly demonstrated for vector/image/text/audio/graph data, how to best adapt these models to functional data remains under-explored. Functional data are sample of random functions. The simplest examples are random curves defined over a (univariate) real-valued interval with one curve per individual subject in our dataset. Without loss of generality we assume that the interval is $[0, 1]$. The curve for one subject is thus a random function $X(t), t \in [0, 1]$, which can be viewed as a continuous stochastic process on $[0, 1]$. Extensively studied in the statistics literature (Ferraty & Vieu, 2006; Ramsay & Silverman, 2007; Hsing & Eubank, 2015; Wang et al., 2016), functional data appear frequently in scientific studies and daily life, such as in datasets of: air pollution, fMRI scans, growth curves, and sensors like wearable devices.

A fundamental property that distinguishes functional data from other data types is that they are intrinsically infinite dimensional and generated by smooth underlying processes. While high-dimensionality poses challenges in modeling and prediction, it brings benefits when data are generated from smooth or continuous functions. This is because the observed measurements at one location t_0 can inform us the values of $X(t)$ for t at nearby locations, thereby increasing the estimation efficiency. The smoothness assumption also makes functional data resilient to noise contamination as the magnitude of the noise can be estimated and statistical methods designed for functional data can accommodate noise in the observed data.

In practical applications, we are interested in using these continuous curves $X(t)$ to infer their relationship to some response variable Y, often to predict the response. More formally, we have a dataset $\{(X_i(t), Y_i)\}_{i=1}^n$ of i.i.d. samples of $X(t)$ and the associated Y, where $X(t)$ is a mean-square continuous process defined over $t \in [0, 1]$. For each subject i in our dataset, the observations of X_i serve as a functional covariate to predict scalar response variable Y_i, which may be either continuous or discrete. In many practical applications, the continuous process $X(t)$ is only be observed on a discrete time grid $\{t_i, \ldots, t_{J+1}\}$ in each observed X_i. Assuming there is an underlying map $\mathcal{T} : X(t) \mapsto Y$, our goal is to estimate $\mathcal{T}$ from the data (e.g. using a neural network).

A common pipeline for *functional data analysis* (FDA) is to summarize the information contained in each function into a finite-dimensional vector and then carry out the analysis using existing models for the resulting multivariate data. Two popular dimension reduction approaches are Functional Principal Component Analysis (FPCA) (Besse & Ramsay, 1986; Rice & Silverman, 1991; Silverman, 1996; Yao et al.,

[1]UC Davis [2]Amazon (work done prior to joining Amazon). Correspondence to: Junwen Yao <jwyao@ucdavis.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

2005; Li et al., 2010) and preselected basis expansions using, for example, B-splines (Rice & Wu, 2001; Cardot et al., 2003a).

Existing approaches to deep learning for functional data rely on a straightforward pipeline that first applies classic functional data analysis methods and then a neural network in sequence. Rossi et al. (2002) handled functional data through discretization and functional parameterization of weight matrices, while Rossi et al. (2005) and Guss (2016) use basis function expansion to convert functional inputs into a vector form that can then be directly fed into to a standard neural network. Some universal approximation theory for functional neural networks has been established by Rossi et al. (2005) and Guss & Salakhutdinov (2019). However this two-stage fitting process in prior work is unable to fully leverage the representation-learning power and flexibility of deep learning.

In this paper, we propose to improve existing architectures by replacing these pre-specified choice of basis functions with adaptively learned bases that are implemented via micro neural networks (Lin et al., 2013) to which we back-propagate information regarding the response variable. A similar variant of our basic idea was briefly suggested by Rossi & Conan-Guez (2005), but their work never pursued the idea beyond a short comment stating it could be the possibile to implement the weight function of their functional neural network as an Multilayer Perceptron (MLP). Our design eliminates the need for a preprocessing step to convert random functions into vector inputs that otherwise typically requires a manually-prespecified choice of basis functions. Our architecture synchronizes the dimension reduction step and the nonlinear mapping step by adjusting the learned basis functions such that they only capture the information in $X(t)$ that is relevant to the output. Existing basis function representations instead seek to retain as much information about the input as possible, which may actually make supervised learning more difficult (Tishby & Zaslavsky, 2015).

Our main contribution is to propose an alternative neural architecture for *end-to-end* FDA that consists of a novel *Basis Layer* (BL) implemented via micro networks. We name the new network an *Adaptive Functional Neural Network* (AdaFNN). We study some theoretical properties of AdaFNN, establishing convergence and generalization error guarantees. Adding a BL into a neural network as feature extractors for functional inputs, the resulting model is empirically more accurate than existing methods and is simultaneously more parsimonious (meaning it requires less basis functions to model the random functions). Two types of regularizers are introduced to encourage basis orthogonality and basis sparsity. This regularization improves the resulting learned representations as well as the interpretability of the learned bases (especially when a small number of basis nodes are used). Moreover, our model can be trained end-to-end and thus composed with arbitrary differentiable operations without any alteration.

2. Related Work

2.1. Discretization of Functions

A straightforward application of neural networks to functional data is to treat the discretely observed functional values $\{X(t_1), \ldots, X(t_{J+1})\}$ as a high-dimensional vector and then input this vector into a neural network. This approach has been explored in Rossi et al. (2002); Rossi & Conan-Guez (2005); Rossi et al. (2005); Guss & Salakhutdinov (2019). An alternative common approach is to treat the discretely sampled data from subjects as time series. However this approach has several disadvantages as it does not leverage the smoothness of the underlying data-generating process $X(t)$ and relies on additional strong assumptions such as stationarity. Lastly, most time-series methods are not designed for replicate observations (here we observe numerous draws of the underlying $X(t)$, one per subject). Thus we instead review the vector-based approach below. Since the $X(t)$ process is only observed at discrete time points $\{t_j\}_{j=1}^{J+1}$, one could use the vector $v_x = [X(t_1), \ldots, X(t_{J+1})]$ as the input of a standard neural network. Using the vector v_x, the original mapping $\mathcal{T}$ can be approximated by $\mathcal{T}_{\text{finite}} : v_x \mapsto Y$. Classical results imply that $\mathcal{T}_{\text{finite}}$ can be well approximated by a neural network with a sufficient number of parameters. Furthermore, by increasing the partition resolution J, we can approximate $\mathcal{T}$ using $\mathcal{T}_{\text{finite}}$ with arbitrarily small error.

Drawbacks. In order to preserve critical information about the functional inputs, the discretization approach may require a high-dimensional vector, which hampers subsequent learning due to the curse of dimensionality. Furthermore, these discrete vector dimensions may fail to reflect the smoothness inherent to many functional covariates if they are contaminated by noise.

2.2. Basis Representation of Functions

To overcome the disadvantage of discretization, Rossi & Conan-Guez (2005) proposed to make use of the continuity of functional data and find a better finite-dimensional representation of a functional input before feeding it into a network. To be specific, let $\{\varphi_k(t)\}_{k=1}^{K}$ be a set of K continuous basis functions defined on $[0, 1]$. The task is to represent $X(t)$ using a vector $v_a = [a_1, \ldots, a_K]$ such that:

$$X(t) \approx \sum_{k=1}^{K} a_k \varphi_k(t). \tag{1}$$

Commonly used basis functions are Fourier basis functions, B-splines, or eigenfunctions obtained via spectral decompo-

sition of $\text{cov}(X(s), X(t))$. After finding a basis expansion of $X(t)$, we can use the vector v_a instead of v_x as the input to a feedforward network. Normally the dimension K of v_a is much smaller than the dimension $J+1$ of the discretized data v_x, a clear advantage. Furthermore, the basis expansion in (1) automatically produces a smooth approximation of $X(t)$, which can reduce the noise contained in v_x.

Drawbacks. While the basis representation approach in (1) could recover the underlying smooth process of functional input, it does not take advantage of the key information contained in the response Y during its dimension reduction stage. Besides, both dimension reduction of functional inputs and parameterization of weight matrices (see FMLP on p.55, Rossi & Conan-Guez (2005)) require selection of basis functions. These bases are typically selected a priori and the number of bases needs to be chosen as well. We address these questions in this paper by proposing an adaptive approach to find the optimal bases that utilizes the information on Y and the specific learning task.

2.3. Micro Network Inside of a Network

Embedding smaller neural networks within a larger overall network architecture has been previously explored. For example, the Network in Network (NIN) model of Lin et al. (2013) replaces linear convolutions by a micro MLP. Empirically, the enhanced nonlinearity improves the model's ability to extract good features. Although NIN is conceptually related to our proposal, their operations differ in two ways. In our design, a basis node in a basis layer, which is also a micro MLP, is applied to the whole input. In contrast, a NIN micro network performs local convolution operations. Second, the micro MLP in NIN takes a small region of an image as its input and this process is slid across the whole image via convolution. Our basis micro network instead takes a fixed time point t as its input and this process operates on a full functional input $[X(t_1), \ldots, X(t_{J+1})]$ via numerical integration.

3. Methodology

To address the drawbacks of discretization and basis representation, we propose a novel neural network that adaptively learn the best basis functions for supervised learning tasks with functional inputs. Figure 1 shows the basic architecture of such a network (AdaFNN), and Algorithm 1 details the network computations used to produce predictions in a forward pass. After the initial basis layer, our network shares the same structure as a standard feedforward network. Each node in a BL outputs a scalar value computed as the inner product between the input function and the corresponding basis function at that node (Figure 2). Unlike handcrafted functions used in existing methods, we parameterize each basis function with a micro neural network that takes a scalar

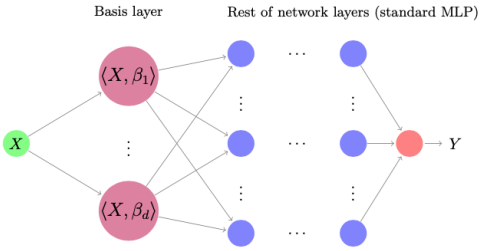

Figure 1: Neural network with our Basis Layer

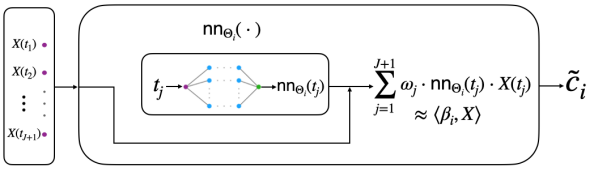

Figure 2: The i-th basis node in a Basis Layer.

t as its input and outputs the value the basis function takes at t.

In the network depicted in Figure 1, the BL consists of d basis nodes (for some user-specified value of d). Each node represents the application of some basis function $\beta_i(t)$, for $i = 1, \ldots, d$ and $t \in [0, 1]$. The value output by each basis node, i.e., the *score* of $X(t)$ with respect to the basis function $\beta_i(t)$, is computed as:

$$c_i = \langle \beta_i, X \rangle = \int \beta_i(t) \cdot X(t) \, \mathrm{d}t. \tag{2}$$

The BL outputs form a vector $c = [c_1, \ldots, c_d] \in \mathbb{R}^d$. This vector is then fed through the rest of the AdaFNN network's layers, which after the BL are all standard fully-connected layers (as in a standard MLP, where Θ denotes the weights of all layers after the BL). The output of the overall network is $\hat{Y} = \widehat{\mathcal{T}}(X) = \sigma_L\big(\cdots \sigma_1\big(W_1 c + b_1\big)\big)$, where $\sigma_1, \ldots, \sigma_L$ are the activation functions at each layer.

The bases β_i are used to project the functional input $X(t)$ to a vector representation c. While the form of these bases could in theory be selected a priori, such ad hoc selection violates the principle of end-to-end learning that drives deep learning's success. We would instead like to backpropagate information about Y through the network in order to adaptively identify optimal bases β_i. For this reason, we prefer the representation of input functions in (2) over (1), as the latter is less amenable to basis learning via backpropagation.

In the BL, each basis function $\beta_i(t)$ is itself parameterized by another neural network $\text{nn}_{\Theta_i}(t) : t \mapsto \sigma_L^i\big(\cdots \sigma_1^i\big(W_1^i t + b_1^i\big)\big)$ with parameters Θ_i consisting of weights $\{W_l^i\}_{\ell=1}^L$ and biases $\{b_l^i\}_{\ell=1}^L$. In this work, these micro networks

Algorithm 1 AdaFNN Forward Pass

Input: data $x = [X(t_1), \ldots, (X_{J+1})]$
Output: prediction $\hat{y}$
Parameters: basis layer NN $\{\Theta_i\}_{i=1,\ldots,d}$, output NN Θ, integration weights $\omega_1, \ldots, \omega_{J+1}$, e.g. for trapezoid rule with equally-spaced t_j: $\omega_j = \frac{1}{J}$ if $2 \leq j \leq J$, else $= \frac{1}{2J}$

for $i = 1$ **to** d: $\tilde{c}_i \leftarrow \sum_{j=1}^{J+1} \omega_j \cdot \text{nn}_{\Theta_i}(t_j) \cdot X(t_j)$
$\tilde{v}_c \leftarrow [\tilde{c}_1, \ldots, \tilde{c}_d] \in \mathbb{R}^d$
$\hat{y} \leftarrow \text{nn}_{\Theta}(\tilde{v}_c)$
return $\hat{y}$

nn_{Θ_i} are simply MLPs whose only input are the values t_j at which the functional covariate $X(t)$ is measured (our micro MLPs employ modern techniques such as skip connections, batch/layer normalization, and dropout). Note that these are the only locations at which we need to evaluate the learned basis function β_i in order to approximate $\langle \beta_i, X \rangle$. In practice, the integral in (2) must be approximated numerically, for example, using the rectangular or trapezoidal rule. Suppose that the domain $[0, 1]$ is equally discretized by the partition $\{t_j = (j-1)/J\}_{j=1}^{J+1}$. For each i, the score c_i is approximated by $\tilde{c}_i = \sum_{j=1}^{J+1} \omega_j \cdot \text{nn}_{\Theta_i}(t_j) \cdot X(t_j)$, where $\{\omega_j\}_{j=1}^{J+1}$ are weights used in a numerical integration algorithm. For instance, if trapezoidal rule is used in the network, then we would have $\omega_{J+1} = \omega_1 = 1/(2J)$ and $\omega_j = 1/J$ for $j = 2, \ldots, J$. Here equal spacing is not necessary. Our method works well as long as the grid is dense enough for the numerical integration to be accurate (see our provided code for a demonstration of AdaFNN with non-uniform spacing).

The loss of the network is calculated between a prediction $\hat{Y}$ and the observed response Y, and is written as $\ell(\hat{Y}, Y)$. Here we employ a standard loss function for prediction such as mean-squared-error for regression or cross-entropy for classification. The overall loss is the average loss across the whole sample (or a mini-batch of data for stochastic gradient training) and is denoted as $L(\{\Theta_i\}_{i=1}^d, \Theta) = \frac{1}{n} \sum_{i=1}^n \ell(\hat{Y}_i, Y_i)$. The micro-networks and subsequent fully-connected network layers are all simultaneously trained (end-to-end) using this single objective $L(\{\Theta_i\}_{i=1}^d, \Theta)$ applied to the output predictions. In the next section, we also consider possible addition of orthogonality or sparsity regularization to this objective.

We make three observations about the proposed model. First, it can clearly be trained end-to-end with the BL. There is no need to select and even fine tune what type of basis functions should be used in representing input functions (or weight matrices). Second, since the parameters Θ_i are updated to minimize prediction loss, our learned basis functions are likely better suited for the desired task than handcrafted basis functions chosen without this information. The experiments in Section 5 show that our BL architecture can achieve higher accuracy than existing basis expansion methods while utilizing fewer basis functions (as each individual learned basis function can capture more predictive signal). Lastly, since each micro neural network is a composition of continuous functions (as a standard MLP), all of the learned basis functions within our model will be continuous.

3.1. Regularization

Encouraging Basis Orthogonality. Without constraints, two BL nodes might learn similar basis functions and extract redundant information from the same functional input $X(t)$. To encourage different BL nodes to represent different (uncorrelated) information about the function, we can regularize them to be orthogonal. Recall that $L\big(\{\Theta_i\}_{i=1}^d, \Theta\big)$ is the loss function of a BL network. To encourage basis orthogonality, we introduce a regularization term which penalizes the cosine similarity between each pair of basis functions. The resulting loss optimized by the regularized network is

$$L_{\text{perp}}\big(\{\Theta_i\}_{i=1}^d, \Theta\big) = L\big(\{\Theta_i\}_{i=1}^d, \Theta\big) + \lambda_1 \cdot \frac{1}{\binom{d}{2}} \sum_{j \neq j'} \frac{|\langle \text{nn}_{\Theta_j}, \text{nn}_{\Theta_{j'}} \rangle|}{\|\text{nn}_{\Theta_j}\|_2 \|\text{nn}_{\Theta_{j'}}\|_2},$$

where $\lambda_1 > 0$ controls the strength of the penalty. When a BL contains many nodes, enumerating all pairs becomes computationally expensive. Instead we randomly sample a few pairs at each mini-batch update and employ their average absolute cosine similarity as a stochastic estimate of the regularizer against which we optimize network parameters.

Encouraging Basis Sparsity. We can also regularize the shape of our learned basis functions. In domain selection problems (James et al., 2009; Zhou et al., 2013; Wang et al., 2021), the response is only related to the functional input over an (a priori unknown) subset of its domain, i.e. $Y \perp\!\!\!\perp \{X(t')\}_{t' \notin \mathcal{I}} \mid \{X(t)\}_{t \in \mathcal{I}}$ for some $\mathcal{I} \subset [0, 1]$. To encourage this desired property, it is sensible to learn a basis function whose value is zero outside of $\mathcal{I}$. While the number of nonzero values taken by the basis function is hard to optimize, we can penalize their L_1 norm as a tight convex relaxation of an L_0 norm to enforce basis sparsity. The resulted loss function is

$$L_{\text{sprs}}\big(\{\Theta_i\}_{i=1}^d, \Theta\big) = L\big(\{\Theta_i\}_{i=1}^d, \Theta\big) + \lambda_2 \cdot \frac{1}{s} \sum_{i \in \mathcal{S}} \int |\beta_i(t)| \, \mathrm{d}t,$$

where $\mathcal{S} \subseteq \{1, \ldots, d\}$ indicates which subset of basis functions we wish to sparsify, s is the number of elements in $\mathcal{S}$, and $\lambda_2 > 0$ controls the strength of the L_1 penalty. Even when we are not sure whether the domain selection assumption truly hold, learning sparse basis functions via this L_1

penalty can greatly improve the overall interpretability of our prediction model (Figure 6).

4. Theoretical Analysis

Here we discuss theoretical properties of the architecture proposed in the previous section. We provide a universal approximation theorem for this design and prove it achieves low generalization error under mild regularity conditions.

Let $\mathcal{C}([0,1])$ denote the space of continuous functions defined on the compact interval $[0,1]$. Assume that the underlying mapping $\mathcal{T}: X \mapsto Y$ is a composite of a finite-dimensional linear transformation and a subsequent non-linear transformation. We can write $\mathcal{T} = h \circ g$, where $g: \mathcal{C}([0,1]) \to \mathbb{R}^q$ is a linear continuous map, and $h: \mathbb{R}^q \to \mathbb{R}$ is a non-linear continuous map. By Riesz representation theorem, there exist square-integrable function γ_i with $i = 1, \ldots, q$ such that $g(X) = [\langle \gamma_1, X\rangle, \ldots, \langle \gamma_q, X\rangle]$. The approximate network parameterized by weights $\{\Theta_i\}_{i=1}^q$ and Θ is denoted as $\widehat{\mathcal{T}}$.

Theorem 1 (Consistency of the network)**.** *With the notations defined previously and following the conventions in the literature, we assume that:*

(i) the numerical integration at each basis node can be accurately evaluated as $J \to \infty$,
(ii) each network in $\widehat{\mathcal{T}}$ can have sufficient capacity.

Then for any $\epsilon > 0$, there exists a network $\widehat{\mathcal{T}}^*$ with weights $\{\Theta_i^*\}_{i=1}^q$ and Θ^* such that

$$\sup_{f \in \mathcal{C}([0,1]), \|f\|_2 \le 1} |\widehat{\mathcal{T}}^*(f) - \mathcal{T}(f)| < \epsilon.$$

Hence, we have the following result: Let X be a continuous process defined on $[0,1]$. For any $\delta > 0$, there exists a network $\widehat{\mathcal{T}}^\star$ with weights $\{\Theta_i^\star\}_{i=1}^q$ and $\Theta^\star$ such that

$$\mathbb{P}\big(|\widehat{\mathcal{T}}^\star(X) - \mathcal{T}(X)| < \delta\big) > 1 - \delta.$$

Remark 1. Although Theorem 1 provides the consistency of the proposed architecture, it is not equivalent to identifying each true basis function consistently. The reason is that the individual basis functions are not identifiable since there are multiple ways to parameterize one map.

Remark 2. To make adequate predictions, the number of basis nodes d in AdaFNN should be sufficiently larger than the dimensionality q of $g(X)$, introduced in the assumptions of Theorem 1. In practice, we could also vary the choice of the number of basis nodes to find out which yields the best performance (lowest validation loss or fewest bases with low validation loss). Once the training is done, by investigating the learned bases, one can decide which ones seem to be relevant and use those as the basis functions to re-train a smaller subsequent network.

Next, we prove the proposed architecture can achieve small generalization error. Let $S = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be n i.i.d. copies of (X, Y), and there exist two constants $M_1, M_2 > 0$ such that both $\sup_{t\in[0,1]} |X(t)| \le M_1$ and $|Y| \le M_2$ hold almost surely. Let $\widehat{\mathcal{T}}_\Theta$ be a model proposed in Section 3 with its architecture fixed. In a slight abuse of notation, we use Θ to denote all the weights, including $\{\Theta_i\}_{i=1}^d$ and Θ, used in $\widehat{\mathcal{T}}_\Theta$. We use $\ell\big(\widehat{\mathcal{T}}_\Theta(X), Y\big)$ to denote the loss function. The population risk is defined as $r(\Theta) = \mathbb{E}\big[\ell\big(\widehat{\mathcal{T}}_\Theta(X), Y\big)\big]$, and the empirical risk is $r_n(\Theta) = \frac{1}{n}\sum_{i=1}^n \ell\big(\widehat{\mathcal{T}}_\Theta(X_i), Y_i\big)$. Usually the weights Θ are estimated via $\widehat{\Theta}$ produced by a random algorithm A based on the sample S. The generalization error is defined as $\big|\mathbb{E}_{S,A}[r(\widehat{\Theta}) - r_n(\widehat{\Theta})]\big|$. Suppose that we use a variant of stochastic gradient descent to train our model. At the t-th iteration ($1 \le t \le T$), the weights are updated with $\widehat{\Theta}_{t+1} = \widehat{\Theta}_t - \alpha_t \nabla_\Theta \ell\big(\widehat{\mathcal{T}}_\Theta(X_{i_t}), Y_{i_t}\big)\big|_{\Theta=\widehat{\Theta}_t}$, where the indices i_t are randomly chosen (e.g., uniformly), and the learning rate α_t is monotonically non-increasing with $\alpha_t \le c/t$ for some fixed constant $c > 0$. The following result is an application of Theorem 3.12 in Hardt et al. (2016).

Theorem 2 (Small generalization error)**.** *Assume that*

(i) the weight Θ is restricted on some compact region,
(ii) the loss function $\ell(\cdot,\cdot)$ and its gradient $\nabla\ell(\cdot,\cdot)$ are both Lipschitz.

Then for every pair of observation (X, Y), both $\ell\big(\widehat{\mathcal{T}}_\Theta(X), Y\big)$ and $\nabla_\Theta \ell\big(\widehat{\mathcal{T}}_\Theta(X), Y\big)$ are Lipschitz with respect to Θ. Hence, there exists some constant $c > 1$ such that

$$\big|\mathbb{E}_{S,A}[r(\widehat{\Theta}) - r_n(\widehat{\Theta})]\big| \lesssim \frac{T^{1-1/c}}{n}.$$

Remark 3. Assumption (i) in Theorem 2 is not restrictive since we can convert a constraint optimization problem to an equivalent penalized problem. In practice, we could minimize the regularized empirical risk, i.e., $r_n(\Theta) + \rho\|\mathrm{vec}(\Theta)\|_2$, where $\mathrm{vec}(\Theta)$ is the vectorized weights Θ, and $\rho > 0$ is a tuning parameter.

5. Experiments

Throughout, our experiments focus on methods that can handle general functional inputs, rather than approaches tailored for specific tasks like predicting future observations. The proposed AdaFNN network is compared to three baseline models: 'Raw data (#) + NN' (Rossi et al., 2002), 'B-spline (#) + NN' (Rossi et al., 2005), and 'FPCA$_p$ + NN'(Rossi et al., 2005). Here the integer # denotes the input dimension of each network. The subscript p of 'FPCA' is the fraction of variation explained (FVE) used in selecting the number of principal components. The latter two baselines (B-spline and FPCA) have their roots in the field of functional data analysis (Cardot et al., 1999; 2003b; Müller &

Stadtmüller, 2005; Dou et al., 2012) and classification tasks (Müller, 2005; Leng & Müller, 2006; Song et al., 2008; Chiou, 2012).

The first baseline simply discretizes the raw functional input as a vector that is fed into the network, so the dimension of the input vector is the number of points t at which $X(t)$ has been observed. The other two baseline models consist of two steps. The first step involves transforming the functional input into a vector of scores from its B-spline or FPCA expansion (cf. (1)). The resulting vector representation is then fed as input into a network in the second step.

The number of bases used in the two baseline models is often determined by how well a function can be represented in the functional space spanned by these basis functions. For example, when using B-splines, we look at how well the selected spline functions can capture the trend of the raw data. A small number of B-splines may not be able to recover the trajectory very well. However, too many B-splines might overfit the functions. The choice of the number of principal components is based on the desired FVE by these principal components. Often, a sizeable FVE is expected, e.g., 90% to 99%. Similar to the selection of B-splines, we should not simply choose an FVE as large as possible. Large FVE may include (functional) principal components whose corresponding eigenvalues are very small, hence difficult to estimate. A good rule of thumb is to use the first few components that capture the bulk of the variation.

We write 'AdaFNN (λ_1,λ_2)' to indicate the level of regularization, where λ_1 (and λ_2) controls the degree of orthogonality (and L_1) regularization. AdaFNN was trained with 9 different combinations of the orthogonal regularization penalty $\lambda_1 \in \{0, 0.5, 1\}$ and L_1 regularization penalty $\lambda_2 \in \{0, 1, 2\}$. The performance of each configuration is reported for all simulations and real data tasks. Throughout we use * to indicate the λ_1, λ_2 values that performed best on the validation data, as these are the hyperparameter values that would be typically used in practice.

All models, including AdaFNN and the other NN baselines, employ the same architecture and training hyperparameters. A network with 3 hidden layers and 128 fully connected nodes per layer was used for Tasks 1-7 (real data) and our simulation studies. For Tasks 8 and 9 with small sample sizes, we used a smaller network with 2 hidden layers and 64 nodes and added dropout during training. All networks were trained up to 500 epochs (with 200-epoch early stopping patience) using mini-batches of size 128. AdaFNN merely uses 2 bases in simulation Cases 1 & 4, 3 bases in simulation Cases 2 & 3, and 4 bases in our applications to all 9 prediction tasks with real data. In contrast, we allowed the B-spline/FPCA baselines to either rely on a similar number or more basis functions since these models cannot optimize each of their bases (e.g. $\text{FPCA}_{0.99}$ often selected more than 10 principal components).

5.1. Simulation Studies

We demonstrate through simulations that baseline functional neural network models may miss relevant information but AdaFNN is able to capture the true signal while relying on fewer basis functions than the baselines. Four different simulation settings were considered, each is purposefully designed to illustrate a particular conceptual shortcoming of one of the baseline methods (with an extra fifth setting to highlight the utility of our proposed regularization).

We first describe the underlying data-generating process in each of the four settings. For $t \in [0, 1]$, define $\phi_1(t) = 1$ and $\phi_k(t) = \sqrt{2}\cos((k-1)\pi t)$, $k = 2, \ldots, 50$. Consider the process $X(t) = \sum_{k=1}^{50} c_k \phi_k(t)$, where $c_k = z_k r_k$, and r_k are i.i.d. uniform random variables on $[-\sqrt{3}, \sqrt{3}]$. The actual observations for $X_i(t)$ is the discrete data $\{X_i(t_j), j = 1, \ldots, 51\}$. We report the *mean squared prediction error* (MSE) achieved by each method on the test data.

Case 1: We set: $z_1 = 20, z_2 = z_3 = 5$, and $z_k = 1$ for $k \geq 4$. The response is $Y = c_3^2$, that is, $Y = (\langle X, \phi_3 \rangle)^2$, where the function ϕ_3 corresponds to the true predictive *signal*. This case is designed to show that a small FVE in FPCA may not suffice to capture the relevant signal.

Case 2: We set: $z_1 = z_3 = 5, z_5 = z_{10} = 3$, and $z_k = 1$ for other k. The response is $Y = c_5^2 = (\langle X, \phi_5 \rangle)^2$. Here the function ϕ_5 corresponds to the true predictive *signal* and is more complex than ϕ_3. The squared operation ensures a nonlinear relationship between the response and functional covariate. This case is designed to show that a small number of B-splines may not suffice to represent the $X(t)$ information relevant to the response.

Case 3: In Cases 1 & 2, both the response Y is free of noise and the functional input $X(t)$ is not contaminated by measurement errors. However, this setup is rarely realistic in practice. The goal here is to evaluate functional estimators in the presence of both outcome noise and measurement errors in $X(t)$. We use the same model as Case 2 except that a mean zero Gaussian noise is added to the response, and the observation of $X(t)$ at each time point is perturbed by an additive mean zero Gaussian measurement error. The signal-to-noise ratios (SNR), defined as

$$\text{SNR}(X) = \frac{\sqrt{\int_0^1 (X(t))^2 \, \mathrm{d}t}}{\text{standard deviation of measurement error}}$$

for the functional input (due to $\mathbb{E}[X(t)] = 0$), is $\sqrt{10}$ to 1.

Case 4: This case studies how well AdaFNN captures multiple signals and its application in domain selection. The functional covariates $X_i(t)$ are generated similarly as in Cases

1 & 2, except that in Case 4: z_i are all taken to be 1. Two signals β_1 and β_2 are chosen as: $\beta_1(t) = (4-16t)\cdot 1\{0 \le t \le 1/4\}$ and $\beta_2(t) = (4-16|1/2-t|)\cdot 1\{1/4 \le t \le 3/4\}$. The response is $Y = \langle \beta_2, X\rangle + (\langle \beta_1, X\rangle)^2$. Centered Gaussian noise is added to Y, and $X(t)$ is also contaminated by measurement error.

Case 5: The same setup as Case 4, but now with double the noise variance in Y (used to highlight our regularization).

Table 1: Test-set MSE of predictions in simulation study. For each case, the asterisk indicates the best AdaFNN hyperparameters on the validation set, and the method with the best test MSE is marked in bold.

METHOD	CASE 1	CASE 2	CASE 3	CASE 4
RAW DATA (51) + NN	0.015	0.038	0.275	0.334
B-SPLINE (4) + NN	0.050	0.984	0.971	0.369
B-SPLINE (15) + NN	0.013	0.019	0.206	0.251
$\text{FPCA}_{0.9}$ + NN	0.917	0.023	0.134	0.855
$\text{FPCA}_{0.99}$ + NN	0.003	0.036	0.239	0.667
ADAFNN (0.0, 0.0)	**0.001***	**0.003**	0.979	**0.193***
ADAFNN (0.0, 1.0)	0.995	0.007	0.978	0.982
ADAFNN (0.0, 2.0)	0.996	0.992	0.978	0.981
ADAFNN (0.5, 0.0)	0.004	0.005*	0.137*	0.571
ADAFNN (0.5, 1.0)	0.983	0.005	0.978	0.590
ADAFNN (0.5, 2.0)	0.134	0.008	0.978	0.981
ADAFNN (1.0, 0.0)	1.000	0.004	**0.127**	0.196
ADAFNN (1.0, 1.0)	0.009	0.006	0.974	0.606
ADAFNN (1.0, 2.0)	0.051	0.009	0.978	0.981

Table 1 reports the MSEs for all methods. Under columns Cases 1 & 2, we see that AdaFNN exhibits the best performance in both settings. The performance of the baseline models is mixed. The top two principal components (with FVE at least 90%) are not able to detect any useful predictive signal in the data under Case 1, same with four B-splines bases in the simulation under Case 2. Thanks to its success in capturing the true basis function, AdaFNN performs strongly in both simulation settings. It is interesting to observe that each fitted basis function contains a fraction of the true signal function, and together they are able to recover it (Figures 3 and 4). That is, the true signal function can be represented as a linear combination of the fitted bases.

For Case 3, Table 1 (column 'Case 3') shows that all methods performed worse with noise added, but AdaFNN with orthogonality regularization remains superior to other methods. On the other hand, the L_1 regularizer is not helpful in Case 3. This is expected, because the true signal ϕ_5 does not have any zero region in its domain. Note that feeding the raw data as a vector into neural networks performs comparably to the two functional baseline models in the noiseless simulations. However, in noisy settings (Cases 3 & 4), this approach is inferior to the other two baseline models. This demonstrates the importance of exploiting functional properties whenever the input is a smooth function. The performance of the functional 'B-spline + NN' and 'FPCA + NN' approaches is mixed; each has its own advantage over the other in different scenarios.

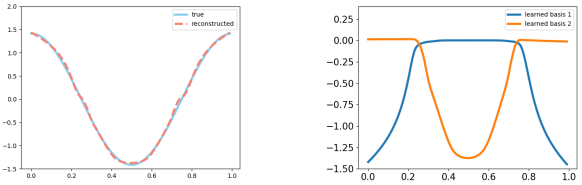

Figure 3: The left plot shows the true signal ϕ_3 (solid) and the reconstructed signal $\hat{\phi}_3$ (dashed) from AdaFNN$(0, 0)$ (the regularization values with best validation MSE) in Case 1. The right plot shows each learned bases $\hat{\beta}_1$ and $\hat{\beta}_2$ from the same experiment. Note that: $\phi_3 \approx \hat{\phi}_3 = \hat{\beta}_2 - \hat{\beta}_1$.

For Case 4, Table 1 (column 'Case 4') shows that AdaFNN outperforms the other methods in this setting. At the same time, the proposed method also learns meaningful bases that correctly identify the relevant domain of interest (Figure 5). That on $[3/4, 1]$ both β_1 and β_2 are zero implies that the values of $X(t)$ over $[3/4, 1]$ have no effect on the response. None of the baseline methods is able to show this information, and thus AdaFNN is not only more accurate than existing models, but also more *interpretable*. In summary, while some baseline methods perform well in certain simulation settings, none of these methods can achieve consistently strong performance like AdaFNN across all cases.

A final simulation, Case 5, demonstrates the utility of our proposed regularizationn. In this case, Table 2 shows that AdaFNN$(0.5, 0)$ and AdaFNN$(0, 0.1)$ clearly outperform AdaFNN without regularization, as well as all other methods (despite our regularized AdaFNN using **only 2 bases**, under which the other methods performed very poorly). Without any regularization, AdaFNN$(0, 0)$ learns **2 very similar bases** (left plot in Figure 6), while using either one of our proposed regularizers helps AdaFNN recover the true underlying bases (middle/right plots in Figure 6) and greatly improves its predictive performance.

5.2. Application to Real Functional Datasets

Next, we evaluate the performance of AdaFNN and other neural FDA methods in nine different regression and classification tasks, using four datasets. In regression tasks, the performance is again measured by MSE, while the performance is measured by the *area under the ROC curve* (ROC AUC) in classification tasks. Since our simulations show that B-spline (4) and $\text{FPCA}_{0.9}$ empirically underperform (Table 1), we subsequently only consider the use of B-spline (15) and $\text{FPCA}_{0.99}$ on real data.

Electricity Data: Electricity consumption readings for 5567 London homes, where each household's electricity usage is recorded every half hour (UK Power Networks,

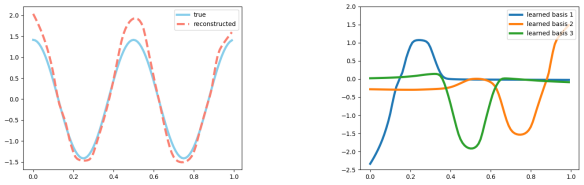

Figure 4: The left plot shows the true signal ϕ_5 (solid) and the reconstructed signal $\hat{\phi}_5$ (dashed) from AdaFNN$(0.5, 0)$ (the regularization values with best validation MSE) in Case 2. The right plot shows each learned bases $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$ from the same experiment. Note that: $\phi_5 \approx \hat{\phi}_5 = \hat{\beta}_3 - \hat{\beta}_1 - \hat{\beta}_2$.

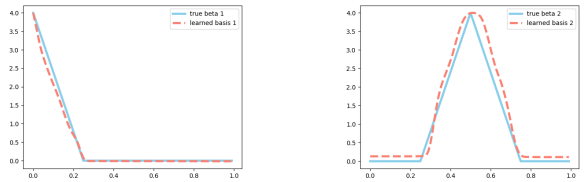

Figure 5: The left plot shows the true signal β_1 (solid) and a (scaled) learned signal $\hat{\beta}_1$ (dashed) from AdaFNN$(0, 0)$ (the regularization values with best validation MSE) in Case 4. The right plot shows β_2 and a (scaled) $\hat{\beta}_2$ from the same experiment.

2015). The functional covariate $X(t)$ is defined as the 48 measurements per household that constitute one day's electricity usage curve. Based on this $X(t)$, we consider four prediction tasks with different response variables:

1. Predict a household's total electricity consumption in the next week (week 2) based on its $X(t)$. [*regression*]
2. Predict a household's total electricity consumption in a later week (week 5) based on its $X(t)$. [*regression*]
3. Predict whether a household's morning (6am-12pm) electricity consumption in week 5 exceeds a certain threshold based on its $X(t)$. [*classification*]
4. Predict whether a household's consumption during the day (8am-17pm) exceeds night usage (17pm-12am) by a threshold in week 5 based on its $X(t)$. [*classification*]

The threshold values in tasks 3 and 4 are selected to ensure approximate class-balance in these classification problems. Because household consumption behavior is likely to change over a longer period, Tasks 2-4 are expected to be harder than Task 1. Results for these tasks are reported in columns Tasks 1-4 in Table 3.

Wearable Device Data: This data consist of wearable device data from the National Health and Nutrition Examination Survey (NHANES) (NCHS, CDC 2020). Each subject in the study wore a device that continuously measures the intensity level of their physical activities within one week. The functional covariate $X(t)$ is the average activity levels every 30 minutes for one full day, resulting in a curve of 48 observations per subject. We examine whether physical activities are predictive of various health outcomes:

Table 2: Test-set MSE of predictions in Case 5. AdaFNN with active regularization is highlighted in bold.

METHOD	NO. BASES	MSE
RAW (51) + NN	51	0.339
B-SPLINE (4) + NN	4	0.382
B-SPLINE (15) + NN	15	0.257
$\text{FPCA}_{0.9}$ + NN	20	0.807
$\text{FPCA}_{0.99}$ NN	28	0.693
AdaFNN$(0, 0.0.0)$	2	0.598
AdaFNN$(0.5, 0.0)$	2	**0.231**
AdaFNN$(0.0, 0.1)$	2	**0.207**

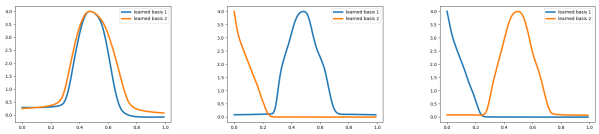

Figure 6: The (scaled) bases under simulation Case 5 learned by: AdaFNN$(0, 0)$ on the left, AdaFNN$(0.5, 0)$ in the middle, and AdaFNN$(0, 0.1)$ on the right.

5. Predict whether a subject has diabetes. [*classification*]
6. Predict if subject feels chest pain. [*classification*]
7. Predict whether a subject experiences shortness of breath on stairs. [*classification*]

Tasks 6 and 7 aim at predicting a subject's cardiovascular health. Results for Tasks 5-7 are reported in column Task 5 to column Task 7 in Table 3.

Mexfly and Medfly Data: The final two datasets pertain to Mexican fruit flies (Mexfly) (Carey et al., 2005) and Mediterranean fruit flies (Medfly) (Chiou et al., 2003), recording the number of eggs laid daily for each fly. Our task is to use early trajectories of egg-laying (daily number of eggs laid) to predict the *lifetime reproduction*, defined as the total number of eggs laid by the fly over its lifetime:

8. Predict lifetime reproduction of a Mexfly using its egg-laying curve $X(t)$ from day 1 to 30. [*regression*]
9. Predict lifetime reproduction of a Medfly using its egg-laying curve $X(t)$ from day 1 to 20. [*regression*]

The choice of the thresholds, day 20 and 30, is motivated by predicting lifetime reproduction based on early reproduction pattern in pre-peak period (peak usually occurs after 20 or 30 days depending on the species). Results are presented in Table 3 in columns Tasks 8 and 9.

Results. Empirically, AdaFNN performs better than all baseline methods in all 9 prediction tasks, demonstrating its advantage for diverse forms of real functional data spanning

Table 3: Comparing test-set performance of different methods' predictions on 9 functional datasets (MSE in regression, $1 - \text{AUC}$ in classification). For each dataset, the asterisk indicates which AdaFNN hyperparameters performed best on the validation set, and the best performing method on the test data is indicated in bold.

METHOD	TASK 1	TASK 2	TASK 3	TASK 4	TASK 5	TASK 6	TASK 7	TASK 8	TASK 9
RAW DATA (48) + NN	0.099	0.284	0.124	0.296	0.380	0.488	0.472	0.406	0.373
B-SPLINE (15) + NN	0.094	0.306	0.137	0.326	0.335	**0.477**	0.429	0.413	0.387
$\text{FPCA}_{0.99}$ + NN	0.119	0.339	0.143	0.306	0.363	0.493	0.431	0.429	0.378
ADAFNN (0.0, 0.0)	**0.084***	0.290*	0.129*	0.311	0.365	**0.477**	**0.410***	0.377*	0.375
ADAFNN (0.0, 1.0)	0.094	0.276	0.126	0.327	0.561	0.479*	0.498	0.374	0.392
ADAFNN (0.0, 2.0)	0.097	0.276	0.129	0.324	0.596	0.481	0.473	0.381	0.445
ADAFNN (0.5, 0.0)	0.108	**0.260**	0.130	0.310*	0.380*	0.490	**0.410**	0.376	**0.368***
ADAFNN (0.5, 1.0)	0.089	0.279	0.126	0.324	0.616	0.486	0.494	**0.362**	0.413
ADAFNN (0.5, 2.0)	0.098	0.280	0.128	0.345	0.392	0.509	0.444	0.373	0.450
ADAFNN (1.0, 0.0)	**0.084**	0.288	**0.118**	**0.294**	**0.339**	0.485	0.413	0.378	0.406
ADAFNN (1.0, 1.0)	0.097	0.282	0.133	0.320	0.651	0.502	0.456	0.371	0.394
ADAFNN (1.0, 2.0)	0.092	0.279	0.127	0.326	0.371	0.510	0.414	0.374	0.416

regression/classification problems. In contrast, none of the baseline methods consistently outperformed all other baselines across these tasks. Basis orthogonality and sparsity were used to improve learned representations and possibly get a better fit of the data (but like all regularization, the effectiveness of our proposed regularizers varies from dataset to dataset). Many of the best reported results are from AdaFNN with penalty $\lambda_1 > 0$, demonstrating that our orthogonal regularization technique improves the learned functional representations. While $\lambda_2 > 0$ only produces the most accurate AdaFNN model for one of the tasks, the L_1 penalty can remain useful for interpretability of the model. As with all regularizers, the optimal degree of regularization to employ also varies from dataset to dataset. By leveraging its superior representational capabilities, AdaFNN is also able to achieve superior accuracies with fewer bases than the B-spline + NN or FPCA + NN baselines.

6. Conclusion

This work presents a new approach to adapt representation learning techniques for functional data. Our proposed architecture does not require handcrafted bases to handle functional inputs, and learns the optimal bases for a particular dataset in an end-to-end manner. The Basis Layer compresses functional covariates in a linear fashion into a low-dimensional vector that reflects only those factors of variation most relevant to the response value. Traditional dimension reduction techniques like FPCA instead attempt to capture all variation in the functional input itself, regardless of its relationship to the response. There are many disadvantages to retaining global information about $X(t)$ rather than merely what is needed to infer Y, some of which are outlined in the *information bottleneck* principle of Tishby & Zaslavsky (2015).

Note that AdaFNN can be easily extended to vector-valued functional data, where either $t \in \mathbb{R}^p$ or $X(t) \in \mathbb{R}^p$ for $p > 1$. In the former case, each basis layer micro NN simply operates on vector-valued inputs t, while in the latter case, these micro NN would have larger output layers to produce vectors rather than scalars. Furthermore, our method can also be applied to data with both multiple functional covariates (can simply employ separate basis layers for each and pool their outputs) as well as auxiliary vector covariates in addition to $X(t)$ (can simply concatenate our basis layer output $\tilde{v}_c$ with these additional covariates before it is fed into the subsequent feedforward network).

As previously mentioned, AdaFNN is also directly applicable to non-uniform observations of $X(t)$, where the t_j are not equally spaced, although such cases require proper selection of the integration weights ω_j. However, successful application of AdaFNN to sparsely observed functional data, where the underlying process $X(t)$ is only observed at few locations t_j, remains nontrivial and likely requires improved numerical integration strategies as well as stronger inductive bias in the micro NN architecture (Gunter et al., 2014). Nonetheless, we expect our adaptive Basis Layer will find broad applicability as a general-purpose representation learning tool for domains with functional data such as wearable devices, climatology, genomics, or neuroimaging.

7. Acknowledgements

We would like to thank the reviewers for their constructive feedback. The research of Jane-Ling Wang was supported by NSF grant 19-14917.

References

Besse, P. and Ramsay, J. O. Principal components analysis of sampled functions. *Psychometrika*, 51(2):285–311, 1986.

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

Cardot, H., Ferraty, F., and Sarda, P. Spline estimators for the functional linear model. *Statistica Sinica*, pp. 571–591, 2003a.

Cardot, H., Ferraty, F., and Sarda, P. Spline estimators for the functional linear model. *Statistica Sinica*, pp. 571–591, 2003b.

Carey, J. R., Liedo, P., Müller, H.-G., Wang, J.-L., Senturk, D., and Harshman, L. Biodemography of a long-lived tephritid: reproduction and longevity in a large cohort of female mexican fruit flies, anastrepha ludens. *Experimental Gerontology*, 40(10):793–800, 2005.

Chiou, J.-M. Dynamical functional prediction and classification, with application to traffic flow prediction. *Annals of Applied Statistics*, 6(4):1588–1614, 2012.

Chiou, J.-M., Müller, H.-G., Wang, J.-L., and Carey, J. R. A functional multiplicative effects model for longitudinal data, with application to reproductive histories of female medflies. *Statistica Sinica*, 13(4):1119, 2003.

Dou, W. W., Pollard, D., and Zhou, H. H. Estimation in functional regression for general exponential families. *Annals of Statistics*, 40(5):2421–2451, 2012.

Ferraty, F. and Vieu, P. *Nonparametric Functional Data Analysis: Theory and Practice*. Springer Science & Business Media, 2006.

Gunter, T., Osborne, M. A., Garnett, R., Hennig, P., and Roberts, S. J. Sampling for inference in probabilistic models with fast bayesian quadrature. In *Advances in Neural Information Processing Systems*, 2014.

Guss, W. H. Deep function machines: Generalized neural networks for topological layer expression. *arXiv preprint arXiv:1612.04799*, 2016.

Guss, W. H. and Salakhutdinov, R. On universal approximation by neural networks with uniform guarantees on approximation of infinite dimensional maps. *arXiv preprint arXiv:1910.01545*, 2019.

Hardt, M., Recht, B., and Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, volume 48, pp. 1225–1234. JMLR, 2016.

Hsing, T. and Eubank, R. *Theoretical Foundations of Functional Data Analysis, with an Introduction to Linear Operators*, volume 997. John Wiley & Sons, 2015.

James, G. M., Wang, J., Zhu, J., et al. Functional linear regression that's interpretable. *Annals of Statistics*, 37 (5A):2083–2108, 2009.

Leng, X. and Müller, H.-G. Classification using functional data analysis for temporal gene expression data. *Bioinformatics*, 22(1):68–76, 2006.

Li, Y., Hsing, T., et al. Uniform convergence rates for nonparametric regression and principal component analysis in functional/longitudinal data. *Annals of Statistics*, 38 (6):3321–3351, 2010.

Lin, M., Chen, Q., and Yan, S. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

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

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

National Center for Health Statistics (NCHS), Centers for Disease Control and Prevention (CDC). National health and nutrition examination survey data., 2020.

Ramsay, J. O. and Silverman, B. W. *Applied Functional Data Analysis: Methods and Case Studies*. Springer, 2007.

Rice, J. A. and Silverman, B. W. Estimating the mean and covariance structure nonparametrically when the data are curves. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(1):233–243, 1991.

Rice, J. A. and Wu, C. O. Nonparametric mixed effects models for unequally sampled noisy curves. *Biometrics*, 57(1):253–259, 2001.

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

Rossi, F., Conan-Guez, B., and Fleuret, F. Functional data analysis with multi layer perceptrons. In *Proceedings of the 2002 International Joint Conference on Neural Networks.*, volume 3, pp. 2843–2848. IEEE, 2002.

Rossi, F., Delannay, N., Conan-Guez, B., and Verleysen, M. Representation of functional data in neural networks. *Neurocomputing*, 64:183–210, 2005.

Silverman, B. W. Smoothed functional principal components analysis by choice of norm. *Annals of Statistics*, 24 (1):1–24, 1996.

Song, J. J., Deng, W., Lee, H.-J., and Kwon, D. Optimal classification for time-course gene expression data using functional data analysis. *Computational Biology and Chemistry*, 32(6):426–432, 2008.

Tishby, N. and Zaslavsky, N. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop*, pp. 1–5. IEEE, 2015.

UK Power Networks. Smartmeter energy consumption data in london households, 2015.

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

Wang, Q., Lu, Y., Zhang, X., and Hahn, J. Region of interest selection for functional features. *Neurocomputing*, 422: 235–244, 2021.

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

Zhou, J., Wang, N.-Y., and Wang, N. Functional linear model with zero-value coefficient function at sub-regions. *Statistica Sinica*, 23(1):25, 2013.