UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

COMPARISON OF MACHINE LEARNING AND STATISTICAL APPROACHES FOR PREDICTING TRAVEL TIMES IN THE OKLAHOMA HIGHWAY SYSTEM

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

BY

Said Jalal Saidi
Norman, Oklahoma
2020

COMPARISON OF MACHINE LEARNING AND STATISTICAL APPROACHES FOR PREDICTING TRAVEL TIMES IN THE OKLAHOMA HIGHWAY SYSTEM

A MASTER'S THESIS APPROVED FOR THE GALLOGLY COLLEGE OF ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Dean F. Hougen

Dr. Andrew H. Fagg

Dr. Charles Nicholson

© Copyright by Said Jalal Saidi 2020
All Rights Reserved.

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Dean Hougen for his continuous support, guidance, and great knowledge in all the time of research and writing of this thesis. I can not thank him enough for providing me the opportunity to involve in various research projects.

Also, I would like to thank the rest of my thesis committee: Prof. Andrew Fagg and Prof. Charles Nicholson for their valuable feedback, encouragement, and invaluable constructive criticism.

Last but not the least, I would like to thank my parents: Said Kamal and Zahra Saidi; My wife and sons: Morsal, Yusof, and Yunos Saidi; And my brother: Dr. Jawad Saidi for supporting me spiritually throughout writing this thesis and all stages of my life.

Abstract

Traffic management systems play a vital role in supporting the smooth flow of traffic in road networks. By accurately predicting travel time, a traffic condition parameter that is extensively used in such systems, we can significantly improve the efficiency of these systems, decision-makers, and travelers. In this work, we use a dataset from the Oklahoma Department of Transportation to compare the accuracy of statistical and machine learning approaches to predicting travel time.

We establish baseline accuracy by constructing a traditional statistical model using the seasonal autoregressive integrated moving average (SARIMA) approach. We compare this baseline to two machine learning models: one-dimensional convolutional neural networks (1-D CNNs) and long short-term memory (LSTM) networks. Our results show that our 1-D CNN and LSTM models have better performance than the statistical model. As an example, in a 4-step architecture (a model structure that simultaneously predicts travel time four periods ahead), the median root means squared relative error (RMSRE) scores for our LSTM and 1-D CNN models are 0.060 and 0.063, respectively. These compare to the median RMSRE score of 0.12 for the corresponding 4-step SARIMA model. The results also indicate that the machine learning approaches have significantly lower computation time compared to SARIMA. In addition, the 1-D CNN model has the least error variance across all architectures and among all modeling meth-

ods. Finally, the 1-D CNN approach is more consistent in terms of prediction error across the experimented architectures compared to the LSTM appraoch. Therefore, based on the results, we highly recommend using machine learning approaches, specifically, 1-D CNNs, for estimating travel time in roadway systems and for other similar time-series prediction problems.

Contents

Acknowledgements **iii**

Abstract **v**

List of Figures **viii**

List of Tables **ix**

1 Introduction **1**
1.1 Research Objectives 3
1.2 Contribution to the Research Community 4
1.3 Organization of the Thesis 5

2 Background **6**
2.1 Glossary 6
2.1.1 Transportation Terms 6
2.1.2 Modeling Terms 8
2.2 Machine Learning with Artificial Neural Networks 18
2.2.1 Artificial Neuron 19
2.2.2 Artificial Neural Network 20
2.2.3 Dense and Sparse Layer 21
2.2.4 Feedforward Neural Network 22
2.2.5 Backpropagation Learning Algorithm 22
2.2.6 Recurrent Neural Network 23
2.2.7 Long Short-Term Memory 24
2.2.8 One-Dimensional Convolutional Neural Networks 27
2.2.9 The Max-Pooling Operation 28
2.3 Related Work 29

3 Data **32**
3.1 Datasets 32
3.1.1 Travel Time Dataset 32
3.1.2 TMC Dataset 33

3.2 Data Preparation 33
3.3 Feature Engineering 35

4 Methodology **38**
4.1 SARIMA Approach 39
4.1.1 Urban Code 43
4.2 Machine Learning Approaches 43
4.3 LSTM Approach 45
4.3.1 LSTM Model Architecture 46
4.3.2 LSTM Input and Output Dimension 47
4.3.3 Number of Trainable Parameters in LSTM 47
4.3.4 Training the LSTM Model 49
4.4 1-D CNN Approach 50
4.4.1 1-D CNN Model Architecture 50
4.4.2 1-D CNN Input and Output Dimensions 51
4.4.3 Number of Trainable Parameters in 1-D CNN 51
4.4.4 Training the 1-D CNN Model 53

5 Results and Discussion **55**
5.1 Results for SARIMA 55
5.1.1 Rural Area SARIMA Model 56
5.1.2 Small Urban SARIMA Model 57
5.1.3 Large Urban SARIMA Model 58
5.1.4 Discussion for the SARIMA Models 59
5.2 Results for LSTM 60
5.3 Results for 1-D CNN 61
5.4 Comparison of the Proposed Models 63

6 Conclusions and Future work **70**

Bibliography **74**

List of Figures

2.1 Example of Internal and External TMCs 8
2.2 Travel Time Estimation when a Traveler Traverse a Route From Point A to Point B 8
2.3 Distribution of TMC Length in 2018 18
2.4 Artificial Neuron 20
2.5 Artificial Neural Network with an Input, Hidden, and Output Layer 21
2.6 Dense and Sparse Layers 22
2.7 A Compact (Left Side) and Expanded (Right Side) Form a Simple RNN 24
2.8 A Sample LSTM Network 25
2.9 A Detailed LSTM Cell 27
2.10 Performing One-Dimensional Convolution Operation on an Input Sequence Results in a Feature Sequence 28
2.11 Performing Max Pooling Operation Downsamples the Feature Sequence 29

3.1 Travel Time Distribution (Before Transformation) 36
3.2 Speed Distribution (After Transformation) 36
3.3 Travel Time Plot of a TMC (Before Transformation) 37
3.4 Speed Plot of a TMC (After Transformation) 37

4.1 Display 24-hour Seasonality Using Time Plots 40
4.2 Display 24-hour Seasonality Using ACF Plots 40
4.3 PACF and ACF Plots Before Differencing Operation 41
4.4 PACF and ACF Plots After Differencing Operation 41
4.5 Data Generator Uses a Time Sequence for Creation of Lag-Step Pair 45

5.1 SARIMA MRE and RMSRE Results for Rural Areas 57
5.2 SARIMA MRE and RMSRE Results for Small Urban Areas . . . 58
5.3 SARIMA MRE and RMSRE Results for Large Urban Areas . . . 59
5.4 The Proposed LSTM Model's MRE and RMSRE Results 61
5.5 The Proposed 1-D CNN Model's MRE and RMSRE Results . . . 63
5.6 Comparison of 1-D CNN, LSTM, and SARIMA Results 64

List of Tables

4.1 Proposed LSTM Architecture 47
4.2 Proposed 1-D CNN Architecture 51

5.1 Summary of MRE and RMSRE Errors for 1-Step Architecture . . 65
5.2 Summary of MRE and RMSRE Errors for 2-Step Architecture . . 65
5.3 Summary of MRE and RMSRE Errors for 3-Step Architecture . . 66
5.4 Summary of MRE and RMSRE Errors for 4-Step Architecture . . 66
5.5 Statistical Analysis of Mean and Variance of the Errors for the 1-Step Architecture for 1-D CNN & LSTM 68
5.6 Statistical Analysis of Mean and Variance of the Errors for the 2-Step Architecture for 1-D CNN & LSTM 68
5.7 Statistical Analysis of Mean and Variance of the Errors for the 3-Step Architecture for 1-D CNN & LSTM 68
5.8 Statistical Analysis of Mean and Variance of the Errors for the 4-Step Architecture for 1-D CNN & LSTM 69

Chapter 1

Introduction

Travel time is the time cost for a traveler to commute between any two points of interest in a road segment. *Travel time prediction* refers to predicting and calculating the experienced travel time before a vehicle has traversed the route of interest (Billings and Yang, 2006). Travel time is an important roadway traffic condition parameter, which is comprehensible both for motorists and transportation officials (Zhang and Haghani, 2015). Travelers can make better decisions regarding the departure time, route selection, and mode choice by obtaining reliable travel time estimates from advanced traveler information systems (Zhang and Haghani, 2015). Additionally, travel time prediction assists transportation decision-makers in developing advanced traffic management system strategies proactively (Zhang and Haghani, 2015). Researchers have explored various methods for travel time prediction. Typically, their work follows two main paradigms: parametric approaches and non-parametric approaches (Ma et al., 2015). In *parametric* or *model-based* travel time prediction methods, the structure of the model and its parameters are predetermined based on theoretical assumptions (Ma et al., 2015). Traffic simulation models are examples of model-based approaches. These

models require building a virtual road network and performing dynamic traffic simulation on it by utilizing theories such as traffic flow (Duan et al., 2016; Ma et al., 2015). There are some shortcomings associated with theoretical models that make them ineffective for the modern traffic systems. Developing models for ideal situations without considering the dynamic nature of human behaviors are instances of these weaknesses (Ma et al., 2015). Autoregressive integrated moving average (ARIMA) models are time-series analysis models that are part of the parametric approach family (Adhikari and Agrawal, 2013). ARIMA models are proposed for travel time prediction with reasonable accuracy (Billings and Yang, 2006). Also, a modified version of ARIMA is used to estimate traffic speed and volume (Chandra and Al-Deek, 2009).

Non-parametric methods, on the other hand, do not consider fixed structures or parameters for travel time predictive models (Ma et al., 2015). *Machine learning models* are non-parametric models that are widely used in travel time prediction. Linear machine learning can provide effective modeling approaches to travel time problems (Zhang and Rice, 2003; Rice and Van Zwet, 2004). *Linear machine learning* models are model types that utilize a linear combination of features to explain travel time. K-nearest neighbor is one of the linear machine learning approaches that is used to predict travel time (Myung et al., 2011). *Artificial neural networks* (ANNs) are machine learning-based models that have several advantages such as flexible model structure, generalization, and learning capability (Ma et al., 2015). The power of ANNs have made them a model of choice to address the travel time prediction problem (Ma et al., 2015). *Long short-term memory* (LSTM) neural networks (Hochreiter and Schmidhuber, 1996) provide a modeling approach suitable for sequence processing problems such as natural language processing (Chen et al., 2016), machine translation, image and video

captioning (Vinyals et al., 2015; Gao et al., 2017), and time-series forecasting (Sagheer and Kotb, 2019). LSTMs are applied to predict the travel time (Duan et al., 2016) as well as travel speed (Ma et al., 2015), which are both important traffic factors. The results indicate that LSTM networks are successful to model and predict travel time due to their ability to capture long time dependencies within travel time sequences. *Convolutional neural networks* (CNNs) (Albawi et al., 2017) are another type of ANN that has outstanding performance in machine learning problems, specially for computer vision. Some examples are image classification (He et al., 2016), object detection (Redmon et al., 2016), and image reconstruction (Liu et al., 2018). A *one-dimensional convolutional neural network* (1-D CNN) is a type of CNN for signal processing applications such as electrocardiogram processing (Kiranyaz et al., 2015, 2017). 1-D CNNs are also applied to sequence processing and time series problems (Chollet, 2017). One study indicate that a combination of 1-D CNNs and LSTMs results in powerful predictive models for travel time information (Petersen et al., 2019).

In this work, we compare and report the performance of three models both from parametric and non-parametric approaches. We have chosen LSTMs and 1-D CNNs from the non-parametric models. To the best of our knowledge, this is the first time that pure 1-D CNN architecture is used for travel time prediction. For the parametric models, we have tested the predictive capabilities of seasonal autoregressive integrated moving average (SARIMA) models these are equivalent to ARIMA models equipped with seasonality factor.

1.1 Research Objectives

The objectives of this research are to:

1. Prepare travel time data to be used in three modeling processes: SARIMA, LSTM, and 1-D CNN
2. Develop SARIMA, LSTM, 1-D CNN models to predict multi-step travel time
3. Evaluate the accuracy of each modeling approach using the test data across different architectures for different steps in future
4. Analyze the results for each modeling approach based on:
   - Accuracy of the prediction across different architectures
   - Sensitivity to large errors across different architectures
5. Compare the results within each modeling approach and determine the best performing architecture
6. Compare the modeling approaches among themselves and determine the best performing model(s) and/or architecture(s)

1.2 Contribution to the Research Community

In this work, I have experimented with three modeling approaches to predict travel time in the highway network of Oklahoma. Theses approaches include two machine learning techniques, namely 1-D CNN and LSTM, and one statistical approach called SARIMA. The proposed LSTM method is based on a work for travel time prediction from the literature (Duan et al., 2016). I ensured that the 1-D CNN has the same number of layers as the LSTM model. The result of this work shows that both LSTMs and 1-D CNNs outperform the traditional time-series modeling approach, SARIMA. This research shows that 1-D CNNs can

be utilized to solve the similar sequence-based or time series problems. Also, it shows that applying appropriate feature engineering such as transforming travel time to speed can be beneficial for having accurate travel time predictions.

1.3 Organization of the Thesis

This thesis is organized in 6 chapters. Chapter 1 overviews the travel time problem. Chapter 2 introduces the key technical terms and concepts required to understand the approaches, results, and discussions and also it reviews the literature related to this work. Chapter 3 explains the travel time datasets used in the modeling process as well as a transformation applied to them. Chapter 4 explains the methodologies utilized in the modeling process including 1-D CNN, LSTM, and SARIMA. Chapter 5 describes and compares the results of our experiments on our models and analyzes the results. Finally, Chapter 6 concludes the thesis and presents future work.

Chapter 2

Background

This chapter explains key terms and concepts, as well as previous work related to traffic condition prediction, that are required to understand the rest of the thesis. This chapter is divided into three sections. The first section is a glossary introducing transportation and modeling terms. The second section explains the terms and fundamental concepts of artificial neural networks. In the last section, previous research related to traffic prediction is reviewed.

2.1 Glossary

The glossary is divided into two sections, one for transportation terms and one for machine learning terms.

2.1.1 Transportation Terms

In this part, we define the transportation terms that are used throughout the thesis.

NPMRDS

National Performance Management Research Data Set (NPMRDS) is the reference dataset designed to meet federal congestion and freight performance reporting requirements (Bitar, 2016). NPMRDS provides comprehensive and consistent data for passenger and commercial freight roadway performance across the National Highway System, as well as over 25 key Canadian and Mexican border crossings.

Traffic Message Channel

According to Federal Highway Administration (FHWA)- Department of Transportation (2020), NPMRDS uses the *Traffic Message Channel* (TMC) standard as a unique identifier for each road segment. TMCs have unique codes composed of nine characters. For the purpose of brevity, the first three characters of the TMC codes are omitted here. TMCs represent a road section from one exit or entrance ramp to the next, which is either internal or external. An internal TMC represents a portion of a road within an interchange (e.g., between an exit ramp and entrance ramp). An external TMC represents a stretch of road between interchanges. The first character of TMC in Figure 2.1 denotes the type and direction of TMCs, which are described as the following:

- "P" denotes northbound or eastbound internal TMC.
- "N" denotes southbound or westbound internal TMC.
- "+" denotes northbound or eastbound external TMC.
- "-" denotes southbound or westbound external TMC.

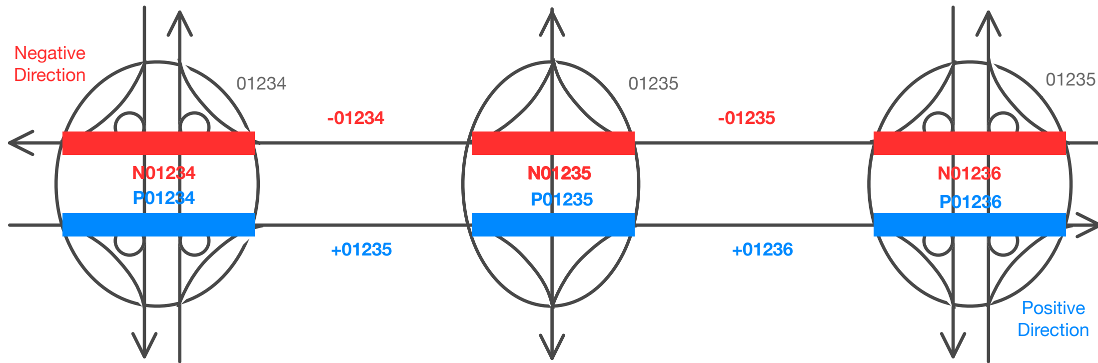

Figure 2.1: Example of Internal and External TMCs

Travel time

Travel time x_t is the time cost to traverse road segment L, from point A to point B, departing at time t, as shown in Figure 2.2 (Duan et al., 2016). In this thesis, we use travel times of TMCs in the Oklahoma highway network.

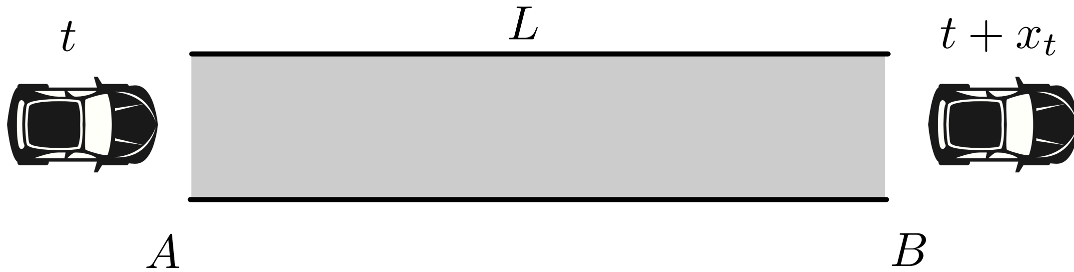

Figure 2.2: Travel Time Estimation when a Traveler Traverse a Route From Point A to Point B

2.1.2 Modeling Terms

In this part, we explain the modeling terms and concepts we have used in this work.

ARIMA Model

Autoregressive integrated moving average (ARIMA) is popular statistical model for forecasting and analysis of timeseries datasets. An ARIMA model is made up of three components, an autoregressive part denoted by AR, an integrated part denoted by I, and a moving average part represented by MA.

We first focus on the AR model. This model assumes that the predicted variable is a linear combination of the same variable at the previous time steps. Let p be the number of time steps in the past. AR(p) or *autoregressive model of order* p means that the variable is regressed against itself using the value of the same variable for p times in the past. The equation for AR(p) is written as

$$y_t = c + \sum_{i=1}^{p} \varphi_i y_{t-i} + \epsilon_t$$

where y_t and ϵ_t are the actual value of the predicted variable and the random error at current time step t, respectively, c is a constant term, and φ_i is the coefficient of the equation at previous timestep i, where ($i = 1, 2, \dots, p$). The tuning of an AR model consists of tuning φ_i to approximate the data. This equation can be written more concisely by introducing the Lag operator L as

$$L^n y_t = y_{t-n}$$

where n is the number of previous lags. The AR process can be written as an order p polynomial function L by

$$y_t = \Theta(L)^p y_t + \epsilon_t$$

where Θ is polynomial operator for the AR process. A polynomial operator transforms an equation to a polynomial expression form. Please take note that the constant c has been absorbed into the polynomial.

Second, in a given sequence, the integrated part calculates the difference of an entry in the sequence with its previous entry. Then, the value of the entry is replaced by the result of the differencing operation. This process is applied to all entries. The *order* of the integrated part determines how many times the differencing operation should be applied to a sequence consecutively. I(d) or *integrated order* of d replaces an item in a sequence with the value differenced for d times. We introduce an integration operator Δ^d the integrated component uses and define the equation for I(d) using

$$\Delta^d y_t = y_t^{[d]} = y_t^{[d-1]} - y_{t-1}^{[d-1]}$$

where $y_t^{[0]} = y_t$ and d is the order of differencing used. The expanded form is

$$\begin{aligned} y_t^{[1]} &= y_t - y_{t-1} \\ y_t^{[2]} &= y_t^{[1]} - y_{t-1}^{[1]} \\ &\vdots \\ y_t^{[d]} &= y_t^{[d-1]} - y_{t-1}^{[d-1]}. \end{aligned}$$

In each differencing operation, the number of entries is reduced by one. The differencing technique is employed for transforming a non-stationary series to stationary. A time series is called *stationary* if the summary statistics calculated over time are consistent. A stationary time series has one main characteristic where the calculated summary statistics over time are consistent. Summary

statistics can be the mean or the variance. If the time series shows trends or seasonality then it is non-stationary. In *non-stationary* time series the summary statistics change as time passes. Since classical time series methods perform well on stationary data, integrated use differencing of raw observations to remove the trend and support time series data with a trend.

Third, the moving average model leverages the past forecast errors in a regression-like model. Let q be the number of time steps in the past. MA(q) or *moving average of order* q, uses the errors in q previous time points to predict current and future values. The equation for MA(q) is written as

$$y_t = \mu + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t$$

where y_t and ϵ_t are the actual value of the predicted variable and the random error at current time step t, respectively. Also, μ is the mean of the time series and θ_j is the coefficient of the equation at previous timestep j, where (j = 1, 2, ... , q). The tuning of an MA model consists of tuning θ_j to approximate the data. The MA process can be written as

$$y_t = \Phi(L)^q \epsilon_t + \epsilon_t$$

where Φ is the polynomial operator for the MA process.

Finally, the complete ARIMA process can be obtained by combining the AR, I, and MA processes and is written as

$$\Delta^d y_t = \Theta(L)^p \Delta^d y_t + \Phi(L)^q \Delta^d \epsilon_t + \Delta^d \epsilon_t.$$

The equation can be further simplified as

$$\Theta(L)^p \Delta^d y_t = \Phi(L)^q \Delta^d \epsilon_t.$$

SARIMA Model

Seasonal autoregressive integrated moving average (SARIMA) is an extension of ARIMA that supports the seasonality of time series datasets. SARIMA models consider seasonality by applying an ARIMA model to lags that are integer multiples of seasonality. Once the seasonality is modelled, an ARIMA model is used to formulate non-seasonal part. The seasonal ARIMA model is formulated as SARIMA$(p, d, q)(P, D, Q)_s$, where p, d, q are from the ARIMA process and P, D, Q, and s constructs the seasonal part of the model.

First we introduce the seasonal differencing operator Δ_s^D to take the seasonal differences of the time series. Here s is the number of time lags comprising one full period of seasonality. D is similar to d in ARIMA models, but instead applies to seasonal lags. We also introduce the seasonal lag operator L^s. Using seasonal lags P and Q in ARMA (the combination of both AR and MA) process results

$$\Delta_s^D y_t = \theta(L^s)^P \Delta_s^D y_t + \phi(L^s)^Q \Delta_s^D \epsilon_t + \Delta_s^D \epsilon_t$$

where θ and ϕ are the polyonimal operators for the seasonal AR and MA, respectively. The equation can be written in concise form as

$$\theta(L^s)^P \Delta_s^D y_t = \phi(L^s)^Q \Delta_s^D \epsilon_t.$$

We include the non-seasonal part to have to complete SARIMA equation as

$$\Theta(L)^p \theta(L^s)^P \Delta^d \Delta_s^D y_t = \Phi(L)^q \phi(L^s)^Q \Delta^d \Delta_s^D \epsilon_t.$$

Autocorrelation and Partial Autocorrelation Functions

The *autocorrelation function* (ACF) of n determines the correlation of the current time-series variable with the values of same variable at n previous time steps. The *partial autocorrelation function* (PACF) is similar to ACF, but it only considers the direct correlation between two steps by removing the implicit influence of other time series values in between. For example, if $n = 5$, ACF finds the correlation between the current time series and the time series at the 5 previous time steps, while the indirect influence of 4 steps in between are preserved. On the other hand, the PACF disregards the implicit effect of the other 4 values and calculates the direct correlation value. ACF and PACF plots are used to estimate AR and/or MA lags.

Learning Algorithm

An algorithm that can learn from data is called a *learning algorithm* (Goodfellow et al., 2016). According to Mitchell (1997) "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." The terms task, performance measure, and experience are the next terms explained.

Task

A *task* is a type of job that is expected from a machine learning algorithm to perform. The task can be any problem that is hard to address by the traditional, fixed, and human-designed computer programs (Goodfellow et al., 2016). There are various types of tasks that machines can learn including but not limited to classification and regression problems. In *classification*, the algorithm identifies and/or predicts the category of a given item. For example, categorizing objects within an image is a classification problem. *Regression* is predicting a numerical value for a given input. The prediction of the amount of sales for the next day, given the sales information for the past week, is an example of regression. In this document, the task we are working on is a regression problem because we attempt to predict the travel time which is a numerical value.

Experience

Learning algorithms can have a different experience of the data they encounter. There are two broad types of learning algorithm depending on how the algorithm experiences the data, which are supervised and unsupervised (Goodfellow et al., 2016). In *supervised* learning the algorithm is presented with a dataset containing input and output vectors. A *vector* is an ordered collection of items. An *input vector* is a vector containing the data used as an input for the learning algorithm. An *output vector* is a vector of data associated with the input vector. In an *annotated* dataset, the input and output vectors are explicitly tied together. The supervised learning algorithm uses the annotated data to discover the relationship between the input and output vectors. The dataset used in the learning process is called a *training set*. On the other side, in *unsupervised* learning, the dataset

is not annotated. The algorithm attempts to discover patterns from the dataset it is presented with. *Reinforcement learning* is another learning paradigm in which the learner (agent) learns through interaction with the environment. The agent takes an action based on its current situation (state) and receives feedback from the environment or an external evaluator. The feedback can be in form of a reward (positive), punishment (negative), or neutral. The goal of the agent is to gather the maximum possible reward through a set of trials and errors. The feedback mechanism helps the agent to learn which action is best associated with which state. In this document, we are investigating various methods' ability to predict travel time in the highway network of Oklahoma using an annotated travel time dataset. Therefore, the machine learning algorithms used to address this problem are supervised algorithms.

Performance Measure

A *performance measure* is a systematic way to quantitatively measure how well the algorithm performs the task. The design of the performance measure highly depends on the underlying task performed by the learning algorithm. For example, performance measures used in regression problems are quite different from the ones used in classification problems. Performance measures help the learning algorithm to automatically adjust the learning process. Travel time prediction is a regression problem. We use mean squared error in the learning process. The *mean squared error* (MSE) is the average of the square of the errors. The *error* is the difference between the ground truth and the prediction. The MSE is computed as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

where n is the number of samples, Y_i is the ground truth (the output vector in an annotated dataset explained in the Experience section), and $\hat{Y}_i$ is the predicted value. MSE includes a quadratic term, which helps by emphasizing large errors and adjusting the learning algorithm accordingly.

Evaluation metrics

In this document we develop two types of models, machine learning and statistical. Typically, a machine learning process includes two stages, the training and testing. The *training* stage includes the process of constructing the model. The *testing* stage includes the process of evaluating the model once it is constructed. Therefore, the dataset used in the learning process is divided for training and testing purposes. The test data is never utilized in the training process; therefore, the testing procedure can objectively determine how the constructed model performs beyond the training data. In this document, *performance measure* is a term used to evaluate and guide the training process. One the other hand, *evaluation metric* or *metric* is used to evaluate the testing process. Evaluation metrics help comparing the final results of different modeling approaches. We define two evaluation metrics, mean relative error (MRE) and root mean squared relative error (RMSRE). *Mean relative error* (MRE) is calculated as

$$\mathrm{MRE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i - \hat{Y}_i|}{Y_i}$$

where n is number of samples, Y_i is the ground truth, and $\hat{Y}_i$ is the predicted value. The *root mean square relative error* (RMSRE) is calculated as

$$\text{RMSRE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{Y}_i}{Y_i}\right)^2}$$

where n is the number of samples, Y_i is the ground truth, and $\hat{Y}_i$ is the predicted value.

We use two evaluation metrics to address two issues. The first issue is the variation of errors caused by different length TMCs. For example, Figure 2.3 shows the distribution of TMC lengths in 2018. We can see some TMC lengths beyond 15 miles, whereas most TMC lengths are less than 2 miles. To mitigate this concern, we employ relative metrics such as MRE and RMSRE instead of absolute metrics. The *absolute metrics* are calculated using the magnitude of the difference between the ground truth and the prediction. They have the same unit as the problem domain and are therefore, easier to interpret. However, in this context, they are considerably correlated with the lengths of TMCs. The *relative errors* are also based on the absolute errors but normalized by the magnitude of the ground truth. They can help us understand the system's prediction error compared to actual travel time values. The second concern for the system is the disparity of predictions. RMSRE helps us to understand whether the system produces large errors. It includes a quadratic term, which results in exaggerating larger errors. Using RMSRE, not only can we compare different approaches, we can analyze a single model's behavior. For example, we can objectively determine the difference between the MRE and RMSRE in a specific model.

The reason that we only employ MSE during training is that MSE is an

absolute metric. As discussed in Section 3.3, we create a new feature for the training process by applying the TMC length to the travel time data (training data). After developing the model, during the testing phase, we transform the model prediction from the new feature to travel time by applying TMC length. Therefore, as discussed earlier, we need relative metrics to handle error variation caused by TMC length.

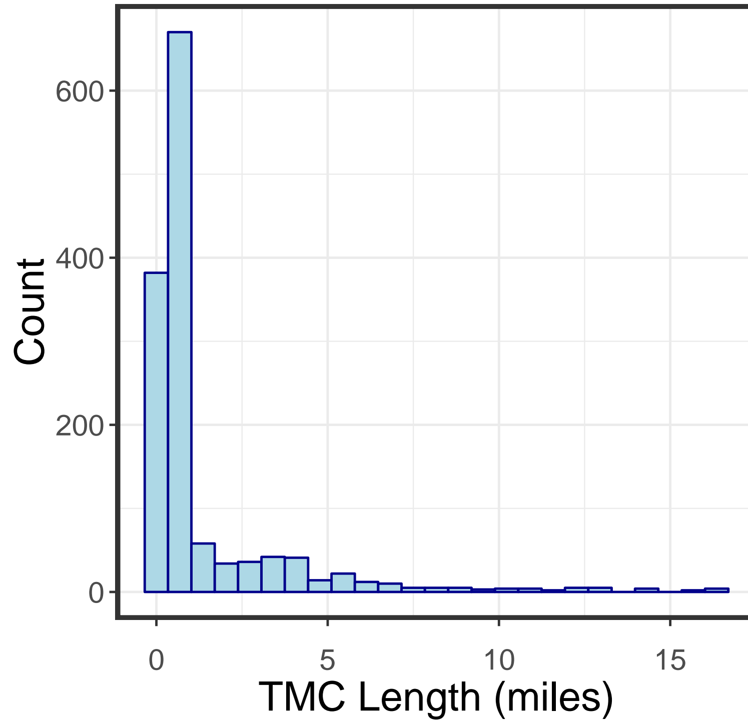

Figure 2.3: Distribution of TMC Length in 2018

2.2 Machine Learning with Artificial Neural Networks

In this work, the two machine learning approaches we have utilized are based on artificial neural networks. The following subsections explain the required concepts

to understand those approaches.

2.2.1 Artificial Neuron

An *artificial neuron* (AN) is a computational model inspired by biological nerve cells or neurons (Engelbrecht, 2007). As illustrated in Figure 2.4, the AN receives input via various connections from the environment or other ANs and computes and fires the output signal. Let x_i be the value for an input connection i, for $1 \leq i \leq n$, where n is the number of input connections. Each input connection has a *weight* that influences the strength of an input signal. Let w_i be the weight for input connection i, for $1 \leq i \leq n$, where n is the number of input connections. To calculate the output signal, the neuron first computes the *net* value. There are two primary types of ANs in terms of calculating the net value, *summation units* and *product units*. Summation units calculate the net value as

$$net = \sum_{i=1}^{n} x_i w_i.$$

Product units calculate the net value as

$$net = \prod_{i=1}^{n} x_i^{w_i}.$$

ANs have a *threshold* or *bias* value that influences the net value. ANs calculate the output signal by subtracting the bias from the net value and feeding it to an *activation function*. An activation function is a linear or non-linear transformation function that receives the net value and determines the AN's output strength (Engelbrecht, 2007). Denote the threshold θ and the activation function

with f_{AN}. The output signal o is

$$o = f_{AN}(net - \theta).$$

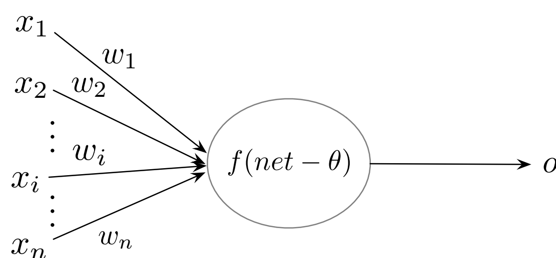

Figure 2.4: Artificial Neuron

2.2.2 Artificial Neural Network

An *artificial neural network* (ANN) is a computational network typically with one or more layers of ANs (Figure 2.5). ANNs are inspired by the decision process in networks of nerve cells of the biological central nervous system (Graupe, 2013). According to Haykin et al. (2009), an ANN is "a machine that is designed to model the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on a digital computer." ANNs are typically composed of three types of layers of ANs: input layers, output layers, and hidden layers. The *input layer* is the first layer and entry point in the ANN. This layer accepts initial input from the environment and sends it to the next layer for further processing. *Hidden layers* are located after the input layer. ANs in the hidden layers perform transformations of the input they receive. The transformation operation depends on various factors including the type of the activation function. In Figure 2.5, there is only one hidden layer; however, ANNs can have more than one hidden

layer (Engelbrecht, 2007). The final layer in the ANN is the *output layer*. This layer determines the final result of the network.

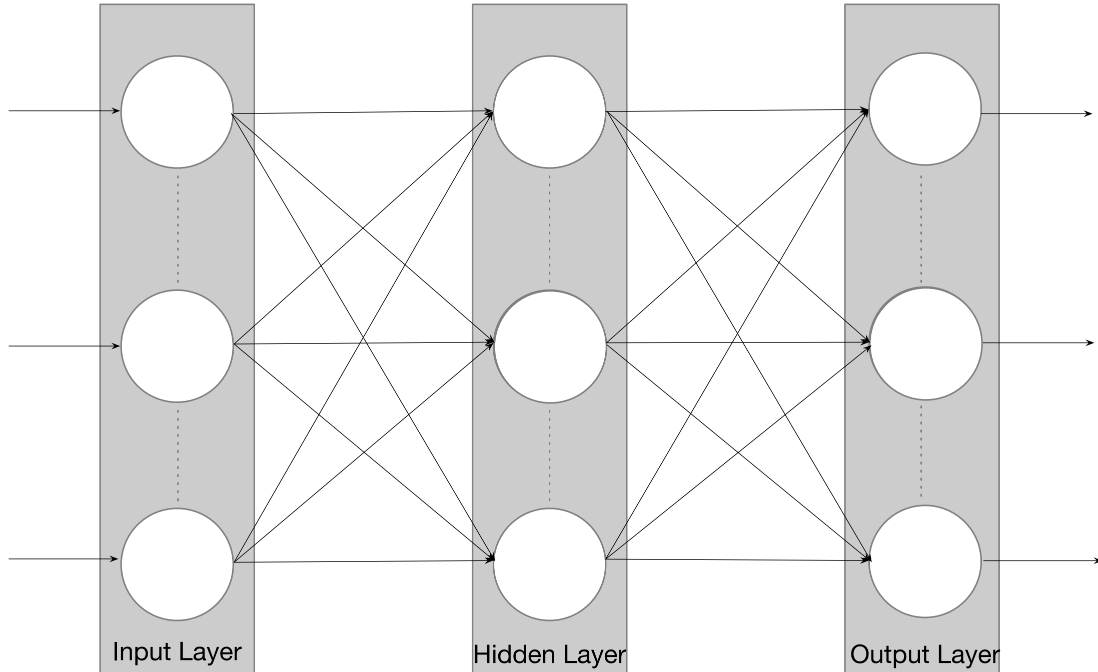

Figure 2.5: Artificial Neural Network with an Input, Hidden, and Output Layer

2.2.3 Dense and Sparse Layer

The ANs located in an ANN layer can connect to the ANs in the previous layer either completely or partially. A layer is called *dense* or *fully-connected* if all ANs within that layer have a connection to all of the ANs in the previous layer. It means that all ANs within the dense layer receive input from all of the ANs in the previous layer. A layer is *sparse* if one or more of its ANs are not connected to at least one or more of the ANs within the previous layer. Typically, the sparse layers lack a considerable number of connections from the previous layers. Figure 2.6 illustrates a dense and sparse layer side by side.

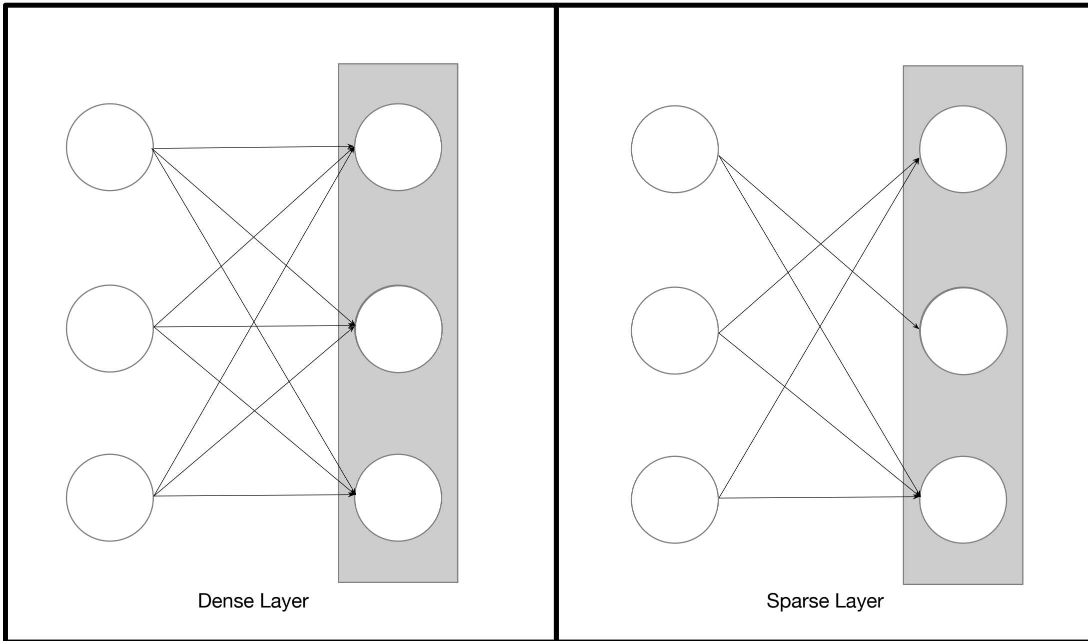

Figure 2.6: Dense and Sparse Layers

2.2.4 Feedforward Neural Network

A *feedforward neural network* (FFNN) is a type of ANN that takes the input signal from the environment and propagates it throughout the network to produce the output (Engelbrecht, 2007). The final output of an FFNN for any input pattern is calculated with a single forward pass through the network that starts with the input layer and ends with the output layer (Engelbrecht, 2007). Layers in FFNNs don't have feedback connections to their previous layers (Engelbrecht, 2007).

2.2.5 Backpropagation Learning Algorithm

One of the popular algorithms used in FFNNs is *backpropagation* (Rumelhart et al., 1986; Engelbrecht, 2007). Before starting the learning process, the candidate weights and biases of the network are typically randomly initialized. Then,

the candidate weights and biases are adjusted through learning iterations. Each learning iteration in backpropagation is called an *epoch*. Typically, an epoch includes two stages:

1. *Forward propagation*: As explained in the previous section, the input samples are passed through the FFNN and the actual output of the network is calculated.

2. *Backward propagation*: In this phase, a function called a *loss function* calculates the error of the network. An *error* is the difference between the actual output of the network and a target output values associated with the input samples. The loss function is equipped with a performance metric and helps the learning algorithm to choose a set of weights and biases with minimum error. Finally, the calculated error value in the output layer is propagated back toward the input layer and the weights and biases of the neurons in the network are updated based on the error (Engelbrecht, 2007). In this work, we use MSE as the loss function during the training process.

2.2.6 Recurrent Neural Network

A *recurrent neural network* (RNN) is a type of network useful for processing sequences and temporal data. RNNs iterate through sequences and maintain a hidden state containing information about all the items they have encountered so far. Figure 2.7 depicts a simple RNN; the left side represents the compact form of the RNN and the right side represents the form expanded through all time steps. At timestep t, the hidden state of the network h_t is calculated by applying the activation function F to the input x_t and the hidden state from the

previous timestep h_{t-1}. This calculation method in the RNN provides an internal memory, retaining the influence of the previous time step on the next one.

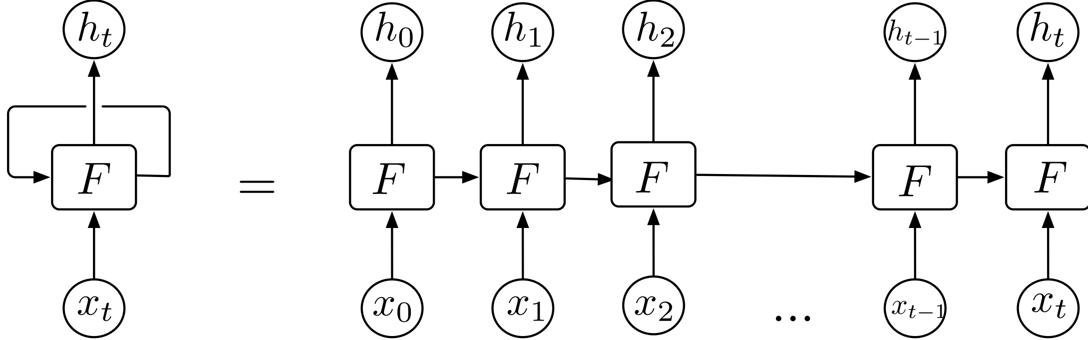

Figure 2.7: A Compact (Left Side) and Expanded (Right Side) Form a Simple RNN

2.2.7 Long Short-Term Memory

A *long short-term memory* (LSTM) neural network is a type of RNN which has special building units called LSTM cells (Figure 2.8). LSTM cells calculate and maintain two hidden states, namely cell state c and cell output h. Having two hidden states allows LSTMs to address the issue of *vanishing gradients*, an effect in which networks are unable to preserve information about previous time steps. This issue is common in traditional RNNs and also very deep non-recurrent neural networks. LSTMs carry the cell state c across many time steps preventing older information from gradually fading during processing (Chollet, 2017).

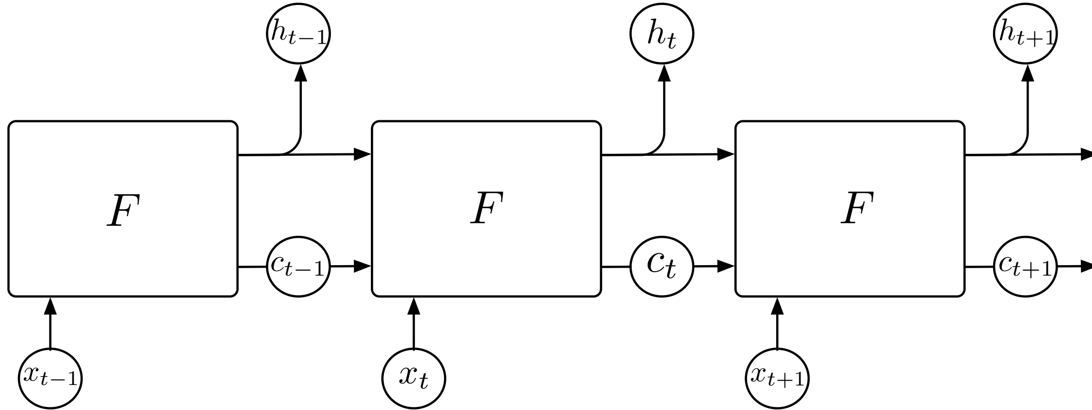

Figure 2.8: A Sample LSTM Network

To calculate the cell state and output state, LSTM cells incorporate an internal calculation mechanism called gates. The LSTM cell has four gates, which are the input gate, forget gate, output gate, and input modulation gate (Figure 2.9). These gates regulate the flow of information through the cell and neural network (Duan et al., 2016). At time step t, the *input gate* i_t is calculated using

$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right)$$

the *forget gate* f_t is calculated using

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right)$$

the *output gate* o_t is calculated using

$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right)$$

and the *input modulation gate* $\widetilde{c}_t$, which is the preliminary step to calculate the

cell state c, is calculated using

$$\widetilde{c}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right)$$

where the x_t is the input at time t, h_{t-1} is the output state at the previous timestep, W_i, W_f, W_o, and W_c are weight matrices for the input x_t vector in the four gates, U_i, U_f, U_o, and U_c are the weight matrices for the previous output state h_{t-1} in the four gates, b_i , b_f, b_o, and b_c are the bias terms in the four gates. Note that σ symbolizes the logistic sigmoid function $\frac{1}{1+e^{-x}}$ and tanh represents the hyperbolic tangent function $\frac{e^x - e^{-x}}{e^x + e^{-x}}$. After the calculation of the gates, the cell state c_t is calculated using

$$c_t = i_t \odot \widetilde{c}_t + \ f_t \odot c_{t-1}$$

where c_{t-1} is the cell state at the previous timestep. Please note that $\odot$ represents the element-wise multiplication operator. At the last stage, the cell output is calculated as

$$h_t = o_t \odot \tanh\left(c_t\right)$$

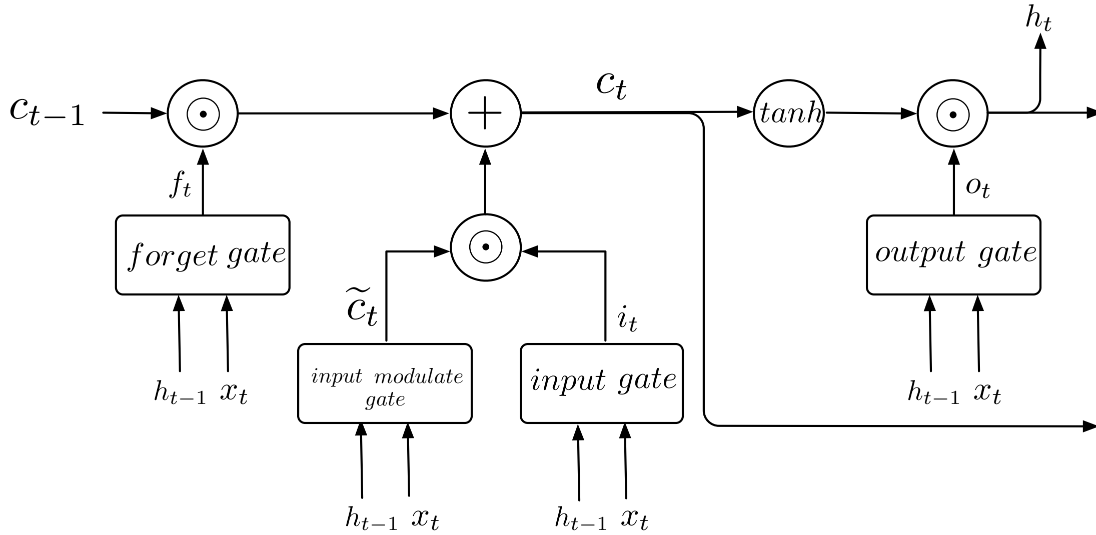

Figure 2.9: A Detailed LSTM Cell

2.2.8 One-Dimensional Convolutional Neural Networks

We use *one-dimensional convolutional neural networks* (1-D CNNs) to process temporal data and sequences. In 1-D *convolutions*, we extract 1-D *patches* (sub-sequences) from the input sequence (Figure 2.10). Then we calculate the dot product of each patch with a 1-D vector called the *kernel* or *filter*. The kernel essentially contains the weights that the CNN learns, and it has the same length as the extracted 1-D patch. The scalar results of each dot product are lined up and result in another sequence called the *feature sequence*, which has a smaller length than the input sequence. Each index m of the feature sequence R is calculated according to the formula

$$R[m] = (S \cdot K)[m] = \sum_{i=0}^{w-1} S[m+i] \ \ K[i]$$

where S is the input sequence, and K is the kernel, w is the length of the kernel. If the input sequence has a length of l, the feature sequence will be $l - w + 1$ due

to the *border effect*. Therefore, $0 \leq m \leq l - w + 1$.

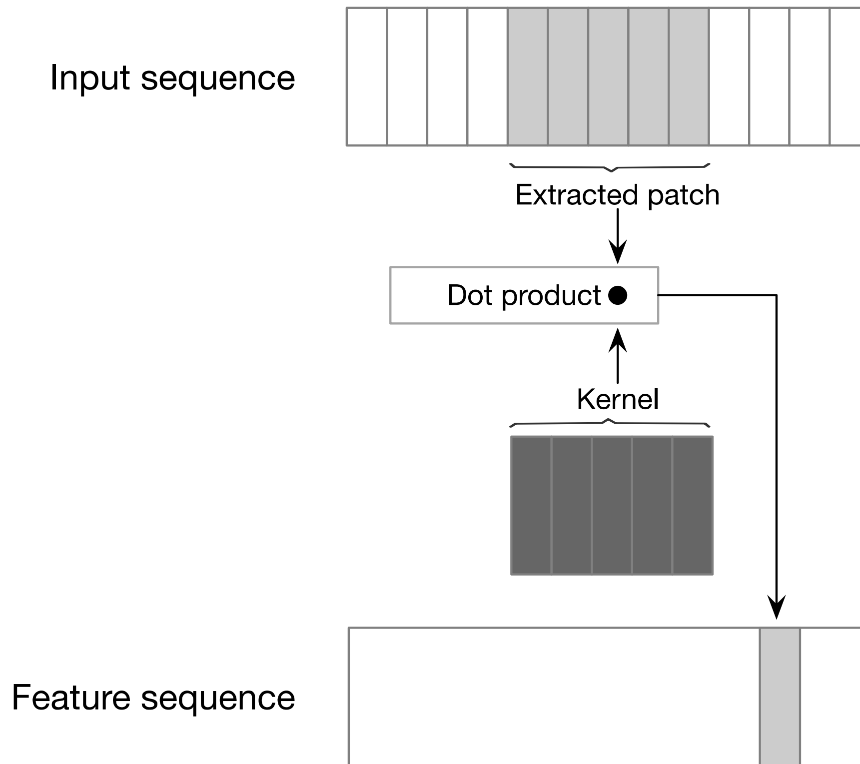

Figure 2.10: Performing One-Dimensional Convolution Operation on an Input Sequence Results in a Feature Sequence

2.2.9 The Max-Pooling Operation

The *max-pooling* in 1-D CNN downsamples the length of the feature sequence and reduces its size. The max-pooling has a window of a fixed length, which slides over the feature sequence and performs the maximum operation. If the length of the window in the max-pooling operation is 2, it halves the feature sequence by extracting the maximum value within every two elements. Figure 2.11 illustrates the max-pooling operation with a window of length 2.

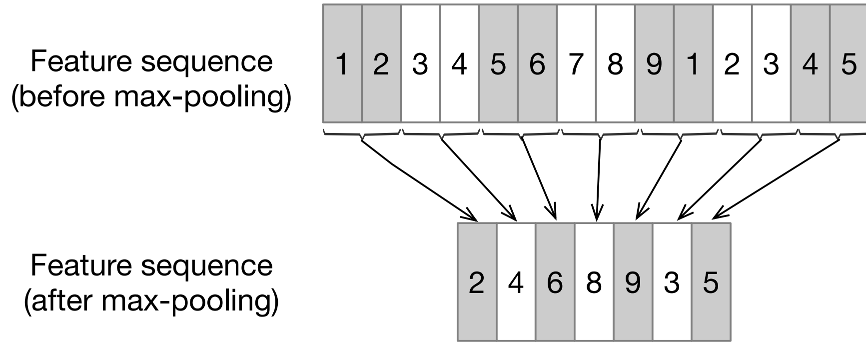

Figure 2.11: Performing Max Pooling Operation Downsamples the Feature Sequence

2.3 Related Work

There are various research efforts related to travel time prediction. Autoregressive integrated moving average (ARIMA) is a traditional statistical time series method, that is used for traffic forecasting (Williams, 2001; Williams and Hoel, 2003; Billings and Yang, 2006; Chandra and Al-Deek, 2009). One of the constraints of ARIMA-based methods are to approximate the proper model parameters before starting the modeling process and fitting time series data into the model.

Machine Learning (ML) based data-driven approaches have become more popular due to the promising results as well as abundance of traffic data and computing power (Duan et al., 2016). *Data-driven* methods utilize historical travel time and/or other historical factors to predict travel time (Duan et al., 2016). Using historical travel time average in linear models is one of the earliest methods to predict travel time (Wall and Dailey, 1999; Zhang and Rice, 2003; Rice and Van Zwet, 2004; Sun et al., 2007). These models are simple in terms of the data and computational power requirement, which make them suitable for areas with static traffic patterns (Petersen et al., 2019). However, they can not reflect

the immediate effect of unanticipated external events such traffic incidents for accurate traffic condition predictions (Petersen et al., 2019).

Linear Kalman filtering (KF) is also applied to travel time prediction. The KF method used historical data for travel time forecasting and calibrated the prediction using the real-time data (Chien and Kuchipudi, 2003). Linear KF is employed either independently (Shalaby and Farhan, 2004) or combined with other methods (Yu et al., 2010). Despite some success to tackle travel time prediction problems, linear KFs are not able to capture non-linear dynamics of travel time in highly urban areas (Petersen et al., 2019).

Another ML approach that has been applied to traffic prediction is support vector regression (SVR) (Smola and Schölkopf, 2004). SVR, which is a special support vector machine (SVM), maps historical travel time data to a higher dimensional feature space and predicts travel time using the mapped data (Wu et al., 2004). Previous work shows that SVM-based models have better performance comparing to some regression and time-series models (Ma et al., 2015).

In addition, ANNs are widely explored for traffic forecasting. As an example, comparing a simple multi-layer FFNN with KF proposed that ANNs outperform KFs (Kumar et al., 2014). CNNs are ANN-based models that have been widely used to solve variety complex tasks including image processing. Recently, some variation of CNNs have been applied to the travel time prediction problems (Hou and Edara, 2018; Ran et al., 2019; Wang et al., 2018). More complex ANNs such as RNNs are also used for travel time prediction. An example is state space neural network (SSNN) which is a variant of RNNs (van Lint et al., 2005). Another class of RNNs are LSTMs, which are capable of capturing information from longer travel time sequences. LSTMs have been used in travel time (Duan et al., 2016) and speed prediction (Ma et al., 2015). A recent RNN-based model is a hybrid

architecture, which is called ConvLSTM (Petersen et al., 2019). This model combines LSTM and CNN architectures resulting a mean absolute percentage error (MAPE) of 4.75% for 3-step ahead prediction.

Chapter 3

Data

This chapter explains the data used in the modeling process. It describes how the datasets are structured as well as how the data are prepared and transformed for training purposes.

3.1 Datasets

This thesis uses two sets of data. The first is the travel time dataset and the second is the TMC dataset. Both datasets are described in the following subsections.

3.1.1 Travel Time Dataset

The travel time dataset contains travel time and speed reports from the TMCs in the Oklahoma road network system for the span of two years from January 1st, 2018 to December 31st, 2019. There are a total of 3881 TMCs reporting for 2018 and an additional 297 TMCs reporting for 2019. Each TMC should have a record of speed and travel time in five-minute increments; however, due

to technical issues, there are missing reports for some of the TMCs. There are 4 functional classes of roads in this dataset, namely Classes 1, 2, 3, and 4. We performed the modeling on functional class 1 TMCs, which represent interstate road systems. We focused only on interstate TMCs because of their significant role in conveying traffic. Also, they have considerably fewer missing reports compared to other classes in the dataset. This narrows down the number of TMCs to 1449 for 2018 and an additional 51 for 2019.

3.1.2 TMC Dataset

This dataset contains the details about TMCs. TMCs are identified by a unique code and the date range during which they are active, i.e., from 2018 to 2019. They also have attributes such as road name, county, latitude, longitude, length, and urban code. We use some attributes like TMC length in feature engineering before performing the modeling.

3.2 Data Preparation

To prepare the data for travel time prediction modeling, we convert each TMC to a travel time sequence. We frame travel time prediction as a univariate multi-step time-series problem. First, we define a *lag*, a minimum amount of travel time to predict some steps of travel time in the future. We will slide a window over each TMC travel time sequence and generate the lag-step pairs to fit in the models. The current travel time data reports for every five minutes for each TMC. This five-minute increment causes two issues in modeling. First, due to errors, some TMCs are unable to report travel time every five minutes. This results in missing values in the travel time dataset and consequently in the generated lag-step pairs.

Time series models generally fail to converge when there are missing values in the training data. Second, it takes a considerable amount of time to train the model with all TMC sequences with a five-minute increment. Because there are 2,949 interstate TMCs and the maximum length of the TMC sequence is 105,120 reports (365 days $\times$ 24 hours per day $\times$ 12 reports per hour), and we would need to slide the window over each sequence, it would be very time-consuming to train on all available data. Therefore, to address the issue, we downsample each TMC by transforming the reports to be for every hour. We achieve this by calculating the average of each 12 consecutive travel time reports, which represents an hour.

In some TMCs, there are occasions that all travel time reports within specific hours are missing. Therefore, even after downsampling, the reports associated with those hours are missing. To address the issue, we developed an algorithm to analyze and filter the TMCs with a severe level of missing values. This algorithm includes only the sequences with a maximum of 4 missing travel times within a span of 80 hours. In other words, if the TMC's sequence has more than 4 missing values in each 80-hour window, that portion of the sequence is considered a gap that cannot be imputed and should be removed from the sequence. If the sequence meets the condition, it remains intact; otherwise, it may be split to smaller subsequences. After running the algorithm on the interstate TMCs, we are left with a total of 4,108 sequences. These sequences come from 1,347 and 1,358 TMCs accounting for 93% and 90% for 2018 and 2019, respectively. There are 1,297 TMCs that are common in both 2018 and 2019. The length of travel time sequences ranges from 2 to 12 months.

3.3 Feature Engineering

The lengths of TMCs are in miles and are not equal. Travel time is correlated with the length of the TMC, resulting a highly variable travel time distribution. In order to mitigate the role of the TMC length, we created a new feature called *speed* by dividing the travel time by the length of each TMC. We performed the modeling using speed; however, when we want to calculate the error of prediction, we convert speed back to travel time by multiplying by length. We can see in Figure 3.1 and 3.2 the travel time distribution before and after the transformation. Speed transformation preserves the relation between the travel time observations within a sequence. Figure 3.3 and 3.4 depict the time plot for travel time for a random TMC. They show the travel time before and after transformation within the range of 500 hours. The signal is inversed; however, the relation between the values are intact.

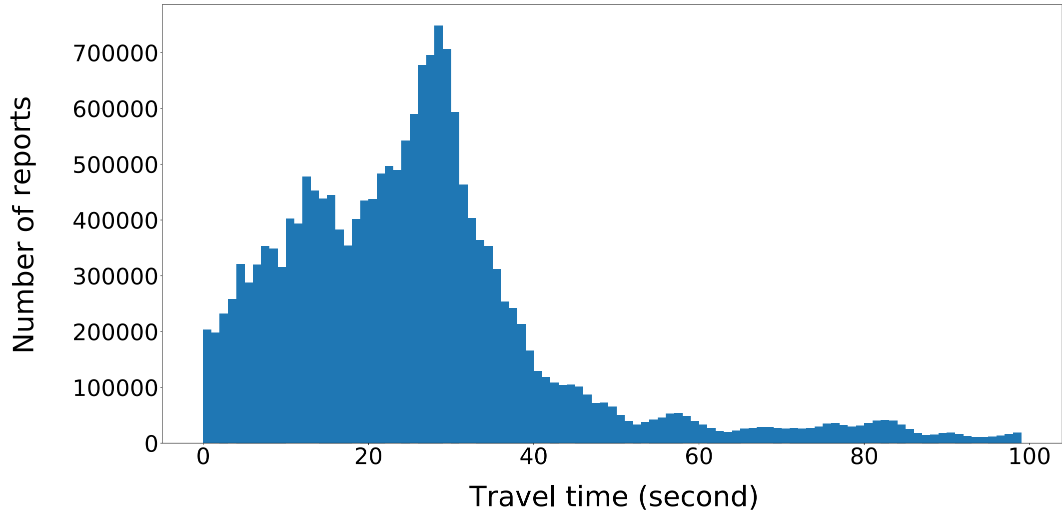

Figure 3.1: Travel Time Distribution (Before Transformation)

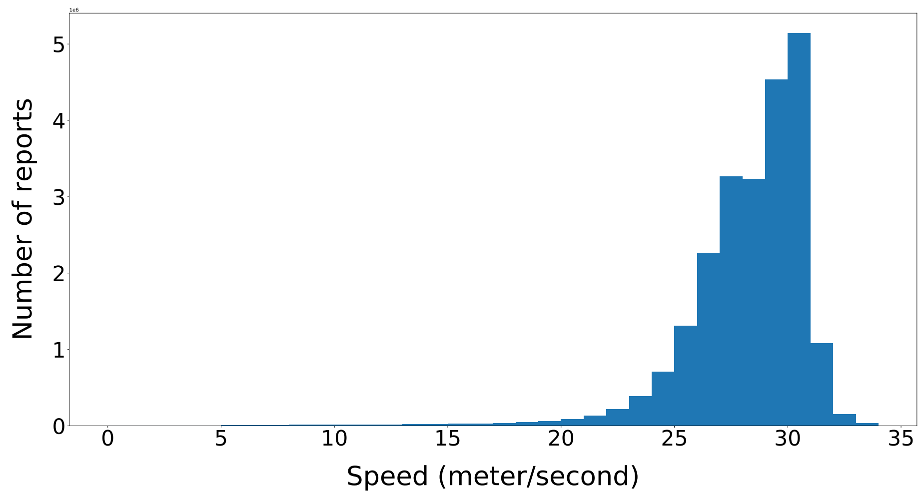

Figure 3.2: Speed Distribution (After Transformation)

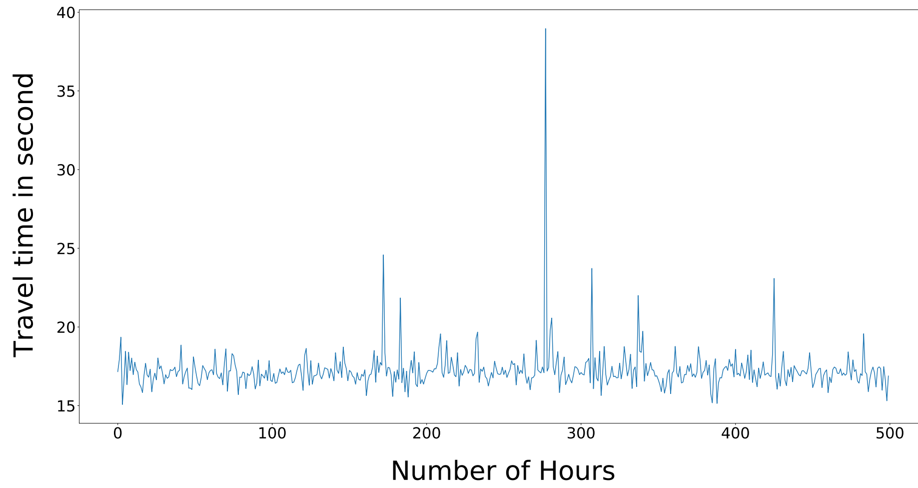

Figure 3.3: Travel Time Plot of a TMC (Before Transformation)

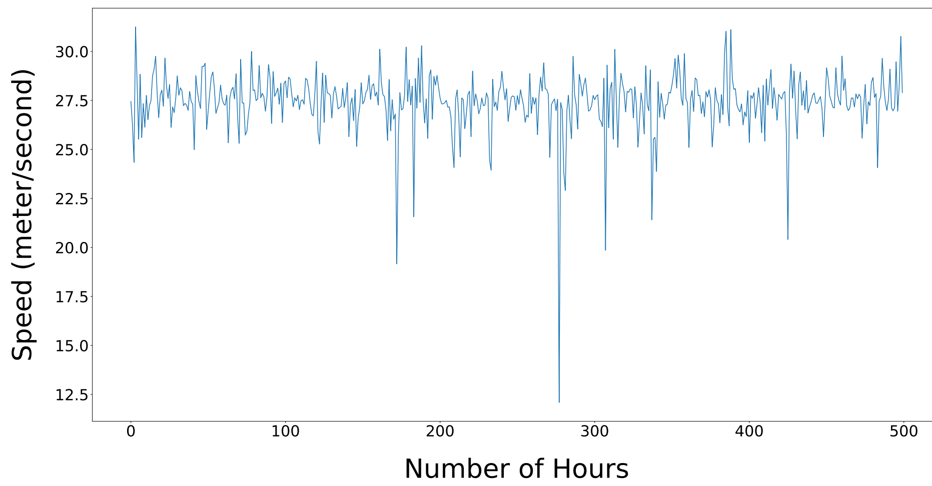

Figure 3.4: Speed Plot of a TMC (After Transformation)

Chapter 4

Methodology

We use the statistical SARIMA method as well as two machine learning approaches known as LSTM and 1-D CNN in our modeling process. The SARIMA model is our baseline and we compare machine learning results with it. The purpose of modeling is to capture the relation between travel time occurrences in a sequence and predict the travel time for the next 4 hours in the Oklahoma interstate road network system. Our goal is to discover appropriate architectures for the statistical and machine learning techniques. The proposed architecture for each approach should be able to predict the travel time with minimal error. The experimental environment is in Google Colab, which is a cloud service that provides a user-friendly interface to run the Python code. It also provides computing resources enabling us to run the deep learning models, offering a Tensor Processing Unit (TPU) and 35 GB of RAM.

In this chapter, first, we explain the SARIMA methodology. Then, we describe the LSTM and 1-D CNN approaches in detail.

4.1 SARIMA Approach

The most important step in SARIMA modeling is determining the model's parameters. SARIMA is an extension to the traditional ARIMA approach. ARIMA is comprised of three processes, autoregressive (AR), integrated (I), and moving average (MA). Each process has a parameter associated with it; they are p for AR, q for MA, and the difference parameter d for I. SARIMA includes 4 extra parameters, which are P, Q, D, and s. We have estimated the parameters d, D, and s, by analyzing the time plots and ACF plots. Figure 4.1 illustrates time plots for 4 random TMCs for 240 hours (10 days). The differencing parameter d is used when the time-series has a trend, i.e., it is increasing or decreasing. As we can see in the figure, the travel time sequences do not seem to have trends; therefore, d=0. Also, Figure 4.2 shows ACF plots for the same TMCs used in Figure 4.1. The number of lags in the ACF plots are 240. It can be observed that sequences have seasonality of 24 hours. This means that a similar travel time pattern occurs every 24 hours. Therefore, we have assigned 24 to s. SARIMA modeling converges if the seasonality of the sequence is removed. Therefore, we have assigned 1 to D. That means that each travel time in a sequence will be subtracted from a travel time point that is located 1 season (24 hours) ahead. Then, the transformed (differenced) data will be used for the modeling.

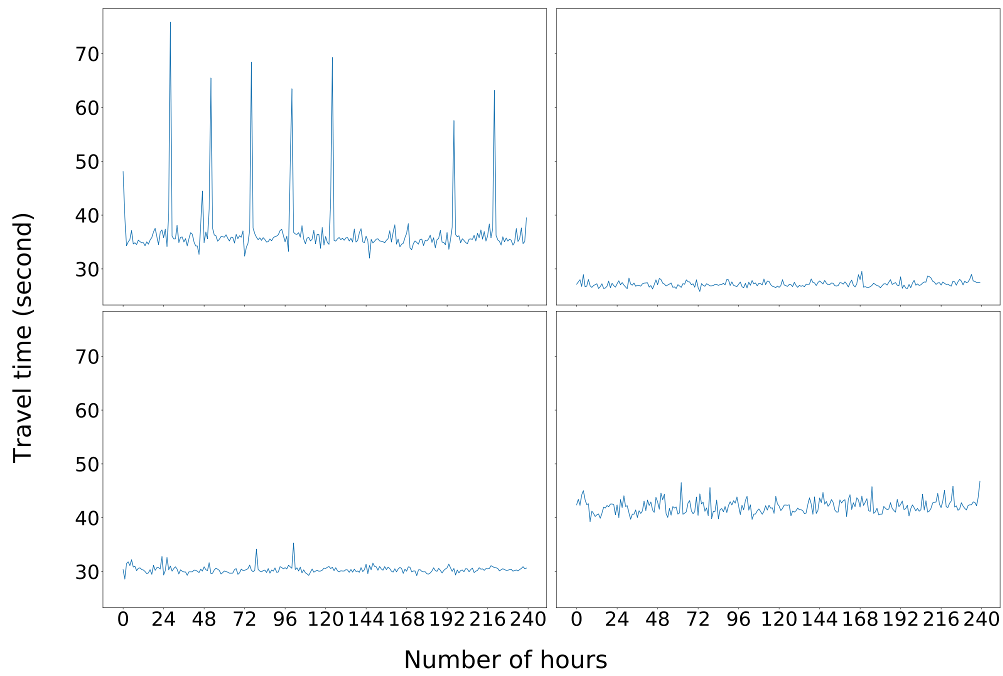

Figure 4.1: Display 24-hour Seasonality Using Time Plots

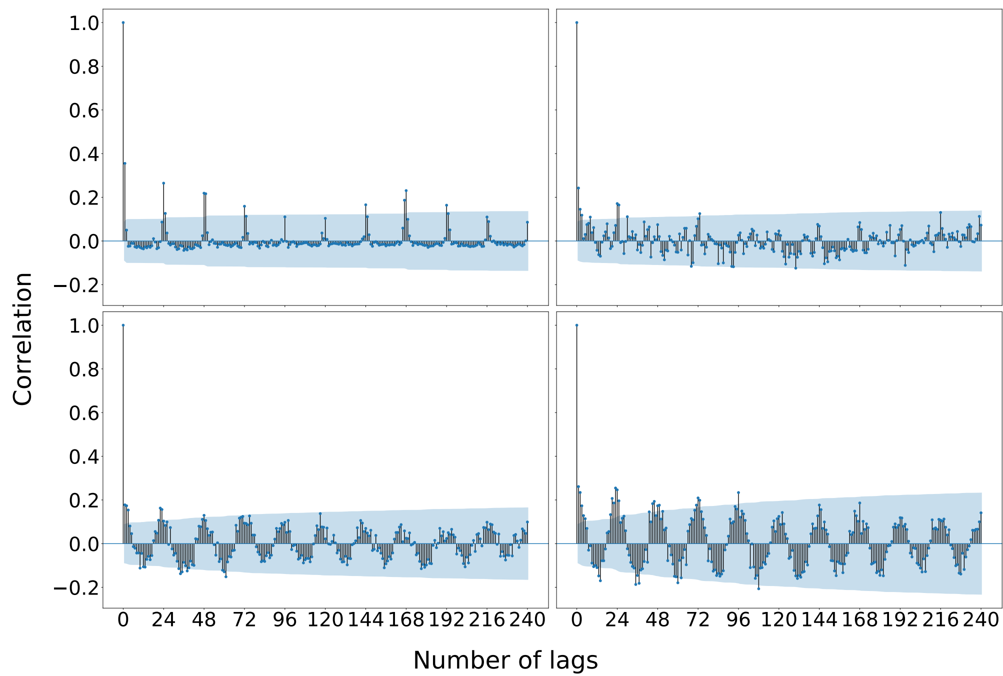

Figure 4.2: Display 24-hour Seasonality Using ACF Plots

We have estimated parameters of p, q, P, and Q using ACF and PACF plots

of the differenced data. Figures 4.3 and 4.4 shows the ACF and PACF plots for a sample TMC 111P05174 before and after differencing for 240 hours, respectively.

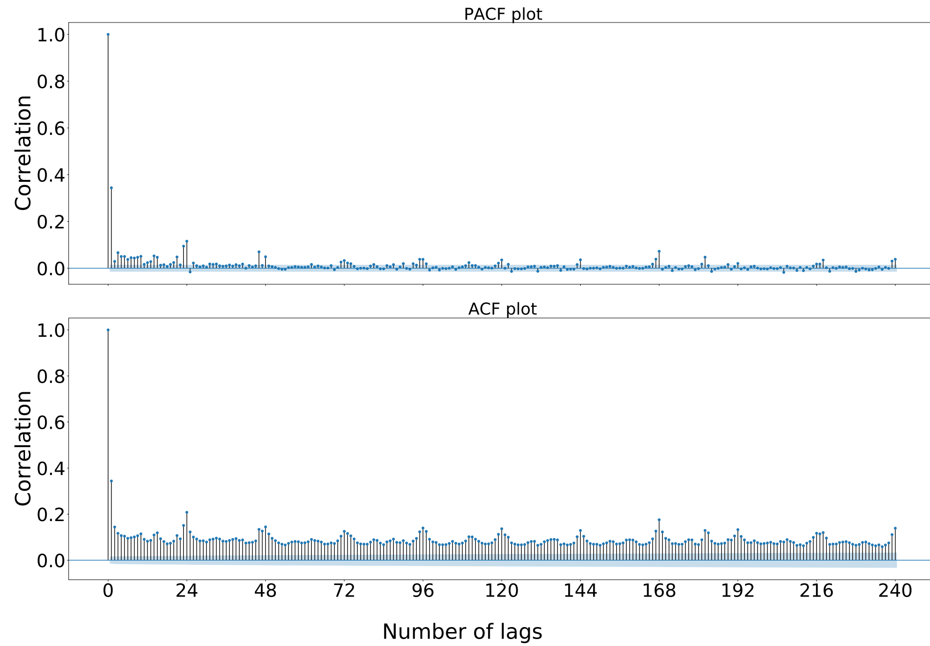

Figure 4.3: PACF and ACF Plots Before Differencing Operation

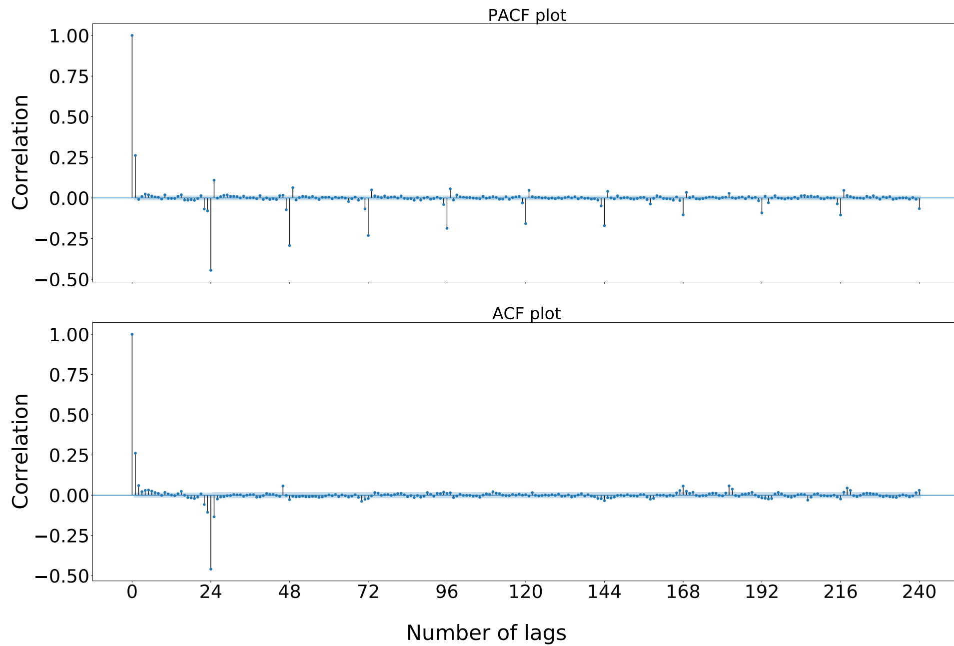

Figure 4.4: PACF and ACF Plots After Differencing Operation

The values for p and P can be determined by observing the PACF plot. The order p of the AR process can be zero, if the PACF value steadily approaches zero. If the partial autocorrelation is significant at lag k and abruptly drops after lag k, then we can try an autoregressive model of order $p = k$. In PACF plot in Figure 4.4, the value of partial autocorrelation suddenly declines after the first lag. Therefore, we can choose p=1. The seasonal term P is zero, if the partial autocorrelation for seasonal lags smoothly approaches zero. If this value is significant until season k and abruptly drops after season k, then we can try $P = k$. From the PACF plot in Figure 4.4, we can see that the partial autocorrelation value smoothly decreases in several seasons, where each season is 24 lags. Therefore, we choose P=0.

The order q of the MA process can be obtained by observing the ACF plot of the differenced data. The value of q is zero if the autocorrelation value smoothly approaches zero. Otherwise, q is the number of lags at which the value of ACF abruptly drops. From the ACF plot in Figure 4.4, we can choose $q = 2$. The seasonal term Q is zero if the autocorrelation for the seasonal lags smoothly approaches zero. If this value is significant until season k and abruptly drops after season k, then we can try $Q = k$. From the ACF plot in Figure 4.4, we can see that the autocorrelation value drops after the first season. Therefore, we choose Q=1.

Based on the above example we choose the SARIMA $(1, 0, 2) \times (0, 1, 1)_{24}$ model. However, since the seasonal part of the SARIMA is computationally expensive, we have set the value for Q to zero, which should prevent the model from canceling the seasonal differencing.

4.1.1 Urban Code

The TMCs in the travel time dataset belong to three categories in terms of the urban area code to which they belong. There are three urban codes in the data namely, rural, small urban, and large urban. We have divided the data based on the urban code and identified the SARIMA parameters for each category. There are 1,362 rural, 386 are small urban, and 2,396 large urban. After analysis of the ACF and PACF plots from samples of each of urban code, following SARIMA parameters are chosen:

- $(3, 0, 2) \times (0, 1, 0)_{24}$ for the rural area
- $(1, 0, 2) \times (0, 1, 0)_{24}$ for the small urban areas
- $(2, 0, 2) \times (0, 1, 0)_{24}$ for the large urban areas

4.2 Machine Learning Approaches

Before explaining LSTM and 1-D CNN, here, we summarize the common procedures in both approaches. First, they are trained and tested using Keras. Keras is an open-source neural-network library written in Python. It is designed to enable developers to conduct fast experimentation with deep neural networks. In addition, both LSTM and 1-D CNN follow the similar training procedures. The training dataset includes 4,108 transformed travel time sequences ranging from 1,500 hours (two months) to 8,760 hours (12 months). The process starts with splitting the data into the training, validation, and test sets. We use the training set in the training process to find the model weights. Then, we examine the accuracy of the trained model by using it to predict the travel time in the validation set. We compare the error in the prediction of the validation set with

that of the training set. We use the magnitude of the error to determine whether the model is overfitting. *Overfitting* is an error that the underlying function in the modeling process fits too closely to the data points in the training dataset. In other words, overfitting results in a overly complex models attempting to explain all characteristics of the dataset. Since the training dataset includes errors and noise and does not completely reflect the real world, an overly complex model based on training data can not be used to explain other datasets. In case of overfitting, it is necessary to tune the model hyperparameters. *Hyperparameters* are network parameters that influence the learning process. Unlike the model parameters, the hyperparameters are not learned in the training process, and should be adjusted in advance. The hyperparameters are specific to each type of modeling. Examples of the hyperparameters in 1-D CNN are number of hidden layers and kernel size. In LSTM, an example is the number of units. In the context of this document, units in LSTM are different from the ANN neurons and they correspond to dimensions of the weight matrix in an LSTM cell. We also examine different hyperparameters for each modeling approach to minimize the validation set error. The set of hyperparameters that gives us the least validation error is used for the final stage, i.e., the test stage. We use test data, the dataset which is not seen by the models, to examine our model's performance. This is the error that we will report and use to compare the models with each other. Finally, we have defined a Python generator that generates input-output pairs from a given travel time sequence for both approaches. The generator takes two parameters called lag and step, which are equivalent to input and output, respectively. The *lag* is the number travel times used for prediction by the model. The *step* is the number travel times the model predicts in future. As shown in Figure 4.5, the generator slides over the sequence by an increment of one and generates lag-step

pairs. The generator also has a batch size parameter. The *batch size* controls how many lag-step pairs are returned every time the generator is requested to generate training data. The lag-step pairs generation continues until it exhausts all possible pairs for the given sequence then it switches to the next sequence.

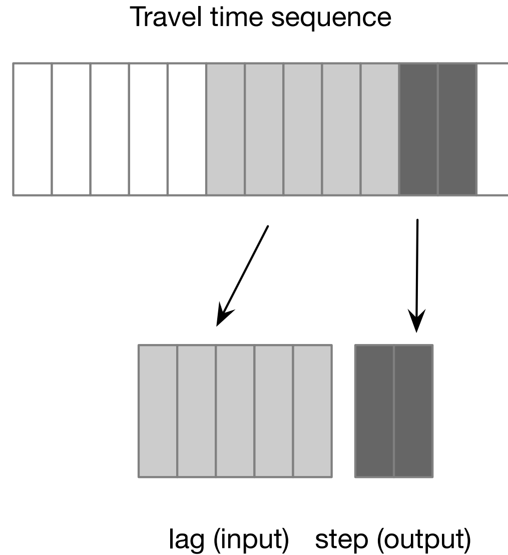

Figure 4.5: Data Generator Uses a Time Sequence for Creation of Lag-Step Pair

4.3 LSTM Approach

LSTM is a machine learning modeling approach that is appropriate for processing sequences. We examine LSTMs to develop a model capable of predicting up to four hours of travel time. We train the model twice. In the first training setup, we explore different hyperparameters and determine an appropriate architecture. Then, we use the selected architecture in the final training process and report the result.

4.3.1 LSTM Model Architecture

We initially followed the LSTM architecture for travel time prediction proposed by Duan et al. (2016). This LSTM model consists of 1 input layer, 1 hidden layer, and 1 output layer. Duan et al. (2016) develops a specific LSTM model for each of the 66 travel time reporting locations in Highways in England. The number of units in their architecture is considered a hyperparameter and it ranges from 1 to 5. They have experimented with values from 1 to 5, inclusive, to set unit numbers for each model and chosen the architecture that yields the minimum validation error. In our case, the total number of sequences is 4108. Due to the extensive number of sequences, and limited time and resources, it is not feasible for us to experiment with all possible unit numbers for each of the 4108 sequences. Instead, we have proposed an architecture similar to that of Duan et al. (2016) but with a higher capacity, which can potentially predict the travel time for the majority of TMCs in the interstate road network of Oklahoma. Our search space for the unit number is the multiples of 24, ranging from 24 to 120. We have observed 24-hour patterns in the travel time plots; therefore, we assumed that 24-based unit numbers are appropriate. We have started with a network including one LSTM layer, one hidden dense layer, and an output layer. The number of neurons in the hidden dense layer and output layer are fixed, and they are set to 24 and 1, respectively. This means that the proposed network predicts one step in the future because the number of neurons in the output layer is 1. The process of determining an appropriate architecture includes iterating through the stated range of possible unit numbers, training the model, calculating the error on the validation set, and choosing the unit number that produces the minimum error. We have found that the unit number 24 gives the least validation error.

Table 4.1: Proposed LSTM Architecture

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 24)	2496
dense_1 (Dense)	(None, 24)	600
dense_2 (Dense)	(None, 1)	25
Total params: 3,121		
Trainable params: 3,121		
Non-trainable params: 0		

Therefore, we have chosen a 24-unit LSTM architecture for the next round of training. Table 4.1 shows the proposed LSTM architecture.

4.3.2 LSTM Input and Output Dimension

The LSTM requires the input sequence to be 3-dimensional. The first dimension is the batch size. The second and third dimensions are individual input dimensions. The batch size is 36 and lag size is 24; therefore, the data generator generates two-dimensional sequences with the shape of 36×24. To comply with the LSTM input dimension requirement, the input is reshaped to 36×24×1. The output shape of the LSTM network is one travel time at a time. That means for each batch of 24 hours travel time, the network predicts one travel time. Because the size of the batches are 36, the output shape of the network is 36×1.

4.3.3 Number of Trainable Parameters in LSTM

The number of units in the LSTM layer determines the dimension of the weight tensors in an LSTM cell. As described in the background section, there are four gates in an LSTM layer. Each gate has two sets of weight tensors and a bias term. At a given timestep, W is the weight tensor corresponding to the input sequence, U is the weight tensor for the output from the previous timestep, and

b is the bias term. The number of parameters for a single LSTM operation is calculated as follows:

$$W\ parameters = number\ of\ units \times input\ sequence\ third\ dimension,$$
$$U\ parameters = number\ of\ units \times number\ of\ units,$$
$$b\ parameters = number\ of\ units,$$
$$LSTM\ parameters = 4 \times (W\ parameters + U\ parameters + b\ parameters).$$

Based on the above equations, the number of parameters in the LSTM layer is

$$4 \times (24 \times 1 + 24 \times 24 + 24) = 2496.$$

The second layer in the architecture is a hidden dense layer with 24 neurons. The number of weights in a dense layer is calculated using

$$\begin{aligned} Weights\ in\ a\ dense\ layer = {} & number\ of\ neurons, \\ & \times output\ shape\ of\ the\ previous\ layer, \\ & + number\ of\ neurons. \end{aligned}$$

Since the shape of the output from the LSTM layer is 24, therefore the number of parameters in the dense layer is

$$24 \times 24 + 24 = 600.$$

The output layer is also a dense layer which has only one neuron, therefore the number parameters in the output layer is

$$1 \times 24 + 1 = 25.$$

Therefore, the total trainable parameters in the proposed architecture is $2,496 + 600 + 25 = 3,121$. In this document we also compare different architectures. Therefore, we use the proposed LSTM model and modify the number of neurons in the last layer. We experiment with two, three, and four neurons in the final layer, which results in 3,146, 3,171, and 3,196 parameters, respectively.

4.3.4 Training the LSTM Model

After choosing an appropriate architecture, we train the LSTM model on the dataset by randomly dividing it to the training, validation, and test sets and assigning 2,100, 1,000, and 1,000 sequences to them, respectively. The selected model only predicts one travel time at a time, therefore, we have experimented with and modified the architecture four times by setting the number of neurons in the output layer from one to four. In other words, in each trial, the model is trained four times separately using the same training data to predict one value at a time, two values at a time, three values at a time, and four values at a time. The number of epochs in the training and validation process is set to 100 epochs. Moreover, we have used MSE as our performance metric both for training and validation. We have utilized the step-per-epoch feature in Keras. In each step of an epoch, Keras requests the generator to generate and return an input-output pair. Due to time and resource constraints, we defined 2,500 steps for training and 1,200 steps for validation. Consequently, Keras could only request the training generator 100×2,500=250,000 times which resulted exhausting 1,600 sequences out of 2,100 training sequences. Also, Keras requested the validation generator 100×1,200=120,000 times which resulted in exhausting 800 sequences out of 1,000 validation sequences. We follow the same procedure for three times

to help account for inherent randomness in the initial weight values and sampling order.

4.4 1-D CNN Approach

1-D CNN is a machine learning modeling approach that is suitable for processing sequences. We examine the 1-D CNN approach to develop a model capable of predicting up to four hours of travel time. We have trained the model twice. In the first training setup, we have explored different hyperparameters and determined an appropriate architecture. Then, we have used the selected architecture in the final training process and we have reported the result.

4.4.1 1-D CNN Model Architecture

We have designed a 1-D CNN architecture to be similar to the LSTM architecture for comparison purposes. Therefore, we have started the initial design of the network with one 1-dimensional convolution layer equipped with max-pooling and flattening operations. The flattening operation reshapes a 2-D array to a 1-D array. The convolution layer is followed by a hidden dense layer, and concluded by the output layer. We have identified the number of filters (kernels) in the convolution layer and the number of neurons in the hidden layer as hyperparameters. Our search space for the hyperparameters include multiples of 8, ranging from 8 to 32. The length of a kernel in the model is fixed and is set to 4. The process of finding an appropriate architecture includes iterating through the set of candidate hyperparameters, training the model, calculating the error on the validation set, and choosing the hyperparameter pairs that yield the minimum error. We have found that the combination of 24 filters and 24 neurons, gives the

Table 4.2: Proposed 1-D CNN Architecture

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 21, 24)	120
max_pooling1d_1 (MaxPooling1D)	(None, 10, 24)	0
flatten_1 (Flatten)	(None, 240)	0
dense_1 (Dense)	(None, 24)	5784
dense_2 (Dense)	(None, 1)	25
Total params : 5,929		
Trainable params : 5,929		
Non-trainable params : 0		

least validation error. Table 4.2 shows the proposed 1-D architecture

4.4.2 1-D CNN Input and Output Dimensions

The 1-D CNN architecture requires the input sequence to be 3-dimensional. The first dimension is the batch size. The second and third dimensions are individual input dimensions. The batch size is 36 and lag size is 24; therefore, the data generator generates 2-dimensional sequences with the shape of 36×24. To comply with the 1-D CNN input dimension requirement, the input is reshaped to $36 \times 24 \times 1$. The output shape of the 1-D CNN network is 1 travel time at a time. That means for each batch of 24 hours travel time, the network predicts 1 travel time. Since the size of the batches are 36, the output shape of the network is 36×1.

4.4.3 Number of Trainable Parameters in 1-D CNN

The number of and the size of the kernel in 1-D convolution layer determines the number of weights. The number of parameters in a single 1-D convolution layer is

$$number\ of\ kernels\ \times kernel\ size\ + number\ of\ kernels$$

Based on the above equations, the number of parameters in a 1-D convolution layer is

$$24 \times 4 + 24 = 120.$$

The output shape of a 1-D convolution layer is

$$output\ feature\ sequence\ length \times number\ of\ filters.$$

Therefore, the output shape is 21×24. The max-pooling operation halves the feature sequence, which results in a 10×24 output feature sequence. The flattening operation converts the 2-D feature sequence to a 1-D vector. Therefore, the shape the output before the hidden layer is 10×24 = 240. The next layer in the architecture is a hidden dense layer with 24 neurons. The number of parameters in a dense layer is calculated using

$$\begin{aligned} Weights\ in\ a\ dense\ layer &= number\ of\ neurons, \\ &\times output\ shape\ of\ the\ previous\ layer, \\ &+ number\ of\ neurons. \end{aligned}$$

Therefore, the number of parameters for the hidden layer is

$$24 \times 240 + 24 = 5784.$$

The output layer is also a dense layer which has only one neuron, therefore the number parameters in the output layer is calculated using

$$1 \times 24 + 1 = 25.$$

Therefore, the total trainable parameters in the proposed architecture is 120 + 5,784+25 = 5,929. In this document we also compare different architectures. Therefore, we use the proposed 1-D CNN model and modify the number of neurons in the last layer. We experiment with two, three, and four neurons in the final layer, which results in 5,954, 5,979, and 6,004 parameters, respectively.

4.4.4 Training the 1-D CNN Model

After choosing an appropriate architecture, we train the 1-D CNN model on the dataset by randomly dividing it to training, validation, and test sets and assigning 2,100, 1,000, and 1,000 sequences to them, respectively. The selected model only predicts one travel time at a time; therefore, we have experimented with and modified the architecture four times by setting the number of neurons in the output layer from one to four. In other words, the model is trained four times separately using the same training data to predict one value at a time, two values at a time, three values at a time, and four values at a time. The number of epochs in the training and validation process is set to 100 epochs. Moreover, we have used MSE as our performance metric both for training and validation. We have used the step-per-epoch feature in Keras. In each step of an epoch, Keras requests the generator to generate and return an input-output pair. Due to time and resource constraints, we defined 2,500 steps for training and 1,200 steps for validation. Consequently, Keras could only request the training generator 100×2,500=250,000 times which resulted exhausting 1,600 sequences out of 2,100 training sequences. Also, Keras requested the validation generator 100×1,200=120,000 times which resulted in exhausting 800 sequences out of 1,000 validation sequences. We follow the same procedure three times to help account

for inherent randomness in the initial weight values and sampling order.

Chapter 5

Results and Discussion

In this chapter, first we present the evaluation results for each of our proposed models: SARIMA, LSTM and 1-D CNN. Then, we compare the performance of all proposed models and discuss the results.

5.1 Results for SARIMA

To test SARIMA, we define and test different parameters for each urban code. These code are rural, small urban, and large urban area codes. The travel time sequences are randomly sampled based on their urban code and fit in to their corresponding SARIMA model. The fitting process includes feeding an input travel time sequence into the SARIMA model and predicting a limited number of steps in future. The number of steps defines the number of travel times a model predicts simultaneously given an input sequence. We also have tested each model's predictive capabilities under 4 different architectures, which are 1-step, 2-step, 3-step, and 4-step ahead predictions. The SARIMA model predicts the travel time using the predefined parameters, i.e., number of autoregressive (AR) or moving

average (MA) lags. The length of the process is highly dependent on parameters because model parameters such as MA lags determine the number of previous travel times the model should use in the calculation process. Therefore, using large lags often causes delays in model convergence. We have followed a similar procedure to machine learning approaches in terms of generating the data for the fitting process. The model partitions each input sequences using a sliding window. The model fits the travel time portion and predicts the required. The sliding continues by the increment of 1 until it reaches to the end of the sequence. The input sequence length ranges from 2 to 12 months. Unfortunately, it is not feasible to exhaust all possible travel time portions from a given sequence. Therefore, we have limited the maximum amount of sliding to 720 hours (1 month) for each sequence. The starting point of the sliding process is randomly chosen. In other words, the model picks a random start position for each input sequence and generates 720 portions for the fitting process. The rest of this section presents the results for the rural, small urban, and large urban area SARIMA models.

5.1.1 Rural Area SARIMA Model

We have randomly selected 36 out of 1,362 total rural area sequences. Fig 5.1 depicts the MRE and RMSRE errors for rural areas. The model generally does not perform well since the number of errors greater than 1 in both error types is considerable. Also, the number of large errors increases when the model predicts multiple steps simultaneously. For example, there are more than 18 RMSRE values greater than 1 at the 4-step architecture which accounts for half of the sequences.

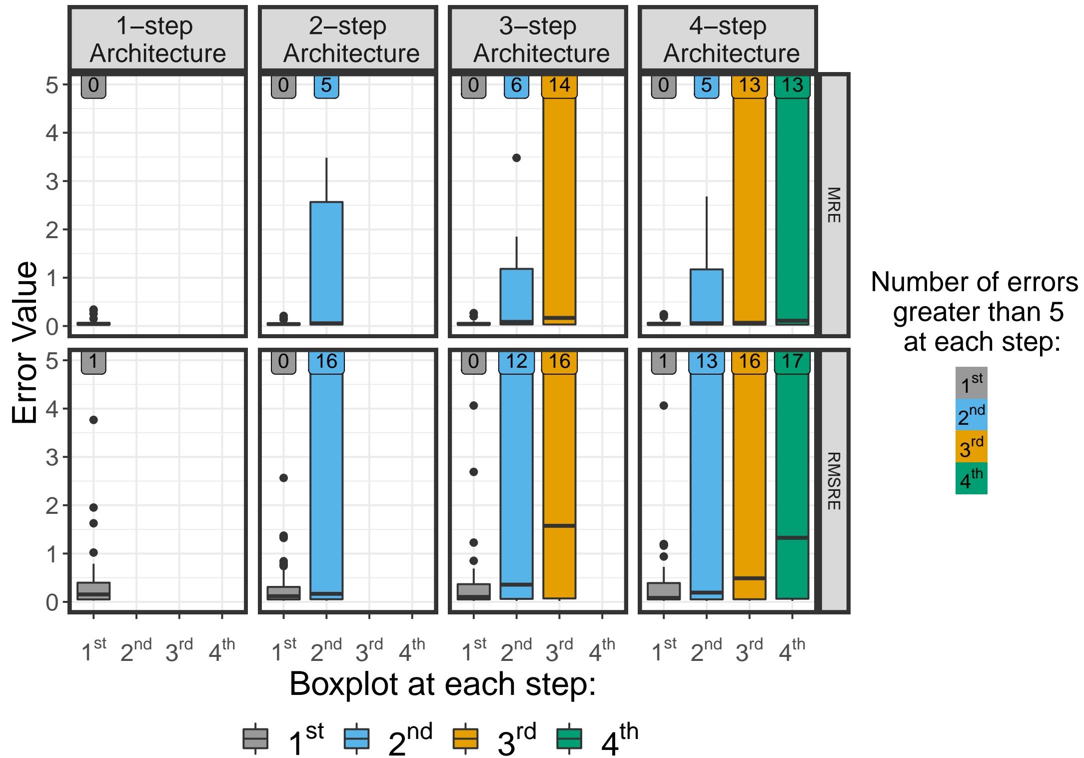

Figure 5.1: SARIMA MRE and RMSRE Results for Rural Areas

5.1.2 Small Urban SARIMA Model

We have randomly selected 38 out of 386 small urban sequences. Fig 5.2 illustrates the MRE and RMSRE errors for 4 different architectures. As we can see the size of both error types increases as the architecture size increases. This also applies to the lower steps in multi-step architectures. For example, prediction of one step ahead in 4-step architecture is more erroneous compared to its 1-step architecture. We can also observe relative errors more than 1, which is an indicator of larger error predictions. Some errors exceeds 5, so we chose to color code them to keep the graph vertical axis within a specific range.

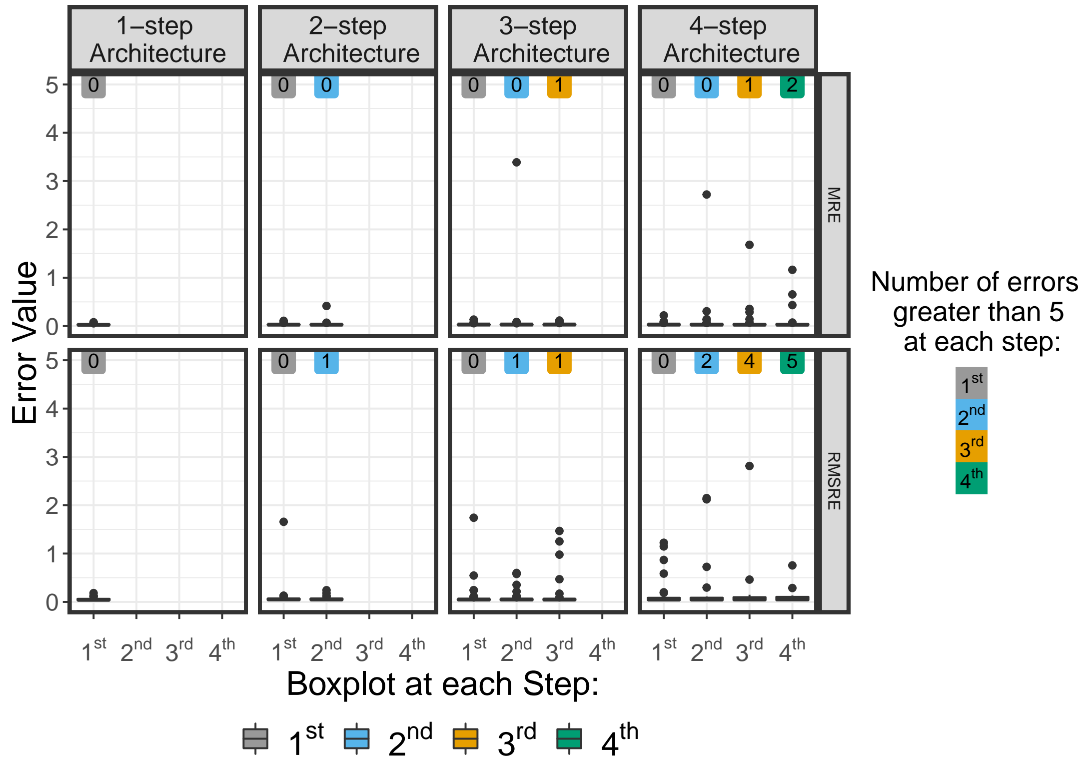

Figure 5.2: SARIMA MRE and RMSRE Results for Small Urban Areas

5.1.3 Large Urban SARIMA Model

There are a total of 2,396 sequences belong to the large urban areas. We have used 46 of them for the modeling purpose in SARIMA. Fig 5.3 shows the break down of MRE and RMSRE errors in modeling large urban sequences. The number of large errors (e.g., greater than 1) is trivial in single-step architecture. However, we can observe that performance degrades in all steps of the larger architectures.

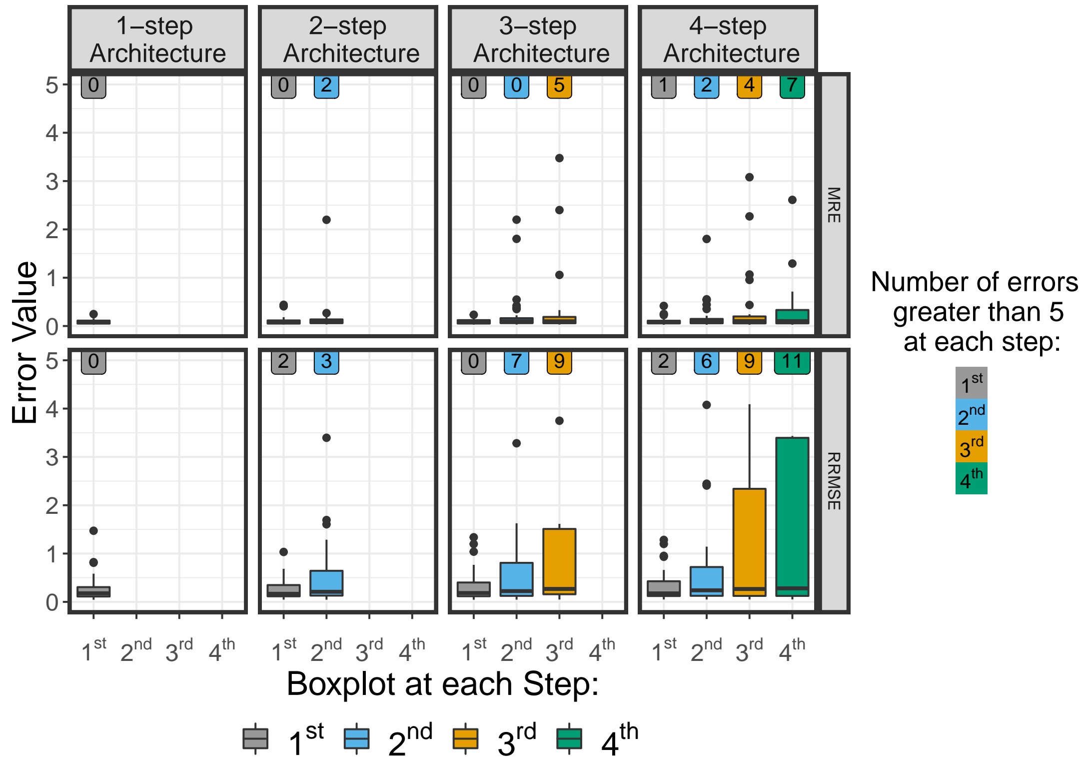

Figure 5.3: SARIMA MRE and RMSRE Results for Large Urban Areas

5.1.4 Discussion for the SARIMA Models

Among the three SARIMA models, the small Urban SARIMA has the best performance. One explanation can be the way we have chosen the model parameters. The models' parameters are defined in advance. Therefore, it is possible that we have not chosen appropriate parameters for other models. Also, the performance is negatively correlated with the number of prediction steps. As the number of steps increases, the errors in prediction increase.

5.2 Results for LSTM

To train the LSTM model, we have allocated 1000 sequences for test purposes; however, we could calculate the error for only 800 sequences due to time and resource constraints. We have used two metrics, RMSE and RMSRE, for evaluation. As mentioned in Chapter 3, the dataset is transformed for feature engineering purposes. Therefore, before calculating the prediction error, we have converted the model prediction and the ground truth back to travel time (seconds). To obtain more reliable results, we have repeated the procedure three times. Figure 5.4 presents the results of the LSTM networks evaluated by the relative metrics, MRE and RMSRE. The top half shows the boxplots for MRE scores and the bottom half depicts the boxplots for RMSRE scores. As discussed in 4.3.4, we have experimented with 4 different LSTM architectures, in which the output layer is modified to predict 1, 2, 3, and 4 travel times separately. There is an interesting pattern in LSTM errors, where the errors increase when the number of steps in architecture is odd and decrease when the number of steps in architecture is even. This experimental setup helps us to understand how the change in the output layer influences the predictive capability of the network. Figure 5.4 also shows that the median of the MREs for each step of each architecture is around 0.05. This applies to the median of RMSRE as well, where the median of score is below 0.09 at each step. There is a pattern in error fluctuation depending on architecture type. We can see the that the upper bound and median of both error types in the odd-step architectures increases and in the even-step architectures decreases. This means that modifying the number of neurons in the output layer affects the prediction accuracy. Finally, we can notice that there is a difference between the MRE and RMSRE at each step of each architecture. The

median and maximum of the RMSRE errors are substantially larger compared to median and maximum of MRE at each step. This can suggest the prediction of the larger errors by the LSTM model.

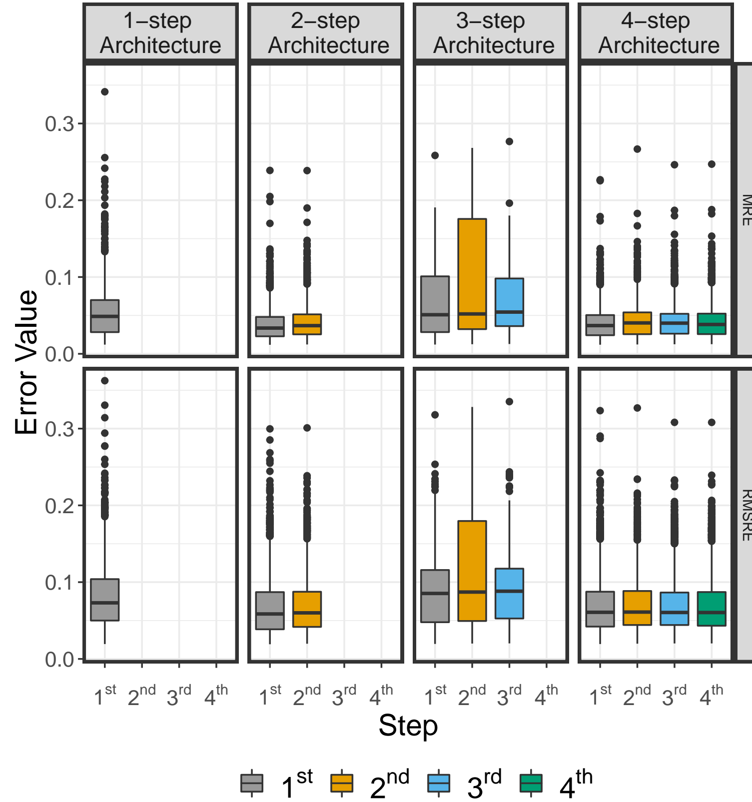

Figure 5.4: The Proposed LSTM Model's MRE and RMSRE Results

5.3 Results for 1-D CNN

To train the 1-D CNN model, we have allocated 1,000 sequences for test purposes; however, we could calculate the error only for 800 sequences due to time and resource constraints. We have used two different metrics, MRE and RMSRE, for evaluation. As mentioned in Chapter 3, the dataset is transformed for feature engineering purposes. Therefore, before calculating the prediction error, we have

converted the model prediction and the ground truth back to travel time (seconds). The model's architecture is modified to provide 1-step, 2-step, 3-step, and 4-step prediction using the same test dataset. We have repeated the procedure three times to have a more reliable results. Figure 5.5 shows the MRE and RMSRE scores for the 1-D CNN architecture. Both results are relative which is an appropriate evaluator for the overall performance of the network. As mentioned in Section 5.2, we report the result for four different architectures. We can see that both errors are almost constant across architectures and steps. The median of MRE and RMSRE are around 0.04 and 0.06 respectively. In contrast to LSTM, the results indicate that the addition of neurons to the output layer does not substantially affect the model's prediction stability. Finally, we can notice that there is a difference between the MRE and RMSRE at each step of each architecture. The median and maximum of the RMSRE errors are substantially larger compared to median and maximum of MRE at each step. This can suggest the prediction of the larger errors by the 1-D CNN model.

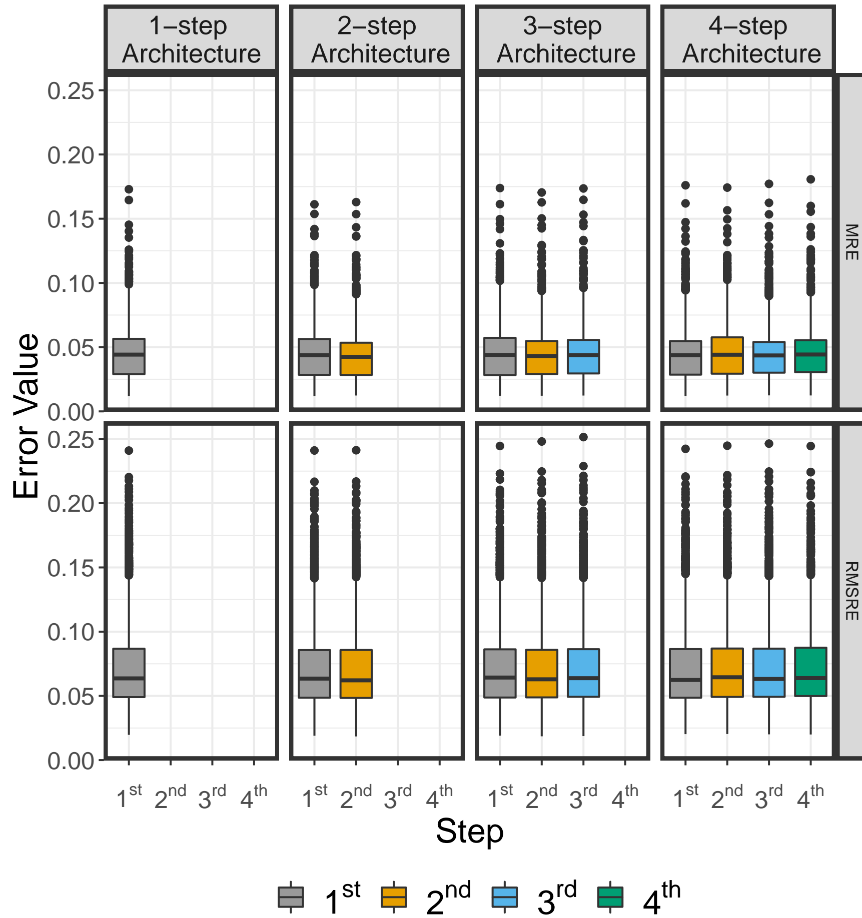

Figure 5.5: The Proposed 1-D CNN Model's MRE and RMSRE Results

5.4 Comparison of the Proposed Models

Figure 5.6 shows the comparison of three modeling approaches: 1-D CNN, LSTM, and SARIMA. The errors of steps of each architecture are combined and sorted and a single boxplot is generated per metric. Therefore, for each method and architecture, there are two metric boxplots, MRE, and RMSRE. The 1-D CNN and LSTM methods have similar performance and they both outperform the SARIMA model. The SARIMA model suffers from highly skewed errors, especially in higher step architectures, which go beyond relative errors of 1.0. On the

other hand, 1-D CNN and LSTM errors never pass 0.25 and 0.37, respectively. Also, contrary to the SARIMA model, the performance of the machine learning models is not affected by increasing the architecture size. Tables 5.1, 5.2, 5.3, and 5.4 show the median, mean, and standard deviation of MRE and RMSRE errors of all architectures. The largest statistical summary for each error metric is colored with red. We can see from the tables that SARIMA has the largest mean and standard deviation across all architectures. Especially, the magnitude of mean and standard deviation of errors in SARIMA in higher step architectures are considerable. Also, in 6 out of 8 median error reports, SARIMA has the maximum error value. Therefore, the machine learning approaches outperform SARIMA.

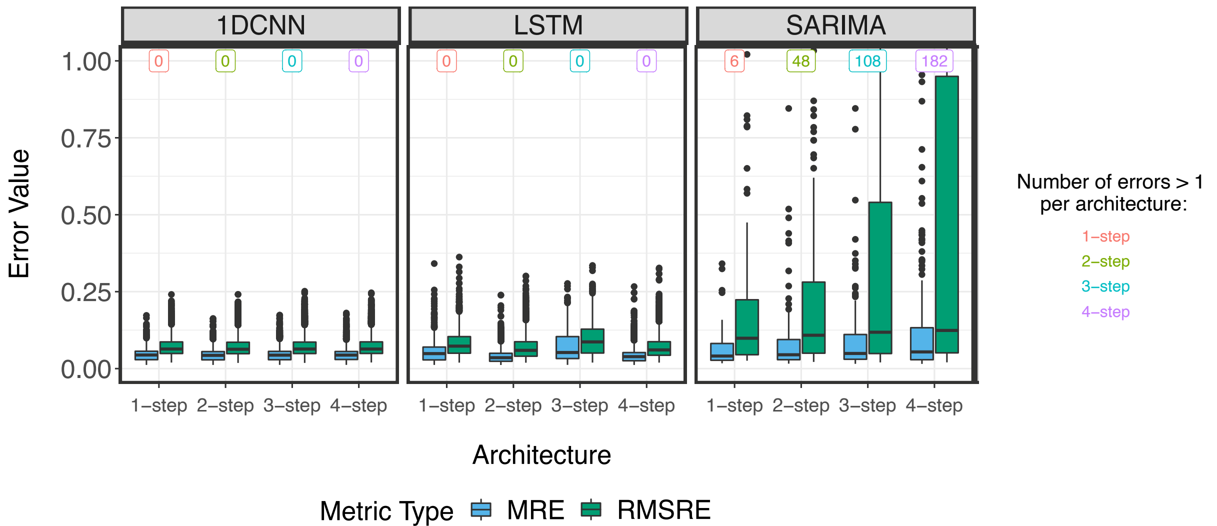

Figure 5.6: Comparison of 1-D CNN, LSTM, and SARIMA Results

Table 5.1: Summary of MRE and RMSRE Errors for 1-Step Architecture

Architecture	Metric Type	Method	Median	Mean	Standard Deviation
1-step	MRE	SARIMA	4.08e-2	**6.3e-2**	**5.62e-2**
		LSTM	**4.87e-2**	5.33e-2	3.11e-2
		1-D CNN	4.41e-2	4.55e-2	2.02e-2
	RMSRE	SARIMA	**9.86e-2**	**2.91e-1**	**7.63e-1**
		LSTM	7.31e-2	8.21e-2	4.38e-2
		1-D CNN	6.36e-2	7.25e-2	3.52e-2

Table 5.2: Summary of MRE and RMSRE Errors for 2-Step Architecture

Architecture	Metric Type	Method	Median	Mean	Standard Deviation
2-step	MRE	SARIMA	**4.5e-2**	**1.07e0**	**7.81e0**
		LSTM	3.54e-2	4e-2	2.19e-2
		1-D CNN	4.31e-2	4.44e-2	1.97e-2
	RMSRE	SARIMA	**1.08e-1**	**2.59e1**	**2.01e2**
		LSTM	5.93e-2	6.99e-2	3.92e-2
		1-D CNN	6.29e-2	7.16e-2	3.47e-2

Table 5.3: Summary of MRE and RMSRE Errors for 3-Step Architecture

Architecture	Metric Type	Method	Median	Mean	Standard Deviation
3-step	MRE	SARIMA	4.92e-2	**8.67e2**	**1.31e4**
		LSTM	**5.23e-2**	7.34e-2	5.18e-2
		1-D CNN	4.37e-2	4.52e-2	2.03e-2
	RMSRE	SARIMA	**1.18e-1**	**2.31e4**	**3.49e5**
		LSTM	8.68e-2	9.61e-2	5.32e-2
		1-D CNN	6.35e-2	7.23e-2	3.51e-2

Table 5.4: Summary of MRE and RMSRE Errors for 4-Step Architecture

Architecture	Metric Type	Method	Median	Mean	Standard Deviation
4-step	MRE	SARIMA	**5.42e-2**	**7.6E+36**	**1.67E+38**
		LSTM	3.89e-2	4.25e-2	2.21e-2
		1-D CNN	4.4e-2	4.54e-2	2.01e-2
	RMSRE	SARIMA	**1.24e-1**	**2.04E+38**	**4.48E+39**
		LSTM	6.07e-2	7.11e-2	3.82e-2
		1-D CNN	6.36e-2	7.26e-2	3.5e-2

We compare the MRE and RMSRE errors in 1-D CNN and LSTM, by performing statistical analysis on the mean and variance. We employe two-sided t-test to compare the mean of 1-D CNN and LSTM. The null hypothesis is that the mean of errors of 1-D CNN is equal to that of LSTM. We also use two-sided F-test to compare the variance of 1-D CNN and LSTM. The null hypothesis for the variance is that the ratio of the variance of errors in 1-D CNN to the variance of errors in LSTM equals 1. The significance level for rejecting and failing to

reject the null hypothesises is 0.05 which is a commonly used value in statistical testings. Tables 5.5, 5.6, 5.7, and 5.8 show the result of statistical analysis of mean and variance of the errors of 1-D CNN and LSTM. The immediate take away from the p-values of all the performed tests is that in each case thethe p-value is smaller than 0.05(significance level). Thus all null hypotheses are rejected and indeed the compared means and variances are different. Based on the results of the statistical analysis, the minimum value for each error metric type is colored with dark blue. All results are rounded off to two significant digits

We can see from the results that 1-D CNN has the minimum variance across all architectures. In addition, 1-D CNN has the minimum mean in 1-step and 3-step architectures, while LSTM has the minimum mean in 2-step and 4-step architectures. There is an interesting pattern in LSTM errors, where the mean of errors increases when the number of steps in architecture is odd and it decreases when the number of steps in architecture is even. The exact reason for this fluctuation is unknown. However, it can be due to the recurrent mechanism in LSTM networks. The major difference between LSTM and 1-D CNN is that LSTMs constantly retain relevant information and/or forget irrelevant information in a travel time sequence. This information maintenance process combined with the architecture design choice are the subjects of our future work for better understanding of the behavior.

Table 5.5: Statistical Analysis of Mean and Variance of the Errors for the 1-Step Architecture for 1-D CNN & LSTM

Architecture	Metric Type	Method	Mean	Standard Deviation	Mean Hypothesis Test		Variance Hypothesis Test	
					Degrees of Freedom	*p*-value	Degrees of Freedom	*p*-value
1-step	MRE	1-D CNN	**4.55e-2**	**2.02e-2**	3259	≤0.001	numerator = 1929 denominator = 1899	≤0.001
		LSTM	5.33e-2	3.11e-2				
	RMSRE	1-D CNN	**7.25e-2**	**3.52e-2**	3634	≤0.001	numerator = 1929 denominator = 1899	≤0.001
		LSTM	8.21e-2	4.38e-2				

Table 5.6: Statistical Analysis of Mean and Variance of the Errors for the 2-Step Architecture for 1-D CNN & LSTM

Architecture	Metric Type	Method	Mean	Standard Deviation	Mean Hypothesis Test		Variance Hypothesis Test	
					Degrees of Freedom	*p*-value	Degrees of Freedom	*p*-value
2-step	MRE	1-D CNN	4.44e-2	**1.97e-2**	7553	≤0.001	numerator= 3859 denominator= 3799	≤0.001
		LSTM	**4e-2**	2.19e-2				
	RMSRE	1-D CNN	7.16e-2	**3.47e-2**	7518	0.0291	numerator= 3859 denominator= 3799	≤0.001
		LSTM	**6.99e-2**	3.92e-2				

Table 5.7: Statistical Analysis of Mean and Variance of the Errors for the 3-Step Architecture for 1-D CNN & LSTM

Architecture	Metric Type	Method	Mean	Standard Deviation	Mean Hypothesis Test		Variance Hypothesis Test	
					Degrees of Freedom	*p*-value	Degrees of Freedom	*p*-value
3-step	MRE	1-D CNN	**4.52e-2**	**2.03e-2**	7384	≤0.001	numerator= 5789 denominator= 5699	≤0.001
		LSTM	7.34e-2	5.18e-2				
	RMSRE	1-D CNN	**7.23e-2**	**3.51e-2**	9854	≤0.001	numerator= 5789 denominator= 5699	≤0.001
		LSTM	9.61e-2	5.32e-2				

Table 5.8: Statistical Analysis of Mean and Variance of the Errors for the 4-Step Architecture for 1-D CNN & LSTM

Architecture	Metric Type	Method	Mean	Standard Deviation	Mean Hypothesis Test		Variance Hypothesis Test	
					Degrees of Freedom	p-value	Degrees of Freedom	p-value
4-step	MRE	1-D CNN	4.54e-2	**2.01e-2**	15141	≤0.001	numerator= 7719 denominator= 7599	≤0.001
		LSTM	**4.25e-2**	2.21e-2				
	RMSRE	1-D CNN	7.26e-2	**3.5e-2**	15166	0.01	numerator= 7719 denominator= 7599	≤0.001
		LSTM	**7.11e-2**	3.82e-2				

We can review the disparity of errors for each specific model by comparing the mean and standard deviation of MRE and RMSRE. For example, in both LSTM and 1-D CNN, the mean and standard deviation of RMSRE are considerably larger than those of MRE at each architecture. For example, in a 4-step LSTM architecture, the mean and standard devation of RMSRE are 0.0711 and 0.0382, respectively. They are nearly twice as large as the mean and standard deviation of MRE, which are 0.0424, 0.0220 respectively. This suggests that both 1-D CNN and LSTM produce larger errors, which are noticeable in the difference between the RMSRE and MRE.

Chapter 6

Conclusions and Future work

In this work we formulate travel time prediction as a time series problem. We use the travel time from the travel time and speed reports from the TMCs in the Oklahoma highway system for the span of two years. This comprises measurements from a total of 1,449 TMCs reporting for 2018 and an additional 51 TMCs reporting for 2019. These datasets provide an excellent opportunity to measure the accuracy of deep learning models in a real world context rather than using synthetic datasets. With real world datasets, it is expected to observe issues such as missing values. TMCs have travel time reports for each 5 minutes and in some of them, due to technical issues, missing values are present. We handle this issue by downsampling the dataset. Instead of including each travel time entry, we calculate the average of them in a window of 60 minutes. In order to apply this methodology to other states we need to make sure that those datasets are also using constructs similar to TMCs as used in this dataset.

In this thesis, we experiment with three approaches including two machine learning methods and one traditional statistical approach known as SARIMA. Also, we observe each methodology in four different architectures to see how

predicting multiple steps in the future influences the forecasting power of the models. To compare the models among themselves we use relative error performance metrics RMSRE and MRE. We show that even our simple LSTM and 1-D CNN models outperform the traditional statistical approach SARIMA. The results also show that LSTMs and 1-D CNNs are more stable in multi-step predictions compared to their SARIMA counterpart.

In addition, 1-D CNNs have similar performance to LSTMs. Both 1-D CNNs and LSTMs have the minimum mean of errors in two architectures. However, by analyzing the standard deviation of the errors, we find that errors in 1-D CNNs are less variable compared to the LSTM approach. In other words, our proposed 1-D CNN learned and captured the relation between travel times reports in the Oklahoma interstate travel time sequence almost the same as the proposed LSTM model but with less variability. We also find that 1-D CNN is more stable in terms of prediction errors across the experimented architectures compared to LSTM. We note that modifying architecture to predict different steps in future resulted in fluctuation in error pattern in LSTM; however, 1-D CNN remained quite stable. Generally, LSTM-based models are considered to perform better in time series problems (Gers et al., 2002). One explanation for these results can be attributed the architecture of the models. In this work we develop the LSTM architecture similar to Duan et al. (2016) and strive to have models with the same size, in terms of the number of neurons, to ensure the fairness in our evaluations.

One implication of this is that before choosing a network type for a given problem we should be extra cautious.

This work inherits the shortcoming of finding an appropriate architecture. Our model architectures are rather simple, yet we still need to investigate whether a more complex model can further improve the results.

Explainability is the second shortcoming of this work. We still need to investigate "how" our deep learning models outperform the traditional statistical approaches. Using other metric types such as absolute errors also could help the explainability of the results. Absolute errors have same units as the problem domain, which is seconds for travel time prediction. Relative or absolute errors of a model can have different meaning when used for evaluating TMCs of different lengths. For example, if a model produces moderate absolute errors, it might be acceptable for longer TMCs; however, it might result in large relative errors for shorter TMCs, which would suggest that the model is not able to handle shorter TMCs. On the other hand, if a model produces small relative errors, it might give large absolute errors for longer TMCs, which might be concerning for travelers who care only for total number of seconds/minutes late for their final destination. Therefore, using combination of both absolute and relative error metrics and analysis of the results are suggested.

In this research, different statistical SARIMA models were developed to address the different urban area codes. On the other hand, the machine learning models were trained based on an assumption that the statistics for the TMCs are the same. It is possible to extend this work by, first, analyzing and identifying the TMCs distribution; second, constructing machine learning models for the TMCs based on similarity in distributions.

It is possible to extend this work by experimenting with and evaluating the accuracy of different LSTM-based architectures including stacked, bi-directional, and encoder-decoder LSTM architectures. Also, hybrid models such as ConvLSTM and CNN LSTM can be used for travel time prediction.

In this work, we used heuristics to find the model's hyperparameters. In our future work, we can employ more systematic algorithms to determine the model's

hyperparameters such as genetic algorithms.

While the focus of this thesis is to predict travel times at individual TMCs, we can extend this problem to predict the travel times between any two points. In this case, we need to predict the travel for paths that comprise multiple TMCs. The extended version of this problem has excellent use cases in the personal and commercial navigation applications. In order to tackle such problems we might need alternative model architectures as we have to deal with the spatial aspects as well. We leave this problem for our future work.

Bibliography

Ratnadip Adhikari and Ramesh K Agrawal. An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*, 2013.

Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE, 2017.

Daniel Billings and Jiann-Shiou Yang. Application of the ARIMA models to urban roadway travel time prediction - a case study. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2529–2534, Taipei, Taiwan, October 2006. doi: 10.1109/ICSMC.2006.385244.

Naim Bitar. Big data analytics in transportation networks using the npmrds. Master's thesis, University of Oklahoma, Norman, OK, 2016.

Srinivasa Ravi Chandra and Haitham Al-Deek. Predictions of freeway traffic speeds and volumes using vector autoregressive models. *Journal of Intelligent Transportation Systems*, 13(2):53–72, 2009.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.

Steven I-Jy Chien and Chandra Mouly Kuchipudi. Dynamic travel time prediction with real-time and historic data. *Journal of transportation engineering*, 129(6):608–616, 2003.

Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017. ISBN 1617294438.

Yanjie Duan, LV Yisheng, and Fei-Yue Wang. Travel time prediction with LSTM neural network. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1053–1058. IEEE, 2016.

A.P. Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2nd edition, 2007.

Federal Highway Administration (FHWA)- Department of Transportation. National travel time data processing and utilization, 2020. URL https://www.fhwa.dot.gov/policyinformation/presentations/hisconf/mon04$_{n}ational_{t}rai$ $2020 - July - 01$.

Lianli Gao, Zhao Guo, Hanwang Zhang, Xing Xu, and Heng Tao Shen. Video captioning with attention-based LSTM and semantic consistency. *IEEE Transactions on Multimedia*, 19(9):2045--2055, 2017.

Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying LSTM to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193--200. Springer, 2002.

I. Goodfellow, Y. Bengio, and A. Courvill. *Deep learning*. MIT press, 2016.

D. Graupe. *Principles of artificial neural networks*. World Scientific, 2013.

Simon S Haykin et al. Neural networks and learning machines/simon haykin., 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770--778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. LSTM can solve hard long time lag problems. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, pages 473--479, Cambridge, MA, USA, 1996. MIT Press.

Yi Hou and Praveen Edara. Network scale travel time prediction using deep learning. *Transportation Research Record*, 2672(45): 115--123, 2018.

Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. Real-time patient-specific ecg classification by 1-d convolutional neural networks. *IEEE Transactions on Biomedical Engineering*, 63(3): 664--675, 2015.

Serkan Kiranyaz, Turker Ince, and Moncef Gabbouj. Personalized monitoring and advance warning system for cardiac arrhythmias. *Scientific Reports*, 7(1):1--8, 2017.

Vivek Kumar, B Anil Kumar, Lelitha Vanajakshi, and Shankar C Subramanian. Comparison of model based and machine learning approaches for bus arrival time prediction. In *Proceedings of the 93rd Annual Meeting*, pages 14--2518. Transportation Research Board, 2014.

Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 85--100, 2018.

Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187--197, 2015.

T.M. Mitchell. *Machine learning*. McGraw-Hill, New York, 1997.

Jiwon Myung, Dong-Kyu Kim, Seung-Young Kho, and Chang-Ho Park. Travel time prediction using k nearest neighbor method with combined data from vehicle detector system and automatic toll collection system. *Transportation Research Record*, 2256(1): 51--59, 2011.

Niklas Christoffer Petersen, Filipe Rodrigues, and Francisco Camara Pereira. Multi-output bus travel time prediction with convolutional LSTM neural network. *Expert Systems with Applications*, 120:426--435, 2019.

Xiangdong Ran, Zhiguang Shan, Yong Shi, and Chuang Lin. Short-term travel time prediction: a spatiotemporal deep learning approach. *International Journal of Information Technology & Decision Making*, 18(04):1087--1111, 2019.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779--788, 2016.

John Rice and Erik Van Zwet. A simple and effective method for predicting travel times on freeways. *IEEE Transactions on Intelligent Transportation Systems*, 5(3):200--207, 2004.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533--536, 1986.

Alaa Sagheer and Mostafa Kotb. Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing*, 323:203--213, 2019.

Amer Shalaby and Ali Farhan. Prediction model of bus arrival and departure times using avl and apc data. *Journal of Public Transportation*, 7(1):3, 2004.

Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199--222, 2004.

Dihua Sun, Hong Luo, Liping Fu, Weining Liu, Xiaoyong Liao, and Min Zhao. Predicting bus arrival time on the basis of global positioning system data. *Transportation Research Record*, 2034 (1):62--72, 2007.

J.W.C. van Lint, S.P. Hoogendoorn, and H.J. van Zuylen. Accurate freeway travel time prediction with state-space neural networks under missing data. *Transportation Research Part C: Emerging Technologies*, 13(5-6):347--369, October 2005. doi: 10.1016/j.trc.2005.03.001. URL https://doi.org/10.1016/j.trc.2005.03.001.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156--3164, 2015.

Z Wall and DJ Dailey. An algorithm for predicting the arrival time of mass transit vehicles using automatic vehicle location data. In *78th annual meeting of the transportation research board*, pages 1--11. Citeseer, 1999.

Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. When will you arrive? estimating travel time based on deep neural networks. In *AAAI*, volume 18, pages 1--8, 2018.

Billy M Williams. Multivariate vehicular traffic flow prediction: evaluation of arimax modeling. *Transportation Research Record*, 1776(1):194--200, 2001.

Billy M Williams and Lester A Hoel. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. *Journal of transportation engineering*, 129(6):664--672, 2003.

Chun-Hsin Wu, Jan-Ming Ho, and Der-Tsai Lee. Travel-time prediction with support vector regression. *IEEE Transactions on Intelligent Transportation Systems*, 5(4):276--281, 2004.

Bin Yu, Zhong-Zhen Yang, Kang Chen, and Bo Yu. Hybrid model for prediction of bus arrival times at next station. *Journal of Advanced Transportation*, 44(3):193--204, 2010.

Xiaoyan Zhang and John A Rice. Short-term travel time prediction. *Transportation Research Part C: Emerging Technologies*, 11(3-4): 187--210, 2003.

Yanru Zhang and Ali Haghani. A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies*, 58:308--324, 2015.