Functional Neural Networks for Scalar Prediction

Barinder Thind

Simon Fraser University

Masters Defence

April 7, 2020

Overview

Methodology

- Neural Networks
- Functional Data
- Functional Neural Networks

Results

- Real World Data
- Simulation Studies

Conclusions

- Recap
- References

Section I

Methodology

Layers & Neurons - Setup

- Made up of *layers*, each comprised of some number of *neurons*
  - Index these as n_1 to n_u where the subscript u refers to the u^{th} hidden layer and n_u is the number of neurons in that particular layer
- For example, 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 $\boldsymbol{W}^{(1)}$ and $\boldsymbol{b}^{(1)}$ are the weights and biases and $g(\cdot)$ is some *activation function* that transforms the resulting linear combination

Layers & Neurons - Dimensionality

- The dimensionality of $\boldsymbol{W}^{(1)}$ would be $(n_1 \times J)$ where J is the number of scalar covariates associated with each observation, i
- The vector $\boldsymbol{x}$ corresponds to a single observation of our data set
- The resulting vector $\boldsymbol{v}^{(1)}$ is n_1-dimensional
  - Passed onto the next layer
- Process repeated for all u layers $\Longrightarrow$ forward pass complete

Backpropagation

- Now we have our predictions, what next?
- Use calculus to minimize some loss function
  - Minimize with respect to the parameters of the network: the weights and biases
  - We update them: $a = a - \frac{\partial R}{\partial a}\gamma$ where a is some parameter, R is the loss function (think: MSE), and γ is a learn rate
- Basic approach: stochastic gradient descent but there are better approaches
  - e.g. Incorporating "momentum"

Backpropagation Jargon

- The derivative of a is actually different for each observation
- We update based on the average of the derivatives
- The average usually doesn't come from the full data set because it's often computationally expensive to compute all those derivatives simultaneously
- We do it in "mini batches" - partitions of the data
- Once we have done updates based on all partitions, we have completed one "epoch"
- Updates are repeated for some number of epochs

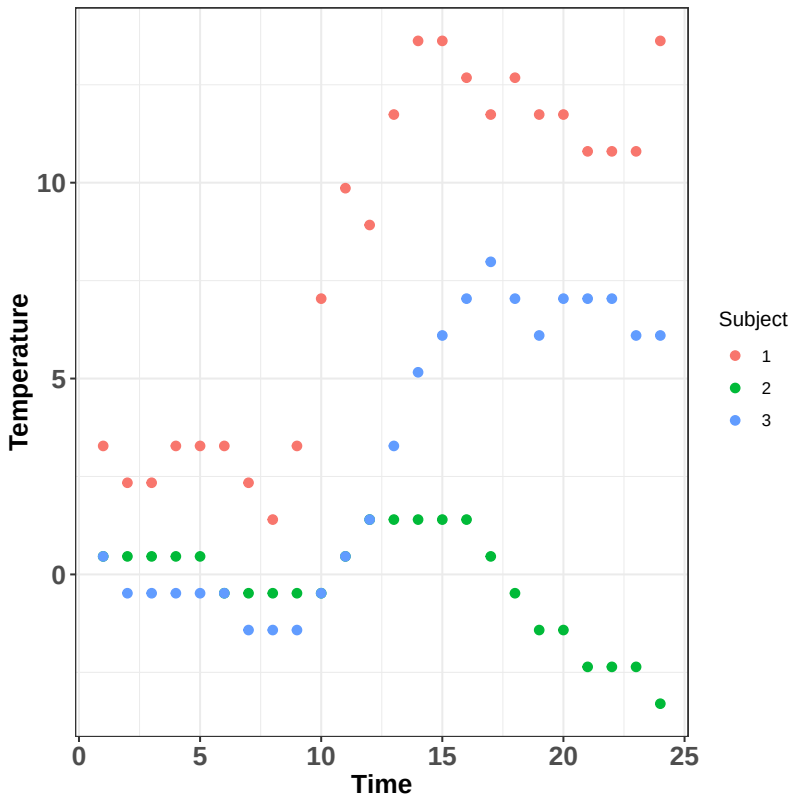
We've Made an Assumption

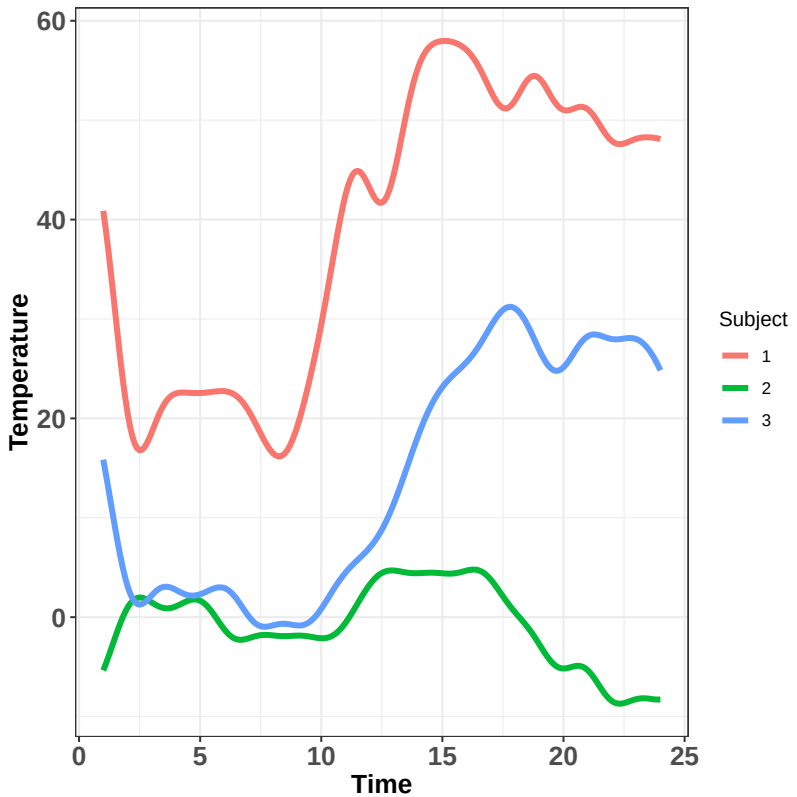
- We've assumed that $\boldsymbol{x}$ is J dimensional
- We wish now to consider the case when our input is infinite dimensional defined over some domain $\mathcal{S}$
  - In other words, what if we want to pass in **functional data**?

What is Functional Data?

- You have repeat observations for some set of subjects
  - Traditionally, you look at them as though they were discrete
  - What if there was an underlying data generating function?
  - We can analyze the functions instead

Illustration of Functional Data





Some Questions about Functional Data

- Here are some questions:
  - How do we estimate these functions?
  - What are the advantages of FDA?
  - What kind of models can we build?

Some Answers about Functional Data

- **How do we estimate these functions?**
  - We can approximate the underlying function using basis expansions
- **What are the advantages of FDA?**
  - We can observe the effect of the covariate at any point along the continuum. We can also take derivatives of observations to observe rates of change - those can also be covariates.
- **What kind of models can be build?**
  - Functional neural networks!

Functional Neural Networks

- We now know the form of our input so let's put it together
- Must weigh the covariate $x(s)$, at every point along its domain and so, our weight parameter must be infinite dimensional as well
- Define this coefficient as $\beta(s)$

$$v_n^{(1)} = g\left(\int_{\mathcal{S}} \beta_n(s)x(s)ds + b_n^{(1)}\right)$$

Some Neuron Manipulation

- The functional coefficient $\beta(s)$, is cemented in the network as its M-term basis expansion

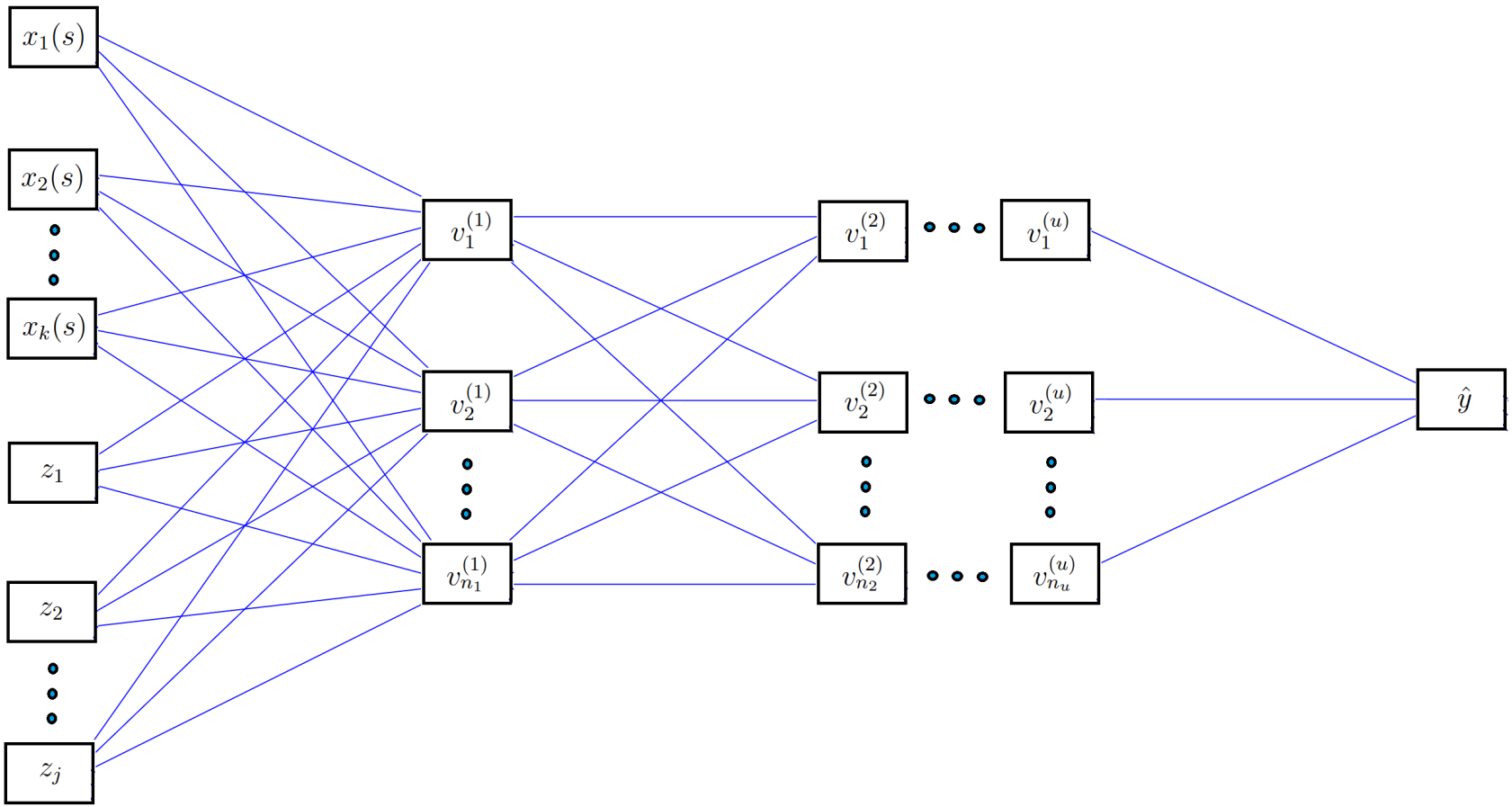
$$
\begin{aligned}
v_n^{(1)} &= g\left(\int_{\mathcal{S}} \beta_n(s)x(s)ds + b_n^{(1)}\right) \\
&= g\left(\int_{\mathcal{S}} \sum_{m=1}^{M} c_{nm}\phi_{nm}(s)x(s)ds + b_n^{(1)}\right) \\
&= g\left(\sum_{m=1}^{M} c_{nm}\int_{\mathcal{S}} \phi_{nm}(s)x(s)ds + b_n^{(1)}\right)
\end{aligned}
$$

Generalizing

- We can now consider the generalization for K functional covariates and J scalar covariates.

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

Schema



Generalizing Cont. (1)

- Using this input set, we can define the general formulation of the first hidden layer as:

$$v_n^{(1)} = g\left(\sum_{k=1}^{K}\int_{\mathcal{S}}\sum_{m=1}^{M} c_{kmn}\phi_{kmn}(s)x_k(s)ds + \sum_{j=1}^{J} w_{jn}^{(1)} z_j + b_n^{(1)}\right)$$

$$= g\left(\sum_{k=1}^{K}\sum_{m=1}^{M} c_{kmn}\int_{\mathcal{S}} \phi_{kmn}(s)x_k(s)ds + \sum_{j=1}^{J} w_{jn}^{(1)} z_j + b_n^{(1)}\right).$$

As Universal Approximators

- Adding this additional term to the neuron preserves one of the defining features of neural networks
- In other words, we have shown that networks of the form $h(s)$ are universal approximators

$$h(s) = \sum_{n=1}^{n_1} \Psi_n g \left(\sum_{k=1}^{K} \left(\int_{\mathcal{S}} \beta_{nk}(s) x_k(s) ds \right) + \sum_{j=1}^{J} w_{nj} z_j + b_n \right)$$

Network Training

- 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:

$$\theta' = \left\{ \bigcup_{k=1}^{K} \bigcup_{m=1}^{M_k} \bigcup_{n=1}^{n_1} \frac{\partial R}{\partial c_{kmn}}, \bigcup_{u=1}^{U} \bigcup_{j=1}^{J_u} \bigcup_{n=1}^{n_u} \frac{\partial R}{\partial w_{ujn}}, \bigcup_{u=1}^{U} \bigcup_{n=1}^{n_u} \frac{\partial R}{\partial b_{un}} \right\}$$

- We minimize for this set of parameters using the same optimization techniques as in the usual neural network

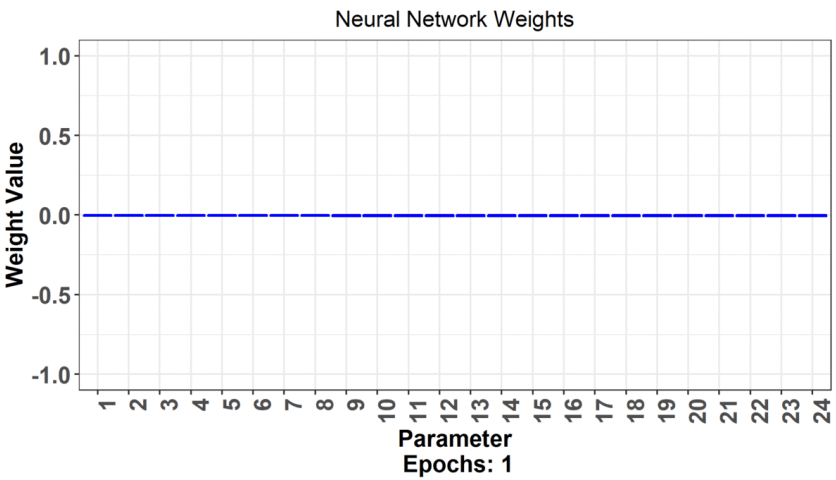
Parameter Counts

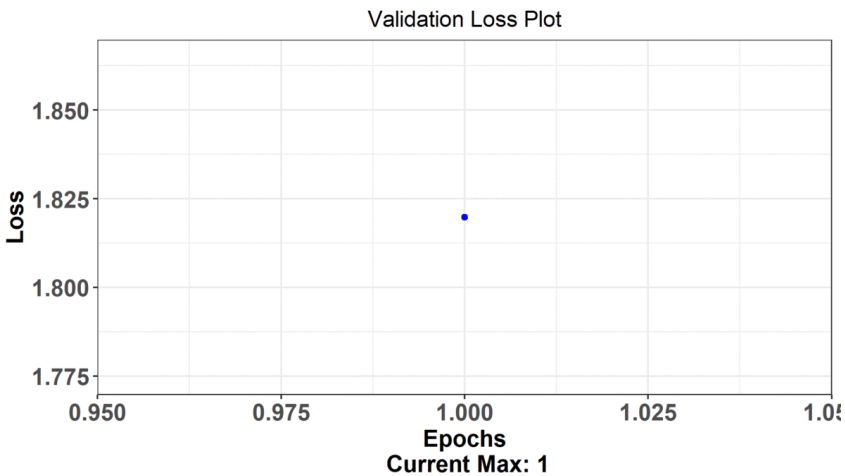
- Number of parameters in the network presented here has decreased significantly
- Consider a longitudinal data set where we have n observations and J_{ml} scalar repeat measurements of some covariate at different points along a continuum
  - Number of parameters in the first layer for a usual neural network would be $(J_{ml} + 1) \cdot n_1$
  - Number of parameters for an FNN will be: $(M + 1) \cdot n_1$ where $M < J_{ml}$

Functional Neural Coefficients

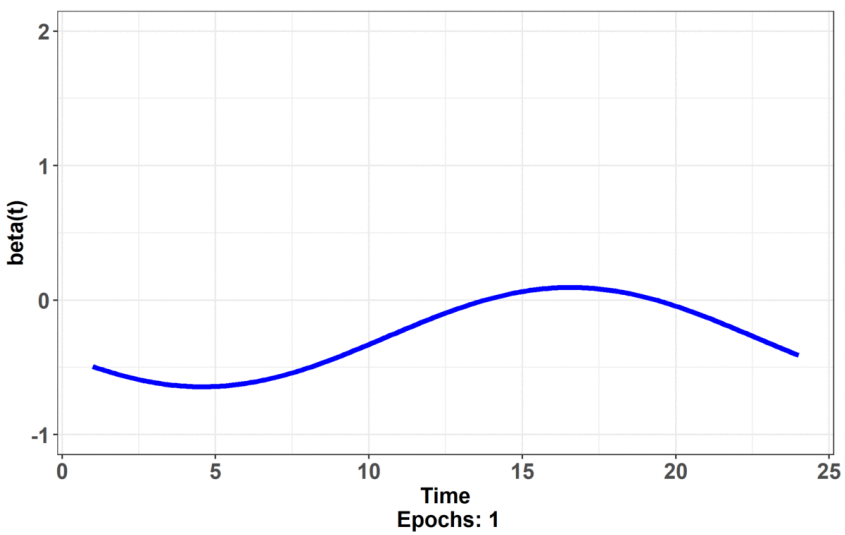
- Something to think about: it's difficult to make sense of the way parameter values change in a neural network - this contributes to the black-box reputation of such models
- Counter-intuitively, it's easier to try and make sense of these parameters when they are infinite dimensional!
  - This is because we can visualize these objects - they are functions

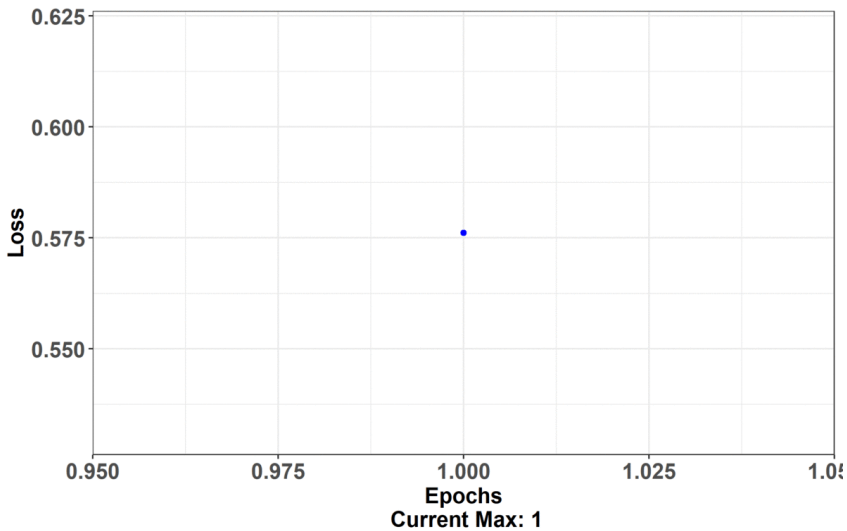
What is Going On?





Functional Neural Coefficient Demonstration





Computing Functions

- We are developing a package for general use
- There are 5 main functions:
  - `FNN()`
  - `FNN_Predict()`
  - `FNN_FNC()`
  - `FNN_CV()`
  - `FNN_Tune()`

Section III

Results

Bike Data Example

- Let's revisit this example
- In total, there were 102 functional observations (the daily temperature readings) made using a 31-basis fourier expansion
- Predictions are made for the number of daily bike rentals
- We measure two criteria
  - $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$
  - $\text{MSPE}_{CV} = \frac{\sum_i \left(x_i^{\text{CV}} - x_i^{\text{true}}\right)^2}{n}$
- In this case, we use a 10-fold cross-validation to obtain the results

Bike Prediction Results

Model	MSPE_{CV}	R^2
Functional Linear Model (Basis)	0.0723	0.515
Functional Non-Parametric Regression	0.143	0.154
Functional PC Regression	0.0773	0.503
Functional PC Regression (2nd Deriv Penalization)	0.128	0.0481
Functional PC Regression (Ridge Regression)	0.0823	0.464
Functional Partial Least Squares	0.0755	0.458
Functional Partial Least Squares (2nd Deriv Penalization)	0.0701	0.545
Functional Neural Networks	**0.0669**	**0.582**

Table 1: The italics indicate second- best model performance, whereas the bolded cells indicate best performance.

Tecator Data Example

- The goal is to predict the fat contents of some 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 a sample
  - Samples with higher water composition may exhibit different spectral features than samples with higher protein content

Tecator Data Example Cont. (1)

- We use the same set up as (Febrero Bandeand Oviedo de la Fuente, 2012)
- They compared a number of methods using this data set by making predictions on the final 50 observations of the data (in total, there are 215 absorbance curves)
- We use a 29 fourier basis expansion to build the functional observations
- The authors used: $\text{MEP} = \frac{\text{MSPE}}{\text{Var}(y)}$
- We use the second derivative as the functional covariate and water as a scalar covariate for this problem

Tecator Prediction Results

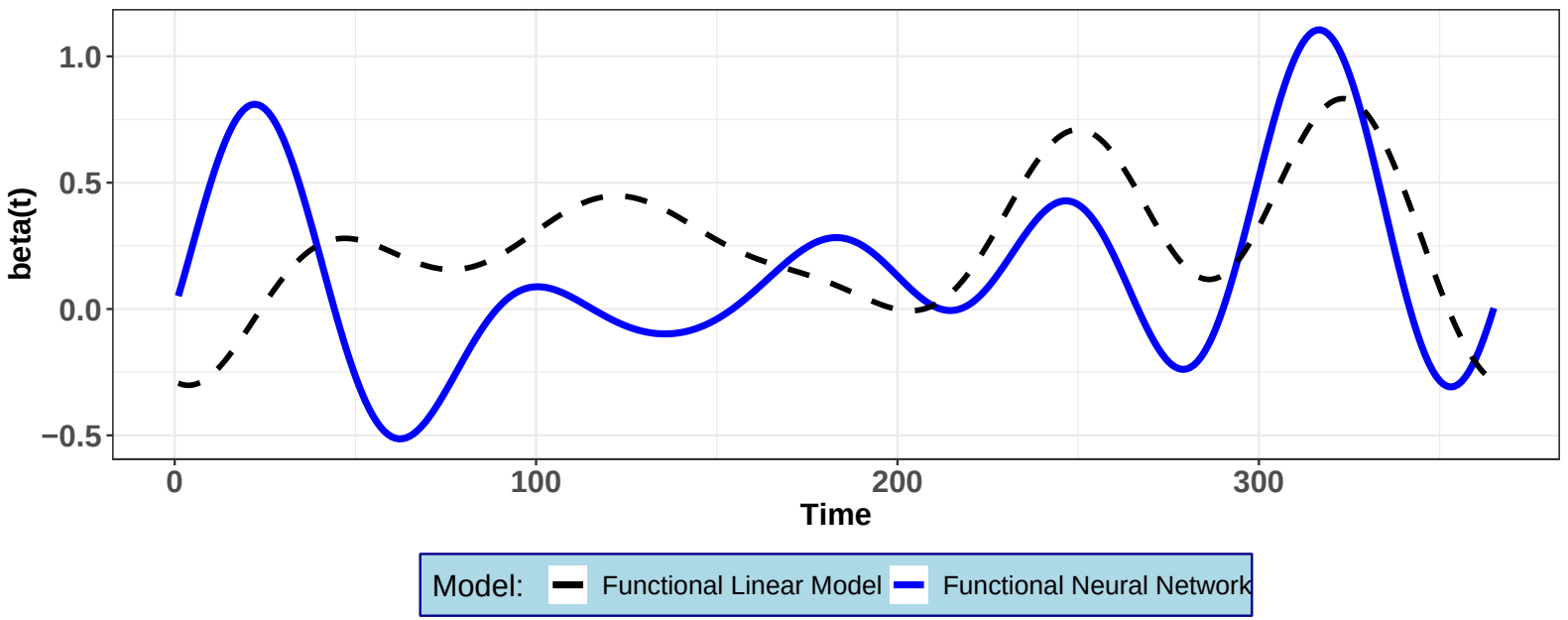
Model	MEP	R^2
fregre.basis(X.d1, Fat)	0.0626	0.928
fregre.basis.cv(X.d2, Fat)	0.0566	0.965
fregre.pc(X.d1, Fat)	0.0580	0.950
fregre.pc(X.d2, Fat)	0.0556	0.954
fregre.pls(X.d1, Fat)	0.0567	0.951
fregre.pls(X.d2, Fat)	0.0487	0.962
fregre.lm(Fat, X.d1 + Water)	0.0097	0.987
fregre.lm(Fat, X.d2 + Water)	0.0119	0.986
fregre.np(X.d1, Fat)	0.0220	0.987
fregre.np(X.d2, Fat)	0.0144	0.996
fregre.plm(Fat, X.d1 + Water)	0.0090	0.996
fregre.plm(Fat, X.d2 + Water)	0.0115	**0.997**
FNN(Fat, X.d2 + Water)	**0.00883**	0.965

Weather Prediction Results

Model	$\texttt{MSPE}_{CV}$	R^2
Functional Linear Model (Basis)	0.123	0.00312
Functional Non-Parametric Regression	0.0561	0.0900
Functional PC Regression	0.0272	0.352
Functional PC Regression (2nd Deriv Penalization)	0.0930	0.00298
Functional PC Regression (Ridge Regression)	0.0259	0.382
Functional Partial Least Squares	0.0449	0.177
Functional Partial Least Squares (2nd Deriv Penalization)	0.0483	0.155
Neural Networks	0.126	0.0453
Functional Neural Networks	**0.0194**	**0.541**

Table 2: The italics indicate second- best model performance, whereas the bolded cells indicate best performance.

Weather Coefficient Functions



Simulations - General

- Two main goals with simulations: Recovery of $\beta(s)$ and prediction
- In both set-ups, we generate 300 functional observations *per* simulation
- For recovery, we measure our results across 250 simulations
- For prediction, we use 100 simulations

Simulations - General Cont. (1)

- The response is generated using:

$$y^* = g\left(\alpha + \int_{\mathcal{S}} \beta(s)(\sum_i \phi_i \psi_i(s))ds\right) + \epsilon^*$$

- α is sampled from the uniform distribution, $X \sim \mathcal{U}(d, e)$
- ϵ^* is sampled from the Gaussian distribution, $Y \sim \mathcal{N}(0, 1)$
- The true data from which $x(s)$ is generated comes from either $a \cdot sin(a) + b$ or $c \cdot \exp(a) + sin(a) + b$, where $a \sim \mathcal{N}(0, 1)$, $b \sim \mathcal{N}(0, \frac{i}{100})$, and $c \sim \mathcal{N}(0, 1)$ are parameters that govern the difference between observations

Simulation Scenarios

$$\text{Simulation 1} : y^* = \alpha + \int_{\mathcal{S}} \beta(s)(\sum_i \phi_i \psi_i(s))ds + \epsilon^*$$

$$\text{Simulation 2} : y^* = \exp\left(\alpha + \int_{\mathcal{S}} \beta(s)(\sum_i \phi_i \psi_i(s))ds\right) + \epsilon^*$$

$$\text{Simulation 3} : y^* = \frac{1}{1 + \exp\left(\alpha + \int_{\mathcal{S}} \beta(s)(\sum_i \phi_i \psi_i(s))ds\right)} + \epsilon^*$$

$$\text{Simulation 4} : y^* = \log\left(\left|\alpha + \int_{\mathcal{S}} \beta(s)(\sum_i \phi_i \psi_i(s))ds\right|\right) + \epsilon^*.$$

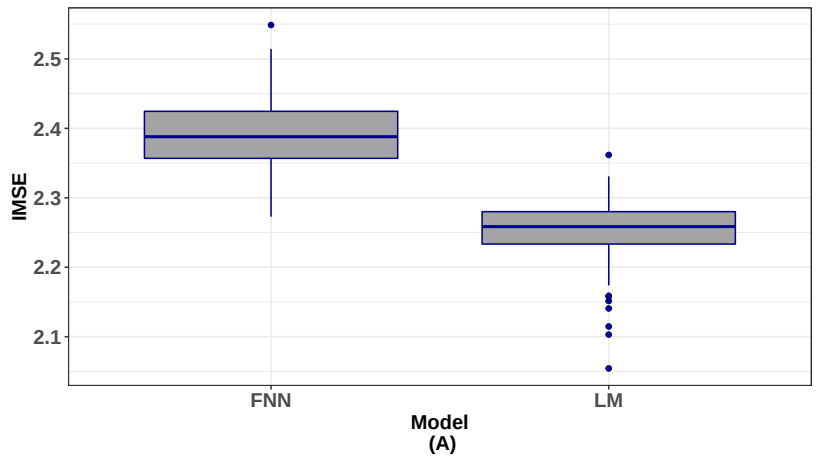
Simulations - Coefficient Recovery Setup

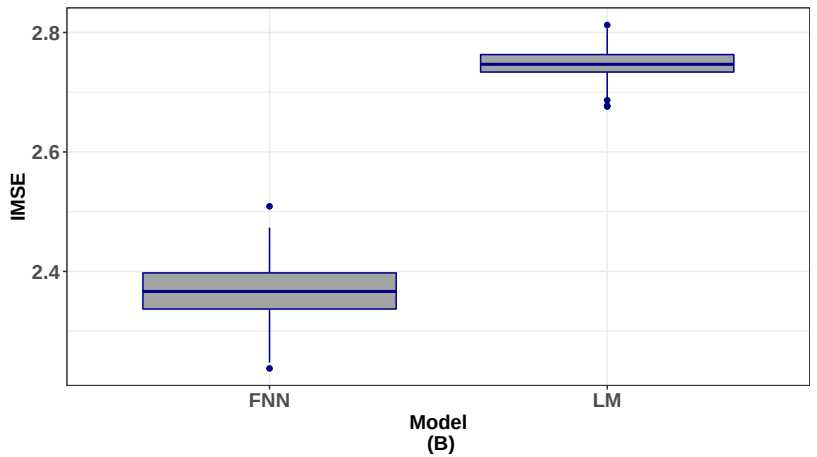
- We use integrated mean squared error to see how well the recovery of $\beta(s)$ is done relative to the functional linear model

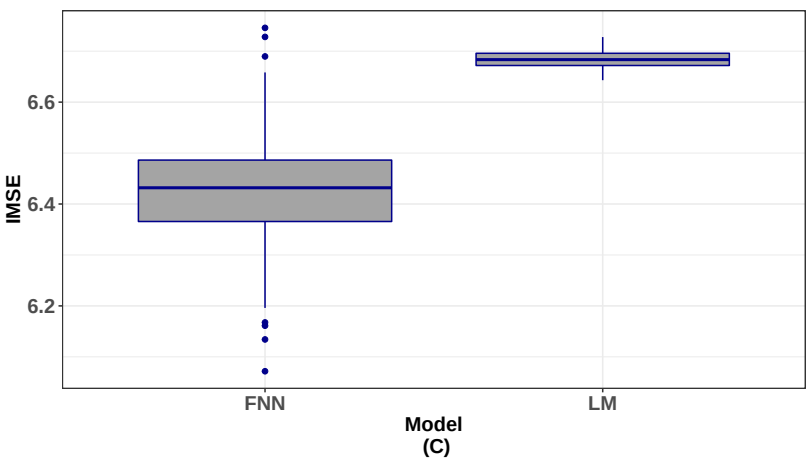
$$\text{IMSE} = \frac{1}{|\mathcal{S}|}\int_{\mathcal{S}}(\beta(s) - \hat{\beta}(s))^2 ds$$

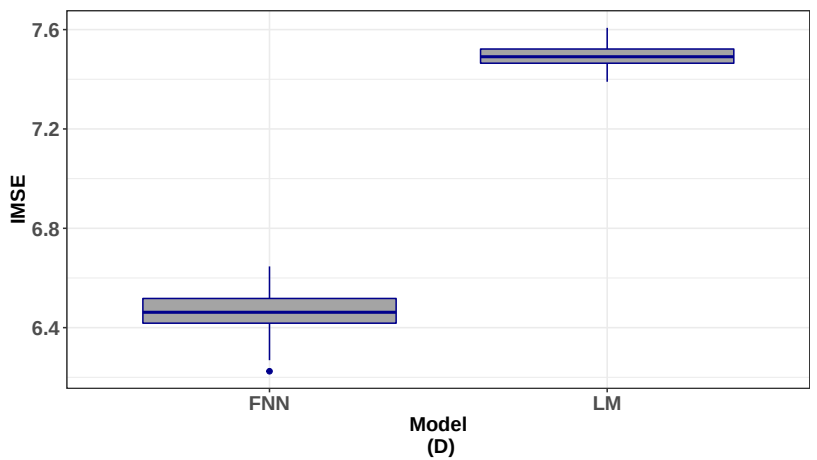
- We cross-validate over a grid for λ in order to find a smooth (and less volatile) estimate of $\beta(s)$ from the functional linear model

Simulations - Coefficient Recovery Results







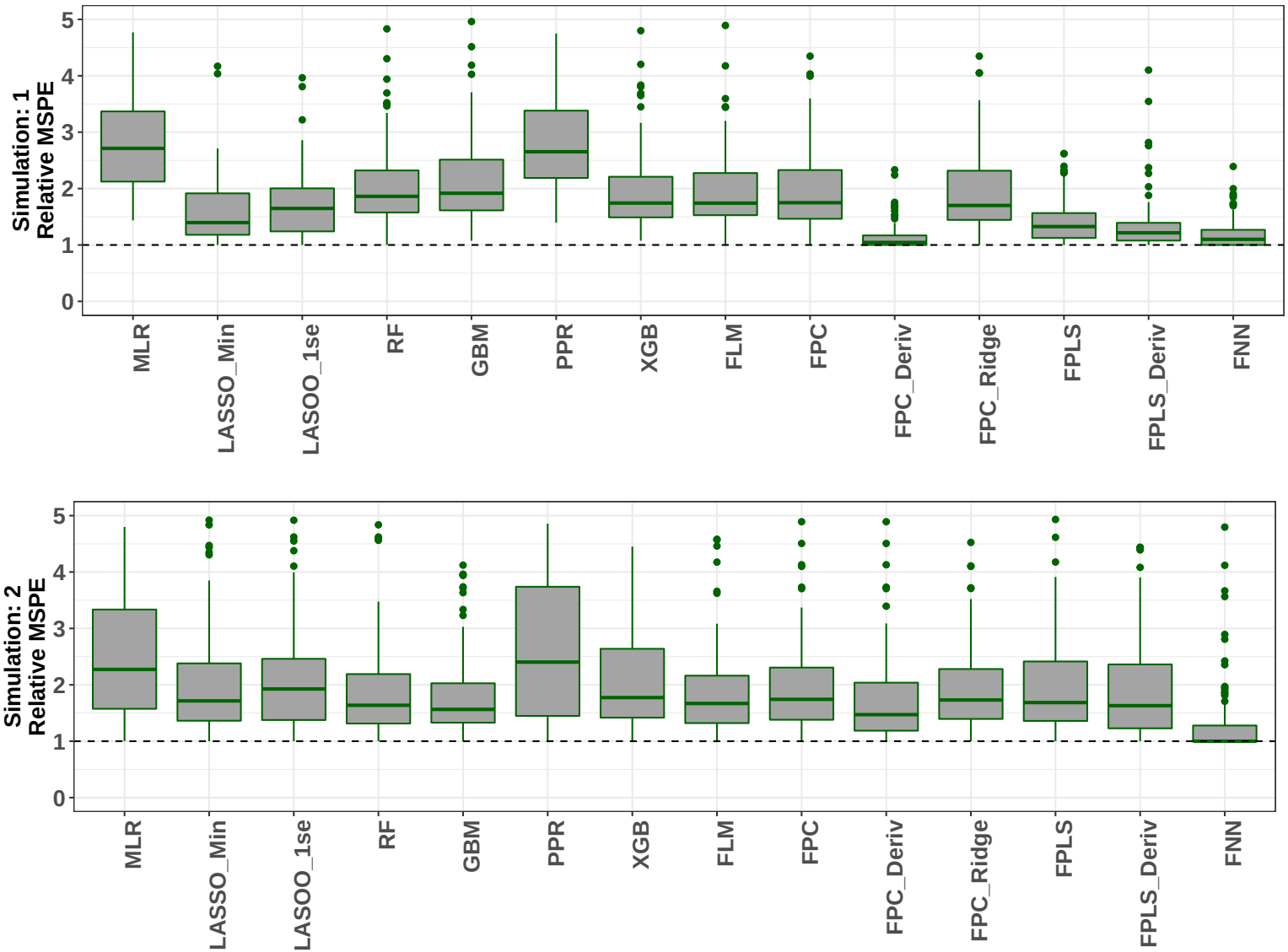


Simulations - Prediction

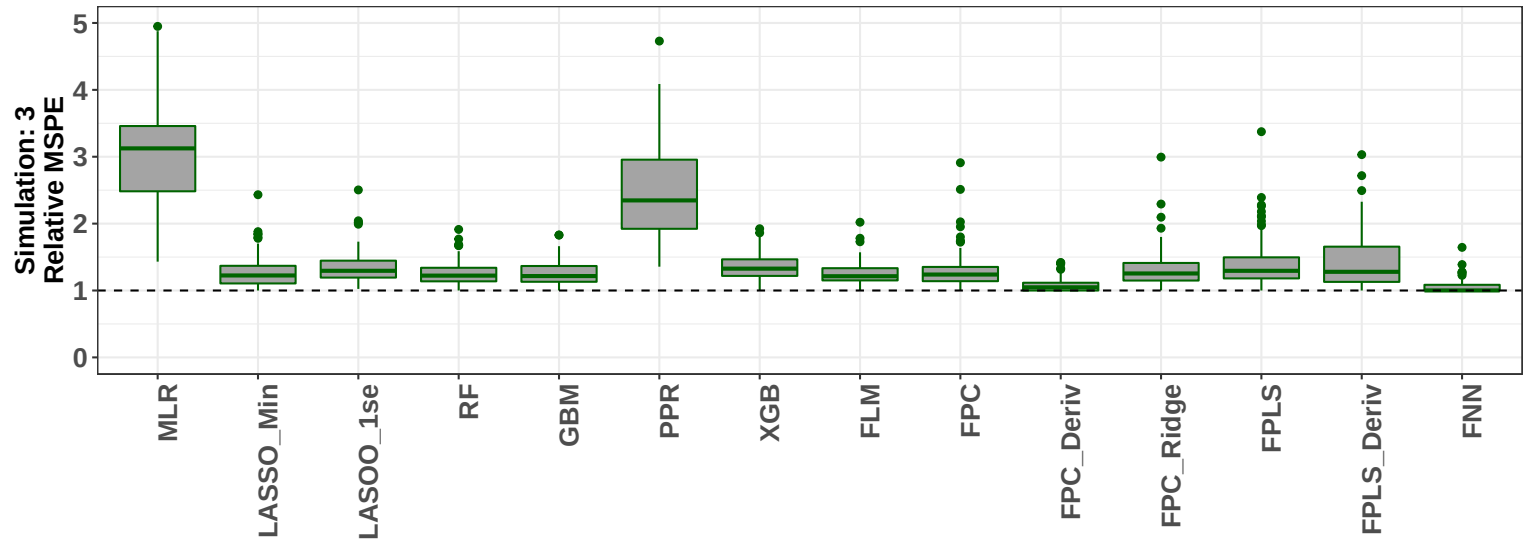
- We are interested in seeing how FNNs perform versus functional and multivariate approaches
- The multivariate methods include: least squares, LASSO, random forests, gradient boosting, xgboost, and projection pursuit regression
- We use the relative MSPE to measure performance:

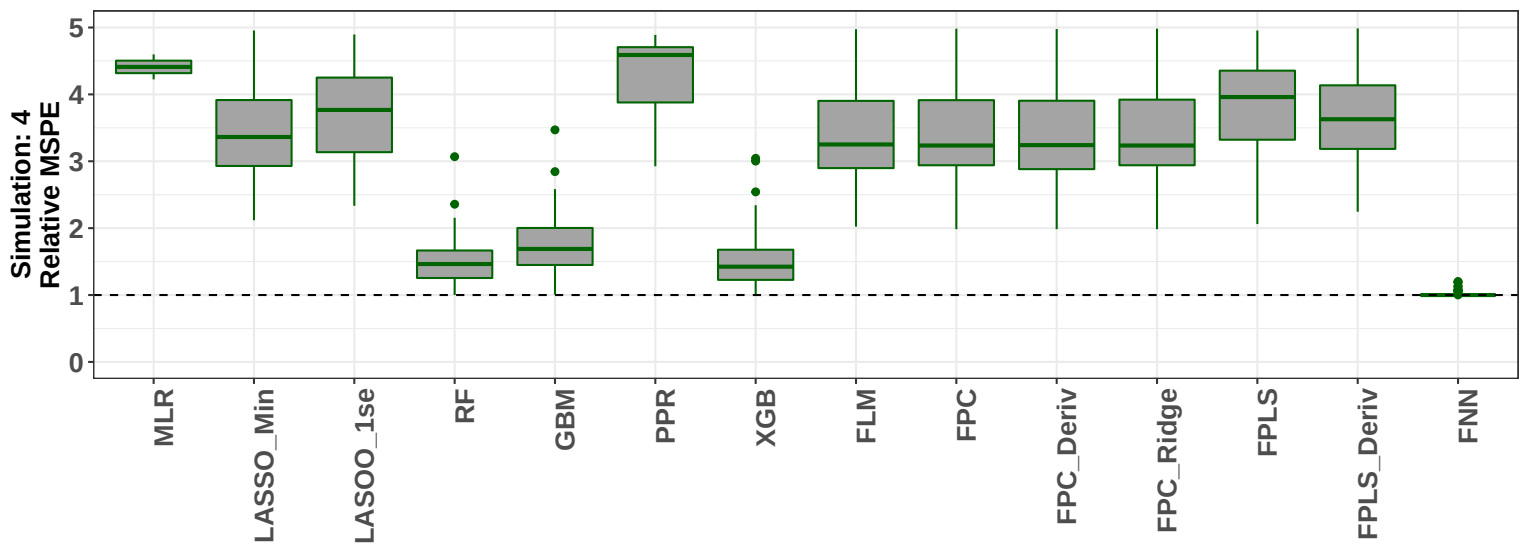
$$\text{rMSPE} = \frac{\text{MSPE}}{\min\limits_{\text{all models}} \text{MSPE}}.$$

Simulations - Prediction (Sim 1 and 2)



Simulations - Prediction (Sim 3 and 4)





Section IV

Conclusions

Conclusions

- Methodology developed to incorporate functional data into neural network framework
- FNNs perform on-par (if not better) than many multivariate and functional methods for longitudinal data (at least in our examples)
- They also seem to do a better job of recovering the underlying coefficient function
- Due to the nature of the parameters, there is potential to interpret them
- Over a dozen hyperparameters to play with
- General functions developed for use with any functional data set

Future Work

- Work is underway to extend this methodology for functional responses (predicting curves, densities, etc.)
- Gives way to develop an analogue to recurrent neural networks - time series prediction
- Can add penalizations to neurons in the first layer (compact domain, second derivative, etc.)
- Perhaps we know something about the way the functional coefficient should behave? Bayesian priors?
- Completing the R package

References

Robert Tibshirani, Trevor Hastie, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

Guido Fubini. Sugli integrali multipli. *Rend. Acc. Naz. Lincei*, 16:608–614, 1907.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

YK Kim and JB Ra. Weight value initialization for improving training speed in the backpropagation network. In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pages 2396–2401. IEEE, 1991.

References Cont. (1)

Mercedes Fernández-Redondo and Carlos Hernández-Espinosa. Weight initialization methods for multilayer feedforward. In *ESANN*, pages 119–124, 2001.

François Chollet et al. Keras. `https://keras.io`, 2015.

Hadi Fanaee-T and Joao Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2(2-3):113–127, 2014.

Hans Henrik Thodberg. Tecator meat sample dataset. statlib datasets archive, 2015.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, and Yuan Tang. Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4, 2015.

Jerome H Friedman and Werner Stuetzle. Projection pursuit regression. *Journal of the American statistical Association*, 76(376):817–823, 1981.

Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL `https://CRAN.R-project.org/doc/Rnews/`.

Jane-Ling Wang, Jeng-Min Chiou, and Hans-Georg Müller. Functional data analysis. *Annual Review of Statistics and Its Application*, 3:257–295, 2016.

References Cont. (2)

Manuel Febrero Bande and Manuel Oviedo de la Fuente. Statistical computing in functional data analysis: The r package fda. usc. 2012.

James O. Ramsay, Giles Hooker, and Spencer Graves. *Functional data analysis with R and MATLAB*. Springer New York, 2009.

Jim Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer New York, 2010.

Hervé Cardot, Frédéric Ferraty, and Pascal Sarda. Functional linear model. *Statistics & Probability Letters*, 45(1):11–22, 1999.

Hervé Cardot, Frédéric Ferraty, and Pascal Sarda. Spline estimators for the functional linear model. *Statistica Sinica*, pages 571–591, 2003.

Trevor Hastie and Colin Mallows. [a statistical view of some chemometrics regression tools]: Discussion. *Technometrics*, 35(2):140–143, 1993.

Hans-Georg Müller, Ulrich Stadtmüller, et al. Generalized functional linear models. *the Annals of Statistics*, 33(2):774–805, 2005.

Ci-Ren Jiang, Jane-Ling Wang, et al. Functional single index models for longitudinal data. *The Annals of Statistics*, 39(1):362–388, 2011.

Cristian Preda, Gilbert Saporta, and Caroline Lévéder. Pls classification of functional data. *Computational Statistics*, 22(2):223–235, 2007.

References Cont. (3)

Frédéric Ferraty and Philippe Vieu. *Nonparametric functional data analysis: theory and practice*. Springer Science & Business Media, 2006.

Germán Aneiros-Pérez and Philippe Vieu. Semi-functional partial linear regression. *Statistics & Probability Letters*, 76(11):1102–1110, 2006.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

George AF Seber and Alan J Lee. *Linear regression analysis*, volume 329. John Wiley & Sons, 2012.

Thanks for Listening!