2021

EXTENDING BOOTSTRAP AGGREGATION OF NEURAL NETWORKS FOR PREDICTION WITH AN APPLICATION TO COVID-19 FORECASTING

Mohsen Tabibian
University of Montana, Missoula

EXTENDING BOOTSTRAP AGGREGATION OF NEURAL NETWORKS FOR PREDICTION WITH AN APPLICATION TO COVID-19 FORECASTING

By

MOHSEN TABIBIAN

Master of Science, Data Science, The University of Montana, Missoula, MT, 2020
Master of Art, Mathematical Statistics, Tarbiat Modares University, Tehran, Iran, 2010
Bachelor of Science, Statistics, Islamic Azad University, Markazi, Iran, 2007

Dissertation

presented in partial fulfillment of the requirements
for the degree of

PhD
in Mathematics

The University of Montana
Missoula, MT

May 2021

Approved by:

Scott Whittenburg, Dean of The Graduate School
Graduate School

Dr. Brian Steele, Chair
Department of Mathematical Sciences

Dr. Jonathan Graham,
Department of Mathematical Sciences

Dr. Johnathan Bardsley,
Department of Mathematical Sciences

Dr. Javier Perez Alvaro
Department of Mathematical Sciences

Dr. Erin Landguth
Department of Health Sciences

Tabibian, Mohsen, Ph.D., May 2021 Mathematics

Abstract

Chairperson: Dr. Brian Steele

The aim of this study is to improve the forecasting accuracy of artificial neural networks (ANNs) and construct prediction bands for ANN models. The focus is on forecasting for epidemiological purposes, and in particular, the problem of predicting new case and death counts from seven to h days into the future for spatially contiguous regions. The task poses several challenges: datasets are quite small, and both spatially and temporally correlated. To overcome these, the methods attempt to exploit information induced by spatial and temporal dependencies. More importantly, we have developed a fusion of ANNs and bootstrap methods. Bootstrap aggregation (bagging) is an ensemble technique used for reducing the prediction variance and concurrently improving predictive accuracy and constructing prediction bands. Random forests extend bagging by sampling predictors in addition to observations with the result of often dramatic improvement in accuracy. The method developed herein resembles random forests to improve predictive accuracy and to construct prediction bands. We refer to this new approach as extended-bagging (E-Bagging).

Covid-19 is a highly contagious virus that has disrupted life around the world. Accurate predictions of disease trajectory in the near term are critical. Recurrent neural networks based on gated recurrent units (GRU) are a subclass of ANNs that exploits temporal data structures; however, they are problematic to use and remain poorly understood by researchers. Hence, we propose a simple alternative referred to as weighted neural networks and use this with E-Bagging. To investigate and compare these innovations with standard ANN approaches, we apply the methods to Covid-19 datasets using four counties as the spatial units. The predictive functions forecast the number of deaths for 14 days ahead using four of the most populous US counties. The performance of models is quantified by the mean absolute error. The E-Bagging of GRU models yields highly informative predictions and outperformed the other prediction models. The assessment of constructed prediction bands is measured by coverage probability and the GRU model with the E-Bagging technique performed best. These methods can be applied to a wide variety of other situations from Ebola outbreak mitigation to intra and inter-day stock price forecasting.

Acknowledgements

Throughout the completion of this dissertation, I have received a great deal of support and assistance. I would first like to thank my advisor, Dr. Brian Steele, whose expertise was invaluable in formulating the research topic and methodology in particular.

I would like to thank my wife, Parto, for her love and constant support and help. But most of all, thank you for being my best friend. I owe you everything.

This dissertation is dedicated to Covid-19 frontline workers and their families.

Table of Contents

1 Introduction 1

1.1 Historical Introduction 3

1.2 Neural Networks in Plain Terms 4

1.3 Grand Challenge, Pitfalls, Limitations, and Future Directions 7

2 Literature review 11

2.1 Bootstrapping 11

2.2 Bootstrapping Neural Networks 11

2.3 Using Neural Networks for Medical Problems 12

2.4 Covid-19 Models 13

2.4.1 Using Neural Network for Covid-19 Predictions 15

3 Methodology 17

3.1 Data 17

3.1.1 Pre-processing of Data 23

3.2 Artificial Neural Networks 24

3.2.1 Activation Functions 24

3.2.2 Architecture of a Feed-forward Network 29

3.3 General Architecture of Feed-forward Networks 32

3.3.1 Network learning 33

3.4 Gradient Descent Optimization 34

3.4.1 Batch gradient descent 35

3.4.2 Stochastic gradient descent 35

3.4.3 Mini-batch gradient descent 36

3.4.4 Adam Optimizer 36

3.5 Regularizing Deep Neural Networks 37

3.5.1 Dropout 38

3.5.2 Dropout Rate 38

3.5.3 Use of Wider Neural Networks 39

3.5.4 Early Stopping Approach 39

3.6 Assessment of fits 40

3.7 Recurrent Neural Networks 40

3.7.1 Gated Recurrent Unit 42

3.8 Innovations 46

3.8.1 Weighted Neural Networks 46
3.8.2 Bootstrap Aggregation of Neural Networks 48
3.8.3 Hyper-Parameter Tuning 53
4 Results 59
4.1 Fitting Models Based on the GRU 59
4.1.1 Data preprocessing 60
4.1.2 GRU Architecture 60
4.2 Fitting Models Based on the WNN 86
4.2.1 WNN Architecture 86
5 Discussion and Conclusions 102
6 Appendix A 105
6.1 Loading Required Modules 105
6.2 A Function to Create Raw and Smoothed Datasets 105
6.3 Loading Dataset for LA County 106
6.4 Codes for Creating Figure 4 106
6.5 Codes for Creating Figure 6 107
6.6 Plotting Smoothed and Forecasted Cumulative COVID-19 Deaths Using Different Models 107
6.7 Build Rolling Window Matrix (R) and Split and Scale Train and Test Sets 110
6.8 A Function for Creating a Model for Tuning Hyperparameters 110
6.9 Create the Best LA GRU Model 113
6.10 Define a list of All Unique Combinations of Counties 114
6.11 Build Rolling Window Matrix (R) and Split Train and Test Sets, and scale them for E-Bagging 115
6.12 Create Residual Predictor Network 115
6.13 Create E-Bagging Models 116
6.14 Create Error Distribution 118
7 References 120

List of Tables

Table 1: MAE of different models for each county on the training and test sets 71
Table 2: MAE of different models for each county on the training and test sets 95
Table 3: Results of MAE and coverage probability of different models 102

List of Figures

Figure 1: "Network" graph of the multilinear regression (model $f1$) 5
Figure 2: "Network" graphical representation of the $f2$ model 6
Figure 3: "Network" graph of the $f3$ model. Here we have a neural network with 2 features, 1 hidden layer with 3 nodes and 1 output. 7
Figure 4: Real Cumulative Covid-19 confirmed cases in neighboring counties of LA County 19
Figure 5: Daily Covid-19 confirmed deaths in LA County as of 05/19/2021 20
Figure 6: Unsmoothed and smoothed daily Covid-19 deaths using 2 types of smoothing in LA County as of 05/19/2021 21
Figure 7: Unsmoothed and smoothed daily Covid-19 deaths in Cook County as of 05/19/202 22
Figure 8: Unsmoothed and smoothed daily Covid-19 deaths in Harris County as of 05/19/2021 22
Figure 9: Unsmoothed and smoothed daily Covid-19 deaths in NY County as of 05/19/2021 23
Figure 10: Left: Sigmoid activation function; Right: Tanh activation function 26
Figure 11: ReLU activation function 28
Figure 12: ELU activation function 28
Figure 13: SELU activation function 28
Figure 14: A diagram of a neural network 29
Figure 15: Flow and operations in a GRU cell, Figure source [43] 43
Figure 16: Partitioning a sample to $n - m + 1$ rolling windows. 44
Figure 18: Diagnostic results with different hyperparameter combinations when batch size is 100 and learning rate is 0.001 in the LA model; in total, 320 models were fit to produce this boxplot. 63
Figure 19: Diagnostic results with different hyperparameter combinations when batch size is 100 and learning rate is 0.005 in the LA model; in total, 320 models were fit to produce this boxplot. 64
Figure 20: Diagnostic results with different hyperparameter combinations when batch size is 300 and learning rate is 0.001 in the LA model; in total, 320 models were fit to produce this boxplot. 65
Figure 21: Diagnostic results with different hyperparameter combinations when batch size is 300 and learning rate is 0.005 in the LA model; in total, 320 models were fit to produce this boxplot. 66
Figure 21: the smoothed daily Covid-19 confirmed deaths (blue curve) and forecasted daily deaths using GRU (red curve) for different test sets as of 05/03/2021 for LA County. 68
Figure 22: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for LA County. (Bottom) cumulative counts. The box zooms in on the test set. 70
Figure 23: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Cook County. (Bottom) cumulative counts. The box zooms in on the test set. 72
Figure 24: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Harris County. (Bottom) cumulative counts. The box zooms in on the test set. 73
Figure 25: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for NY County. (Bottom) cumulative counts. The box zooms in on the test set. 74

Figure 26: (Left) Coverage probability distribution, (Right) MAE distribution for different E-Bagging samples for LA County; in total, 9970 models were fit to produce each boxplot. 76
Figure 27: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 *for LA County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 78
Figure 28: Prediction error distributions for the best GRU model (blue histogram) and for the E-Bagging (red histogram) for LA County; both are centered at zero. The latter has less variance. 79
Figure 29: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 *for Cook County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 81
Figure 30: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 *for Harris County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 82
Figure 31: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 *for NY County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 83
Figure 32: Prediction error distributions for the GRU model (blue histogram) and for E-Bagging (red histogram) for Cook County (top); for Harris County (bottom); both are centered at zero. The E-Bagging errors have less variance. 84
Figure 33: Prediction error distributions for the GRU model (blue histogram) and for E-Bagging (red histogram) for NY County; both are centered at zero. The latter has less variance. 85
Figure 34: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is Tanh, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot. 89
Figure 35: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is SELU, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot. 90
Figure 36: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is ELU, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot. 91
Figure 37: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is ReLU, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot. 92
Figure 38: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the

test set from May 02, 2021, to May 16, 2021 *for LA County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 96
Figure 39: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 *for Cook County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 97
Figure 40: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 *for Harris County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 98
Figure 41: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 *for NY County. (Bottom) cumulative counts. The boxes zoom in on the test set.* 99
Figure 42: Prediction error distributions for the WNN model (blue histogram) and for E-Bagging (red histogram) for LA County (top); for Cook County (bottom); both are centered at zero. The E-Bagging errors have less variance. 100
Figure 43: Prediction error distributions for the WNN model (blue histogram) and for E-Bagging (red histogram) for Harris County (top); for NY County (bottom); both are centered at zero. The E-Bagging errors less variance. 101

1 Introduction

Recently discovered coronavirus disease 2019 (Covid-19) has affected many people of all ages and shut down the world economy. Most Covid-19 patients do not need any treatment to recover; however, some older people and those individuals with certain health problems such as heart disease, lung disease, cancer, diabetes, and asthma are at higher risk and more likely to become seriously ill. Covid-19 cases started on Dec 31, 2019, in China [1] and continued to spread across the globe, increasing the number of deaths due to Covid-19 on a daily basis. On January 20, 2020, the first case of Covid-19 in the United States was confirmed by a laboratory and reported to the Centers of Disease Controls (CDC) on January 22, 2020, [2]. As of the writing of this work, May 20, 2021, there are 33.1 million confirmed Covid-19 cases in the US with 588,654 deaths attributed to Covid-19.

The Covid-19 virus is likely to spread from one county to adjacent counties. Thus, the spatial nature of the contagion is an important attribute of Covid-19 spread and the death counts attributed to Covid-19 in adjacent counties are likely spatially correlated. Los Angeles County in California, Cook County in Illinois, Harris County in Texas, and New York County in New York are populous US counties that experienced many Covid-19 death counts as of May 19, 2021, [3]. For example, in LA County there were 1.24 million confirmed Covid-19 cases and 24,143 deaths due to the disease as of May 19, 2021. Covid-19 can attain exponential growth in its spread in densely populated counties very easily. The number of deaths due to this invisible disease increases daily. Hence, we decided to forecast the number of deaths

due to Covid-19 using deaths data from spatially adjacent counties for two weeks beyond May 02, 2021. For all forecasts, we assume that existing social distancing measures did not change through the projected 2-week time period.

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks (RNNs) that belong to a class of artificial neural networks (ANNs) where connections between nodes form a directed cycle. This creates an internal state (memory) of the network which allows them to exhibit dynamic temporal behavior. Hence, GRUs would be able to process temporal data like daily Covid-19 death counts. Also, to make ANNs more powerful to be able to handle temporal datasets, we modified them by assigning more weight to more recent observations to preserve and exploit the temporal structure; we refer to these as weighted neural networks (WNNs). Moreover, for improving predicted results and constructing prediction bands, we extended the bootstrap aggregation or bagging mechanism to introduce a greater degree of differences from bootstrap-to-bootstrap NNs and refer to this new approach as **extended bagging** (E-Bagging).

We use GRUs, WNNs, and their E-Bagging versions to train our models. Predictions using these models are useful because they help us understand the most likely results and can help researchers and policymakers make decisions that can lead to best outcomes. To measure the performance of neural network models, the mean absolute error (MAE) was computed. For assessing and comparing constructed prediction bands using E-Bagging techniques, coverage probability (CP) was computed.

The proposed models can be applied to other counties as well as other epidemic diseases. Additionally, they can be applied to datasets at the state or the country level. To our knowledge, the development of bagging techniques to improve multi-step predictions and to construct prediction bands for recurrent neural network-based

models is novel to this work. Additionally, this is the first development of conventional neural networks to handle temporal data.

1.1 Historical Introduction

The origin of the term 'neural network' comes from finding mathematical representations of information processing in biological systems. McCulloch and Walter [4], a neurophysiologist and a mathematician, respectively, wrote a paper on how neurons might work and modeled a simple neural network based on mathematics and logical algorithms. The 1956 Dartmouth Summer Research Project on Artificial Intelligence provided a boost to both artificial intelligence and neural networks research. In 1956, Rochester et al. [5] created some neural network computational machines. In 1958, Rosenblatt [6], a neurobiologist, began work on the perceptron which was built as hardware and is the oldest neural network still in use today.

Between 1960 and 1980 was a time of regression for artificial neural networks (ANNs). Minsky and Papert [7] proved that a single layer perceptron is limited. Until 1981, progress on neural networks research discontinued for a decade. As a result, much of the funding was cut. In 1982, Japan announced their Fifth-Generation effort at the US-Japan Joint Conference on Neural Networks. This led to the US worrying about being left behind and prompted renewed funding efforts. Rumelhart et al. [8] used the backpropagation algorithm, a robust and well-known network learning procedure, for multilayer neural networks. They showed that such networks can learn useful internal representations of data.

In 1997, Schmidhuber and Hochreiter [9] introduced a RNN framework called Long Short-Term Memory. In 1998, LeCun et al. [10] studied the convolutional neural

network with backpropagation for document analysis. In 2009, Bengio [11] introduced deep neural networks (DNNs) which are ANNs with multiple hidden layers between the input and output layers. In 2012, Krizhevsky, Sutskever, and Hinton [12] wrote a paper on image classification with deep convolutional neural networks using Graphical Processing Units (GPUs); this was a great success for neural networks because of the massive use of GPUs. The availability of massively large data sets, advanced GPUs, and the development of techniques for training DNNs have resulted in an acceleration of research on ANNs again.

1.2 Neural Networks in Plain Terms

The objective of this section is to give some insights and an overview of neural networks. Machine learning (ML) algorithms are computer algorithms that improve or learn automatically using data. An ML algorithm builds a model based on training data, to make predictions without being explicitly programmed to do so. Hence, in ML algorithms, we provide models to a computer that define possible rules, training datasets, and plans to find better rules, while in a standard algorithm, rules are provided to a computer to do an explicit task. The simple linear regression model is one of the most basic machine learning models.

For example, consider a dataset containing house prices (p) and their corresponding sizes (s). Consider a simple linear model that takes the size of a house and returns a predicted price of such a house as given, $\hat{p} = f(s)$, where function f expresses the relationship between predicted house prices as output and house sizes as input. We have: $\hat{p} = f(s) = as + b$, where a and b are estimates of unknown coefficients. In theory, for the linear regression model, the least squares approach consists of choosing a and b such that the sum of squared errors between true outputs, p's, and predicted values, $\hat{p}$'s, is minimized. In a linear regression problem, there are three

parts: a regression model, a dataset, and an optimization algorithm. If we have no idea what function f looks like, we can rely on data to build our rules in a machine learning environment. Neural network architectures are a complex generalization of the linear regression model. They are capable of representing complex relationships between predictor and response variables.

Now, consider the situation of two inputs, denoted by (i_1, i_2), and suppose we want to find a function f of these inputs that explains the observed corresponding outputs, denoted by (o). Suppose further that there is no prior knowledge about the relationship between these inputs and the output values. Therefore, we want to find the function f such that $f(i_1, i_2)$ is a good model for o. We could then suggest the following initial model:

$$E(o) = f_1(i_1, i_2) = w_{11}\, i_1 + w_{12}\, i_2,$$

where w_{11} and w_{12} are scalars. For simplicity, there is no constant term in the model. Such a model is a multilinear regression model and is represented by the schematic in Figure 1.

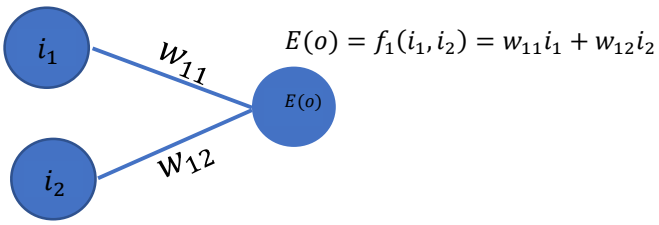


Figure 1: "Network" graph of the multilinear regression (model f_1)

In this case, the model is simple and easy to fit but there is no non-linearity. In order to introduce a non-linearity, a modified model is as follows:

$$f_2(i_1, i_2) = \zeta(w_{11}\, i_1 + w_{12}\, i_2),$$

where $\zeta(.)$ is a non-linear *activation* function. Notice that the inner expression $w_{11}\, i_1 + w_{12}\, i_2$ is still a linear combination, but after applying a non-linear function to the linear combination, the transformation is nonlinear and, hence, is possibly closer to our non-linearity assumption. This model is represented by the schematic in Figure 2.

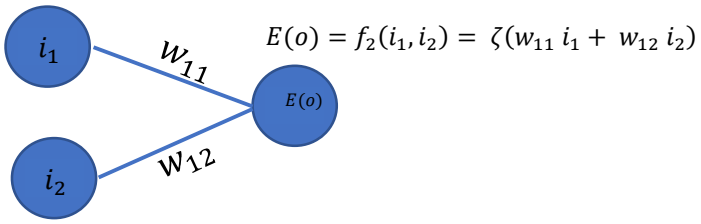


Figure 2: "Network" graphical representation of the f_2 model

However, there is still limited complexity in this model. To enrich the model, we can consider a new intermediate layer of new features of our model called l, which implies that the expression $\zeta(w_{11}\, i_1 + w_{12}\, i_2)$ is no longer the final output. Then we could build three such features in the same way: $l_1 = \zeta_{11}(w_{11}\, i_1 + w_{12}\, i_2)$, $l_2 = \zeta_{12}(w_{21}\, i_1 + w_{22}\, i_2)$, and $l_3 = \zeta_{13}(w_{31}\, i_1 + w_{32}\, i_2)$, where the ζ_{ij}'s are activation functions and the w_{ij}'s are scalars. Note that the scalars and nonlinear activation functions could be different for these three features. Lastly, the final output, o, is built based on these new intermediate features with the same template: $\zeta_2(\, v_1 l_1 + v_2 l_2 + v_3 l_3)$ where v_i's are weights for each new intermediate feature. Therefore, the final composite model is represented in Figure 3 as follows:

$$\begin{aligned} f_3(i_1, i_2) &= \zeta_2(\, v_1 l_1 + v_2 l_2 + v_3 l_3) \\ &= \zeta_2\{v_1 \zeta_{11}(w_{11}\, i_1 + w_{12}\, i_2) \\ &\qquad + v_2 \zeta_{12}(w_{21}\, i_1 + w_{22}\, i_2) \\ &\qquad + v_3 \zeta_{13}(w_{31}\, i_1 + w_{32}\, i_2)\}. \end{aligned}$$

This final model is a basic feedforward neural network with two input features (i_1 and i_2), one hidden layer with three nodes (l_1, l_2 and l_3) and one final output (o). To add complexity, we could add another intermediate hidden layer between the input and output layers and the final output in the same way as we added the l_i's in the first hidden layer. The resulting model would be a neural network with two hidden layers. Alternatively, we could continue with just one hidden layer but add two more nodes. Thus, there are many different ways to develop models.

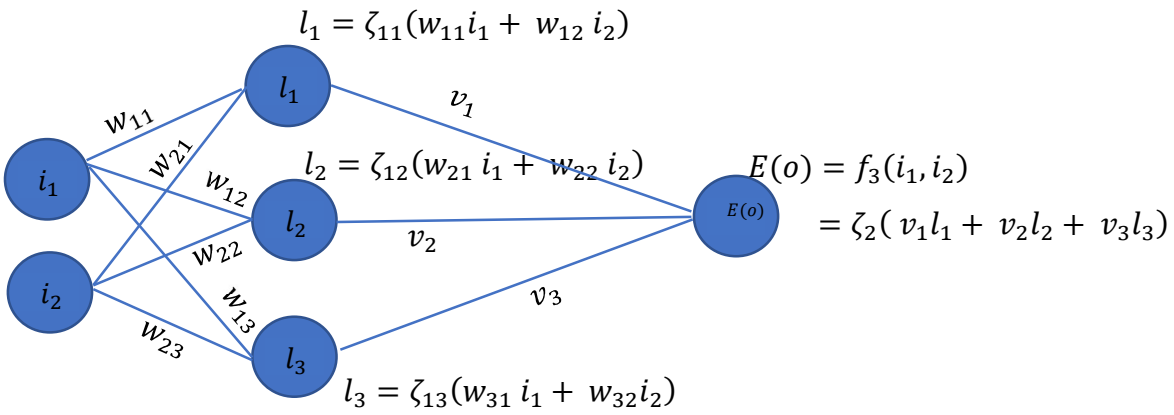


Figure 3: "Network" graph of the f_3 model. Here we have a neural network with 2 features, 1 hidden layer with 3 nodes and 1 output.

1.3 Grand Challenge, Pitfalls, Limitations, and Future Directions

We can find an optimal model for many simple ML algorithms. For example, we can find coefficients of a linear regression model by minimizing the sum of squared errors. However, for a nonlinear model such as a logistic regression model, using an optimization algorithm usually requires solving a convex optimization problem; An optimization algorithm guarantees convergence when finding an optimal set of model parameters. Although, the story is different in DNNs or even when optimizing a convex error surface of a model with an ill-conditioned Hessian matrix consisting of second-order partial derivatives.

A big challenge in DNNs with many layers and hundreds of nodes in each layer is not only the estimation of model parameters with current hardware technology, but also guaranteeing global convergence of the estimates. This results from using nonlinear activation functions in neural networks coupled with an optimization problem on a non-convex error surface for the model. Solving such optimization problems is challenging, because the non-convex error surface of the model contains many local minima, saddle points, and cliffs. To work with a non-convex error surface, we must use an iterative process. In fact, the most challenging and time-consuming part of using a neural network is training the model and computing model parameters.

Normally, training a neural network can be challenging and choosing a good architecture is difficult and sometimes impossible; particularly when choosing the initial hyperparameters, which are the parameters whose values are defined prior to the beginning of the learning process. Small changes in these hyperparameters' values can lead to large changes in the performance of models.

The grand challenge of neural networks is their nature. They are often considered black boxes. We feed known input data into neural networks and know our model's fitted parameters, and how they are assembled. But we usually do not understand relationships between predictors and output. For example, when we feed a neural network model an image of a dog and it predicts a cat, it is very hard to understand what caused this bizarre prediction.

While neural networks are not able to give the same level of interpretability and insight as many statistical models can, it is a mistake to view them as complete black boxes with the assumption that we know nothing about neural network structures.

The nonlinear and nonparametric nature of NN models has caused many pitfalls. While this nature is desirable for many real-world applications, it opens more possibilities to go wrong in the modeling process. Compared to linear statistical models, they have fewer assumptions, more parameters to estimate, and many more options to choose from in the modeling process, all of which increase the chance of failing to build a near-optimal productive model.

Another conceptual error is thinking that the neural network can automatically exploit the most important features during the model process. However, including many unnecessary input variables in the model not only increases the model complexity and the likelihood of overfitting, but also wastes time and effort in training. Such pitfalls have been detected in building and selecting neural networks. In published articles, many applications of neural networks do not reveal the detail of many aspects of the modeling process including the data, data processing, experimental design, model selection, tuning hyperparameters, and other choices made during the process. This behavior can hide errors and misuses of the techniques employed in studies.

The greater the network complexity, the more hyperparameters that need to be tuned. More hyperparameters require greater amounts of data to be optimally determined. To make sure that the deep learning algorithms deliver desired results we often need large datasets. Finding these datasets can be a limitation. For training a neural network with a large dataset, systems need to have adequate processing power such as high-performing GPUs which can be another limitation. Moreover, employing several GPUs together can be another limitation because they consume a lot of power and are costly.

One future direction of neural networks is combining neural networks with complementary technology, like symbolic functions, to perhaps compensate for any weaknesses among them. Another future direction of neural networks is the construction of training tools to help trainees gain competence and confidence in difficult medical diagnoses. With technological advancements, we can make CPUs and GPUs cheaper and/or faster, enabling the production of bigger, more efficient algorithms. We can also design neural networks capable of processing more data, or processing data faster, so they learn to recognize patterns with just 10,000 examples, instead of 100,000.

The future directions of neural networks do not live solely in attempts to simulate reality. All current neural network techniques will most likely be vastly improved upon in the future as researchers develop better training techniques and network designs. Neural networks could expand horizontally as they are applied to more diverse applications rather than only advancing vertically in terms of faster processing power and more complexity. Many big industries and startups in different areas such as eCommerce, security, logistics, healthcare, and finance could feasibly use neural networks to operate more efficiently or develop new products.

This dissertation is structured as follows. In chapter 2, we have undertaken a literature review of several advancements in the use of neural networks; in chapter 3, we describe the Covid-19 data, the preprocessing involved and corresponding methodology. Results and a comparative study are presented in chapter 4.

2 Literature review

2.1 Bootstrapping

There are two common methods of bootstrapping in regression models. One approach is to take samples with replacement from the original data randomly to compute separate estimates for every single bootstrap; this is called the bootstrap pairs method. A second common approach is to take samples from model residuals; this is called the bootstrap residual method.

Efron et al. [13] introduced the idea of bootstrapping for estimating the variance of a statistic. The variation in these independent estimates is a good measure of the certainty of the estimation. Efron and Tibshirani [14] wrote a review of applications of bootstrapping and stated that the bootstrap method has very nice properties for estimating standard errors and constructing confidence intervals for model parameters, as well as evaluating the quality of predictions.

2.2 Bootstrapping Neural Networks

Paass [15] and Carney [16] applied bootstrap methods to neural networks to estimate the predictive distribution for unobserved inputs and to calculate prediction intervals. They showed that bootstrap methods offer practical advantages for model performance with respect to a new input.

Tibshirani [17] compared the two common bootstrap methods (bootstrap pairs and bootstrap residual) to two other techniques, the Delta method based on the Hessian and the sandwich estimator, for estimating the standard error of neural network predictions. He found that the bootstrap methods perform best partially because they account for variability due to the choice of initial weights.

Heskes [18] applied bootstrap methods to compute prediction intervals for neural networks by taking bootstrap samples from individuals. He reported that for a small dataset, the width of a prediction interval depends on both the variance of the target distribution and the accuracy of the estimator of the mean of the target. The advantage of applying bootstrap approaches to neural nets is that averaging over network predictions improves the network performance.

Khosravi et al. [19] compared four techniques including bootstrap methods for constructing prediction intervals for neural network forecasts. They found that bootstrap methods are best in terms of low computational load and the variability of prediction intervals. Other techniques, such as the Delta and Bayesian methods required computing the Hessian matrix which is computationally expensive for a large dataset.

2.3 Using Neural Networks for Medical Problems

Neural network-based models have been developed for a variety of medical problems. They can improve learning algorithms by training some neural network models over time and combining their results to reduce the error rate of models. For example, Mantzaris et al. [20] used a multilayer perceptron to distinguish types of dementia. Zhang et al. [21] used a convolutional neural network-based model to solve a multi-label classification system of chronic diseases to improve the

classification performance. Their model outperformed other models with a prediction accuracy of 81.13%. Ren et al. [22] used a hybrid neural network model based on a bi-directional long-short term memory (Bi-LSTM) network and an Autoencoder network to predict kidney problems in hypertensive patients. Their proposed model outperformed the strong neural baseline systems.

2.4 Covid-19 Models

Wang et al. [23] used a new Patient Information-based algorithm and estimated the number of daily deaths due to COVID-19 in China. They estimated the number of days from hospital admission to death is 13 days with a standard deviation of 6 days. They predicted a case fatality of 1.6% for Covid-19 patients and reported that the death rate due to Covid-19 ranged from 0.75% to 3% with 95% confidence at the beginning of pandemic. Moreover, they reported that the mortality rate would vary with respect to temperature and climate. Gupta et al. [24] pointed out that there is a direct association between temperature and Covid-19 new cases based on an analysis of US spread. They predicted that there would be a strong reduction in the number of new cases in India during the summer months, though this did not actually occur.

Ceylan [25] developed Autoregressive Integrated Moving Average (ARIMA) models with different parameters for predicting the total confirmed cases of Covid-19 in Spain, Italy, and France. They found that the best models were an ARIMA (1,2,0), ARIMA (0,2,1), and ARIMA (0,2,1) for Spain, Italy, and France, respectively, in terms of mean absolute percentage error (MAPE), with MAPE values of 5.849, 4.752, and 5.634, respectively. Ahmar and del Val [26] used ARIMA and Sutte ARIMA models for forecasting the number of new daily cases of Covid-19 in Spain 3 days into the future. The Sutte ARIMA model outperformed the standard ARIMA model with MAPE values of 0.036 and 0.066, respectively.

Fanelli and Piazza [27] used simple mean-field models to analyze Covid-19 data from January 22 to March 15, 2020, in Italy. Their model predicted the peak daily number of Covid-19 cases to be near 26,000. On March 21, 2020, the number of deaths at the end of the epidemic in Italy was about 18,000. Additionally, they predicted the mortality rate of Covid-19 in Italy to be between 4% and 8%. They also estimated the number of ventilators needed in Italy to be 2500 units for the peak.

Chande et al. [28] created an interactive online website for estimating the risk associated with the presence of at least one Covid-19 case in gatherings of different sizes in the US. This console used county level Covid-19 confirmed cases to visualize the information. Zhou et al. [29] proposed a spatio-temporal epidemiological model to forecast county level Covid-19 spread in the US t days ahead in time. Their model estimated the outbreak risk caused by intercounty traveling.

Mehta et al. [30] developed a three-stage model to forecast Covid-19 cases for short-term predictions and risks at the county level in the US. Their model employed the extreme gradient boosting (XGBoost) method to measure the likelihood of being infected with COVID-19 and to estimate the number of possible cases for unaffected counties. They found that population, population density, and percentage of adults over the age of 70 play an important role in Covid-19 predictions. They reported model sensitivity greater than 71% and specificity greater than 94%.

Ives and Bozzuto [31] estimated the Covid-19 outbreak rate among 160 counties in the US using numbers of deaths by May 23, 2020. They found that four factors explained the variability in the outbreak rate: timing of outbreak, population density and size, and spatial location. They showed that different variations of SARS-CoV2 dominant in different parts of the US may be responsible for some of the effect of spatial location.

2.4.1 Using Neural Network for Covid-19 Predictions

Wieczorek et al. [32] developed a 7 hidden layers neural network model for predicting Covid-19 spread. They used a dataset provided by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University. They tested a few different time-steps and found that the past 14-day time-step is the best option to predict unseen values. Also, they pointed out that using an RNN for predicting total cases performed about 1–2% worse than the ANN-based model, and training with the same parameters took about 3 times as long.

Saba et al. [33] used an ARIMA model and a model based on nonlinear autoregressive ANNs (NARANNs) to forecast cumulative confirmed Covid-19 cases in Egypt. They trained models on daily COVID-19 cases for the period between March 1, 2020, and May 10, 2020. Their proposed NARANN model had an absolute percentage error of less than 5% and outperformed the ARIMA model.

Reddy and Zhang [34] predicted the end date of the Covid-19 pandemic to be around June 2020 in Canada using an LSTM model based on daily Covid-19 cases prior to March 31, 2020. Their network accuracy was 94% for a short-term period of time (2 days) and 93% for a long-term period (14 days). Note that how difficult it is to predict Covid-19 development into future.

Arora et al. [35] predicted the number of Covid-19 confirmed cases for the next day and one week ahead for all states and territories of India using an RNN based on long-short term memory (LSTM) variants such as stacked LSTM, Bi-directional LSTM, and Convolutional LSTM models for data between March 14, 2020, and May 14, 2020. Their proposed stacked LSTM model outperformed other models with error less than 3% for daily predictions and less than 8% for weekly predictions.

Shastri et al. [36] reported on Covid-19 forecasting for India and the USA by applying variants of LSTM such as Stacked LSTM, Bi-directional LSTM, and Convolution LSTM models. They forecasted the daily number of confirmed and death cases of Covid-19 for one month ahead. They found that the Convolution LSTM outperformed the other two models and predicted daily Covid-19 deaths with the lowest mean absolute percentage error values of 3.33% and 2.50% for India and the USA, respectively.

Shahid et al. [37] proposed ARIMA, support vector regression with polynomial kernel, LSTM, and Bi-directional LSTM forecast models for predicting the daily number of confirmed, death and recovered cases in ten major countries affected by Covid-19. They found that the Bi-LSTM model outperformed the other models and predicted daily Covid-19 confirmed cases, deaths, and recoveries with the lowest values of MAE and $RMSE$.

3 Methodology

3.1 Data

In this study, we used the Covid-19 deaths dataset from the USAFACTS website [38]. This dataset contains counts of Covid-19 documented deaths for the USA at the county level. According to the Centers for Disease Control (CDC), it is not possible to report the exact number of deaths due to delays in reporting, and because counties designate Covid-19 related deaths in different ways. The data of interest are the counts of deaths per day from 1/24/2020 to 05/19/2021 (481 days). We want to forecast the number of Covid-19 deaths for a reasonable time period, say $h = 14$ days, in the future $(y_{n+1}, \dots y_{n+14})$, where n is the number of days in the training data set. Our overall goal of choosing $h = 14$ days is to provide a practical model with good performance.

It is assumed that people commute between neighbor counties more often than traveling to more distant counties or states. Thus, the Covid-19 virus is more likely to spread from one county to adjacent counties. In other words, spatial contagion is an important aspect of the Covid-19 spread and the numbers of Covid-19 deaths in adjacent counties are spatially correlated. If the daily count of Covid-19 deaths increases in one county, the daily number of Covid-19 deaths in neighboring counties will tend to increase as well. In other words, there is information in Covid-19 cases/deaths in neighboring counties and we want to use this information for predictive purposes. For this reason, we predicted the number of deaths in the 4 US counties considered (Los Angeles County in California, Cook County in Illinois,

Harris County in Texas, and New York County in New York) using the numbers of Covid-19 deaths in neighboring counties.

According to the 2018 official estimate by the US Census Bureau, the most populous US counties are LA County, Cook County, and Harris County with about 9.82, 5.2, and 4.1 million people, respectively [39]; NY County's population was 1.59 million people with a rank of 21 in the list of most populous US counties. Covid-19 can attain exponential growth in its spread in these densely populated counties very easily. LA County reached 24,143 confirmed Covid-19 deaths in total as of 05/19/2021. This total for Cook County was 10,745, for Harris County 6,372, and for NY County 4,454.

The neighboring counties of LA County are Orange County, San Diego, Riverside, San Bernardino, Kern, Ventura, Santa Barbara, Imperial, and San Luis Obispo. The neighboring counties of Cook County in IL are Lake, McHenry, DuPage, Kane, Will, Kankakee, Kendall, and Lake County, Indiana which borders Illinois. The neighboring counties of Harris County are Montgomery, Liberty, Chambers, Galveston, Brazoria, Fort Bend, Walker, and Austin. The surrounding counties of NY County are Richmond, Queens, Kings, Bronx, Westchester, Rockland, Nassau, Bergen, and Hudson, the last two of which are in New Jersey. Visualization of confirmed deaths in the neighboring counties to LA county shows an upward trend over time for all neighboring counties (Figure 4). LA County (the top green line) had the largest number of deaths among all nearby counties in CA as of May 19, 2021.

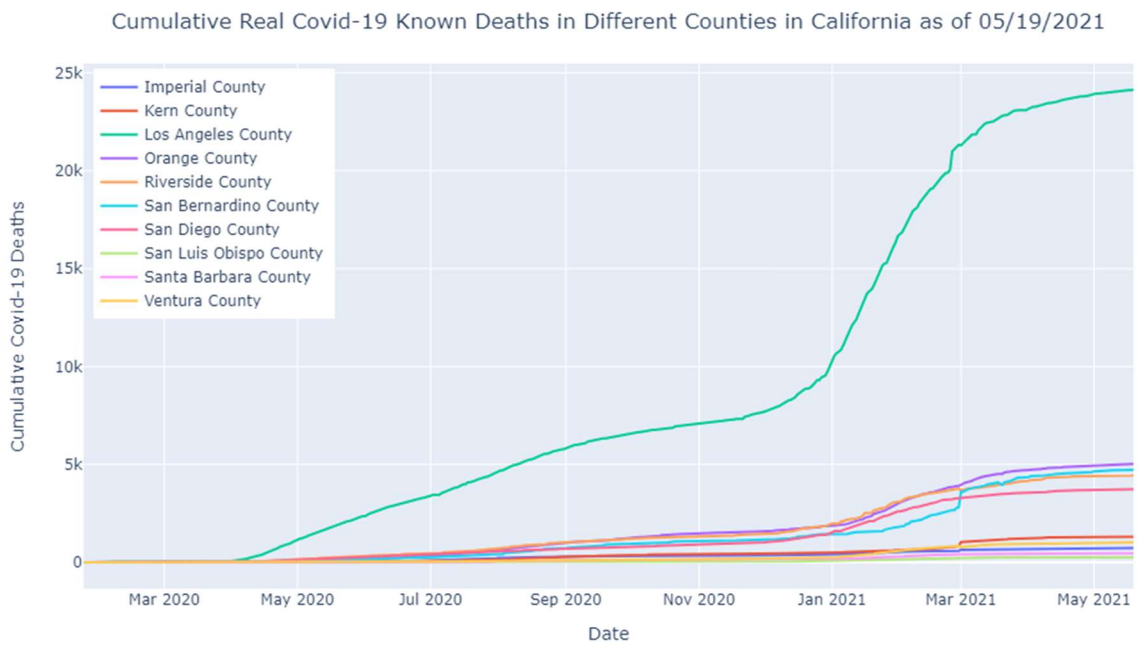


Figure 4: Real Cumulative Covid-19 confirmed cases in neighboring counties of LA County

Death counts due to Covid-19 at the county level and state level are variable from day to day, and it is difficult to quantify trends on a daily basis. For example, Figure 5 shows the daily data reported from Los Angeles County as of May 19, 2021. The blue bars represent daily confirmed deaths. This graph shows the granular nature of deaths data because many counties do not consistently report daily data and many display weekly cycles of under-reporting on weekend days and over-reporting on days following a weekend. As a result, some counties made corrections to their data throughout the pandemic. Los Angeles' data (Figure 5) includes several corrections they have made to the deaths data in different months (the tall bars). For example, 930 people are reported to have died on one day on February 25, 2021; but this value is inflated; Los Angeles County just reported a large correction on that day. These over-reporting spikes significantly undermine the development of data analytic models for those periods that include corrections. To overcome this, we used a moving average smoothing technique to reduce these reporting abnormalities.

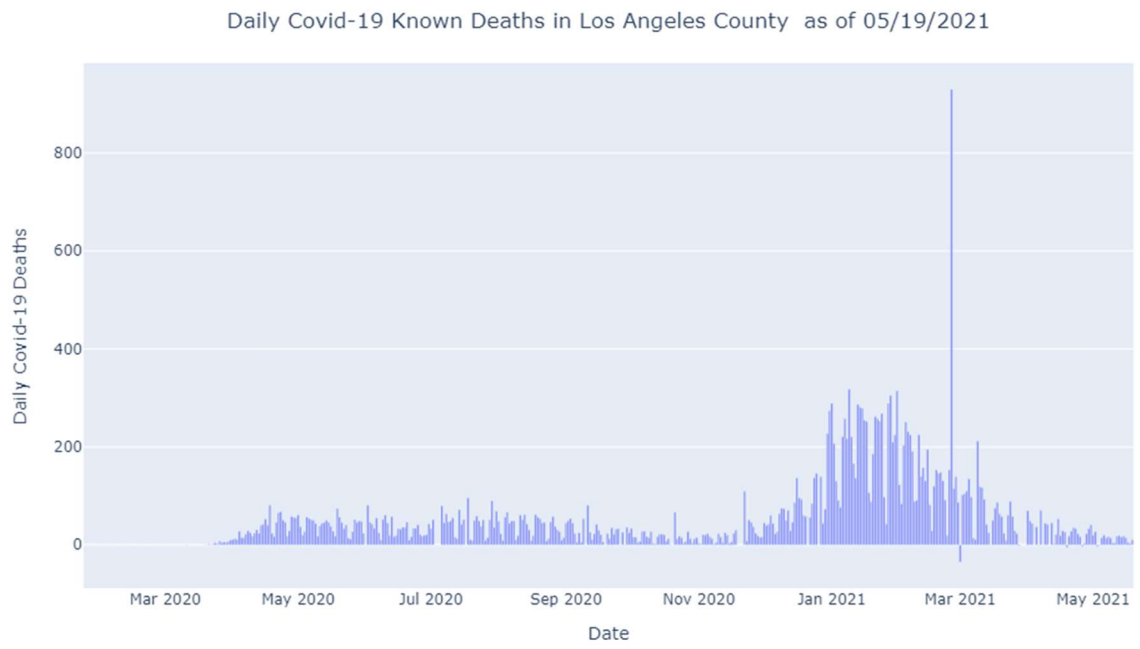


Figure 5: Daily Covid-19 confirmed deaths in LA County as of 05/19/2021

A full 7-day moving average curve was computed to obtain an accurate representation of the data for each week. Figure 6 illustrates that the seven-day moving average presents a clearer and more accurate picture of the underlying death process trend than the unsmoothed daily numbers of deaths in LA County. Note that the spike on February 25, 2021is much less unusual in the smoothed data. This figure also illustrates which 7 days should be selected in the moving average to present the clearer picture of the trend. In viewing this figure, the green curve represents a moving average that includes the report day and the 6 days following. The red curve shows a moving average that includes 3 days before the date of the report, the date of the report, and 3 days after the date of the report. Since the green curve is somewhat offset from the original trend in the unsmoothed death counts compared to the red curve, we used the second of these moving averaging definitions. It is worth noting that any other smoothing periods longer than 7 days (except for multiples of 7) or shorter than 7 days increases the risk of improperly weighting the days on which results are overreported or underreported.

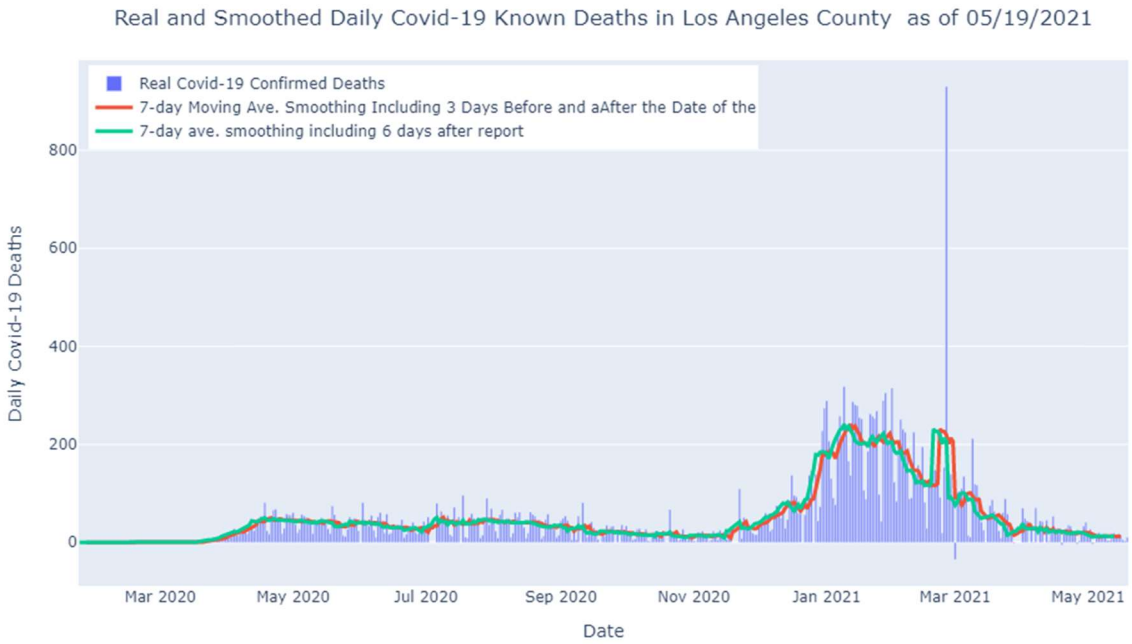


Figure 6: Unsmoothed and smoothed daily Covid-19 deaths using 2 types of smoothing in LA County as of 05/19/2021

Figure 7 - Figure 9 include time series representing the seven-day moving averages and unsmoothed daily numbers of Covid-19 deaths in the other three counties. In viewing these graphs, the red curve is the moving average that includes 3 days before the date of the report, the date of the report, and 3 days after the date of the report. The blue bars represent the unsmoothed daily Covid-19 deaths. These figures illustrate that seven-day moving averages present a clearer picture of the trend. It is worth noting that a 14- or 21-day moving average curve would smooth the large spikes even more than a 7-day moving average, but the latter is a natural time scale in the epidemiological context.

This smoothing has a minimal effect on our overall sample size, as we lose the first and last 3 observations from the dataset, thus reducing our data to the period from 1/27/2020 to 05/16/2021 (475 days instead of 481). We considered the data from May 02, 2021, to May 16, 2021, as the test set. Our objective is to predict the number of Covid-19 deaths for the test set for each of the four counties considered.

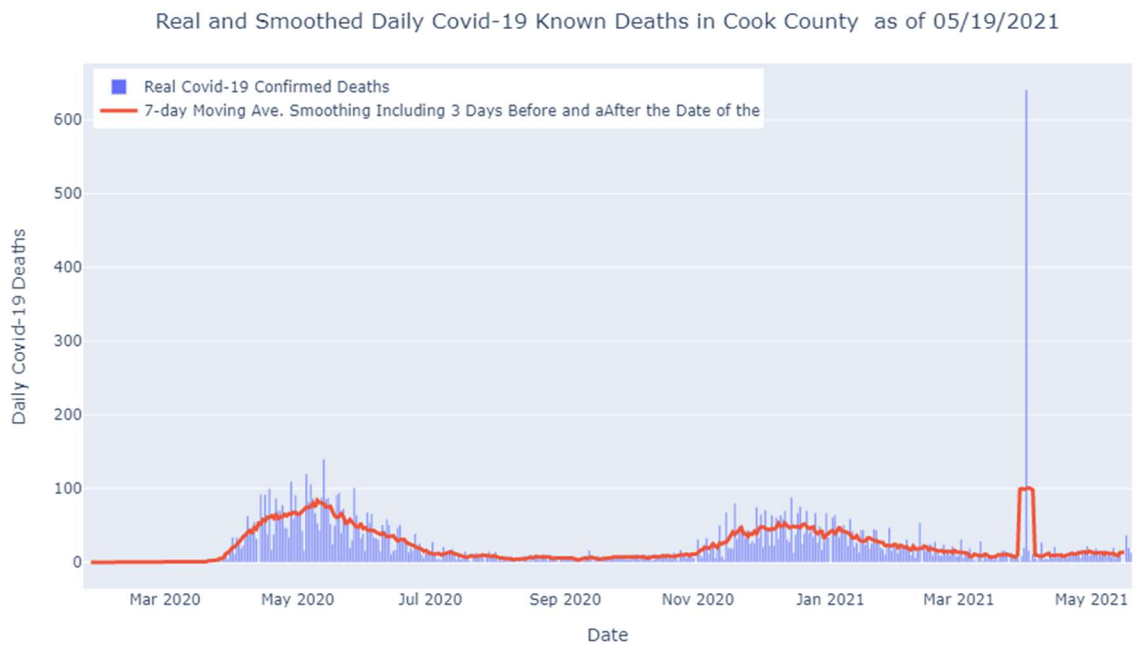


Figure 7: Unsmoothed and smoothed daily Covid-19 deaths in Cook County as of 05/19/202

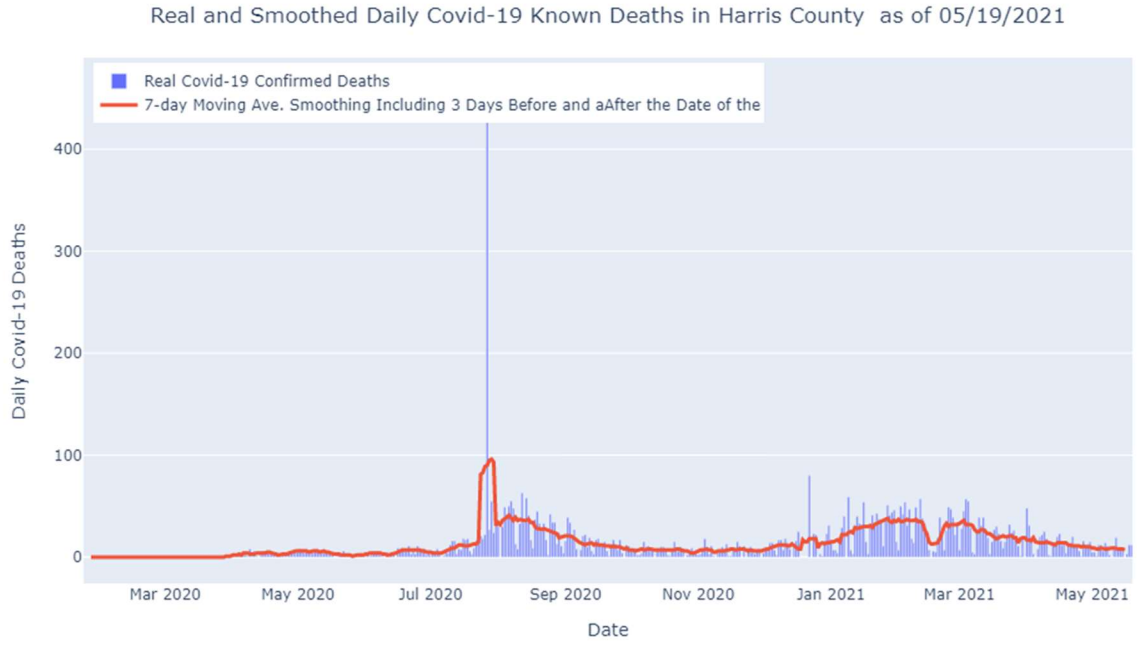


Figure 8: Unsmoothed and smoothed daily Covid-19 deaths in Harris County as of 05/19/2021

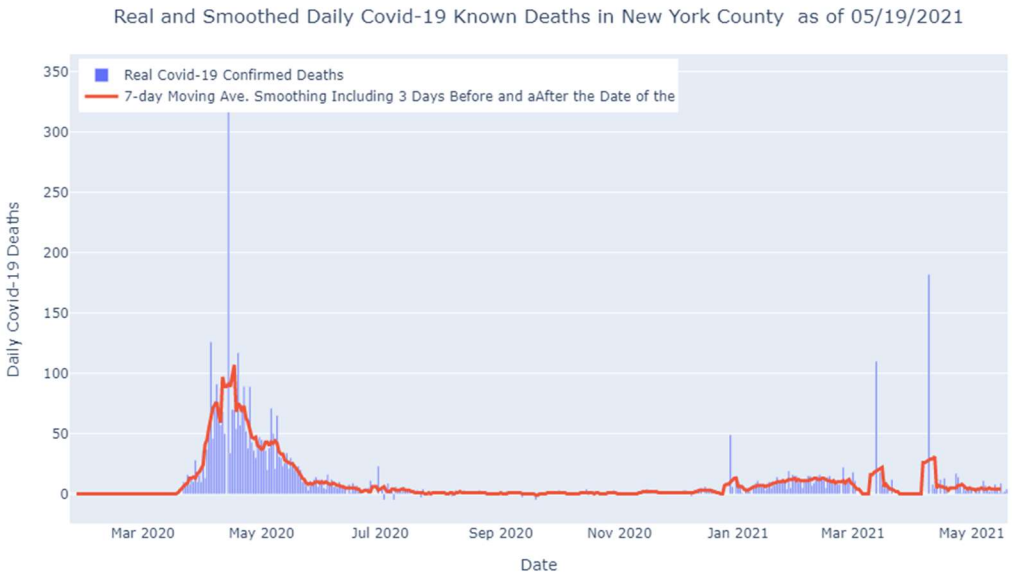


Figure 9: Unsmoothed and smoothed daily Covid-19 deaths in NY County as of 05/19/2021

3.1.1 Pre-processing of Data

The original dataset contains the cumulative number of Covid-19 deaths at the county level. To make this dataset suitable for our models, we computed the first lag differences to find the daily number of Covid-19 deaths. The data are preprocessed in this way before being passed to the training algorithms. In addition, we normalized the data as discussed in Section (3.1.1.1) since the numbers of deaths in nearby counties vary considerably, potentially complicating our analyses.

3.1.1.1 Data Normalization

Using the unscaled data slows down the convergence process of the loss function. A loss function measures the accuracy of the predictive function over the training epochs (an epoch refers to one cycle running the model through the full training dataset). Data normalization is a rescaling technique that maps the numerical dataset from the original range scale to values between 0 and 1. The data normalization approach subtracts the minimum value of a feature from each data value and divides the result by the range of the feature. The range is the difference between the

unscaled maximum and unscaled minimum values. This normalization preserves the shape of the distribution of the unscaled data. It requires accurately estimating the minimum and maximum values from the available unscaled data. Mathematically, we normalized the features in the smoothed collected deaths data as follows:

$$x_{ik} = \frac{death_{ik} - min_k}{max_k - min_k},$$

where x_{ik} and $death_{ik}$ are the normalized and non-normalized daily Covid-19 death counts in the k^{th} county on the i^{th} day, respectively, and min_k and max_k are the minimum and maximum values for the k^{th} county, respectively. Similarly, we normalized the target variable.

3.2 Artificial Neural Networks

The idea behind artificial neural networks (ANNs) came from theories of how the human brain solves complex challenges. ANNs have become popular in recent years, though, as early as the 1940's, McCulloch and Pitts [4] performed primary studies describing how neurons could work. ANNs with many layers are called deep neural networks. These days, big technology companies are investing heavily in ANNs and deep learning research and both are popular areas of academic research.

3.2.1 Activation Functions

Activation functions are used to introduce nonlinear transformations into the NN algorithm and make multilayer NNs more powerful. The backpropagation learning algorithm is an implementation of the chain rule followed by gradient descent. In order to use the backpropagation algorithm, activation functions are required to be differentiable, and are more useful if they are bounded.

A linear activation function ($y = cx$) creates an output signal proportional to the input. However, this function has two main problems. The first problem is that its derivative with respect to the input x is the constant c, and does not depend on x. Therefore, it is not possible to use the gradient descent optimization to train the model since we cannot go back and understand which input weights make better predictions. The second problem is that a linear activation function turns a deep neural network into a single-layer NN because a linear combination of linear activation functions will be a linear function. In other words, NNs with a linear activation function are simply linear regression models. They have limited ability and power to handle the complexity of input data.

Non-linear activation functions are essential for learning and modeling complex data, such as those originating from images, videos, and audios. NNs that use non-linear activation functions can create complex mappings between the network's inputs and network's outputs. These functions address the problems of linear activation functions. Because most often they have a derivative function that depends on the inputs, then they allow for backpropagation learning. Also, they allow a NN to have meaningful multiple hidden layers and thus enable deep learning. In this study, we used several non-linear activation functions including the Sigmoid, Tanh, ReLU, ELU, and SELU functions. These are explained in the following sections.

3.2.1.1.1 Sigmoid Activation Function

The most common non-linear continuously differentiable monotonic activation function is the sigmoid function. This function is also called the logistic sigmoid function. It takes a real value as input and its output is bounded on the unit interval $(0,1)$. This activation function is illustrated in Figure 10 to the left.

$$\sigma(z) = (1 + e^{-z})^{-1}, z \in \mathbb{R}. \quad (3\text{-}1)$$

3.2.1.1.2 Hyperbolic Tangent Activation Function

Another popular non-linear differentiable monotonic activation function is the hyperbolic tangent (Tanh) activation function which has a similar shape to the logistic function, but the values map to the interval [-1, 1]. This function is given by Formula (3-2), and takes the shape shown in Figure 10 to the right. Tanh is much steeper around 0 than sigmoid; therefore, its gradient is stronger and larger than the sigmoid.

$$\text{Tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, z\epsilon\mathbb{R}. \tag{3-2}$$

The Tanh activation function may cause the vanishing gradient problem. It has gradients in the range $(0, 1)$, and the backpropagation algorithm computes gradients by the chain rule. In an n-layer neural network, this has the effect of multiplying n of these small numbers to compute gradients of the early layers. This means that the gradient decreases exponentially with n while the early layers train slowly.

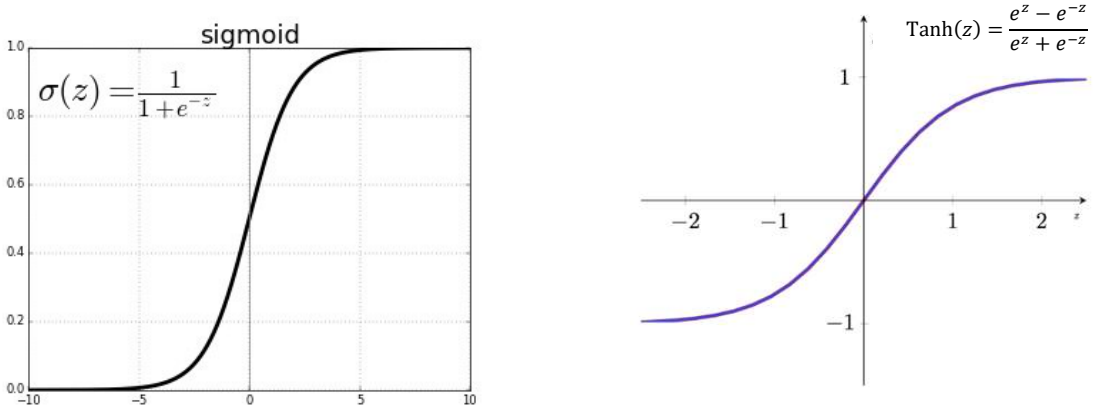


Figure 10: Left: Sigmoid activation function; Right: Tanh activation function

3.2.1.1.3 Rectified Linear Unit Activation Function

The rectified linear unit (ReLU) activation function given in Equation (3-3) and illustrated in Figure 11 (leftmost graph) is computationally more efficient than the

Tanh and Sigmoid functions, which allows the network to converge very quickly. The ReLU function only needs to choose $max\{0, z\}$ and thus does not perform expensive exponential operations as with both Sigmoid and Tanh.

$$ReLU(z) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases} = max\{0, z\}. \quad (3\text{-}3)$$

In other words, the ReLU function replaces negative values with 0 and leaves positive values unchanged. ReLU should only be used within hidden layers. This function is differentiable everywhere except at a singular point $z = 0$. In practice, it is relatively rare to have the point $z = 0$ in the context of deep learning, hence, we usually do not have to worry too much about the ReLU derivative at zero. Typically, we set it either to 0, 1, or 0.5. For $z > 0$, RELU tends to inflate the activation value with an output range of $[0, \infty)$.

A potential problem with ReLU activation functions is the case where a large weight update can mean that the summed input to the activation function is always negative, regardless of the input values. In other words, a node with this problem will always output an activation value of 0. The affected cell cannot contribute to the NN learning and its gradient stays at zero. So, while ReLUs avoid the problem of vanishing gradients, they give rise to a new problem referred to as dying ReLUs.

3.2.1.1.4 Exponential linear Unit Activation Function

The exponential linear unit (ELU) activation function given in Formula (3-4) is a function that tends to converge the loss function to zero faster and produce more accurate results. This ELU function is illustrated in Figure 12 and is more computationally expensive than the ReLU function.

$$ELU(z) = \begin{cases} \alpha(e^z - 1) & z \leq 0, \alpha > 0 \\ z & z > 0 \end{cases} \quad (3\text{-}4)$$

A common value for α is between 0.1 and 0.3. ELU is the same as ReLU for $z > 0$ and leaves positive values unchanged but gives a value slightly below zero for $z \leq 0$. Hence, unlike the ReLU function, ELU can produce negative values and avoids the dying ReLU problem. For $z > 0$, ELU can inflate the activation value with the output range of $[0, \infty)$.

3.2.1.1.5 Scaled Exponential Linear Unit Activation Function

The scaled exponential linear unit (SELU) activation function is a scaled variant of the ELU function. SELU learns faster and better than other activation functions. The SELU function is given in Equation (3-5) and illustrated in Figure 13.

$$SELU(z) = \lambda \begin{cases} \alpha(e^z - 1) & z \leq 0 \\ z & z > 0 \end{cases}, \tag{3-5}$$

where the values of the two fixed parameters α and λ are derived from the inputs. However, for standardized inputs the suggested values are $\alpha = 1.6733$ and $\lambda = 1.0507$. The SELU activation function has the important property of self-normalization meaning that output from SELU will preserve the mean of 0 and standard deviation of 1. The SELU function solves the vanishing gradients problem. As opposed to the ReLU activation function, the SELU avoids the dying ReLU problem as well.

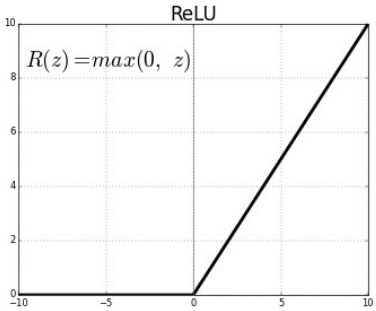


Figure 11: ReLU activation function

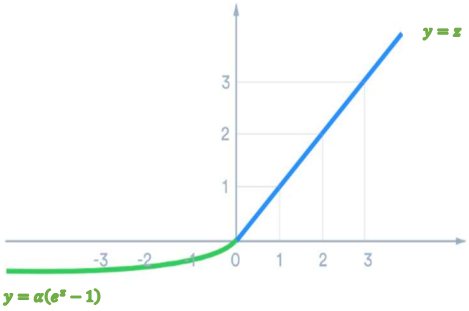


Figure 12: ELU activation function

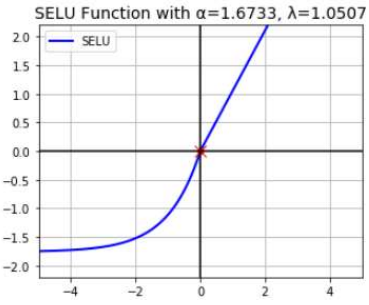


Figure 13: SELU activation function

3.2.2 Architecture of a Feed-forward Network

A feed-forward network is an ANN that sends inputs from one layer to another layer. Each layer consists of a set of neurons or *nodes* where each node is a function $f(x)$ that is fully connected to all other nodes in the next layer and each function receives the same input (x). The output of a network is the output of the last layer. Any layer between the first layer and the last layer is called a hidden layer. It is common to use one or two hidden layers. Figure 14 represents an example of a feed-forward network architecture having an input layer with three inputs, one hidden layer with four nodes, and an output layer with a single output. For context, suppose that the objective is to map a 1×3 input vector $\boldsymbol{x}$ to a scalar prediction $\hat{y}$.

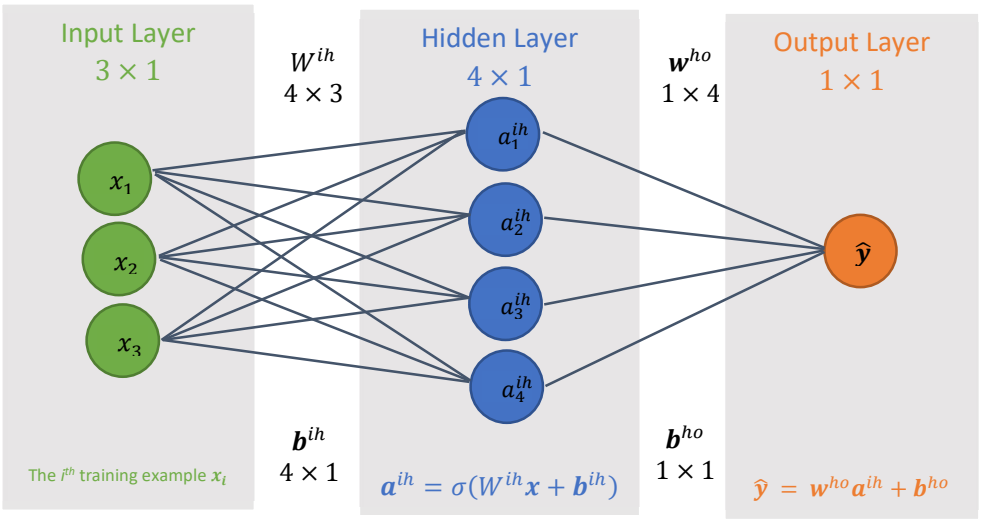


Figure 14: A diagram of a neural network

Figure 14 shows a three-four-one feed-forward network structure. The input layer receives signals x_1, x_2, and x_3. Weights w_{ij}^{ih} and bias terms b_i^{ih} (for $i = 1, 2, 3, 4$ and $j = 1, 2, 3$) are scalars as shown in Equation (3-6) where the superscript ih indicates that they are assigned to the connections between all the nodes in the input layer and the hidden layer .

$$W_{4\times3}^{ih} = \begin{bmatrix} w_{11}^{ih} & w_{12}^{ih} & w_{13}^{ih} \\ w_{21}^{ih} & w_{22}^{ih} & w_{23}^{ih} \\ w_{31}^{ih} & w_{32}^{ih} & w_{33}^{ih} \\ w_{41}^{ih} & w_{42}^{ih} & w_{43}^{ih} \end{bmatrix}, \quad \boldsymbol{b}_{4\times1}^{ih} = \begin{bmatrix} b_1^{ih} \\ b_2^{ih} \\ b_3^{ih} \\ b_4^{ih} \end{bmatrix}. \tag{3-6}$$

Weights w_k^{ho} and bias term b^{ho} (for all $k = 1, 2, 3, 4$) are scalars as shown in Equation (3-7) where the superscript ho indicates that they are assigned to the connections of all the nodes between the hidden layer and the output layer.

$$\boldsymbol{w}_{1\times4}^{ho} = [w_{11}^{ho} \quad w_{12}^{ho} \quad w_{13}^{ho} \quad w_{14}^{ho}], \quad b_{1\times1}^{ho} = b_1^{ho}. \tag{3-7}$$

Matrix W^{ih} is multiplied by the input vector $\boldsymbol{x}$ and the bias term $\boldsymbol{b}^{ih}$ is added to the product in the hidden layer. Additionally, an activation function, such as the sigmoid function defined in Equation (3-1) is applied to this sum to give new input signals a_j^{ih} for $j = 1,2,3,4$ for the next layer as given in Equation (3-8).

$$\boldsymbol{a}_{4\times1}^{ih} = \sigma\left(W^{ih}\boldsymbol{x} + \boldsymbol{b}^{ih}\right). \tag{3-8}$$

The vector $\boldsymbol{a}^{ih}$ is fed-forward and multiplied by the weights $\boldsymbol{w}^{ho}$ and added to the bias scalar b^{ho}. This resulting signal is sent through the identity activation function to give the network output $\hat{y}$ given in Formula (3-9):

$$\hat{y} = \boldsymbol{w}^{ho}\boldsymbol{a}^{ih} + b^{ho}. \tag{3-9}$$

Hence, the number of trainable weights in this feed-forward neural network is

$$\#\,\text{Trainable Parameters} = \underbrace{(3\times4)}_{For\ W^{ih}} + \underbrace{(4\times1)}_{For\ \boldsymbol{w}^{ho}} + \underbrace{(4+1)}_{For\ Biases} = 21.$$

To obtain the output, we used two differentiable linear and non-linear activation functions. Let ζ denote the activation functions (Section 3.2.1). In general, a one-hidden layer neural network can be expressed as the composition of two mappings. The first mapping sends a $p \times 1$ input vector $\boldsymbol{x}$ to q different outputs by multiplying

a $q \times p$ weight matrix W^{ih} by a $p \times 1$ input vector $\boldsymbol{x}$ and adding a $q \times 1$ bias term $\boldsymbol{b}^{ih}$ to the product in the hidden layer. Additionally, an activation function $\zeta^{ih}: \mathbb{R} \to \mathbb{R}$ is applied to this sum to give a new $q \times 1$ input vector $\boldsymbol{a}^{ih}$ for the next layer. The transformation of the vector $W^{ih}\boldsymbol{x} + \boldsymbol{b}^{ih}$ by activation function ζ^{ih} is defined in Equation (3-10).

$$\boldsymbol{a}^{ih}_{q\times 1} = \zeta^{ih}\left(W^{ih}\boldsymbol{x} + \boldsymbol{b}^{ih}\right). \tag{3-10}$$

Note that the same p input signals are sent to each of the hidden layer nodes. However, the signal received by the j^{th} node is $\zeta^{ih}\left(\boldsymbol{w}^{ih}_{j.}\boldsymbol{x} + b^{ih}_{j}\right)$ where $\boldsymbol{w}^{ih}_{j.}$ is the j^{th} row of W^{ih}. Since the weight vectors $\boldsymbol{w}^{ih}_{1.}, \ldots, \boldsymbol{w}^{ih}_{q.}$ are generally different, the signals received at different hidden layer nodes differ.

The second mapping sends the new $q \times 1$ signal input vector $\boldsymbol{a}^{ih}$ received in the hidden layer to s different network outputs by multiplying an $s \times q$ weight matrix W^{ho} by the new $q \times 1$ input $\boldsymbol{a}^{ih}$ and adding an $s \times 1$ bias term $\boldsymbol{b}^{ho}$ to the product in the output layer. Additionally, an activation function $\zeta^{ho}: \mathbb{R} \to \mathbb{R}$ is applied to this sum to give an $s \times 1$ network output $\widehat{\boldsymbol{y}}$. The transformation of the vector $W^{ho}\boldsymbol{a}^{ih} + \boldsymbol{b}^{ho}$ by function ζ^{ho} is defined in Equation (3-11).

$$\widehat{\boldsymbol{y}}_{s\times 1} = \zeta^{ho}\left(W^{ho}\boldsymbol{a}^{ih} + \boldsymbol{b}^{ho}\right). \tag{3-11}$$

Hence, the number of trainable weights is

$$\begin{aligned}\#\,\text{Trainable Parameters} &= \underbrace{(q \times p)}_{For\ W^{ih}} + \underbrace{(s \times q)}_{For\ W^{ho}} + \underbrace{(q + s)}_{For\ Biases} \\ &= N(i + o) + (N + o),\end{aligned}$$

where N is the number of hidden nodes, i is the number of inputs, and o is the number of outputs. In the previous example, $i = 3$, $N = 4$, and $o = 1$.

3.3 General Architecture of Feed-forward Networks

We turn now to determining good (perhaps optimal) weight matrices W^{ih} and W^{ho}, and bias vectors $\boldsymbol{b}^{ih}$ and $\boldsymbol{b}^{ho}$ where ih indicates that those weights and biases are for the connections between the input layer and the hidden layer. Similarly, ho indicates that those weights and biases are for the connections between the hidden layer and the output layer. We add a row of ones corresponding to the bias vectors to the $p \times n$ matrix of the predictors X and then say the matrix has been augmented and denote the $(p+1) \times n$ augmented matrix of the predictors as $\begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}$. Also, we combine the weights and biases together in each connection and introduce them as augmented matrices $[W^{ih} \quad \boldsymbol{b}^{ih}]$ and $[W^{ho} \quad \boldsymbol{b}^{ho}]$.

In general, a one-hidden layer neural network can be expressed as the composition of two mappings. The first mapping sends a $(p+1) \times n$ input augmented matrix $\begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}$ to a $q \times n$ matrix of outputs $\boldsymbol{A}^{ih}$ by multiplying a $q \times (p+1)$ weight matrix $[W^{ih} \quad \boldsymbol{b}^{ih}]$ by a $(p+1) \times n$ augmented matrix $\begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}$ in the hidden layer. Additionally, an activation function $\zeta^{ih}: \mathbb{R} \to \mathbb{R}$ is applied to this product to give a new $q \times n$ input matrix $\boldsymbol{A}^{ih}$ for the next layer. The transformation of the matrix $[W^{ih} \quad \boldsymbol{b}^{ih}] \begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}$ by the activation function ζ_{ih} is defined in Equation (3-12).

$$A^{ih}_{q \times n} = \zeta^{ih}\left([W^{ih} \quad \boldsymbol{b}^{ih}] \begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}\right) \tag{3-12}$$

The second mapping sends the new $(q+1) \times n$ signal input augmented matrix $\begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix}$ received in the hidden layer to n different network outputs by multiplying a $1 \times (q+1)$ weight vector $[\boldsymbol{w}^{ho} \quad b^{ho}]$ by the new $(q+1) \times n$ input augmented

matrix $\begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix}$ in the output layer. Additionally, an activation function $\zeta^{ho}: \mathbb{R} \to \mathbb{R}$ is applied to this product to give a $1 \times n$ network output $\hat{\boldsymbol{y}}$. The transformation of the matrix $[\boldsymbol{w}^{ho} \quad b^{ho}]\begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix}$ by the function ζ^{ho} is defined in Equation (3-13).

$$\hat{\boldsymbol{y}}_{1\times n} = \zeta^{ho}\left([\boldsymbol{w}^{ho} \quad b^{ho}]\begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix}\right). \tag{3-13}$$

Here, the number of trainable weights is

$$\begin{aligned} \#\,\text{Trainable Parameters} &= N(i+o) + (N+o) \\ &= q \times (p+1) + (q+1). \end{aligned}$$

3.3.1 Network learning

NNs learn new patterns by updating their weights between all the layers. The architecture of an NN determines its learning ability of which there are two major types: supervised and unsupervised learning. In supervised learning, the training set (X_i, y_i) provides the learning rule. The inputs X_i are entered into the neural network to compute the outputs $\hat{y}_i$, which are then compared to the targets $y_i, i = 1, \ldots, n$. One supervised learning network procedure is the backpropagation algorithm, a well-known network learning procedure. The backpropagation algorithm is an implementation of the chain rule followed by gradient descent.

We now optimize the ANN by determining the optimal weights and biases $[W^{ih} \quad \boldsymbol{b}^{ih}]$ and $[\boldsymbol{w}^{ho} \quad b^{ho}]$ through minimization of the loss function using gradient descent (Section 3.4). The learning rule is then used to update the network's weights and biases to minimize the squared error loss between the predicted vector $\hat{\boldsymbol{y}}_{1\times n}$ and the observed vector $\boldsymbol{y}_{1\times n}$, given as:

$$L([W \quad \boldsymbol{b}]) = (\boldsymbol{y}_{1\times n} - \hat{\boldsymbol{y}}_{1\times n})(\boldsymbol{y}_{1\times n} - \hat{\boldsymbol{y}}_{1\times n})^T = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where $L([W \quad \boldsymbol{b}])$ is the network's squared error loss function parameterized by the model's weights and biases of interest $[W \quad \boldsymbol{b}] \in \mathbb{R}^{p+1}$.

Analytically, minimizing the loss function with respect to W by computing the partial derivatives, setting them to zero and solving for W, yields an optimal weights matrix. Unfortunately, for most neural networks, the loss function is nonlinear. Thus, there exists no closed form solution and so it is not possible to use this approach. However, the backpropagation system using gradient descent optimization computes the network weights iteratively and requires the partial derivatives $\nabla_W L([W \quad \boldsymbol{b}])$. These derivatives are calculated using the chain rule. For example, $\nabla_{\boldsymbol{w}^{ho}} L([W \quad \boldsymbol{b}])$ is calculated as follows:

$$\begin{aligned}
\underbrace{\nabla_{\boldsymbol{w}^{ho}} L([W \quad \boldsymbol{b}])}_{1\times(q+1)} &= \frac{\partial L([W \quad \boldsymbol{b}])}{\partial \hat{\boldsymbol{y}}} \frac{\partial \zeta^{ho}}{\partial \boldsymbol{w}^{ho}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \zeta^{ho}} \\
&= -2(\boldsymbol{y} - \hat{\boldsymbol{y}}) \left(\zeta'_{ho} \left([\boldsymbol{w}^{ho} \quad b^{ho}] \begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix} \right) \odot \frac{\partial \boldsymbol{w}^{ho} \boldsymbol{A}^{ih}}{\partial \boldsymbol{w}^{ho}} \right) \\
&= -2 \underbrace{(\boldsymbol{y} - \hat{\boldsymbol{y}})}_{(1\times n)} \underbrace{\left(\underbrace{D_{\zeta'_{ho}\left([\boldsymbol{w}^{ho} \quad b^{ho}]\begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix}\right)}}_{(n\times n)} \underbrace{\left(\boldsymbol{A}^{ih}\right)^T}_{(n\times(q+1))} \right)}_{(n\times(q+1))},
\end{aligned}$$

where $\odot$ denotes the Hadamard or element-wise product of matrices and $D_{\zeta'_{ho}\left([\boldsymbol{w}^{ho} \quad b^{ho}]\begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix}\right)}$ is the corresponding diagonal matrix for the $(1 \times n)$-vector $\zeta'_{ho} \left([\boldsymbol{w}^{ho} \quad b^{ho}] \begin{bmatrix} \boldsymbol{A}^{ih} \\ \mathbf{1} \end{bmatrix} \right)$ with this vector as its main diagonals.

3.4 Gradient Descent Optimization

Gradient descent is an iterative optimization algorithm and the most common way to optimize NNs by minimizing a network's loss function $L(\boldsymbol{w})$ parameterized by a model's weights $\boldsymbol{w} \in \mathbb{R}^p$. The loss function describes how well the model will perform given the current set of weights, and gradient descent algorithm is a way to

find the best set of weights by updating the weights in the opposite direction of the gradient of the loss function $\nabla_{\boldsymbol{w}} L(\boldsymbol{w})$ with respect to the weights.

There are three variants of gradient descent optimization: Batch gradient descent, Stochastic gradient descent, and Mini-batch gradient descent. They vary in the amount of data we use to compute the gradient of the loss. Depending on how much data we use to compute $\nabla_{\boldsymbol{w}} L(\boldsymbol{w})$, there is a trade-off between the accuracy of the weight update and the time it takes to perform the update.

3.4.1 Batch gradient descent

Batch gradient descent (BGD) calculates the gradient of the loss function $L(\boldsymbol{w})$ with respect to the weights $\boldsymbol{w}$ for the entire training dataset:

$$\boldsymbol{w} = \boldsymbol{w} - \eta \cdot \nabla_{\boldsymbol{w}} L(\boldsymbol{w}),$$

where η is the learning rate and determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the loss function downhill until we reach a valley. Hence, BGD converges to the minimum of the basin the weights are placed in. Since we need to compute the gradients for the entire dataset to do just one update, BGD cannot only be very slow but is also problematic for datasets with large memory requirements. Moreover, it does not allow us to update our model online (with new examples).

3.4.2 Stochastic gradient descent

Stochastic gradient descent (SGD) performs a weight update for each training example $x^{(i)}$ and label $y^{(i)}$:

$$\boldsymbol{w} = \boldsymbol{w} - \eta \cdot \nabla_{\boldsymbol{w}} L\left(\boldsymbol{w}; x^{(i)}; y^{(i)}\right).$$

The training examples in the training set are randomly permuted at every epoch. SGD performs one update at a time and is usually much faster allowing us to update our model online. It performs many updates with high variance that causes the loss

function to fluctuate greatly. These fluctuations enable SGD to jump to new and potentially better local minima but complicates convergence to the exact minimum.

3.4.3 Mini-batch gradient descent

Mini-batch gradient descent (Mini-BGD) performs an update for every mini-batch of n training examples:

$$\boldsymbol{w} = \boldsymbol{w} - \eta \cdot \nabla_{\boldsymbol{w}} L\left(\boldsymbol{w}; x^{(i:i+n)}; y^{(i:i+n)}\right).$$

Mini-BGD not only reduces the variance of the weight updates, which can lead to more stable convergence, but also can make computing the gradient with respect to a mini-batch very efficient.

Mini-BGD does not guarantee good convergence and offers a few challenges that need to be addressed. For example, selecting an appropriate learning rate can be difficult. While choosing a very small η leads to slow convergence, choosing a very large η can hinder convergence and cause the loss to fluctuate around the minimum or even to diverge. There are some algorithms like the Adam optimizer that are widely used in NNs to deal with such challenges.

3.4.4 Adam Optimizer

Adaptive Moment Estimation (Adam), introduced by Kingma and Ba [40], is an optimization method that computes adaptive learning rates for each weight. In addition to storing an exponentially decaying average of past squared gradients v_t, Adam also retains an exponentially decaying average of past gradients m_t:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_{\boldsymbol{w}} L(\boldsymbol{w}_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)[\nabla_{\boldsymbol{w}} L(\boldsymbol{w}_t)]^2,$$

where m_t and v_t are estimates of the first and second moments of the gradients, respectively. β_1 and β_2 are the exponential decay rates for the corresponding m_t and

v_t. Since these estimates are initialized as vectors of 0's, the authors of the Adam method observe that they are biased towards 0. These biases are neutralized by calculating bias-corrected estimates of the first and second moments:

$$\hat{m}_t = \frac{m_t}{1-\beta_1}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2}.$$

The Adam rule for updating the weights is as follows:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t,$$

where ϵ is a very small number to prevent any division by zero. The authors suggest a default value of 0.9 for β_1, a default value of 0.999 for β_2, and a default value of 10^{-8} for ϵ. They illustrate empirically that the Adam optimizer works well in practice using these default values.

3.5 Regularizing Deep Neural Networks

Deep learning neural networks are likely to overfit a relatively small training dataset and to increase generalization error which results in poor performance when the model is evaluated on a test dataset. An effective regularization method to reduce overfitting in large neural networks (more layers or nodes) of all kinds is the dropout method. This method is applied to a model by randomly dropping out nodes during the training model.

Another approach to regularization is fitting all possible different neural network models on the same training dataset and taking average predictions from each model. In practice, this is not feasible, but can be done using an *ensemble approximation* (a small collection of different models). One problem with this is that it requires multiple models to be fit, which can be a problem for large datasets that require multiple days to train and tune.

3.5.1 Dropout

Dropout is a regularization method that approximates training many neural networks with different designs in parallel. It makes the training process noisy and forces nodes within a hidden layer to probabilistically take different responsibility for the inputs. In other words, setting the dropout rate to a specific rate (say 50%) indicates that the prior nodes should be dropped from training with the assigned rate or the probability of a certain node to be dropped from the training as 50%, and so the remaining nodes will have different worth.

Dropout regularization can be used with most types of neural networks such as the multilayer perceptron and gated recurrent unit (GRU). It can also be used with most types of layers, such as dense and recurrent layers like the GRU layer. For example, the input and recurrent connections in the GRU layer may use different dropout rates.

When training a NN model, some outputs of its hidden layers are randomly ignored or *dropped out*. This treats the current layer as having a different number of nodes and connectivity to the prior layer.

3.5.2 Dropout Rate

Dropout rate is the probability of training a given node in a layer. Because the outputs of a layer under dropout regularization are randomly subsampled, this reduces the capacity of the network during training. Hence, the network may require more nodes when using the dropout method. A good dropout rate in a hidden layer is between 0.5 and 0.8. A common dropout rate is 0.5. Rates close to 1.0 indicate no outputs from the layer and rates close to 0.0 mean no dropout. Note that input layers use larger rates.

Using dropout makes the model's weights larger. Hence, before the final network layer, the weights should be rescaled by the chosen dropout value. Then the network model can be used to make predictions as normal.

3.5.3 Use of Wider Neural Networks

Using dropout has made it possible to use larger networks with less risk of overfitting. In fact, larger neural networks are probably required as dropout will probabilistically thin the network. As a rule of thumb, the new larger network including the dropout regularization contains a greater number of nodes. This is computed by dividing the number of nodes in the previous network layer before using dropout by the chosen dropout rate [41].

3.5.4 Early Stopping Approach

The early stopping approach is perhaps one of the oldest and most widely used forms of neural network regularization. When training a large neural network, there will be a point where the model will stop generalizing and start overfitting the training dataset, making the model less useful at predicting new data. It is important to train a neural network long enough that it can learn the data, but not train the network so long that it overfits the training data.

The way this approach works is to train a network once over a larger number of epochs than would normally be required to give the network plenty of room to fit and just start to overfit the training dataset. At the point where the performance of the model on the validation dataset starts to degrade, the training process is stopped. The model at the time of stopping is then used and is known to have good generalization performance.

An alternative approach to overcoming overfitting is to treat the number of training epochs as a hyperparameter, train the network multiple times with different epochs, and then choose the number of epochs that results in the best performance on the training or test dataset. This can be computationally expensive and time-consuming.

3.6 Assessment of fits

In all algorithms, the mean absolute error (MAE) is used to evaluate the performance of the models. The MAE is a scale-dependent accuracy measure that uses the same scale as the data being measured.

$$MAE = \frac{1}{n}\sum |y_i - \hat{y}_i|, \tag{3-14}$$

where $|.|$ is the absolute value function, y_i is the i^{th} target value, $\hat{y}_i$ is the i^{th} predicted value, $i = 1, \dots, n$, and n is the number of observations.

To assess a prediction band, the coverage probability (CP) of the band is computed. The coverage probability is the percentage of target values y_i's covered by the prediction band using the predicted values:

$$\text{CP} = \frac{1}{n}\sum_{i=1}^{n} 1_{y_i \in (L_i, U_i)}, \tag{3-15}$$

where L_i and U_i are the lower and upper limits of the i^{th} prediction band, respectively. The performance of a prediction band can be evaluated by comparing the coverage probability of the interval and the nominal prediction level. A forecasted model performs well if the coverage probability and the nominal prediction level are close.

3.7 Recurrent Neural Networks

Rumelhart et al. (1986) [8] introduced recurrent neural networks or RNNs which are a family of NNs. They are capable of remembering past information for processing new events accordingly. Hence, they are designed for modeling sequence data. In a feedforward NN, information flows from input layers to hidden layers, and then from hidden layers to output layers. In RNNs, the hidden layers take their inputs from both the input layers of the current time steps and the hidden layers from the previous time steps. The main idea behind RNNs is to use not only the input sequence $\boldsymbol{x}_t$ at

the current time step t (where t indicates the time step of an input), but also the previous outputs $\boldsymbol{h}_{t-1}$ for making the current prediction. Hence, an RNN accepts an input sequence $\boldsymbol{x}_t$ at time t and assumes that there is a hidden state $\boldsymbol{h}_{t-1}$ to represent the system status at time $t-1$. The status is updated by a nonlinear mapping function *f* from one time step to the next:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t).$$

One common way of defining the recurrent unit *f* is with a linear transformation plus a nonlinear function ζ which applies to every component of the inputs, e.g.,

$$\boldsymbol{h}_t = \zeta\left(\underbrace{[W^{ih} \quad \boldsymbol{b}^{ih}]}_{q\times(p+1)} \underbrace{\begin{bmatrix}\boldsymbol{x}_t \\ \mathbf{1}\end{bmatrix}}_{(p+1)\times 1} + \underbrace{W^{hh}}_{q\times q}\underbrace{\boldsymbol{h}_{t-1}}_{q\times 1}\right), \tag{3-16}$$

where W^{ih} and W^{hh} are the weight matrices related to $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$, respectively, and $\boldsymbol{b}^{ih}$ is a vector of bias terms.

The task of an RNN is to learn the weight matrices W^{ih} and W^{hh} and the bias vector $\boldsymbol{b}^{ih}$. An RNN may also have an optional output $\boldsymbol{y}_t$ in addition to the hidden state $\boldsymbol{h}_t$. A simple RNN only has simple recurrent operations without any gates to control the flow of information among the cells. The gates are a linear transformation plus the nonlinear sigmoid activation function. However, an RNN in simple form suffers from the vanishing or exploding gradient problem, especially in long sequences ([42], [9]). In machine learning, these problems can occur when training NNs using gradient-based learning algorithms and backpropagation. In such algorithms, each of the NN weights receives an update proportional to the partial derivative of the loss function with respect to the current weight in each iteration of training. The problems are that in some cases, the gradient will be either vanishingly small, effectively preventing the weight from updating, or so large, effectively overflow the weight. These may completely stop the NN from further training. We will begin by introducing a widely used NN called gated recurrent units to solve this problem.

3.7.1 Gated Recurrent Unit

Cho et al. (2014) introduced the gated recurrent unit (GRU), an RNN that is comprised of two internal gates which are: an update gate $\boldsymbol{z}_t$ and a reset gate $\boldsymbol{r}_t$ given in Formula (3-17) and (3-18). This means that we have dedicated procedures for when a hidden state should be *updated* and also when it should be *reset*. For instance, a reset gate would allow us to control how much of the previous state we might still want to remember. An update gate decides how much information is updated. If the reset gate is set to zero, it reads input sequences and forgets the previously calculated state. Unlike other RNNs, the network's internal gates allow the model to be trained successfully using backpropagation through time and avoids the vanishing gradients problem. The data flow and operations in the GRU are illustrated in Figure 15. The GRU's rules are given in Formula (3-17)- (3-20),

$$\textit{Update Gate:} \qquad \boldsymbol{z}_t = \sigma(W_z[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_{\boldsymbol{z}}) \tag{3-17}$$

$$\textit{Reset Gate:} \qquad \boldsymbol{r}_t = \sigma(W_r[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_{\boldsymbol{r}}), \tag{3-18}$$

where $\sigma(.)$ is the sigmoid function and is applied to every component of the input to transform input values to the unit interval $(0,1)$. The matrices W_z and W_r contain the combined weights for $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$, and $\boldsymbol{b}_{\boldsymbol{z}}$ and $\boldsymbol{b}_{\boldsymbol{r}}$ are the bias vectors. The vector $\boldsymbol{z}_t$ is referred to as the *update gate* and is determined by the weights in $W_{\boldsymbol{z}}$ and $\boldsymbol{b}_{\boldsymbol{z}}$. The vector $\boldsymbol{r}_t$ is referred to as the *reset gate* and is determined by the weights in $W_{\boldsymbol{r}}$ and $\boldsymbol{b}_{\boldsymbol{r}}$. These formulae have the same form. The difference comes in the weights and biases and the gate's usage. When $\boldsymbol{x}_t$ is plugged into these gates, it is multiplied by its own weight. The same is true for $\boldsymbol{h}_{t-1}$ which holds the information for the previous $t-1$ units and is multiplied by its own weight. Both results are added together, and a sigmoid function is applied to force the result between 0 and 1. There

is a new memory content called the candidate hidden state which uses the reset gate to store the relevant information from the past and is expressed as follows:

Candidate Hidden State: $$\widetilde{\boldsymbol{h}}_t = \tanh(W_h[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_{\boldsymbol{h}}) \quad (3\text{-}19)$$

where $\odot$ means the component-wise product between two vectors. The activation function "tanh", defined in Formula (3-2), is applied to every element of its input. The candidate hidden state $\widetilde{\boldsymbol{h}}_t$ is a linear transformation plus the nonlinear tanh activation function determined by the weights in $W_{\boldsymbol{h}}$ and $\boldsymbol{b}_{\boldsymbol{h}}$. The task of the GRU is to learn these weights and biases. Lastly, to control information inside the network units, there is a hidden state $\boldsymbol{h}_t$ as shown in Formula (3-20):

Hidden State: $$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \widetilde{\boldsymbol{h}}_t \quad (3\text{-}20)$$

where the update gate $\boldsymbol{z}_t$ determines the portion of $\boldsymbol{h}_{t-1}$ and $\widetilde{\boldsymbol{h}}_t$ to be updated.

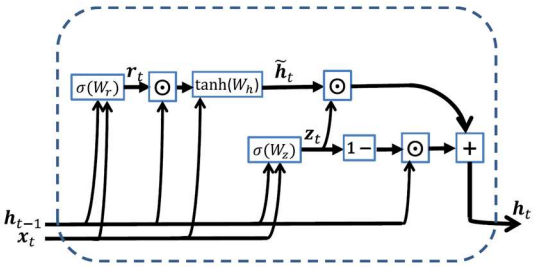


Figure 15: Flow and operations in a GRU cell, Figure source [43]

A GRU cell may also have an optional output $\boldsymbol{y}_t$. It has no additional memory cell to retain information; therefore, it can only control information inside the unit.

3.7.1.1 Number of Trainable Weights in a GRU

In a GRU, the number of trainable weights is given by:

$$g[N(N + i) + N], \quad (3\text{-}21)$$

where g is the number of Feed Forward Neural Networks (FFNNs) in the GRU (which is 3), N is the number of hidden nodes, and i is the number of inputs. In the next chapter, we will see the GRU's application in modeling sequence data.

3.7.1.2 Producing Rolling Windows

Because we aim to exploit predictive information induced by serial correlation, it is necessary to preserve the temporal structure of the data. Hence, rolling (or moving) windows are used to capture this temporal information. The rolling window size (m) is the number of consecutive days per rolling window. The size of the rolling window should be tuned for each model.

We assume that the time increment between successive rolling windows is one day. The first rolling window contains the first m observations from day 1 through day m; the second one contains observations from day 2 through day ($m + 1$), and so on. This is illustrated in Figure 16. Thus, if we continue this procedure, we can build $n - m + 1$ rolling windows of length m from the entire dataset where n is the number of observations.

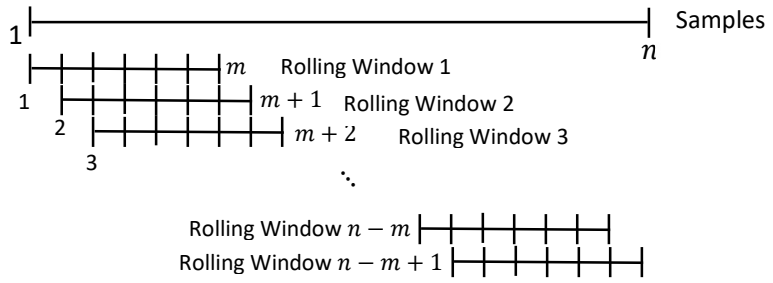


Figure 16: Partitioning a sample to $n - m + 1$ rolling windows.

Let $\boldsymbol{R}$ denote the set of all rolling windows. An element of $\boldsymbol{R}$ is a rolling window R_t for $t = 1, \ldots, n - m + 1$, where m is the rolling window size, as given by:

$$R_t = \begin{bmatrix} x_{t1} & x_{t2} & \cdots & x_{tp} \\ \vdots & \vdots & \ddots & \vdots \\ x_{t+m-1,1} & x_{t+m-1,2} & \cdots & x_{t+m-1,p} \end{bmatrix}, \tag{3-22}$$

where in our Covid-19 application, p is the number of neighboring counties and the x_{ik}'s are the daily Covid-19 deaths in county k on day i. Hence, the 3-dimensional rolling windows array $\boldsymbol{R}$ has the following form:

$$\boldsymbol{R} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_{n-m-h+1} \\ \vdots \\ R_{n-m+1} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,p} \end{bmatrix}_{(m,p)} \\ \begin{bmatrix} x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m+1,1} & x_{m+1,2} & \cdots & x_{m+1,p} \end{bmatrix}_{(m,p)} \\ \vdots \\ \begin{bmatrix} x_{n-m-h+1,1} & x_{n-m-h+1,2} & \cdots & x_{n-m-h+1,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-h,1} & x_{n-h,2} & \cdots & x_{n-h,p} \end{bmatrix}_{(m,p)} \\ \vdots \\ \begin{bmatrix} x_{n-m+1,1} & x_{n-m+1,2} & \cdots & x_{n-m+1,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix}_{(m,p)} \end{bmatrix}_{(n-m+1,m,p)} \tag{3-23}$$

where m and h are the rolling window size and the number of days ahead to forecast, respectively. Note that the last h rolling windows are only for forecasting purposes and we have not trained any models on them.

3.7.1.3 Producing Targets

In this study, we want to predict the number of daily Covid-19 deaths only using data from earlier time points for h days into the future in a desired county, as denoted $(y_{n+1}, \dots y_{n+h})$. Note that each of the p counties can be the target. Hence, we need to create a supervised training dataset to train a model. By considering the creation of rolling windows as discussed in Section (3.7.1.2), the first observation of the target vector $\boldsymbol{y}$ is the daily Covid-19 death count on the $(h+m)^{\text{th}}$ day in the targeted county where m is the rolling window size. The second observation of $\boldsymbol{y}$ is the death count on the $(h+m+1)^{\text{th}}$ day and so on. Hence, the target vector is given as

$$\boldsymbol{y} = (y_{h+m}, y_{h+m+1}, \dots, y_n). \tag{3-24}$$

3.8 Innovations

Artificial neural networks cannot handle temporal data since they assign each observation equal weights. However, to compare the results of GRU models with conventional NNs, we modified the NNs by adding observation weights to conventional neural networks (Section 3.8.1).

3.8.1 Weighted Neural Networks

We assume that the daily Covid-19 death count for a reported day is correlated with reported death counts for the previous m days. Thus, Covid-19 deaths counts are temporal data and fitting an ANN ignores the temporal structure. RNNs (Section 3.7) preserve and exploit temporal structure but are problematic to use and are computationally expensive. Hence, we propose a simple alternative referred to as the Weighted Neural Network (WNN). We modified the ANN by assigning a sample weight to each observation to learn the dynamic dependence structure present in the daily Covid-19 deaths data. To set up a WNN model, we place more importance on recent observations by introducing sample weights (denoted by v_j's) to the deaths data and modify the loss function accordingly. Note that these sample weights differ from the model's weights.

One might use the exponential decay weights given in Equation (3-25) to assign sample weights to the data. Let $\boldsymbol{v}$ be the $n \times 1$ exponential decay sample weights vector used to weight our dataset where each element of $\boldsymbol{v}$ is defined as:

$$v_j = \frac{\alpha(1-\alpha)^{j-1}}{\sum_j \alpha(1-\alpha)^{j-1}};\ j = 1, 2, \dots, n, \tag{3-25}$$

where $0 < \alpha < 1$ is a tuning constant to control the influence of observations. The sum of the sample weights is normalized to be exactly one.

We turn now to determining good (perhaps optimal) weight matrices W^{ih} and W^{ho}, and bias vectors $\boldsymbol{b}^{ih}$ and $\boldsymbol{b}^{ho}$ for training a one-hidden layer WNN using a set of training data where "ih" indicates the weights and biases for connections between the input layer and the hidden layer. Similarly, "ho" indicates weights and biases for connections between the hidden layer and the output layer. Recall that we added a row of ones corresponding to the bias vectors to the $p \times n$ matrix of predictors X and denoted the resulting $(p+1) \times n$ augmented matrix of the predictors as $\begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}$. If we augment a constant 1 corresponding to the bias to the $q \times 1$ input vector $\boldsymbol{a}^{ih}$, this gives a $(q+1) \times 1$ augmented vector of the inputs: $\begin{bmatrix} \boldsymbol{a}^{ih}_{q\times 1} \\ 1 \end{bmatrix}$. We then combined all weights and biases together from each connection to create the augmented matrices $[W^{ih} \quad \boldsymbol{b}^{ih}]$ and $[W^{ho} \quad \boldsymbol{b}^{ho}]$.

A one-hidden layer WNN can be expressed as the composition of two mappings. The first mapping sends the product of the $(p+1) \times n$ input augmented matrix $\begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}$ with the $n \times 1$ exponential decay sample weight $\boldsymbol{v}$ to a $q \times 1$ vector of outputs $\boldsymbol{a}^{ih}$ by multiplying the product $\begin{bmatrix} X \\ \mathbf{1} \end{bmatrix} \boldsymbol{v}$ by the $q \times (p+1)$ weight matrix $[W^{ih} \quad \boldsymbol{b}^{ih}]$ in the hidden layer. Additionally, an activation function $\zeta_{ih}: \mathbb{R} \to \mathbb{R}$ is applied to the final product to give a new $q \times 1$ input $\boldsymbol{a}^{ih}$ for the next layer. This transformation by the function ζ_{ih} is defined in Equation (3-26).

$$\boldsymbol{a}^{ih}_{q\times 1} = \zeta_{ih}\Bigg(\underbrace{\underbrace{[W^{ih} \quad \boldsymbol{b}^{ih}]}_{q\times(p+1)}\Bigg(\underbrace{\underbrace{\begin{bmatrix} X \\ \mathbf{1} \end{bmatrix}}_{(p+1)\times n} \underbrace{\boldsymbol{v}}_{n\times 1}}_{(p+1)\times 1}\Bigg)}_{q\times 1}\Bigg). \tag{3-26}$$

The second mapping sends the new $(q+1)\times 1$ augmented vector of signal input $\begin{bmatrix} \boldsymbol{a}^{ih}_{q\times 1} \\ 1 \end{bmatrix}$ received in the hidden layer to an $s\times 1$ vector of different network outputs by multiplying an $s\times(q+1)$ weight matrix $[W^{ho} \quad \boldsymbol{b}^{ho}]$ by the new $(q+1)\times 1$ input augmented vector $\begin{bmatrix} \boldsymbol{a}^{ih}_{q\times 1} \\ 1 \end{bmatrix}$ in the output layer. Additionally, an activation function $\zeta_{ho}\colon \mathbb{R}\to\mathbb{R}$ is applied to this product to give an $s\times 1$ network output $\widehat{\boldsymbol{y}}$. The transformation of the matrix $[W^{ho} \quad \boldsymbol{b}^{ho}]\begin{bmatrix} \boldsymbol{a}^{ih}_{q\times 1} \\ 1 \end{bmatrix}$ by the function ζ_{ho} is:

$$\widehat{\boldsymbol{y}}_{s\times 1} = \zeta_{ho}\underbrace{\left(\underbrace{[W^{ho} \quad \boldsymbol{b}^{ho}]}_{s\times(q+1)}\underbrace{\begin{bmatrix} \boldsymbol{a}^{ih}_{q\times 1} \\ 1 \end{bmatrix}}_{(q+1)\times 1}\right)}_{s\times 1}. \tag{3-27}$$

We then optimize the WNN model by determining optimal weights and biases $[W^{ih} \quad \boldsymbol{b}^{ih}]$ and $[W^{ho} \quad \boldsymbol{b}^{ho}]$ through minimization of the loss function that measures the accuracy of the predictive function over the training epochs using gradient descent (Section 3.4).

3.8.2 Bootstrap Aggregation of Neural Networks

One disadvantage of NNs is that they do not provide prediction bands (PIs) for forecasted values. PIs quantify how well we can estimate the target values. Such interval estimates could make our forecasts more meaningful. One way to estimate PIs in the NN setting is with bootstrapping. This involves drawing many bootstrap samples by repeatedly randomly sampling with replacement the original dataset to approximate the distribution of the statistic of interest. Efron et al. [13] introduced the idea behind bootstrapping to estimate the variance of estimators. The variance of the independent estimates of a parameter is assumed to be a good estimate of the true variance of the estimator of that parameter.

Bootstrap aggregation or bagging (Breiman [44], Hall and Samworth [45], Steele [46]) is an ensemble technique for reducing the variance of estimated predictions. Bagging appears to work especially well for low-bias and high-variance statistics (Hastie et al. [47]). Bagging is performed by taking B bootstrap samples from the training dataset $\boldsymbol{Z} = (X, \boldsymbol{y})$, retraining the prediction function, predicting the target $\boldsymbol{y}$ from each bootstrap sample, and then averaging the bootstrap predictions.

Random forests (Breiman [48]) is a significant modification of bagging. Recall that the idea of bagging is to average many noisy but roughly unbiased model predictions thereby reducing the variance. In the bootstrapping method, we only take random samples from the original data; however, in random forests, we take random samples not only from the data, but also from the predictor variables. A random forest is created by taking B samples of the same size from the training dataset $Z = (X, \boldsymbol{y})$, selecting q variables at random from the p existing variables, making a prediction from each bootstrap data set, and averaging the predictions.

We translated this idea to the NN prediction setting. Specifically, we extended the bagging mechanism to introduce a greater degree of variation from the bootstrap-to-bootstrap NNs. We used the random forest idea of drawing random samples from the available features (i.e., predictor variables). We refer to this new approach as "*extended bagging"* (E-Bagging). In brief, we take random samples not only from the original observations, but also from the p available features in the dataset.

3.8.2.1 Extended Bagging Neural Networks

We consider neural networks for time series forecasting. It is assumed that target values y_i's can be modeled by

$$y_i = f(\boldsymbol{Z}; [W_{\boldsymbol{f}} \quad \boldsymbol{b_f}]) + \epsilon_i;\ i = 1, \ldots, n, \tag{3-28}$$

where $f(\boldsymbol{Z}; [W_{\boldsymbol{f}} \quad \boldsymbol{b_f}])$ is the neural network model in which $W_{\boldsymbol{f}}$ and $\boldsymbol{b_f}$ are the weight matrix and bias vector of the model, respectively. Also, ϵ_i denotes noise (error) with a zero-mean. It is also assumed that the errors are independent and identically distributed. In practice, an estimate of the model is obtained using some estimator $\hat{y}_i = \hat{f}\left(\boldsymbol{Z}; \left[\widehat{W}_{\hat{\boldsymbol{f}}} \quad \widehat{\boldsymbol{b}}_{\hat{\boldsymbol{f}}}\right]\right)$. Thus, we have

$$y_i - \hat{y}_i = \left[f(\boldsymbol{Z}; [W_{\boldsymbol{f}} \quad \boldsymbol{b_f}]) - \hat{f}\left(\boldsymbol{Z}; \left[\widehat{W}_{\hat{\boldsymbol{f}}} \quad \widehat{\boldsymbol{b}}_{\hat{\boldsymbol{f}}}\right]\right)\right] + \epsilon_i. \tag{3-29}$$

Prediction bands try to quantify the uncertainty associated with the difference between the target values (y_i's) and the predicted values ($\hat{y}_i$'s). If the two terms on the right-hand side of (3-29) are statistically independent, the total variance associated with the NN model outputs is given by the sum of the model misspecification variance denoted by $\sigma_{\hat{y}}^2$ and the noise variance denoted by $\sigma_{\hat{\epsilon}}^2$ and can be calculated as follows:

$$\sigma_{p_i}^2 = \sigma_{\hat{y}_i}^2 + \sigma_{\hat{\epsilon}_i}^2, \tag{3-30}$$

Upon proper estimation of these variances, prediction intervals can be constructed for the NN outputs. The E-Bagging technique can be used to estimate the first component, $\sigma_{\hat{y}}^2$. Maximum likelihood techniques can be used to estimate the second component, $\sigma_{\hat{\epsilon}}^2$.

The extended bagging technique is carried out by taking B random samples of the same size as the original dataset with replacement from the training dataset $\boldsymbol{Z} = (\boldsymbol{R}, \boldsymbol{y})$, consisting of a 3-dimensional rolling window array $\boldsymbol{R}$ defined in Equation (3-23) and an output vector $\boldsymbol{y}$ defined in Equation (3-24), selecting q features ($1 \leq q \leq p$) at random with replacement from the set of all possible unique combinations up to size p of all available features in the dataset, training a separate NN with the same hyperparameters on each resample, making predictions for h days ahead from each resample $\hat{\boldsymbol{y}}^{*b} = (\hat{y}_1, \dots, \hat{y}_n)^{*b}$, and averaging the predictions. If B resamples

$\boldsymbol{Z}^{*1}$, ..., and $\boldsymbol{Z}^{*B}$ are drawn and used to compute $\widehat{\boldsymbol{y}}^{*b} = f(\boldsymbol{R}|\boldsymbol{Z}^{*b})$, then the E-Bagging estimator of $\boldsymbol{y}$ is as follows:

$$\widehat{\boldsymbol{y}}^{*} = \frac{1}{B}\sum_{b=1}^{B}\widehat{\boldsymbol{y}}^{*b} = \frac{1}{B}\sum_{b=1}^{B} f(\boldsymbol{R}|\boldsymbol{Z}^{*b}), \tag{3-31}$$

where the superscript (*) indicates that the predictions has been computed by E-Bagging. To estimate prediction bands for the NN predictions, we must compute an estimate of the total variance σ_p^2 given in (3-30), (Heskes [18]). Assuming that the NN models are unbiased, the first component $\sigma_{\hat{y}_i}^2$ on the right-hand side of (3-30) can be estimated by the sample variance of B model outputs $\widehat{\boldsymbol{y}}^{*1}, \ldots, \widehat{\boldsymbol{y}}^{*B}$ as follows:

$$\widehat{\sigma}_{\widehat{\boldsymbol{y}}^{*}}^2 = \frac{1}{B}\sum_{b=1}^{B}(\widehat{\boldsymbol{y}}^{*b} - \widehat{\boldsymbol{y}}^{*})^2. \tag{3-32}$$

This variance is mainly due to the random initialization of network weights and the use of different datasets for training B neural network models. The remaining task is to estimate the noise variance $\sigma_{\hat{\boldsymbol{\epsilon}}}^2$ using maximum likelihood techniques. We assume that the ϵ_i's are normally distributed such that it suffices to compute their sample variance which may however depend on the input $\boldsymbol{R}$. Mathematically, from (3-30), this variance can be calculated as follows:

$$\sigma_{\hat{\boldsymbol{\epsilon}}}^2 \cong E[(\boldsymbol{y} - \widehat{\boldsymbol{y}}^{*})^2] - \widehat{\sigma}_{\widehat{\boldsymbol{y}}^{*}}^2, \tag{3-33}$$

where $\widehat{\boldsymbol{y}}^{*}$ and $\widehat{\sigma}_{\widehat{\boldsymbol{y}}^{*}}^2$ are obtained from (3-31) and (3-32), respectively. Thus, based on this equation, the squared residuals denoted by r_i^2 can be calculated as follows:

$$r_i^2 = max\left\{(y_i - \hat{y}_i^{*})^2 - \hat{\sigma}_{\hat{y}_i^{*}}^2, 0\right\}, \tag{3-34}$$

where $\hat{y}_i^{*}$ and $\hat{\sigma}_{\hat{y}_i^{*}}^2$ are obtained from (3-31) and (3-32), respectively. These residuals are linked by their corresponding inputs $\boldsymbol{R}$ to form a new training set $\boldsymbol{Z}_{\boldsymbol{r}^2} = (\boldsymbol{R}, \boldsymbol{r}^2)$

where $\boldsymbol{r}^2 = \{r_i^2\}_{i=1}^n$. The variance $\sigma_{\hat{\epsilon}}^2$ can be approximated by training the following network referred to as the residual predictor NN,

$$r_i^2 = g(\boldsymbol{Z}_{r_i^2}; [W_{\boldsymbol{g}} \quad \boldsymbol{b_g}]) + \epsilon_{2i};\ i = 1, \ldots, n,$$

where $W_{\boldsymbol{g}}$ and $\boldsymbol{b_g}$ are the weight matrix and bias vector of the new NN, respectively. In this new NN, the squared residuals r_i^2's are used as the target values. The negative loglikelihood:

$$L = -\sum_{i=1}^{n} \log\left\{\frac{1}{\sqrt{2\pi\sigma_{\hat{\epsilon}_i}^2}} exp\left(-\frac{r_i^2}{2\sigma_{\hat{\epsilon}_i}^2}\right)\right\},$$

where $\log(.)$ is the natural logarithm, becomes the measure of error. In this case, the loss function that is being minimized in fitting is:

$$L = \frac{1}{2}\sum_{i=1}^{n}\left\{\log(\sigma_{\hat{\epsilon}_i}^2) + \frac{r_i^2}{\sigma_{\hat{\epsilon}_i}^2}\right\}, \tag{3-35}$$

and then an estimate of the noise variance would be:

$$\hat{\sigma}_{\hat{\boldsymbol{\epsilon}}}^2 \approx \hat{g}(\boldsymbol{Z}_{\boldsymbol{r}^2}; [\widehat{W}_{\hat{\boldsymbol{g}}} \quad \widehat{\boldsymbol{b}}_{\hat{\boldsymbol{g}}}]). \tag{3-36}$$

The activation function of the residual predictor NN output layer is selected to be the exponential function, forcing a positive value for all predicted noise variances. The minimization of the new loss function can be done using a variety of methods, including gradient descent algorithms. To construct prediction intervals, we totally need to train $(B + 1)$ NN models. Then we can construct the following approximate $100(1 - \alpha)\%$ prediction interval:

$$\boldsymbol{PI} = \left(\hat{\boldsymbol{y}}^* - z^*_{\left(1-\frac{\alpha}{2}\right)}\sqrt{\hat{\sigma}_{\hat{\boldsymbol{y}}^*}^2 + \hat{\sigma}_{\hat{\boldsymbol{\epsilon}}}^2},\ \hat{\boldsymbol{y}}^* + z^*_{\left(1-\frac{\alpha}{2}\right)}\sqrt{\hat{\sigma}_{\hat{\boldsymbol{y}}^*}^2 + \hat{\sigma}_{\hat{\boldsymbol{\epsilon}}}^2}\right), \tag{3-37}$$

where $z^*_{\left(1-\frac{\alpha}{2}\right)}$ is the critical z-value, corresponding to the $\left(1 - \frac{\alpha}{2}\right)$ cutoff of the standard normal distribution.

3.8.3 Hyper-Parameter Tuning

Hyperparameters are parameters that define network architecture and are not model parameters (weights). Hence, they define how the network is structured and cannot be directly obtained from the training data. Model weights are learned during training when we optimize a loss function using the backpropagation learning algorithm. These hyperparameters might address model design concerns such as the best number of nodes in a hidden layer, the best rate of dropout, the best optimizer and learning rate, or the best number of epochs or batch size. A batch size refers to the number of training examples utilized in one iteration.

The number of nodes affects the network's learning capacity. In general, more nodes help a network learn more structure from the problem which makes the training time longer. Moreover, more learning capacity creates the problem of potentially overfitting the training data. Dropout regularization addresses this problem. Activation functions are used to add nonlinearity to models, which allows models to learn nonlinear prediction boundaries and make multilayer NNs more powerful.

In order to tune network architecture, we generally resort to experimentation to figure out what works best. Here, we used a sophisticated technique called error analysis to optimize hyperparameters. Briefly, we analyzed the resulting errors from fitting a model by computing MAEs at different times for each of the different combinations of NN hyperparameters.

The first half of the data is used as the first and larger training dataset, T_1, to feed into the NN model for training. The other half of the observations is split into U consecutive and disjoint sets $(S_1, \dots, S_U)$ such that each of them contains h observations. The first set S_1 is used as the first test set to evaluate the first fitted model by computing the MAE. Next, the set S_1 is combined with the first training

dataset T_1 to build the second training dataset T_2. The NN model is updated on the set S_1. One benefit of using advanced adaptive learning rate optimization algorithms such as Adam when training is that a model would be capable of updating on new observations, instead of training the model on the entire dataset. This can reduce training time exponentially and so is computationally less expensive. The set S_2 is used as the second test set to evaluate the 2nd fitted model by computing the MAE.

We continue these procedures until all remaining sets $S_2, S_3, \ldots$, and S_{U-1} have been used to train the NN model. In more detail, at each step, a set S_u $(u = 2, \ldots U-1)$ is combined with the previous training set T_u to build the next training dataset T_{u+1}. The NN model is updated on this new set. The following set S_{u+1} is used as another test set to evaluate the updated training NN model using the MAE. Finally, to complete the overall evaluation of the NN model, we average all resulting MAEs. We repeatedly run the error analysis technique r times to obtain r overall MAEs for each of the different combinations of NN hyperparameters. We create a side-by-side boxplot to show the resulting MAE distributions for all combinations. The best set of hyperparameters is selected based on a tradeoff between the median MAEs and their variance. We now turn to the algorithm used to tune the model hyperparameters for training the one-hidden layer WNN.

3.8.3.1 Hyperparameter Tuning for WNN Architecture

We split the observations into the following disjoint sets. The first set is the first and largest training set which contains the first half of the observations and is denoted by $T_1 = X_{p\times m}$ (where $m = \left\lfloor \frac{n}{2} \right\rfloor$ is the largest integer less than $\frac{n}{2}$). The other half of the observations are split into U equivalent sets $(S_1, \ldots, S_U)$ such that each of them contains h observations. Note that m may not evenly divide h, so we do not consider those last observations to make all sets the same size.

Step 1:

The one-hidden layer WNN can be expressed as the composition of two mappings. We use the largest training dataset T_1 and its corresponding sample weights $\boldsymbol{v}_{m\times 1}$, with a tuning constant $\alpha = \frac{c}{m}$ where c is a positive integer, to feed into the WNN model to train. The first mapping sends the product of the $(p+1)\times m$ augmented input matrix $\begin{bmatrix} T_1 \\ \mathbf{1}_{1\times m} \end{bmatrix}$ with the $m \times 1$ exponential decay sample weight $\boldsymbol{v}$ to a $(p+1)$ vector of outputs $\boldsymbol{a}^{ih}$ by multiplying a $q \times (p+1)$ weight matrix $[W^{ih} \quad \boldsymbol{b}^{ih}]$ with the $(p+1)$-vector $(\begin{bmatrix} T_1 \\ \mathbf{1} \end{bmatrix} \boldsymbol{v}_{m\times 1})$ in the hidden layer. The transformation of the resulting q-vector $[W^{ih} \quad \boldsymbol{b}^{ih}] \left(\begin{bmatrix} T_1 \\ \mathbf{1} \end{bmatrix} \boldsymbol{v}_{m\times 1}\right)$ by the activation function ζ_{ih} is:

$$\boldsymbol{a}^{ih}_{q\times 1} = \zeta_{ih}\Bigg(\underbrace{\Bigg(\underbrace{[W^{ih} \quad \boldsymbol{b}^{ih}]}_{q\times(p+1)} \underbrace{\Bigg(\underbrace{\begin{bmatrix} T_1 \\ \mathbf{1} \end{bmatrix}}_{(p+1)\times m} \underbrace{\boldsymbol{v}}_{m\times 1}\Bigg)}_{(p+1)\times 1}\Bigg)}_{q\times 1}\Bigg).$$

The second mapping is the same as the second mapping for the WNN discussed in Section (3.8.1); hence,

$$\widehat{\boldsymbol{y}}_{s\times 1} = \zeta_{ho}\Bigg(\underbrace{\Bigg(\underbrace{[W^{ho} \quad \boldsymbol{b}^{ho}]}_{s\times(q+1)} \underbrace{\begin{bmatrix} \boldsymbol{a}^{ih}_{q\times 1} \\ 1 \end{bmatrix}}_{(q+1)\times 1}\Bigg)}_{s\times 1}\Bigg).$$

We now optimize the WNN model using the first training dataset T_1 by determining optimal weights and biases matrices $[W^{ih} \quad \boldsymbol{b}^{ih}]$ and $[W^{ho} \quad \boldsymbol{b}^{ho}]$ through minimization of the mean squared error (MSE) loss function over the training epochs using gradient descent (Section 3.4).

The set S_1 with their sample weights is used as the first test set to evaluate the fitted model by computing the following MAE:

$$MAE_1 = \frac{1}{h}\sum_{i=m+1}^{m+h} |y_i - \hat{y}_i|,$$

where the subscript "1" indicates that it is computed for the first test set. We then need to update our training model on the set S_1 as explained in the next step.

Step 2:

In the second step, the set S_1 which contains the observations $[\boldsymbol{x}_{.m+1}, \ldots, \boldsymbol{x}_{.m+h}]$ (where $\boldsymbol{x}_{.j}$ is the j^{th} column of X) will be added to the previous training set T_1 to build the second training set $T_2 = X_{p\times(m+h)}$. Their sample weights $\boldsymbol{v}_{(m+h)\times 1}$ will be recomputed based on the new length, with a new tuning constant $\alpha = \frac{c}{m+h}$ where c is a positive integer. The WNN model will be updated on this new set S_1 using its new weights. This updating can be expressed as the composition of two mappings. The first mapping sends the product of a $(p+1)\times h$ augmented input matrix $\begin{bmatrix} S_1 \\ \mathbf{1} \end{bmatrix}$ with an $h \times 1$ exponential decay sample weight $\boldsymbol{v}$ to a $(p+1)$-vector of outputs $\boldsymbol{a}^{ih}$ by multiplying a $q \times (p+1)$ weight matrix $[W^{ih} \quad \boldsymbol{b}^{ih}]$ with a $(p+1)$-vector $\left(\begin{bmatrix} S_1 \\ \mathbf{1} \end{bmatrix}\boldsymbol{v}\right)$ in the hidden layer. Additionally, a function ζ_{ih} is applied to this product:

$$\boldsymbol{a}^{ih}_{q\times 1} = \zeta_{ih}\left(\underbrace{[W^{ih} \quad \boldsymbol{b}^{ih}]}_{q\times(p+1)}\left(\underbrace{\begin{bmatrix} S_1 \\ \mathbf{1} \end{bmatrix}}_{(p+1)\times h} \underbrace{\boldsymbol{v}}_{h\times 1}\right)\right).$$

The second mapping is the same with the step 1. We now optimize the updated model by determining optimal weights and biases matrices $[W^{ih} \quad \boldsymbol{b}^{ih}]$ and $[W^{ho} \quad \boldsymbol{b}^{ho}]$ through minimization of the MSE loss measuring the accuracy of the

predictive function over the training epochs. The set S_2 with its sample weights is used as the second test set to evaluate the fitted model by again computing the MAE:

$$MAE_2 = \frac{1}{h}\sum_{i=m+h+1}^{m+2h} |y_i - \hat{y}_i|,$$

Step 3:

We repeat step 2 with the new set $S_u = \left[\boldsymbol{x}_{.(m+(u-1)h+1)}, \ldots, \boldsymbol{x}_{.(m+uh)}\right]$ for $u = 2, \ldots, U$ where $U = \left\lfloor \frac{m}{h} \right\rfloor$ until all these small sets except the last one S_U are fed into the model for training. Note that the last set, S_U, is only for evaluation.

In other words, the set S_u for $u = 2, \ldots, U-1$ is added to the previous training set $T_u = X_{p\times(m+(u-1)h)}$ with their sample weights $\boldsymbol{v}_{(m+uh)\times 1}$ recomputed based on the new length, and a tuning constant $\alpha = \frac{c}{m+uh}$ where c is a positive integer. The model is updated on the set S_u for $u = 2, \ldots, U-1$ using their new sample weights. A hidden layer outputs $\boldsymbol{a}^{ih}$, gives network outputs $\hat{\boldsymbol{y}}$ which has the same form as previous steps and computes the MAE as specified in the following forms:

$$\boldsymbol{a}^{ih}_{q\times 1} = \zeta_{ih}\Bigg(\underbrace{\underbrace{[W^{ih} \quad \boldsymbol{b}^{ih}]}_{q\times(p+1)}\Bigg(\underbrace{\begin{bmatrix} S_u \\ \underbrace{\mathbf{1}}_{(p+1)\times h} \end{bmatrix}}_{(p+1)\times 1} \underbrace{\boldsymbol{v}}_{h\times 1}\Bigg)}_{q\times 1}\Bigg),$$

$$MAE_u = \frac{1}{h}\sum_{i=m+(u-1)h+1}^{m+uh} |y_i - \hat{y}_i|, u = 1, \ldots, U. \qquad (3\text{-}38)$$

Step 4:

Finally, to compute the overall evaluation (MAE) of the model, we average the resulting MAEs at the different stages of updating the WNN model:

$$MAE_{\text{overall}} = \frac{1}{U}\sum_{u=1}^{U} MAE_u. \tag{3-39}$$

We repeatedly run these steps r times for each of the different combinations of WNN hyperparameters and create a boxplot to show the resulting MAE distribution. The best set of hyperparameters is selected based on a tradeoff between the median MAEs and their variance.

4 Results

The model fits explored in this chapter are performed using open source libraries such as NumPy (Oliphant [49]), Pandas (McKinney et al. [50]), and Keras (Chollet et al. [51]) with the TensorFlow backend (Abadi et al. [52]) in the Python programming language (Van Rossum [53]). Models based on GRUs in Section 4.1 and WNNs in Section (0) are used to learn the dynamic dependence structure present in the Covid-19 death counts data and also to map the learning sequence to produce future forecasts of the number of Covid-19 deaths in four particular US counties.

We used Covid-19 deaths data ranging from 1/27/2020 to 05/16/2021 (475 days) for training and data from May 02 to May 16, 2021for testing our prediction models. Hyper-parameter optimization of each of these models for four counties is performed, but only selection procedures of these hyperparameters for LA County (the most populous county) are explained in detail in Section (4.1.2.1) for the GRU and in Section (4.2.1.1) for the WNN. In all these models, the Adam optimizer is used for optimization with the mean squared error (MSE) loss function.

The errors of these NN models are evaluated on the test datasets and based on this evaluation, the best model with the smallest MAE is selected as the prediction model for Covid-19 deaths.

4.1 Fitting Models Based on the GRU

In this section, we report the process of predicting the number of daily Covid-19 deaths forecasted $h = 14$ days ahead using GRU models in the following counties:

LA County, CA; Cook County, IL; Harris County, TX; and NY County, NY. For each county of interest, the target prediction of daily Covid-19 deaths is obtained using the procedure in Section (3.7.1.3).

4.1.1 Data preprocessing

A full 7-day moving average of Covid-19 deaths including 3 days before and 3 days after the date of the report and the reporting date itself is computed to obtain a smoothed estimate of the trend for one week. The smoothed daily number of Covid-19 deaths is normalized using the MinMaxScaler function from the Preprocessing package in the scikit-learn library [54].

The raw data are converted to a 3-dimensional rolling window array defined in Formula (3-23) to have a proper GRU input structure. The input for a GRU model is a 3-dimensional tuple (batch size, # sequences, # features), where the first dimension corresponds to the batch size, the second dimension is the number of sequences, and the last dimension corresponds to the number of features in our datasets. The first two dimensions are hyperparameters and were chosen for our data through tuning. We considered the tuple of (100, 28, 9 or 10) as the GRU input shape for each county.

4.1.2 GRU Architecture

A sequential model including a hidden GRU layer followed by a hidden dense layer was built and trained using Keras with the TensorFlow backend. In the next sections, we explain how GRU models are obtained, and we go further and explain the tuning of hyperparameters in detail for LA County only. Other counties also have similar architectures.

4.1.2.1 GRU Architecture for LA County

All predictions on the test dataset are retained and the mean absolute error (MAE) calculated to summarize the quality of the model. Note that the MAE has the same units as the forecast data. The idea is to compare the model configurations using MAE summary statistics over a large number of runs (10 runs) and see exactly which of the configurations perform better on average. We created boxplots to visually represent the MAE values for the 10 configurations. Tuning model hyperparameters is a tradeoff between the median MAE and the variability of the MAEs; an ideal result would have a small median MAE with low variability.

Since hyperparameter tuning is time consuming and there are a lot of them here to be tuned, it could last forever. To avoid this, we set the following hyperparameters in advance based on experience. The number of epochs was set to be 1000. In addition, the early stopping approach was used to monitor the loss to avoid overfitting when training NNs. Additionally, the number of epochs with no improvement (patience) in the early stopping approach was set to 50 based on experimentation. The patience is an argument name in the early stopping function.

We used a Tanh activation function on the output layer. The Adam optimizer was used to minimize the MSE loss function. The LA GRU model hyperparameters we tuned are the batch size, learning rate, number of GRU layer nodes, activation function on the first dense layer, dropout rates for both dropout layers, and number of dense layer nodes.

The initial LA model used a sequential model consisting of an input layer of 10 features and a one-layer GRU, followed by a dropout layer. A dense layer followed by a dropout layer was added. This also was followed by an output layer of a single

node with a Tanh activation function. The loss was minimized using the Adam optimizer.

We explored the effect of training this configuration for different batch sizes (100, 300), learning rates (0.001, 0.005), number of GRU layer nodes (20, 80), activation functions (Tanh, SELU) on the first dense layer, dropout rates (0.1, 0.5) for both dropout layers, and number of dense layer nodes (20, 80). We used the error analysis method and repeated this training regimen 10 times and calculating the overall MAE given in Equation (3-39) for each configuration on the 17 test sets created from the second half of the observations. Boxplots of the overall MAE distributions on these test sets after each training with different hyperparameter combinations are shown in Figure 17 - Figure 20.

The overall MAE values range from 21.5 to 32 deaths. Comparing these boxplots shows that the choice of setting the batch size to 100, learning rate to 0.005, number of GRU layer nodes to 80, activation functions to Tanh on the first dense layers, dropout rate to 0.5 for the first dropout layer, dropout rate to 0.1 for the second dropout layer, and number of dense layer nodes to 20 offers the best performance on average in terms of both median MAE and variability. Also, we tuned the rolling window size (m) as a hyperparameter and found that the choice of setting m to 28 offers the best performance on average (graphs are not shown).

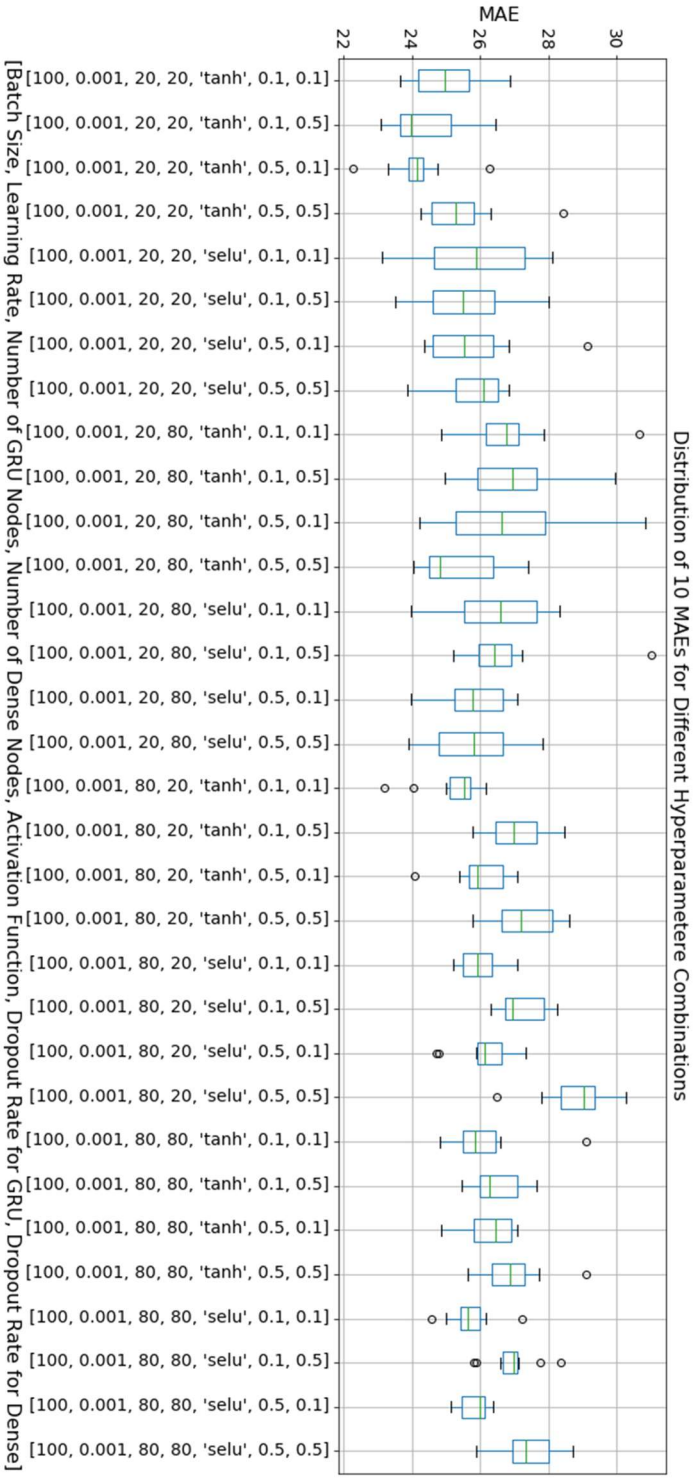


Figure 17: Diagnostic results with different hyperparameter combinations when batch size is 100 and learning rate is 0.001 in the LA model; in total, 320 models were fit to produce this boxplot.

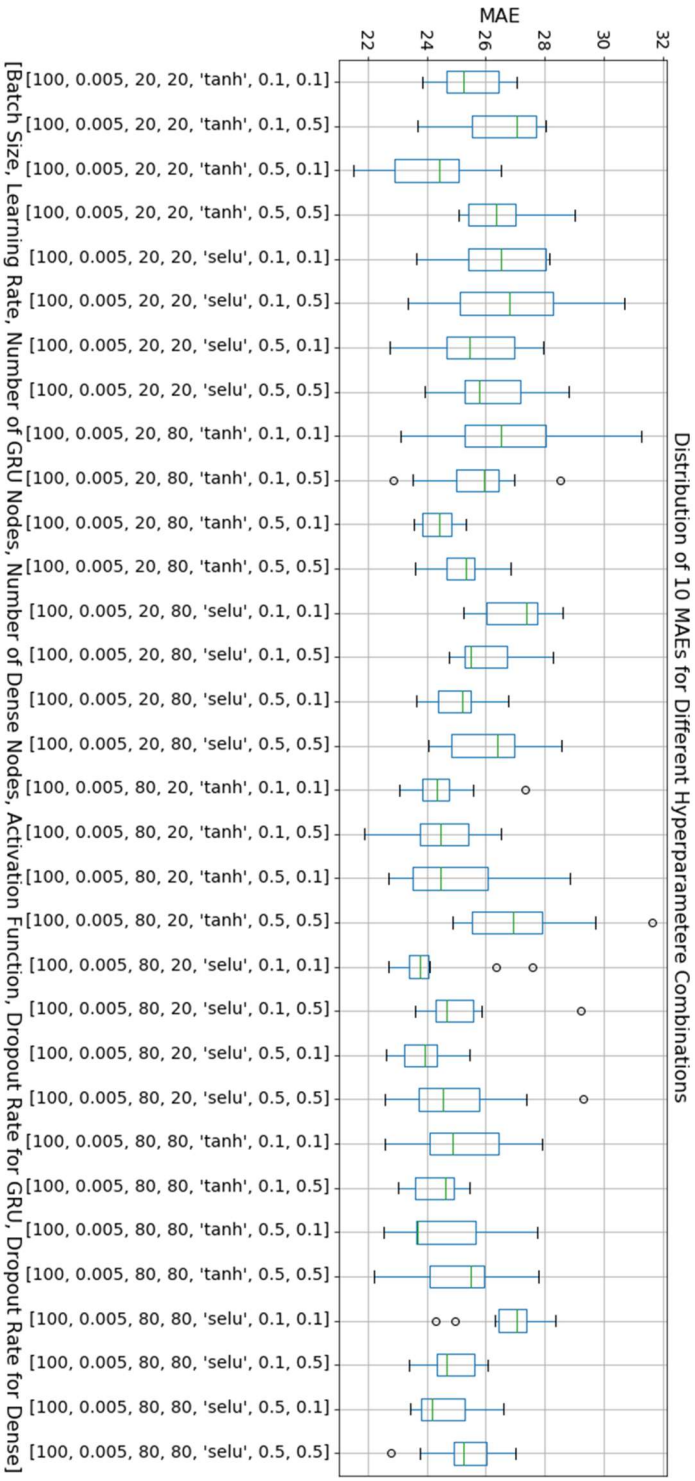


Figure 18: Diagnostic results with different hyperparameter combinations when batch size is 100 and learning rate is 0.005 in the LA model; in total, 320 models were fit to produce this boxplot.

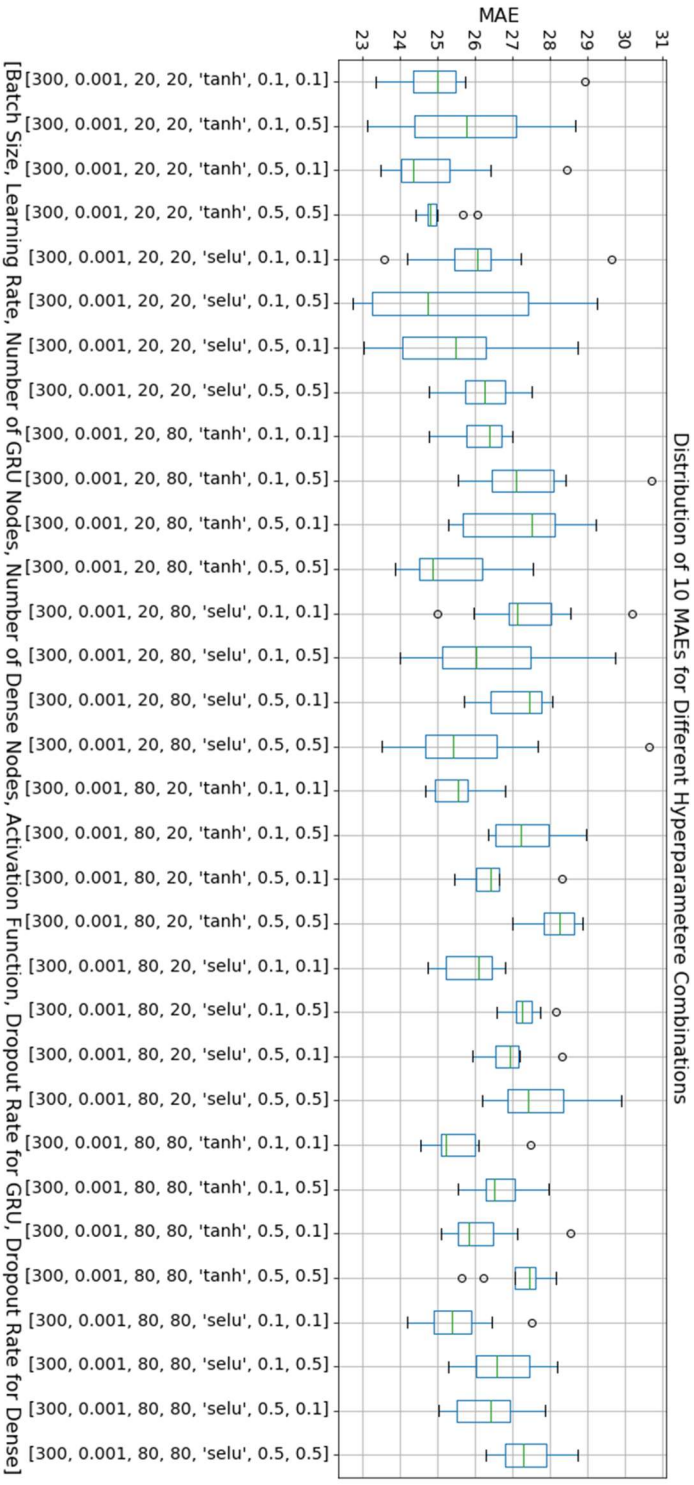


Figure 19: Diagnostic results with different hyperparameter combinations when batch size is 300 and learning rate is 0.001 in the LA model; in total, 320 models were fit to produce this boxplot.

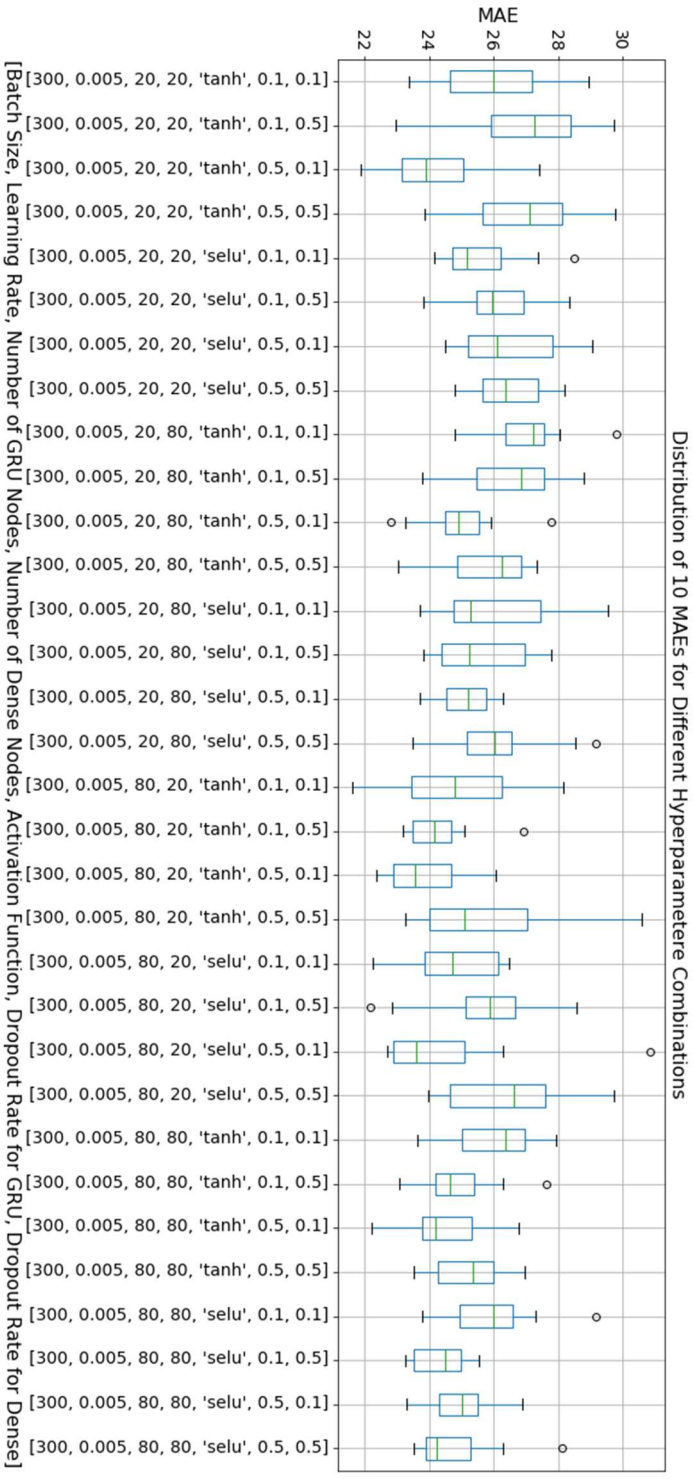


Figure 20: Diagnostic results with different hyperparameter combinations when batch size is 300 and learning rate is 0.005 in the LA model; in total, 320 models were fit to produce this boxplot.

4.1.2.1.1 Visualizing the Error Analysis for the best LA GRU Model

To visualize how the error analysis technique was used on different test sets through tuning the hyperparameters for the LA GRU model for computing each MAE value in the boxplots, we plotted all two-weeks forecasted predictions using the near optimal hyperparameters for the LA GRU model in Figure 21. The blue curve represents the smoothed daily Covid-19 confirmed deaths and the red curve shows the 14-days-ahead forecasted daily Covid-19 deaths using the best GRU model for different test sets separated by vertical black lines from September 21, 2020, to May 03, 2021, for LA County. Their Corresponding MAEs at different times are reported below the curves.

To create Figure 21, the first half of the Covid-19 death counts for LA County was used as the first and larger training dataset, T_1, up to September 21, 2020, to feed into the best LA GRU model for training. The other half of the observations was split into $U = 16$ equivalent sets $(S_1, \dots, S_{16})$ such that each of them contains $h = 14$ observations. The 2-weeks-ahead forecasted daily Covid-19 deaths for the first set S_1 from September 21, 2020, to October 05, 2020, resulted in an MAE of 12.9. Next, the best LA GRU model is updated on this new set S_1 and the 2-weeks-ahead forecasted daily Covid-19 deaths for the second set S_2 between October 05-19, 2020, resulted in an MAE of 3.5. We continue these procedures until all remaining sets S_2, S_3, …, and S_{15} were used to train the best LA GRU model. The MAE values for different sets are reported below the curves. Lastly, the best LA GRU model is updated on the set S_{15} and the 2-weeks-ahead forecasted daily Covid-19 deaths for the last set S_{16} from April 19 to May 03, 2020, resulted in an MAE of 4.8. To find the overall MAE of the best LA GRU model, we averaged all 16 resulting MAE values which is 25.3.

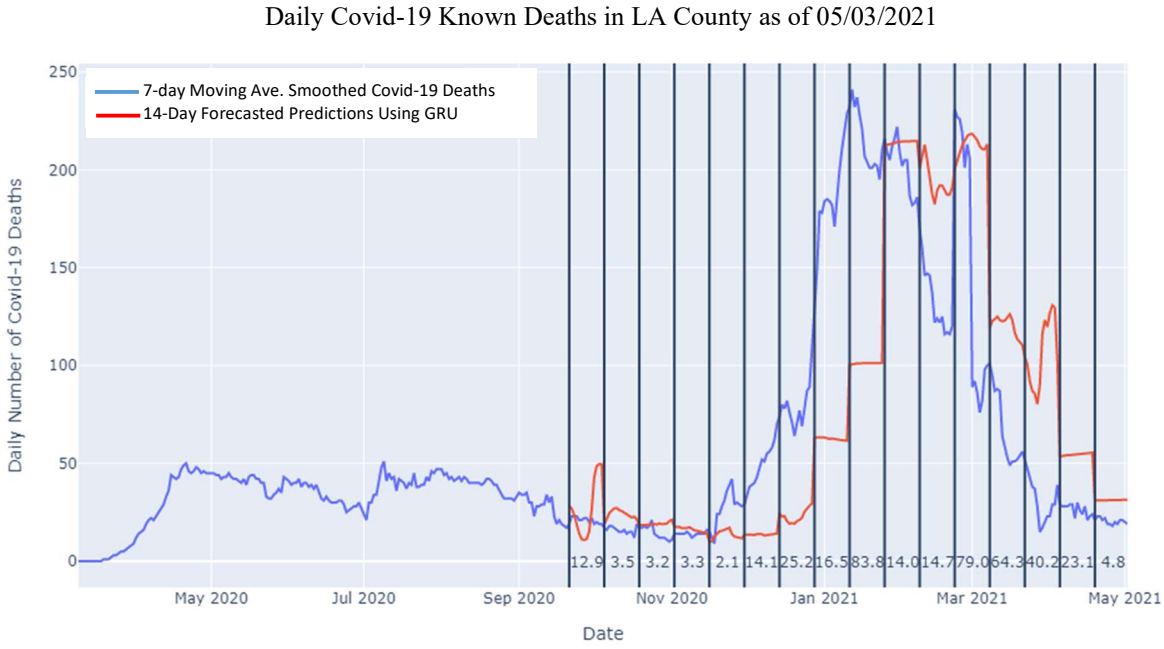


Figure 21: the smoothed daily Covid-19 confirmed deaths (blue curve) and forecasted daily deaths using GRU (red curve) for different test sets as of 05/03/2021 for LA County.

4.1.2.1.2 Predictions for LA County

According to the hyperparameter tuning in the LA GRU model, the best LA GRU architecture was obtained using a sequential model consisting of an input layer containing 10 features and a hidden GRU layer of 80 nodes. It is immediately followed by a dropout layer to prevent the model from overfitting with a dropout rate of 0.5 which was applied to the non-recurrent connections. Following the dropout layer, we added a dense layer of 20 nodes with a Tanh activation function, followed by a dropout layer with a dropout rate of 0.1. Lastly, this was followed by an output layer of a single node. The activation function on the output layer was a Tanh. The MSE loss function was minimized using the Adam optimizer. The learning rate was set to 0.005. The batch size and the number of epochs were 100

and 1000, respectively. The early stopping method was used to monitor the loss with patience of 50.

The number of trainable parameters in the WNN layer as given in Formula (3-21) for LA County was $g[u(u+i)+u] = 3[80(80+11)+80] = 22{,}080$. Since the numbers of trainable parameters in the first and second dense layers were 1620 and 21, respectively, the total number of trainable parameters is thus $22{,}080 + 1620 + 21 = 23{,}721$.

The smoothed daily Covid-19 confirmed deaths (blue curve) and the 14-days-ahead predicted daily Covid-19 deaths using the GRU model (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 for LA County are shown in Figure 22 (top graph). The vertical black line separates the training and test sets. The red curve in all plots represents 2-week ahead forecasts only for the test set period from May 02 to May 16, 2021, and to make all predictions prior to May 01, 2021, we used all the training data. The 14-days-ahead predicted daily Covid-19 deaths for the training set (to the left of the vertical black line) resulted in an MAE of 7.363, while for the test set (to the right of the vertical black line) resulted in an MAE of 9.343.

To see how well the best LA GRU model has done it suffices to compute the cumulative Covid-19 deaths of the daily 14-days-ahead predictions. We then added them to the end of the smoothed cumulative deaths and plotted them. The cumulative counts are shown in Figure 22 (bottom graph) and the model predictions are only plotted for the test set from May 02, 2021, to May 16, 2021 (to have a closer look into the test data, see the box on the graph).

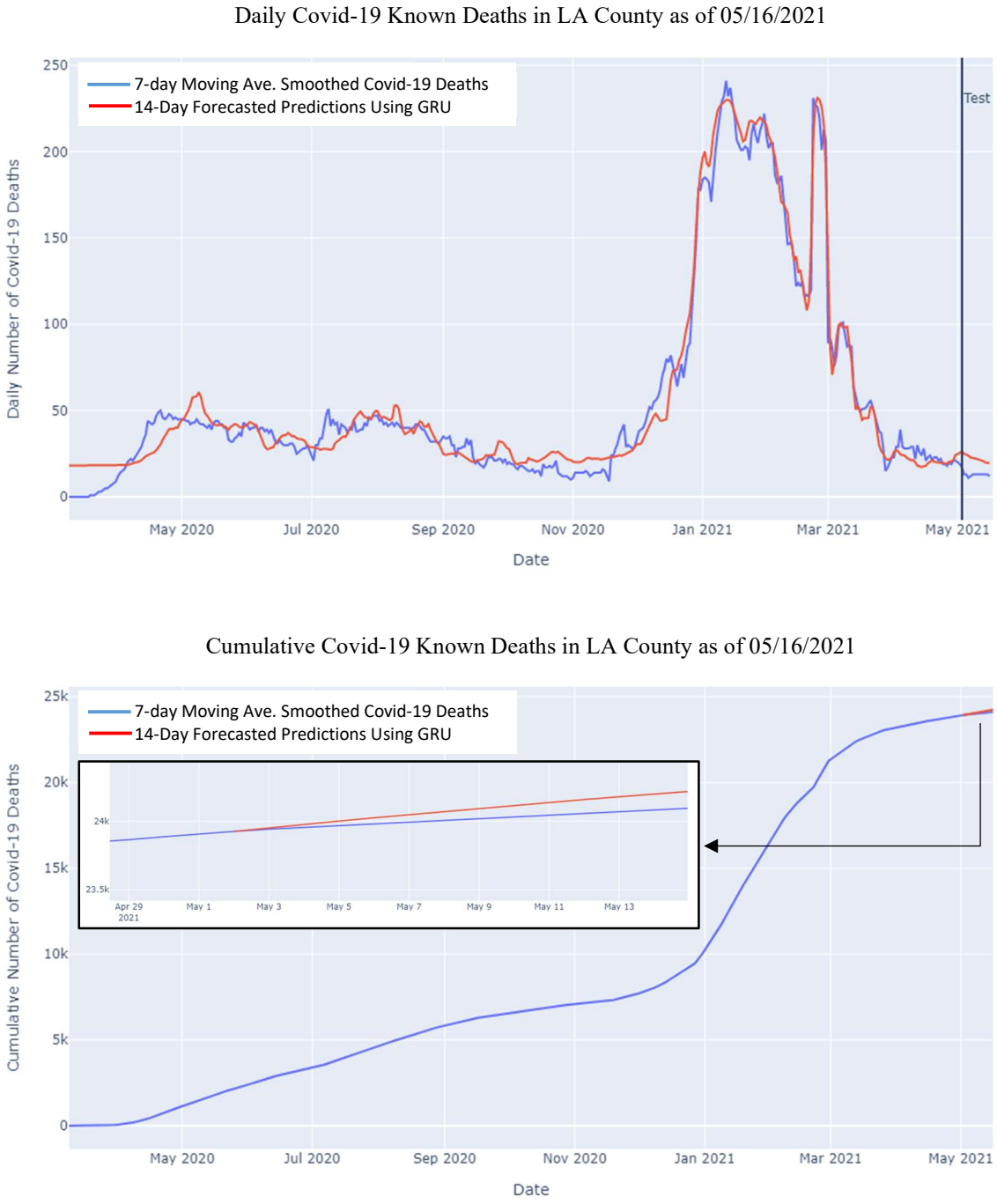


Figure 22: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for LA County. (Bottom) cumulative counts. The box zooms in on the test set.

4.1.2.2 GRU Model Predictions for Other Counties

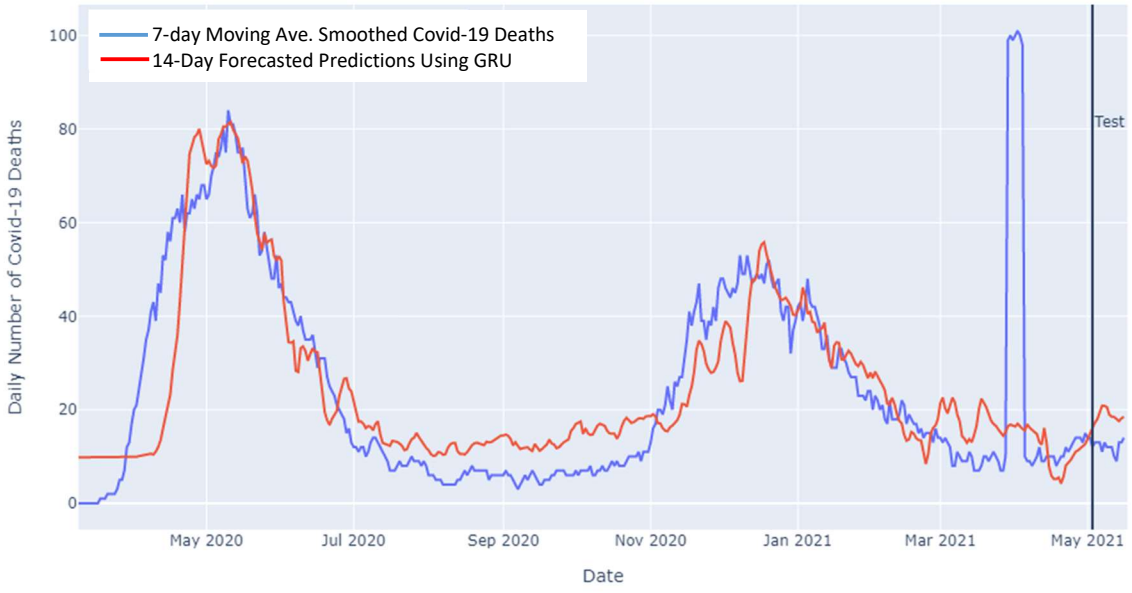
This section presents the three sets of GRU model predictions for the following counties: Cook County, IL; Harris County, TX; and NY County, NY. We considered their architectures and hyperparameters in the same way as with the best LA GRU model. The smoothed daily Covid-19 confirmed deaths (blue curve) and the 14-days-ahead predicted daily Covid-19 deaths using the GRU model (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 for Cook County are shown in Figure 23 (top graph); for Harris County in Figure 24 (top graph); and for NY County in Figure 25 (top graph). The vertical black line in each figure separates the training and test sets. The MAEs of the GRU models for the different counties on the training and test sets are reported in Table 1.

The cumulative counts for Cook County are shown in Figure 23 (bottom graph) and the model predictions are only plotted for the test set from May 02, 2021 to May 16, 2021; for Harris County in Figure 24 (bottom graph); and for NY County in Figure 25 (bottom graph). The fit was not good in NY County.

Table 1: MAE of different models for each county on the training and test sets

	Train MAE		Test MAE		CP (%)
	GRU	E-Bagging	GRU	E-Bagging	
LA County	7.363	4.155	9.343	5.884	99.8
Cook County	8.373	5.023	6.528	1.073	98.6
Harris County	4.547	2.018	4.904	2.386	100
NY County	5.099	4.466	2.575	1.277	97.9
Average	6.346	3.916	5.838	2.655	99.1

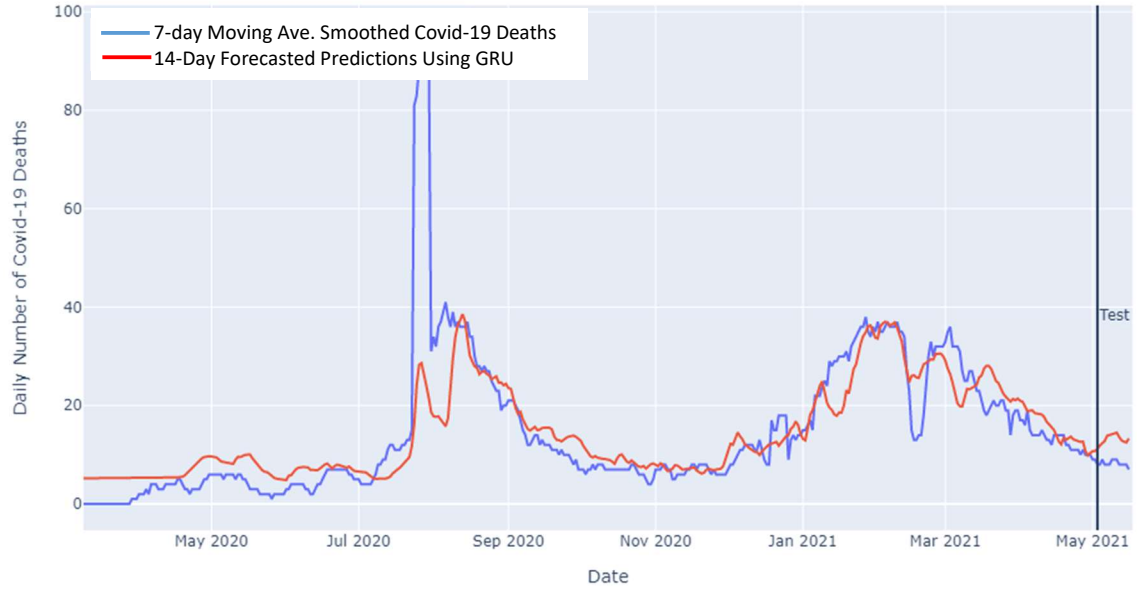
Daily Covid-19 Known Deaths in Cook County as of 05/16/2021



Cumulative Covid-19 Known Deaths in Cook County as of 05/16/2021

Figure 23: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Cook County. (Bottom) cumulative counts. The box zooms in on the test set.

Daily Covid-19 Known Deaths in Harris County as of 05/16/2021



Cumulative Covid-19 Known Deaths in Harris County as of 05/16/2021

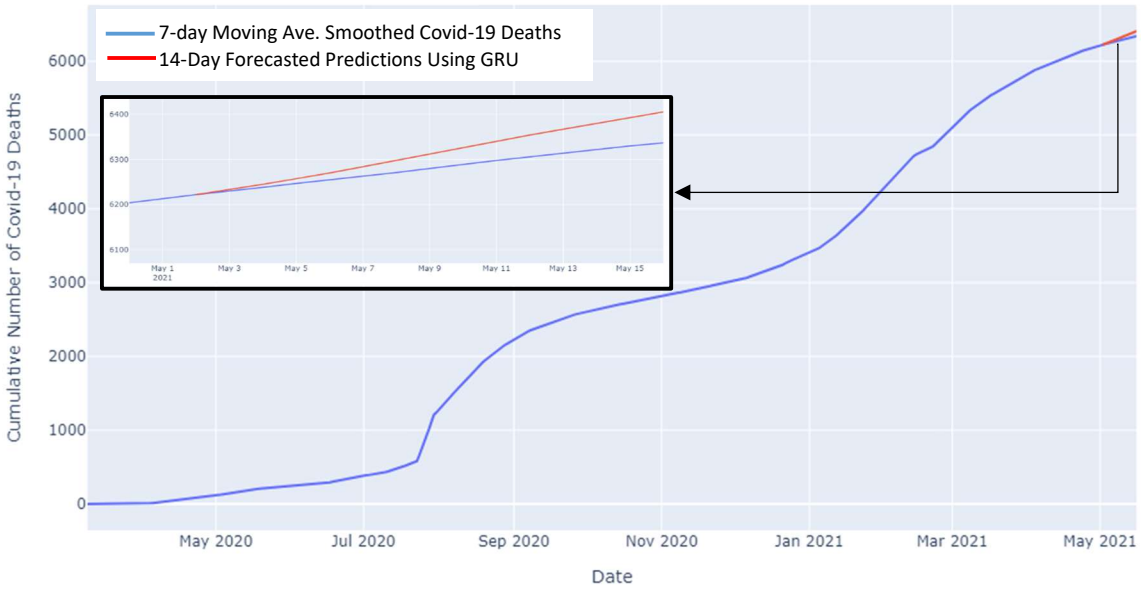
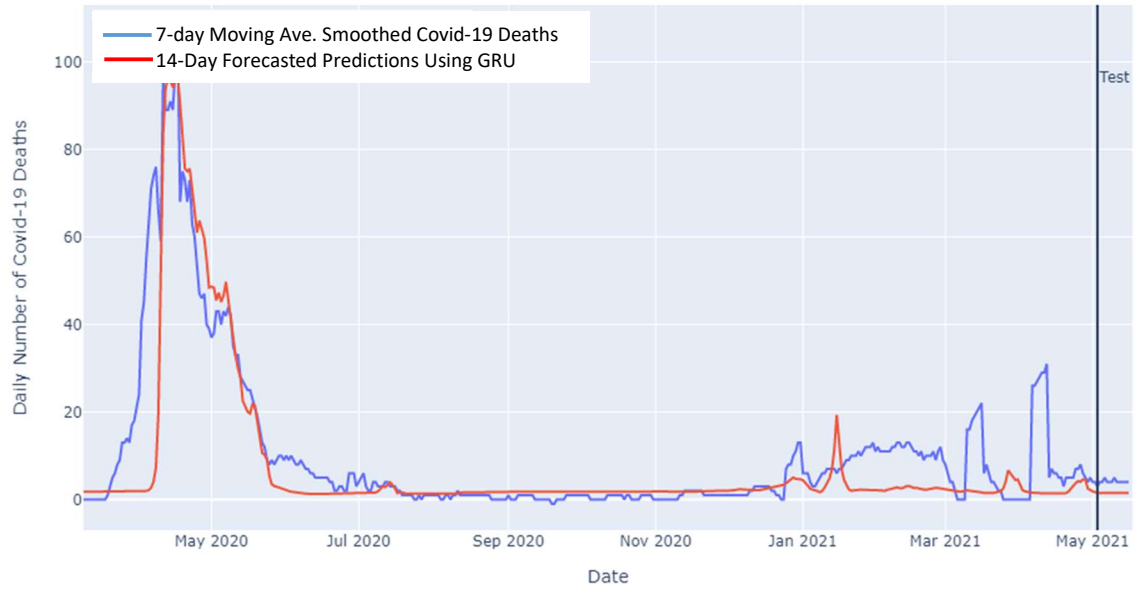


Figure 24: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Harris County. (Bottom) cumulative counts. The box zooms in on the test set.

Daily Covid-19 Known Deaths in NY County as of 05/16/2021



Cumulative Covid-19 Known Deaths in NY County as of 05/16/2021

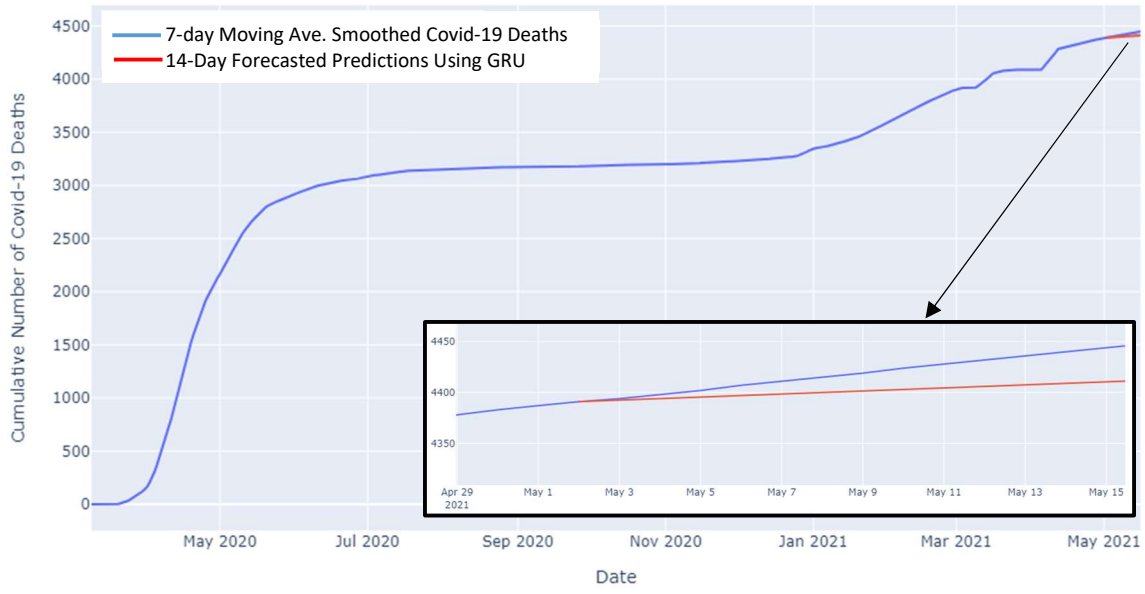


Figure 25: (Top) smoothed daily Covid-19 confirmed deaths (blue curve) and predicted daily deaths using GRU (red curve) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for NY County. (Bottom) cumulative counts. The box zooms in on the test set.

4.1.2.3 E-Bagging of GRU Models

This section includes the results of the extended bagging technique on the GRU models for different counties. After finding the GRU models for each county considered, we performed the E-Bagging technique on the GRU models to predict the Covid-19 death counts and construct 95% prediction bands for 14 days of forecasted values in each county. In the next sections, we explain how the GRU models were improved using the E-Bagging technique and for LA County only, tuning the size of E-Bagging samples as a hyperparameter for the E-Bagging technique is explained in detail.

4.1.2.3.1 E-Bagging of the Best LA GRU Model

To find the optimal size of extended bagging samples (B) for the best LA GRU model, we fit the same best model on each of B samples of sizes 32, 64, 128, and 512 and iterate 10 times. For each iteration, we average the B predictions to compute an overall prediction, defined in Equation (3-31); we compute the variance of the B predictions, defined in Equation (3-32); and, we compute the squared residuals, defined in Equation (3-34), to estimate the noise variance, defined in Equation (3-36). We fit a new NN on the squared residuals as new target variable values. The loss function that is being minimized in fitting the residual predictor NN is defined in Equation (3-35). Thus, we need to train $(B + 1)$ NN models for each iteration. Now, we can construct prediction bands for the overall prediction.

To find the best B, we calculated the MAE and coverage probability of the intervals, defined in Equation (3-15), and reported in Figure 26. Intuitively, the optimal B occurs when the model gives low MAE and coverage probability close to 95% on average.

The left boxplot in Figure 26 demonstrates how many E-Bagging samples are required to obtain a coverage probability for the prediction band close enough to the nominal 95% prediction level. The right boxplot shows how many E-Bagging samples are required to obtain a model with less error and higher accuracy.

Comparing boxplots illustrates that all MAE distributions are almost in the same range. All the E-Bagging sample sizes offer a coverage probability of our prediction bands exceeding the specified level of 95% in each case except one single case when $B = 256$. Papadopoulos et al. [55] in 2001 found that bootstrap methods consistently overestimate prediction band coverage probability. This is consistent with our results. It might be helpful to further optimize the residual predictor NN to construct prediction bands closer to the specified coverage level. Since there is no real difference among the choice of selection of B on average, we used a size of 128. This size of bootstrap samples will be used when employing the E-bagging method for other counties as well.

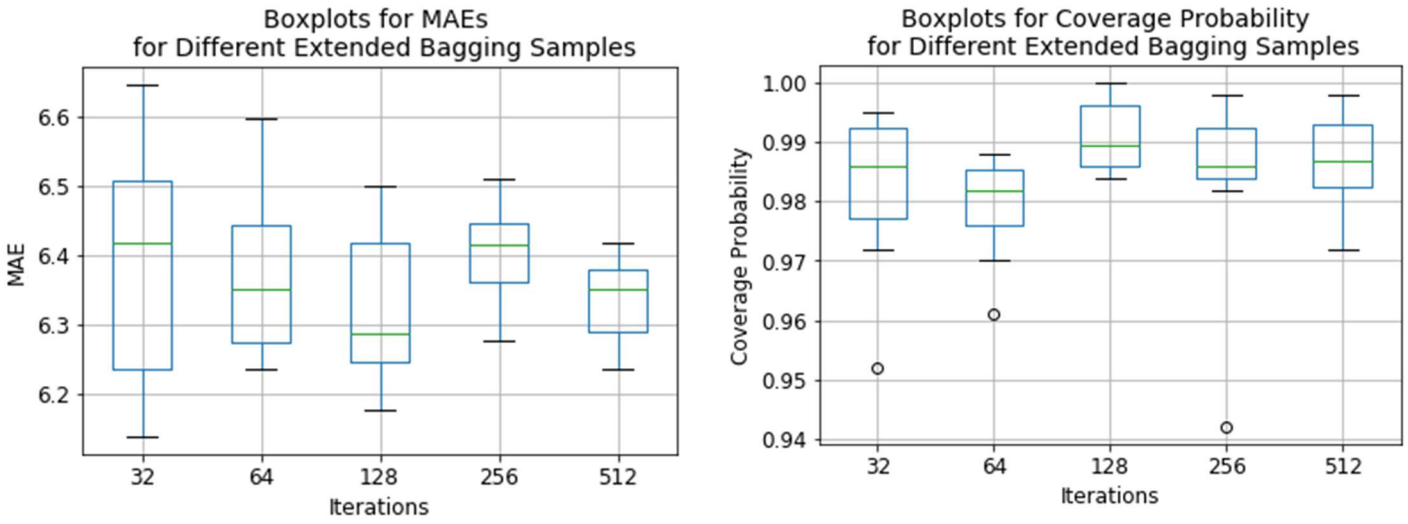


Figure 26: (Left) Coverage probability distribution, (Right) MAE distribution for different E-Bagging samples for LA County; in total, 9970 models were fit to produce each boxplot.

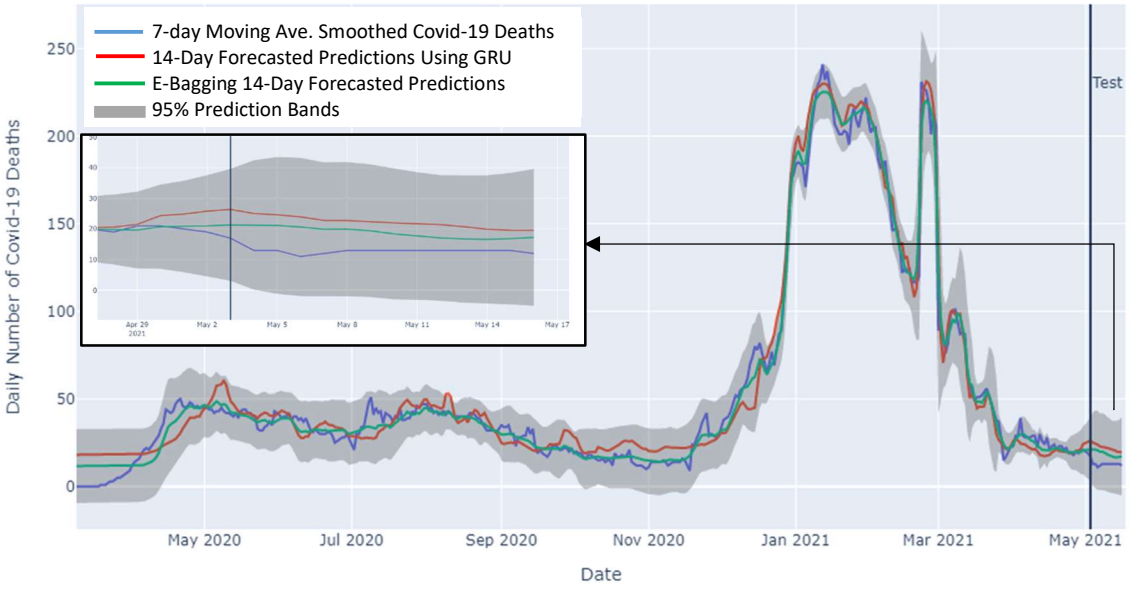
4.1.2.3.2 E-Bagging of the Best LA GRU Model

The smoothed daily Covid-19 confirmed deaths (blue curve), their corresponding 14-day forecasted predictions using the best LA GRU model (red curve), the 14-day forecasted predictions using E-Bagging of the best LA GRU model (green curve), and the corresponding 95% prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 for LA County are shown in Figure 27 (top graph). The vertical black line separates the training and test sets. In viewing these two graphs, the forecasted results using E-Bagging (green curve) are generally closer to the blue curve (with MAE values of 4.155 and 5.884 on the training and test sets, respectively) than the red curve (with MAE values of 7.363 and 9.343 on the training and test sets, respectively).

The prediction error distributions for the best LA GRU model (blue histogram) and for E-Bagging of the best LA GRU model (red histogram) are shown in Figure 28. While both distributions are centered at zero, the red histogram is more symmetric. This graph also shows that using the E-Bagging technique on the best LA GRU model yielded less variability in predictions. These results generally indicate that the E-Bagging technique has improved our predictions. Applying the E-Bagging technique to the best LA GRU model provides a coverage probability of 0.998 for the entire dataset, which is higher than what we specified.

The cumulative counts are shown in Figure 27 (bottom graph) and both model predictions and prediction bands are only plotted for the test set from May 02, 2021, to May 16, 2021 (especially see the black box). According to this figure, the 14-days-ahead forecasted cumulative Covid-19 deaths using E-Bagging with GRU model (green curve) is closer to the true values than the red curve.

Daily Covid-19 Known Deaths in LA County as of 05/16/2021



Cumulative Covid-19 Known Deaths in LA County as of 05/16/2021

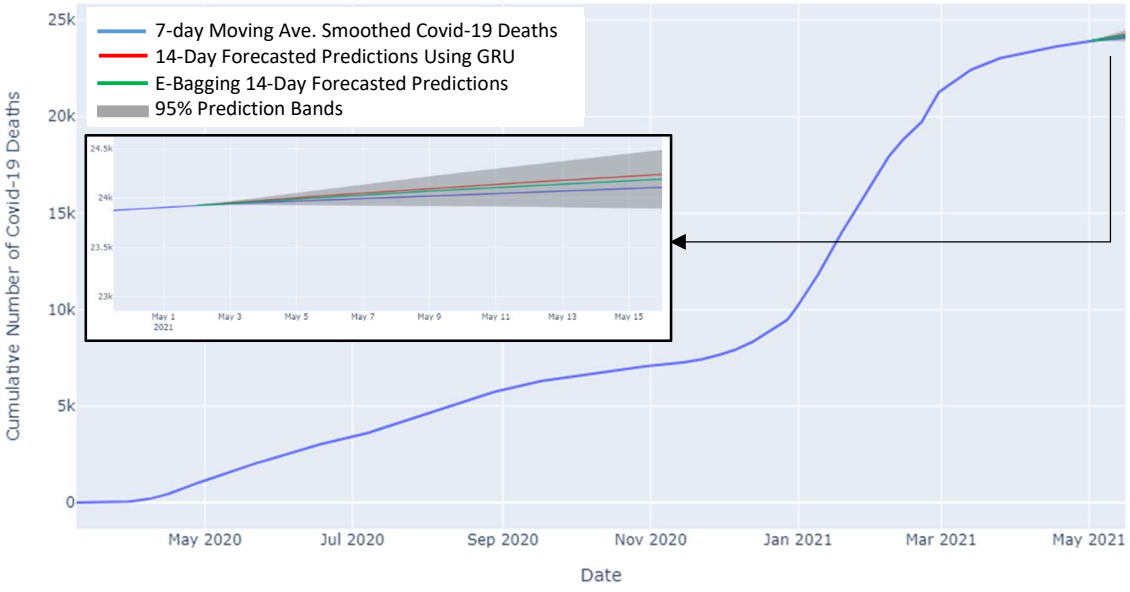


Figure 27: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for LA County. (Bottom) cumulative counts. The boxes zoom in on the test set.

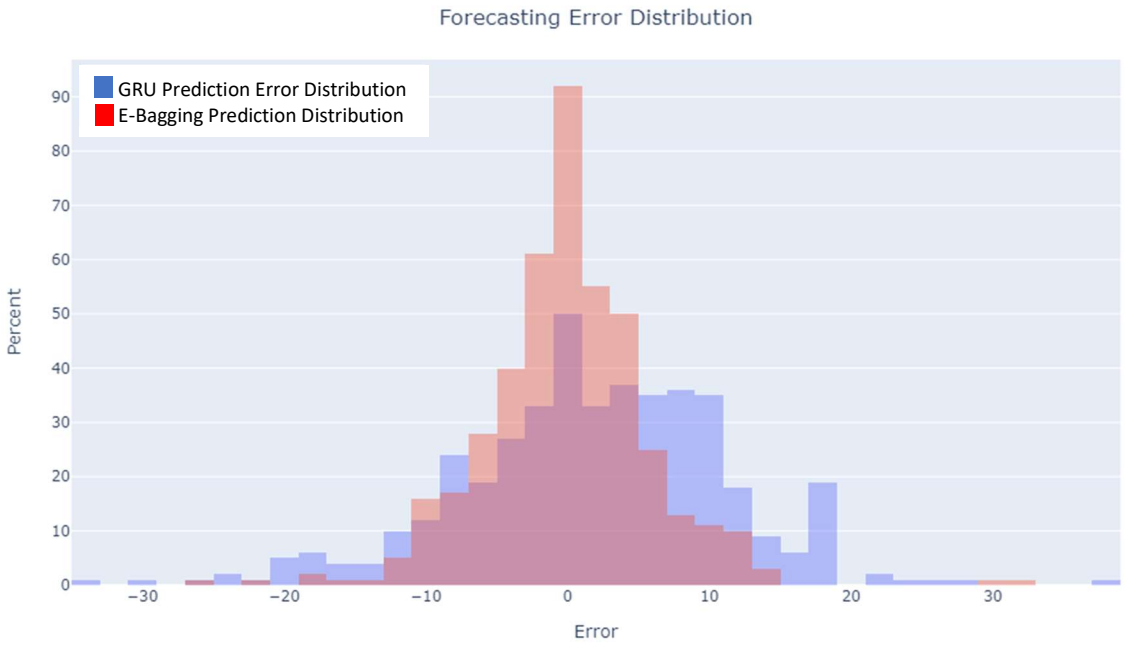


Figure 28: Prediction error distributions for the best GRU model (blue histogram) and for the E-Bagging (red histogram) for LA County; both are centered at zero. The latter has less variance.

4.1.2.3.3 E-Bagging of the GRU Model for Other Counties

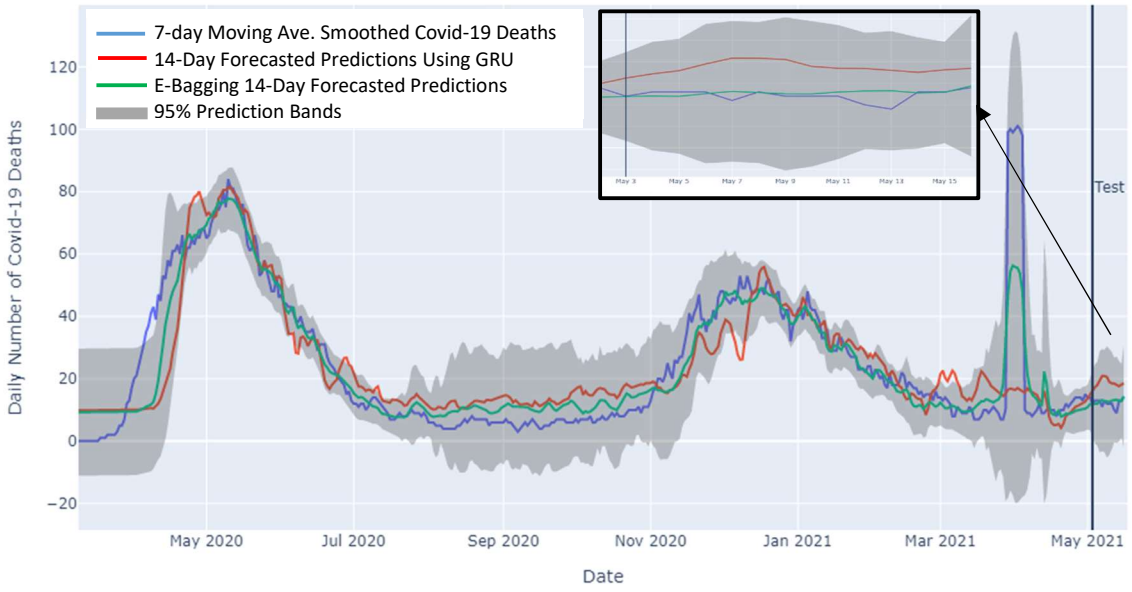
This section presents the results for the extended bagging technique applied to the GRU models for the three other counties. The predictions were made with approximate 95% prediction bands. The smoothed daily Covid-19 confirmed deaths (blue curve), their corresponding 14-day forecasted predictions using the GRU model (red curve), 14-day forecasted predictions using E-Bagging of the GRU model (green curve), and the corresponding 95% prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 for Cook County are shown in Figure 29 (top graph); for Harris County in Figure 30 (top graph); and for NY County in Figure 31 (top graph). The vertical black line in each graph separates the training and test sets. The MAE values of the

E-Bagged GRU models along with the MAE values of the GRU models for each county considered on the training and test sets are reported in Table 1. All E-Bagging MAE values on both sets are less than their corresponding GRU MAE's. According to the graphs and reported MAE values, the forecasted results using the E-Bagging method (green curve) outperformed the single GRU model (red curve) in every county.

The prediction error distributions for the GRU model (blue histogram) and for E-Bagging of the GRU model (red histogram) for Cook County are shown in Figure 32 (top graph); for Harris County are shown in Figure 32 (bottom graph); and for NY County are shown in Figure 33. While both distributions in each graph are centered at zero, the red distributions are symmetric, but the blue distributions are skewed to the left. These graphs also show that applying the E-Bagging technique to the GRU models yielded less variability in predictions. These results generally indicate that the E-Bagging technique has improved our predictions. The prediction band coverage probabilities using the E-Bagging technique on the GRU models for each county dataset are available in Table 1 (last column). They are higher than what we specified.

The cumulative counts for the different counties are shown in Figure 29 - Figure 31 (bottom graphs). In these graphs, both model predictions and prediction bands are only plotted for the test set from May 02, 2021, to May 16, 2021. According to these graphs, the 14-days-ahead forecasted cumulative Covid-19 deaths using the E-Bagging technique with the GRU models (green curves) are closer to the true values.

Daily Covid-19 Known Deaths in Cook County as of 05/16/2021



Cumulative Covid-19 Known Deaths in Cook County as of 05/16/2021

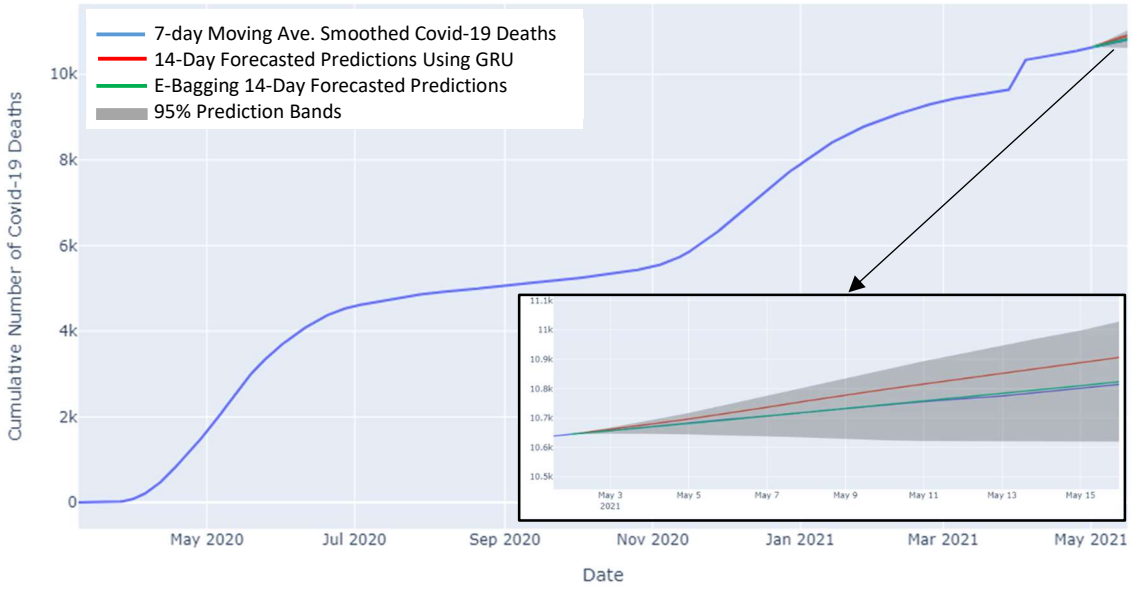
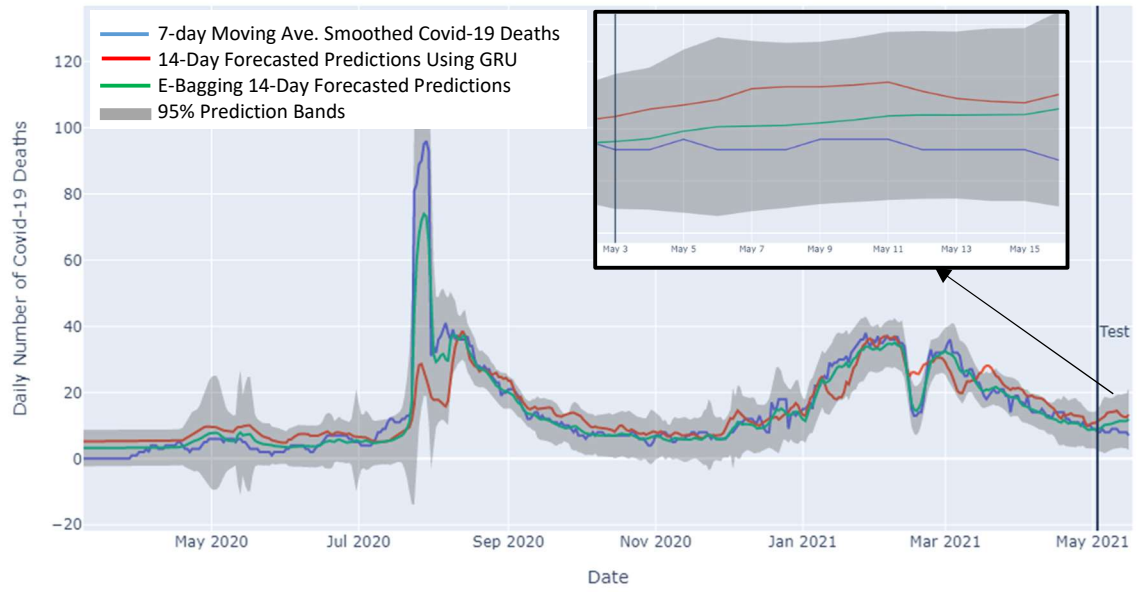


Figure 29: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Cook County. (Bottom) cumulative counts. The boxes zoom in on the test set.

Daily Covid-19 Known Deaths in Harris County as of 05/16/2021



Cumulative Covid-19 Known Deaths in Harris County as of 05/16/2021

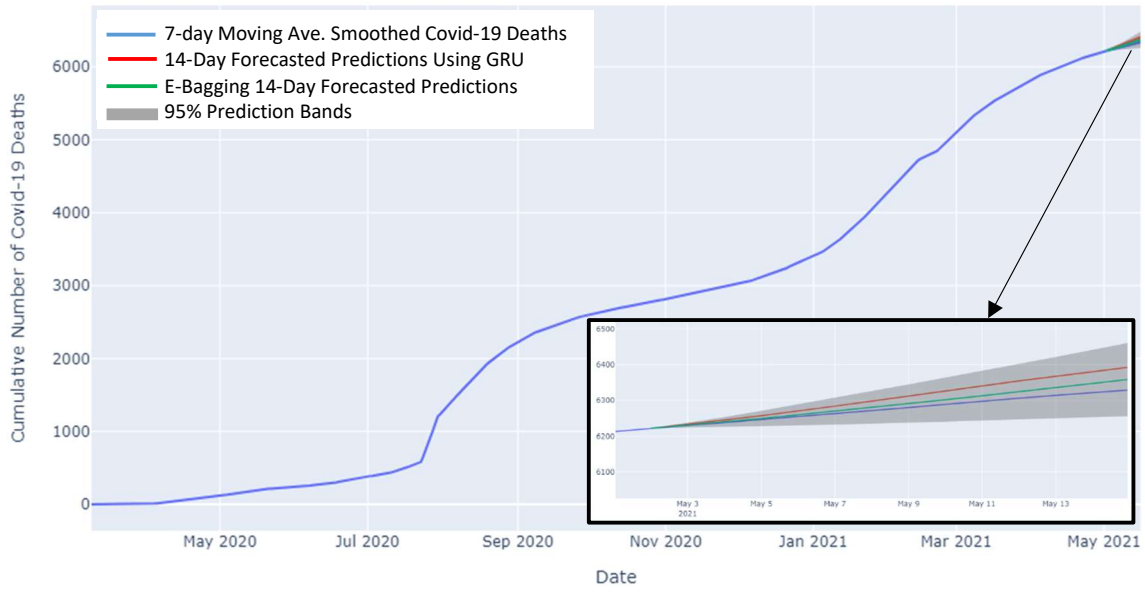
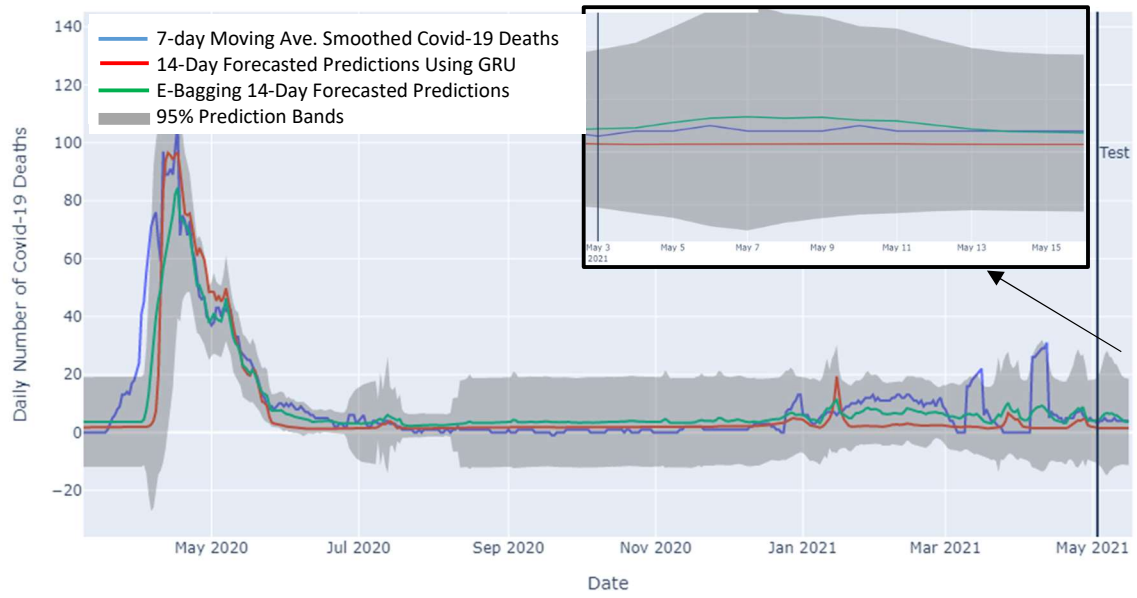


Figure 30: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Harris County. (Bottom) cumulative counts. The boxes zoom in on the test set.

Daily Covid-19 Known Deaths in NY County as of 05/16/2021



Cumulative Covid-19 Known Deaths in NY County as of 05/16/2021

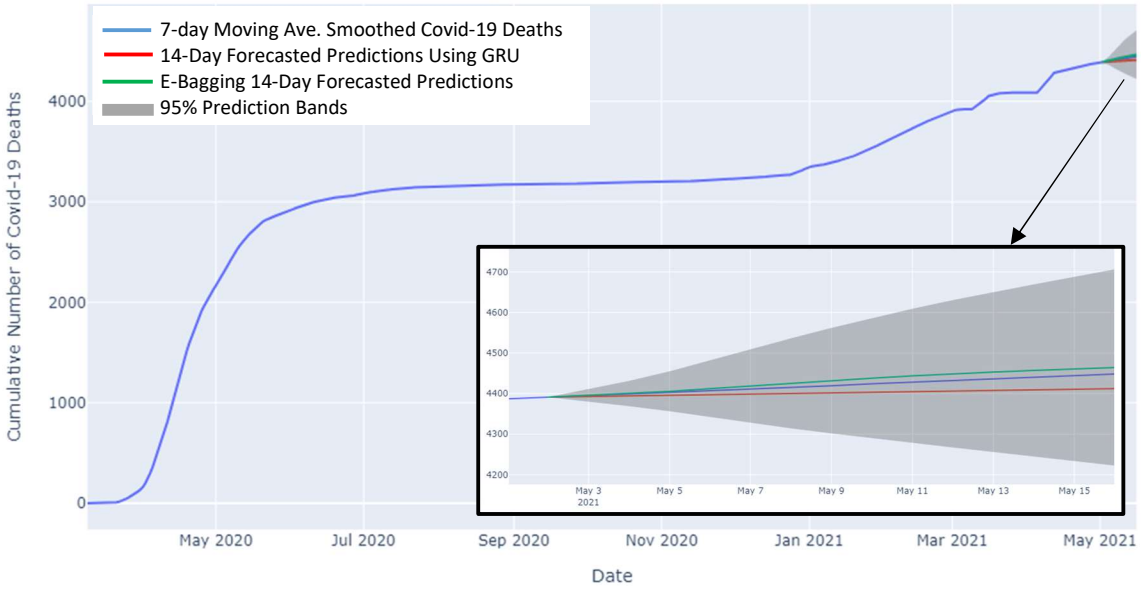


Figure 31: (Top) smoothed daily Covid-19 deaths (blue curve), GRU predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for NY County. (Bottom) cumulative counts. The boxes zoom in on the test set.

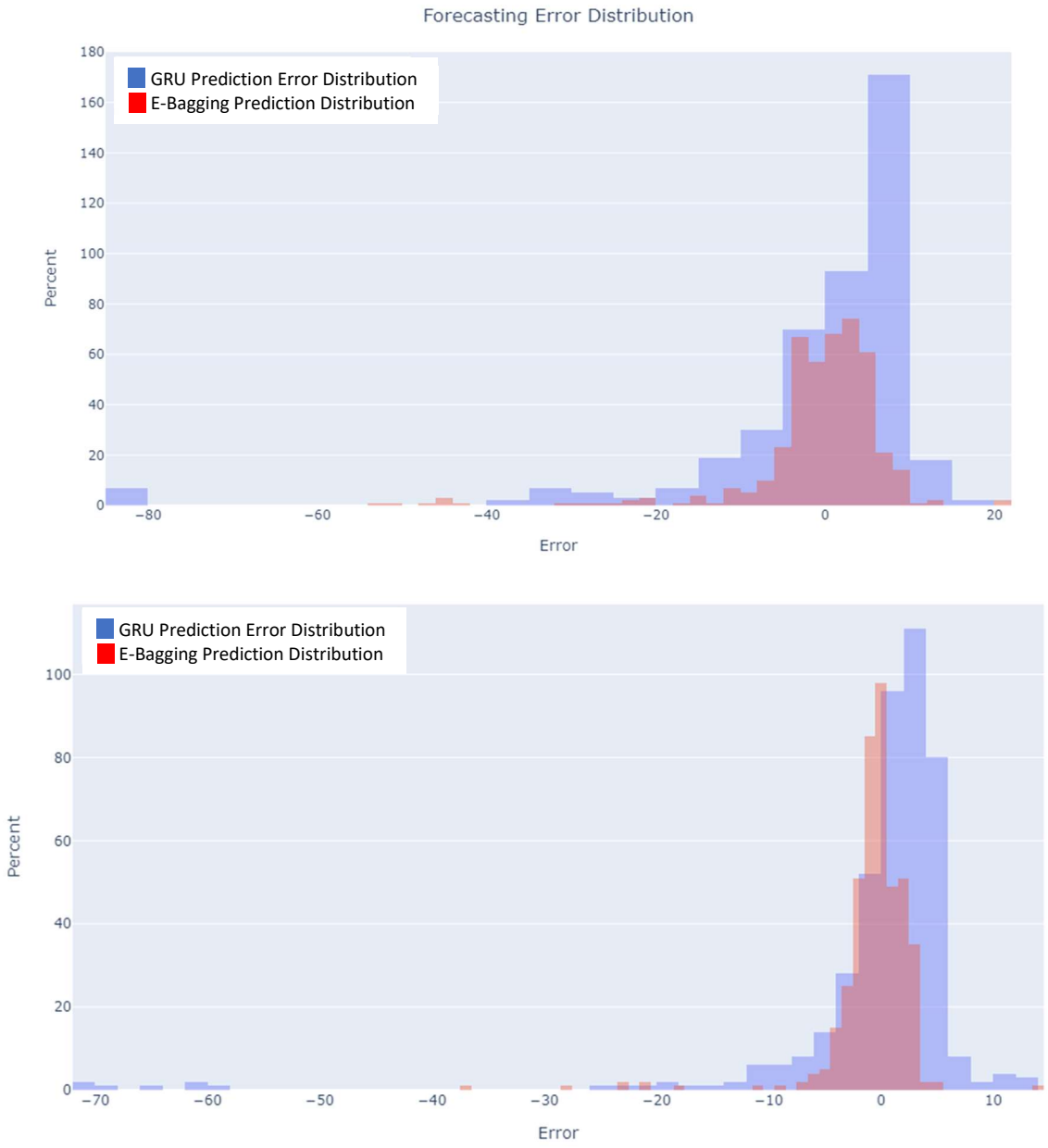


Figure 32: Prediction error distributions for the GRU model (blue histogram) and for E-Bagging (red histogram) for Cook County (top); for Harris County (bottom); both are centered at zero. The E-Bagging errors have less variance.

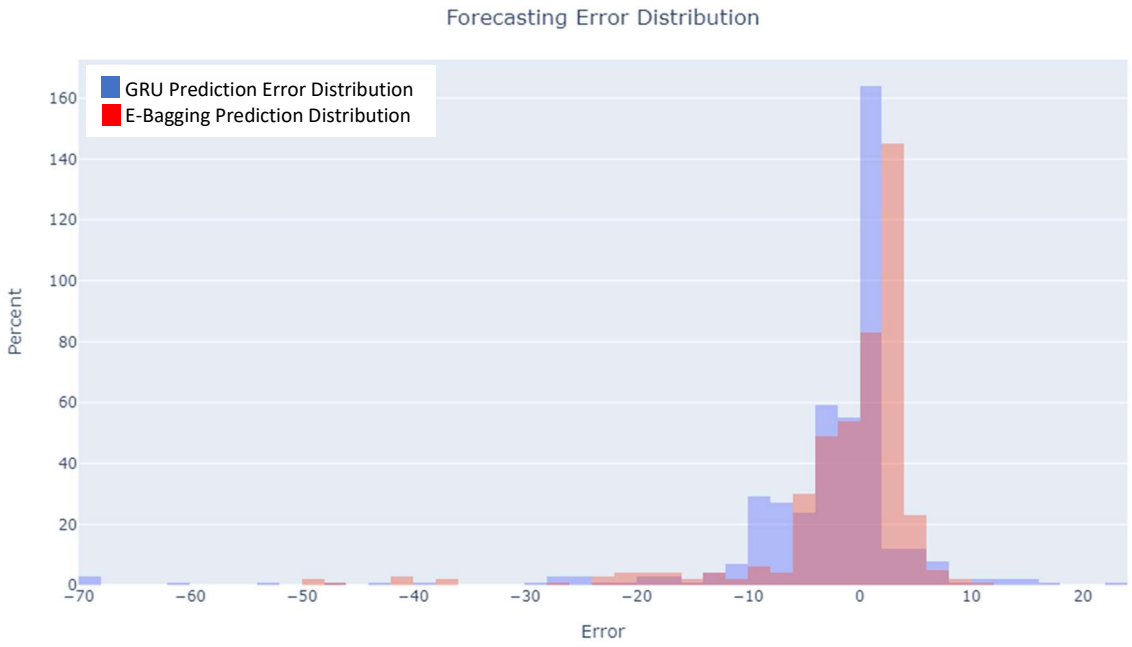


Figure 33: Prediction error distributions for the GRU model (blue histogram) and for E-Bagging (red histogram) for NY County; both are centered at zero. The latter has less variance.

4.2 Fitting Models Based on the WNN

In this section, we describe the process of predicting the number of daily Covid-19 deaths forecasted $h = 14$ days ahead based on WNN models in the following counties: LA County, CA; Cook County, IL; Harris County, TX; and NY County, NY. For each county, the target is daily Covid-19 deaths starting at the 14^{th} observation of the county of interest. The last 14 observations were considered as the test set. A full 7-day moving average was computed to obtain an accurate smoothed picture of the data for each week. The smoothed daily number of Covid-19 deaths were then normalized.

4.2.1 WNN Architecture

A sequential model including two stacked hidden layers was built and trained using Keras with the TensorFlow backend. In the next sections, we explain how WNN models are obtained, and we go further and explain the tuning of hyperparameters in detail for LA County only. NN models for other counties have similar architectures.

4.2.1.1 WNN Architecture for LA County

All predictions on the test dataset are retained and the MAE calculated to summarize the quality of the model. The idea is to compare the model configurations using MAE summary statistics over many runs (10 runs) and see exactly which of the configurations perform better on average.

Since hyperparameter tuning is time consuming and there are a lot of them here to be tuned, this could be very time-consuming. To avoid this, we set the following hyperparameters in advance based on experience. The number of epochs was set to be 1000. The early stopping approach was used to monitor the MSE to avoid

overfitting. Additionally, the number of epochs with no improvement (patience) in the early stopping approach was set to 50. The Adam optimizer with a learning rate of 0.005 was used to minimize the MSE. The batch size was set to 32. The LA WNN model hyperparameters we tuned are the numbers of nodes for all dense layers, activation functions on the dense layers, and dropout rates for both dropout layers. Another hyperparameter we tuned is the tuning constant α in the sample weights as given in Equation (3-25). This constant is a multiple of m^{-1} ($\alpha = \frac{c}{m}$) and hence we should tune the value c.

The initial LA model used a sequential model consisting of an input layer of 10 features and two dense hidden layers. This also was followed by an output layer of a single node with an ELU function.

We explored the effect of training this configuration for different numbers of nodes for the 1st dense layer (20, 30, 40), numbers of nodes for the 2nd dense layer (40, 60, 80), activation functions (Tanh, SELU, ELU, ReLU) on the first two dense layers, activation functions (Tanh, SELU, ELU) on the 3rd dense layer, dropout rates (0.1, 0.5) for both dropout layers, and values (1, 4, 7) for c. In total there are 5,184 different hyperparameter combinations. We used the error analysis method and repeated this training regimen 10 times, calculating the overall MAE given in Equation (3-39) for each configuration on the 17 test sets created from the second half of the observations. Boxplots of the overall MAE distributions on these test sets after each training with different hyperparameter combinations are shown in Figure 34 - Figure 37. Note that from all possible 5,184 different hyperparameter combinations, just 144 are reported here.

The overall MAE values range from 21.5 to 36. Based on examination of the median MAE and variance for all hyperparameter combinations, the choice of setting the

number of 1st dense layer nodes to 30, activation function on the 1st dense layer to SELU, dropout rate for the 1st dropout layer to 0.5, number of 2nd dense layer nodes to 40, activation function on the 2nd dense layer to Tanh, dropout rate for the 2nd dropout layer to 0.1, activation function on the 3rd dense layer to ELU, and value c to 1 offers the best performance on average in terms of both median MAE and variability.

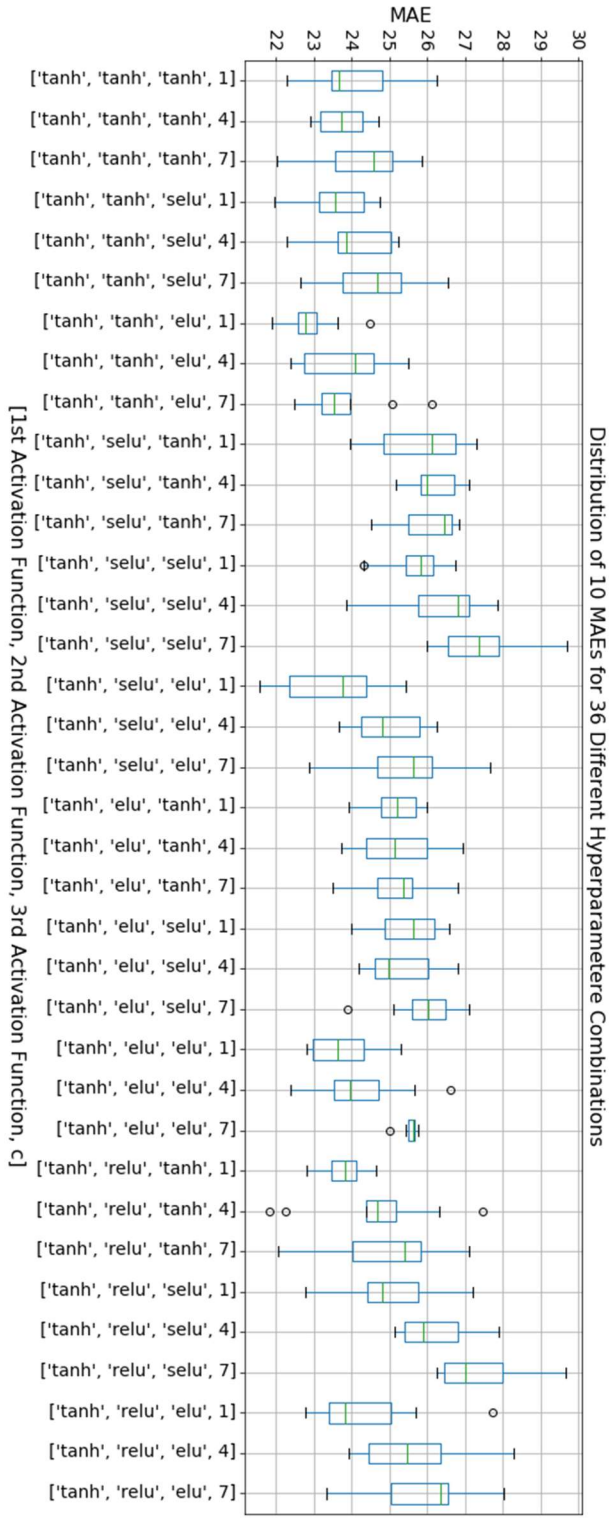


Figure 34: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is Tanh, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot.

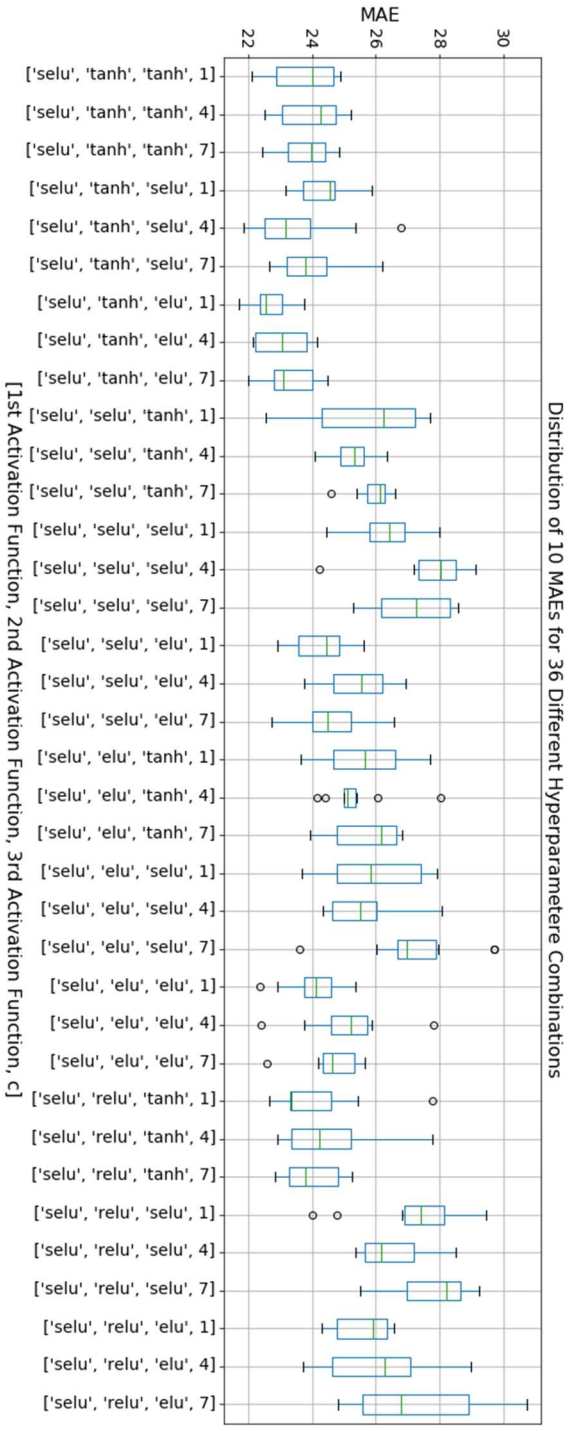


Figure 35: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is SELU, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot.

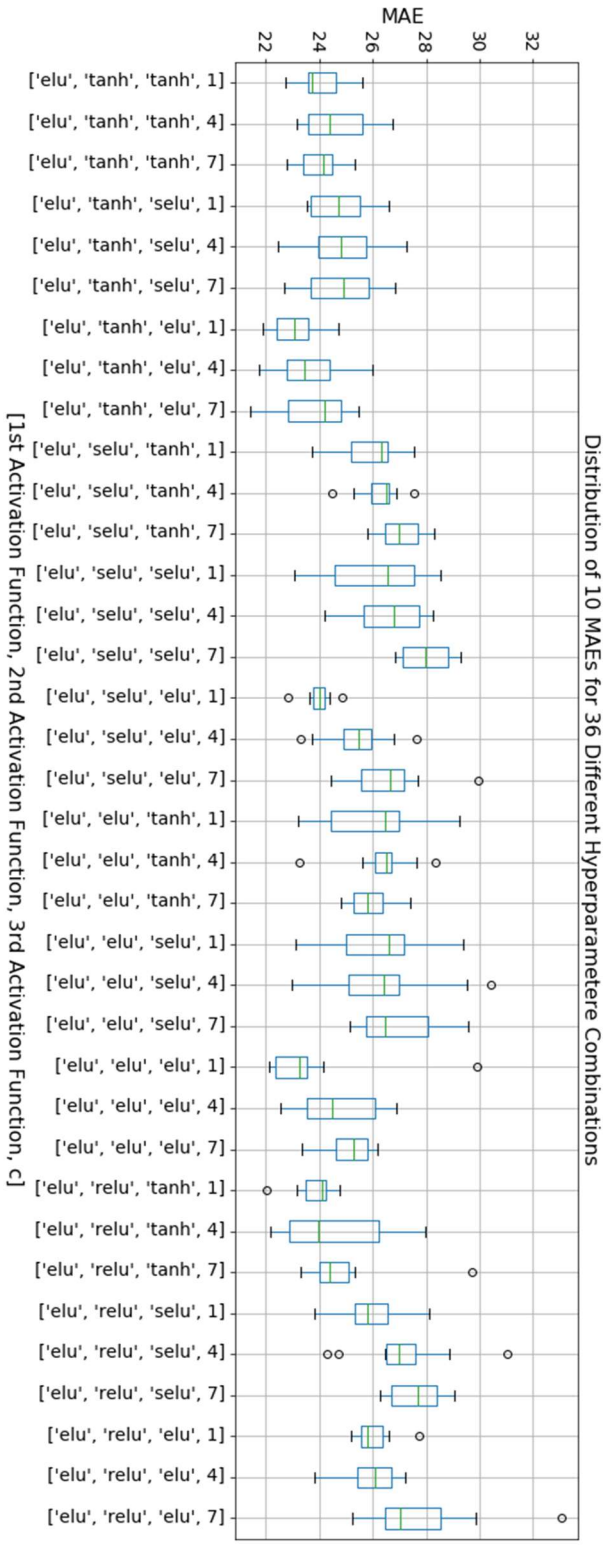


Figure 36: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is ELU, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot.

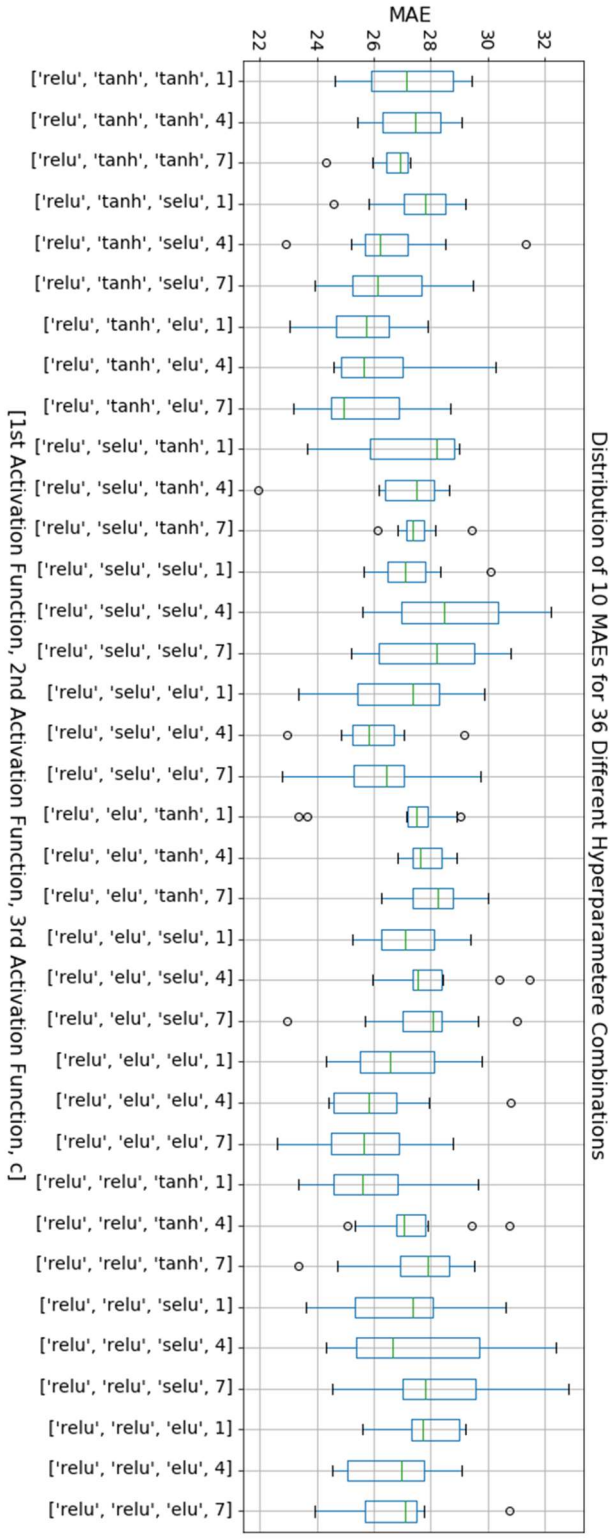


Figure 37: Diagnostic results with different hyperparameter combinations1 in the LA model (when # 1st dense layer nodes is 30, activation function on the 1st dense layer is ReLU, dropout rate for the 1st dropout layer is 0.5, # 2nd dense layer nodes is 40, and dropout rate for the 2nd dropout layer is 0); in total, 360 models were fit to produce this boxplot.

4.2.1.2 E-Bagging of WNN Models

According to the hyperparameter tuning in the LA WNN model, the best LA WNN architecture was obtained using a sequential model consisting of an input layer containing 10 features. This was followed by a dense layer of 60 nodes with an SELU activation function. It is immediately followed by a dropout layer to prevent the model from overfitting with a dropout rate of 0.5. Following the dropout layer, we added another dense layer of 40 nodes with a Tanh activation function, followed by a dropout layer with a dropout rate of 0.1. Lastly, this was followed by an output layer of a single node. The activation function on the output layer was an ELU. The MSE was minimized using the Adam optimizer with a learning rate of 0.005. The batch size and the number of epochs were 32 and 1000, respectively. The early stopping method was used to monitor the MSE with a patience (number of epochs with no improvement in the loss) of 50.

The total number of trainable parameters for the best LA WNN model is 1,611, where 330 parameters were trained in the first dense layer, 1,240 parameters were trained in the 2nd dense layer, and lastly, 41 parameters were trained in the output layer. Note that this total number is much less than the number for the best LA GRU model which was 23,721. Hence, using this WNN architecture is roughly 7 times faster in fitting than the GRU models.

Other NN models for different counties have a similar architecture to the best LA WNN model as well. However, they may have different numbers of input features. After finding the WNN models for each county considered, we performed the E-Bagging technique on the WNN models using $B = 128$ to predict the Covid-19 death counts and construct 95% prediction bands for 14 days of forecasted values in each county. Since there was no real difference in results when tuning the GRU model as a function of B, we chose $B = 128$ to match the GRU model choice.

The smoothed daily Covid-19 confirmed deaths (blue curve), their corresponding 14-day predictions using the WNN model (red curve), using E-Bagging of the WNN model (green curve), and the corresponding 95% prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021 for LA County are shown in Figure 38 (top graph); for Cook County in Figure 39 (top graph); for Harris County in Figure 40 (top graph); and for NY County in Figure 41 (top graph). The vertical black line in each graph separates the training and test sets. The MAE values of the E-Bagged of the WNN models along with the MAE values of the WNN models for each county considered with the training and test sets are reported in Table 2. All E-Bagging MAE values for both sets are less than their corresponding WNN MAE's. According to the graphs and reported MAE values, the forecasted results using the E-Bagging method (green curve) outperformed using the single WNN model (red curve) in every county.

The prediction error distributions for the WNN model (blue histogram) and for E-Bagging of the WNN model (red histogram) are shown in Figure 42 and Figure 43. While both distributions in each graph are centered at zero, the red distributions are symmetric, but the blue distributions are skewed to the left. These graphs also show that applying the E-Bagging technique to the WNN models yielded less variability in predictions. These results generally indicate that the E-Bagging technique has improved our predictions. The prediction band coverage probabilities using the E-Bagging technique on the WNN models for each county dataset are available in Table 2 (last column). They are higher than what we specified.

The cumulative counts for the different counties are shown in Figure 38 - Figure 41 (bottom graphs). In these graphs, both model predictions and prediction bands are only plotted for the test set from May 02, 2021, to May 16, 2021.

Table 2: MAE of different models for each county on the training and test sets

	Train MAE		Test MAE		CP (%)
	WNN	E-Bagging	WNN	E-Bagging	
LA County	14.158	11.269	12.346	8.768	98.1
Cook County	9.360	8.098	4.441	2.079	96.8
Harris County	5.786	5.661	9.550	5.841	97.4
NY County	7.274	4.761	1.150	0.512	96.5
Average	9.145	7.445	6.872	4.300	97.2

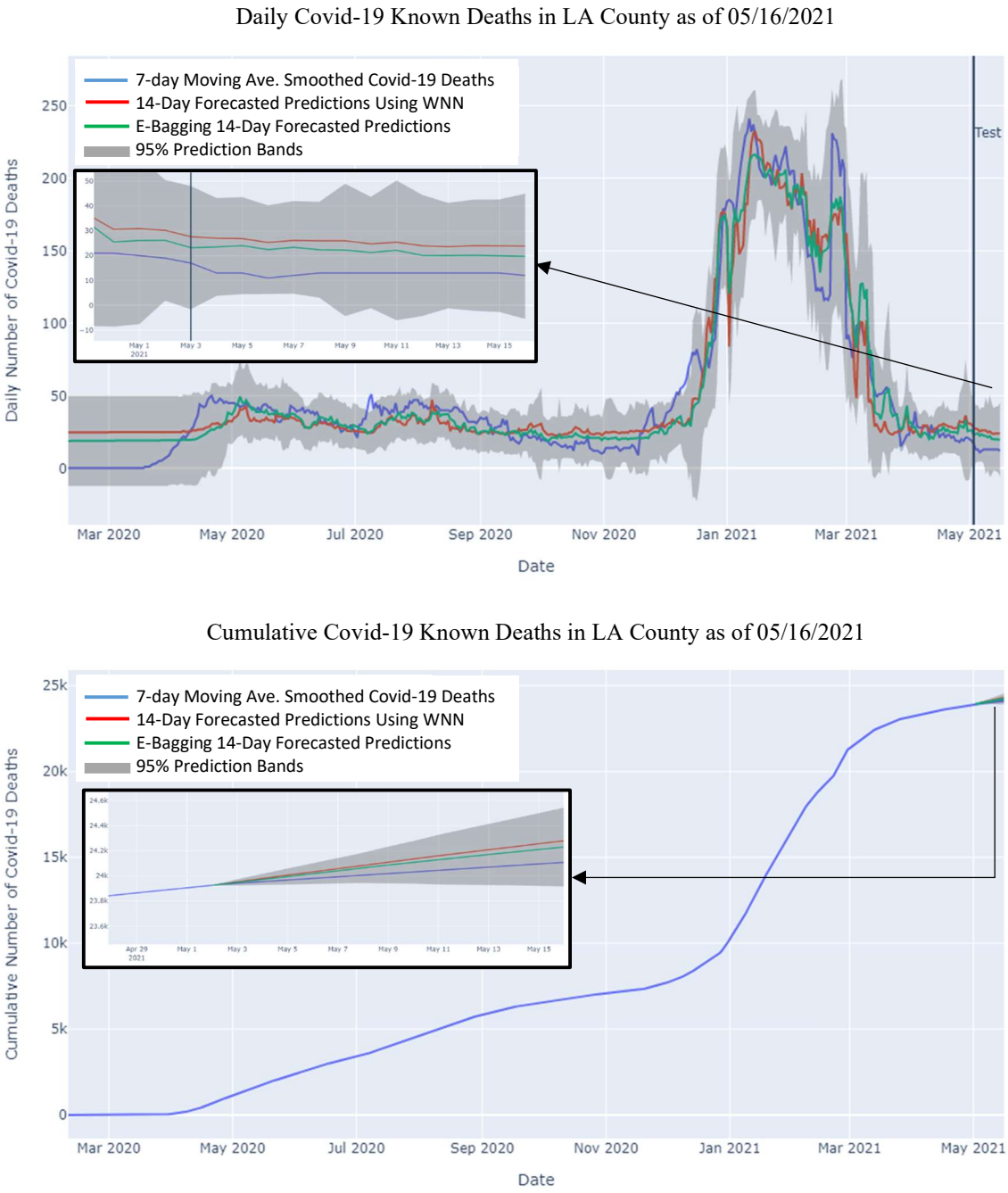
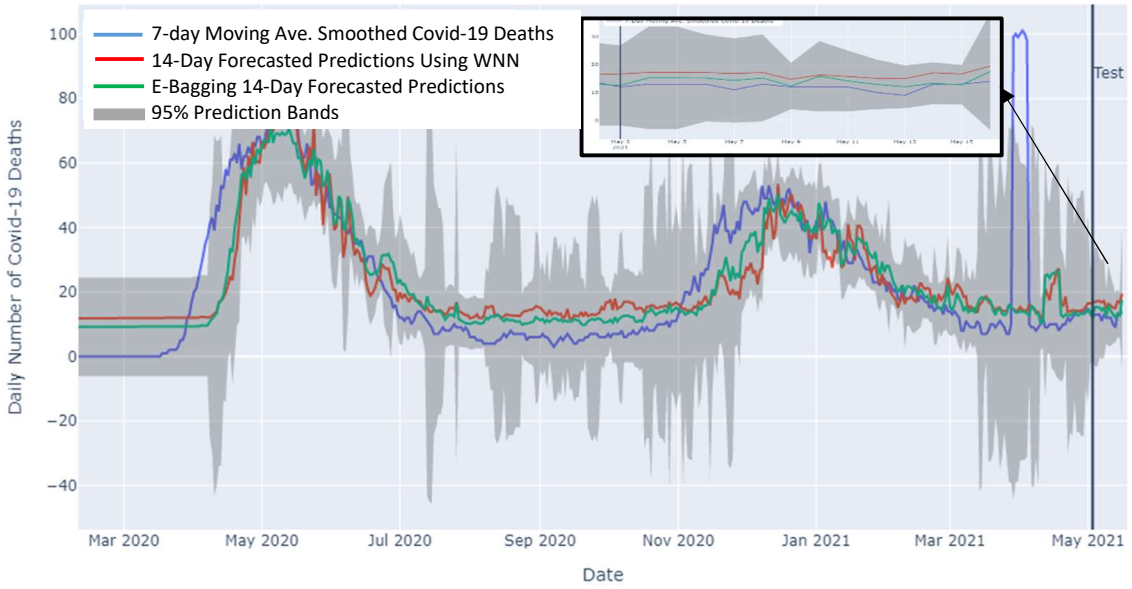


Figure 38: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for LA County. (Bottom) cumulative counts. The boxes zoom in on the test set.

Daily Covid-19 Known Deaths in Cook County as of 05/16/2021



Cumulative Covid-19 Known Deaths in Cook County as of 05/16/2021

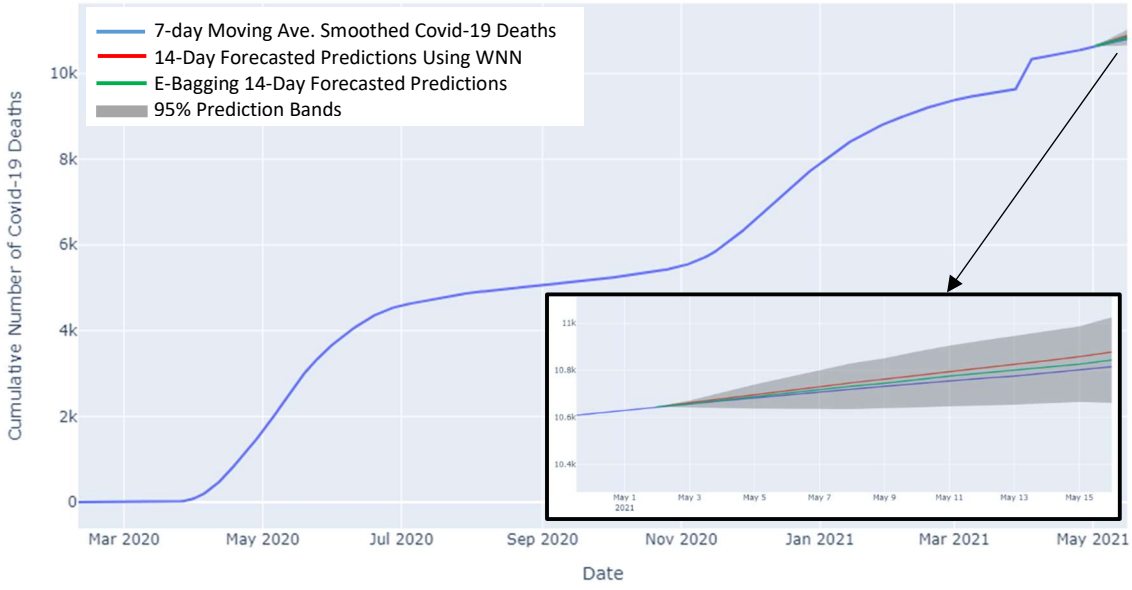
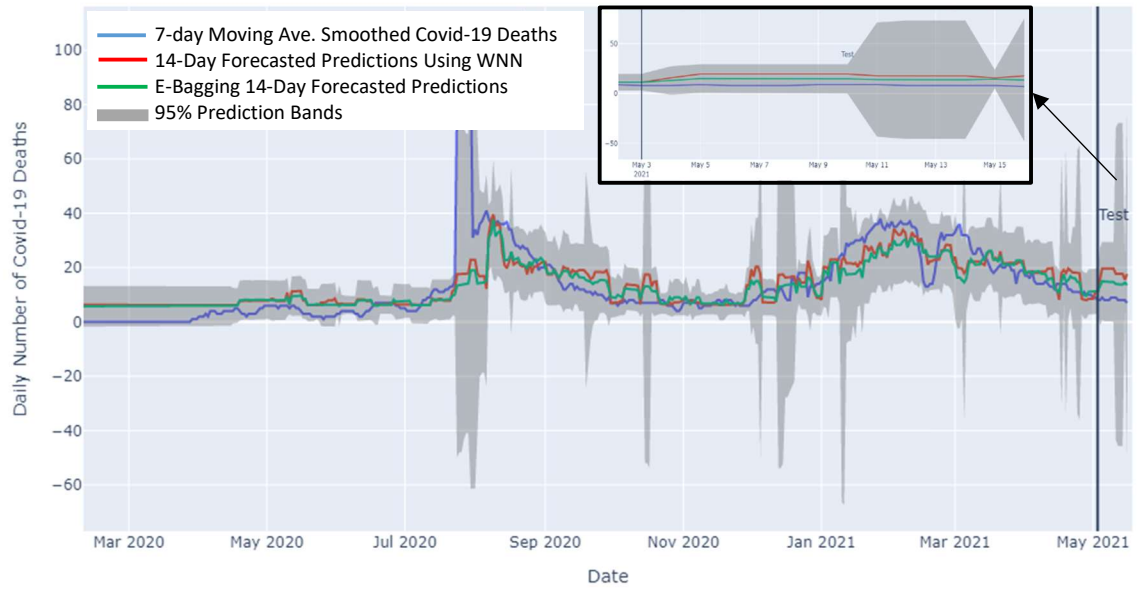


Figure 39: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Cook County. (Bottom) cumulative counts. The boxes zoom in on the test set.

Daily Covid-19 Known Deaths in Harris County as of 05/16/2021



Cumulative Covid-19 Known Deaths in Harris County as of 05/16/2021

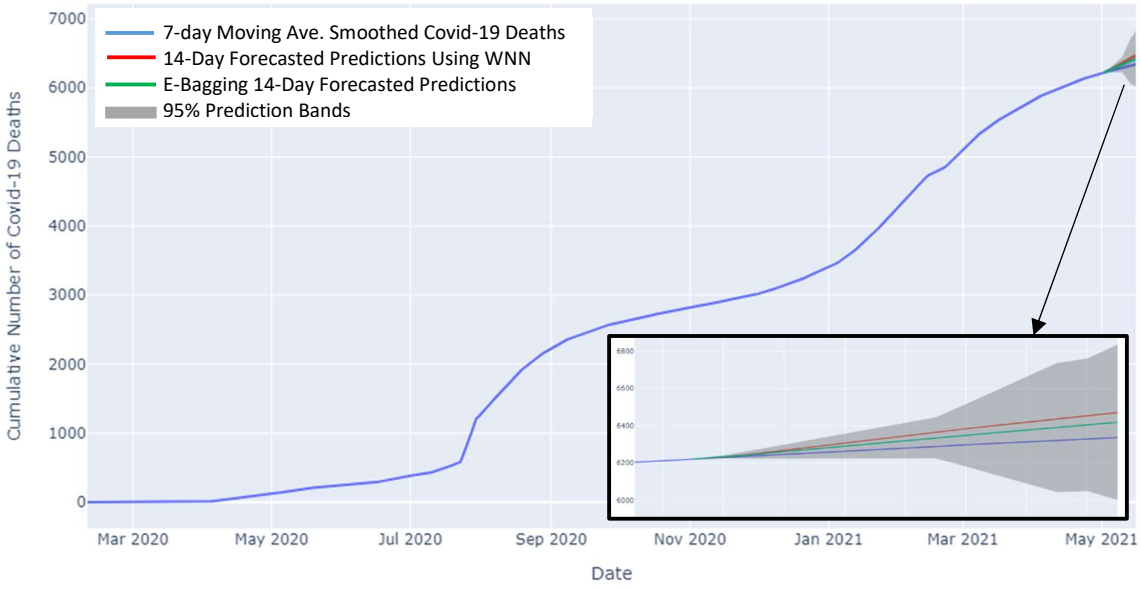
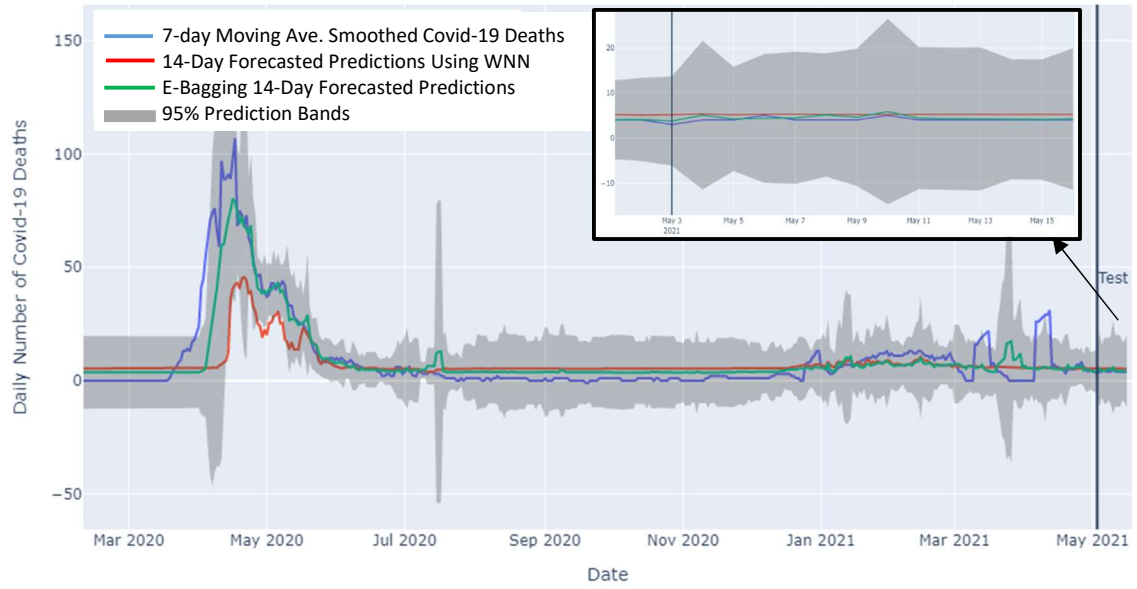


Figure 40: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for Harris County. (Bottom) cumulative counts. The boxes zoom in on the test set.

Daily Covid-19 Known Deaths in NY County as of 05/16/2021



Cumulative Covid-19 Known Deaths in NY County as of 05/16/2021

Figure 41: (Top) smoothed daily Covid-19 deaths (blue curve), WNN predictions (red curve), E-Bagging predictions (green), and prediction bands (gray ribbon) for the training set up to 05/01/2021 and for the test set from May 02, 2021, to May 16, 2021, for NY County. (Bottom) cumulative counts. The boxes zoom in on the test set.

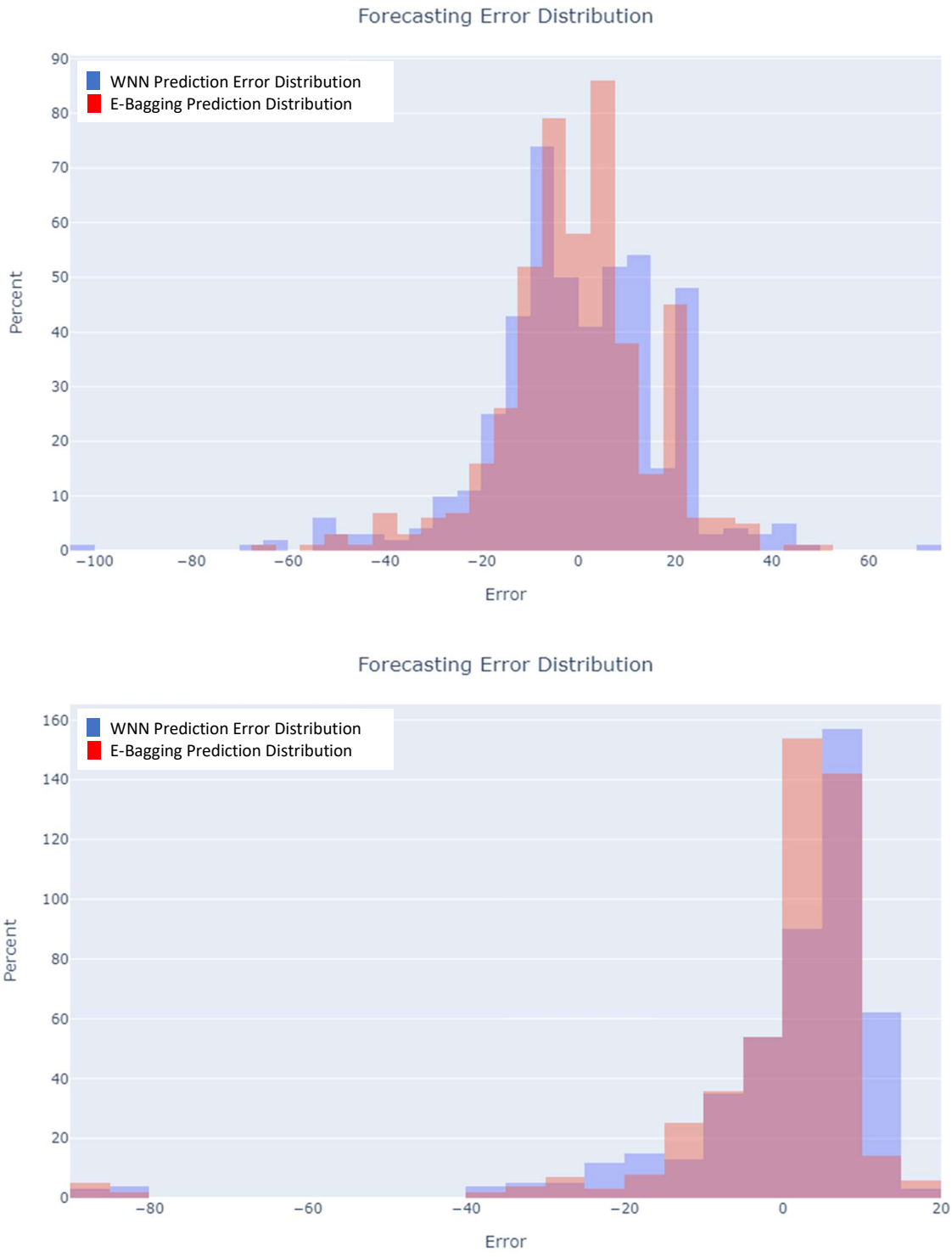


Figure 42: Prediction error distributions for the WNN model (blue histogram) and for E-Bagging (red histogram) for LA County (top); for Cook County (bottom); both are centered at zero. The E-Bagging errors have less variance.

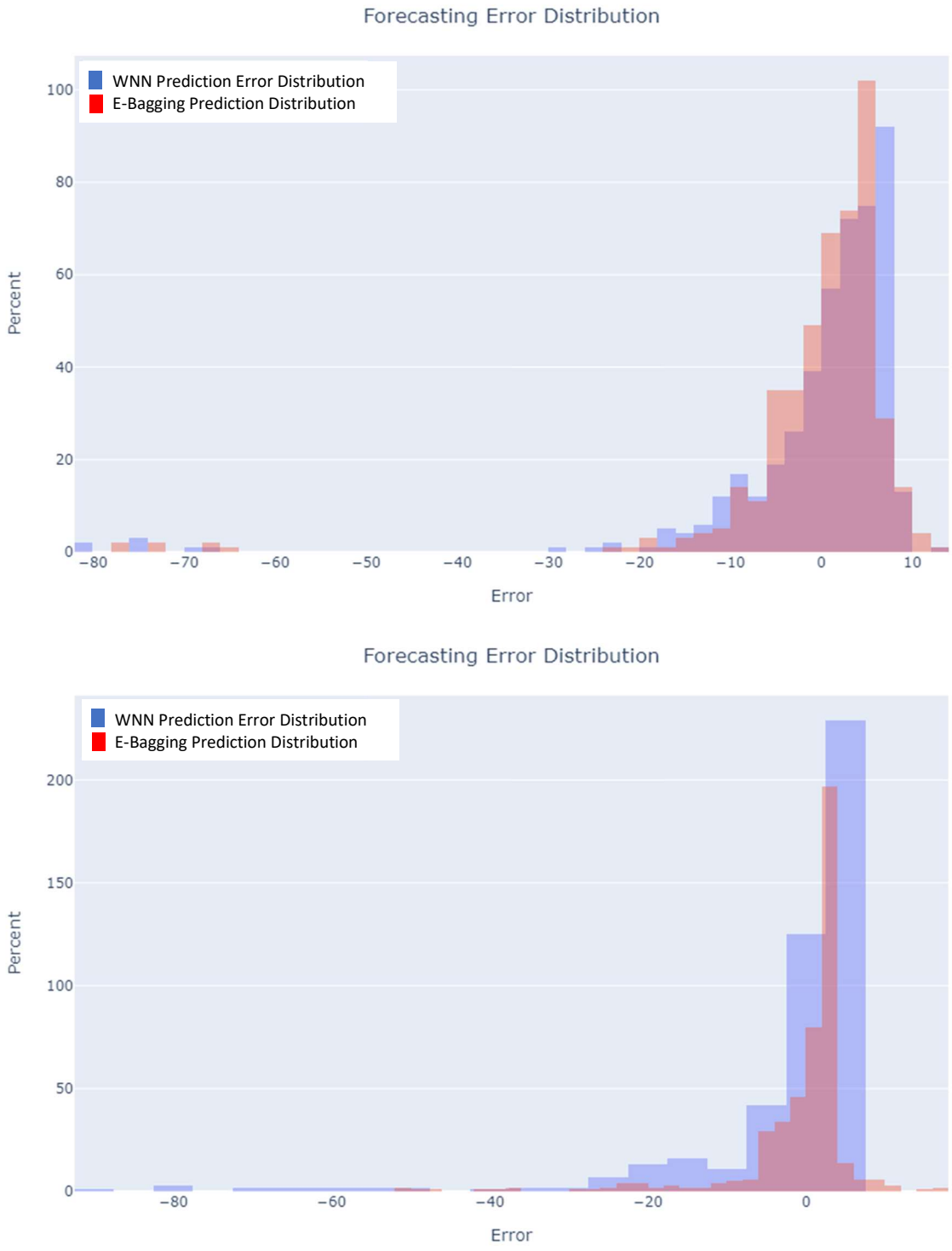


Figure 43: Prediction error distributions for the WNN model (blue histogram) and for E-Bagging (red histogram) for Harris County (top); for NY County (bottom); both are centered at zero. The E-Bagging errors less variance.

5 Discussion and Conclusions

In this study, we have proposed deep learning models for predicting the number of Covid-19 death counts at the US county level. GRUs, WNNs and E-Bagging of these two networks are used as prediction models. The MAE performance measure was used to compare the models. Models were tested on four US counties and based on the MAE, the model with minimum error was chosen. The results of the MAE of NN models and the coverage probability for assessment of constructed prediction bands using the E-Bagging technique for four counties are available in Table 3. In viewing this table, WNN model predictions have higher errors than other model predictions on both training and test sets on average. The WNNs predict the number of deaths on both training and test sets with the highest average MAE values of 9.145 and 6.872, respectively.

Table 3: Results of MAE and coverage probability of different models

	Train MAE				Test MAE				CP (%)	
	WNN		GRU		WNN		GRU			
	Conventional	E-Bagging	Conventional	E-Bagging	Conventional	E-Bagging	Conventional	E-Bagging	WNN	GRU
LA County	14.158	11.269	7.363	4.155	12.346	8.768	9.343	5.884	98.1	99.8
Cook County	9.360	8.098	8.373	5.023	4.441	2.079	6.528	1.073	96.8	98.6
Harris County	5.786	5.661	4.547	2.018	9.550	5.841	4.904	2.386	97.4	100
NY County	7.274	4.761	5.099	4.466	1.150	0.512	2.575	1.277	96.5	97.9
Average	9.145	7.445	6.346	3.916	6.872	4.300	5.838	2.655	97.2	99.1

E-Bagging applied to the GRU models outperformed other models on average and gave lower average MAE values with the training set than the test set. On the test

set, it gives the least error for LA County, Cook County, and Harris County while for NY County, E-Bagging applied to the WNN model gives the least MAE. E-Bagging applied to NNs shows improved predictions when compared with conventional GRU and WNN predictions, depicting lower average prediction errors. In viewing the columns of the coverage probabilities in Table 3, it seems all prediction bands have a higher coverage probability than the specified level of 95%. Our residual predictor NN could probably have been further optimized in order to generate prediction bands with coverage probabilities closer to the specified coverage level. Applying the E-Bagging technique to the GRU models also provides higher (worse) coverage probability (99.1%) than to the WNN models (97.2) on average.

It can be concluded that E-Bagging of GRU models is an appropriate prediction technique for such spatially and temporally correlated data. It is capable of forecasting 14 days into the future with improved accuracy. However, prediction intervals tended to be too wide for all cases. These predictions and their prediction bands may be helpful to county and state authorities, researchers, and planners who manage services such as arranging medical infrastructure, the public health system, closure, and lockdown.

If a valid model predicts a severe increase in the number of deaths due to Covid-19 in a particular county for the two coming weeks, this can put enormous pressure on the healthcare system. With accurate forecasting, we can control the virus spread in advance through various public health measures from mask mandates to lockdown.

The proposed model can be used by other counties, states, and nations for Covid-19 predictions and can be applied to a wide variety of other situations from Ebola epidemic mitigation to intra- and inter-day stock price forecasting.

For future work, we can apply the proposed E-Bagging techniques to other RNNs such as the Bi-directional GRU, stacked GRU or variations of LSTM architectures. Also, we can apply the proposed models over other temporally correlated data such as stock market price datasets for forecasting opening prices for h days into the future.

The output of our proposed models can help planners and authorities decide on appropriate measures to limit Covid-19 spread. The county-wide predictions can help county authorities balance the load taken on by the medical infrastructure. Such predictions can also help authorities manage appropriate levels of economic activity in the community.

Public awareness is an important and key step in implementing Covid-19 mitigation strategies. Los Angeles County is the most populous county in the US with a population of around 9.82 million. Large counties are of great concern due to their high population density and higher likelihood of spread. Covid-19 can attain exponential growth in its spread in densely populated areas very easily.

6 Appendix A

Python Codes for GRU and E-Bagging

6.1 Loading Required Modules

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from sklearn.preprocessing import MinMaxScaler
import keras
import tensorflow.compat.v1 as tf
import keras.backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, GRU, Input, Activation
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping
from itertools import combinations
from scipy.stats import norm
from itertools import compress
```

6.2 A Function to Create Raw and Smoothed Datasets

```
def dataset(data, countyFIPS):
  dropped_variables = ['countyFIPS', 'State', 'stateFIPS']
  data = data[data.countyFIPS.isin(countyFIPS)]
  data.set_index('County Name', inplace=True)
  data = data.drop(data.columns[[0, 2,3]], axis=1).T.reset_index()
  data = data.rename(columns = {'index': 'Date'}, inplace = False)
  data.set_index('Date', inplace=True)
  data = data[1:].diff(periods=1,axis=0)[1:]
  smoothed_data = data.rolling(window=7, center=True).mean()
  smoothed_data = smoothed_data.dropna()
  return data, smoothed_data
```

6.3 Loading Dataset for LA County

```
df = pd.read_csv('covid_deaths_usafacts_3.csv')                    # Read in the csv file

counties = ['Los Angeles County ', 'Orange County ', 'San Diego County ', 'Riverside County ', 'San Bernardino County ', 'Kern County ', 'Ventura County ', 'Santa Barbara County ', 'Imperial County ', 'San Luis Obispo County ']
                                                                   # Select the US neighboring counties of LA County and find their FIPS in the excel file

countyFIPS = [6037, 6059, 6073, 6065, 6071, 6029, 6111, 6083, 6025, 6079]
deaths, smoothed_deaths = dataset(df, countyFIPS)                  # Create raw and smoothed datasets using "dataset" function
target_name = "Los Angeles County "                                # Set a county of target to be analyzed

target = deaths.columns.get_loc(target_name)                       # Find the corresponding column number of the county of target
```

6.4 Codes for Creating Figure 4

```
# Create a line chart for Cumulative raw COVID-19 deaths in different counties around the target

fig = go.Figure()
for i in range(deaths.shape[1]):
   fig.add_trace(go.Scatter(x=deaths.index, y=deaths.iloc[:,i].cumsum(), name= deaths.columns[i]))

fig.update_layout(legend=dict(x=0.01, y=.99),
                  title={'text': "Cumulative Real Covid-19 Known Deaths in Different Counties in California as of 05/19/2021 ",'y':0.9, 'x':0.5,
                     'xanchor': 'center', 'yanchor': 'top'},
                  xaxis_title="Date",
                  yaxis_title="Cumulative Covid-19 Deaths",
                  width=1000, height=600)
fig.show()
```

6.5 Codes for Creating Figure 6

```
# Create a bar chart with a smoothed line for daily COVID-19 deaths in County of target
fig = go.Figure()

fig.add_bar(name= 'Real Covid-19 Confirmed Deaths',
   x=deaths.index,
   y=deaths.iloc[:, target])

fig.add_trace(go.Scatter(name= '7-day Moving Ave. Smoothing Including 3 Days Before and
aAfter the Date of the Report',
   x=deaths.index[3:-3],
   y=smoothed_deaths.iloc[:,target],
   line=dict(width=3)))

fig.update_layout(legend=dict(x=0.01, y=.99),
            title={'text': "Real and Smoothed Daily Covid-19 Known Deaths in %s as of
05/19/2021"%target_name,'y':0.9, 'x':0.5,
                  'xanchor': 'center', 'yanchor': 'top'},
            xaxis_title="Date",
            yaxis_title="Daily Covid-19 Deaths",width=1000, height=600)

fig.show()
```

6.6 Plotting Smoothed and Forecasted Cumulative COVID-19 Deaths Using Different Models

```
def visualize(data, y_predicted, Ebagging_interval, Ebagging_y_pred, lower_bound,
upper_bound):
  y = data.iloc[rolling_window+days_ahead:, target].values
# Plot Smoothed vs. Forecasted Daily COVID-19 Deaths in County of target
  fig = go.Figure()

  fig.add_trace(go.Scatter(x=data.index[rolling_window+days_ahead:],
                    y=y,
                    name='7-Day Moving Ave. Smoothed Covid-19 Deaths',))

  fig.add_trace(go.Scatter(x=pd.date_range(data.index[rolling_window+days_ahead],
periods=len(y_predicted)),
                    y=y_predicted,
                    name='14-Day Forecasted Predictions Using GRU',
                    fill=None))
  fig.add_vline(x=data.index[-days_ahead])
```

```
  fig.add_annotation(x=data.index[-days_ahead+7], y=np.max(y_predicted),
          text="Test", showarrow=False)

  if Ebagging_interval == True:
    fig.add_trace(go.Scatter(x=pd.date_range(data.index[rolling_window+days_ahead],
periods=len(y_predicted)),
                      y=Ebagging_y_pred,
                      name='Extended Bagging 14-Day Forecasted Predictions',
                      fill=None))

fig.add_trace(go.Scatter(x=pd.date_range(smoothed_deaths.index[rolling_window+days_ahead],
periods=len(y_predicted)),
                    y=lower_bound,
                    #name='Lower Bound',
                    showlegend=False,
                    mode='lines',
                    marker=dict(color="#444"),
                    line=dict(width=0) ))


fig.add_trace(go.Scatter(x=pd.date_range(smoothed_deaths.index[rolling_window+days_ahead],
periods=len(y_predicted)),
                    y=upper_bound,
                    name='95% Prediction bands',
                    mode='lines',
                    marker=dict(color="#444"),
                    line=dict(width=0),
                    fillcolor='rgba(68, 68, 68, 0.3)',
                    fill='tonexty'  ))

  fig.update_layout(legend=dict(x=0.01, y=.99),
                title={'text':  "Daily  Covid-19  known  deaths  in  %s  as  of
05/16/2021"%target_name,'y':0.9, 'x':0.5,
                     'xanchor': 'center', 'yanchor': 'top'},
                xaxis_title="Date",
                yaxis_title="Daily Number of Covid-19 Deaths",
                height=600, width=1000)

  fig.show()


##############################################################################
  # Plot Smoothed vs. Forecasted Cumulative COVID-19 Deaths in County of target
  fig = go.Figure()
```

```
 fig.add_trace(go.Scatter(x=data.index[rolling_window+days_ahead:],
                  y=y.cumsum(),
                  name='7-Day Moving Ave. Smoothed Covid-19 Deaths',))

 fig.add_trace(go.Scatter(x=pd.date_range(data.index[-days_ahead-1], periods=days_ahead+1),
                  y=np.append(y.cumsum()[-days_ahead-1],                y.cumsum()[-days_ahead-
1]+y_predicted[-days_ahead:].cumsum()),
                  name='14-Day Forecasted Covid-19 Deaths Using GRU',
                  mode='lines',
                  fill=None))

 if Ebagging_interval == True:
  fig.add_trace(go.Scatter(x=pd.date_range(data.index[-days_ahead-1], periods=days_ahead+1),
                  #y=Ebagging_y_pred.cumsum(),
                  y=np.append(y.cumsum()[-days_ahead-1],                y.cumsum()[-days_ahead-
1]+Ebagging_y_pred[-days_ahead:].cumsum()),
                  name='Extended Bagging 14-Day Forecasted Predictions',
                  mode='lines',
                  fill=None))
  fig.add_trace(go.Scatter(x=pd.date_range(data.index[-days_ahead-1], periods=days_ahead+1),
                 #y=lower_bound.cumsum(),
                 y=np.append(y.cumsum()[-days_ahead-1],                y.cumsum()[-days_ahead-
1]+lower_bound[-days_ahead:].cumsum()),
                 #name='Lower Bound',
                 showlegend=False,
                 mode='lines',
                 marker=dict(color="#444"),
                 line=dict(width=0) ))

  fig.add_trace(go.Scatter(x=pd.date_range(data.index[-days_ahead-1], periods=days_ahead+1),
                  #y=upper_bound.cumsum(),
                  y=np.append(y.cumsum()[-days_ahead-1],                y.cumsum()[-days_ahead-
1]+upper_bound[-days_ahead:].cumsum()),
                  name='95% Prediction bands',
                  mode='lines',
                  marker=dict(color="#444"),
                  line=dict(width=0),
                  fillcolor='rgba(68, 68, 68, 0.3)',
                  fill='tonexty'  ))

 fig.update_layout(legend=dict(x=0.01, y=.99),
              title={'text':  "Cumulative  COVID-19  known  deaths  in  %s  as  of
05/16/2021"%target_name,'y':0.9, 'x':0.5,
                   'xanchor': 'center', 'yanchor': 'top'},
             xaxis_title="Date",
             yaxis_title="Cumulative Number of Covid-19 Deaths",
```

```
        height=600, width=1000)

  fig.show()
```

6.7 Build a 3-Dimensional Rolling Window Array (R) and Split and Scale Train and Test Sets

```
def split_train_test(data):
  target = data.columns.get_loc(target_name)
  dataset = data.values
  unscaled_y = dataset[rolling_window + days_ahead : , target].reshape(-1, 1)
  n = len(unscaled_y) - days_ahead

  # Transform the data to have a specific scale. Specifically, to rescale the data to [-1,1] to meet the tanh activation function of the GRU model.
  y_normaliser = MinMaxScaler(feature_range=(-1, 1))
  y_normaliser.fit(unscaled_y[:n])

  y_normalised = y_normaliser.transform(unscaled_y)

  data_normaliser  = MinMaxScaler(feature_range=(-1, 1))
  data_normaliser.fit(dataset[:n+rolling_window])

  data_normalized = data_normaliser.transform(dataset)

  X_normalised = np.array([data_normalized[i : i + rolling_window] for i in range(len(data_normalized) - rolling_window - days_ahead)])

  # split data into train and test-sets
  X_train, y_train = X_normalised[:n], y_normalised[:n]
  X_test, y_test = X_normalised[n:], y_normalised[n:]

  return X_normalised, y_normalised, X_train, y_train, X_test, y_test, unscaled_y[n:], y_normaliser
```

6.8 A Function for Creating a Model for Tuning Hyperparameters

```
def create_model_for_hyperparameter_tuning(smoothed_data, n_epoch, n_epoch1, batch_size, optimizer, learning_rate,
```

```
                    gru_neuran1, dropout_rate1, node2, activation_fun2, dropout_rate2,
activation_fun3):
  callback = EarlyStopping(monitor="loss", patience=20, mode="min")
# prepare model
  model = Sequential()
  model.add(GRU(units=gru_neuran1, input_shape=(None, smoothed_data.shape[1])))
  model.add(Dropout(dropout_rate1))
  model.add(Dense(node2, activation=activation_fun2))
  model.add(Dropout(dropout_rate2))
  model.add(Dense(1, activation=activation_fun3))
  model.compile(optimizer=optimizer(lr=learning_rate),                    loss='mse',
metrics=[tf.keras.metrics.MeanAbsoluteError(name='MAE')])

  # Initial fitting model and evaluate over test set
  MAE_TEST = []
  m = int(len(smoothed_data)/2)

  _,    _,    X_train,    y_train,    X_test,_,    unscaled_y_test,    y_normaliser    =
split_train_test(smoothed_data[:m])
  history = model.fit(x=X_train, y=y_train, batch_size=batch_size, epochs=n_epoch,
                shuffle=False, verbose=0, callbacks=[callback])
  y_test_predicted = model.predict(X_test)
  y_test_predicted = y_normaliser.inverse_transform(y_test_predicted).reshape(-1)
  MAE_TEST.append(np.abs(unscaled_y_test - y_test_predicted).mean())
  #visualize_history(history)

  # fitting model over a few more data and evaluate over new test set through for loop
  i = 1
  while m+i*days_ahead < len(smoothed_data)-days_ahead:

    _,    _,    X_train,    y_train,    X_test,_    ,    unscaled_y_test,    y_normaliser    =
split_train_test(smoothed_data[:m+i*days_ahead])
    model.fit(x=X_train[-days_ahead:],    y=y_train[-days_ahead:],    batch_size=batch_size,
epochs=n_epoch1,
            shuffle=False, verbose=0, callbacks=[callback])
    y_test_predicted = model.predict(X_test)
    y_test_predicted = y_normaliser.inverse_transform(y_test_predicted).reshape(-1)
    MAE_TEST.append(np.abs(unscaled_y_test - y_test_predicted).mean())
    i += 1

  return MAE_TEST

def run(repeats, batch, lr, unit1, unit2, act2, drop1, drop2):
  rolling_window = 28
  MAEs = []
  for i in range(repeats):
```

```
    MAE_TEST = create_model_for_hyperparameter_tuning(smoothed_deaths, n_epoch=1000,
n_epoch1=100, batch_size=batch,
                                    optimizer=optimizers.Adam, learning_rate=lr,
                                    gru_neuran1=unit1, dropout_rate1=drop1,
                                    node2=unit2, activation_fun2=act2, dropout_rate2=drop2,
                                    activation_fun3="tanh")
    MAEs.append(np.mean(MAE_TEST))
  return MAEs

batch_size = [100, 300]
learning_rates = [5e-3, 1e-3]

units = [20, 80]
act_funs = ['tanh', 'selu']
dropouts = [.1, .5]

results6 = pd.DataFrame()
for b in batch_size:
    print(b)
    for l in learning_rates:
        for u1 in units:
            for u2 in units:
                for a2 in act_funs:
                    for d1 in dropouts:
                        for d2 in dropouts:
                            results6[str([b,l,u1,u2,a2,d1,d2])] = run(repeats = 10, batch=b, lr=l,
                                                    unit1=u1, unit2=u2, act2=a2, drop1=d1, drop2=d2)

axes6 = results6.boxplot(grid=True, figsize=(21,5), fontsize=14, rot=90)
axes6.set_title('Distribution of 10 MAEs for 128 Different Hyperparametere Combinations',
fontsize=16)
axes6.set_xlabel("[Batch Size, Learning Rate, Number of GRU Nodes, Activation Function,
Dropout Rate for GRU, Number of Dense Nodes, Dropout Rate for Dense]", fontsize=16)
axes6.set_ylabel("MAE", fontsize=16)
plt.show()

axes6 = results6.iloc[:,:32].boxplot(grid=True, figsize=(21,5), fontsize=14, rot=90)
axes6.set_title('Distribution of 10 MAEs for Different Hyperparametere Combinations',
fontsize=16)
axes6.set_xlabel("[Batch Size, Learning Rate, Number of GRU Nodes, Number of Dense Nodes,
Activation Function, Dropout Rate for GRU, Dropout Rate for Dense]", fontsize=16)
axes6.set_ylabel("MAE", fontsize=16)
plt.show()

axes6 = results6.iloc[:,32:64].boxplot(grid=True, figsize=(21,5), fontsize=14, rot=90)
```

```
axes6.set_title('Distribution of 10 MAEs for Different Hyperparametere Combinations', fontsize=16)
axes6.set_xlabel("[Batch Size, Learning Rate, Number of GRU Nodes, Number of Dense Nodes, Activation Function, Dropout Rate for GRU, Dropout Rate for Dense]", fontsize=16)
axes6.set_ylabel("MAE", fontsize=16)
plt.show()

axes6 = results6.iloc[:,64:96].boxplot(grid=True, figsize=(21,5), fontsize=14, rot=90)
axes6.set_title('Distribution of 10 MAEs for Different Hyperparametere Combinations', fontsize=16)
axes6.set_xlabel("[Batch Size, Learning Rate, Number of GRU Nodes, Number of Dense Nodes, Activation Function, Dropout Rate for GRU, Dropout Rate for Dense]", fontsize=16)
axes6.set_ylabel("MAE", fontsize=16)
plt.show()

axes6 = results6.iloc[:,96:].boxplot(grid=True, figsize=(21,5), fontsize=14, rot=90)
axes6.set_title('Distribution of 10 MAEs for Different Hyperparametere Combinations', fontsize=16)
axes6.set_xlabel("[Batch Size, Learning Rate, Number of GRU Nodes, Number of Dense Nodes, Activation Function, Dropout Rate for GRU, Dropout Rate for Dense]", fontsize=16)
axes6.set_ylabel("MAE", fontsize=16)
plt.show()
```

6.9 Create the Best LA GRU Model

```
def create_model(smoothed_data, node1, drop1, node2, act2, drop2, act3, opt, learn, batch, epoch):
  callback = EarlyStopping(monitor="loss", patience=20, verbose=1, mode="min")
  # prepare model
  model = Sequential()
  model.add(GRU(units=node1, input_shape=(None, smoothed_data.shape[1])))
  model.add(Dropout(drop1))
  model.add(Dense(units=node2, activation=act2))
  model.add(Dropout(drop2))
  model.add(Dense(units=1, activation=act3))

  model.compile(optimizer=opt(lr=learn), loss='mse', metrics=[tf.keras.metrics.MeanAbsoluteError(name='MAE')])
  #model.summary()

  X_normalised, y_normalised, X_train, y_train, _,_ , _, y_normaliser = split_train_test(smoothed_data)

  model.fit(x=X_train, y=y_train, batch_size=batch, epochs=epoch, shuffle=False, verbose=0, callbacks=[callback])
```

```
  y_predicted = model.predict(X_normalised)
  # These transforms are inverted on forecasts to return them into their original scale before calculating an error score.
  y_predicted = y_normaliser.inverse_transform(y_predicted).reshape(-1)

  return y_predicted

days_ahead = 14
rolling_window = 28
yhat = create_model(smoothed_deaths, node1=80, drop1=.5, node2=20, act2="tanh", drop2=.1, act3="tanh",
                    opt=optimizers.Adam, learn=.005, batch=100, epoch=1000)
mae_GRU_Train = np.abs(yhat[:-days_ahead] - smoothed_deaths.iloc[rolling_window+days_ahead:-days_ahead,target]).mean()
mae_GRU_Test = np.abs(yhat[-days_ahead:] - smoothed_deaths.iloc[-days_ahead:,target]).mean()
print("MAE of GRU Model for Tain set is %.3f and for Test set is %.3f." %(mae_GRU_Train, mae_GRU_Test))
visualize(smoothed_deaths, yhat, False, _, _, _)
```

6.10 Define a list of All Unique Combinations of Counties

```
def define_a_list_of_all_unique_combinations_of_counties(counties):
  all_combinations = []
  for r in range(len(counties) + 1):
      combinations_object = combinations(counties, r)
      combinations_list = list(combinations_object)
      for i, item in enumerate(combinations_list):
          if target_name not in item:
              item += (target_name,)
              combinations_list[i] = item
      all_combinations += combinations_list
  #print(all_combinations)

  unique_combin_counties = list(set( [ tuple(sorted(i)) for i in all_combinations]))
  unique_combin_counties.sort(key=lambda x: len(x))

  return unique_combin_counties
```

6.11 Build a 3-Dimensional Rolling Window Array (R) and Split Train and Test Sets, and scale them for E-Bagging

```
def scale_Ebagging(resampled_smoothed_data):

  target              = resampled_smoothed_data.columns.get_loc(target_name)
  unscaled_y          = np.array([resampled_smoothed_data.iloc[rolling_window + days_ahead : -
days_ahead, target].copy()]).reshape(-1, 1)

  # Transform the data to have a specific scale. Specifically, to rescale the data to [-1,1] to meet
the tanh activation function of the GRU model.
  y_normaliser        = MinMaxScaler(feature_range=(-1, 1))
  y_normalised        = y_normaliser.fit_transform(unscaled_y)

  data_normaliser     = MinMaxScaler(feature_range=(-1, 1))
  data_normalized     = data_normaliser.fit_transform(resampled_smoothed_data)

  X_normalised        = np.array([np.array(data_normalized[i : i + rolling_window].copy()) for i
in range(len(data_normalized) - rolling_window - days_ahead )])

  idx                 = np.random.choice(np.arange(len(y_normalised)), len(y_normalised),
replace=True)
  resampled_X_normalised = X_normalised[:-days_ahead][idx]
  resampled_y_normalised = y_normalised[idx]

  return X_normalised, resampled_X_normalised, resampled_y_normalised, y_normaliser
```

6.12 Create Residual Predictor Network

```
def custom_loss(y_true, y_pred):
    y_pred = K.max(tf.concat([y_pred, tf.zeros_like(y_pred)], axis=1) , axis=1)
    y_pred = tf.reshape(y_pred, tf.shape(y_true)) + 1e-17
    loss = K.sum( K.log(y_pred) + y_true / y_pred ) / 2
    return loss
#############################################################
def create_second_model(smoothed_data, yhat_Ebagging, yhat_var):
    callback = EarlyStopping(monitor="loss", patience=100, mode="min", verbose=0)
    # prepare model
    model = Sequential()
    model.add(GRU(units=200, input_shape=(None, smoothed_data.shape[1])))
    model.add(Dropout(.1))
    model.add(Dense(units=100, activation="tanh"))
    model.add(Dropout(.1))
```

```
    model.add(Dense(1, activation="exponential"))
    model.compile(optimizer=optimizers.Adam(lr=5e-3),  loss=custom_loss)
    ##########################################################################
    dataset = smoothed_data.values
    r_squared = np.maximum((y - yhat_Ebagging)**2 - yhat_var, np.zeros(len(y))).reshape(-1, 1)
    n = len(r_squared) - days_ahead

    # Transform the data to have a specific scale. Specifically, to rescale the data to [-1,1] to meet the tanh activation function of the GRU model.
    r_squared_normaliser = MinMaxScaler()
    r_squared_normaliser.fit(r_squared[:n])
    r_squared_normalised = r_squared_normaliser.transform(r_squared)

    data_normaliser  = MinMaxScaler(feature_range=(-1, 1))
    data_normaliser.fit(dataset[:n+rolling_window])
    data_normalized = data_normaliser.transform(dataset)

    X_normalised = np.array([data_normalized[i : i + rolling_window] for i in range(len(data_normalized) - rolling_window - days_ahead)])

    # split data into train and test-sets
    X_train, r_squared_train = X_normalised[:n], r_squared_normalised[:n]
    ##########################################################################
    model.fit(x=X_train, y=r_squared_train, batch_size=100, epochs=1000, shuffle=True, verbose=0, callbacks=[callback])

    r_squared_predicted = model.predict(X_normalised);
    # These transforms are inverted on forecasts to return them into their original scale before calculating an error score.
    r_squared_predicted = r_squared_normaliser.inverse_transform(r_squared_predicted).reshape(-1)

    return r_squared, r_squared_predicted
```

6.13 Create E-Bagging Models

```
def create_model_Ebagging(resampled_smoothed_data):

  X_normalised, resampled_X_normalised, resampled_y_normalised, y_normaliser = scale_Ebagging(resampled_smoothed_data)

  callback = EarlyStopping(monitor="loss", patience=50, mode="min", verbose=0)
  # prepare model
  model = Sequential()
  model.add(GRU(units=80, input_shape=(None, resampled_X_normalised.shape[2])))
  model.add(Dropout(.5))
```

```
  model.add(Dense(units=20, activation="tanh"))
  model.add(Dropout(.1))
  model.add(Dense(1, activation="tanh"))
  model.compile(optimizer=optimizers.Adam(lr=5e-3),  loss='mse')

  model.fit(x=resampled_X_normalised,        y=resampled_y_normalised,        batch_size=100,
epochs=1000,
          shuffle=False, verbose=0, callbacks=[callback])

  y_predicted = model.predict(X_normalised);
  # These transforms are inverted on forecasts to return them into their original scale before
calculating an error score.
  y_predicted = y_normaliser.inverse_transform(y_predicted).reshape(-1)

  return y_predicted
##############################################################################
def Ebagging(smoothed_data, counties, y_predicted, iterations):

  unique_com          = define_a_list_of_all_unique_combinations_of_counties(counties)
  Ebagging_y_predicteds = []

  for iter in range(iterations):
    q                  = randrange(1, len(unique_com))
    resampled_smoothed_data = smoothed_data[list(unique_com[q])]
    pred               = create_model_Ebagging(resampled_smoothed_data)
    Ebagging_y_predicteds.append(pred)

  yhat_Ebagging = np.mean(Ebagging_y_predicteds, axis=0)
  yhat_var      = np.var(Ebagging_y_predicteds, axis=0)
  MAE_Ebagging                                =                np.abs(yhat_Ebagging                -
smoothed_deaths.iloc[rolling_window+days_ahead:, target]).mean()

  return yhat_Ebagging, yhat_var, MAE_Ebagging
##############################################################################
yhat_Ebagging, yhat_var, MAE_Ebagging = Ebagging(smoothed_deaths, counties, yhat,
iterations=128)
##############################################################################
ci = 0.95

z_critical = norm.ppf(1-(1-ci)/2)

lower_lim = yhat_Ebagging - z_critical * np.sqrt(yhat_var + r_squared_predicted)

upper_lim = yhat_Ebagging + z_critical * np.sqrt(yhat_var + r_squared_predicted)

y = smoothed_deaths.iloc[rolling_window+days_ahead:, target].values

visualize(smoothed_deaths, yhat, True, yhat_Ebagging, lower_lim, upper_lim)
```

```
mae_GRU_Train = np.abs(yhat[:-days_ahead] - y[:-days_ahead])).mean()
mae_GRU_Test = np.abs(yhat[-days_ahead:] - y[-days_ahead:]).mean()
print("MAE of GRU Model for Tain set is %.3f and for Test set is %.3f." %(mae_GRU_Train, mae_GRU_Test))

mae_Ebagging_GRU_Train = np.abs(yhat_Ebagging[:-days_ahead] - y[:-days_ahead]).mean()
mae_Ebagging_GRU_Test = np.abs(yhat_Ebagging[-days_ahead:] - y[-days_ahead:]).mean()
print("MAE of Extended Bagging of  GRU Model for Tain set is %.3f and for Test set is %.3f." %(mae_Ebagging_GRU_Train, mae_Ebagging_GRU_Test))

print("Predicted value for 14 days ahead =", yhat_Ebagging.cumsum()[-1])

coverage_prob = round(np.count_nonzero((lower_lim < y) & (upper_lim>y))/len(y),3)
print("coverage_Probability =", coverage_prob)
```

6.14 Create Error Distribution

```
fig = go.Figure()
fig.add_trace(go.Histogram(
   x=yhat - smoothed_deaths.iloc[rolling_window+days_ahead:,target],
   name='Prediction Error distribution'))
fig.add_trace(go.Histogram(
   x=yhat_Ebagging - smoothed_deaths.iloc[rolling_window+days_ahead:,target],
   name='Extended bagging Prediction Distribution' ))
fig.update_layout(legend=dict(x=0.01, y=.99),
                  title={'text': "Forecasting Error Distribution",'y':0.9, 'x':0.5,
                      'xanchor': 'center', 'yanchor': 'top'},
                xaxis_title="Error",
                yaxis_title="Percent",
```

```
        barmode='overlay', # Overlay both histograms
        height=600, width=1000)
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.4)
fig.show()
```

7 References

[1] W. H. Organization, "Archived: WHO Timeline - COVID-19," 2020. [Online]. Available: https://www.who.int/news/item/27-04-2020-who-timeline---covid-19.

[2] K. E. Fullerton, E. K. Stokes, L. D. Zambrano, K. N. Anderson, E. P. Marder, K. M. Raz, S. El Burai Felix, Y. Tie and K. E. Fullerton, "Coronavirus Disease 2019 Case Surveillance — United States, January 22–May 30, 2020," *Center of disease control and prevention,* 2020.

[3] "Coronavirus (COVID-19) statistics data," [Online]. Available: https://www.google.com/search?rlz=1C1SQJL_enUS810US810&sxsrf=ALeKk02NbfkORId-jFXuKR4i8cu4F9bR7w%3A1607048370539&ei=spzJX7KvIMP29AP90L-4Cg&q=covid+map+usa&oq=covid+map+usa&gs_lcp=CgZwc3ktYWIQAzIICAAQsQMQyQMyBQgAELEDMgIIADICCAAyAggAMgIIADICCAAyAggAMgIIADICC.

[4] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics,* vol. 5, no. 4, p. 115–133, 1943.

[5] N. Rochester, J. Holland, L. Habit and W. Duda, "Tests on a cell assembly theory of the action of the brain, using a large digital computer," *IRE Transactions on Information Theory,* vol. 2, no. 3, pp. 80-93, 1956.

[6] F. Rosenblatt, "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain," *Psychological Review,* vol. 65, no. 6, p. 386–408, 1958.

[7] M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry, MIT Press, 1969.

[8] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, no. 6088, p. 533–536, 1986.

[9] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[10] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[11] Y. Bengio, "Learning Deep Architectures for AI," *Machine Learning,* vol. 2, no. 1, pp. 1-27, 2009.

[12] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing,* vol. 25, no. 2, p. 2447–2455, 2012.

[13] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," *The Annals of Statistics,* vol. 7, no. 1, pp. 1-26, 1979.

[14] B. Efron and R. Tibshirani, "Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy," *Statistical science,* pp. 54-75, 1986.

[15] G. Paass, "Assessing and Improving Neural Network Predictions by the Bootstrap Algorithm," *NIPS,* 1992.

[16] J. G. Carney, P. Cunningham and U. Bhagwan, "Confidence and prediction intervals for neural network ensembles," *Neural Networks,* 1999.

[17] R. Tibshirani, "A comparison of some error estimates for neural network models," *Neural Computation,* vol. 8, pp. 152-163, 1996.

[18] T. Heskes, "Practical confidence and prediction intervals," *In Advances in neural information processing systems,* pp. 176-182, 1997.

[19] A. Khosravi , S. Nahavandi, D. Creighton and A. F. Atiya, "Comprehensive review of neural network-based prediction intervals and new advances," *IEEE Transactions on neural networks,* vol. 22, no. 9, pp. 1341-1356, 2011.

[20] D. H. Mantzaris, G. C. Anastassopoulos and D. K. Lymberopoulos, "Medical disease prediction using Artificial Neural Networks," in *2008 8th IEEE International Conference on BioInformatics and BioEngineering*, Athens, Greece, 2008.

[21] X. Zhang, H. Zhao, S. Zhang and R. Li, "A Novel Deep Neural Network Model for Multi-Label Chronic Disease Prediction," *Frontiers in Genetics,* vol. 10, p. 351, 2019.

[22] Y. Ren, H. Fei, X. Liang, D. Ji and M. Cheng , "A hybrid neural network model for predicting kidney disease in hypertension patients based on electronic health records," *BMC Medical Informatics and Decision Making,* vol. 19, 2019.

[23] L. Wang, J. Li, S. Guo and N. Xie, "Real-time estimation and prediction of mortality caused by COVID-19 with patient information based algorithm," *Total Environment,* vol. 727, 2020.

[24] S. Gupta, G. S. Raghuwanshi and A. Chanda, "Effect of weather on COVID-19 spread in the US: A prediction model for India in 2020," *The Total Environment,* vol. 728, 2020.

[25] Z. Ceylan, "Estimation of COVID-19 prevalence in Italy, Spain, and France," *The Total Environment,* vol. 729, 2020.

[26] A. S. Ahmar and E. B. del Val, "SutteARIMA: Short-term forecasting method, a case: Covid-19 and stock market in Spain," *The Total Environment,* vol. 729, 2020.

[27] D. Fanelli and F. Piazza, "Analysis and forecast of COVID-19 spreading in China, Italy and France," *Chaos, Solitons & Fractals,* vol. 134, 2020.

[28] A. Chande, S. Lee, M. Harris, Q. Nguyen, S. J. Beckett, T. Hilley, C. Andris and J. S. Weitz, "Real-time, interactive website for US-county-level COVID-19 event risk assessment," *Nature Human Behaviour,* vol. 4, pp. 1313-1319, 2020.

[29] . Y. Zhou, L. Wang, . L. Zhang, L. Shi and K. Yang, "A Spatiotemporal Epidemiological Prediction Model to Inform County-Level COVID-19 Risk in the United States," *Harvard Data Science Review,* 2020.

[30] M. Mehta, J. Julaiti, P. Griffin and S. Kumara, "Early Stage Machine Learning–Based Prediction of US County Vulnerability to the COVID-19 Pandemic: Machine Learning Approach," *JMIR Public Health Surveill.,* no. 3, 2020.

[31] A. Y. Ives and C. Bozzuto, "Estimating and explaining the spread of COVID-19 at the county level in the USA," *Communications Biology,* vol. 4, no. 1, 2020.

[32] M. Wieczorek, a. Siłka and M. Woźniak, "Neural network powered COVID-19 spread forecasting model," *Chaos, Solitons & Fractals,* vol. 140, 2020.

[33] A. I. Saba and A. H. Elsheikh, "Forecasting the prevalence of COVID-19 outbreak in Egypt using nonlinear autoregressive artificial neural networks," *Process Saf Environ Prot,* vol. 141, pp. 1-8, 2020.

[34] V. K. R. Chimmula and L. Zhanmg, "Time series forecasting of COVID-19 transmission in Canada using LSTM networks," *Chaos Solitons Fractals,* vol. 135, 2020.

[35] P. Arora, H. Kumar and B. Panigrahi, "Prediction and analysis of COVID-19 Positive cases using deep learning models: a descriptive case study of India," *Chaos, Solitons and Fractals,* vol. 10.1016, 2020.

[36] S. Shastri, K. Singh, S. Kumar, P. Kour and V. Mansotra, "Time series forecasting of Covid-19 using deep learning models: India-USA comparative case study," *Chaos, Solitons and Fractals,* vol. 140, pp. 1-10, 2020.

[37] F. Shahid, A. Zameer and M. Muneeb, "Predictions for COVID-19 with deep learning models of LSTM, GRU and Bi-LSTM," *Chaos, Solitons and Fractals,* vol. 140, p. 110212, 2020.

[38] "USAFACT," [Online]. Available: https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/.

[39] "List of the most populous counties in the United States," [Online]. Available: https://en.wikipedia.org/wiki/List_of_the_most_populous_counties_in_the_United_States.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint,* vol. arXiv:1412.6980, 2014.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. 15, no. 56, p. 1929−1958, 2014.

[42] Y. Bengio, P. Simard and P. Frasconi, "Learning Longterm Dependencies with Gradient Descent is Difficult," *IEEE Trans. Neural Networks,* vol. 5, no. 2, pp. 157-166, 1994.

[43] G. Zhou, J. Wu, C. Zhang and Z. Zhou, "Minimal Gated Unit for Recurrent Neural Networks," *International Journal of Automation and Computing,* vol. 13, p. 226–234, 2016.

[44] L. Breiman, "Bagging predictors," *Machine Learning,* vol. 24, p. 123–140, 1996.

[45] P. Hall and R. L. Samworth, "Properties of bagged nearest neighbor classifiers," *Journal of the Royal Statistical Society, Series B,* vol. 67, no. 3, p. 363–379, 2005.

[46] B. M. Steele, "Exact bootstrap k-nearest neighbor learners," *Machine Learning,* vol. 74, p. 235–255, 2009.

[47] T. Hastie, R. Tibshirani and J. Friedman, The elements of statistical learning, New York: Springer, 2001.

[48] L. Breiman, "Random forests," *Machine Learning,* vol. 45, p. 5–32, 2001.

[49] T. E. Oliphant, A guide to NumPy,, vol. 1, Trelgol Publishing USA, 2006.

[50] W. McKinney, "Data Structures for Statistical Computing in Python," *Proceedings of the 9th Python in Science Conference,* pp. 51-56, 2010.

[51] F. Chollet and others, "Keras," *GitHub repository,* 2015.

[52] M. Abadi and Others, "TensorFlow: a System for Large-Scale Machine Learning on Heterogeneous Systems," *12th {USENIX} Symposium on Operating Systems Design and Implementation (OSDI),* vol. 16, pp. 265-283, 2016.

[53] G. a. D. J. F. L. Van Rossum, Python tutorial, Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[54] F. a. V. G. a. G. A. a. M. V. a. T. B. a. G. O. a. B. M. a. P. P. Pedregosa, "Scikit-learn: Machine Learning in Python}," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[55] G. Papadopoulos, P. J. Edwards and A. F. Murray, "Confidence estimation methods for neural networks: A practical comparison," *IEEE transactions on neural networks,* vol. 12, no. 6, pp. 1278-1287, 2001.

[56] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," *IEEE TRANSACTIONS ON SIGNAL PROCESSING,* vol. 45, no. 11, pp. 2673-2681, 1997.

[57] G. Ganssle, "Neural networks," *The Leading Edge,* vol. 37, no. 8.

[58] "Rolling-Window Analysis of Time-Series Models," [Online]. Available: https://www.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html.

[59] "Tracking COVID-19 in California," [Online]. Available: https://covid19.ca.gov/state-dashboard/.

[60] D. Hebb, The Organization of Behavior: A Neuropsychological Theory publish, Psychology Press, 1949.

[61] G. E. Hinton, S. Osindero and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation,* vol. 18, pp. 1527-1554, 2006.

[62] J. Rocca, "A gentle journey from linear regression to neural networks," 2018. [Online]. Available: https://towardsdatascience.com/a-gentle-journey-from-linear-regression-to-neural-networks-68881590760e.

[63] H. R. Niazkar and . M. Niazkar, "Application of artificial neural networks to predict the COVID-19 outbreak," *Global Health Research and Policy,* vol. 5, 2020.