DEEP-LEARNING: $PM_{2.5}$ CONCENTRATIONS WITH BI-LSTM, GRU, AND TCN

by

ADOLPHE ANSA SOME

(Under the Direction of Dr. Lixin Li)

ABSTRACT

Fine particle matter ($PM_{2.5}$) is a pollutant particulate matter with diameter less than 2.5 micrometer. There exist many stations installed in the world to measure its concentration. Some areas without any proper equipment nor any installation must rely on interpolation techniques to approximate its concentration. So, there is a need of interpolation technique to approximate the concentration of pollutant in those areas. The faster and more accurate interpolation technique can help identify more polluted areas and thus efficiently take some measures to reduce $PM_{2.5}$ harmful effects.

We used three different neural networks models to estimate the concentration of $PM_{2.5}$ over the southeast region of the USA. We explored two data preprocessing techniques and the effects of spatiotemporal correlation on the models. We finally compared all models and made a choice on the model that is more appropriate for estimating the concentration of $PM_{2.5}$

INDEX WORDS: $PM_{2.5}$, Interpolation, Deep neural network, Bi-LSTM, GRU, TCN, k-NN

DEEP-LEARNING: $PM_{2.5}$ CONCENTRATIONS WITH BI-LSTM, GRU, AND TCN

by

ADOLPHE ANSA SOME

B.S., Allen E. Paulson College of Engineering and Computing, Statesboro, GA 30458

A Project Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

GEORGIA SOUTHERN UNIVERSITY

DEEP-LEARNING: $PM_{2.5}$ CONCENTRATIONS WITH BI-LSTM, GRU, AND TCN

by

ADOLPHE ANSA SOME

Committee Chair:	Dr. Lixin Li
Committee:	Dr. Daniel Liang
	Dr. Hong Zhang
	Dr. Weitiang Tong

Electronic Version Approved:
May 2021

DEDICATION

I dedicate this project to my honorable professors, my friends, my people overseas that always worry about me.

ACKNOWLEDGMENTS

I would like to thank Dr. Lixin Li without whom this project was not going to take place. I would also like to thank Dr. Hong Zhang, Dr. Daniel Y. Liang and Dr. Weitiang Tong who accepted to be members of the committee. Thank you to all the professors and staffs in the CS department of Statesboro and Armstrong Campus for all the care that I received during all those years.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

1. B. C. Durak, *Artificial Neural Networks*, Accessed: 2018-03-27.

2. Weitian Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN," *Air Quality, Atmosphere & Health* 12 (April 2019): 1–13, https://doi.org/10.1007/s11869-018-0647-4.

3. Weitian Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN," *Air Quality, Atmosphere & Health* 12 (April 2019): 1–13, https://doi.org/10.1007/s11869-018-0647-4.

CHAPTER 1

INTRODUCTION

This introduction first explains the motivation of choosing such a topic. It gives some examples to illustrate the arguments by referencing our research history on this topic. Secondly, it briefly explains the contribution of the work done on this paper.

1.1 MOTIVATION

$PM_{2.5}$ particles are particulate matters with a diameter smaller than 2.5 μm. There are indoor and outdoor $PM_{2.5}$ particles. The $PM_{2.5}$ indoor particles originate from indoor burning activities such as smoking, cooking, operating fireplaces, and fuel-burning heaters.[1] The outdoor $PM_{2.5}$ particles are the most common. They also originate from all burning activities in the environment such as all vehicles and machines with fuel-based engines, industries, wildfires, and even volcanic eruptions.[2] Most particles are formed in the atmosphere as a result of complex reactions of chemicals such as sulfur dioxide and nitrogen oxides, which are pollutants emitted from power plants, industries, and automobiles.[3] They have been studied and proven to be responsible for many cardiovascular problems such as irregular heartbeat, aggravated asthma, decreased lung function, increase respiratory symptoms as irritation of the airways, coughing or difficulty breathing[4][5].[6]

1. John D. Spengler and Ken Sexton, "Indoor air pollution: a public health perspective," *Web of Science.* (New York, NY, USA) 221, no. 4605 (July 1983), https://doi.org/10.1145/3161192, https://doi.org/10.1145/3161192.

2. Commissioner Howard Zucker M.D., "Fine Particles (PM 2.5) Questions and Answers," *NY State Department of Health* 1, no. 1 (2018): 1–3, https://www.health.ny.gov/environmental/indoors/air/pmq_a.htm.

3. Howard Zucker.

4. Spengler and Sexton, "Indoor air pollution: a public health perspective."

5. Howard Zucker, "Fine Particles (PM 2.5) Questions and Answers."

6. J. Jason West and Co, "What We Breathe Impacts Our Health: Improving Understanding of the Link between Air Pollution and Health," *Environmental Science & Technology 50 (10), 4895-4904*, 2016, https://doi.org/10.1021/acs.est.5b03827.

To monitor and reduce the $PM_{2.5}$ pollution, many countries have installed several monitoring stations which measure the pollutants such as $PM_{2.5}$ particles concentrations in an area. The obtained data can be later analyzed and lead to make decisions against $PM_{2.5}$ pollution. The United States Environmental Protection Agency(EPA) established Nation Ambient Air Quality Standards for $PM_{2.5}$ since 1990 and each US State has multiple monitoring sites which measure the air quality, and such data is used to determine the pollution for the areas with a monitoring site.[7]

Unfortunately, many polluted places are not equipped with monitoring stations. Sometimes, places with monitoring stations don't record any data for months. To fill the missing or non-recorded data, many interpolation techniques are used.

Interpolation techniques can estimate the unknown concentration of $PM_{2.5}$ particles at some spatial locations for a specific time.

Spatial interpolations can be grouped in two types. The first type is point interpolation(based on field data) and area interpolation (based on entity data).[8] Point interpolations can be further divided into two sub-parts which are the exact point interpolation and approximate point interpolation. The most popular exact point interpolation techniques are Inverse Distance Weighting, kriging and shape functions[9].[10]

7. US Environmental Protection Agency and William K. Reilly, "State Implementation Plans; General Preamble for the Implementation of Title I of the Clean Air Act Amendments of 1990," November 1990, https://www.epa.gov/criteria-air-pollutants/naaqs-table.

8. Cromley Merwin David and Co, "A Neural Network-based Method for Solving "Nested Hierarchy" Areal Interpolation Problems," *Cartography and Geographic Information Science* 36 (March 2013): 347–365, https://doi.org/10.1559/152304009789786335.

9. Lixin Li et al., "Fast Inverse Distance Weighting-Based Spatiotemporal Interpolation: A Web-Based Application of Interpolating Daily Fine Particulate Matter PM2.5 in the Contiguous U.S. Using Parallel Programming and k-d Tree," *International journal of environmental research and public health* 11 (September 2014): 9101–9141, https://doi.org/10.3390/ijerph110909101.

10. Lixin Li and Peter Revesz, "A Comparison of Spatio-temporal Interpolation Methods" (September 2002), 145–160, https://doi.org/10.1007/3-540-45799-2_11.

Spatio-temporal methods incorporate time and space simultaneously by using the known spatiotemporal measurements in the interpolation algorithm[11][12].[13]

Neural networks are classified as approximate point interpolation techniques because they approximate the true values by incorporating multiples features or parameters whether they are continuous or discrete features from spatial or temporal entities. The interpolation based on time in neural networks is called time series forecasting.

1.2 CONTRIBUTION

Our contribution is divided into the following parts.

The first part is to explain the theory behind the neural network models used and the reason why they are used.

In the second part, we explain the origin and basis of the project and explore some alternatives to the interpolation problem.

The third part is to develop two datasets by using two data pre-processing techniques and then apply the k-nearest neighbors algorithm to each dataset.

The forth part is to build three neural network models following the same chart.

The fifth part is to test each neural network for each dataset, and then compare the models based on the results.

---

11. Lixin Li and Peter Revesz, "Interpolation methods for spatio-temporal geographic data," (New York, NY, USA), 2004, https://doi.org/10.1016/S0198-9715(03)00018-8, https://www.elsevier.com/locate/compenvurbsys.

12. Lixin Li et al., "Estimating Population Exposure to Fine Particulate Matter in the Conterminous U.S. using Shape Function-based Spatiotemporal Interpolation Method: A County Level Analysis," *GSTF international journal on computing* 1 (September 2015): 24–30.

13. Peter Revesz, "Spatiotemporal Interpolation Algorithms" (January 2014), 1–5, https://doi.org/10.1007/978-1-4899-7993-3_803-2.

CHAPTER 2

RELATED WORKS

This project is based on a research[1] with other research papers that served as inspiration in the design of new neural network models.

2.1 BACKGROUND

Li and Co[2] proposed a spatiotemporal technique that takes in account the relationship between each monitoring site and its k-nearest neighbors and also between each monitoring site and its own measurements at different days. Such neural network can estimate the concentration of pollutants at multiples sites based on multiple variables or features such as space and time.

There exist many neural networks models for spatial analysis. Vector autoregression models[3] as well as clustering techniques could be used to achieve a similar result but our focus is on applying a k-d tree algorithm to capture the relationship between k-nearest neighbors. This algorithm's advantage is that it structures the data in a form of tree model which speeds up the process of finding the nearest neighbors of each location. Applying the k-nearest neighbors yields improvements on the estimations.[4] Li and Co then implemented two recurrent neural network models which are the LSTM and Bi-LSTM and performed three experiments.

The first experiment sets up the network structure. The number of k-nearest neighbors is fixed to k=1 and the number of influential days is fixed to t=1. The neural networks were

1. Weitian Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN," *Air Quality, Atmosphere & Health* 12 (April 2019): 1–13, https://doi.org/10.1007/s11869-018-0647-4.

2. Tong et al.

3. Guokun Lai et al., "Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks," March 2017,

4. Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN."

trained with and without cross validation at several dropouts from 0 to 0.30. The *MAE*, *RMSE* and *MAPE* errors were recorded.

The second experiment tested several values of influencing neighbors and days. The results were recorded and compared by *MAPE* errors for each number of neighbors and then for each number of influencing days.

The third and last experiment compared the LSTM and Bi-LSTM using the results of the two previous experiments.

We observed that the data pre-processing technique is efficient but the neural network models used overfitted which led to the use of 10 fold cross validation. So we looked for different research papers to apply different techniques to solve the problem.

One research we read, is about using a hybrid of wavelet transform, stacked autoencoder and LSTM to forecast the Concentration of $PM_{2.5}$.[5] The model takes the dataset which goes through a wavelet decomposition to separate the data based on one layer of features. Each partial data then go through a combination of stacked encoder-LSTM which again extracts another layer of features. All partial data is merge at the end by wavelet reconstruction to obtain the forecast results of $PM_{2.5}$. Wavelet transform and stacked autoencoder are both feature extractors mostly use for image processing in deep learning. Such a model is efficient when the dataset has multiple correlated features such as $PM_{2.5}$, wind direction, strength, temperature but it fails to take in account the spatiotemporal correlation as needed in our project. So, this technique was not implemented for the project.

Another research used linear interpolation, spearman's rank-order correlation, LSTM and stacked LSTM, Bi-LSTM to estimate the $PM_{2.5}$ concentration.[6] The dataset has 9

5. Weibiao Qiao et al., "The Forecasting of PM2.5 Using a Hybrid Model Based on Wavelet Transform and an Improved Deep Learning Algorithm," *IEEE Access* PP (September 2019): 1–1, https://doi.org/10.1109/ACCESS.2019.2944755.

6. Preeti Bamane and Mangal Patil, "INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY A comparative study of different LSTM neural networks in predicting air pollutant concentrations," *Indian Journal*

features. They started the data preprocessing by filling the missing data with some dummy data. After that, the data is normalized, and spearman's rank-order correlation is used to find a monotonic relationship between the features. After that, the dataset is trained through an LSTM, a stacked LSTM and Bi-LSTM to estimate the concentration of $PM_{2.5}$. The method here again fails to capture the spatiotemporal correlation in the dataset, and it focuses more in finding the correlation between the features. So, this technique was also not implemented.

One research used an ensemble technique to estimate the concentration of $PM_{2.5}$ particles.[7] The ensemble technique is a machine learning method that combines different base models to produce one optimal predictive model.[8] The model called APNet is an ensemble of CNN-LSTM model. The dataset has two features: wind speed and rain. The data preprocessing was barely mentioned. Only the data normalization was mentioned, and the correlation of the features was not taken in account. The model was trained with a stacked of 3 layers of CNN followed by a LSTM and a fully connected dense layer. A batch normalization is done between the second and third CNN layer and also between the last CNN and LSTM layer. The activation function of the CNN models is a scaled exponential linear units and a sigmoid function in the dense layer. The ensemble gives a good estimate of the $PM_{2.5}$ concentration of the next 1 *hour*. The model is inspiring because its goal is to improve the accuracy, but it lacks the information about the data preprocessing and the spatiotemporal correlation.

Another research is using an ensemble neural networks of *k*-nearest neighbor-LSTM to predict a traffic flow. The data preprocessing was not mentioned. The model is designed

of Science and Technology 13 (November 2020): 3664, https://doi.org/10.17485/IJST/v13i35.1276.

7. Chiou-Jye Huang and Ping-Huan Kuo, "A Deep CNN-LSTM Model for Particulate Matter (PM2.5) Forecasting in Smart Cities," *Sensors* 18 (July 2018): 2220, https://doi.org/10.3390/s18072220.

8. Zhi-Hua Zhou and Wei Tang, "Selective Ensemble of Decision Trees," vol. 2639 (April 2003), https://doi.org/10.1007/3-540-39205-X_81.

with a first k-nearest neighbors neural network layer which is used to extract the k related stations based on their spatiotemporal correlation. After that, the data go through k-LSTM layers and then joined before output. The accuracy is better than using a single LSTM.The $k-NN$-LSTM model is taking in account the spatiotemporal correlation as in the research of Li and Co[9] but the application domain is different. It's also possible to use a $k-NN$ model with a different algorithm other than a k-d tree or use a different distance measurement other than the Euclidean distance. This method gave us an inspiration as to how we could feed the data to a neural network model and experiment on it.

Another research is using a combination of LSTM and CNN to do forecast multiple variables such as traffic and electricity.[10] The research paper focused on developing a neural network which could make better assimilation and prediction of non-related variable such as traffic and electricity. They combined multiple neural networks as an ensemble of CNN-LSTM with a linear autoregressive model. This design is similar to our model in the goal of capturing the relationship between different variable inputs. The difference is in the datasets and data preprocessing. Our model is made to perform well on spatiotemporal data.

Perhaps the most inspiring document is from a book.[11] This book mentions some of the biggest problems of sequential neural network models which is how to find the best sequence structure of the data. They mentioned problems such as spatiotemporal features detection problems, the static representation of the input and the problem of timing, when to present the input and output pattern. They also mentioned problems such as when "Local encoding is restricted, random representations lack systematicity, and the feature-based representations are limited and rather artificial. All those representations keep Neural Net-

9. Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN."

10. Lai et al., "Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks."

11. Jain L-C, "Recurrent neural networks: Design and Applications" (Boca Raton, FL: CRC Press, 2001), 391.

works away from real data". They also mentioned some data preprocessing techniques that could be wrongly used such as presenting the patterns immediately or using a delay. Those techniques are not very effective. They proposed a feature engineering for data preprocessing and a recurrent associate neural network model that is "an auto-associative task in which the input sequence starts to be reproduced after the whole input sequence has been represented to the input, followed by a unique pattern, a trigger, indicating the end of the sequence". They also talked about the ordering of the data and also mentioned another paper[12] which "compared training recurrent networks with a lengthening input, ordering, to training the same recurrent networks with a random ordering of strings".

12. Guo Zheng Das S Giles, "Using Prior Knowledge in a NNPDA to Learn Context-Free Languages," (College Park, MD), 1993, 5.

CHAPTER 3

METHODS

We want to estimate the concentration of $PM_{2.5}$ particles at any monitoring sites for any time using *three* different neural networks models. If we suppose there are n different monitoring stations $\{S_1, \ldots, S_n\}$ in an area $\mathscr{A}$. The observation from each station S_i at a specific time stamp t can be described as a tuple $_{i,t} = (lon_i, lat_i, t, v_i)$, where v_i is the observed air pollutant concentration, lon_i and lat_i describe the longitude and latitude of the station S_i, respectively. Therefore, the input data set can be denoted as n time series, $\{ts_1, \ldots, ts_n\}$. Each time series $ts_i = (_{i,1}, \ldots, _{i,T})$ is the sequence of observed data at a single station S_i.[1]

Estimating v for any position in $\mathscr{A}$ at any time is called multivariate time series forecasting. There are multiple time series forecasting methods that can be applied to air pollutants predictions and can be grouped in two main categories.

- *Deterministic methods*:[2]

  These techniques can be considered model-based methods since their architecture is based on certain theoretical assumptions, and it is possible to calculate through precise prior knowledge of their parameters. Example of these methods are k-nearest neighbors and Decision Tree Models.

- *Statistical methods*:[3]

  These methods do not use complex theoretical techniques but employ statistical based techniques to forecast air quality. The most broadly used strategies include logistic regression and multiple linear regression (MLR), the autoregressive moving

1. Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN."

2. MARCO MIGLIONICO, "A Deep Learning Framework for Air Pollution Forecasting and Interpolation" (July 2019), 14–15, https://doi.org/10027/23862.

3. MIGLIONICO.

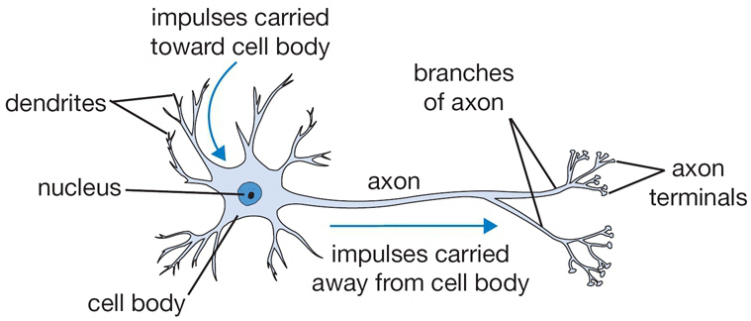

Figure 3.1: A cartoon drawing of a biological neuron (Durak 2018)[4]

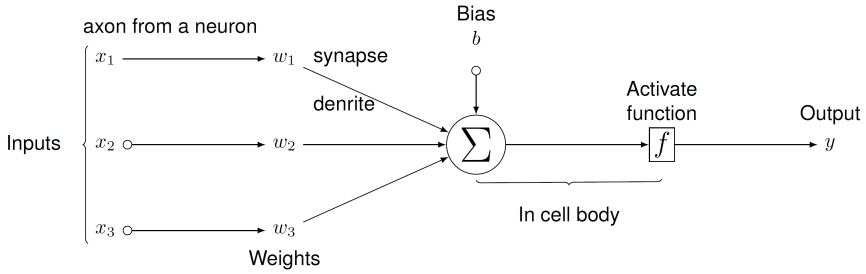

Figure 3.2: Mathematical model of neuron (Weitian 2018)[5]

average (ARMA),the support vector regression (SVR) and the artificial neural network (ANN). In particular, ANN methods have shown to be self-adaptive and robust for the time series prediction task, thanks to neural networks' ability to perform nonlinear mapping.

3.1 Understanding of Neural Networks

Artificial Neural Network (ANN):

The *neurons* in a human's brain are the centerpiece of human intelligence. An artificial neural network in its basic form is an imitation of the *neuron* in the human brain (fig:3.1). The neuron's *dendrites* and *axons* are equivalent to the artificial neuron's *synapse* while the *signal* a neuron receives and transmits are similar to the *inputs* and *outputs* on the artificial neuron (fig:3.2). The goal of imitating the human's neuron is to allow (train) the

artificial neuron network to perform some human-like tasks. A basic ANN consists of 3 layers. One layer of inputs is similar to human's neuron signals that are received from other neurons. A second layer which is the *neuron* itself. And the last layer is the layer of output. In practice, an ANN could have multiple *inputs* (in the input layer), multiples *neurons* (in the neuron's layer) that form a hidden layer and then more hidden layers and finally an *output* layer of multiples *outputs*. Such complicated neural networks are referred to as *deep neural networks*. In an ANN, a user can determine the *inputs* of the neural network and each *synapse* has a strength or *weight*. Such *weight* determines how important the *input* is. It allows the neural network to learn. In an ANN, each *input* goes through a process in the *synapse* call *activation function*. The *activation function* in a neuron serves in the regulation of how much of the input information should be output.[6] The *activation function* converts linear input to non-linear output. There are many types of *activation functions* used in ANN and some common activation functions are the *sigmoid* activation function, the *threshold* function, the *rectifier* function, and the *hyperbolic tangent* function and mostly keep the signal either -1 and 1 or 0 and 1. So the learning process of an ANN is to learn the *synaptic weights* and to control the strength of influence of one neuron to another. In a basic ANN with multiple hidden layers or *feed forward neural network* (FNN), the inputs do not change. The neural network learns through the *loss function*. *Loss function* helps figure out the performance of the model in prediction. It computes the error for every training by comparing the output to the desired result and feeds back ($back-propagates$) the signal to the network and adjusts the *weights* for a better prediction. The network keeps adjusting the *weights* on the *synapses* until the *predicted output* is closer to the *true output*. The process in which the network keeps feeding the end data back to the network and adjusts the weighted synapses between the inputs and the neuron is called $back-propagation$. It allows the signal to flow backwards through the network, in

6. Bengio Goodfellow and Courville, "Deep Learning Book," (Cambridge, Massachusetts), 2016,

order to compute the *gradient*.[7] A *gradient descent* is a time saving method which allows the neural network to find the optimal *weights* which will result in the best *output* possible. The *gradient descent* is used to perform the learning process of a neural network through a *gradient* technique.[8] There exist many *gradient descent* techniques such as the *batch gradient descent*, the *stochastic gradient descent* and the *mini batch gradient descent*. Our models are using the *stochastic gradient descent* which is a technique that considers the learning process just one example at a time to take a single step. It means that our models get their *weights* updated at every epoch instead of waiting to process the whole batch and then update the *weights* as in the case of *batch gradient descent*. An *epoch* is one cycle through the full training dataset.

Limitation of Feed forward Neural Networks

In general, FNN can't remember the sequential structure in a dataset. In a FNN, each hidden state or neuron works independently from the previous ones which means that it does not give any special treatment of what the previous inputs might have been. Such networks can't be used to train data where order and importance of the previous data matters.

3.2 Sequential Neural Networks

Figure (fig:3.3) shows multiple relations of sequential and recurrent neural networks such as Bi-LSTM and GRU. Sequential neural networks input and output the data in a form of sequence. A red rectangle represents an input, a green one represents a neural unit and a blue rectangle is an output.

In a $one-to-one$ relationship of a sequential neural network, the unit takes one input and produces one output. These operations are called $timesteps$. One $timestep$ corresponds

7. Goodfellow and Courville, "Deep Learning Book."
8. Goodfellow and Courville.

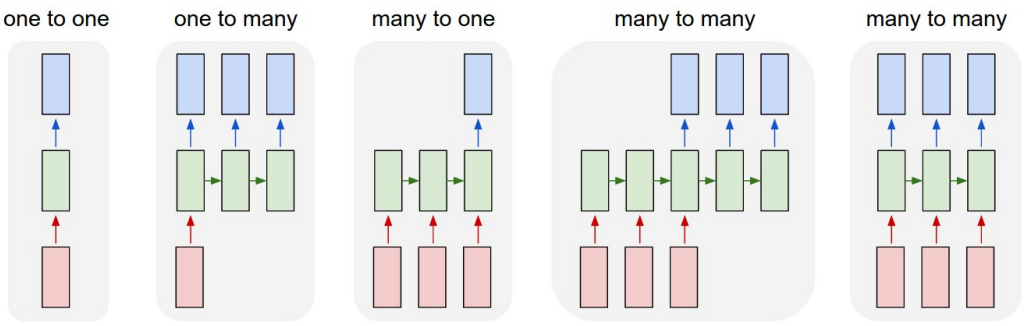

Figure 3.3: Relations of Sequential Neural Networks

to ticks of time when a sequential neural network unit gets one input and or produces one output. A $timestep$ is similar to 1 iteration where the neural network unit gets one input or gives one output. During one $timestep$, a sequential unit gets one input or gives away one output.

The second $one-to-many$ relationship takes one input sequence and gives three outputs. The number of green rectangles determines the number of $timesteps$.

In the figure of $many-to-one$, the neural unit(green rectangle further left) receives one input (red rectangle bottom left) at $timesteps\_1$ without giving any blue output at that $timesteps$ but it has some output for itself that will be used as input later in time. At $timesteps\_2$, it receives a different input from the data and also the input from its own previous state $timesteps\_1$ and still does not produce any output at the $timesteps\_2$. In the last and final step, the unit gets another data input with an input from its own previous state and finally gives an output. This relationship is the most convenient for our design because we want each unit to be trained on multiples sequences of our data. After that, we can make unique predictions based on multiples previous inputs and the recurrent unit will learn the relationship between the sequences of inputs.

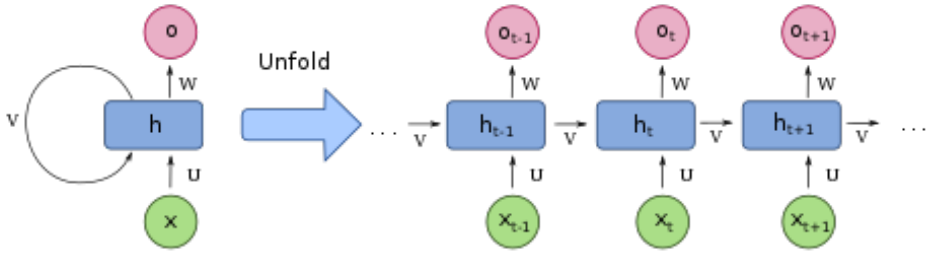

Figure 3.4: One to one Recurrent Neural Network

3.2.1 Recurrent neural networks (RNN)

A RNN is a development of FNN with a feedback (recurrence) to itself. While non-recurrent unit can not receive inputs from it previous state (timestep), a recurrent unit receives inputs from its previous timestep and outputs a feedback data for its next timestep (fig:3.4). RNN can encode dependencies between inputs but has a problem when handling long data sequences. When encoding long data dependencies, the back-propagation of the signal goes through multiples layers of neural networks. As the signal travels through more layers using certain *activation functions*, the gradients of the *loss function* increases exponentially or vanishes, making the neural network unable to learn. This is called the vanishing gradient problem. The simplest solutions are to use the right *activation functions* or perform a *batch normalization*. A *batch normalization* is a technique to standardize the data. It is believed to make the neural network stable and faster during the training.

Long short-term memory recurrent neural networks (LSTM):

LSTM networks are built to overcome problems associated with the long-term problems associated with the RNN.[9] It comprises of different gates. There is an input gate for the input layer, a forget gate, and an output gate.[10] The cell state and the gates are the core

9. Mike Schuster and Kuldip Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions on* 45 (December 1997): 2673–2681, https://doi.org/10.1109/78.650093.

10. Xianglong Luo et al., "Spatiotemporal traffic flow prediction with KNN and LSTM," *Journal of Ad-*

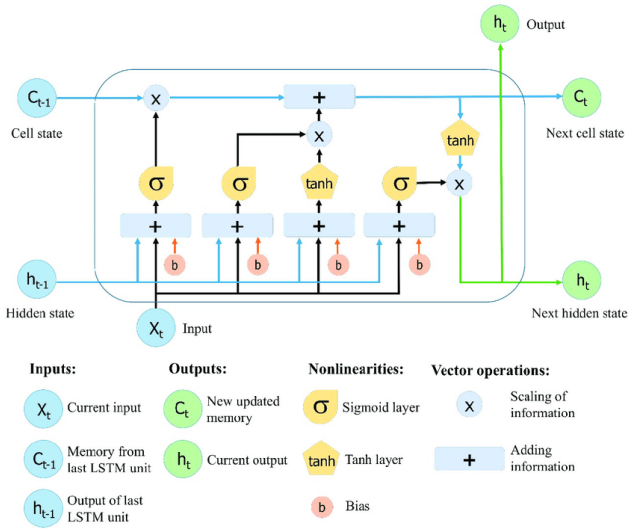

Figure 3.5: Long-Short term memory Unit

concept of LSTM. This is because the cell state allows for the transportation of relative information to the sequence chain. It serves as the memory of the network. The cell state makes it possible to store and transport relevant information throughout the processing of the sequence. Necessitating the availability of information from past steps and ensuring they get to later steps. Hence reducing the effects of short-term memory.[11] The cell state gets more information throughout its journey. Some information is also removed via the gates. The gates decide what to retain and what to forget in this journey.

Gated recurrent Unit (GRU):

GRU is a recurrent neural network similar to LSTM but lacks an output gate and has fewer parameters which aims to solve the vanishing gradient problem experienced in LSTM.[12] GRU only have hidden states. It also only has two gates, a reset gate, and an update gate (fig:3.7). The gates are regulating the flow of information flowing through

vanced Transportation 2019 (February 2019): 1–10, https://doi.org/10.1155/2019/4145353.

11. Huang and Kuo, "A Deep CNN-LSTM Model for Particulate Matter (PM2.5) Forecasting in Smart Cities."

12. Luo et al., "Spatiotemporal traffic flow prediction with KNN and LSTM."

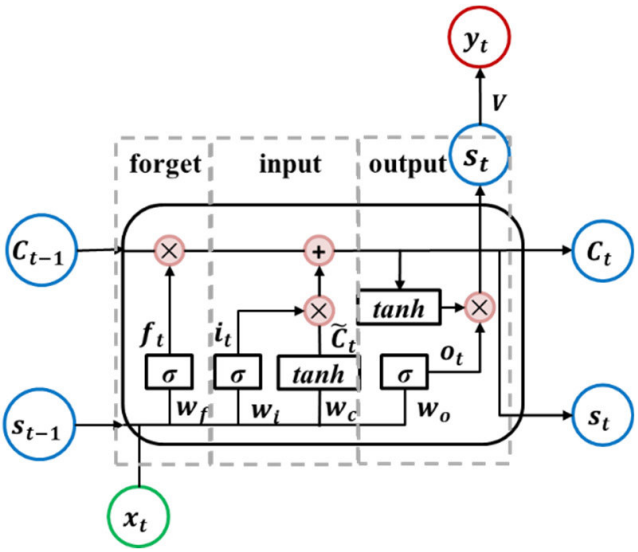

Figure 3.6: LSTM forget-input-output gates

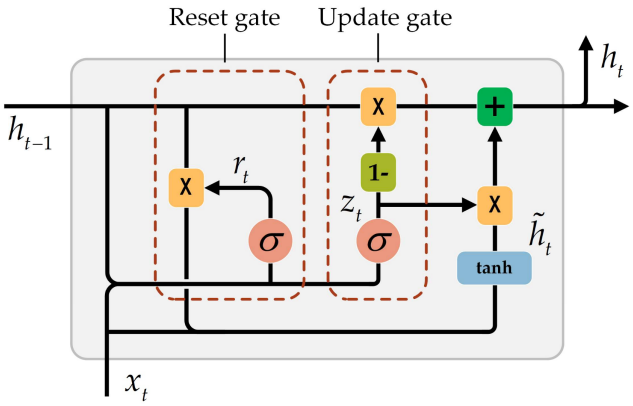

Figure 3.7: GRU gates

and that allow the GRU to solve the vanishing gradient problem of a standard RNN.[13] The update and reset gate are vectors deciding the information that goes through to the output.[14] Those gates store and filter the information. The update gate in the model is used to determine the degree of the past information from past steps to carry to the future. This means that it can copy relevant information from the past and get rid of the risk of the vanishing gradient. The reset gate decides how much of the past data is irrelevant and worth forgetting.[15] Both the update and reset gates use the same formula. The only difference is the weights and gates usage. The gates affect the final output of the model. The last step in the unit consists of a vector that transfers information to the network from the current unit. To transfer information, the update gate is needed because it is responsible for determining what to collect from the current memory content.[16] GRU eradicates the vanishing gradient problem because it basically does not wash out the new input every single time but rather stores the relevant information and passes it down to future-forward steps of the network. Having also fewer operations allows GRU to be faster to train compared to LSTM.

LSTM and GRU were created to address the issue of short-term memory in RNN.[17] Even though the RNN, LSTM and GRU use feed backs, they performance and usage are different. LSTM and GRU are used in deep learning applications such as speech recognition, natural language understanding, and speech synthesis but with a careful training, GRU can easily outperform a LSTM.

13. Yitian Chen et al., "Probabilistic Forecasting with Temporal Convolutional Neural Network," *Neurocomputing* 399 (March 2020), https://doi.org/10.1016/j.neucom.2020.03.011.

14. Huang and Kuo, "A Deep CNN-LSTM Model for Particulate Matter (PM2.5) Forecasting in Smart Cities."

15. Dario Rethage, Jordi Pons, and Xavier Serra, "A Wavenet for Speech Denoising" (April 2018), 5069–5073, https://doi.org/10.1109/ICASSP.2018.8462417.

16. Huang and Kuo, "A Deep CNN-LSTM Model for Particulate Matter (PM2.5) Forecasting in Smart Cities."

17. Schuster and Paliwal, "Bidirectional recurrent neural networks."

Bidirectional recurrent neural network(Bi-LSTM):

Bi-LSTM occurs when Long Short-Term Memory (LSTM) and Bi-directional Recurrent Networks (Bi-RNN) are combined. This structure makes it possible for networks to access backward and forward information on sequences at every level. The Bi-RNN can handle inputs information from both the back and the front. Bidirectional allows the two inputs to operate one from the future to the past and another from the past to the future. Recent years have seen an increase in approaches combining recommendation systems and deep learning.[18] Merging the LSTM and Bi-RNN increases the storage capability in LSTM cell memory and the access information abilities of Bi-RNN hence making the Bi-LSTM better.[19] Bi-LSTM ability to handle data with long-range dependence allows them to improve performance on sequence classification problems.[20] Figure (3.5) represent a LSTM unit.

3.2.2 Convolutional Neural networks (CNN):

A CNN is similar to a FNN with the difference that a CNN has one or more convolutional layers.[21] A convolutional layer is similar to a hidden layer of an FNN but it uses one or more filters. Filters have input weights and generate an output neuron. The goal of the convolutional layer is to extract features mostly from input image and preserve the spatial relationship by learning features using small squares of input data.[22] CNN were

18. Huang and Kuo, "A Deep CNN-LSTM Model for Particulate Matter (PM2.5) Forecasting in Smart Cities."

19. Schuster and Paliwal, "Bidirectional recurrent neural networks."

20. Chen et al., "Probabilistic Forecasting with Temporal Convolutional Neural Network."

21. Saad Albawi, Tareq Abed Mohammed, and Saad ALZAWI, "Understanding of a Convolutional Neural Network" (August 2017), https://doi.org/10.1109/ICEngTechnol.2017.8308186.

22. Navin Kumar Manaswi, "Deep Learning with Applications Using Python," (Cambridge, Massachusetts), 2018, 219–227.

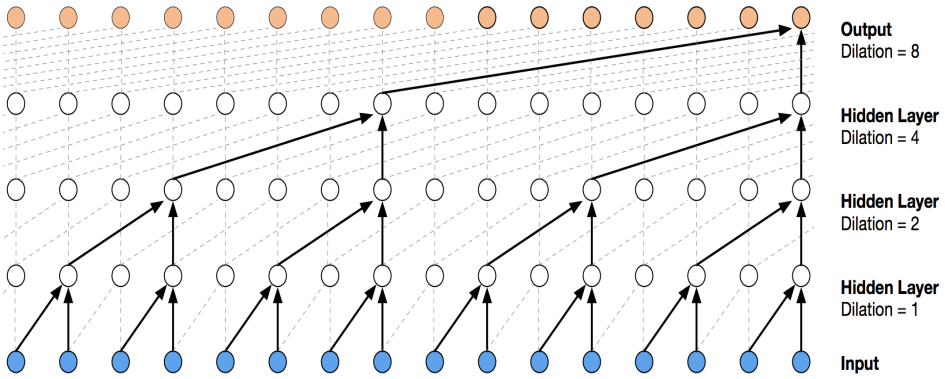

Figure 3.8: Temporal Convolutions

first employed to learn the correlation between image and sentence.[23] However, a variation of *CNN* called *Temporal Convolution Networks* (TCN) has been developed for sequence modelling tasks.

Temporal Convolutional Neural Networks (TCN):

TCN consist of dilated, causal 1*D* (one dimensional) convolutional layers used to convulse the output with the past elements of a sequence. This allows TCN to be effective in sequence predictions hence their utilization in weather predictions.[24] A 1*D* convolutional network takes as input a 3-dimensional tensor and also outputs a 3-dimensional tensor. One single 1*D* convolutional layer receives a unique input tensor and outputs a tensor of similar unique traits to ensure an output tensor has the same length as the input tensor, zero paddings could be applied.[25] In a forecasting model, the value of a specific entry in the output depends on all previous entries in the input. This is made possible when

23. Samuel Schulter et al., "Deep Network Flow for Multi-Object Tracking," June 2017,

24. Yaguang Li et al., "Graph Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting," July 2017,

25. Colin Lea et al., "Temporal Convolutional Networks for Action Segmentation and Detection" (July 2017), 1003–1012, https://doi.org/10.1109/CVPR.2017.113.

the receptive field has the same size input length. Most convolutional hidden layers end with a pooling layer whose job it is to distill the output of the last convolutional layer to the most important elements. Temporal convolutional networks do not have time steps. They treat the temporal data as a sequence over which convolutional read operations can be performed. TCN are combined with RNN in the segmentation of video-based action by filtering low-level features which are responsible for encoding spatial-temporal information and segregating features into a classifier capturing high-level temporal information using RNN. At least two convolutional layers are needed for video-based segmentation.[26] The TCN approach for catching two levels of information hierarchically is known as the encoder-decoder framework.[27]

We implemented a vanilla model for TCN day 1 estimation and later we stacked all the models. A vanilla model is a base model without hidden layers. Stacked neural networks are models where multiples based units are stacked together to make a deep neural network. Stacked models are used to train and extract features in complex data.

26. Colin Lea et al., "Temporal Convolutional Networks: A Unified Approach to Action Segmentation" (August 2016), https://doi.org/10.1007/978-3-319-49409-8_7.

27. Guirguis Karim et al., "SELD-TCN: Sound Event Localization & Detection via Temporal Convolutional Networks" (March 2020), https://doi.org/10.23919/Eusipco47968.2020.9287716.

CHAPTER 4

EXPERIMENTAL DATASET

The dataset is a daily measurement of $PM_{2.5}$ in 2009 over the contiguous southeast region of the U.S., i.e. Florida. This data set was measured by the *U.S. EPA's Air Quality System* (*AQS*) monitoring sites and it describes the weather conditions at several locations.[1] There are 53 sites and each site has different numbers of observations of $PM_{2.5}$. The dataset has 5 columns which are the site id, the daily timestamp for each measurement, the latitude and longitude and the daily concentration $PM_{2.5}$ measurement (fig:4.1). Some sites don't have any measurement for some of the 365 days. The objective is to modify this data to obtain two different formats of the same original data. Then, use both datasets to train three neural networks models to predict the concentration of $PM_{2.5}$ at multiple days of the year. For this research, the prediction have been done for 1 to 6 days.

4.1 Feature Engineering techniques

Most neural networks performance greatly depend on the feature engineering or the data processing technique. The data processing is usually divided in two steps. The first step is the data preprocessing during which, the data is prepared and structured to be fitted

1. United States Environmental Protection Agency, "Outdoor Air Quality Data," 2009, https://www.epa.gov/outdoor-air-quality-data/download-daily-data.

	id	year_month_day	longitude	latitude	pm25
0	120010023	12/30/2009	-82.387778	29.706111	7.3
1	120010023	12/27/2009	-82.387778	29.706111	6.5
2	120010023	12/27/2009	-82.387778	29.706111	6.6
3	120010023	12/24/2009	-82.387778	29.706111	5.5
4	120010023	12/18/2009	-82.387778	29.706111	2.6
...	...	...	...	...	...
19469	121290001	1/16/2009	-84.161111	30.092500	6.9
19470	121290001	1/4/2009	-84.161111	30.092500	6.1
19471	121290001	1/4/2009	-84.161111	30.092500	6.3
19472	121290001	1/1/2009	-84.161111	30.092500	5.3
19473	121290001	1/1/2009	-84.161111	30.092500	5.3

[19474 rows x 5 columns]

Figure 4.1: Original Data

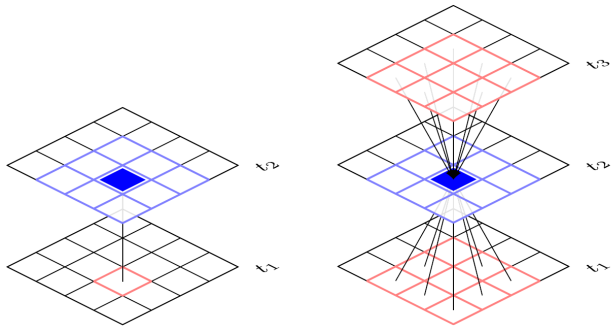

Figure 4.2: A brief illustration of the spatial and temporal correlations[4]

to the neural networks models. Each neural network needs a sequential data to be trained with. There exist multiple data preprocessing technique dealing with multivariate multi-step time series forecasting and that raises many questions about the best data prepocessing techniques..[2] We experimented on data ordering. Data ordering in specific sequences could lead to a better accuracy or speed up the learning process, but not all data ordering could be efficient in all cases and all models. Sometimes, a good data ordering could lead to bad predictions while wrong data ordering could lead to a better result.[3]

4.2 Data preprocessing

There is one original data used with two preprocessing techniques to create two datasets. The goal of the first preprocessing technique was to obtain a similar dataset as Li and Co[5] so that we can test our own data preprocessing and neural networks models with it and see how well our new models and techniques perform. The common procedure to obtain both

2. Neal Wagner et al., "Intelligent techniques for forecasting multiple time series in real-world systems," *International Journal of Intelligent Computing and Cybernetics* 4 (August 2011), https://doi.org/10.1108/17563781111159996.

3. L-C, "Recurrent neural networks: Design and Applications."

5. Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN."

		x	y	pm25
year_month_day	id			
2009-12-27	121290001	2	2	2
2009-10-19	121290001	2	2	2
2009-10-28	121290001	2	2	2
2009-03-11	121290001	2	2	2
2009-12-09	121290001	2	2	2
...	...	...	...	...
2009-05-10	120010023	1	1	1
2009-05-07	120010023	1	1	1
2009-10-16	120010023	2	2	2
2009-11-03	120010023	1	1	1
2009-01-01	120010023	2	2	2

14645 rows × 3 columns

Figure 4.3: Duplicates in data

dataset 1 and *dataset* 2 is to implement the spatiotemporal technique in (fig:4.2). That's a method to query the k-nearest neighbors from space and also from time. We will use k-d tree algorithm and Euclidean distance in the method to calculate the nearest neighbors of each location to interpolate. The data preprocessing is divided as followed.

- *Describe the data:*

  This data has 19475 rows of 53 sites and 5 columns. Each row has and *id* per site, a *date* time of one record, a $longitude$, $latitude$ of the site and the concentration $PM_{2.5}$ at that site. The time measurement is between *January* 1*st* 2009 and *December* 31*st* 2009. Some sites have duplicates (fig:4.3) on some daily records and some sites don't have some daily records.

- *Remove duplicates:* The data grouped by *id* and time to show the total count of records for each site and each day. This grouping shows that some days have more

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19474 entries, 0 to 19473
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              19474 non-null  int64
 1   year_month_day  19474 non-null  object
 2   x               19474 non-null  float64
 3   y               19474 non-null  float64
 4   pm25            19474 non-null  float64
dtypes: float64(3), int64(1), object(1)
memory usage: 760.8+ KB
```

Figure 4.4: Data Info

than one record of $PM_{2.5}$ at one site. We need to preprocess the data to get only a single observation for each of the 365 days per site. The following *Pandas* function is applied: $df.drop\_duplicates(subset = ['id','year\_month\_day'], keep = 'last', inplace = True)$. Once all duplicates removed, each site ends up having at most 1 daily observation.

- *Fill the missing data:*

  There is no *NaN* (Not-a-Number) in the data (fig:4.4). Li and Co used the technique of removing the data with low entries which lead to a very low size of dataset and it made it hard to train the neural network. In our case, we have decided to test multiples datasets.

  The data grouped by each day of the year shows that some of the 53 sites don't record any data for some days. So we need here a technique to fill the missing data. Not filling the missing values can produce biased estimates, leading to invalid conclusions.

	year_month_day	pm25	id	x	y
0	1	6.200000	120010023	-82.391389	29.703333
1	2	7.426147	120010023	-82.391389	29.703333
2	3	7.426147	120010023	-82.391389	29.703333
3	4	8.200000	120010023	-82.391389	29.703333
4	5	7.426147	120010023	-82.391389	29.703333
...	...	...	...	...	...
11675	361	6.500000	121275002	-81.053056	29.206667
11676	362	7.426147	121275002	-81.053056	29.206667
11677	363	7.426147	121275002	-81.053056	29.206667
11678	364	5.100000	121275002	-81.053056	29.206667
11679	365	7.426147	121275002	-81.053056	29.206667

[11680 rows x 5 columns]

Figure 4.5: Dataset 1 of 32 sites with 365 days each

4.3 DATASET 1

The first dataset preprocessing is made to obtain the same dataset as Li and Co. To obtain such dataset, built a python code for data preprocessing. We first drop the duplicates, then grouped the dataset by id and kept the first 32 *ids* with the most daily measurements. After that, we calculate the mean for the $PM_{2.5}$. We then group the data by the site *ids* and iterate to compare the date in each group to the 365 *days* of the year. If a certain day is not present in the group of same *id*, we create a row with the missing day and complete the rest of the columns with the mean of $PM_{2.5}$. Such function is used to fill the missing days for all of our dataset. Once the dataset is filled, we apply our *k*-nearest neighbor algorithm to the whole dataset. The *k*-nearest neighbor is designed to find the *k*-nearest neighbors for each site and then reshape the nearest neighbors as features. The nearest neighbors algorithm was applied to have the nearest neighbors from 1 to 6 neighbors per site. Once the preprocessing is complete, we have 6 text files with 32 monitoring sites in each. Each site has 365 measurements. Each file has a different *k*-nearest neighbor applied to it (fig:4.5)

```
       year_month_day     pm25         id          x          y
0                   1  6.20000  120010023 -82.387778  29.706111
1                   2  8.42704  120010023 -82.387778  29.706111
2                   3  8.42704  120010023 -82.387778  29.706111
3                   4  8.20000  120010023 -82.387778  29.706111
4                   5  8.42704  120010023 -82.387778  29.706111
...               ...      ...        ...        ...        ...
19340             361  5.10000  121290001 -84.161111  30.092500
19341             362  8.42704  121290001 -84.161111  30.092500
19342             363  8.42704  121290001 -84.161111  30.092500
19343             364  5.90000  121290001 -84.161111  30.092500
19344             365  8.42704  121290001 -84.161111  30.092500

[19345 rows x 5 columns]
```

Figure 4.6: Dataset 2 of 53 sites with 365 days each

4.4 DATASET 2

This dataset 2 is obtained by keeping all the 53 original sites instead of dropping the sites with few measurements in 2009 as in dataset 1. This dataset was obtained by filling those missing values with the *mean* of the dataset. The missing values were also filled with the overall *mean* of $PM_{2.5}$ concentration. The *k*-nearest neighbors algorithm is also applied to this dataset and produced one file per *k*-nearest neighbor. This dataset also produced 6 other new dataset and adding all the original dataset, we have also 7 dataset for dataset 2. Once the preprocessing is done, we have 6 text files with 53 monitoring sites in each. Each site has 365 measurements and each file has been obtained by applying a unique *k*-nearest neighbor with each *k* value between 1 and 6 (fig:4.6)

CHAPTER 5

IMPLEMENTATION

We made a chart (fig:5.1) to show the flow of the whole implementation from obtaining the data until obtaining the final errors as result. When implementing a neural network, it is important to normalize and scale the data to avoid overfitting. In our case, it is even more important to normalize and scale the data between 0 and 1 because we are working with outliers and sigmoid activation function. The boxplot on (fig:5.2) shows that there are multiples outliers on the lower and upper bounds. We set up two types of experiments with each dataset. the first experiment is to see if a neural network model can improve based on the number of neighbors and the number of influencing days. For all neural networks implemented, each dataset was divided in three sets:

- *Training set:* 81%
- *Validation set:* 9%
- *Testing set:* 10%

Measure of Performance: Four (4) measured of performance were used. They are the mean absolute error *MAE*, the root-mean-square error *RMSE*, and the mean absolute percentage error *MAPE* and the mean-square error *MSE* calculated as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|O_i - P_i|$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(O_i - P_i)^2}$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{|O_i - P_i|}{O_i}$$

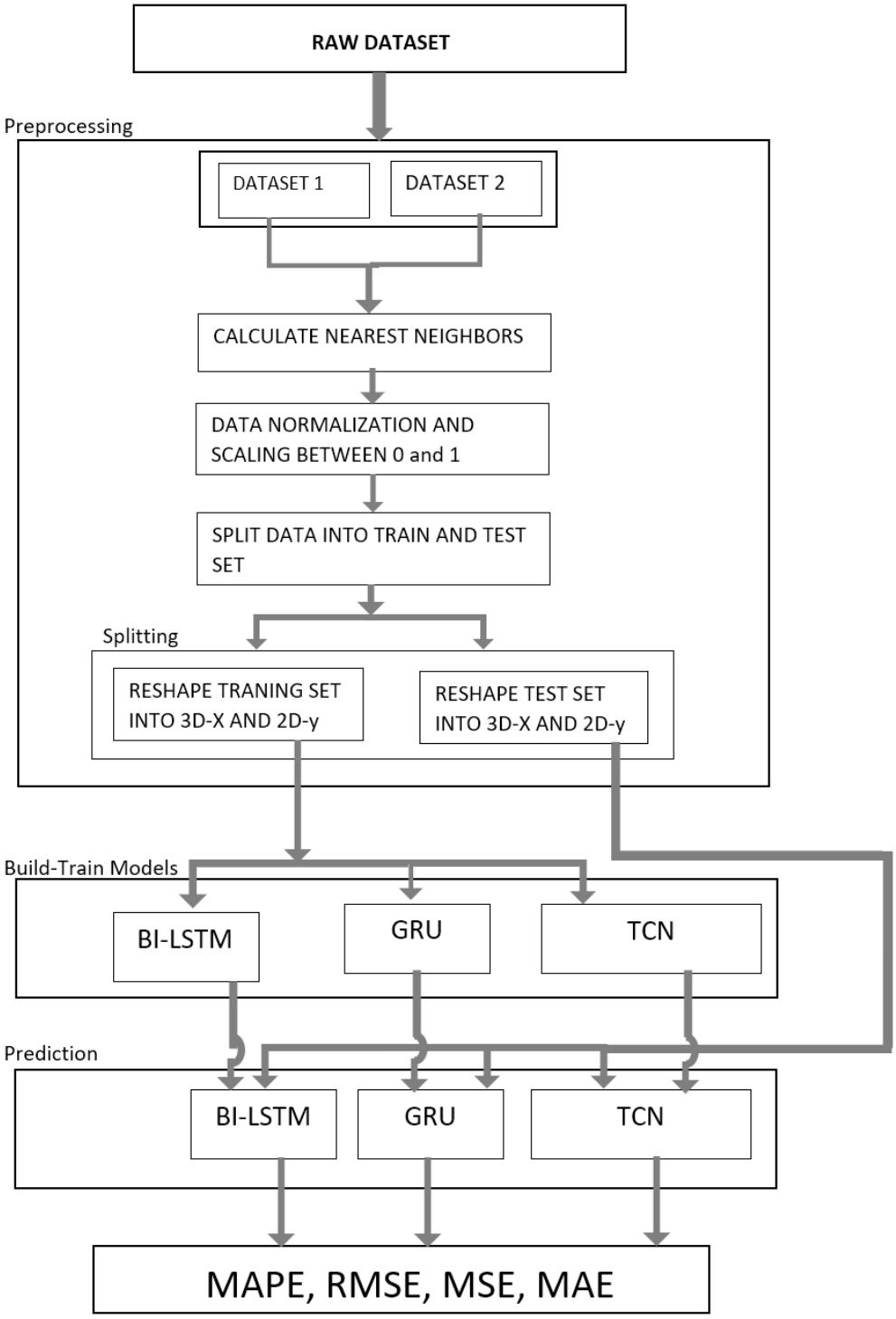

Figure 5.1: Flowchart of the Models

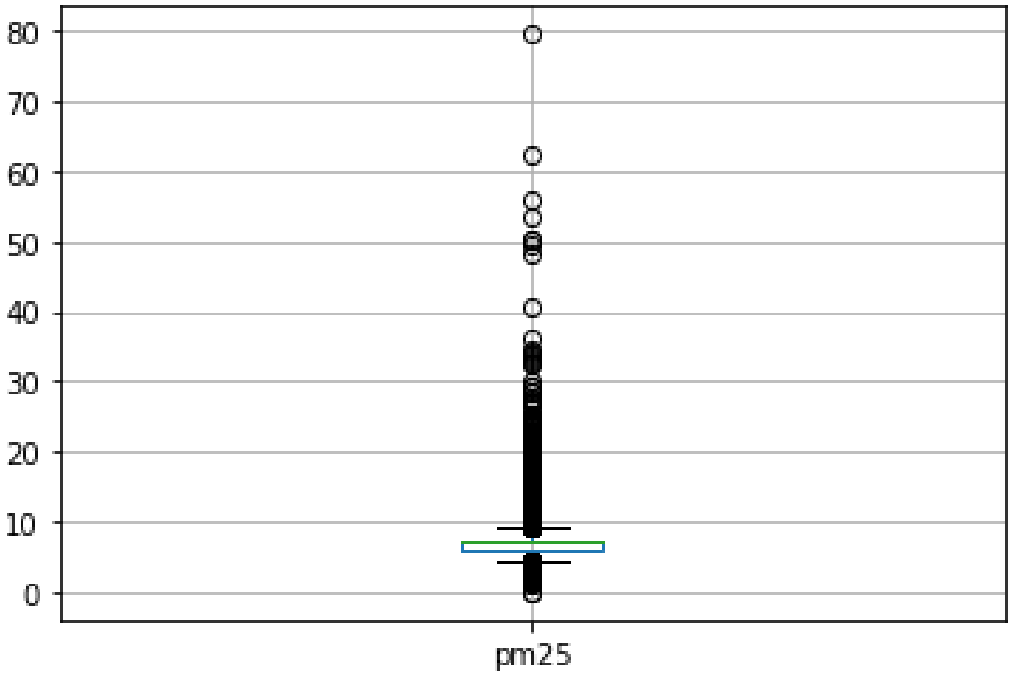

Figure 5.2: Boxplot with multiples outliers

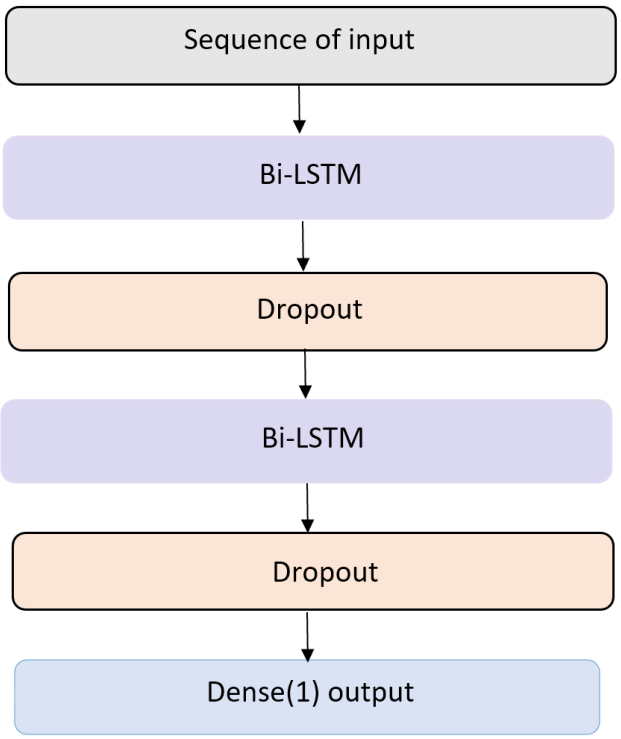

Figure 5.3: Long-Short term memory Unit

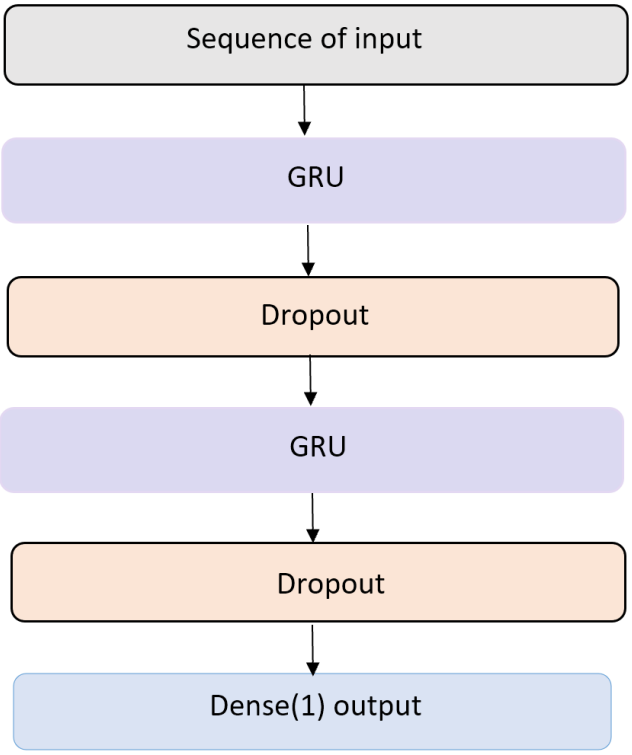

Figure 5.4: Gated Recurrent Unit

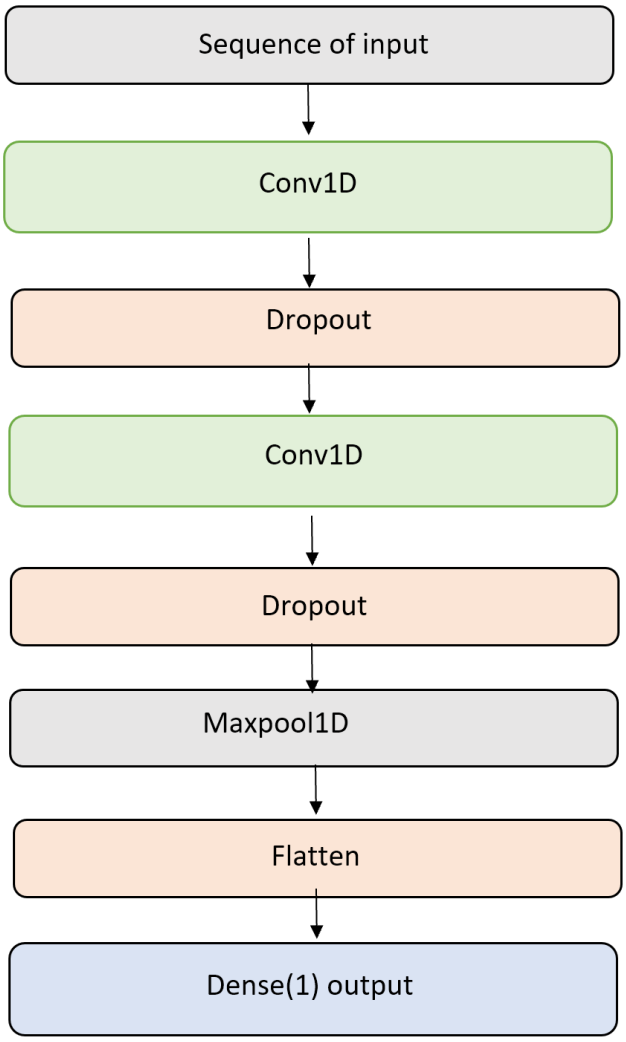

Figure 5.5: Long-Short term memory Unit

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(O_i - P_i)^2$$

N = Number of evaluation samples

O_i = Observed concentrations of particles

P_i = Predicted concentration of $PM_{2.5}$ particles

We designed one deep Bi-LSTM (fig 5.3) and one GRU (fig 5.4) recurrent neural networks following mostly the same parameters. Each of the two recurrent networks is stacked by 2 layers and finally one dense layer. We randomly and uniformly initialized the the kernel's weights using random uniform for both the Bi-LSTM and GRU models. We applied the sigmoid activation function defined as follow:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

We used MAPE as the $loss\ function$ and *adam* as an optimizer.

The TCN was built with one and later with two deep convolutional layers (fig: 5.5). Both convolutional layers have each a kernels size and a dilation rate that varies for each dataset. A $max-pool$ is added after the second convolutional layer to help reduce some feature dependencies. After the $max-pool$, the outputs are flattened to pass into a dense layer with one neuron as our output.

While training the neural networks, some of the models were overfitting as the performance on the training set was better than the validation set. To solve such problems, we added tested multiple techniques such as reducing the number of units, adding dropout and use the early stopping method which consists of terminating the training when the error reaches a certain threshold.

- *Experimental setup:*

  IDE: Spyder 4.1.5

Operating system: Windows 10

Processor: Intel(R) Core(TM) i7(R)-10510U CPU @ 1.80GHz

System memory: 16 GB DDR4

GPU: 2 GB NVIDIA GeForce MX250

Initial k = 1

Initial t = 1

$batch\_size$ = $\{4, 8, 16, 32, 64, 128, 256\}$

Number of neurons = $\{1, 8, 16, 32, 64, 128\}$

$Dropout = \{0.0, 0.1, 0.2, 0.3\}$

Number of filters = $\{4, 8, 16, 32, 64, 128\}$

CHAPTER 6

RESULTS

We tested the parameters as followed:

6.1 EXPERIMENT 1: NUMBER OF INFLUENCING DAYS AND NEIGHBORS

6.1.1 DATASET 1

- *Bi-Directional Long-Short term memory:*

  Find optimal batch size:

  A neural network is not trained with all the training data all at once. Instead, the data is divided into fixed *batches* which are fed sequentially to the layers. Small *batch* sizes are proven to make the training loss converge faster in few *epochs* while bigger batches can be processed in parallel hence are computational efficient.[1] To choose the optimal batch size, we tried several batch sizes and chose the optimal batch size based on the *MAPE*. The batch table: 6.1 shows the relations between each batch size and the resulting *MAPE*. The graph: 6.2 shows the evolution of the *MAPE* as a function of the *batch size*.

1. Aditya Devarakonda, Maxim Naumov, and Michael Garland, *AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks*, 2018, arXiv: 1712.02029 [cs.LG].

Batch Size	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.47166	2.626252	21.95836	6.897202	323.5052
8	1.462372	2.618509	22.27221	6.856591	171.4519
16	1.460063	2.618445	22.68553	6.856254	89.42714
32	1.462206	2.621673	22.95873	6.873168	51.70563
64	1.461789	2.621172	22.92425	6.870543	30.12648
128	1.46046	2.619546	22.79555	6.862022	19.59539
256	1.552644	2.690982	21.83746	7.241385	17.14775

Figure 6.1: Bi-LSTM Batch Table

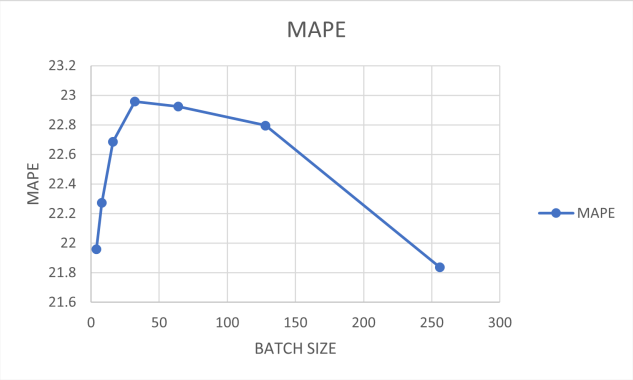

Figure 6.2: Bi-LSTM Batch Graph

Epochs	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.461844	2.621236	22.92868	6.870877	9.003222
8	1.695705	2.700662	27.72871	7.293575	11.34329
16	1.462735	2.618752	22.25132	6.857862	17.56079
32	1.460076	2.61849	22.69089	6.856487	29.36996
64	1.462337	2.621829	22.96914	6.873986	53.02854
128	1.462568	2.622095	22.98662	6.87538	94.30881
256	1.462714	2.622268	22.99774	6.876288	81.09458

Figure 6.3: Bi-LSTM Epochs Table

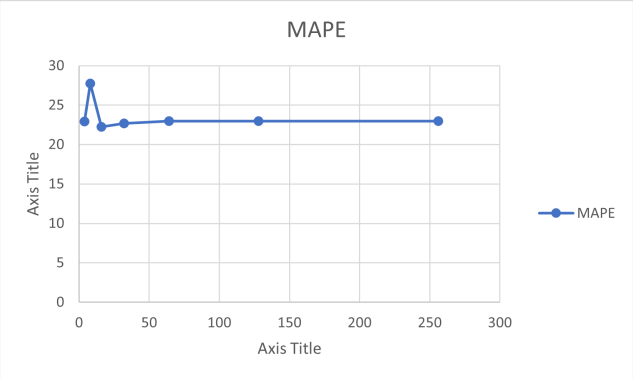

Figure 6.4: Bi-LSTM Epochs Graph

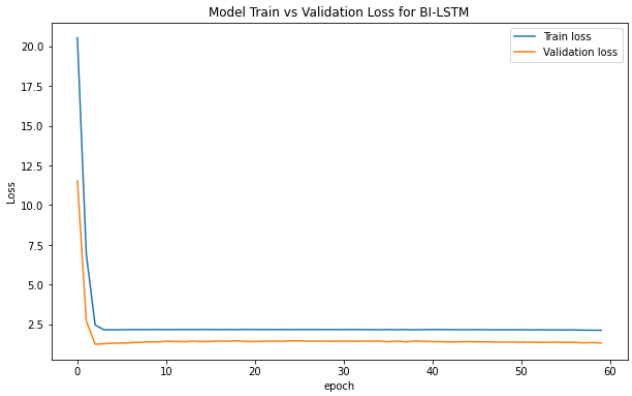

Figure 6.5: Bi-LSTM Loss-Validation Curves

Find optimal Number of epochs:

During one epoch, all the *batches* are used only once to train the neural network. If a neural network is trained on small number of cycle, it will not learn enough. So, we might think that we should use more epochs to learn more which might be the solution. While it is common sense to think that the more you train the better you learn; but that's not the case for neural networks. A neural network which learns too much will overfit meaning that it will learn to perform perfectly on the training data and perform poorly on the validation or testing data. So we must choose an optimal number of epochs to avoid overfitting. The table: 6.3 shows the result of training under different epochs. We then used the data in the *MAPE* column and epochs to make a graph to have a look on the *MAPE* evolution on different epochs. The graph: 6.4 shows the general trend of the number of epochs on the *MAPE*.

While testing the number of epochs, we must keep track on the validation and loss curve to make sure that the network does not overfit. The figure: 6.5 shows that the network is not overfitting. Overfitting is noticeable when the training curve crosses and goes below the validation curve.

We completed more tests and some of our first choice parameters were changing

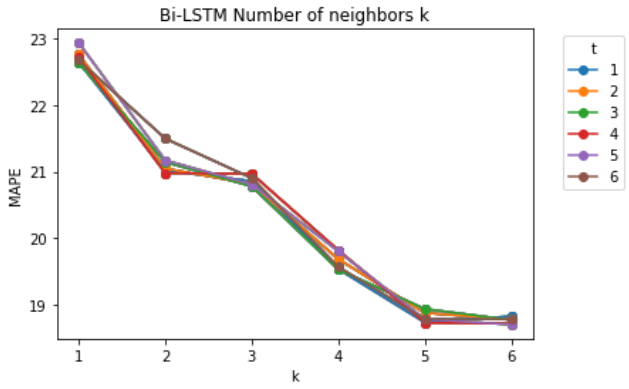

Figure 6.6: Bi-LSTM Number of influencing days

drastically per test. So we always picked the three parameters that lead to the three smallest *MAPE*. After that, we completed the same tests again for three more times. The parameter with the highest probability of getting a low *MAPE* is chosen. The final Bi-LSTM parameters that we chose for training on dataset 1 are the following:

Bi-LSTM Units per layer = 5

Batch size = 16

Number of epoch = 32

Number of Bi-LSTM layers = 2

Number of Dense layer = 1

Dropout = 0.0

Influencing days and neighbors:

We wanted to know how the neural network performs on different days. the graph: 6.6 shows the number of days in the horizontal axis and the resulting neural network *MAPE* on the vertical axis. The *MAPE* fluctuates but does not decrease when the number of days increases from 1 to 6. It means that a small change in the number of days may not be very influential in the error calculation. In general, more days lead to

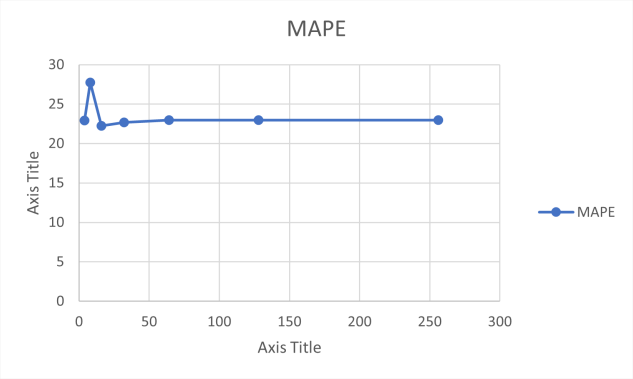

Figure 6.7: Bi-LSTM Number of neighbors

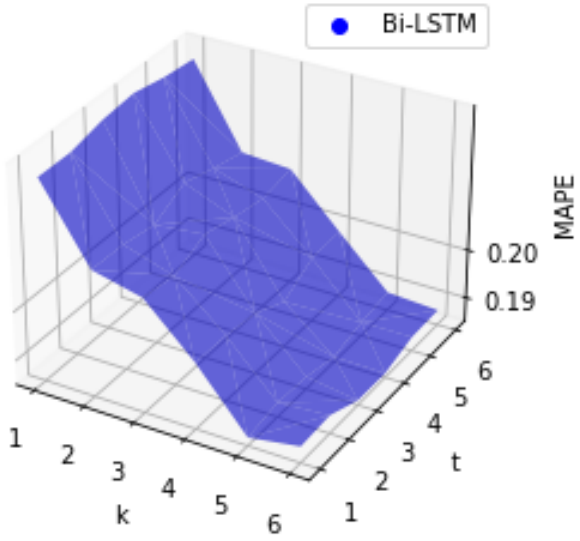

Figure 6.8: Bi-LSTM MAPE

Batch Size	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.47135	2.625978	21.96459	6.895763	270.6237
8	1.462273	2.618448	22.27791	6.85627	130.3596
16	1.460025	2.618315	22.66918	6.855573	68.19878
32	1.462291	2.621777	22.96565	6.873713	37.63035
64	1.460971	2.62024	22.85365	6.865658	23.03905
128	1.460505	2.61964	22.80368	6.862513	13.47705
256	1.459963	2.618018	22.6272	6.854018	10.11925

Figure 6.9: GRU Batch Table

a worse prediction because our Bi-LSTM model is a single prediction model which predicts one value for all stations. Such model makes the operations computationally efficient but it is subject to a known common error called Error Accumulation which increases the more days we try to predict.[2]

The figure : 6.7 shows that when we use more nearest neighbors, the *MAPE* decreases. So, our Bi-LSTM can learn the correlation between the number of neighbors

The last figure: 6.8 is showing the relation between the number of days, the number of neighbors and the *MAPE*.

- *Gated Recurrent Unit:*

  Find optimal batch size:

  The process of finding the batch size for the GRU model is the same as the Bi-LSTM. The batch table: 6.9 shows that *MAPE* is increasing with the increasing batch size.

  The batch graph: 6.10 shows the *MAPE* increasing until batch size 32 and started to decrease.

  Find optimal Number of epochs:

2. Yanlai Zhou et al., "Explore a deep learning multi-output neural network for regional multi-step-ahead air quality forecasts," *Journal of Cleaner Production* 209 (October 2018), https://doi.org/10.1016/j.jclepro.2018.10.243.

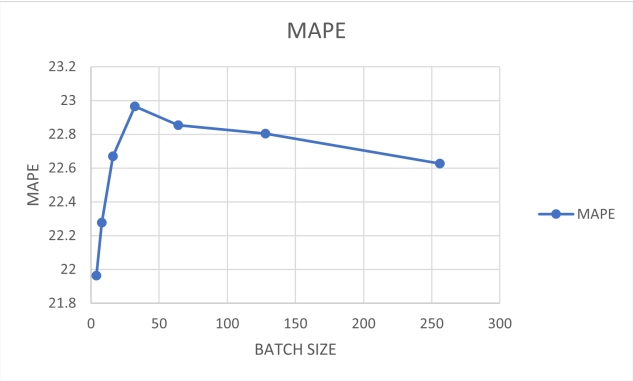

Figure 6.10: GRU Batch Graph

Epochs	MAE	RMSE	MAPE	MSE	Ex_Time
4	2.278859	3.020093	38.12708	9.120964	5.470087
8	1.481895	2.632573	23.64478	6.930443	9.071565
16	1.462572	2.618641	22.26074	6.857278	12.1776
32	1.459983	2.618093	22.63824	6.85441	22.46944
64	1.462371	2.621868	22.97174	6.874191	38.05761
128	1.462596	2.622127	22.98874	6.875552	84.79466
256	1.462505	2.622022	22.9819	6.875	151.5951

Figure 6.11: GRU Epochs Table

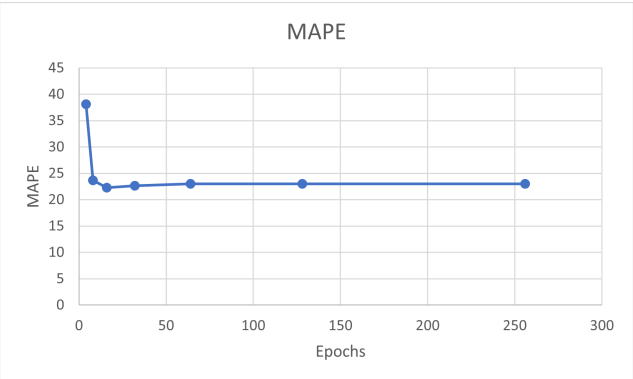

Figure 6.12: GRU Epochs Graph

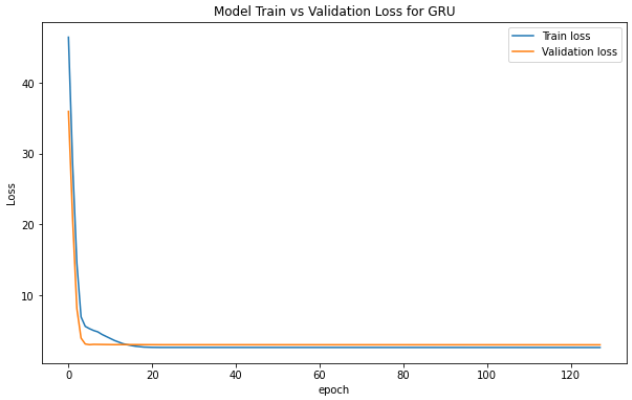

Figure 6.13: GRU Loss-Validation Curves

The Epochs table: 6.11 shows that the *MAPE* decreases with greater number of epochs. It means that the neural network is learning.

While the table misleads us to think that the neural network is learning on more epochs, the epochs graph: 6.12 shows that the learning process becomes insignificant when the number of epochs passes a certain threshold.

The Validation and Loss: 6.13 figure confirms that the neural network training does not converge anymore after a certain number of epochs. It also shows that the network is overfitting a little. So we decided to add a dropout.

After doing more tests, we picked the optimal parameters as follow:

GRU Units per layer = 5

Batch size = 16

Number of epoch = 32

Number of GRU layers = 2

Number of Dense layer = 1

Dropout = 0.3

Influencing days and neighbors:

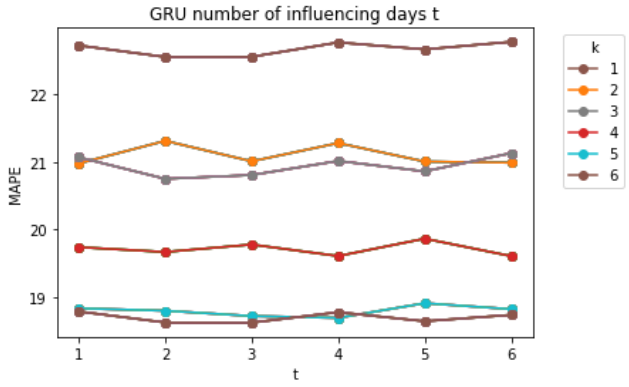

Figure 6.14: GRU Number of influencing days

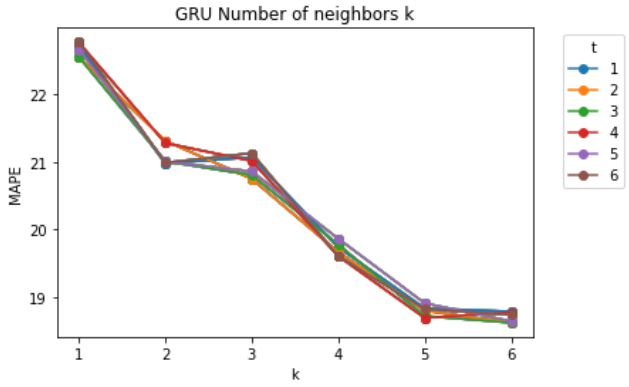

Figure 6.15: GRU Number of neighbors

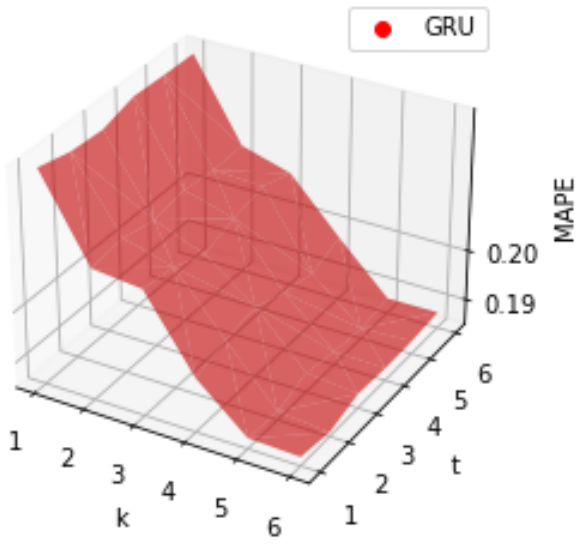

Figure 6.16: GRU MAPE

The figure: 6.14 shows that the *MAPE* fluctuates when the number of days increases but the change is not significant.

The figure : 6.15 shows that the more number of neighbors we add, the lower the *MAPE*. So the number of neighbors influence the error significantly.

This last figure: 6.16 is a 3D showing the relationship between the number of days, the number of neighbors and the resulting *MAPE*.

- *Temporal Convolution Neural Network:*

  The way of finding the optimal parameters for the TCN is different than the Bi-LSTM and GRU models. TCN layers has filters instead of neuron units. Each layer output size is different than the input size because TCN unit has dilation rates which yields an output different than the input. So, the size of input should match the TCN units requirements. TCN input takes the following (number of samples, number of steps, number of features). We get an error our input data does not match the required input. We also need to manually adjust the hidden layers to match the output of each

Batch Size	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.575733	3.107699	23.88338	9.657791	97.82327
8	1.399156	2.717456	23.36261	7.384569	52.07021
16	1.517412	3.288502	23.16181	10.81425	28.07754
32	1.552875	3.026786	24.14521	9.161434	15.10343
64	1.483259	2.815336	30.24784	7.926118	8.203065
128	1.502759	2.612006	27.33314	6.822576	5.412325
256	1.594034	3.142262	28.12091	9.873813	3.767615

Figure 6.17: TCN Batch Table

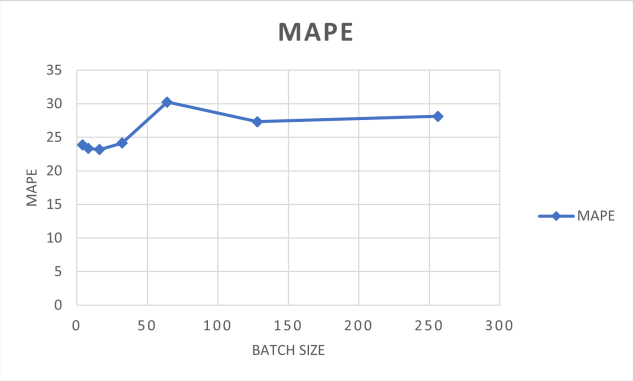

Figure 6.18: TCN Batch Graph

TCN layer.

Find optimal batch size:

The batch table: 6.17 shows different batch sizes that we tested. These data were obtained with $k = 1$ and $t = 1$ and kernel size =1. The batch graph: 6.18 shows how the *MAPE* changes when the batch size changes. It is increasing for low batch sizes and then stabilized for large batch sizes.

Find optimal Number of epochs:

The epochs table: 6.19 shows the influence of different epochs on different errors.

The epochs graph: 6.20 shows that the model is not learning after very low epochs. So using a large number of epochs will not help the learning process.

The validation and Loss: 6.21 curve shows that TCN converges fast and won't con-

Epochs	MAE	RMSE	MAPE	MSE	Ex_Time
4	2.293906	3.895824	33.98983	15.17745	1.621808
8	1.635687	3.339997	24.71951	11.15558	2.35916
16	1.622271	3.320666	24.54198	11.02682	4.110749
32	1.595795	3.308152	24.54881	10.94387	3.429568
64	1.608586	3.312746	24.53013	10.97429	3.404691
128	1.602984	3.319115	24.66448	11.01653	3.614789
256	1.611614	3.322392	24.6466	11.03829	3.441077

Figure 6.19: TCN Epochs Table

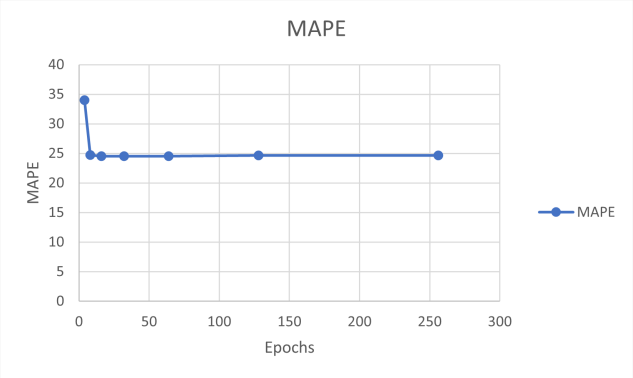

Figure 6.20: TCN Epochs Graph

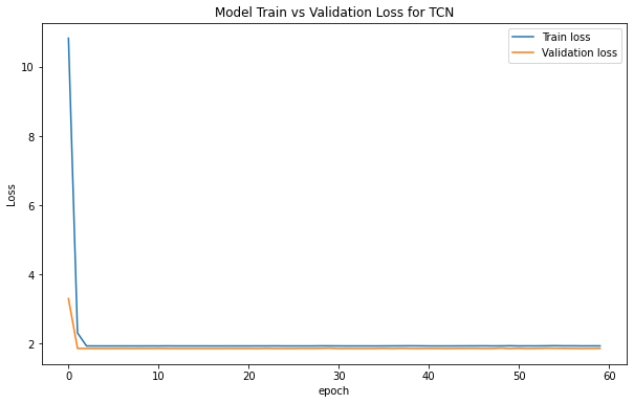

Figure 6.21: TCN Loss-Validation Curves

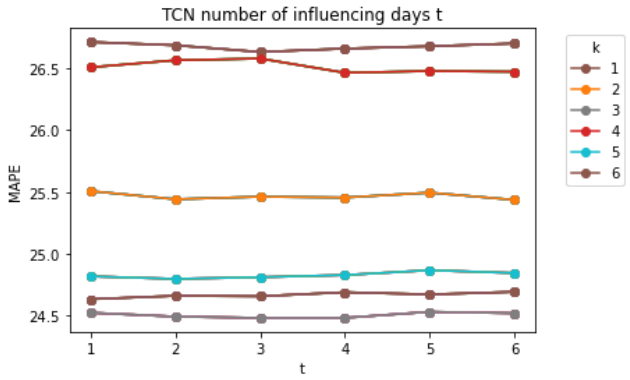

Figure 6.22: TCN Number of influencing days

verge nor get overfitted for large epochs.

After multiples tests. We chose some parameters that could be used to train and test on all k and t as the following:

TCN filters per layer = 5

Batch size = 16

Number of epoch = 32

Number of TCN layers = 1

Number of Dense layer = 1

Dropout = 0.0

Influencing days and neighbors:

The figure : 6.22 shows barely no change to the increase of days. So the number of days either has no influence on the TCN or the increase in days is too small for the network to learn something.

The figure: 6.23 shows that TCN notices the change in nearest neighbors, the *MAPE* go straight down and up instead of keep on going down. The TCN either overestimates or underestimate the impact of nearest neighbors on the calculation of *MAPE*.

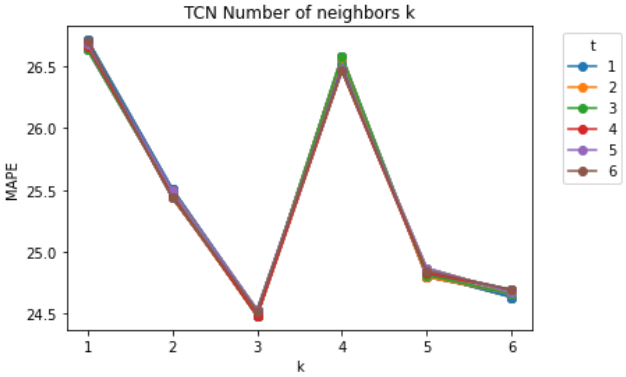

Figure 6.23: TCN Number of neighbors

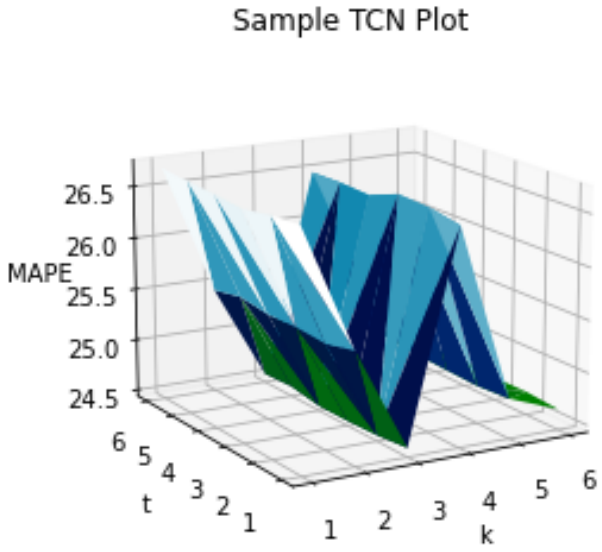

Figure 6.24: TCN MAPE

Batch Size	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.726071	2.297608	26.45715	5.279001	418.1693
8	1.713245	2.282262	26.98225	5.208718	266.0102
16	1.729887	2.291388	28.30594	5.250457	138.2233
32	1.772771	2.331357	29.80807	5.435223	74.67348
64	1.805788	2.364528	30.73903	5.590991	45.21747
128	1.830378	2.390742	31.38814	5.715646	26.55504
256	1.846253	2.40782	31.78841	5.797599	18.61945

Figure 6.25: Bi-LSTM Batch Table

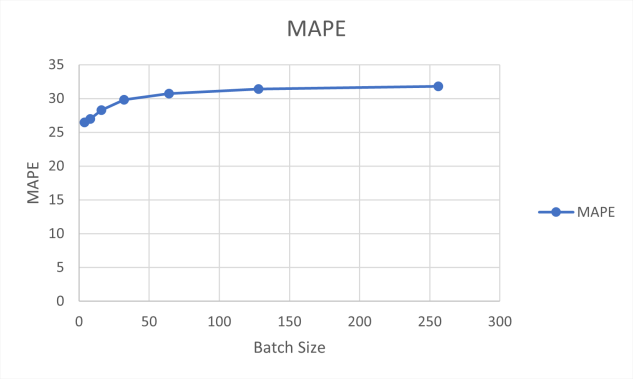

Figure 6.26: Bi-LSTM Batch Graph

The last figure: 6.24 shows the relation between t, k, and the resulting *MAPE*. We can see some sharp changes when the number of neighbors change but the changes are not coherent.

6.1.2 DATASET 2

This dataset has more observations than the first dataset. We performed the same tests and got the results below.

- *Bi-Directional Long-Short term memory:*

  Find optimal batch size:

  The batch table: 6.25 shows the batch size influence on several errors. Note that we are using *MAPE* as our error estimation. The other errors could have been used too.

Epochs	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.715228	2.285379	26.78684	5.222956	14.67701
8	1.713963	2.280705	27.33817	5.201615	23.7331
16	1.716854	2.281895	27.60541	5.207046	41.87406
32	1.729616	2.291166	28.29407	5.249439	86.17817
64	1.730338	2.291757	28.32552	5.25215	145.9758
128	1.729849	2.291357	28.3043	5.250316	267.828
256	1.729916	2.291411	28.30722	5.250567	274.3794

Figure 6.27: Bi-LSTM Epochs Table

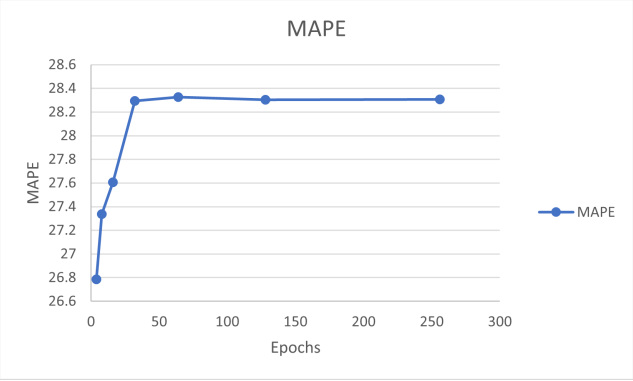

Figure 6.28: Bi-LSTM Epochs Graph

The batch graph: 6.26 gives a visual representation of the influence of batch size on the *MAPE*. We can see the error increasing with the batch size and became stable for higher batch sizes.

Find optimal Number of epochs:

The epochs table: 6.27 shows different number of epochs and the resulting *MAPE*.

The epochs graph: 6.28 shows a sharp increase of *MAPE* with the increasing epochs and then becomes stable for greater epochs numbers.

The validation and Loss: 6.29 curves gives us an answers of why the sharp increase in the epochs graph. The neural network is overfitting. This case was found interesting because training a neural network with more data is a known solution to solve overfitting and while dataset 1 which has less data did not overfit, dataset 2 with more data

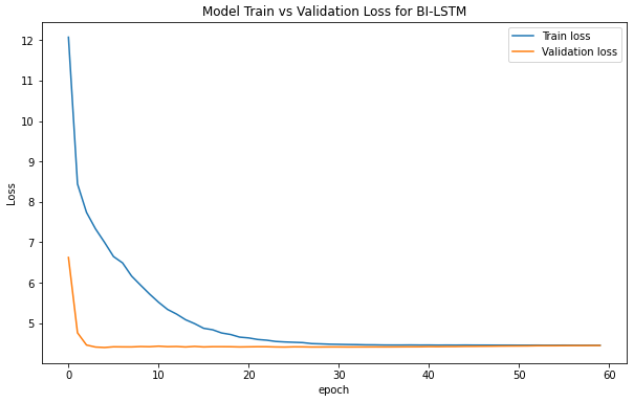

Figure 6.29: Bi-LSTM Loss-Validation Curves

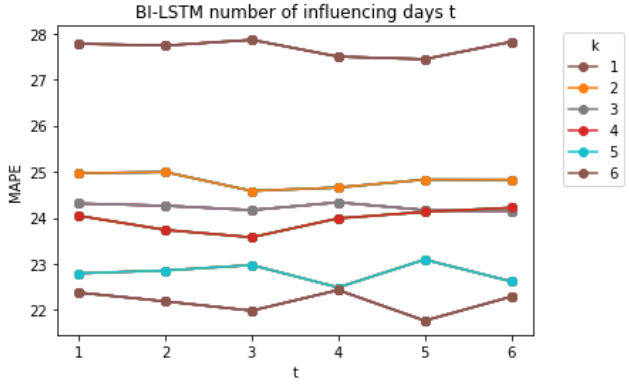

Figure 6.30: Bi-LSTM Number of influencing days

overfitted. To solve overfitting, we added a dropout and obtained the results below.

Bi-LSTM Units per layer = 5

Batch size = 16

Number of epoch = 16

Number of Bi-LSTM layers = 2

Number of Dense layer = 1

Dropout = 0.3

Influencing days and neighbors:

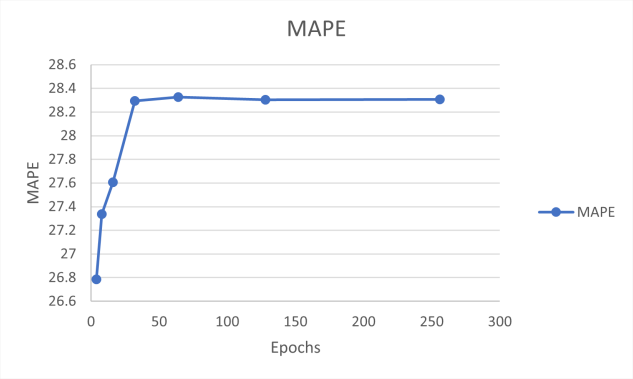

Figure 6.31: Bi-LSTM Number of neighbors

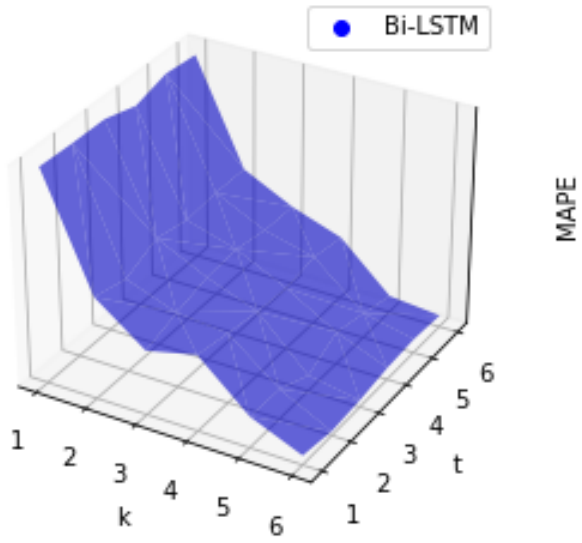

Figure 6.32: Bi-LSTM MAPE

Batch Size	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.726868	2.298509	26.44142	5.283145	316.6551
8	1.71321	2.282148	26.99296	5.208201	222.4437
16	1.729546	2.291107	28.29094	5.249172	113.587
32	1.77165	2.330287	29.77494	5.430237	62.56998
64	1.809006	2.367966	30.82676	5.607263	34.76101
128	1.825624	2.38567	31.26594	5.691421	21.11692
256	1.835442	2.396117	31.51648	5.741376	12.96671

Figure 6.33: GRU Batch Table

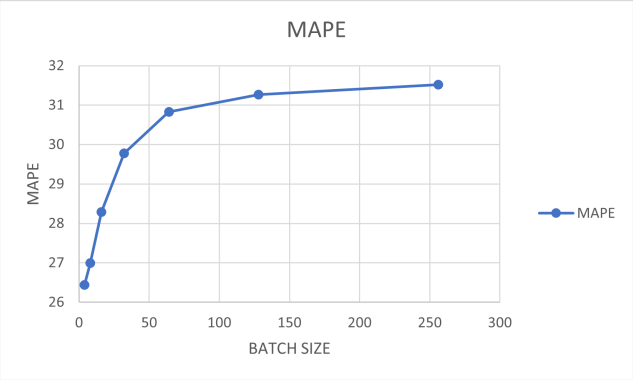

Figure 6.34: GRU Batch Graph

The figure: 6.30 shows that the number of days does not influence much on the MAPE.

The figure: 6.31 shows that increasing the number of nearest neighbors lowers the *MAPE*.

The figure: 6.32 is a 3D representation of the k, t and *MAPE*.

- *Gated Recurrent Unit:*

  Find optimal batch size:

  The batch table: 6.33 shows that increasing the batch size decreases the runtime but the *MAPE* is increasing with the batch size.

  The batch graph: 6.34 shows that the *MAPE* increases with the batch size but tends

Epochs	MAE	RMSE	MAPE	MSE	Ex_Time
4	1.713416	2.280692	27.26241	5.201557	10.38095
8	1.713481	2.280685	27.27231	5.201523	18.89874
16	1.718917	2.283115	27.74414	5.212614	31.15781
32	1.729929	2.291422	28.30778	5.250615	61.62354
64	1.729679	2.291218	28.29686	5.249679	119.5296
128	1.729858	2.291365	28.30471	5.250351	150.2277
256	1.730055	2.291525	28.31326	5.251086	155.093

Figure 6.35: GRU Epochs Table

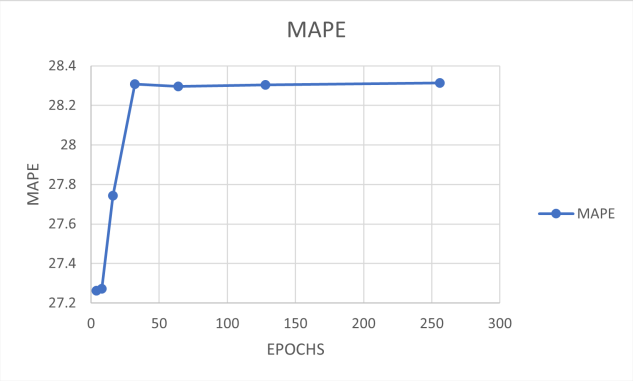

Figure 6.36: GRU Epochs Graph

to stabilize for larger batch sizes.

Find optimal Number of epochs:

Epochs table: 6.35

Epochs graph: 6.36

Validation and Loss: 6.37

GRU Units per layer = 5

Batch size = 16

Number of epoch = 16

Number of GRU layers = 2

Number of Dense layer = 1

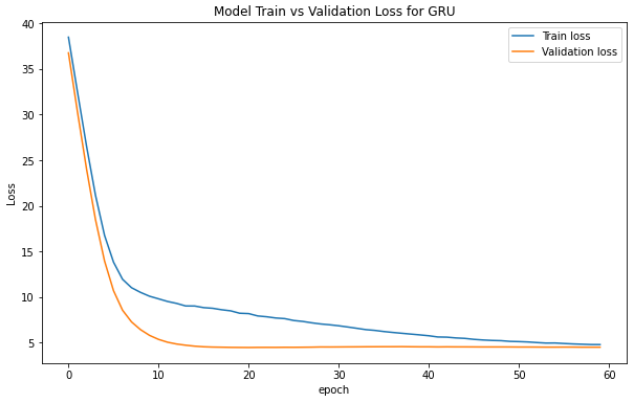

Figure 6.37: GRU Loss-Validation Curves

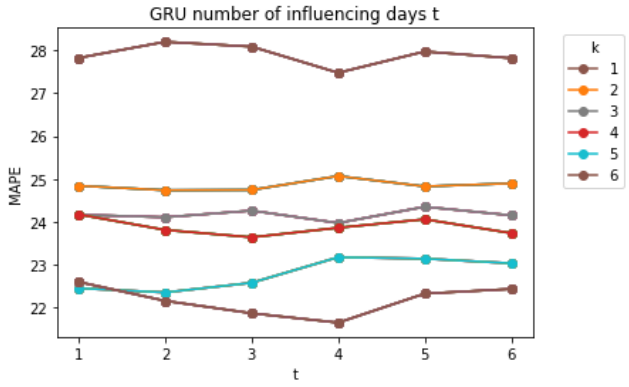

Figure 6.38: GRU Number of influencing days

Dropout = 0.0

Influencing days and neighbors:

The figure : 6.38 shows that *MAPE* fluctuates when t is increasing. We can not say that t is influencing the *MAPE* because the changes are not obvious.

The figure: 6.39 shows that the number of more number of neighbors we passed to the neural network, the better it learns because the *MAPE* decreases as k increases.

Figure: 6.40 shows a 3D graph of the k, t and *MAPE*. We can see that the decreasing slope is due to the k.

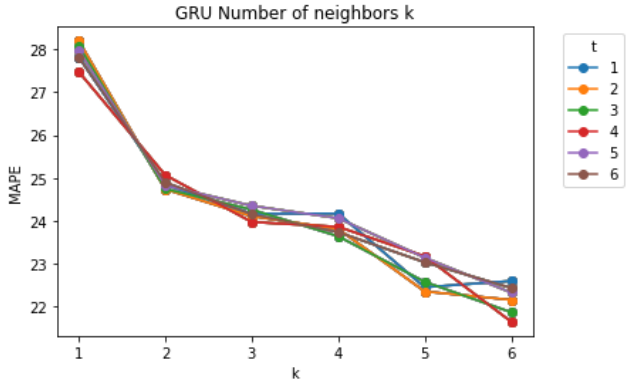

Figure 6.39: GRU Number of neighbors

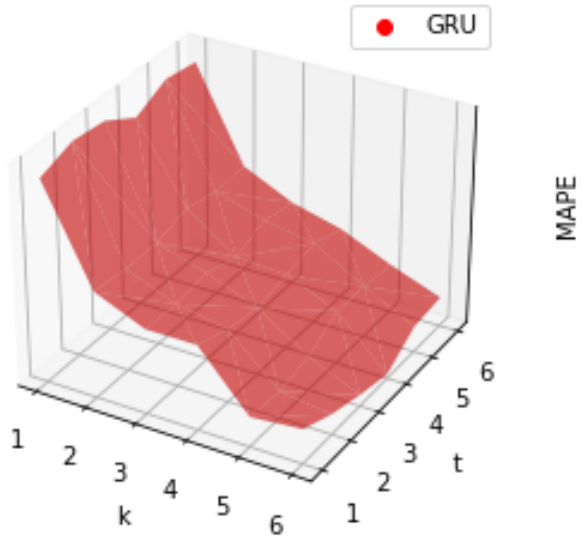

Figure 6.40: GRU MAPE

Batch Size	MAE	RMSE	MAPE	MSE	Ex_Time
4	2.078026	3.401002	37.96826	11.56681	142.8541
8	2.084086	3.199452	30.91112	10.2365	80.61807
16	2.113387	3.328948	30.10564	11.08189	40.65471
32	2.09446	3.176455	31.72592	10.08986	23.37254
64	2.257916	3.398839	39.12374	11.55211	12.4536
128	1.990292	3.131303	33.74913	9.805059	7.149449
256	2.203721	3.421049	31.89844	11.70358	4.514999

Figure 6.41: TCN Batch Table

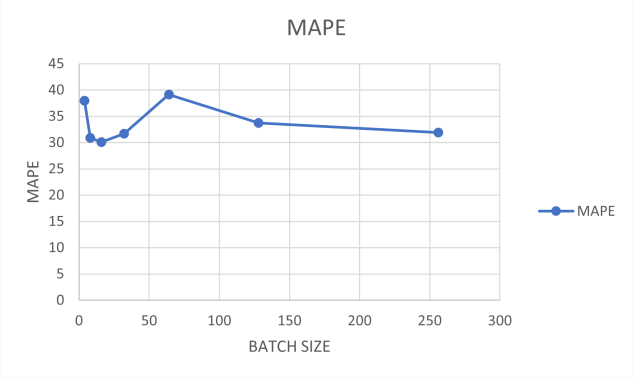

Figure 6.42: TCN Batch Graph

- *Temporal Convolution Neural Network:*

  Find optimal batch size:

  The batch table: 6.41 shows that while the batch size increases, the *MAPE* fluctuates.

  The figure: 6.42 shows that the *MAPE* decreases for very low batch size and then increases for larger batch sizes until a certain level and then starts to decrease again.

  Find optimal Number of epochs:

  The figure: 6.43 shows that the *MAPE* varies with the increase of number of epochs. This table also shows a phenomenon that we have not seen so far. It is the resulting *MAPE* for 32 epochs which is inf or infinity. As shown above, the formula to calculate *MAPE* has a fraction and the observed values in the denominator could lead to a division by a number close to zero. That's the case in the table.

EPOCHS	MAE	RMSE	MAPE	MSE	Ex_Time
4	2.066643	3.047233	31.25427	9.285627	3.041314
8	2.014636	3.125362	29.69613	9.76789	5.546745
16	2.184891	3.367651	31.67832	11.34108	10.5204
32	2.228419	3.440554	inf	11.83741	20.78382
64	2.180242	3.250309	34.36397	10.56451	41.12397
128	2.139332	3.2212	40.92452	10.37613	64.66172
256	2.122732	3.211847	32.65506	10.31596	33.89075

Figure 6.43: TCN Epochs Table

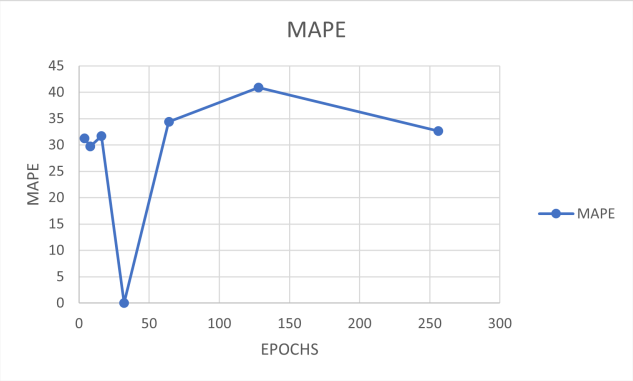

Figure 6.44: TCN Epochs Graph

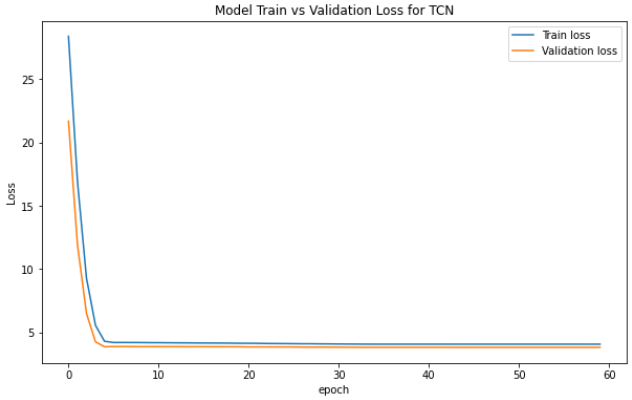

Figure 6.45: TCN Loss-Validation Curves

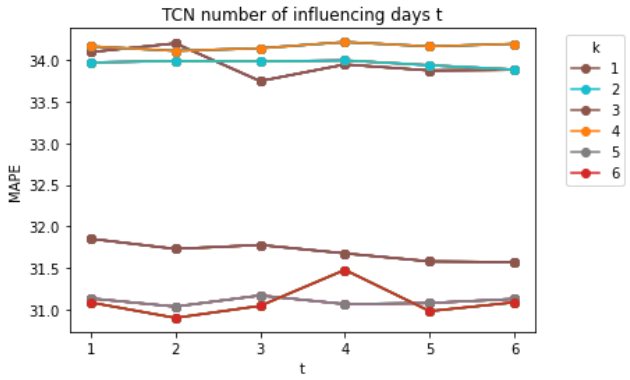

Figure 6.46: TCN Number of influencing days

The epochs graph: 6.44 shows that the *MAPE* suddenly dropped to zero and then increases again. Such drop does not mean that the batch size is optimal. It means that the error measurement is not optimal. *MSE* is usually used when the dataset contains values near zero but can not be dropped. In our case, we have recorded both *MAPE* and *MSE* which helped us keep track of the influence of the number of epochs on the model.

The validation and Loss: 6.45 shows that the model is learning at small epochs and then gets stable at larger epochs.

TCN filters per layer = 5

Batch size = 16

Number of epoch = 16

Number of TCN layers = 1

Number of Dense layer = 1

Dropout = 0.0 *Influencing days and neighbors:*

The figure : 6.46 shows that more days do not make an impact on the *MAPE*. However, we noticed that the number of lines are grouped in two separated by a large gap.

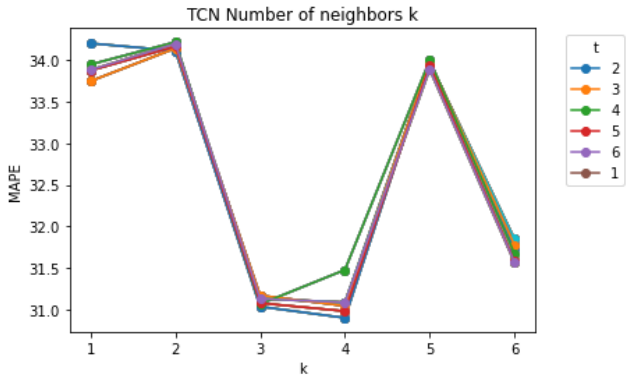

Figure 6.47: TCN Number of neighbors

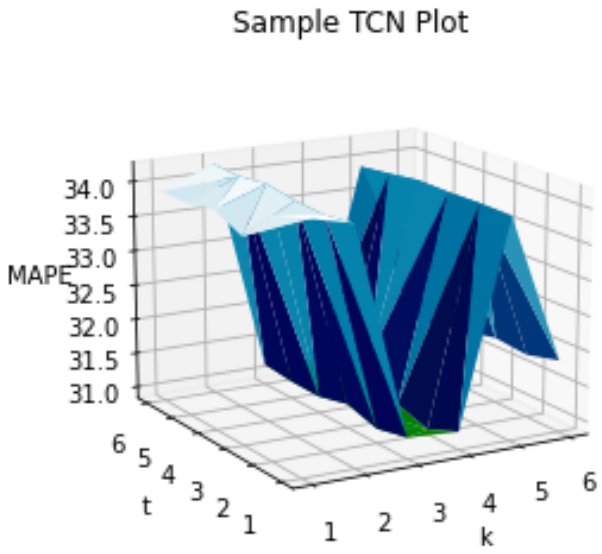

Figure 6.48: TCN MAPE

The *MAPE* for k values 1, 2, 5 are large and similar while *MAPE* for k values of 3, 4, 6 are small and similar. While we don't understand this behavior, we can say that the TCN attempts to estimate the relationship between the nearest neighbors but it fails to estimate it correctly.

The figure: 6.47 show that increasing the number of neighbors changes the *MAPE* in a correlated fashion for all t

The figure: 6.48 shows that while the *MAPE* varies when k increases, the variation pattern is similar for all the data.

6.2 Experiment 2: Comparison between Bi-LSTM, GRU, TCN

In this section, we will show all the measurements that we have for each model.

6.2.1 Dataset 1

Figure: 6.49 shows all the 36 measurements for the Bi-LSTM using dataset 1.

Figure: 6.50 shows all the 36 measurements for the GRU using dataset 1.

Figure: 6.51 Shows all the 36 measurements for the TCN using dataset 1.

By comparing their *MAPE*, we can see that the Bi-LSTM model starts with the lowest *MAPE* followed by the GRU. When we compare the execution time of the three, we can see that the TCN leads the other two and then is followed by the GRU. By comparing the rate of decrease of the *MAPE*, we see that the GRU is the fastest followed by the Bi-LSTM. The GRU is the model that ends with the lowest *MAPE*

6.2.2 Dataset 2

Figure: 6.52 shows all measurements for Bi-LSTM using dataset 2.

Figure: 6.53 shows all measurements for GRU using dataset 2.

MAE	RMSE	MAPE	MSE	Ex_Time(sec)	k	t
1.459993	2.618142	22.64551	6.854667	55.80943	1	1
1.461335	2.620101	22.76136	6.864928	56.27454	1	2
1.460469	2.619564	22.63648	6.862117	61.82934	1	3
1.461691	2.621185	22.72055	6.870611	77.02002	1	4
1.464294	2.624533	22.93741	6.888175	93.66734	1	5
1.463691	2.622856	22.6768	6.879372	90.28486	1	6
1.466322	2.429374	21.01706	5.901857	41.17057	2	1
1.466971	2.429801	21.04703	5.903932	56.55449	2	2
1.465671	2.427032	21.14339	5.890485	69.88561	2	3
1.469165	2.434054	20.97907	5.92462	72.63165	2	4
1.467653	2.429117	21.17102	5.900609	83.28967	2	5
1.4702	2.425392	21.49791	5.882524	90.1072	2	6
1.431287	2.186599	20.85703	4.781213	42.18239	3	1
1.431852	2.188875	20.81414	4.791176	56.81237	3	2
1.431469	2.18987	20.77876	4.795529	61.51445	3	3
1.433718	2.186835	20.96872	4.782249	71.86734	3	4
1.433301	2.191644	20.80796	4.803302	77.44578	3	5
1.434577	2.190473	20.902	4.798172	96.86201	3	6
1.341645	1.991223	19.53429	3.964968	43.54749	4	1
1.342976	1.988201	19.68454	3.952945	58.86083	4	2
1.342324	1.992904	19.52814	3.971665	60.32816	4	3
1.345121	1.987113	19.81916	3.948619	69.34163	4	4
1.345576	1.988362	19.80364	3.953585	79.61906	4	5
1.344734	1.995271	19.56675	3.981105	85.46213	4	6
1.267253	1.841429	18.73256	3.39086	42.00405	5	1
1.270039	1.840231	18.87385	3.386449	53.06629	5	2
1.271258	1.840058	18.93459	3.385814	67.6055	5	3
1.268376	1.843872	18.72783	3.399864	73.26147	5	4
1.269581	1.843815	18.77968	3.399652	85.06481	5	5
1.270304	1.844765	18.78271	3.403158	86.75685	5	6
1.263632	1.780812	18.82309	3.17129	41.75899	6	1
1.263187	1.782209	18.77605	3.176267	53.13632	6	2
1.263059	1.782677	18.76683	3.177938	58.75876	6	3
1.262651	1.784428	18.71198	3.184182	68.92715	6	4
1.262949	1.785553	18.69949	3.1882	77.07594	6	5
1.265009	1.785203	18.78025	3.18695	99.19309	6	6

Figure 6.49: Bi-LSTM Measurements for our algorithm Data 1

MAE	RMSE	MAPE	MSE	Ex_Time(sec)	k	t
1.460167	2.618829	22.72816	6.858263	27.64781	1	1
1.46101	2.618735	22.55699	6.857776	30.61233	1	2
1.460408	2.619187	22.55998	6.860142	36.53509	1	3
1.461872	2.621681	22.77296	6.873212	44.74497	1	4
1.46263	2.621797	22.66992	6.87382	51.75114	1	5
1.463982	2.623673	22.78229	6.883659	52.51981	1	6
1.46726	2.430976	20.97812	5.909647	29.95058	2	1
1.466262	2.423684	21.31372	5.874245	36.54653	2	2
1.467085	2.431126	21.00971	5.910374	44.92777	2	3
1.466668	2.425575	21.28085	5.883414	70.28458	2	4
1.469701	2.434436	21.00668	5.926477	56.6749	2	5
1.471814	2.437049	20.98925	5.939206	60.98529	2	6
1.433602	2.182926	21.0699	4.765164	29.6228	3	1
1.431654	2.190685	20.74977	4.7991	36.05202	3	2
1.431581	2.189134	20.80613	4.792306	40.57423	3	3
1.43427	2.186059	21.0154	4.778855	49.37666	3	4
1.433586	2.190294	20.86052	4.797389	55.44264	3	5
1.43719	2.186604	21.13113	4.781238	59.07152	3	6
1.342829	1.986124	19.73606	3.944689	27.86081	4	1
1.342828	1.988639	19.66536	3.954684	33.7905	4	2
1.343872	1.986762	19.77545	3.947222	39.2521	4	3
1.343279	1.991704	19.60463	3.966884	47.0548	4	4
1.346331	1.98748	19.86306	3.950076	52.91212	4	5
1.344798	1.994077	19.60369	3.976343	58.53346	4	6
1.26875	1.839882	18.82926	3.385166	29.83803	5	1
1.268665	1.841296	18.79379	3.390371	35.59691	5	2
1.267651	1.843166	18.71499	3.397262	41.31305	5	3
1.267906	1.844748	18.68749	3.403094	48.62502	5	4
1.271715	1.842058	18.90543	3.393179	55.60395	5	5
1.270822	1.844176	18.81729	3.400983	61.99344	5	6
1.262835	1.781276	18.78265	3.172944	29.01878	6	1
1.260738	1.785016	18.61891	3.186283	35.91692	6	2
1.260766	1.785434	18.61644	3.187775	43.66646	6	3
1.26367	1.783481	18.77214	3.180805	48.4295	6	4
1.262084	1.786844	18.6374	3.192812	52.62445	6	5
1.264207	1.785989	18.73203	3.189757	56.07748	6	6

Figure 6.50: GRU Measurements for our algorithm Data 1

MAE	RMSE	MAPE	MSE	Ex_Time(sec)	k	t
1.467879	2.918951	26.7119	8.520273	24.36735	1	1
1.46049	2.905257	26.68539	8.440516	24.31072	1	2
1.459267	2.904636	26.63122	8.436912	24.6635	1	3
1.465859	2.905643	26.65877	8.442759	25.672	1	4
1.462575	2.907089	26.67762	8.451165	23.43553	1	5
1.465118	2.908268	26.70039	8.458022	26.32479	1	6
1.605939	3.287497	25.50786	10.80764	24.18524	2	1
1.602614	3.287092	25.44226	10.80498	25.7646	2	2
1.604395	3.288501	25.46256	10.81424	24.75986	2	3
1.605667	3.289446	25.45572	10.82045	26.74616	2	4
1.60505	3.290496	25.49534	10.82736	23.14054	2	5
1.603049	3.291268	25.43691	10.83244	28.08121	2	6
1.552115	2.942025	24.52536	8.655512	24.60634	3	1
1.551264	2.943331	24.49451	8.663197	24.00959	3	2
1.549674	2.926827	24.48396	8.566316	25.04927	3	3
1.544093	2.927912	24.48419	8.572669	25.33426	3	4
1.555099	2.929457	24.533	8.581718	24.13584	3	5
1.548448	2.93033	24.5218	8.586835	26.75101	3	6
1.567341	3.514007	26.50789	12.34824	25.1188	4	1
1.570026	3.515319	26.56449	12.35747	25.38251	4	2
1.570785	3.516783	26.57837	12.36776	26.03333	4	3
1.567293	3.516211	26.46331	12.36374	25.26304	4	4
1.568491	3.517807	26.47743	12.37496	23.56703	4	5
1.567277	3.517236	26.47189	12.37095	27.72845	4	6
1.594843	3.238318	24.81961	10.4867	24.37975	5	1
1.594106	3.238932	24.7991	10.49068	24.53918	5	2
1.59864	3.240538	24.81273	10.50109	25.10979	5	3
1.597769	3.241709	24.82992	10.50868	24.97753	5	4
1.597681	3.242486	24.86895	10.51372	23.16961	5	5
1.596972	3.243994	24.84634	10.5235	26.12092	5	6
1.605568	3.109385	24.63512	9.668278	24.0388	6	1
1.60703	3.110609	24.66218	9.67589	24.20758	6	2
1.607382	3.112042	24.65859	9.684802	24.79461	6	3
1.612261	3.113818	24.69126	9.695861	25.70439	6	4
1.608719	3.114274	24.67343	9.698702	22.9831	6	5
1.610053	3.115608	24.69449	9.707011	26.24271	6	6

Figure 6.51: TCN Measurements for our algorithm Data 1

MAE	RMSE	MAPE	MSE	Ex_Time(sec)	k	t
1.7194	2.283428	27.77423	5.214043	43.29876	1	1
1.719366	2.283484	27.73527	5.214299	56.41494	1	2
1.721547	2.285186	27.86011	5.222077	73.70171	1	3
1.715935	2.282003	27.49505	5.207536	74.50665	1	4
1.715802	2.282256	27.44038	5.20869	63.6239	1	5
1.720474	2.284814	27.81682	5.220375	93.14621	1	6
1.591043	2.102944	24.96869	4.422373	41.73438	2	1
1.592097	2.103684	24.99393	4.425484	51.18318	2	2
1.586701	2.099402	24.5831	4.40749	64.88024	2	3
1.587079	2.100024	24.6576	4.410101	74.79218	2	4
1.58898	2.101921	24.82943	4.418071	63.19527	2	5
1.588572	2.101675	24.82706	4.417038	93.23947	2	6
1.55668	2.021897	24.31195	4.088066	43.85148	3	1
1.555982	2.021456	24.26267	4.086285	55.50242	3	2
1.554171	2.020438	24.16775	4.082169	63.15887	3	3
1.55757	2.022985	24.33515	4.092468	73.41963	3	4
1.554103	2.020755	24.16838	4.08345	62.49127	3	5
1.553168	2.020191	24.13334	4.08117	94.81867	3	6
1.560725	2.007809	24.04443	4.031298	41.43912	4	1
1.553156	2.004426	23.73845	4.017723	51.51027	4	2
1.549951	2.003701	23.57797	4.014819	62.67402	4	3
1.559505	2.007575	23.98839	4.030359	75.95924	4	4
1.563343	2.009777	24.12722	4.039205	62.26869	4	5
1.565914	2.011285	24.21948	4.045268	92.68671	4	6
1.511407	1.92418	22.79565	3.702468	41.95098	5	1
1.51347	1.925474	22.85846	3.707448	51.14992	5	2
1.517349	1.927796	22.97568	3.716397	63.17808	5	3
1.503237	1.921338	22.49117	3.691538	75.32493	5	4
1.521108	1.930256	23.09462	3.72589	62.29661	5	5
1.506623	1.922815	22.6217	3.697218	105.3455	5	6
1.501381	1.894063	22.37899	3.587475	40.51763	6	1
1.494842	1.891033	22.18942	3.576005	53.07465	6	2
1.488179	1.888544	21.98645	3.566599	62.08443	6	3
1.503447	1.895633	22.4338	3.593424	72.37344	6	4
1.481438	1.886742	21.76982	3.559794	63.26683	6	5
1.49837	1.893229	22.29015	3.584314	91.36502	6	6

Figure 6.52: Bi-LSTM Measurements for our algorithm Data 2

MAE	RMSE	MAPE	MSE	Ex_Time(sec)	k	t
1.720268	2.284012	27.82656	5.216711	33.49558	1	1
1.728025	2.289866	28.20153	5.243488	41.62771	1	2
1.725787	2.288338	28.08891	5.236492	50.59596	1	3
1.715765	2.281928	27.47892	5.207195	60.9109	1	4
1.723618	2.287002	27.97552	5.230377	44.96372	1	5
1.720521	2.284845	27.81954	5.220518	76.26182	1	6
1.588709	2.101137	24.84122	4.414776	34.22444	2	1
1.587899	2.100335	24.73831	4.411408	40.30184	2	2
1.588042	2.100683	24.74495	4.412868	49.18062	2	3
1.593615	2.105306	25.06908	4.432314	58.42008	2	4
1.588948	2.101896	24.82725	4.417967	44.15222	2	5
1.589789	2.102605	24.89761	4.420949	74.30915	2	6
1.553556	2.019711	24.16575	4.079233	33.39702	3	1
1.552804	2.019362	24.10739	4.077822	42.48294	3	2
1.55614	2.021764	24.26215	4.087528	49.89361	3	3
1.550435	2.018418	23.97233	4.074009	63.74793	3	4
1.557992	2.023516	24.35184	4.094618	43.92427	3	5
1.553523	2.020419	24.15148	4.082094	81.94114	3	6
1.564405	2.009881	24.17192	4.039622	33.96436	4	1
1.554851	2.00515	23.81124	4.020627	41.21934	4	2
1.55122	2.004052	23.63932	4.016225	49.80807	4	3
1.556168	2.005947	23.8619	4.023823	65.88995	4	4
1.56141	2.008709	24.05971	4.034912	51.01063	4	5
1.552918	2.004592	23.73643	4.01839	78.9596	4	6
1.501756	1.920233	22.4504	3.687297	34.2181	5	1
1.499587	1.920006	22.35253	3.686422	44.74769	5	2
1.505601	1.92212	22.57806	3.694544	51.64602	5	3
1.523796	1.931718	23.17434	3.731536	64.52532	5	4
1.522653	1.931196	23.14108	3.729517	49.84051	5	5
1.519026	1.929106	23.03304	3.72145	85.51736	5	6
1.509259	1.898532	22.59667	3.604426	34.25114	6	1
1.4937	1.890526	22.15588	3.574089	44.46545	6	2
1.48435	1.887279	21.86634	3.561823	54.39527	6	3
1.477783	1.885757	21.64654	3.55608	60.84459	6	4
1.499619	1.893821	22.32477	3.586557	43.11273	6	5
1.50343	1.895868	22.43337	3.594314	74.02412	6	6

Figure 6.53: GRU Measurements for our algorithm Data 2

MAE	RMSE	MAPE	MSE	Ex_Time(sec)	k	t
2.157767	3.191464	34.09431	10.18544	5.734554	1	1
2.154241	3.191908	34.19992	10.18828	5.648307	1	2
2.155457	3.187966	33.74551	10.16313	5.854018	1	3
2.148438	3.188303	33.94602	10.16527	5.896174	1	4
2.152143	3.188477	33.87298	10.16639	5.37117	1	5
2.15262	3.18916	33.88066	10.17074	6.08429	1	6
2.216351	3.436374	34.16397	11.80867	5.639373	2	1
2.218804	3.437404	34.10923	11.81575	7.093043	2	2
2.218142	3.437888	34.14038	11.81907	5.914488	2	3
2.216098	3.438161	34.21571	11.82095	5.998344	2	4
2.217473	3.438268	34.16252	11.82169	5.464407	2	5
2.201185	3.374278	34.1944	11.38575	6.164941	2	6
2.14593	3.348839	31.13356	11.21472	5.666283	3	1
2.142513	3.34976	31.03479	11.22089	5.734752	3	2
2.148063	3.350448	31.1692	11.2255	5.87679	3	3
2.143552	3.351323	31.06485	11.23137	6.083852	3	4
2.143696	3.351654	31.07774	11.23358	5.559548	3	5
2.146084	3.351727	31.12759	11.23408	6.237767	3	6
2.124817	3.384945	31.08494	11.45786	5.693481	4	1
2.136201	3.385923	30.89925	11.46447	5.725504	4	2
2.125315	3.385405	31.04373	11.46096	5.893042	4	3
2.131907	3.387165	31.47661	11.47289	6.011722	4	4
2.131883	3.387118	30.98032	11.47257	5.460275	4	5
2.124231	3.387028	31.08626	11.47196	6.216282	4	6
2.112323	3.222037	33.96971	10.38152	5.690018	5	1
2.1123	3.222823	33.98758	10.38659	5.836971	5	2
2.11314	3.223341	33.98111	10.38993	5.94868	5	3
2.112894	3.224016	33.995	10.39428	5.979046	5	4
2.113902	3.223638	33.93618	10.39184	5.542918	5	5
2.115113	3.223478	33.8847	10.39081	6.175134	5	6
2.062921	3.130994	31.8506	9.803126	5.923366	6	1
2.062335	3.130733	31.7285	9.80149	5.767743	6	2
2.061433	3.131925	31.77583	9.808956	6.104928	6	3
2.068412	3.131627	31.67461	9.807086	6.215631	6	4
2.065227	3.129534	31.57847	9.793985	5.498387	6	5
2.066591	3.130028	31.56942	9.797077	6.39226	6	6

Figure 6.54: TCN Measurements for our algorithm Data 2

Figure: 6.54 shows all measurements for TCN using dataset 2.

The *MAPE* columns show that the errors increased in dataset 2. This is expected because our data augmentation using the mean-value to fill the missing values is not accurate. We also observe that Bi-LSTM has the lowest *MAPE* with and without spatiotemporal changes while GRU is second. TCN has the fastest execution time but the worst *MAPE*. The two datasets required different parameters for each model and we ended up reducing the number of epochs for all experiments on dataset 2. So we can not cross-compare the execution times from one dataset to another dataset.

6.3 SUMMARY

We used two data preprocessing techniques to create two datasets of different size. We then applied the k-d tree algorithm to find the nearest neighbors from 1 to 6. After that, we built three neural networks models and tested the influence of spatiotemporal correlation on the *MAPE* estimation. Once the testing done, we analyzed the outputs to see the patterns in each result. Our proposed prepocessing technique was to help reduce the overfitting of the models and allow them to easily find the patterns which can lead to better estimations. However the results show that increasing the dataset size with dummy data also increases the error estimation and it does not effectively solve the overfitting problem. We are still satisfied with the dataset 2 because all the models still showed the same patterns in both datasets. That means that we did not hide any feature while increasing the data size. From all the observations above, we conclude that Bi-LSTM is optimal for predicting PM2.5 when spatiotemporal data is not considered. Once the spatiotemporal parameters get included, we see that the GRU model takes over and performs better than the Bi-LSTM. GRU is also faster than the Bi-LSTM. Finally, the TCN gets the least accuracy but it is the fastest of the three neural networks models. Unfortunately, it is the model which could not clearly extract the spatial correlation in the data. This is due to its

lack of recurrence. Between, the three models, we choose the GRU because its runtime is multiples times faster that the Bi-LSTM and its accuracy improves faster while extracting spatiotemporal correlation in the data.

CHAPTER 7

CONCLUSION

In this research, we built three neural networks models and used them to estimate the concentration of $PM_{2.5}$. We experimented two data augmentation techniques. One by dropping low sites observations and another technique by filling the low site observations with the mean of all the other observations. Our results prove that inserting mean value dummy data does not change the learning pattern but it increases the errors since those dummy values are not perfect substitutes of the true values. We were unable to prove that adding more days in the prediction can increase the accuracy for a one output neural network model but we showed that using the spatio-temporal technique proposed by Li and Co,[1] yields better results when we used more nearest neighbors. We were also able to verify that the Bi-LSTM model yields better results when time and data correlation are not taken in account. We also showed that even if the GRU model almost as accurate as the Bi-LSTM, and it even outperforms the Bi-LSTM when the execution time and the nearest neighbors are taken in account. Finally, showed that even if TCN is the least accurate of all three, it is also the fastest of the three models. We think that TCN, used as a sequential model should not be used to extract multiple spatial correlation features.

1. Tong et al., "Deep learning PM2.5 concentrations with bidirectional LSTM RNN."

CHAPTER 8

FUTURE WORK

In our models, we used stacking to build our deep neural networks models. There exist two ensemble neural network methods which could be used. They are called bagging and boosting techniques. An ensemble model is built by combining at least two different based models such as a TCN and a LSTM. Bagging is a method that focuses on combining multiples base models in parallel while boosting combines multiples base models sequentially. In the future, we think that using an ensemble neural networks to build hybrid neural networks and adding more features such as temperature, wind and other pollutant particles can increase the stability and accuracy of our model. We could also experiment on the accuracy of estimating one location at a time other multiples locations at once. We increased the error estimation when we used the mean-value to fill the dataset 2. Using an interpolation technique such as linear interpolation could have given better results. Finally, we could test the efficiency of the model in multivariate domains such as traffic data combined with meteorological data which could help better understand humans impact on the climate change.

REFERENCES

Agency, United States Environmental Protection. "Outdoor Air Quality Data," 2009. https://www.epa.gov/outdoor-air-quality-data/download-daily-data.

Agency, US Environmental Protection, and William K. Reilly. "State Implementation Plans; General Preamble for the Implementation of Title I of the Clean Air Act Amendments of 1990," November 1990. https://www.epa.gov/criteria-air-pollutants/naaqs-table.

Albawi, Saad, Tareq Abed Mohammed, and Saad ALZAWI. "Understanding of a Convolutional Neural Network." August 2017. https://doi.org/10.1109/ICEngTechnol.2017.8308186.

Bamane, Preeti, and Mangal Patil. "INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY A comparative study of different LSTM neural networks in predicting air pollutant concentrations." *Indian Journal of Science and Technology* 13 (November 2020): 3664. https://doi.org/10.17485/IJST/v13i35.1276.

Chen, Yitian, K Yanfei, C Yixiong, and W Zizhuo. "Probabilistic Forecasting with Temporal Convolutional Neural Network." *Neurocomputing* 399 (March 2020). https://doi.org/10.1016/j.neucom.2020.03.011.

Das S, Guo Zheng, Giles. "Using Prior Knowledge in a NNPDA to Learn Context-Free Languages." (College Park, MD), 1993, 5.

Devarakonda, Aditya, Maxim Naumov, and Michael Garland. *AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks*, 2018. arXiv: 1712.02029 [cs.LG].

Durak, B. C. *Artificial Neural Networks*. Accessed: 2018-03-27.

Goodfellow, Bengio, and Courville. "Deep Learning Book." (Cambridge, Massachusetts), 2016.

Howard Zucker, Commissioner, M.D. "Fine Particles (PM 2.5) Questions and Answers." *NY State Department of Health* 1, no. 1 (2018): 1–3. https://www.health.ny.gov/environmental/indoors/air/pmq_a.htm.

Huang, Chiou-Jye, and Ping-Huan Kuo. "A Deep CNN-LSTM Model for Particulate Matter (PM2.5) Forecasting in Smart Cities." *Sensors* 18 (July 2018): 2220. https://doi.org/10.3390/s18072220.

Karim, Guirguis, S Christoph, G Andre, and Co. "SELD-TCN: Sound Event Localization & Detection via Temporal Convolutional Networks." March 2020. https://doi.org/10.23919/Eusipco47968.2020.9287716.

L-C, Jain. "Recurrent neural networks: Design and Applications," 391. Boca Raton, FL: CRC Press, 2001.

Lai, Guokun, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. "Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks," March 2017.

Lea, Colin, Michael Flynn, Rene Vidal, Austin Reiter, and Gregory Hager. "Temporal Convolutional Networks for Action Segmentation and Detection," 1003–1012. July 2017. https://doi.org/10.1109/CVPR.2017.113.

Lea, Colin, V René, R Austin, and H Gregory. "Temporal Convolutional Networks: A Unified Approach to Action Segmentation." August 2016. https://doi.org/10.1007/978-3-319-49409-8_7.

Li, Lixin, Travis Losser, Charles Yorke, and Reinhard Piltner. "Fast Inverse Distance Weighting-Based Spatiotemporal Interpolation: A Web-Based Application of Interpolating Daily Fine Particulate Matter PM2.5 in the Contiguous U.S. Using Parallel Pro-

gramming and k-d Tree." *International journal of environmental research and public health* 11 (September 2014): 9101–9141. https://doi.org/10.3390/ijerph110909101.

Li, Lixin, and Peter Revesz. "A Comparison of Spatio-temporal Interpolation Methods," 145–160. September 2002. https://doi.org/10.1007/3-540-45799-2_11.

———. "Interpolation methods for spatio-temporal geographic data." (New York, NY, USA), 2004. https://doi.org/10.1016/S0198-9715(03)00018-8. https://www.elsevier.com/locate/compenvurbsys.

Li, Lixin, Jie Tian, Xingyou Zhang, James Holt, and Reinhard Piltner. "Estimating Population Exposure to Fine Particulate Matter in the Conterminous U.S. using Shape Function-based Spatiotemporal Interpolation Method: A County Level Analysis." *GSTF international journal on computing* 1 (September 2015): 24–30.

Li, Yaguang, Rose Yu, Cyrus Shahabi, and Yan Liu. "Graph Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting," July 2017.

Luo, Xianglong, Danyang Li, Yu Yang, and Shengrui Zhang. "Spatiotemporal traffic flow prediction with KNN and LSTM." *Journal of Advanced Transportation* 2019 (February 2019): 1–10. https://doi.org/10.1155/2019/4145353.

Manaswi, Navin Kumar. "Deep Learning with Applications Using Python." (Cambridge, Massachusetts), 2018, 219–227.

Merwin David, Cromley, and Co. "A Neural Network-based Method for Solving "Nested Hierarchy" Areal Interpolation Problems." *Cartography and Geographic Information Science* 36 (March 2013): 347–365. https://doi.org/10.1559/152304009789786335.

MIGLIONICO, MARCO. "A Deep Learning Framework for Air Pollution Forecasting and Interpolation," 14–15. July 2019. https://doi.org/10027/23862.

Qiao, Weibiao, Tian Wencai, Yu Tian, Quan Yang, Yining Wang, and Jianzhuang Zhang. "The Forecasting of PM2.5 Using a Hybrid Model Based on Wavelet Transform and an Improved Deep Learning Algorithm." *IEEE Access* PP (September 2019): 1–1. https://doi.org/10.1109/ACCESS.2019.2944755.

Rethage, Dario, Jordi Pons, and Xavier Serra. "A Wavenet for Speech Denoising," 5069–5073. April 2018. https://doi.org/10.1109/ICASSP.2018.8462417.

Revesz, Peter. "Spatiotemporal Interpolation Algorithms," 1–5. January 2014. https://doi.org/10.1007/978-1-4899-7993-3_803-2.

Schulter, Samuel, Paul Vernaza, Wongun Choi, and Manmohan Chandraker. "Deep Network Flow for Multi-Object Tracking," June 2017.

Schuster, Mike, and Kuldip Paliwal. "Bidirectional recurrent neural networks." *Signal Processing, IEEE Transactions on* 45 (December 1997): 2673–2681. https://doi.org/10.1109/78.650093.

Spengler, John D., and Ken Sexton. "Indoor air pollution: a public health perspective." *Web of Science.* (New York, NY, USA) 221, no. 4605 (July 1983). https://doi.org/10.1145/3161192. https://doi.org/10.1145/3161192.

Tong, Weitian, Lixin Li, Xiaolu Zhou, Andrew Hamilton, and Kai Zhang. "Deep learning PM2.5 concentrations with bidirectional LSTM RNN." *Air Quality, Atmosphere & Health* 12 (April 2019): 1–13. https://doi.org/10.1007/s11869-018-0647-4.

Wagner, Neal, Zbigniew Michalewicz, Sven Schellenberg, Constantin Chiriac, and Arvind Mohais. "Intelligent techniques for forecasting multiple time series in real-world systems." *International Journal of Intelligent Computing and Cybernetics* 4 (August 2011). https://doi.org/10.1108/17563781111159996.

West, J. Jason, and Co. “What We Breathe Impacts Our Health: Improving Understanding of the Link between Air Pollution and Health.” *Environmental Science & Technology 50 (10), 4895-4904*, 2016. https://doi.org/10.1021/acs.est.5b03827.

Zhou, Yanlai, Fi-John Chang, Li-Chiu Chang, I-Feng Kao, and Yi-Shin Wang. “Explore a deep learning multi-output neural network for regional multi-step-ahead air quality forecasts.” *Journal of Cleaner Production* 209 (October 2018). https://doi.org/10.1016/j.jclepro.2018.10.243.

Zhou, Zhi-Hua, and Wei Tang. “Selective Ensemble of Decision Trees,” vol. 2639. April 2003. https://doi.org/10.1007/3-540-39205-X_81.