Deep Learning with Functional Inputs

Barinder Thind
Department of Statistics & Actuarial Science
Simon Fraser University

Kevin Multani
Department of Physics
Stanford University

Jiguo Cao
Department of Statistics & Actuarial Science
Simon Fraser University

Abstract

We present a methodology for integrating functional data into deep neural networks. The model is defined for scalar responses with multiple functional and scalar covariates. A by-product of the method is a set of dynamic functional weights that can be visualized during the optimization process. This visualization leads to a greater interpretability of the relationship between the covariates and the response relative to conventional neural networks. The model is shown to perform well in a number of contexts including prediction of new data and recovery of the true underlying relationship between the functional covariate and scalar response; these results were confirmed through real data applications and simulation studies. An R package (**FuncNN**) has also been developed on top of `Keras`, a popular deep learning library – this allows for general use of the approach.

Keywords: Functional Data Analysis, Neural Networks, Machine Learning, Prediction

1 Introduction

Functional data analysis (FDA) is a growing statistical field for analyzing curves, surfaces, or any multidimensional functions, in which each random function is treated as a sample element (Ramsay and Silverman, 2005; Ferraty and Vieu, 2006). Functional data is found commonly in many applications such as time-course gene expressions and brain scan images (Tian, 2010). The growing ecosystem of deep learning methodologies has thus far largely excluded the usage of such functional covariates. With the advent and rise of functional data analysis, it is natural to extend neural networks and all of their recent advances to the functional space. In this article, we present a novel method to integrate functional data into a neural network architecture. Specifically, we study the case where the learning task is scalar response prediction.

Let Y be a scalar response variable, and $X(t)$, $t \in \mathcal{T}$, be the functional covariate. Several models have been proposed to predict the scalar response Y with the functional covariate $X(t)$. For instance, when the scalar response Y follows a normal distribution, the conventional functional linear model is defined as:

$$E(Y|X) = \alpha + \int_{\mathcal{T}} \beta(t)X(t)\,\mathrm{d}t, \tag{1}$$

where α is the intercept term, and $\beta(t)$ is the functional coefficient which represents the cumulative linear effect of $X(t)$ on Y (Cardot et al., 1999). This model was extended to the general functional linear model:

$$E(Y|X) = g\left(\alpha + \int_{\mathcal{T}} \beta(t)X(t)\,\mathrm{d}t\right)$$

when the scalar response Y follows a general distribution in an exponential family, where $g(\cdot)$ is referred to as the link function and has a specific parametric form for the corresponding distribution of Y (Müller and Stadtmüller, 2005). When this link function $g(\cdot)$ has no parametric form, the model is called the functional single index model (Jiang and Wang, 2011). Other predictive methods in the realm of FDA are related to various estimation methods of $\beta(t)$. For example, a partial least squares approach was proposed by Preda et al. (2007) where an attempt was made to estimate $\beta(t)$ such that it maximized the covariance between the response, Y and the functional covariate, $X(t)$. Ferraty and Vieu (2006) proposed a non-parametric model:

$$E(Y|X) = r\left(X(t)\right),$$

where $r(\cdot)$ is a smooth non-parametric function that is estimated using a kernel approach. Another model, which serves as an extension to the previous, is the semi-functional partial linear model (Aneiros-Pérez and Vieu, 2006) defined as:

$$E(Y|X) = r\left(X(t)\right) + \sum_{j=1}^{J} w_j Z_j,$$

where $Z_j, j = 1, \ldots, J$, is the scalar covariate that is observed in the usual multivariate case, and the function $r(\cdot)$ is also estimated with no parametric form using kernel methods.

All of the functional models above have been shown to have some level of predictive success. However, we show that the neural network in this article outperforms these models. We propose a novel methodology for deep network structures in a general regression framework for longitudinal data. With respect to neural networks, we have seen a growing number of innovations in architecture, some of which have resulted in previous benchmarks being eclipsed. For example, Krizhevsky et al. (2012) won the ImageNet Large-Scale Visual Recognition Challenge in 2012 bettering next best approach by a 10% increase in predictive accuracy. Rossi and Conan-Guez (2005) proposed a neural network in which they used a single functional variable for classification problems. He et al. (2016) introduced residual neural networks, which allowed for circumvention of vanishing gradients – an innovation that carved a path for networks with exponentially more layers thus further improving error rates. As the architectures have become more complex, it has become an increasingly difficult task to make sense of the network parameters, particularly in the case of multilayer perceptrons as defined in (Tibshirani et al., 2009). On the other hand, conventional linear regression models have a relatively clear interpretation of the parameters estimates (Seber and Lee, 2012) which can be extended to the functional counterpart where the coefficient parameters being estimated are functions $\beta_k(t), k = 1, \ldots, K$, rather than a vector of scalar values. This paper details an approach that makes the functional coefficient traditionally found in the regression model readily available from the neural network process in the form of functional weights; the expectation is that this increases the interpretability of the neural networks while maintaining the superior predictive power.

Our paper has three major contributions. First, we introduce the general framework of functional neural networks (FNNs) with a methodology that allows for deep architectures with multiple functional and scalar covariates, the usage of modern optimization techniques (Kingma and Ba, 2014; Ruder, 2016), and hyperparameters such as early stopping (Yao et al., 2007) and dropout (Srivastava et al., 2014), along with some justifications that underpin the approach; we provide evidence for similar or improved performance on various examples despite a sizeable reduction in parameter counts in FNNs versus other neural networks. Second, we introduce functional weights that are smooth functions of time and can be easier to interpret than the vectors of parameters estimated in the conventional neural network due to visualization. This is exemplified in our applications and simulations. Finally, branching off this work is an R package (**FuncNN**) developed on top of a popular deep learning library (`Keras`) that allows users to apply the proposed method easily on their own data sets (Thind et al., 2020).

The rest of our paper is organized as follows. We first introduce the methodology for functional neural networks in Section 2. Additionally, commentary is provided on the weight initialization and the hyperparameters of these networks. Then, Section 3 provides results from real world examples; this includes prediction comparisons among a number of methods for multiple data sets. In Section 4, we use simulation studies for the purpose of recovering the true underlying coefficient function $\beta_k(t)$, and to test the predictive accuracy of multivariate and functional methods in four different contexts. Lastly, Section 5 contains some closing thoughts and new avenues of research for this kind of network.

2 Methods

2.1 Overview of Functional Neural Networks

We will begin with a quick introduction of conventional neural networks which are made up of objects known as hidden layers each of which contains some number of neurons. Let n_u

be the number of neurons in the u-th hidden layer. Each neuron in each layer is a non-linear transformation of a linear combination of the outputs from the previous layer. Often, the output value is referred to as the activation value of a particular neuron. More formally, the first hidden layer $\boldsymbol{v}^{(1)}$ would be defined as $\boldsymbol{v}^{(1)} = g\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$, where $\boldsymbol{x}$ is a vector of J covariates, $\boldsymbol{W}^{(1)}$ is an n_1 x J weight matrix, $\boldsymbol{b}^{(1)}$ is the intercept (bold indicates a vector or a matrix). The intercept $\boldsymbol{b}^{(1)}$ is often referred to as the bias in machine learning texts. The function $g(\cdot)$ is called the activation function that non-linearly transforms the resulting linear combination (Tibshirani et al., 2009). The choice of the function $g : \mathbb{R}^{n_1} \to \mathbb{R}^{n_1}$ is highly context dependent. The rectifier (Hahnloser et al., 2000) and sigmoidal functions (Han and Moraga, 1995) are popular choices for g. Note that the vector $\boldsymbol{x}$ corresponds to a single observation of our data set. The resulting vector $\boldsymbol{v}^{(1)}$ is n_1-dimensional. This vector contains the activation values to be passed on to the next layer.

Thus far, the assumption has been that $\boldsymbol{x}$ is J-dimensional. However, we wish now to consider the case when our input is infinite dimensional defined over some finite domain $\mathcal{T}$. In this case, our input is a functional covariate $x(t) : \mathcal{T} \to \mathbb{R}$, $t \in \mathcal{T}$. By finite domain we mean that $a < t < b$ for $a, b \in \mathbb{R}$. We must weigh this functional covariate at every point along its domain. Therefore, our weight must be infinite dimensional as well. We define this weight as $\beta(t)$. The form of a neuron with a single functional covariate in the first layer then becomes

$$v_i^{(1)} = g\left(\int_{\mathcal{T}} \beta_i(t)x(t)\,\mathrm{d}t + b_i^{(1)}\right), \tag{2}$$

where the subscript $i \in \{1, 2, \ldots, n_1\}$ is an index that denotes one of the n_1 neurons in this first hidden layer. We omit the superscript on the functional weight $\beta(t)$, because this parameter only exists in the first layer of the network.

The functional weight $\beta_i(t)$ is expressed as a linear combination of basis functions,

$$\beta_i(t) = \sum_{m=1}^{M_k} c_{im}\phi_{im}(t) = \boldsymbol{c}_i^T \boldsymbol{\phi}_i(t), \tag{3}$$

where $\boldsymbol{\phi}_i(t) = (\phi_{i1}(t), \ldots, \phi_{iM_k}(t))^T$ is a vector of basis functions, and $\boldsymbol{c}_i = (c_{i1}, \ldots, c_{iM_k})^T$ is the corresponding vector of basis coefficients. The basis coefficients for $\beta_i(t)$ will be initialized by the network; these initializations will then be updated as the network learns.

Common choices of basis functions are the B-splines and the Fourier basis functions (Ramsay and Silverman, 2005). We also note that the evaluation of the neuron in Equation (2) results in a scalar value. This implies that the rest of the $u-1$ layers of the network can be of any of the usual forms (feed-forward, residual, etc.). Using these basis approximations of $\beta_i(t)$, we can simplify to get that the form of a single neuron is:

$$\begin{aligned} v_i^{(1)} &= g\left(\int_{\mathcal{T}} \beta_i(t)x(t)\,\mathrm{d}t + b_i^{(1)}\right) \\ &= g\left(\int_{\mathcal{T}} \sum_{m=1}^{M_k} c_{imk}\phi_{imk}(t)x(t)\,\mathrm{d}t + b_i^{(1)}\right) \\ &= g\left(\sum_{m=1}^{M_k} c_{imk}\int_{\mathcal{T}} \phi_{imk}(t)x(t)\,\mathrm{d}t + b_i^{(1)}\right), \end{aligned} \tag{4}$$

where the integral in Equation (4) can be approximated with numerical integration methods for each of the K functional covariates. We will denote the number of basis functions for each of the K functional covariates as M_k which is left as a hyperparameter of our model.

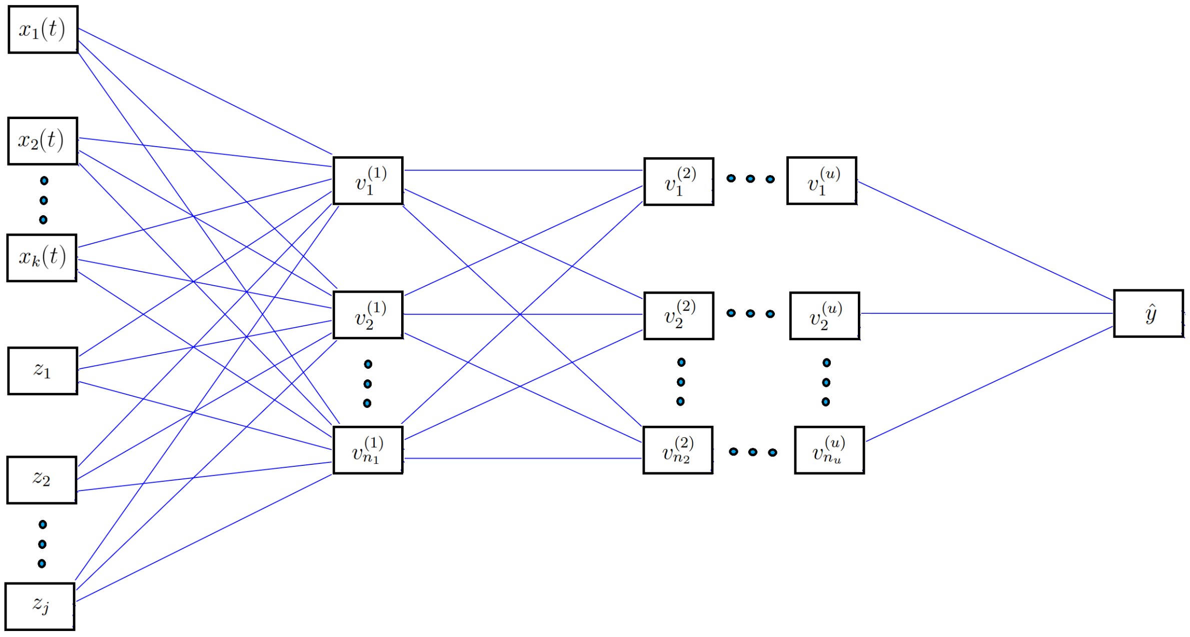

Figure 1: Schematic of a general functional neural network for when the inputs are functions, $x_k(t)$, and scalar values, z_j. The response/output of this network is a scalar value, $\hat{y}$.

We can now consider the generalization for K functional covariates and J scalar covariates. Consider the input layer as presented in Figure 1. The covariates correspond to the ℓ-th observation can be seen as the set:

$$\text{input}_\ell = \{x_1(t), x_2(t), ..., x_K(t), z_1, z_2, ..., z_J\}.$$

Then, the i-th neuron of the first hidden layer corresponding to the ℓ-th observation can be formulated as (we suppress the index, ℓ because this expression does not change with the observation number):

$$v_i^{(1)} = g\left(\sum_{k=1}^{K} \int_{\mathcal{T}} \beta_{ik}(t) x_k(t)\, \mathrm{d}t + \sum_{j=1}^{J} w_{ij}^{(1)} z_j + b_i^{(1)}\right),$$

where

$$\beta_{ik}(t) = \sum_{m=1}^{M_k} c_{ikm}\phi_{ikm}(t), \tag{5}$$

$\phi_{ikm}(t)$ is the basis function and c_{ikm} is the corresponding basis coefficient. This neuron formulation is the core of this methodology. Note that c_{ikm} here is unique at the initialization for each functional weight, $\beta_{ik}(t)$ – the choice of these initializations is discussed later in the article.

To summarize, the general form of the first layer is,

$$\begin{aligned} v_i^{(1)} &= g\left(\sum_{k=1}^{K} \int_{\mathcal{T}} \sum_{m=1}^{M_k} c_{ikm}\phi_{ikm}(t) x_k(t)\, \mathrm{d}t + \sum_{j=1}^{J} w_{ij}^{(1)} z_j + b_i^{(1)}\right) \\ &= g\left(\sum_{k=1}^{K} \sum_{m=1}^{M_k} c_{ikm} \int_{\mathcal{T}} \phi_{ikm}(t) x_k(t)\, \mathrm{d}t + \sum_{j=1}^{J} w_{ij}^{(1)} z_j + b_i^{(1)}\right) \end{aligned} \tag{6}$$

Below we present a corollary to Theorem 1 in Cybenko (1989), that says that the distance of linear combinations of Equation (6) to any continuous function can become arbitrarily small (see Corollary (1)). This feature is important because this model can be used to learn arbitrary uni-variate functions.

Corollary 1. *Let $g : \mathbb{R} \to \mathbb{R}$ be any continuous sigmoidal function, I_n denote the $n-$dimensional hypercube $[0,1]^n$ and $\mathcal{C}(I_n)$ denote the space of continuous functions. Then, the finite sum of the following form, is dense in $\mathcal{C}(I_n)$:*

$$h(t) = \sum_{i=1}^{n_1} \Psi_i g\left(\sum_{k=1}^{K}\left(\int_{\mathcal{T}} \beta_{ik}(t) x_k(t)\, \mathrm{d}t\right) + \sum_{j=1}^{J} w_{ij} z_j + b_i\right),$$

meaning that for any $f(t) \in \mathcal{C}(I_n)$ and for $\epsilon > 0$, the function $h(t)$ obeys:

$$|h(t) - f(t)| < \epsilon.$$

A proof is provided in the supplementary document. After running through this set of initial neurons in the first layer and calculating the activations for the layers following, we can arrive at a final value. The output will be single dimensional. In order to assess performance, we can use a loss function, R. One such function is the mean squared error defined as $R(\theta) = \sum_{\ell=1}^{N} (y_\ell - \hat{y}_\ell(\theta))^2$; in this case, θ is the set of parameters defining the neural network, y_ℓ, $\ell = 1, \ldots, N$, is the observed data for the scalar response, and $\hat{y}_\ell$ is the output from the functional neural network.

2.2 Functional Neural Network Training

Having defined the general formation of functional neural networks, we can now turn our attention to the optimization of this kind of network. We will consider the usual back-propogation algorithm (Rumelhart et al., 1985). While in the implementation, we used the *Adam* optimizer (Kingma and Ba, 2014), we can explain the general process when the optimization scheme uses stochastic gradient descent.

Given our generalization and reworking of the parameters in the network, we can note that the set θ' making up the gradient associated with the parameters is:

$$\theta' = \left\{ \bigcup_{k=1}^{K} \bigcup_{m=1}^{M_k} \bigcup_{i=1}^{n_1} \frac{\partial R}{\partial c_{ikm}}, \bigcup_{u=1}^{U} \bigcup_{j=1}^{J_u} \bigcup_{i=1}^{n_u} \frac{\partial R}{\partial w_{iju}}, \bigcup_{u=1}^{U} \bigcup_{i=1}^{n_u} \frac{\partial R}{\partial b_{iu}} \right\} \tag{7}$$

This set exists for every observation, ℓ. We are trying to optimize for the entirety of the training set, so we will move slowly in the direction of the gradient. The rate at which we move, which is called the learning rate, will be denoted by γ. For the sake of efficiency, we will take a subset of the training observations, which is called a mini-batch, for which we calculate θ'. Then, letting $\bar{a} = \sum_{\ell=1}^{N_b} a'_\ell / N_b$, where $a'_\ell = \partial R / \partial a_\ell$ is the derivative of any parameter $a \in \theta$ for the ℓ-th observation and N_b is the size of the mini-batch. The update for a is $a = a - \gamma \bar{a}$ (Ruder, 2016). This process is repeated until all partitions (mini-batches) of the data set are completed thus completing one training iteration (referred to as an epoch in machine learning texts). The number of epochs is left as a hyperparameter. We summarize the entire network process in Algorithm 1.

Lastly, we would like to emphasize that the number of parameters in the functional neural network presented here can often be lower than the amount required in neural networks that use raw data. Suppose we only have one functional covariate as the input in the neural network, and we have repeat measurements of the functional covariate at P points along its continuum. Passing this information into any conventional multi-layer neural network will mean that the number of parameters in the first layer will be $(P+1)\cdot n_1$, where n_1 is the number of neurons in the first hidden layer. In our network, the number of parameters in the first layer is the number of basis functions we use to define the functional weight. The number of basis functions M_1 will be less than P because there is no need to have a functional weight that interpolates across all our observed points – we prefer a smooth effect across the continuum to avoid fitting to noise. Therefore, a good practice indicates that the number of parameters in the first layer of our network is $(M_1 + 1) \cdot n_1$ where $M_1 < P$. This example trivially generalizes for K functional covariates.

2.3 Functional Weights

Since a leading contributor to the black-box reputation of conventional neural networks is the inordinate amount of changing weights and intercepts, it would be helpful to consider rather a function defined by these difficult-to-interpret numbers. In particular, we consider hidden semantic interpretability which concerns the human ability to understand hidden layers (Fan et al., 2021). The literature on the interpretation of hidden layers/neurons for conventional neural networks is quite limited; most work on this kind of interpretation is associated with computer vision methods (Zhang et al., 2018; Simonyan et al., 2013). In our FNN framework, the functional weights (Equation (5)), which are estimated in the first layer, help us interpret the relationship between the functional covariate and the scalar response. These functional weights are akin to the ones predicted in the functional regression model (Ramsay and Silverman, 2005). The final set of functional weights $\beta_{ik}(t)$ can be compared with the one estimated from a function linear model. In the case of multiple neu-

Algorithm 1: Functional Neural Networks

```
Input: $x_k(t), z_j$ for $k = 1, 2, ..., K$ and $j = 1, 2, ..., J$, $\gamma$, Number of Layers,
       Neurons per Layer, Activation Functions, Decay Rate, Validation Split,
       Functional Weight Basis Expansion Sizes, Functional Weight Basis
       Functions, Functional Weight Domains, Epochs, Batch Size, Early Stop,
       Threshold for Minimum Improvement, Loss Choice, Dropout (Optional)
       (Definitions available in Table S1 in the supplementary document)
Output: $\theta$ (as defined on page 9)

1. Set Hyperparameters:
$\gamma$, Number of Layers, Neurons per Layer, Activation Functions, Decay Rate,
  Validation Split, Functional Weight Basis Expansion Sizes, Functional Weight
  Basis Functions, Functional Weight Domains, Epochs, Batch Size, Early Stop,
  Threshold for Minimum Improvement, Loss Choice, Dropout (Optional)
2. Let q be the reduction in loss from two subsequent training iterations. Initialize
  as 0
3. Let $r > 0$ be the threshold for minimum improvement (our proxy: patience
  parameter)
4. Let $s = 0$
5. while $q < r$
   5i. Forward Pass
       a. $x_k(t), z_j$ passed to first hidden layer
       b. Approximate $\int_{\mathcal{T}} \phi_{km}(t)x(t)\,\mathrm{d}t = \tilde{\phi}_{km}$ for each basis function, $m$ and for
          each functional covariate, $k$ (as defined on page 6)
       c. Calculate $E = g\left(\sum_{k=1}^{K}\sum_{m=1}^{M_k} c_{km}\tilde{\phi}_{km} + \sum_{j=1}^{J} w_j^{(1)} z_j + b^{(1)}\right)$ for each
          neuron (as defined on page 8)
       d. Pass E to the attached network architecture
       e. Calculate loss: $R(\theta)$ (as defined on page 9)
   5ii. Backward Pass
       a. Compute $\theta'$ (as defined on page 9)
       b. $\forall\ a \in \theta_p$, update $a$ as: $a = a - \gamma\bar{a}$ (as defined on page 9)
   5iii. Stopping Criteria Updates
       a. If $s = 0$: $q = R(\theta)$ Else: $q = q - R(\theta)$
       b. $s = s + 1$
6. Return $\theta = \theta_p$
```

rons, we can consider the curves individually or define a summary statistic such as taking the average of the estimated functional weights $\hat{\beta}_k(t) = \sum_{i=1}^{n_1} \hat{\beta}_{ik}(t)/n_1$. While training the network, it is possible to track and visualize changes of the weights from epoch to epoch

– as shown in Figure 2. In this example, the defined stopping criteria would come into effect at the 99th epoch. We can see that the difference between contiguous curves is most pronounced in the beginning and is least pronounced after the model finds some local extrema. In this example, the functional weights were initialized from a uniform distribution but a more drastic change in the shape could be seen with a different initialization and a different choice of basis functions for the functional weight.

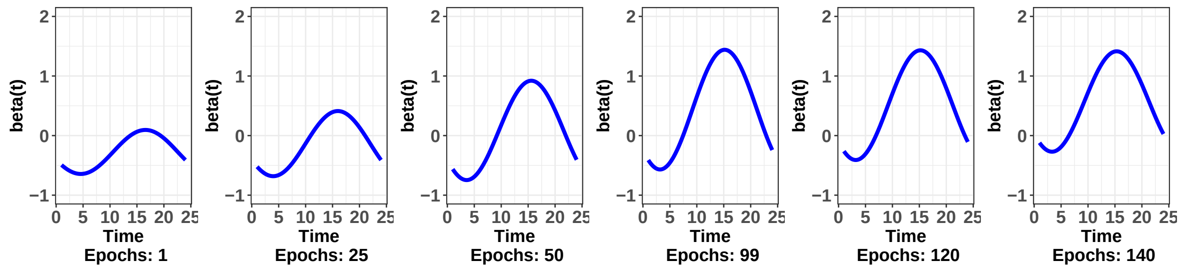

Figure 2: An example of how the functional weight, $\beta_{ik}(t)$, changes over the number of training iterations or epochs as they are referred to in neural network literature.

2.4 Weight Initialization and Parameter Tuning

For any neural network, the weights and intercepts can be initialized in a number of ways. For example, in Kim and Ra (1991) weights are initialized based on a boundary that allowed for quicker convergence. Another approach is to consider a random search; this method looks for a set of weights near the ones that currently minimize the cost function – this process is repeated until some criteria is met (Rastrigin, 1963). In the case of the networks presented here, this is left as a hyperparameter. Since the implementation is built on top of the `Keras` architecture (Chollet et al., 2015), the initialization is dependant on the type of connected layers.

Due to the sheer number of hyperparameters in the network, a tuning approach can be used to find optimal values in our applications. The tuning method is to take a list of possible values for each parameter and run a cross-validation (Tibshirani et al., 2009) for all combinations. The number of folds to use depends on the size of the problem. The general scheme is that the function creates a grid, and calculates the B-fold cross-validated mean squared prediction error MSPE $= \sum_{b=1}^{B} \sum_{l \in S_b} \left(\hat{y}_l^{(-b)} - y_l \right)^2 / N$, where S_b

is the b-th partition of the data set, and $\hat{y}_l^{(-b)}, l \in S_b$, is the predicted value for y_l by training the functional neural network using the rest of the $B-1$ partitions of the data set. The number of data points in S_b depends on the number of folds. The final output of the tuning function is the combination of hyperparameters that have the minimum value of this cross-validated error. A list of hyperparameters is given in Table S1 in the supplementary materials. One important parameter in this particular kind of network is the number of basis functions that govern the functional weights as evidenced via our ablation studies in the supplementary materials (Figure S1). The ablation studies also provide information on how these networks perform when we vary the number of neurons, learn rate, decay rate, epochs, and validation split rate. The study was done by fixing all parameters except 1 which has values chosen from a pre-defined grid. Then, the results for each grid value are measured by a 5-fold MSPE. Lastly, in all the examples to come, we tune all the models to some degree. For example, for the neural network methods, we use the Early Stop parameter (which controls for the number of iterations) to avoid overfitting. Moreover, we initialize multiple weights for these models and select the model that has the lowest error rate. Some additional information on tuning for other models is provided in Section 4.2.

3 Applications

3.1 Bike Sharing Data

An important problem in rental businesses is the amount of supply to keep on-site. If the company cannot meet demands, they are squandering profitability. If they exceed the required supply, they have made investments that are not yielding an acceptable return. Using the bike sharing data set (Fanaee-T and Gama, 2014), we look to model the relationship between the total number of daily rentals (a scalar value) and the hourly temperature throughout the corresponding day (functional observation). It makes intuitive sense for temperature to be related to the number of bike rentals: on average, if it's cold, less people are likely to rent than if it were warm. We also expect there to be a temporal effect on temperature – if we have the same temperature at 1pm and 9pm, we would expect more rentals at 1pm under the assumption that less people are deciding to bike later at night.

The data contains a daily count of bike rentals, along with hourly temperature readings (24 points per day). In total, we used 102 functional observations. To eliminate the effect of day-to-day variation, we only conducted the analysis for Saturdays. We have one functional covariate ($K = 1$) represented using 31 Fourier basis functions. The number of basis functions used to represent the functional weight is $M_k = M_1 = 5$. As per usual, we use the raw data when modelling using non-functional methods i.e., there are 24 covariates as opposed to the single functional observation.

Model	$\texttt{MSPE}_{CV}$	R^2	SE
Functional Linear Model (Basis)	2.83	0.591	0.455
Functional PC Regression	3.18	0.550	0.617
Functional PC Regression (2nd Deriv Penalization)	5.38	0.196	0.701
Functional PC Regression (Ridge Regression)	3.26	0.543	0.531
Functional Partial Least Squares	2.82	0.595	0.448
Functional Partial Least Squares (2nd Deriv Penalization)	2.82	0.598	0.511
Convolutional Neural Networks	2.74	0.626	0.643
Neural Networks	3.08	0.575	0.665
Functional Neural Networks	2.47	0.659	0.469

Table 1: The 10-fold cross-validated mean-squared predication error ($\texttt{MSPE}_{CV}$) and R^2 of eight models, including the functional neural network (FNN). The FNN performs the best with respect to both measures.

To compare results between different models we use $R^2 = 1 - \sum_{l=1}^{N}(y_l - \hat{y}_l)^2 / \sum_{l=1}^{N}(y_l - \bar{y})^2$ and a 10-fold cross-validated mean squared prediction error as defined in Section 2.4. Here, we compare with the usual functional linear model, an FPCA approach (Cardot et al., 1999), a non-parametric functional linear model (Ferraty and Vieu, 2006), and a functional partial least squares model (Preda et al., 2007). We also compare with conventional feed-forward neural networks (multilayer perceptron) as well 1D convolutional neural networks (LeCun et al., 2015). The are 3521, 4001, and 15649 parameters in the FNN, NN, and CNN models used here, respectively. The results are summarized in Table 1. The final model configurations for the functional neural network are provided in Table S2 in the supplemental document. Observe that FNNs outperform all the other models using both criteria but note that the functional partial least squares and convolutional neural network approaches perform comparably. Paired t-tests were conducted and results can be found in Table S3 along with training plots (Figure S2).

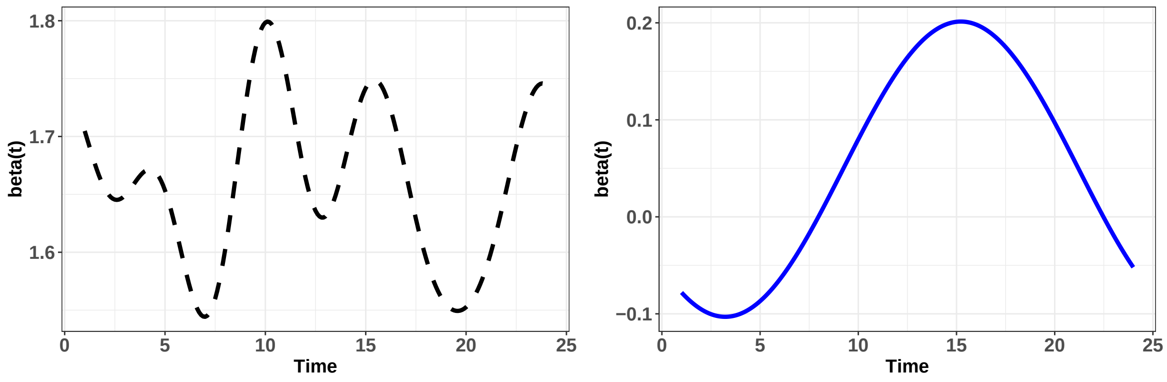

Figure 3: The estimated functional weight $\hat{\beta}(t)$ in the conventional functional linear model (dashed black curve) and the functional neural network (solid blue curve) for the bike rental data set as a function of time.

We can also look to see what the determined relationship is according to the functional linear model and the functional neural network between hourly temperature and daily rentals as indicated by $\beta(t)$. Figure 3 shows an example of the estimated weight function $\hat{\beta}(t)$. The optimal number of basis functions was 11 for the functional linear model and 5 for the functional neural network. For the functional linear model, we note that there seems to be no obvious discernable relationship between hourly temperature and bike rentals. In the case of the functional neural network, we see that there seems to be a positive relationship as we move into the afternoon and that this relationship tapers off as the day ends. We would also expect there to be no effect for when bike rental retailers would be closed, and this is much better reflected in the functional weight from the neural network than the functional coefficient in the functional linear model. We observe different scales for the two and we posit that this difference can be explained by the fact that the functional neural network has a large number of additional parameters that may be explaining some of the variation in the response. Note that this is only an example of what the estimated functional weights could be for a particular fold. The full plot list of estimated weights from each fold can be found in Figure S4 in the supplementary document.

3.2 Tecator Data

We consider the classic Tecator data set (Thodberg, 2015). The data are recorded on a Tecator Infratec Food and Feed Analyzer using near-infrared laser light (wavelength is 850 nm - 1050 nm) to analyze the samples. Each sample contains meat with different moisture, fat, and protein contents. The goal is to predict the scalar value of fat contents of a given meat sample using the functional covariate of the near infrared absorbance spectrum and the scalar covariate associated with the water contents. Absorbance spectroscopy measures the fraction of incident radiation absorbed by the sample. Samples with higher water composition may exhibit different spectral features (absorbance bands) than samples with higher protein content. Since we are working with functional covariates, we also have access to their derivatives. The derivative (rate of change) information can serve as an important predictor and is used as such. Accessing the derivative of a functional covariate is straightforward. We define functional covariates as linear combinations of basis expansions and then differentiate term by term. In this case, we use a Fourier expansion to find $x(t)$ so, to find $x'(t)$ is to find the derivatives of each term. For this example, we also considered the scalar covariate of water which has an impact as well on the meat contents as it relates to spectroscopy.

In total, this data has 215 absorbance curves (functional observations) with a domain of 100 wavelengths. We have one functional covariate ($K = 1$) represented using 29 Fourier basis functions and one scalar covariate ($J = 1$); the number of basis functions used to represent the functional weight is 3, ($M_k = M_1 = 3$). The first 165 absorbance curves are used as the training set, and the predictions are made on the remaining. We used this test/train paradigm instead of cross-validation because we can then directly compare results from Febrero-Bande and de la Fuente (2012). They fitted several models using this paradigm. We present their results along with results from the functional neural network, convolutional neural network, and conventional neural network in Table 2.

Model	Covariates	MEP	R^2
Functional Linear Model	1st Derivative	0.0626	0.928
Functional Linear Model	2nd Derivative	0.0566	0.965
Functional PC Regression	1st Derivative	0.0580	0.950
Functional PC Regression	2nd Derivative	0.0556	0.954
Functional Partial Least Squares	1st Derivative	0.0567	0.951
Functional Partial Least Squares	1st Derivative	0.0487	0.962
Functional Linear Model	1st Derivative, Water	0.0097	0.987
Functional Linear Model	2nd Derivative, Water	0.0119	0.986
Functional Non-Parametric Regression	1st Derivative	0.0220	0.987
Functional Non-Parametric Regression	2nd Derivative	0.0144	0.996
Semi-Functional Partially Linear Model	1st Derivative, Water	0.0090	0.996
Semi-Functional Partially Linear Model	2nd Derivative, Water	0.0115	0.997
Convolutional Neural Networks	Raw Curves, Water	0.0210	0.979
Neural Networks	Raw Curves, Water	0.0219	0.978
Functional Neural Networks	2nd Derivative, Water	0.00883	0.991

Table 2: The mean square error of prediction (MEP) and R^2 of our method and those used in Febrero-Bande and de la Fuente (2012) for analyzing the Tecator data set. The objects X.d1 and X.d2 refer to the first and second derivatives of the near infrared absorbance spectrum curve, respectively.

In the original paper, the authors use the metric MEP $=$ MSPE/Var(y), where MSPE is the average squared errors of the test set and Var(y) is the variance of the observed response (we can think of MEP as a rescaling of the MSPE) to assess the models. They also used R^2, which we tabulate in Table 2. In this example, the functional neural network had 4029 parameters. In contrast, the neural network had 5757 parameters and CNN has 87405 parameters. Our model has the lowest MEP, but has a lower R^2 than the best. Most other models perform worse with the Semi-Functional Partial Linear Model (Aneiros-Pérez and Vieu, 2006) being the most comparable. Both the conventional and convolutional neural networks performed worse. The training plots for this example can be found in the supplementary document (Figure S5). Note that we only presented the results using the second derivative of the spectrum curves as the functional covariate because it was the better performer when compared to the network made using the first derivative or the raw functional observations themselves. We did not use multiple functional covariates here because all the other models only used one functional covariate. The other neural networks used the raw data in their model building process.

3.3 Canadian Weather Data

The data set used here has information regarding the total amount of precipitation in a year and the daily temperature for 35 Canadian cities. We are interested in modelling the relationship total between annual precipitation and daily temperature throughout the year. Generally, you would expect that lower temperatures would indicate higher precipitation rates. However, this is not always the case. In some regions, the temperature might be very low, but the inverse relationship with rain/snow does not hold. Our goal is to see whether we can successfully model these anomalies relative to other methods.

Model	$\texttt{MSPE}_{CV}$	SE
Functional Linear Model (Basis)	0.597	0.488
Functional Non-Parametric Regression	0.0667	0.0369
Functional PC Regression	0.0300	0.0199
Functional PC Regression (2nd Deriv Penalization)	0.0493	0.0364
Functional PC Regression (Ridge Regression)	0.0290	0.0199
Functional Partial Least Squares	0.0480	0.0256
Functional Partial Least Squares (2nd Deriv Penalization)	0.0515	0.0258
Convolutional Neural Networks	0.0427	0.0179
Neural Networks	0.0372	0.0150
Functional Neural Networks	0.0244	0.00919

Table 3: The leave-one-out cross-validated mean-squared predication error (MSPE_{CV}) and R^2 of nine models for the weather data set.

The functional observations are defined for the temperature of the cities for which there are 365 (daily) time points. In total, this data has 35 temperature curves (functional observations) with a domain-size of 365 days and the scalar response is the scaled total precipitation across the year. The functional covariate of temperature ($K = 1$) is represented using 65 Fourier basis functions (Ramsay et al., 2009); there are no scalar covariate ($J = 0$), and the number of basis functions to represent the functional weight is 3, ($M_k = M_1 = 3$). The results from two criteria (R^2 and a 10-fold cross-validated MSPE) are measured for a number of models (same as in Section 3.1). The total number of parameters in the FNNs, CNNs, and NNs are 241, 95009, and 6001, respectively. For additional information, see Table S5 in the supplementary document. We see that the FNN model outperforms all other approaches including the conventional and convolutional neural networks. Table 3 summarizes the predictive results.

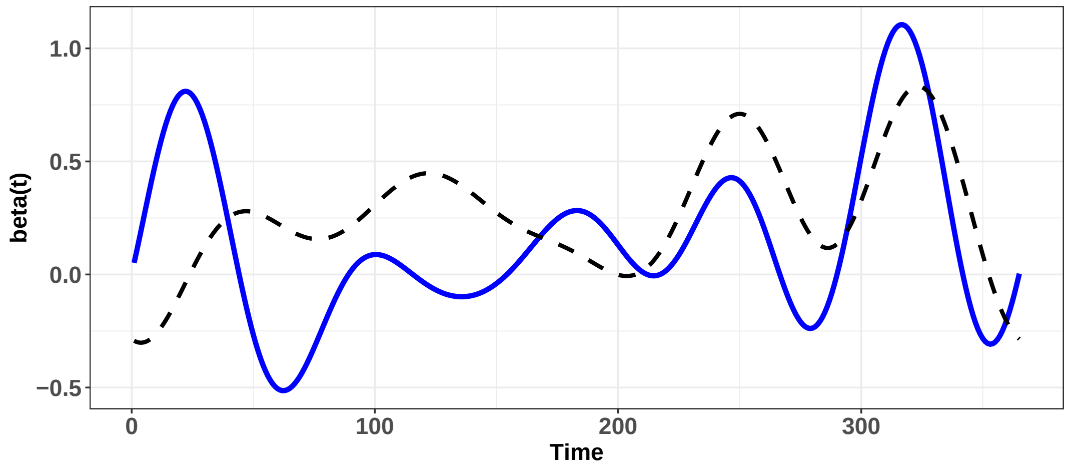

Figure 4: The estimated functional weight for the conventional functional linear model (dashed black curve) and the functional neural network (solid blue curve) for the weather data set.

We can compare the estimated functional weight from the functional linear model with the functional neural network in Figure (4) in a similar fashion to Section 3.1. For the purpose of this example, we decided to keep the number of basis functions the same across both models. In this case we selected the number of basis functions to be 11, motivated by Ramsay et al. (2009). This was to measure how similar the functional coefficients would be under the same conditions. We observe similar patterns between the two models especially over the second half of the domain (Time > 200). Note that the difference between the two recovered functional weights only accounts for some of the difference in R^2. The functional neural network has more parameters allowing for more flexibility in the modelling process and thus the increase in accuracy.

4 Simulation Studies

4.1 Recovery of Functional Weight

In the previous section, we presented application of FNNs on real data and provided comparisons with other functional models, NNs and CNNs. This was a study of the performance among these various models where the performance metrics were associated with $\hat{y}$, the predictions. In this section, we present initial studies on how the FNN performs in inference problems. In other words, we generate data where the functional weight $\beta(t)$ is known, and perform various experiments to learn $\beta(t)$, comparing Functional Neural Networks and Functional Linear Models. In order to measure this, the integrated mean square error (IMSE) is used, which is defined as

$$\text{IMSE} = \frac{1}{|\mathcal{T}|}\int_{\mathcal{T}} (\beta(t) - \hat{\beta}(t))^2 \, \mathrm{d}t,$$

where $\hat{\beta}(t)$ is the estimated functional weight either from the FNN or from the functional regression. The ℓ-th response is generated as

$$y_\ell^* = g\left(\alpha + \int_{\mathcal{T}} \beta(t) x_\ell(t) \, \mathrm{d}t\right) + \epsilon_\ell^*, \;\; \ell = 1, \ldots, L, \tag{8}$$

where our choice for $\beta(t)$ is

$$\beta(t) = m_1 + m_2 \sin(\pi t) + m_3 \cos(\pi t) + m_4 \sin(2\pi t) + m_5 \cos(2\pi t),$$

and the noise ϵ_ℓ^* is sampled from the Gaussian distribution $\mathcal{N}(0, 1)$. The functional observations $x_\ell(t)$, $t \in \mathcal{T} = (0, 1)$ are generated, depending on the simulation scenario, either from $c \cdot \sin(a) + b$ or $c \cdot \exp(a) + \sin(a) + b$, where $a \sim \mathcal{N}(0, 1)$, $b \sim \mathcal{N}(0, \frac{\ell}{100})$, and $c \sim \mathcal{N}(0, t/100)$ are parameters that govern the difference between the functional observations.

This generative procedure will be used for four different simulations. In all four, we generate 300 observations randomly using Equation (8) by varying a, b and c. The coefficients for $\beta(t)$, $m_1, ..., m_5$, are set beforehand. We fit the functional linear model (Equation (1)) and the FNN for each simulation data set. The functional linear model (Equation (1)) is estimated by minimizing the penalized sum squared errors defined as

$$\sum_{\ell=1}^{L} \left(y_\ell^* - \alpha - \int_{\mathcal{T}} \beta(t) x_\ell(t) \, \mathrm{d}t\right) + \lambda \int_{\mathcal{T}} \left(\frac{d^2\beta(t)}{dt^2}\right)^2 \mathrm{d}t \, ,$$

where $\beta(t)$ is expressed as a linear combination of basis function $\beta(t) = \sum_{m=1}^{M} c_m \phi_m(t)$, $\phi_m(t)$ is the basis function and c_m is the corresponding basis coefficient. The smoothing parameter λ controls the smoothness of $\beta(t)$. The smoothing parameter λ is usually tuned for; we cross-validate (Ramsay and Silverman, 2010) over a grid with range [0.0625, 1024] for λ in order to find the optimal estimate of $\beta(t)$ from the functional linear model. The difference is measured using IMSE. The simulation is replicated 250 times for each of the follower scenarios. These simulations are summarized as follows:

$$\text{Simulation 1} : y^* = \alpha + \int_{\mathcal{T}} \beta(t)x(t)\,\mathrm{d}t + \epsilon^* \tag{9}$$

$$\text{Simulation 2} : y^* = \exp\left(\alpha + \int_{\mathcal{T}} \beta(t)x(t)\,\mathrm{d}t\right) + \epsilon^* \tag{10}$$

$$\text{Simulation 3} : y^* = \frac{1}{1 + \exp\left(\alpha + \int_{\mathcal{T}} \beta(t)x(t)\,\mathrm{d}t\right)} + \epsilon^* \tag{11}$$

$$\text{Simulation 4} : y^* = \log\left(\left|\alpha + \int_{\mathcal{T}} \beta(t)x(t)\,\mathrm{d}t\right|\right) + \epsilon^*. \tag{12}$$

The first simulation is for when the link function $g(\cdot)$ is the identity function (see Equation (9)). Here, we would expect the functional linear model to perform comparably (if not better) to the FNN due to its deterministic nature and the linear relationship. In the second simulation, we look to see if our method can recover $\beta(t)$ better for when the link function is exponential (see Equation (10)). The third simulation explores this behaviour for a sigmoidal relationship (see Equation (11)). And lastly, we simulate a logarithmic relationship between the response and the functional covariates (see Equation (12)). All scenarios except simulation 2 use $c \cdot \exp(a) + \sin(a) + b$ for the data generation – this was purely to control the range of y in the exponential example. In all but the fourth scenario, we use a three-layer network with rectifier (Hahnloser et al., 2000) and linear activation functions. In the fourth scenario, we use a one-layer network with a sigmoidal activation function.

In Figure 5, we present the results for these four simulations. We observe that the usual linear model seems to perform better when the relationship is linear. There are far more parameters in the FNN that are contributing to the prediction of y than just the ones in the functional layer. When the relationship is non-linear, the functional linear model struggles where relatively, the FNN does a better job in recovering $\beta(t)$.

The averages of these results along with computation times are provided in Table 4. We observe that the functional neural network, as expected, takes a longer time to run across all simulation scenarios. However, we also observe, as the box plots indicate, that for non-linear simulation scenarios, the functional neural network outperforms the functional linear model. This difference seems to be the most pronounced in simulation scenario 4. We also note that because of the stochastic nature i.e., random weight initialization of functional neural networks, there is a higher variance in our estimates when compared with the deterministic functional linear model.

	Functional Linear Model			Functional Neural Networks			Paired T-Test
	Mean	SE	Avg. Comp. Time	Mean	SE	Avg. Comp. Time	P-Value
Simulation: 1	2.27	.0.00233	0.232s	2.39	0.00301	4.68s	< 0.0001
Simulation: 2	2.53	0.000569	0.232s	2.36	0.00286	4.67s	< 0.0001
Simulation: 3	6.70	.0.00115	0.247s	6.43	0.00684	6.00s	< 0.0001
Simulation: 4	7.43	0.00293	0.258s	6.46	0.00475	6.30s	< 0.0001

Table 4: The square root of the mean along with the associated standard deviation (SE $= \text{SD}/\sqrt{250}$) of the integrated mean square errors (IMSEs) over 250 simulation replicates are provided for both the functional neural networks and the functional linear model. The computation times given are the average per simulation replication.

4.2 Prediction

In this section, we investigate how the functional neural networks compare with other functional and multivariate models under the four different simulation scenarios detailed previously when the task is prediction. We use a variety of multivariate linear methods to compare with the FNN: least squares regression (MLR), LASSO (Tibshirani, 1996), random forests (RF) (Breiman, 2001), gradient boosting approaches (GBM, XGB) (Friedman, 2001; Chen et al., 2015), and projection pursuit regression (PPR) (Friedman and Stuetzle, 1981). We also consider CNNs and NNs as before.

We did not tune our functional neural networks because we found that they performed well in our initial tests irregardless of the tuning. In contrast, we made an effort to tune all the other models. For example, the choice of λ for the LASSO was made using cross-validation. The tree methods including RF, GBM, and XGB were tuned across a number of their hyperparameters such as the node size, and for PPR, we built models with different

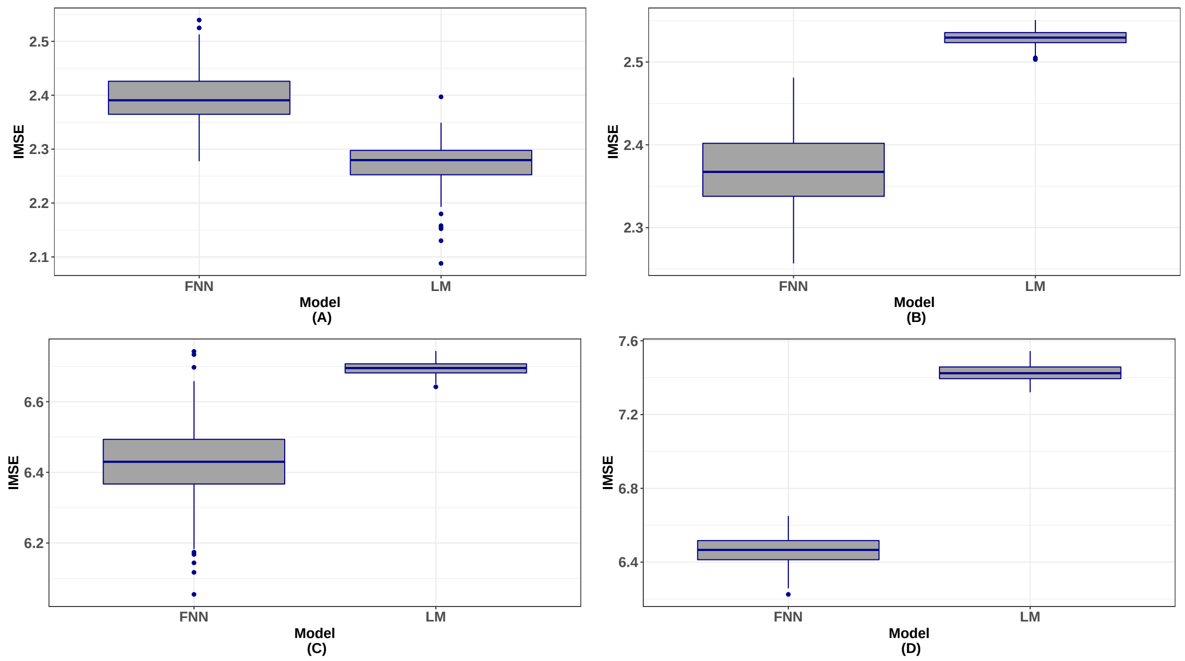

Figure 5: Boxplots of square root of the integrated mean square errors (IMSEs) over 250 simulation replicates for four scenarios. Plot (A) is for when we use the identity link function. Plot (B) are results from the exponential link. The bottom plots, (C) and (D) are the results from simulation 3 and 4, respectively.

numbers of terms and picked the model with the lowest MSPE. Lastly, with respect to the network methods, we initialize weights 10 times and pick the set that resulted in the lower error rate. In these simulations, we did 100 replicates. For each simulation replicate, we generated 300 functional observations in accordance to Equation (8). Next, we split the data randomly, built a model on the training set, and predicted on the test set. This process is repeated for the same four simulation scenarios as given in Section 4.1.

The box plots in the supplemental document (Figure S9) measure the relative error in each simulation replicate. We call this the relative MSPE defined as:

$$\text{rMSPE} = \frac{\text{MSPE}}{\min\limits_{\text{all models}} \text{MSPE}}.$$

For example, on any given simulation replicate, we calculate the MSPE values for each model, and then divide each of them by the minimum in that run. The best model according to this measure will have a value of 1. Error values greater than 1 implies a worse performance. In the supplementary document, Table S10 contains the exact error rates and Table S11 has the results for the paired T-Tests.

The relative measure we use makes it easy to compare each model within a simulation, and across the four simulation scenarios. Notably, we see that the functional neural network performs well (see Figure S9 in the supplementary document). This can be attributed to the addition of the functional information passed into the network; the functional neural network can take into account the temporal trend of the data to learn more about the underlying relationship in each training iteration. Therefore, we gain a model that better estimates the underlying true relationship between the covariates and the response in comparison with multivariate approaches.

As a comparison, we see that generally the tree-based methods perform comparably in scenario 4 but worse in all the others. Also, depending on the scenario, it seems that multivariate methods are capable of outperforming most functional methods with respect to rMSPE with the exception of the functional neural networks; they seem to be consistently good performers across these scenarios – ousting the other neural network methods as well. With respect to outliers, we can see that they are most prevalent in Simulation 2; this is because this simulation was for when the link function $g(\cdot)$ was exponential. In this context, we expect that the difference between our prediction and the corresponding observed value is greater than it would be in the other simulation scenarios. The full set of results can be found in Section 4.5 in the supplementary document.

5 Conclusions and Discussion

In this paper, we establish a novel connection between the fields of deep learning and functional data. Our framework only modifies the first layer of a neural network, making this approach network-architecture independent. As an initial implementation of the framework, we present a functional feed-forward neural network to predict a scalar response with functional and scalar covariates. We developed a methodology which showed the steps required to compute the functional weight for the neural network. Multiple examples were provided which showed that the functional neural network outperformed a number of other functional models and multivariate methods with respect to the mean squared prediction error. It was also shown through simulation studies that the recovery of the true functional weight is better done by the functional neural network than the functional linear model when the true relationship is non-linear. To extend this project, algorithms can be developed for other combinations of input and output types such as the function on function regression models (Morris, 2015). Moreover, one can consider adding additional constraints to the first-layer neurons via penalization or other methods.

SUPPLEMENTAL MATERIALS

Data and R Code: Data and R Code used for each application and simulation study presented in this paper; this includes some of the source for the **FuncNN** R package. A README file is included which describes the files. (JCGS_20_212_Code.zip)

Supplemental Document: Document containing the proof for Corollary 1, tables with descriptions of various parameters, model configurations, and additional simulation results. (supplementalFNN.pdf File)

References

Aneiros-Pérez, G., and Vieu, P. (2006), "Semi-functional partial linear regression," *Statistics & Probability Letters*, 76, 1102–1110.

Breiman, L. (2001), "Random forests," *Machine learning*, 45, 5–32.

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

Chen, T., He, T., Benesty, M., Khotilovich, V., and Tang, Y. (2015), "Xgboost: extreme gradient boosting," *R package version 0.4-2*, 1–4.

Chollet, F. et al. (2015), "Keras," `https://keras.io`.

Cybenko, G. (1989), "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, 2, 303–314.

Fan, F.-L., Xiong, J., Li, M., and Wang, G. (2021), "On interpretability of artificial neural networks: A survey," *IEEE Transactions on Radiation and Plasma Medical Sciences.*

Fanaee-T, H., and Gama, J. (2014), "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, 2, 113–127.

Febrero-Bande, M., and de la Fuente, M. (2012), "Statistical Computing in Functional Data Analysis: The R Package fda.usc," *Journal of Statistical Software, Articles*, 51, 1–28.

Ferraty, F., and Vieu, P. (2006), *Nonparametric functional data analysis: theory and practice*, New York: Springer-Verlag.

Friedman, J. (2001), "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, 29, 1189–1232.

Friedman, J., and Stuetzle, W. (1981), "Projection pursuit regression," *Journal of the American Statistical Association*, 76, 817–823.

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000), "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, 405, 947–951.

Han, J., and Moraga, C. (1995), "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International Workshop on Artificial Neural Networks*, Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2016), "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Jiang, C.-R., and Wang, J.-L. (2011), "Functional single index models for longitudinal data," *The Annals of Statistics*, 39, 362–388.

Kim, Y., and Ra, J. (1991), "Weight value initialization for improving training speed in the backpropagation network," in *IEEE International Joint Conference on Neural Networks*, IEEE.

Kingma, D., and Ba, J. (2014), "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012), "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015), "Deep learning," *nature*, 521, 436–444.

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

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

Preda, C., Saporta, G., and Lévéder, C. (2007), "PLS classification of functional data," *Computational Statistics*, 22, 223–235.

Ramsay, J., and Silverman, B. W. (2010), *Functional data analysis*, "New York": Springer.

Ramsay, J. O., Hooker, G., and Graves, S. (2009), *Functional data analysis with R and MATLAB*, New York: Springer.

Ramsay, J. O., and Silverman, B. W. (2005), *Functional Data Analysis*, New York: Springer.

Rastrigin, L. (1963), "The convergence of the random search method in the extremal control of a many parameter system," *Automaton & Remote Control*, 24, 1337–1342.

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

Ruder, S. (2016), "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*.

Rumelhart, D., Hinton, G., and Williams, R. (1985), "Learning internal representations by error propagation," Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science.

Seber, G. A., and Lee, A. J. (2012), *Linear regression analysis*, Hoboken: John Wiley & Sons.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013), "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014), "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, 15, 1929–1958.

Thind, B., Wu, S., Groenewald, R., and Cao, J. (2020), "FuncNN: An R Package to Fit Deep Neural Networks Using Generalized Input Spaces," *arXiv preprint arXiv:2009.09111*.

Thodberg, H. H. (2015), "Tecator meat sample dataset," *StatLib Datasets Archive*.

Tian, T. S. (2010), "Functional Data Analysis in Brain Imaging Studies," *Frontiers in Psychology*, 1, 35.

Tibshirani, R. (1996), "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, 58, 267–288.

Tibshirani, R., Hastie, T., and Friedman, J. (2009), *The elements of statistical learning: data Mining, inference, and prediction*, New York: Springer.

Yao, Y., Rosasco, L., and Caponnetto, A. (2007), "On early stopping in gradient descent learning," *Constructive Approximation*, 26, 289–315.

Zhang, Q., Wu, Y. N., and Zhu, S.-C. (2018), "Interpretable convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8827–8836.