




Deep Learning With Functional Inputs

Barinder Thind, Kevin Multani & Jiguo Cao

To cite this article: Barinder Thind, Kevin Multani & Jiguo Cao (2022): Deep Learning With Functional Inputs, Journal of Computational and Graphical Statistics, DOI: [10.1080/10618600.2022.2097914](https://doi.org/10.1080/10618600.2022.2097914)

To link to this article: <https://doi.org/10.1080/10618600.2022.2097914>




View supplementary material 



Published online: 06 Sep 2022.



Submit your article to this journal 



Article views: 310



View related articles 



View Crossmark data 



Deep Learning With Functional Inputs

Barinder Thind^a, Kevin Multani^b, and Jiguo Cao^a 

^aDepartment of Statistics & Actuarial Science, Simon Fraser University, British Columbia, Canada; ^bDepartment of Physics, Stanford University, Stanford, CA

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. A supplemental document, the data and R codes are available online.

ARTICLE HISTORY

Received June 2020

Accepted June 2022

KEYWORDS

Functional data analysis;
Machine learning; Neural
networks; 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; Cao and Ramsay 2010). Functional data is found commonly in many applications such as time-course gene expressions and brain scan images (Ainsworth, Routledge, and Cao 2011; Luo et al. 2013; Nie, Wang, and Cao 2017; Dong et al. 2018; Guan et al. 2022). 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

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

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, Ferraty, and Sarda 1999; Lin et al. 2017; Liu, Wang, and Cao 2017; Guan, Lin, and Cao 2020). This model was extended to the general functional linear model:

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

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, Saporta, and Lévêder (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 nonparametric model: $E(Y|X) = r(X(t))$, where $r(\cdot)$ is a smooth nonparametric function that is estimated using a kernel approach. Another model, which serves as an extension to the previous, is the semifunctional partial linear model (Aneiros-Pérez and Vieu 2006) defined as: $E(Y|X) = r(X(t)) + \sum_{j=1}^J w_j Z_j$, where Z_j , $j = 1, \dots, J$, 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. Moreover, we show that the method presented here outperforms convolutional and recurrent neural networks (and some variations) (Hochreiter and Schmidhuber 1997; Graves and Schmidhuber 2005; Chung et al. 2014) for prediction problems. 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, Sutskever, and Hinton (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, Hastie, and Friedman 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 $\beta_k(t)$, $k = 1, \dots, K$, rather than a vector of scalar values. This article 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 article 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, Rosasco, and Caponnetto 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 datasets (Thind et al. 2020).

The rest of our article 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 (such as methods in functional data as well as recurrent (Hochreiter and Schmidhuber 1997) and convolutional neural networks) for multiple datasets. 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 nonlinear 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 $\mathbf{v}^{(1)}$ would be defined as $\mathbf{v}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$, where \mathbf{x} is a vector of J covariates, $\mathbf{W}^{(1)}$ is an $n_1 \times J$ weight matrix, $\mathbf{b}^{(1)}$ is the intercept (bold indicates a vector or a matrix). The intercept $\mathbf{b}^{(1)}$ is often referred to as the bias in machine learning texts. The function $g(\cdot)$ is called the activation function that nonlinearly transforms the resulting linear combination (Tibshirani, Hastie, and Friedman 2009). The choice of the function $g : \mathbb{R}^m \rightarrow \mathbb{R}^m$ 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 \mathbf{x} corresponds to a single observation of our dataset. The resulting vector $\mathbf{v}^{(1)}$ 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 \mathbf{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 $x(t) : \mathcal{T} \rightarrow \mathbb{R}$, $t \in \mathcal{T}$. 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

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

where the subscript $i \in \{1, 2, \dots, n_1\}$ 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,

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

where $\boldsymbol{\phi}_i(t) = (\phi_{i1}(t), \dots, \phi_{iM_k}(t))^T$ is a vector of basis functions, and $\mathbf{c}_i = (c_{i1}, \dots, c_{iM_k})^T$ 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

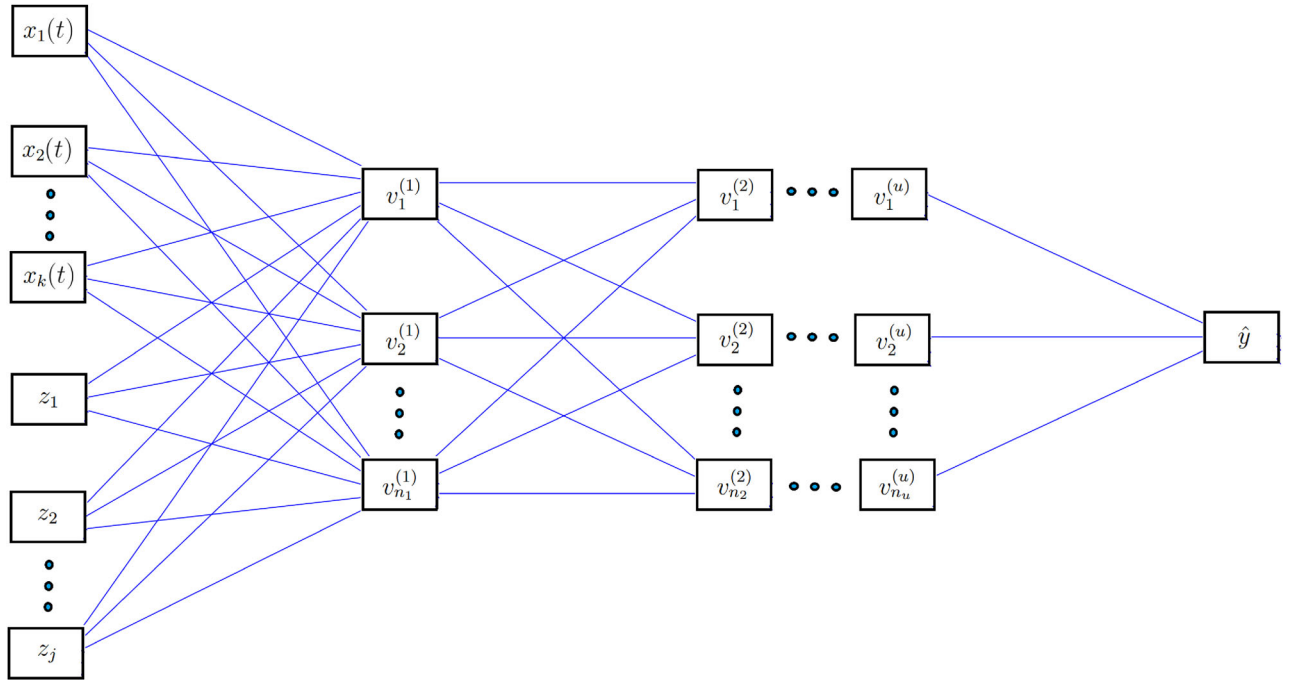


Figure 1. Schematic of a general functional neural network for when the inputs are functions, $x_k(t)$, and scalar values, z_j . The response/output of this network is a scalar value, \hat{y} .

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

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.

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

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

where

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

$\phi_{ikm}(t)$ 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

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

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 $g : \mathbb{R} \rightarrow \mathbb{R}$ 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 $\mathcal{C}(I_n)$:

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

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} , $\ell = 1, \dots, N$, is the observed data for the scalar response, and \hat{y}_{ℓ} 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 **backpropagation algorithm** (Rumelhart, Hinton, and Williams 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

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

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 dataset 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.

Finally, 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.

Algorithm 1: Functional Neural Networks

Input: $x_k(t)$, z_j for $k = 1, 2, \dots, K$ and $j = 1, 2, \dots, J$, γ , 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: The set of parameters defining the neural network, θ

1. Set Hyperparameters:
 γ , 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 $\int_{\mathcal{T}} \phi_{km}(t)x(t)dt = \tilde{\phi}_{km}$ for each basis function, m and for each functional covariate, k
 - 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
 - d. Pass E to the attached network architecture
 - e. Calculate loss: $R(\theta)$
 - 5ii. Backward Pass
 - a. Compute θ'
 - b. $\forall a \in \theta_p$, update a as: $a = a - \gamma \bar{a}$
 - 5iii. Stopping Criteria Updates
 - a. **If** $s = 0$: $q = R(\theta)$ **Else:** $q = q - R(\theta)$
 - b. $s = s + 1$
 6. Return $\theta = \theta_p$
-

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

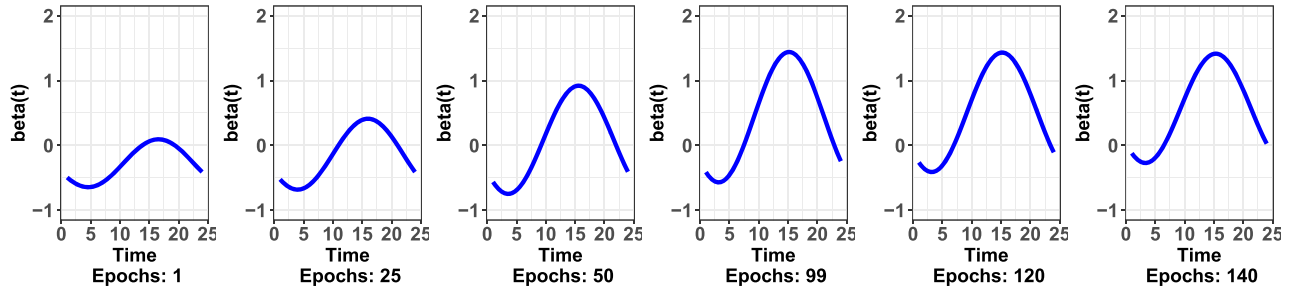


Figure 2. An example of how the functional weight, $\beta_{ik}(t)$, changes over the number of training iterations or epochs as they are referred to in neural network literature.

computer vision methods (Simonyan, Vedaldi, and Zisserman 2013; Zhang, Wu, and Zhu 2018). 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 neurons, 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.

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, Hastie, and Friedman 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 $MSPE = \sum_{b=1}^B \sum_{l \in S_b}^N (\hat{y}_l^{(-b)} - y_l)^2 / N$, where S_b is the b th partition of

the dataset, 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 dataset. 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 predefined 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. We standardize the number of epochs and validation splits across all methods as well. We do note however, that tuning for neural networks is a seemingly never-ending process and we could continue to tune by adjusting one of the many hyperparameters for such methods. Some additional information on tuning for other models is provided in Section 4.2.

3. Applications

3.1. Tecator Data

We consider the Tecator dataset (Thodberg 2015). The data are recorded on a Tecator Infratec Food and Feed Analyzer using near-infrared laser light (wavelength is 850–1050 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.

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, ($M_k = M_1 = 3$). Our goal with this example is to benchmark FNNs versus other modern methods in the realm of neural networks. Namely, we are interested to see how functional neural networks compare with convolutional neural networks (CNNs), long-short-term recurrent neural networks (Hochreiter and Schmidhuber 1997), the bidirectional variants (Graves and Schmidhuber 2005), and gated recurrent units (Chung et al. 2014). We tuned via the initial weights and by using early stop as mentioned earlier. With respect to the CNNs, we considered different filter and kernel sizes to see how much the results varied. We label them as CNN 1 through 4 and the exact configurations of each can be found in Table S4 in the supplementary materials.

Table 1 displays the 5-fold cross-validated mean-squared predication errors ($MSPE_{CV}$) and standard errors (SEs) of nine neural network models for the tecator dataset. The cross-validated MSPE of functional neural networks is lower than any of the other methods for this problem. Moreover, the parameter counts are lowest for the functional neural network with 4029; in contrast, the conventional neural network has 6357, the four variations of the CNNs have 87645, 43741, 85021, and 32325, respectively, LSTM has 22241, the bidirectional variant has 43233, and the GRU variant has 18017. The training plots for each can be found in Figure S2 in the supplementary materials. Not only do the functional neural networks outperform the other methods, they do so in a statistically significant manner when looking at the paired t -tests available in Table S6 of the supplementary materials. We also observe that a kernel size of 2 performs the best for the CNN variants. We also consider a kernel size of 3 (as well as 1 and 5 not shown here) and these configurations performed poorly.

3.2. 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 dataset (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 1 p.m. and 9 p.m., 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 modeling using nonfunctional

Table 1. The 5-fold cross-validated mean-squared predication errors ($MSPE_{CV}$) and standard errors (SEs) of nine neural network models for the tecator dataset.

Model	$MSPE_{CV}$	SE
CNN 1 (Kernel Size: 2; Filters: 64)	5.86	0.962
CNN 2 (Kernel Size: 2; Filters: 32)	5.19	0.967
CNN 3 (Kernel Size: 3; Filters: 64)	> 10	0.314
CNN 4 (Kernel Size: 2; Filters: 16)	6.24	0.959
LSTM	4.35	0.971
LSTM Bidirectional	5.50	0.964
GRU	5.20	0.966
Conventional neural networks	7.43	0.952
Functional neural networks	3.63	0.976

Table 2. The 10-fold cross-validated mean-squared predication errors ($MSPE_{CV}$), standard errors (SEs), and R^2 of nine models, including the functional neural network (FNN).

Model	$MSPE_{CV}$	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
Conventional neural networks	3.08	0.575	0.665
Functional neural networks	2.47	0.659	0.469

NOTE: The FNN performs the best with respect to both measures.

methods that is, there are 24 covariates as opposed to the single functional observation.

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, Ferraty, and Sarda 1999), a nonparametric functional linear model (Ferraty and Vieu 2006), and a functional partial least squares model (Preda, Saporta, and Lévêder 2007). We also compare with conventional feed-forward neural networks (multilayer perceptron) as well 1D convolutional neural networks (LeCun, Bengio, and Hinton 2015). The are 3521, 4001, and 15649 parameters in the FNN, NN, and CNN models used here, respectively. The results are summarized in Table 2. The final model configurations for the neural networks are provided in Section S4.2 in the supplementary materials. 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 S10 along with training plots (Figure S3).

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

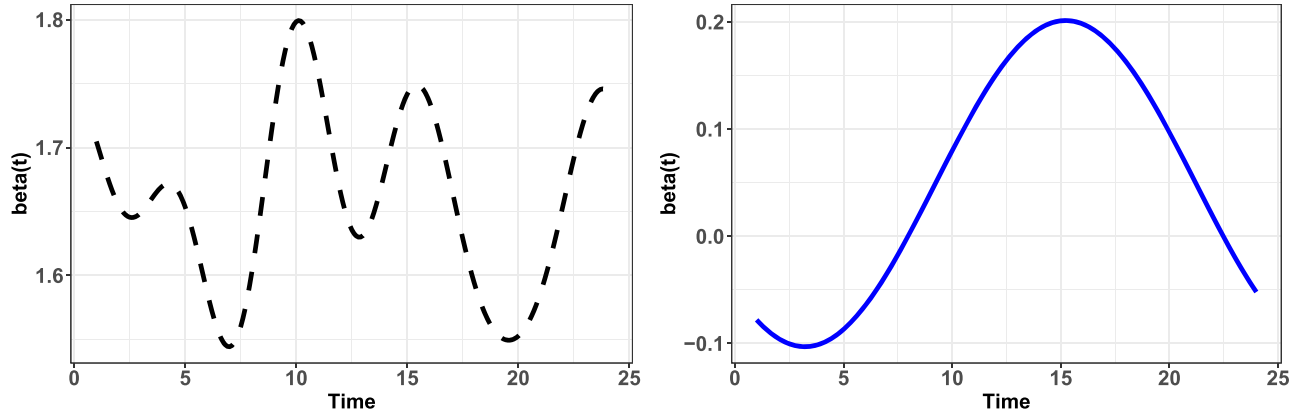


Figure 3. The estimated functional weight $\hat{\beta}(t)$ in the conventional functional linear model (dashed curve) and the functional neural network (solid curve) for the bike rental dataset as a function of time.

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. The full plot list of estimated weights from each fold can be found in Figure S4 in the supplementary materials.

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

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

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

$$y_{\ell}^* = g \left(\alpha + \int_{\mathcal{T}} \beta(t) x_{\ell}(t) dt \right) + \epsilon_{\ell}^*, \quad \ell = 1, \dots, L, \quad (8)$$

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, \dots, m_5 , are set beforehand. We fit the functional linear model (Equation (1)) and the FNN for each simulation dataset. The functional linear model (Equation (1)) is estimated by minimizing the penalized sum squared errors defined as

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

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 2005) 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 following scenarios. These simulations are summarized as follows:

$$\text{Simulation1} : y^* = \alpha + \int_{\mathcal{T}} \beta(t) x(t) dt + \epsilon^* \quad (9)$$

$$\text{Simulation2} : y^* = \exp \left(\alpha + \int_{\mathcal{T}} \beta(t) x(t) dt \right) + \epsilon^* \quad (10)$$

$$\text{Simulation3} : y^* = \frac{1}{1 + \exp \left(\alpha + \int_{\mathcal{T}} \beta(t) x(t) dt \right)} + \epsilon^* \quad (11)$$

$$\text{Simulation4} : y^* = \log \left(\left| \alpha + \int_{\mathcal{T}} \beta(t) x(t) dt \right| \right) + \epsilon^*. \quad (12)$$

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 behavior 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

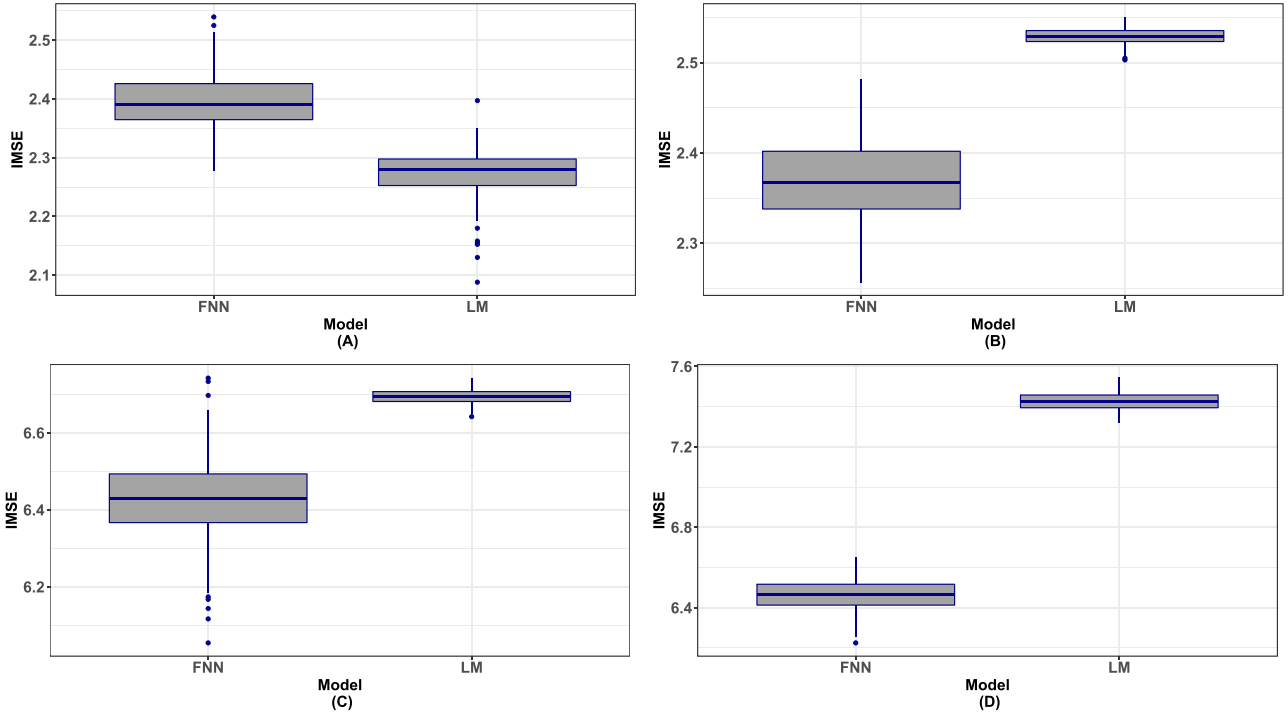


Figure 4. Boxplots of square root of the integrated mean square errors (IMSEs) over 250 simulation replicates for four scenarios. Plot (A) is for when we use the identity link function. Plot (B) are results from the exponential link. The bottom plots, (C) and (D) are the results from simulation 3 and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

in the exponential example. In all but the third scenario, we use a three-layer network with rectifier (Hahnloser et al. 2000) and linear activation functions. In the third scenario, we use a one-layer network with a sigmoidal activation function.

In Figure 4, 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 nonlinear, 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 3. 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 boxplots indicate, that for nonlinear 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 that is, random weight initialization of functional neural networks, there is a higher variance in our estimates when compared with the deterministic functional linear model.

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

Table 3. The square root of the mean along with the associated standard error (SE) of the integrated mean square errors (IMSEs) over 250 simulation replicates are provided for both the functional neural networks and the functional linear model.

	Functional linear model			Functional neural networks			Paired T -test p -Value
	Mean	SE	Avg. Comp. Time	Mean	SE	Avg. Comp. Time	
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

NOTE: The computation times given are the average per simulation replication.

(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 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 boxplots in the supplemental document (Figure S10) measure the relative error in each simulation replicate. We call this the relative MSPE defined as

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

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 S19 contains the exact error rates and Table S20 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 S10 in the supplementary materials). 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 observed 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 S5.2 in the supplementary materials.

5. Conclusions and Discussion

In this article, 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 nonlinear. 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.

Supplementary Materials

Data and R Codes: Data and R Codes used for each application and simulation studies presented in this article are available at <https://github.com/caojiguo/FNN>; this includes some of the source for the **FuncNN** R package. A README file is included which describes the files. (FNN_Code.zip)

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

Acknowledgments

The authors thank to the editor, the associate editor, and two anonymous referees for many insightful comments. These comments are very helpful for us to improve our work. The authors report there are no competing interests to declare.

Funding

This research is supported by the Strategic Partnership grant and the Discover grant to J. Cao from the Natural Sciences and Engineering Research Council of Canada (NSERC).

ORCID

Barinder Thind  <https://orcid.org/0000-0002-9264-5732>

Kevin Multani  <https://orcid.org/0000-0002-5350-6502>

Jiguo Cao  <http://orcid.org/0000-0001-7417-6330>

References

- Ainsworth, L., Routledge, R., and Cao, J. (2011), “Functional Data Analysis in Ecosystem Research: The Decline of Oweekeno Lake Sockeye Salmon and Wannock River Flow,” *Journal of Agricultural, Biological, and Environmental Statistics*, 16, 282–300. [1]
- Aneiros-Pérez, G., and Vieu, P. (2006), “Semi-Functional Partial Linear Regression,” *Statistics & Probability Letters*, 76, 1102–1110. [1]
- Breiman, L. (2001), “Random Forests,” *Machine Learning*, 45, 5–32. [8]
- Cao, J., and Ramsay, J. (2010), “Linear Mixed-Effects Modeling by Parameter Cascading,” *Journal of the American Statistical Association*, 105, 365–374. [1]
- Cardot, H., Ferraty, F., and Sarda, P. (1999), “Functional Linear Model,” *Statistics & Probability Letters*, 45, 11–22. [1,6]
- Chen, T., He, T., Benesty, M., Khotilovich, V., and Tang, Y. (2015), “Xgboost: Extreme Gradient Boosting,” *R package version 0.4-2*, 1–4. [8]
- Chollet, F., et al. (2015), “Keras.” Available at https://keras.io/getting_started/faq/#how-should-i-cite-keras. [5]
- Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014), “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” arXiv preprint arXiv:1412.3555. [1,6]
- Cybenko, G. (1989), “Approximation by Superpositions of a Sigmoidal Function,” *Mathematics of Control, Signals and Systems*, 2, 303–314. [3]
- Dong, J., Wang, L., Gill, J., and Cao, J. (2018), “Functional Principal Component Analysis of Glomerular Filtration Rate Curves after Kidney Transplant,” *Statistical Methods in Medical Research*, 27, 3785–3796. [1]
- 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*, 5, 741–760. [4]
- Fanaee-T, H., and Gama, J. (2014), “Event Labeling Combining Ensemble Detectors and Background Knowledge,” *Progress in Artificial Intelligence*, 2, 113–127. [6]

- Ferraty, F., and Vieu, P. (2006), *Nonparametric Functional Data Analysis: Theory and Practice*, New York: Springer-Verlag. [1,6]
- Friedman, J. (2001), “Greedy Function Approximation: A Gradient Boosting Machine,” *Annals of Statistics*, 29, 1189–1232. [8]
- Friedman, J., and Stuetzle, W. (1981), “Projection Pursuit Regression,” *Journal of the American Statistical Association*, 76, 817–823. [8]
- Graves, A., and Schmidhuber, J. (2005), “Frame-wise Phoneme Classification with Bidirectional LSTM and other Neural Network Architectures,” *Neural Networks*, 18, 602–610. [1,6]
- Guan, T., Lin, Z., and Cao, J. (2020), “Estimating Truncated Functional Linear Models with a Nested Group Bridge Approach,” *Journal of Computational and Graphical Statistics*, 29, 620–628. [1]
- Guan, T., Nguyen, R., Cao, J., and Swartz, T. (2022), “In-game win Probabilities for the National Rugby League,” *The Annals of Applied Statistics*, 16, 349–367. [1]
- 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. [2,8]
- 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. [2]
- He, K., Zhang, X., Ren, S., and Sun, J. (2016), “Deep Residual Learning for Image Recognition,” in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. [2]
- Hochreiter, S., and Schmidhuber, J. (1997), “Long Short-Term Memory,” *Neural Computation*, 9, 1735–1780. [1,2,6]
- Jiang, C.-R., and Wang, J.-L. (2011), “Functional Single Index Models for Longitudinal Data,” *The Annals of Statistics*, 39, 362–388. [1]
- 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* (Vol. 3), pp. 2396–2401. [5]
- Kingma, D., and Ba, J. (2014), “Adam: A Method for Stochastic Optimization,” arXiv preprint arXiv:1412.6980. [2,4]
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012), “Imagenet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*. [2]
- LeCun, Y., Bengio, Y., and Hinton, G. (2015), “Deep Learning,” *Nature*, 521, 436–444. [6]
- Lin, Z., Cao, J., Wang, L., and Wang, H. (2017), “Locally Sparse Estimator for Functional Linear Regression Models,” *Journal of Computational and Graphical Statistics*, 26, 306–318. [1]
- Liu, B., Wang, L., and Cao, J. (2017), “Estimating Functional Linear Mixed-Effects Regression Models,” *Computational Statistics & Data Analysis*, 106, 153–164. [1]
- Luo, W., Cao, J., Gallagher, M., and Wiles, J. (2013), “Estimating the Intensity of Ward Admission and its Effect on Emergency Department Access Block,” *Statistics in Medicine*, 32, 2681–2694. [1]
- Morris, J. S. (2015), “Functional Regression,” *Annual Review of Statistics and Its Application*, 2, 321–359. [9]
- Müller, H.-G., and Stadtmüller, U. (2005), “Generalized Functional Linear Models,” *The Annals of Statistics*, 33, 774–805. [1]
- Nie, Y., Wang, L., and Cao, J. (2017), “Estimating Time-Varying Directed Gene Regulation Networks,” *Biometrics*, 73, 1231–1242. [1]
- Preda, C., Saporta, G., and Lévêder, C. (2007), “PLS Classification of Functional Data,” *Computational Statistics*, 22, 223–235. [1,6]
- Ramsay, J., and Silverman, B. (2005), *Functional Data Analysis*, New York: Springer. [1,2,5,7]
- 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. [5]
- Rossi, F., and Conan-Guez, B. (2005), “Functional Multi-Layer Perceptron: A Non-Linear Tool for Functional Data Analysis,” *Neural Networks*, 18, 45–60. [2]
- Ruder, S. (2016), “An Overview of Gradient Descent Optimization Algorithms,” arXiv preprint arXiv:1609.04747. [2,4]
- Rumelhart, D., Hinton, G., and Williams, R. (1985), “Learning Internal Representations by Error Propagation,” Technical Report, California Univ San Diego La Jolla Inst for Cognitive Science. [4]
- Seber, G. A., and Lee, A. J. (2012), *Linear Regression Analysis*, Hoboken: Wiley. [2]
- Simonyan, K., Vedaldi, A., and Zisserman, A. (2013), “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,” arXiv preprint arXiv:1312.6034. [5]
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014), “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, 15, 1929–1958. [2]
- 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. [2]
- Thodberg, H. H. (2015), “Tecator Meat Sample Dataset,” *StatLib Datasets Archive*. [5]
- Tibshirani, R. (1996), “Regression Shrinkage and Selection via the Lasso,” *Journal of the Royal Statistical Society, Series B*, 58, 267–288. [8]
- Tibshirani, R., Hastie, T., and Friedman, J. (2009), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York: Springer. [2,5]
- Yao, Y., Rosasco, L., and Caponnetto, A. (2007), “On Early Stopping in Gradient Descent Learning,” *Constructive Approximation*, 26, 289–315. [2]
- 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. [5]