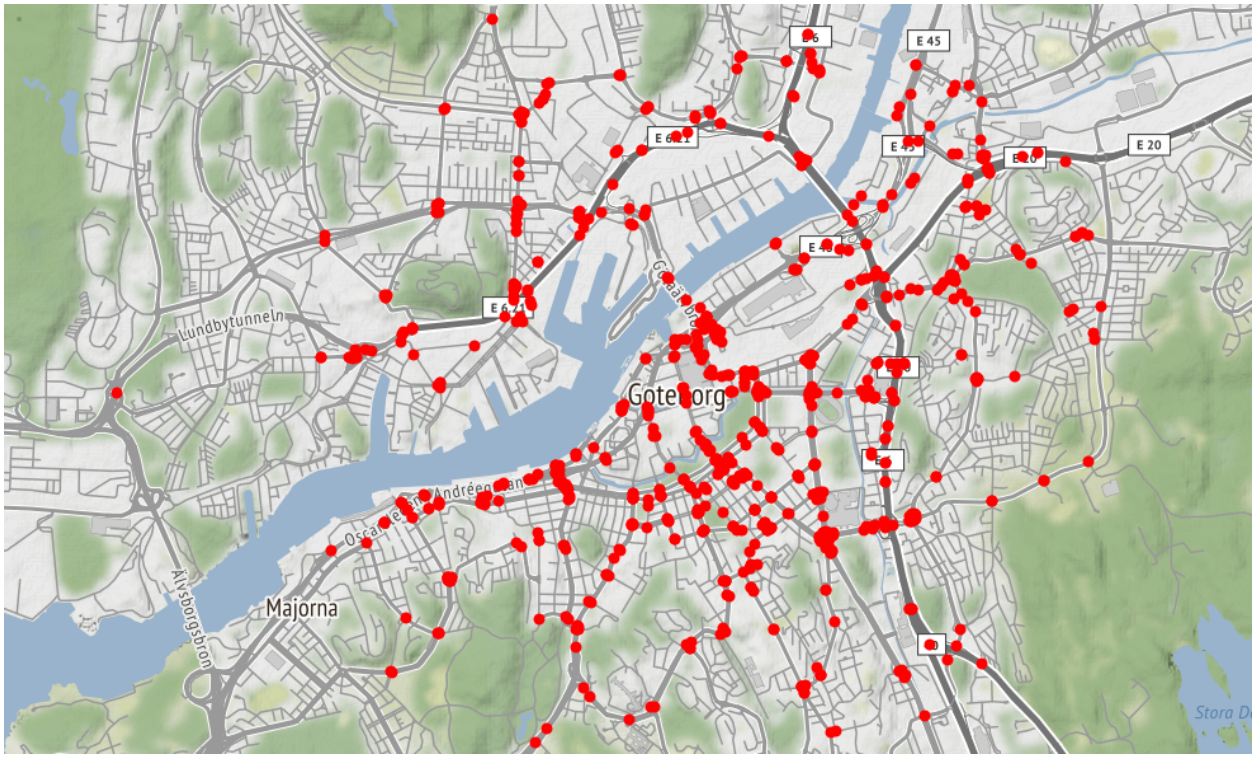


Machine learning applied to traffic forecasting

Degree project report in Computer Science and Engineering

Linus Aronsson & Aron Bengtsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

DEGREE PROJECT REPORT

Machine learning applied to traffic forecasting

Linus Aronsson

Aron Bengtsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Machine learning applied to traffic forecasting
Linus Aronsson, Aron Bengtsson

Supervisor: Ulf Norell, Department of Computer Science and Engineering
Advisor: Lef Eleftherios Filippakis, Cybercom Sweden AB
Examiner: Per Lundin, Department of Computer Science and Engineering

Department of Computer Science and Engineering
Chalmers University of Technology / University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 (0)31-772 1000

Cover: Map of road segments in Gothenburg

Department of Computer Science and Engineering
Gothenburg, Sweden 2019

Machine learning applied to traffic forecasting

Linus Aronsson
Aron Bengtsson
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Forecasting future traffic situations is of great importance for improving existing *Intelligent Transportation Systems* (ITS). The forecasts allow the various ITS technologies to function proactively. This provides safer roads with less accidents and lower congestion levels. Forecasting urban road traffic is challenging as it often follows complex nonlinear temporal patterns. Traditional statistical forecasting techniques therefore struggle to achieve accurate predictions. In recent years the computing power and available historic traffic data has increased drastically. These are two of the main ingredients required for the field of *Machine Learning* (ML) to work. Many ML techniques have been shown to be capable of capturing nonlinear patterns in data, which makes it a good candidate for traffic forecasting. This thesis therefore explores various ML technologies and applies them to time series forecasting. Additionally, some traditional approaches to time series forecasting are evaluated as baselines for comparison. The experiments conducted used traffic data from central Gothenburg, which was manually gathered throughout the project. The conclusion was that the best ML techniques provided a higher forecasting accuracy for both short-term and long-term predictions. More advanced hyperparameter optimization and feature engineering would further improve the ML models.

Keywords: machine learning, artificial neural networks, LSTM, forecasting, traffic flow, time series.

Acknowledgements

We would like to express our deepest gratitude to everyone who has helped us during this project. We would like to thank everyone at Cybercom for establishing this project and for letting us work at their office. We would like to thank our advisor Lef Filippakis at Cybercom for his guidance and assistance. His expertise in data science and machine learning was very valuable to us throughout the project. Finally, we would like to thank our supervisor Ulf Norell at Chalmers for his guidance and very helpful feedback on our thesis.

Linus Aronsson, Aron Bengtsson, Gothenburg, June 2019

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Background and Motivation . . . 1
1.2 Objectives and Scope . . . 2
1.3 Delimitations . . . 3
1.4 Syntax Convention . . . 3

2 Theory **4**
2.1 Traffic Flow . . . 4
2.1.1 Temporal Correlations . . . 4
2.1.2 Spatial Correlations . . . 4
2.2 Forecasting . . . 5
2.2.1 Time Series . . . 6
2.2.2 Time Series Forecasting . . . 6
2.2.3 Time Series Split . . . 7
2.2.4 Evaluating a Forecasting Model . . . 8
2.2.5 Multi-step Forecasting . . . 10
2.3 Machine Learning . . . 11
2.3.1 Introduction . . . 12
2.3.2 Types of Learning Algorithms . . . 12
2.3.3 Overfitting and Underfitting . . . 14
2.3.4 Feature Engineering . . . 14
2.3.5 Data Preparation . . . 15
2.3.6 Hyperparameters . . . 17
2.4 Forecasting Models . . . 17
2.4.1 Baseline Methods . . . 18
2.4.2 ARIMA . . . 18
2.4.3 Feedforward Neural Network . . . 19
2.4.4 Recurrent Neural Network . . . 22
2.4.5 Convolutional Neural Network . . . 23
2.4.6 Support Vector Regression . . . 24
2.5 Database Management . . . 24
2.5.1 NoSQL . . . 24

2.5.2 MongoDB . . . 24
2.6 Previous Work . . . 25

3 Method **27**
3.1 Software . . . 27
3.2 Data Collection . . . 29
3.2.1 Traffic Data . . . 29
3.2.2 Weather Data . . . 31
3.3 Data Format . . . 32
3.4 Data Analysis . . . 33
3.5 Experiments . . . 38
3.5.1 Forecasting Horizon . . . 38
3.5.2 Experiment (1): Univariate input forecast . . . 38
3.5.3 Experiment (2): Univariate input forecast (without holidays) . 38
3.5.4 Experiment (3): Multivariate input forecast (without holidays) 38
3.5.5 Expectations . . . 38
3.6 Data Preparation . . . 39
3.6.1 Feature Engineering . . . 39
3.6.2 Data Split . . . 41
3.6.3 Time Series Forecasting as a Supervised Problem . . . 41
3.6.4 Data Scaling . . . 43
3.7 Model Evaluation . . . 43

4 Results **45**

5 Discussion and Conclusion **49**
5.1 Experiments . . . 49
5.2 Data Collection . . . 51
5.3 Feature Engineering . . . 51
5.4 Future work . . . 52
5.5 Conclusion . . . 52

Bibliography **55**

A Appendix 1 **II**
A.1 Model Implementations . . . II
A.1.1 Naive Approach . . . II
A.1.2 Seasonal Naive Approach . . . III
A.1.3 Seasonal Historical Average . . . III
A.1.4 ARIMA . . . III
A.1.5 FFNN . . . IV
A.1.6 LSTM . . . V
A.1.7 CNN . . . V
A.1.8 Support Vector Regression . . . VI

List of Figures

1.1 Traffic congestion. 2

2.1 Expected traffic flow pattern for one weekday. 5
2.2 Time series components. 7
2.3 Visualization of how the time series was split into a training and test set. 8
2.4 Visualization of producing forecasts using the **expanding window** method. 9
2.5 Visualization of forecast production using the **moving window** method with window size = 3. 10
2.6 Visualization of **multi-step** forecast production with forecasting horizon $h = 2$, using the **moving window** method with window size = 3. 12
2.7 A correlation matrix of 10 features. The subscript illustrates the different features, not time step. 15
2.8 The structure of a (fully connected) Feedforward Neural Network with two hidden layers. x_j represents the input signals, $h_i^{(1)}$ and $h_k^{(2)}$ represent the states of the neurons in the first and second hidden layers. $\hat{y}_\ell$ represents the output signals. $w_{ij}^{(1)}$ represents the weights of the connections between x_j and $h_i^{(1)}$. $\theta_k^{(1)}$ are the thresholds of the neurons in the first layer. The other weights and thresholds in the network are represented equivalently between the two other layers. 20
2.9 RNN architecture. Each square in the figure represents one or more neurons, and each arrow corresponds to a fully connected layer between the neurons on either side of the arrow. Left: RNN with cyclic weights. Right: The same network unrolled over time (which corresponds to a FFNN). 23
2.10 Example of JSON format of a road. 25

3.1 Visualization of the workflow throughout the thesis project. 28
3.2 Proximity of the collected traffic data. 30
3.3 One document in the collection *Roads*. 30
3.4 Map of all the road segments and the information of one segment. . . 31
3.5 One document in the collection *Traffic*. 31
3.6 One document in the collection *Weather*. 32

3.7 One week of traffic flow for segment 58 between 2019-03-11 and 2019-03-17. **Top**: Raw data. **Bottom**: Same data with some noise removed. 34
3.8 One week of traffic flow for segments 6, 31, 355 between 2019-03-11 and 2019-03-17. 35
3.9 One week of average traffic flow (**top**) and average speed (**bottom**) from all 811 road segments between 2019-03-11 and 2019-03-17. . . . 36
3.10 **Top:** The traffic flow of segment 58 for the last 20% of the time series (test set). **Bottom:** Equivalent plot but with holidays removed from entire time series (including 2019-05-1 in test set as can be seen). . . 37
3.11 Map of segments. The blue dot corresponds to segment 58. The red dots correspond to segments 6, 189, 355 and 761. The traffic flow at the red dots are highly correlated with the traffic flow at the blue dot. 39
3.12 Correlation matrix between some of the features used in experiment (3). The weather parameters were not used, but are shown in the graph to illustrate their low correlation. 40

4.1 SVR 10 minute traffic flow forecasts plotted against the actual measured traffic flow. 47
4.2 SVR 1 hour minute traffic flow forecasts plotted against the actual measured traffic flow. 47
4.3 FFNN 6 hour traffic flow forecasts plotted against the actual measured traffic flow. 47
4.4 FFNN 12 hour traffic flow forecasts plotted against the actual measured traffic flow. 48
4.5 CNN 24 hour traffic flow forecasts plotted against the actual measured traffic flow. Top: Original time series used. Bottom: Holidays removed from time series. 48

List of Tables

1.1 Report variable convention. 3

2.1 Forecasting variable convention. 7
2.2 Mappings between input and output for time series forecasting as a supervised problem. $p = T - n - h + 1$. 16

4.1 Forecasting results from **experiment (1)**. 46
4.2 Forecasting results from **experiment (2)**. Holidays were removed from the time series. 46
4.3 Forecasting results from **experiment (3)**. Spatial correlations were taken into account. Models with a (*) were not evaluated for multivariate inputs, and the best results from previous experiments are instead used for comparison. 46

A.1 Chosen window sizes for each forecasting horizon used in all experiments. II
A.2 FFNN hyperparameters. IV
A.3 LSTM hyperparameters. V
A.4 CNN hyperparameters. VI

1
Introduction

This section gives some background on traffic forecasting and motivates the importance of Intelligent Transportation Systems. Additionally, the objectives and limitations of the report are described.

1.1 Background and Motivation

The transportation industry was responsible for 28% of global carbon dioxide emissions in 2014 [1]. The number of traffic-related deaths in 2013 was 1.25 million [2]. Additionally, traffic congestion at peak hours reaches unacceptable levels in many parts of the world. These are all serious issues caused by current transportation systems, and optimization through the usage of modern technologies is necessary for the required improvements. A lot of the innovation that is part of the solution already exists and is what makes up *Intelligent Transportation Systems* (ITS). Directive 2010/40/EU of the European Parliament and of the Council of 7 july 2010 defined ITS in the following way [3]:

> ITS integrate telecommunications, electronics and information technologies with transport engineering in order to plan, design, operate, maintain and manage transport systems.

This definition indicates that any information technology that aids transportation in one way or another can be included as one of the many innovations under the term ITS. Applications that provide travel times or the most efficient route to a given destination are examples of such technologies. Traditionally, these technologies functioned based on simplistic evaluations, and could only be reactively updated based on occuring events. However, proactive adaptation to the ever changing dynamics of urban traffic can be achieved. This is done through approximative forecasting of future traffic patterns. Naturally, this would greatly improve the performance of existing ITS technologies. Achieving this, however, requires historic measurements of the parameters to be forecasted. Such parameters could include the traffic flow and speed at some location. Measurements of these parameters can be done in many ways, such as video detection, inductive loops and magnetic sensors [4]. Additionally, possession of historic data of traffic incidents and various weather parameters may also be useful, as these often impact the traffic quite heavily. Subsequently, this data can be analyzed and may reveal various traffic patterns. In turn, these patterns could make it possible to forecast future traffic situations.

Figure 1.1: Traffic congestion.

Forecasting the future based on historic data dates back to 1805 with techniques like linear regression [5] and is a well studied area [6]. These studies have given rise to many statistical models for predicting some future parameter based on historical data. However, traffic flow as a function of time is not entirely deterministic due to various random events that affect the traffic. There exists a countless number of these events but examples of the most impactful ones are the current weather, traffic incidents and holidays. In many cases it is therefore difficult for the traditional forecasting models to produce good results as they are unable to capture the nonlinearity in the data [7].

Due to the recent advancements in the field of *Artificial Intelligence* (AI) and an exponential growth in historic data, forecasting has experienced great improvements. More specifically, the AI sub field called *Machine Learning* (ML) has a specific set of algorithms that have proven to be capable of capturing nonlinear relationships between input and output data [8]. These algorithms typically go under the name *Deep Learning* [9] and involve *Artificial Neural Networks* (ANN), which are loosely inspired by the functionality of the biological neurons in the brain.

1.2 Objectives and Scope

This report primarily aims to research different machine learning algorithms capable of producing accurate traffic flow forecasts. A few traditional statistical forecasting techniques will also be researched in order to establish a baseline for prediction accuracy. This baseline can then be compared with the results given by the ML algorithms to decide their potential success.

The goal is to answer the following questions.

1. Which datasets are needed to generate traffic forecasts?

2. How to make traffic forecasts with machine learning?
3. Which method for producing traffic forecasts works the best?
4. Which data features are the most important in making traffic flow forecasts?

1.3 Delimitations

The project will not involve the development of any product. Instead, the primary focus will be put towards research about traffic forecasting. The forecasts will be produced for a limited number of road segments in central Gothenburg. Consequently, the project will be limited to dealing with traffic data from a static road network which is described in Section 3.2. As a result, the performance of the various forecasting techniques presented in the report may not necessarily apply to other urban road networks in other cities. Additionally, applying the same techniques on entirely different types of road networks such as freeways, could potentially fail as well. The reason for this is that the traffic patterns in different parts of the world may vary.

Also, this project will mainly focus on forecasting the traffic flow at a single road segment. Forecasting multiple segments simultaneously is however quite straightforward for the ML models. This was in fact done in a previous project that this thesis is a continuation of, and is described in Section 2.6. The previous project as well as this thesis was done in collaboration with Cybercom Sweden AB.

1.4 Syntax Convention

The report will follow a certain syntax convention when it comes to describing various mathematical data structures. These are summarized in Table 1.1.

Variable	**Description**
lower case letter (e.g. x)	scalar
bold lower case letter (e.g. $\boldsymbol{x}$)	vector
bold upper case letter (e.g. X)	matrix

Table 1.1: Report variable convention.

2
Theory

The main objective of the thesis is to forecast the traffic of various road segments. Section 2.1 therefore briefly introduces the meaning of traffic flow. Subsequently, Section 2.2 introduces forecasting as a general concept. This includes the format of the data used as a basis for making forecasts with mathematical models. Now, because the forecasting capabilities of machine learning is to be explored, Section 2.3 introduces some relevant concepts within machine learning. Section 2.4 introduces the theory behind the used forecasting models. Section 2.5 talks about the type of database used as this may help understand some of the decisions regarding the data management. Finally, Section 2.6 mentions a few past projects that touched on a similar topic.

2.1 Traffic Flow

In much of the traffic forecasting literature, the traffic follows a very particular pattern for week days, and is shown in Figure 2.1. In the morning you can see the traffic increasing and eventually reaches a peak (rush hour). After this, the traffic slowly decreases throughout the day, and then eventually increases back up again in the afternoon when people are driving home (rush hour).

When dealing with traffic forecasting there are two types of correlations that must be considered. Namely, *temporal* and *spatial* correlations. Both of these are explained below.

2.1.1 Temporal Correlations

Temporal correlations correspond to relations between past measurements of the traffic flow with the current or future measurements. In other words, the time dependent correlations. The plot in Figure 2.1 displays a *temporal pattern*.

2.1.2 Spatial Correlations

When forecasting the traffic flow at a single road segment, it would be possible to simply consider historic traffic flow from the same segment. However, it may be the case that traffic parameters (flow or speed) from other nearby road segments are good predictors as well. These are spatial correlations. For example, a lot of traffic at segment X may be an indicator of incoming traffic at some other segment Y. This may especially help forecasting far into the future.

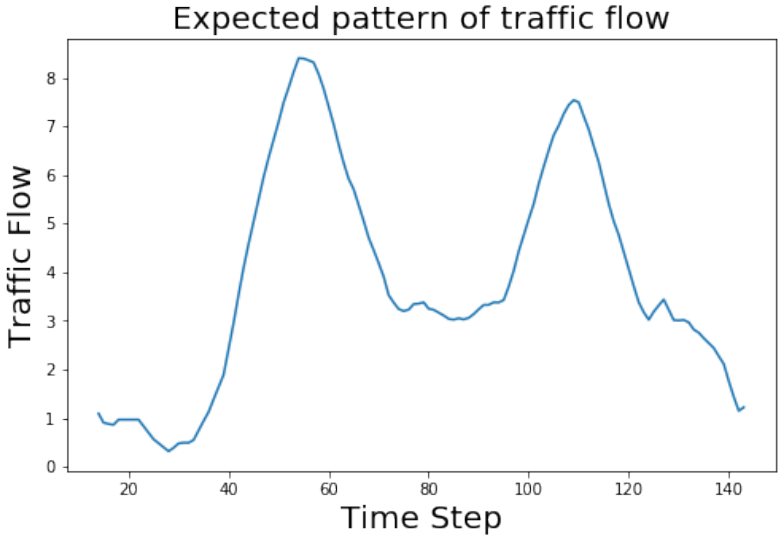


Figure 2.1: Expected traffic flow pattern for one weekday.

2.2 Forecasting

Predicting a future circumstance has been a big part of human nature since the beginning of our species. Humans ability to foresee future events and act proactively has put us ahead of other species. For example, being able to predict the position of predators or locations of food and water used to be key for survival. Making predictions is something that humans, to this day, do on a daily basis. For instance, when reading a sentence it is normal to be capable of predicting the ending of the sentence because it follows a familiar structure. This is possible because of the possession of past knowledge that allows one to make educated guesses about the future.

In the modern professional world, forecasting has many areas of application. Some examples include stock market forecasting, weather forecasting, earthquake prediction, and of course traffic forecasting. When no historic data features exist, forecasting is usually done subjectively through intuition, logic and experience. This is performed by experts in a given field and is referred to as *qualitative forecasting* [10]. Conversely, *quantitative forecasting* models are used to forecast future data based on existing historic data. This data made it possible for mathematicians to develop various mathematical models that could potentially produce more accurate predictions. However, these forecasting techniques are usually made for short- to medium-term predictions. The reason being that long-term predictions are harder to model. It is especially hard if the variable being predicted depends on many random events, and if the historic data is limited in quantity [10]. Instead, one usually uses qualitative forecasting if this is the case.

The historic data corresponds to a number of features that arrive in a time sequence. This type of data is referred to as a *time series*, and analysis of this data format is a well studied area [10]. This concept is further described in the upcoming sections.

2.2.1 Time Series

A time series is a sequence $\mathbf{x}$ of measurements of some observable variable x_t at successive points in time with an *equal* time interval between every point [10]. Equation (2.1) mathematically describes a time series with T time steps. The subscript of each element represents the time step at which the variable was measured.

$$\boldsymbol{x} = \{x_1, x_2, ..., x_T\} \tag{2.1}$$

This time series is a vector with dimensions $T \times 1$. The order of the elements is of importance because it defines the temporal structure of the data points. Furthermore, when only one feature is measured in each time step, it is called a *univariate* time series. Now, as already mentioned, forecasts are more commonly generated based on multiple historic features. For example, in this project the traffic flow is to be forecasted based on several features such as traffic speed and traffic flow (spatial correlations). In that case a *multivariate* time series is necessary. This is essentially just multiple univariate time series concatenated. This would give a matrix, which is described in equation (2.2).

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1T} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2T} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & x_{N3} & \dots & x_{NT} \end{bmatrix} \tag{2.2}$$

Each column of this matrix contains all the features of each time step in the time series. Therefore, the column vectors of the matrix corresponds to feature vectors. There are N features and T timesteps which means that the dimensions of $\mathbf{X}$ is $T \times N$. The matrix $\mathbf{X}$ can alternatively be described as follows,

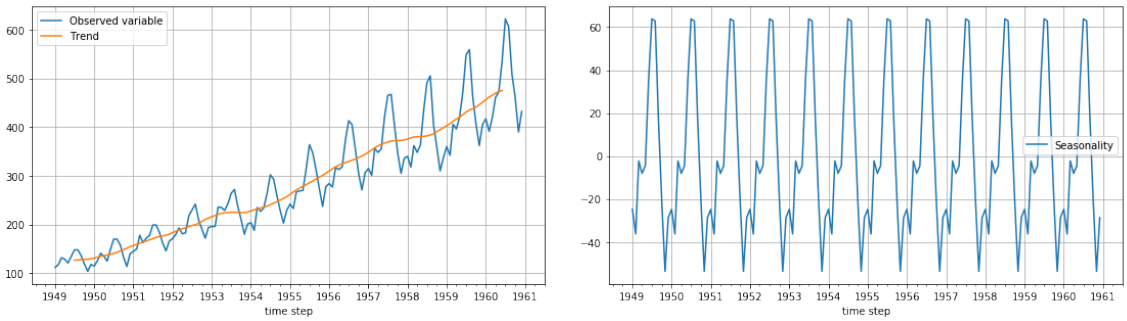
$$\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T\}, \quad \boldsymbol{x}_t \in \mathbb{R}^N$$

where each element is a feature vector of size N.

A time series that can be predicted must consist of repeating temporal patterns that can be modeled. There are three important components of a time series that often needs to be taken into account. Namely, the *seasonality*, *trend* and *noise* [10]. The trend describes the overall increase and decrease of the measured variable x_t. In Figure 2.2 (a), the blue line plots the time steps of a time series, and the orange line visualizes its trend. The seasonality describes the repeating short-term periodicity, which can also be seen in the plotted time series. Figure 2.2 (b) has captured the seasonality of the time series and visualizes it independently of trend and various noise. The noise correspond to random variation in the time series.

2.2.2 Time Series Forecasting

In most forecasting literature the involved variables are denoted as described in Table 2.1. This report will follow the same convention. Now, given a time series of historic data features $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T\}$, the idea is to produce a prediction of some



(a) Real time series (blue) and its trend (orange).

(b) Seasonality of the time series in (a).

Figure 2.2: Time series components.

feature(s) h time steps into the future. h is called the *forecasting horizon*. The prediction is denoted $\hat{y}_{T+h}$, and the generation of this can be described as shown in (2.3).

Variable	Description
x	historic data feature (input)
$\hat{y}$	predicted value (output)
y	actual value
h	forecasting horizon
T	total time steps in time series
t	arbitrary time step in time series
N	number of features

Table 2.1: Forecasting variable convention.

$$\hat{y}_{T+h} = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T), \quad f : \mathbb{R}^{T \times N} \rightarrow \mathbb{R}^1 \tag{2.3}$$

The function f is unknown and is only used as a way to represent the various forecasting models described in Section 2.4. Now, because certain models are capable of producing a multivariate output, it could also be expressed as follows,

$$\hat{\boldsymbol{y}}_{T+h} = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T), \quad f : \mathbb{R}^{T \times N} \rightarrow \mathbb{R}^M$$

where the output of f this time is a vector. This was used in the previous project (see Section 2.6), because multiple road segments were forecasted simultaneously.

2.2.3 Time Series Split

It is common for various forecasting models (and ML models in general) to split the data set into multiple parts [11]. In this project, the time series was split into a training set and a test set. The training set is used for the forecasting model to learn the temporal patterns throughout the time series. The test set is then used

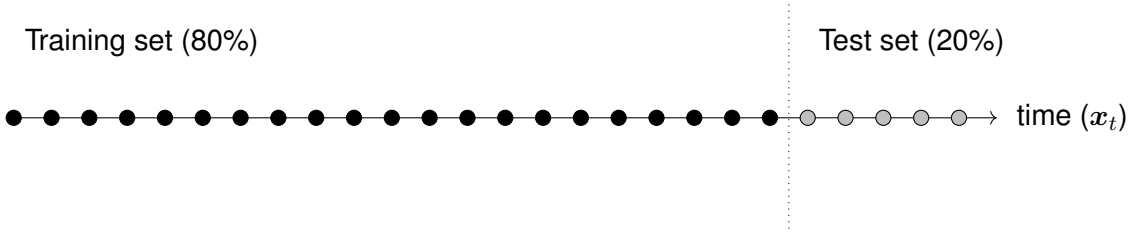


Figure 2.3: Visualization of how the time series was split into a training and test set.

to measure how well the *trained* model can produce forecasts on previously unseen data. The forecasting performance achieved on the test set is then what can be expected when actual forecasts are to be made for future time steps. It is common to have 80% of the original time series for the training set, and the remaining 20% for the test set. This is illustrated in Figure 2.3, where each dot corresponds to a time step in a time series.

It is important that the time series is not shuffled, as this will remove its temporal structure. Also, the test set must arrive *after* the training set to avoid forecasting the past. The upcoming section will talk more about *how* the accuracy of a model is evaluated on the test set.

2.2.4 Evaluating a Forecasting Model

Before a forecasting model can be employed for real-world usage, its accuracy must be evaluated. This is done by comparing the forecasted value $\hat{y}$ with its actual observed value y. The problem with this is that the actual value does not exist yet as it will take place h time steps into the future. The solution is to make forecasts for existing values that are already in the time series. For example, in a time series with T time steps, one could make a prediction for the last value in the time series based on all the previous ones as shown in (2.4).

$$\hat{\boldsymbol{y}}_T = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{T-h}) \tag{2.4}$$

In this case the predicted value is $\hat{\boldsymbol{y}}_T$, which is at a time step within the limits of the time series. The actual value at this time step is accessible from the time series as $\boldsymbol{x}_T$, and is denoted $\boldsymbol{y}_T$. However, in order to get a good estimation of how well a model is performing, many forecasts are necessary. This is done by moving through the entire time series and iteratively produce forecasts. There are two methods that accomplish this. The first method consists of an *expanding window* of time steps that each forecast is based on [12]. This is illustrated in Figure 2.4 on a time series with $T = 12$ time steps. The black dots correspond to the features that are used as the basis for a forecast. The pink dots correspond to the time step being forecasted. The white dots are unused features. Notice that the first forecast is made for x_3. This is because some minimum number of time steps is needed to base the forecast on. In this particular case, the first three time steps are considered the minimum requirement for producing a good forecast. The number of forecasts therefore becomes $T - 3 = 9$. The second method is a *moving window* of constant size that each

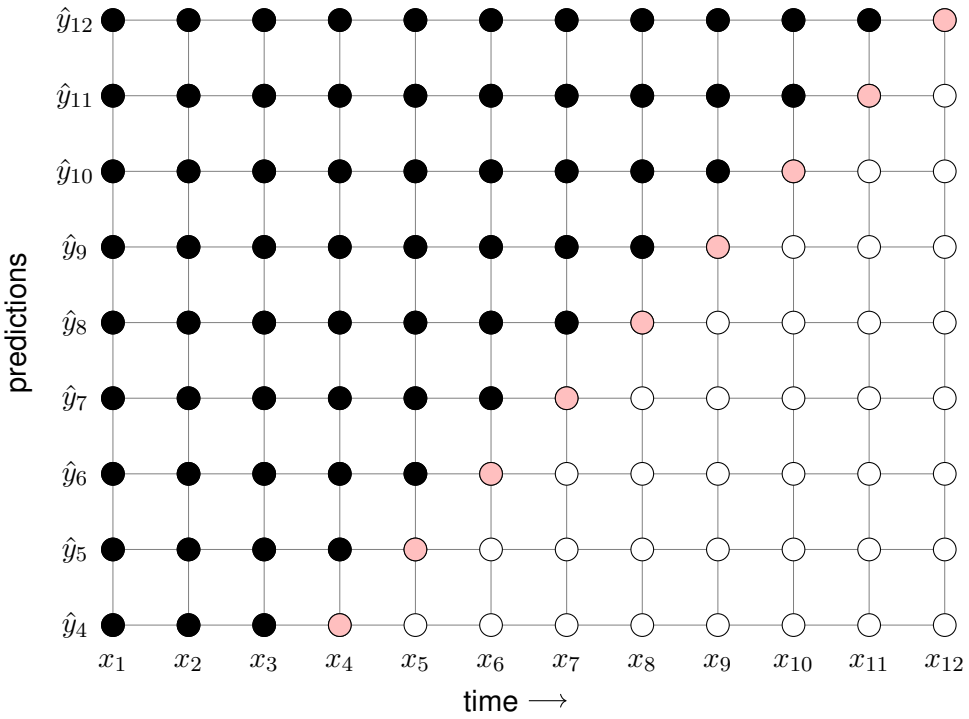


Figure 2.4: Visualization of producing forecasts using the **expanding window** method.

forecast is based on [12]. This is similarly illustrated in Figure 2.5.

Out of the two methods, neither is superior in general. Both of them come with a number of pros and cons, which was discussed by Clark *et al.* [12]. In short, selecting the optimal window size is not easy. It depends on the overall structure of the available data. If the entire time series follows a similar pattern, then the bigger the window size the better. Conversely, if old time steps are of very little relevance in predicting more recent time steps, then a shorter window size may be preferable. Also, if the time series used is very large, using an expanding window will lead to an unfeasible computational complexity. As a consequence, fitting the forecasting model as well as generating forecasts will be very time consuming.

Accuracy Metrics

When a large number of forecasts has been generated, by using either of the two methods above, the next step is to evaluate the accuracy. In this report, the accuracy metric used is called *Mean Absolute Error* (MAE) and is defined as shown in (2.5).

$$MAE = \frac{1}{T}\sum_{i=1}^{T}|y_i - \hat{y}_i| \tag{2.5}$$

MAE measures the average magnitude of the errors across all predictions made. $\hat{y}_i$ corresponds to the i:th forecast, and y_i is its actual value. An error of 0 means that all forecasts were equal to the actual value, which is the best case scenario.

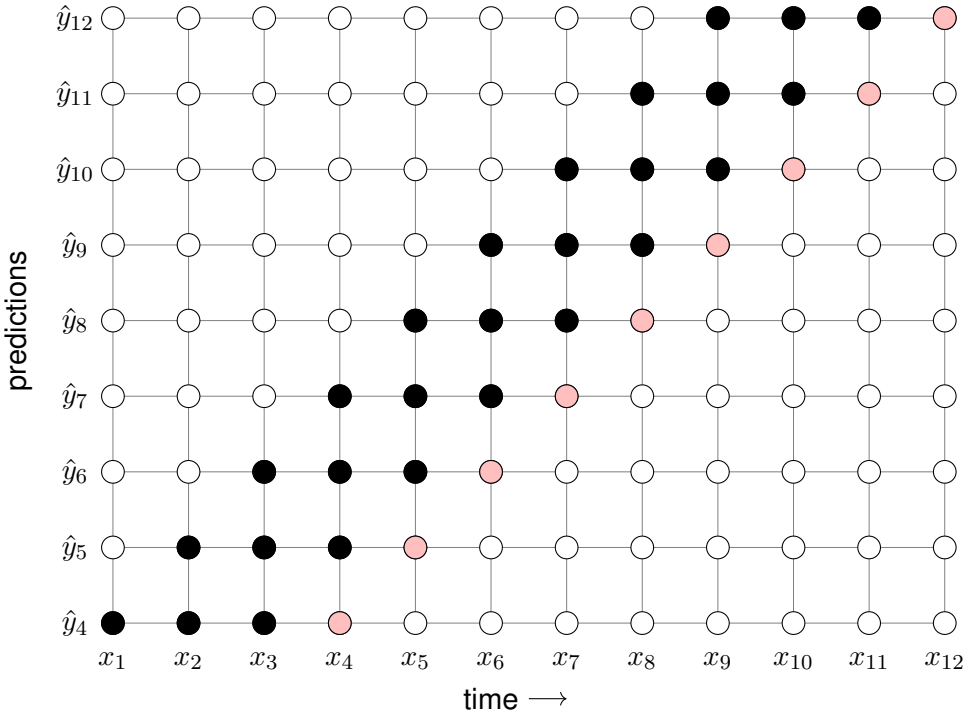


Figure 2.5: Visualization of forecast production using the **moving window** method with window size = 3.

2.2.5 Multi-step Forecasting

Single-step forecasting corresponds to $h = 1$, and is generally quite simple to perform. Most forecasting models produce very accurate results for single-step forecasting, since the value to be predicted is usually similar to the current time step. However, multi-step forecasting is of course much more interesting since this involves predicting further into the future. As expected, forecasting models have a much harder time doing this accurately. Different methodologies for forecasting multiple time steps into the future was explored by Bontempi *et al.* [13], two of which are presented below.

- **Direct Strategy**

  Out of the suggested methods, the one that was the most appropriate for this project, is called the *Direct Strategy*. This is the most intuitive one since it simply consists of training one forecasting model for each prediction horizon h that is of interest. This is expressed in (2.6) .

$$\begin{aligned} \hat{y}_{T+1} &= f_1(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T) \\ \hat{y}_{T+2} &= f_2(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T) \\ &\vdots \\ \hat{y}_{T+h} &= f_h(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T) \end{aligned} \tag{2.6}$$

  Adapting a given forecasting model to a certain value of h is done by for-

matting the data accordingly. This is further explained in Section 2.3.5. The downside of the direct strategy is that separate models must be trained for each value of h, which takes a lot of time. As this project is mainly about comparing the results of different forecasting models, this was sufficient.

- **Multiple Output Strategy**

  If one is interested in producing forecasts for multiple time steps simultaneously from one model, the *Multiple Output strategy* could be used. Not all models are capable of this, but this is done by producing a vector output of the model as shown in (2.7).

$$\begin{bmatrix} \hat{y}_{T+1} \\ \hat{y}_{T+2} \\ \vdots \\ \hat{y}_{T+h} \end{bmatrix} = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T) \tag{2.7}$$

  Here a prediction for all time steps up until the horizon is given. This is similar to producing a multivariate output of a model as discussed in Section 2.2.2, except there the output was a vector of multiple features at the same time step. However, the same principle is applied.

In the previous section, Figure 2.4 and 2.5 displayed how a single-step forecasting model is evaluated. The evaluation of a multi-step forecasting model using the direct strategy is done in an equivalent manner, and is shown in Figure 2.6 for the moving window method. The forecasting horizon used in the example is $h = 2$. Notice how the last ($\hat{y}_{12}$) lands outside of the time series. Consequently, this forecast is unusable since there is no expected value for the given time step. The number of forecasts therefore becomes $T - (h - 1) - 3 = 8$. In general, for a time series with T time steps, a window size of n and a forecasting horizon of h, the number of forecasts becomes $T - h - n + 1$.

2.3 Machine Learning

Many researchers in the field of Artificial Intelligence (AI) are trying to figure out how AI systems can reach, at a minimum, the same level of general intelligence as humans. This is often referred to as *Artificial General Intelligence* (AGI) [14]. Now, as previously mentioned, making predictions is a very important part of human cognition. This means that having AI systems capable of producing accurate forecasts is of great importance to the goal of reaching AGI. Thus far the AI sub-field *Machine Learning* (ML) has had some interesting successes in doing just that. This section introduces ML and describes how it can be applied to making time series forecasts.

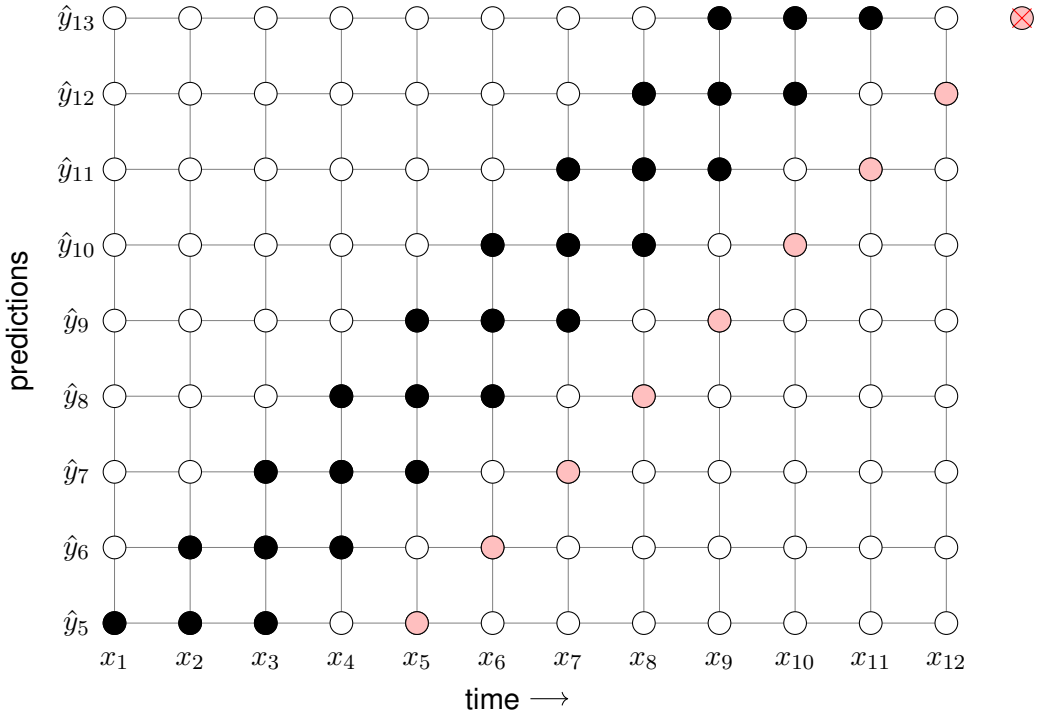


Figure 2.6: Visualization of **multi-step** forecast production with forecasting horizon $h = 2$, using the **moving window** method with window size = 3.

2.3.1 Introduction

Machine learning is the study of algorithms and data which a computer system uses to improve its knowledge of a given task. In order to understand machine learning better, the following quote by Thomas M. Mitchell gives a good definition of the subject [15]:

> 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 performance measure P is a measurement of how well the computer system has learned the given task T. An example of evaluating the performance of a machine learning algorithm was introduced in Section 2.2.4, namely using the accuracy metric MAE. Examples of the experience E and the task T will be discussed in the next section.

2.3.2 Types of Learning Algorithms

The experience E refers to the type of data available to the machine learning algorithm during the learning phase. Expectedly, the training data must be large in quantity if the system is expected to generalize its learned knowledge to previously unseen data [11]. Generally, the type of available data comes in two categories. The first type applies *supervised learning* and the second *unsupervised learning*.

- **Supervised learning**

  In order to apply supervised learning, the data must consist of predetermined mappings between input signals ($\boldsymbol{x}$) and their corresponding output ($\boldsymbol{y}$) [16]. The structure of the data given to the ML model must be a set of p pairs like shown in (2.8).

$$\{(\boldsymbol{x}^{(1)},\ \boldsymbol{y}^{(1)}),\ (\boldsymbol{x}^{(2)},\ \boldsymbol{y}^{(2)}),\ \dots\ ,\ (\boldsymbol{x}^{(p)}, \boldsymbol{y}^{(p)})\},\ \boldsymbol{x} \epsilon \boldsymbol{X}, \boldsymbol{y} \epsilon \boldsymbol{Y} \tag{2.8}$$

  These pairs correspond to the mappings between input and output that the model is meant to learn. By studying all of these pairs, the goal of the ML model is to find a function $f : \boldsymbol{X} \rightarrow \boldsymbol{Y}$ that correctly maps each input to the corresponding output as shown in (2.9).

$$\boldsymbol{y}^{(\mu)} = f(\boldsymbol{x}^{(\mu)}),\ \forall \mu \epsilon \{1, ..., p\} \tag{2.9}$$

  Interestingly, some models are powerful enough to produce the correct output $\boldsymbol{y}^{(\mu)}$ even if the input is only roughly equal to $\boldsymbol{x}^{(\mu)}$. This means that the model is resistant to noise that might occur when deploying the model for real-world usage [17].

  The task T refers to what the used learning algorithm is supposed to achieve. For supervised learning the two most common tasks are *classification* and *regression* [11]. For classification the target outputs ($\boldsymbol{Y}$) correspond to a number of discrete classes. Each input $\boldsymbol{x}^{(\mu)}$ must be mapped to one of these classes. Multiple inputs may be mapped to the same class. A typical use case of classification is object detection [18]. In this case the model is given a large number of images and learns to classify some specified object in each image. For example, with the output classes $\boldsymbol{Y} = \{$"*cat*", "*dog*", "*piano*"$\}$, an example of input/output pairs is given in (2.10).

$$\{(\boldsymbol{x}^{(1)},\ \text{"}piano\text{"}),\ (\boldsymbol{x}^{(2)}, \text{"}cat\text{"}),\ (\boldsymbol{x}^{(3)}, \text{"}dog\text{"}),\ (\boldsymbol{x}^{(4)}, \text{"}cat\text{"}),\ ...\} \tag{2.10}$$

  For regression on the other hand, the output signals are continuous numbers such that $\boldsymbol{Y} \subset \mathbb{R}$. Another way of thinking about regression is approximating a real valued function [15]. In this project, the output of the model will be the predicted traffic flow for some time step. The traffic flow is a real number between 0 and 10, thus regression will be used.

- **Unsupervised learning**

  For unsupervised learning only the input data is given, and the machine learning algorithm is expected to find eventual patterns in the data on its own [15]. For unsupervised learning the tasks include clustering and association. This technique of learning was not utilized in this project and will therefore not be explained in more detail.

2.3.3 Overfitting and Underfitting

When a machine learning model learns unique features to the training set, and consequently can't generalize well to new data (i.e. the test set), the model is said to *overfit* the training data [11]. Various ML models use different *regularization* techniques to avoid overfitting. Conversely, when the model can't learn from the training set at all, it is instead *underfitting* [11]. In this case the model will perform poorly on both the training set as well as the test set. In the case of avoiding underfitting, the quality of the training data must be considered. For example, the data must contain relevant features for the learning objective, and various noise and outliers must be removed. Note that removing noise also helps against overfitting. The reason being that the model risks learning the noise of the training data, which of course does not exist in the test set. Additionally, adding random noise in each iteration during the learning phase can also help against overfitting. This is possible because it prevents the model from learning some existing noise in the training set really well. These problems are considered in more detail in the following two sections.

2.3.4 Feature Engineering

Providing relevant features for a given task is of course crucial for a successful ML model. For example, trying to predict the traffic flow based on the current price of gold is likely not going to be very successful. Intuitively, these two variables are entirely unrelated, which makes it impossible for the model to achieve anything interesting. The process of uncovering the most relevant and useful features is called *feature engineering* [11].

A common way to decide which feature to utilize is by using a *correlation matrix* [11]. For a feature vector with 10 features $\boldsymbol{x}_t = [x_1, x_2, \ ... \ , x_{10}]$, a correlation matrix is usually plotted using a heatmap as shown in Figure 2.7. This heatmap visualizes the correlation between all features. The correlation ranges from -1 to 1. A correlation of 1 represents maximum correlation, whereas -1 represents maximum *inverse* correlation. In both of these cases, the correlation is considered high and such features may be useful for the training phase. In contrast, a correlation of 0 represents no correlation at all. Note that all values across the diagonal are equal to one because each feature is fully correlated with itself. Now, lets say that the feature x_1 is to be predicted. According to the first column of the heatmap, only the features x_{10}, x_5, x_4, and x_3 are strongly correlated with x_1. Using these four features for training the model may be a good place to start. However, deciding which of the features to use as predictors is however not as simple as picking the most correlated ones. The reason being that some nonlinear and more complex models may be able to find temporal patterns among these features that the correlation matrix will not.

There are different types of correlations that could be measured. The type used in this project is called the *spearman correlation*. A high spearman correlation (+1) is achieved when two variables consistently decrease and increase at the same time steps. A high inverse spearman correlation (-1) is achieved when one variable

decreases when the other increases. [19]

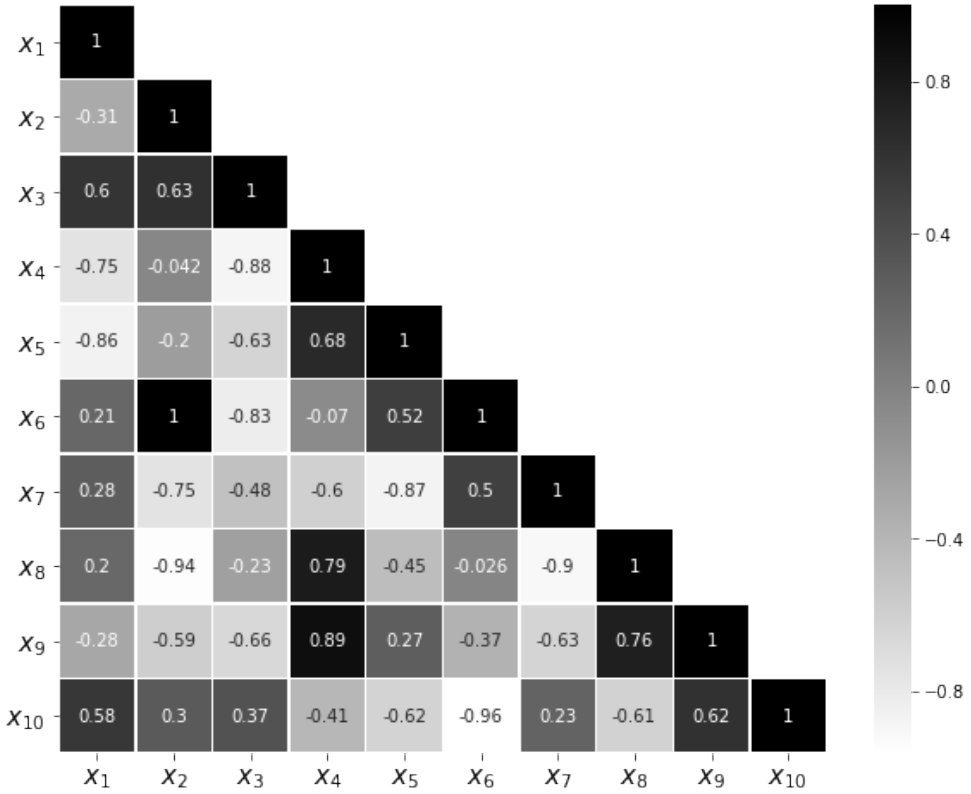


Figure 2.7: A correlation matrix of 10 features. The subscript illustrates the different features, not time step.

2.3.5 Data Preparation

Next up is preparing the data appropriately for a given model. The first part to consider is the quality of the data. Secondly, for a time series forecasting context, the data must be organized in a particular way. These two steps of data preparation are discussed below.

Data Preprocessing

If a feature consists of a few abnormal measurements (e.g. due to problems with a measuring sensor), these may have to be removed. If they are kept in the data, the model might be tricked into learning patterns that it should not, and therefore overfit the training data. Because of this, it is important to visualize the data in various ways such that these abnormalities can be discovered and fixed.

Additionally, many ML models have certain expectations of the data. For example, all features of the dataset may need to be scaled to a given range (e.g. all values between 0 and 1). This is done as shown in (2.11), where $(x_{min},\ x_{max})$ is the specified range.

$$x_{std} = \frac{x - x_{min}}{x_{max} - x_{min}}$$
$$x_{scaled} = x_{std} * (x_{max} - x_{min}) + x_{min} \tag{2.11}$$

Time Series Forecasting as a Supervised Problem

Existing attempts at applying machine learning techniques to time series forecasting have been done through supervised learning. As a result, the data must be formatted in a particular way before feeding it as input to some machine learning model. Achieving this was described in 2002 by Dietterich [20], where he refers to it as a *sequential supervised learning problem*. Furthermore, Bontempi *et al.* [21] gave a more detailed explanation in a paper from 2012. A brief description of how this works is given below.

In order for supervised learning to be applied, the data must be structured such that it follows the structure explained in Section 2.3.2. In other words, multiple mappings between inputs and outputs must be explicitly given to the model. As described in Section 2.2, predicting a value in a time series is done based on many previous values in the same time series, e.g. $\hat{\boldsymbol{y}}_T = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{T-h})$. Therefore, in order to give the model some temporal context, the inputs must be sequences of feature vectors and not just a single feature vector as explained in Section 2.3.2.

Since the machine learning model expects many pairs of inputs and outputs, the moving window approach is applied during the learning phase as well. The number of pairs p corresponds to how many windows the time series is split up into, which was already evaluated in Section 2.2.5 to be $T - n - h + 1$. Because of this, the window size n and forecasting horizon h are two parameters that must be chosen before the data is formatted accordingly. Training different models for each value of h is how the direct strategy for multi-step forecasting is applied. Table 2.2 visualizes these mappings for a given time series $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\}$ with T time steps, a window size of n and a forecasting horizon of h. Note that $\boldsymbol{y}_t$ is synonymous with $\boldsymbol{x}_t \; \forall \; t \in \{1, \ldots, T\}$.

input	output
$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}^{(1)}$	$\boldsymbol{y}^{(1)}_{n+h}$
$\{\boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_{n+1}\}^{(2)}$	$\boldsymbol{y}^{(2)}_{(n+1)+h}$
$\{\boldsymbol{x}_3, \boldsymbol{x}_4, \ldots, \boldsymbol{x}_{n+2}\}^{(3)}$	$\boldsymbol{y}^{(3)}_{(n+2)+h}$
$\vdots$	$\vdots$
$\{\boldsymbol{x}_{T-n-h+1}, \boldsymbol{x}_{T-n-h+2}, \ldots, \boldsymbol{x}_{T-h}\}^{(p)}$	$\boldsymbol{y}^{(p)}_T$

Table 2.2: Mappings between input and output for time series forecasting as a supervised problem. $p = T - n - h + 1$.

The model will now iterate through these mappings window by window and learn the temporal patterns throughout the time series. By the end of the learning phase, the model will be an approximation of the function f shown in (2.12) (i.e. regression). f requires new inputs to also be sequences of size n.

$$\boldsymbol{y}_{t+h} = f(\boldsymbol{x}_{t-n+1}, \boldsymbol{x}_{t-n+2}, \ldots, \boldsymbol{x}_t), \ \forall t \in \{n, \ n+1, \ \ldots, \ T-h\} \tag{2.12}$$

When a new sequence of n inputs is fed into the model, it will produce an output that roughly resembles the output that it was trained to produce for a similar sequence during the learning phase. If the input sequence does not look like anything given during the learning phase, the results will of course be inaccurate. This is a sign that the training data was not informative enough for generalizing to the data in the test set.

2.3.6 Hyperparameters

Machine learning models often depend on many parameters. Some parameters used do not directly belong to the model itself, and remain constant throughout the learning phase. These are called *hyperparameters* and instead belong to the ML algorithm (which is what trains the ML model) [11]. Tuning the hyperparameters is a crucial step in avoiding a poorly performing model. Some examples of hyperparameters are given in Section 2.4 when the used models are presented. Optimizing the hyperparameters is usually a difficult task, as it requires a trial and error approach. A technique called *grid search* is often used to automate the process of empirically finding the best combination of hyperparameters [11].

2.4 Forecasting Models

This section introduces the underlying techniques behind all of the used forecasting models in this project. How the techniques were adapted to a forecasting context will be explained in Chapter 3. In existing forecasting literature, the types of forecasting models are often split into two categories; *parametric* and *non-parametric*. The former consists of models that depend on a finite number of parameters. The latter consists of models where the number of parameters is unspecified [22], and may grow as the amount of data grows [23]. The parametric models include the traditional time series models originally introduced by Box and Jenkins [6]. This type of forecasting has been named the *Box-Jenkins method*, and is an important part of time series analysis. One of these models include *Autoregressive Integrated Moving Average* (ARIMA) and is introduced briefly in Section 2.4.2. Nonparametric models include *Support Vector Machines* (SVM) (with RBF kernel) and Artificial Neural Networks. Both SVM and ANN are machine learning algorithms, meaning that their basic functionality follows that explained in Section 2.3. One SVM model and different types of ANNs are introduced in Sections 2.3.3-5. The nonparametric models are known to perform better on nonlinear data [24]. This would in theory make them good candidates for traffic forecasting due to the stochastic nature of urban traffic.

2.4.1 Baseline Methods

In order to establish a benchmark for the more complex models, four baseline methods will be used. They are described below.

- **Naive Method**

  Sets prediction equal to the most recent time step.

  $$\hat{y}_{T+h} = x_T$$

- **Seasonal Naive Method**

  Sets prediction equal to the corresponding time step one seasonality cycle back in time. The period of the seasonality is denoted as m and k is the smallest integer greater than $\frac{h-1}{m}$. T must be greater than m.

  $$\hat{y}_{T+h} = x_{T+h-km}$$

- **Seasonal Historical Average**

  Sets prediction equal to the average of corresponding time steps of all previous seasons. T must be greater than m.

  $$\hat{y}_{T+h} = \frac{x_{T+h-km} + x_{T+h-(2*km)} + ... + x_{T+h-(T*k)}}{\left\lfloor \frac{T}{m} \right\rfloor}$$

2.4.2 ARIMA

Two traditional forecasting techniques are *Autoregression* (AR) and *Moving Average* (MA). Combining these two methods into one model was introduced in a 1951 thesis by Peter Whittle, and is called *Autoregressive Moving Average* (ARMA). In short, this technique uses past values of a time series and linearly maps it to an output that could be used as a forecast. Because it is a linear model, certain expectations must be put on the time series if the coefficients of the model are to be trusted. In other words, properties of the time series, such as mean and variance, can not depend on the time at which the series is observed. A time series that follows these properties is called *stationary*. Ensuring a constant mean and variance of a time series is done by removing the trend and seasonality, respectively. There are various data transformation techniques that achieve this. One method is called *differencing*, and involves subtracting the previous observation x_{t-1} with the current x_t. Sometimes the time series may need to be differenced several times before reaching a stationary point. The number of times it is differenced is called the degree of differencing. For ARMA the time series must be made stationary before usage. [6]

ARIMA is a generalization of ARMA that does not require the time series to be stationary upon input. Instead, the model accepts a parameter that denotes the required degree of differencing, and performs the appropriate transformation automatically. [6]

2.4.3 Feedforward Neural Network

An *Artificial Neural Network* (ANN) is a type of machine learning technique, and was the primary use in this project. ANNs are inspired by the biological neural networks that make up our brains. The biological neural networks consist of a large number of neurons interconnected by synapses. These neurons receive and output information in the form of electrical signals [17]. In an artificial neural network, a neuron is represented by a real number. Each connection in the network has a weight associated with it, and in a sense decides the importance of a given input. The output of each neuron is evaluated as a weighted sum of all the inputs passed into some nonlinear activation function [8].

Different types of ANNs were historically the first type of ML models used for the purpose of time series forecasting [23]. A *Feedforward Neural Network* (FFNN) is one of these, and also the simplest kind. In an FFNN all the connections between the neurons do not form any cycles (see Figure 2.8), meaning that the flow of information is straightforward through the network [16]. How a neural network is used in the context of supervised learning will be explained below, using the simple network in Figure 2.8 as an example.

An FFNN accepts input data in one dimension $\boldsymbol{x}^{(\mu)} \epsilon \mathbb{R}^n$ and generates an output in a potentially different dimension $\hat{\boldsymbol{y}}^{(\mu)} \epsilon \mathbb{R}^m$ based on a set of parameters. The superscript μ shows that the network is fed multiple input/output mappings that it is meant to learn (supervised learning). For each pattern μ the output is compared to the expected value $\boldsymbol{y}^{(\mu)} \epsilon \mathbb{R}^m$, and updates the network parameters in case of eventual errors. The hope is that the next time the same pattern is given as input, the network will produce an output that is slightly closer to the expected value. Repetition of this process is what corresponds to the network learning over time [16]. The learning process is described below based on the network given in Figure 2.8.

Forward Propagation

The first step is to evaluate the output values $\hat{\boldsymbol{y}}^{(\mu)}$ based on the given input $\boldsymbol{x}^{(\mu)}$. This is done by propagating forward through the network [16]. In the given network shown in Figure 2.8, there are two hidden layers. The first step is to evaluate the states of the neurons in the first hidden layer. This is done by taking the weighted sum of all input signals $x_j^{(\mu)}$ based on the connection weights $w_{ij}^{(1)}$ and the thresholds $\theta_i^{(1)}$ as shown in (2.13).

$$h_i^{(1)} = \phi_1(\sum_{j=1}^{n} w_{ij}^{(1)} x_j^{(\mu)} - \theta_i^{(1)}) \tag{2.13}$$

With the states of the neurons in the first hidden layer, the states of the neurons in the second hidden layer can be evaluated in a similar fashion. This is shown in (2.14).

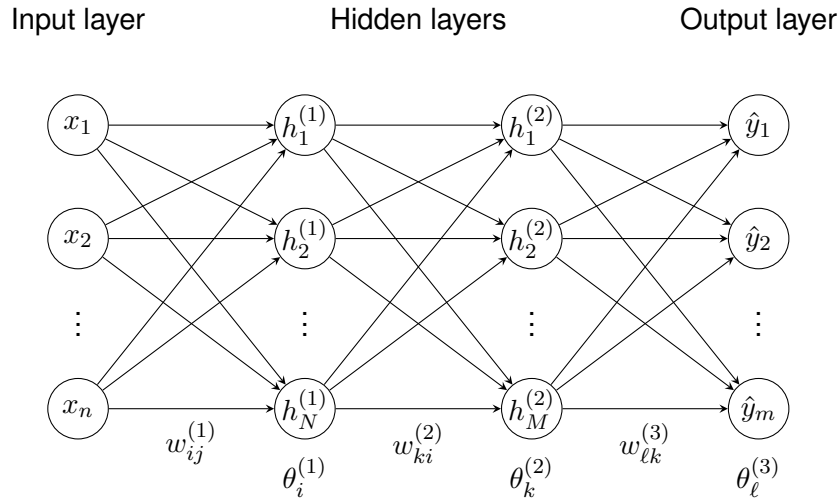


Figure 2.8: The structure of a (fully connected) Feedforward Neural Network with two hidden layers. x_j represents the input signals, $h_i^{(1)}$ and $h_k^{(2)}$ represent the states of the neurons in the first and second hidden layers. $\hat{y}_\ell$ represents the output signals. $w_{ij}^{(1)}$ represents the weights of the connections between x_j and $h_i^{(1)}$. $\theta_k^{(1)}$ are the thresholds of the neurons in the first layer. The other weights and thresholds in the network are represented equivalently between the two other layers.

$$h_k^{(2)} = \phi_2(\sum_{i=1}^{N} w_{ki}^{(2)} h_i^{(1)} - \theta_k^{(2)}) \tag{2.14}$$

Finally, the outputs are evaluated based on the neurons in the second hidden layer, as shown in (2.15).

$$\hat{y}_\ell^{(\mu)} = \phi_3(\sum_{k=1}^{M} w_{\ell k}^{(3)} h_k^{(2)} - \theta_\ell^{(3)}) \tag{2.15}$$

$\phi_{1,2,3}$ are the activation functions. The most common activation function is called ReLU and is defined as $\phi(x) = max(0, x)$ [25]. Using nonlinear activation functions is what give neural networks their nonlinear characteristics [16]. The three activation functions used in each layer may not necessarily be the same [25]. Note that forward propagation is used both during training and also when the network is to make predictions after the learning phase.

Cost Function

With the outputs $\hat{\boldsymbol{y}}^{(\mu)}$ evaluated, the next step is to compare them with the expected values $\boldsymbol{y}^{(\mu)}$. This is done using a *cost function*. An example of a simple one is the *quadratic* cost function [25], and is defined as shown in (2.16) for the given network.

$$H = \frac{1}{2} \sum_{\ell=1}^{m} (y_\ell^{(\mu)} - \hat{y}_\ell^{(\mu)})^2 \tag{2.16}$$

It is clear from (2.16) that the cost function H is minimized (equal to zero) precisely when $\hat{\boldsymbol{y}}^{(\mu)} = \boldsymbol{y}^{(\mu)}$, which is exactly the learning objective of the network. The goal is then to employ various mathematical optimization techniques for minimizing the function H.

Backpropagation

Efficiently minimizing the cost function of an ANN was originally described by Rumelhart *et al.* [8], using an algorithm called *backpropagation*. This is done by propagating backwards through the network after the error is evaluated, and updating the various parameters that define the network (weights and thresholds). When deciding how the parameters are to be updated, a mathematical technique called *gradient descent* [26] is utilized. In short, the partial derivative of H is taken with respect to the weights and thresholds, separately. This derivative then reveals in which direction the parameters are to be updated in order to move one step closer to minimizing H. After each update, forward propagation is done again, which results in a new value of $\hat{\boldsymbol{y}}^{(\mu)}$. The hope is that this new value gives a slightly smaller value of H. This process is then repeated until the cost function is minimized. If the input data was quantitatively large enough and accurately representative of future inputs, the network will be able to generalize its learned knowledge to entirely new data.

Hyperparameters

Thus far several parameters used by the model have been mentioned, e.g. the weights and thresholds. These parameters change over time as the model is trained. There are, however, a few additional parameters that must be chosen. These conversely remain constant throughout the learning phase, and are called hyperparameters. There are no general values to assign the hyperparameters, as it is highly situational. Instead, one must use a trial and error approach in deciding the optimal values. Examples of important hyperparameters associated with ANNs are listed below.

- **Number of hidden layers**

  Adding more hidden layers to the model may allow the network to recognize more complex mappings between the input and output layers. A network with two or more hidden layers is often considered *deep learning* [9], and is currently the most successful technique in machine learning. Adding many layers could improve the accuracy of the model, to a certain degree. Adding too many layers might make the network too complex, and consequently overfit the training data. Too many layers may also make the training time too long. [17]

- **Number of neurons in each hidden layer**

  The number of neurons in each layer is in a similar fashion dependent on

how complex the given problem is. Too few neurons make the network unable to capture all the information. Logically, the more training data and input/output mappings the network must learn, the more neurons are required. However, too many neurons make the network more prone to learning unique features of the training set, and may thus start overfitting. [17]

- **Number of epochs**

  One epoch correspond to one iteration through all pairs of input/output mappings. This is rarely enough for minimizing the cost function. The network may therefore need many epochs of training before all knowledge has been learned. Too many epochs, however, may lead to overfitting. [17]

- **Learning rate**

  During the learning phase the weights and thresholds are iteratively updated in order to minimize the cost function. The learning rate decides how quickly these parameters are to be updated, by limiting the amount of change by a certain factor. A learning rate that is too low makes the training very slow. A learning rate that is too high makes the algorithm overshoot, which increases the risk of missing the global minima of the cost function. [17]

2.4.4 Recurrent Neural Network

While the simple type of FFNN introduced in the previous section has some really interesting use cases, it does have a few limitations. An FFNN is really good at learning mappings between different inputs and outputs. However, each of the mappings μ are entirely independent of each other since information flows in one direction. Consequently, when an FFNN moves through a time series trying to learn its dependencies, it lacks temporal context. A different type of ANN was introduced as a solution to this and goes by the name *Recurrent Neural Network* (RNN). RNNs allow cyclic weights (see left in Figure (2.9)), which allows information to flow backwards. This gives the network a type of memory, such that it can remember past input features when considering newer ones. Achieving this is done by unrolling the recurrent network over time. This yields a new structure that follows that of an FFNN, which can then be trained with the backpropagation algorithm as described in the previous section. More specifically, an algorithm called *Backpropagation Through Time* (BPTT) [27] is used. All of this means that RNNs are very good at dealing with data that arrives in a time sequence, because it can remember past time steps. A typical use case of RNNs is therefore time series prediction, as it can, compared to FFNNs, much better find the temporal patterns of a time series. Other applications of RNNs include machine translation, which involves translating text or speech from one language to another. Sentences can be seen as a sequence of words. When trying to understand the meaning of the sentence, it is crucial to remember the past words when trying to understand the context of new ones. This is something that, for instance, Google uses in their service Google Translate [28].

A problem with ANNs in general is something called the *vanishing gradient problem* [29]. In short, this means that the network stops learning when you introduce many layers in the network. An RNN especially suffers from this since the number of layers depends on how many time steps are used when unrolling the network. For large amounts of time steps, the network stops learning. A particular type of RNN was therefore introduced by Hochreiter as a solution, and is called *Long Short-Term Memory* (LSTM) [29]. LSTM networks are the primary forecasting model used in the project.

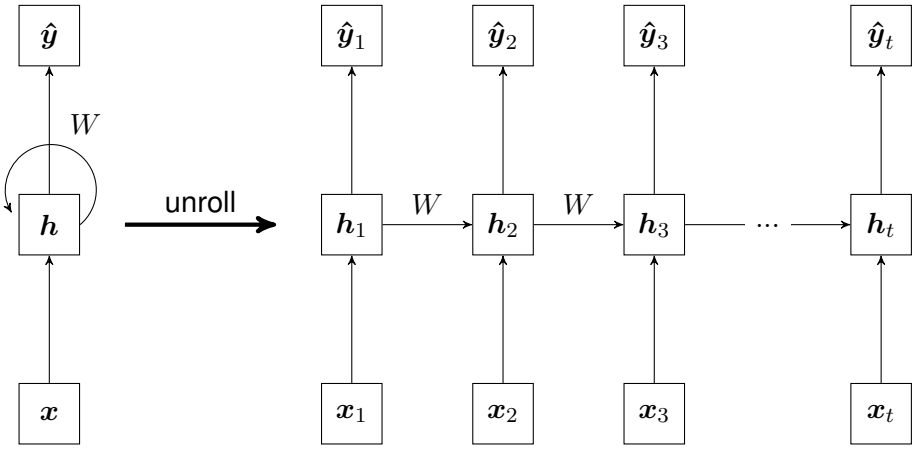


Figure 2.9: RNN architecture. Each square in the figure represents one or more neurons, and each arrow corresponds to a fully connected layer between the neurons on either side of the arrow. Left: RNN with cyclic weights. Right: The same network unrolled over time (which corresponds to a FFNN).

2.4.5 Convolutional Neural Network

A *Convolutional Neural Network* (CNN) is essentially an improved version of regular FFNNs. This means that CNNs also excel at mapping inputs to outputs, with use cases such as object recognition. In 2014 Krizhevsky *et al.* [30] won the ImageNet challenge [18] with a CNN. ImageNet is a competition where people try to get the highest classification accuracy of objects in a large database of images. After this breakthrough, CNNs have become very popular. Note that CNNs are similarly trained using the backpropagation algorithm as described in Section 2.4.3.

What allows CNNs to achieve higher accuracies is their abillity to reduce overfitting through different regularization techniques [31]. A big cause of overfitting in ANNs is having too many neurons. The more neurons you have, the more information can be learned. Consequently, when the network encounters features that are unique to the training set (considered noise), the network will be more affected by this. The reason for the improvement using a CNN lies primarily in the usage of feature maps and max pooling layers. Feature maps allow the network to share weights and thresholds between neurons connected to different parts of the input. This means that less neurons are considered, and thus generalizing each feature in the network

better, which reduces overfitting. A max pooling layer will down-sample the feature maps so that they are just an approximation of the feature. This will decrease the local noise that may appear in the training set, which also helps against overfitting. [31]

2.4.6 Support Vector Regression

Support Vector Machines were originally introduced by Vapnik *et al.* [32] as a supervised learning algorithm in the context of classification. Drucker *et al.* created a version of SVM that could also handle regression problems, named Support Vector Regression (SVR). This technique is utilized in this project.

In short, SVR functions as many other regression techniques; fitting a line to some data by minimizing a cost function. However, as historic traffic flow data is nonlinear, a simple line will not work. SVR offers a solution to this called the kernel trick which allows the algorithm to solve nonlinear regression problems. SVR has been used in the past specifically for traffic flow forecasting [23], [24].

2.5 Database Management

To manage data, a database is required to store the data. There are a number of different database models, such as relational databases that most often use SQL as the query language to operate. This project uses a non-relational database which, in contrast to relational databases, doesn't use SQL as the query language and is therefore referred to as NoSQL [33].

2.5.1 NoSQL

Instead of using tabular relations of columns and rows used in relational databases, NoSQL databases use other means to model the storage and retrieval of data. NoSQL is mainly divided into four types of models: key-value stores, document-oriented databases, wide-column stores and graph stores [33]. Document-oriented databases store data as document objects, normally in JSON (JavaScript Object Notation) format (see Figure 2.10). Each document is paired with a key and each document may consist of multiple key-value-pairs or key-array-pairs [33]. Documents can be queried and retrieved using their key, but also by their other properties [34]. This enables the ability to retrieve and update parts of a document. Some common characteristics shared among NoSQL databases is that they have high performance, are highly scalable and have low complexity [34].

2.5.2 MongoDB

This project uses the document-oriented model of MongoDB which is a document-oriented database program. The database uses collections [35], which is a grouping of documents and is essentially equivalent to tables in relational databases. MongoDB

```
{
    "road": "E6",
    "location": {
        "city": "Gothenburg",
        "country": "Sweden"
    },
    "segments": [
        { "segment": "1", "flow": "4.5" },
        { "segment": "2", "flow": "2.5" },
        { "segment": "3", "flow": "7.5" }
    ],
    "direction": "+",
}
```

Figure 2.10: Example of JSON format of a road.

uses a binary representation of JSON called BSON (Binary JSON) to store JSON documents. JSON consists of the following data types: string, number (signed decimal number), boolean, null, object (unordered set of name-value-pairs) and array [36]. BSON expands the representation of JSON to include other data types, such as int, long, date and float [37]. MongoDB provides drivers for Python (which is used in this project) among other programming languages. It also offers the possibility to store database collections in the public cloud with MongoDB Atlas [38].

2.6 Previous Work

There exists a lot of literature on time series forecasting, including in the context of road traffic predictions using machine learning. The projects mentioned below are just a small portion of the existing work on the subject.

To begin, the project that this thesis is a continuation of will be described briefly. The previous project mainly consisted of learning about general ML concepts as well as traffic forecasting. The only ML technique used was LSTM networks. These were utilized for forecasting the traffic flow of an entire road network in Gothenburg. In other words, all road segments were forecasted simultaneously. The reason for doing this was because the forecasted traffic of the entire road network was to be visualized on a map in a web application. This project, however, does not include a web application and instead focuses on forecasting the flow levels at individual road segments.

Ahmed *et al.* (2010) [39] conducted an experiment in which they compared the time series forecasting capabilities of various different ML techniques. The models used in the experiment were multilayered FFNNs, Bayesian neural networks, radial basis functions, generalized regression neural networks, K-nearest neighbor regression, CART regression trees, support vector regression and Gaussian processes. The models that gave the best results were multilayered FFNN, Gaussian processes and

support vector regression.

Gers *et al.* (2002) [40] investigated the suitability of applying LSTM networks to time series forecasting. The conclusion was that LSTM networks do have capabilities of solving certain time series tasks that FFNNs can't solve. However, when it comes to time series forecasting, it may not necessarily be the case that LSTM networks beat simpler techniques. This includes the statistical approaches such as ARIMA. They suggest that LSTM networks should only be applied when simpler traditional methods fail. Makridakis *et al.* (2018) [41] came to similar conclusions regarding other ML models such as FFNN and SVR.

Schimbinschi *et al.* (2015) [7] investigated traffic forecasting in complex urban networks using big data and machine learning. As perhaps expected, they concluded that more data results in better predictions for the ML models. They also concluded that spatial dependencies between road segments are a *better* predictor compared to temporal patterns. They mention that the accuracy could be further improved if the biggest source of invariance in the data is removed; weekends. Finally, they say that ARIMA based models have trouble forecasting based on spatiotemporal data and are unable to capture complex dynamics. This would make it a bad candidate for traffic forecasting, and a motivator for instead utilizing machine learning.

Lv *et al.* (2015) [24] investigated traffic flow prediction with big data using deep learning. They mention that statistical methods such as linear regression, HA, ARIMA and SARIMA perform well during normal traffic conditions. However, when abnormal traffic patterns appear they do not respond well (i.e. they can't capture the nonlinear nature of urban traffic patterns). They utilized a deep learning technique called stacked autoencoders, and concluded that this method was able to capture the nonlinear spatial and temporal correlations of the traffic data.

Yang *et al.* (2010) [42] investigated short-term traffic flow predictions using a FFNN while considering weather parameters as features. They concluded that predictions based on weather parameters are more accurate than those without.

Guo *et al.* (2010) [43] expectedly confirmed that historical data is less useful in forecasting traffic during abnormal conditions.

3
Method

This chapter describes how the project was executed, and Section 3.1 begins by listing the software technologies used. Now, since machine learning was utilized in this project, a large amount of relevant data was crucial. The first part therefore consisted of gathering data, and is described in Section 3.2. The next step was to analyze the data in order to find patterns or potential complications. This is presented in Section 3.4. Once a good understanding of the data had been established, an iterative process of evaluating different forecasting models could begin. This process consisted of picking a forecasting model, preparing the available data accordingly for the chosen model (Section 3.6), deciding various hyperparameters, fit the model and finally evaluate it (Section 3.7). If the results could be improved, a step back was necessary in order to update the hyperparameters and re-evaluate the model. Three different forecasting experiments were conducted and evaluated separately. These are described in Section 3.5. Implementation details of all the models and various optimizations that were performed are discussed in Appendix A.1. When all models had been evaluated, the final step was to compare the results and draw conclusions. This is done in Chapter 4 and 5, respectively. Figure 3.1 gives an overview of the work flow described above.

3.1 Software

The data analysis and implementation of various forecasting models were done with the programming language *Python* 3.7 [44]. The primary development environment used was *Jupyter Notebook* [45]. This provided a convenient and structured way of implementing the various steps in the working procedure described in the previous section. Several python libraries were utilized in different ways, and are summarized below.

- **PyMongo** [46]. Enables drivers for Python to use MongoDB. Contains tools for working with database management.

- **NumPy** [47]. Enables convenient handling of large multi-dimensional lists and matrices. Consists of several mathematical functions for easy manipulation of these data structures.

- **Pandas** [48]. Used for data analysis and statistics. It was built with the intention of implementing and improving on existing data manipulation tools

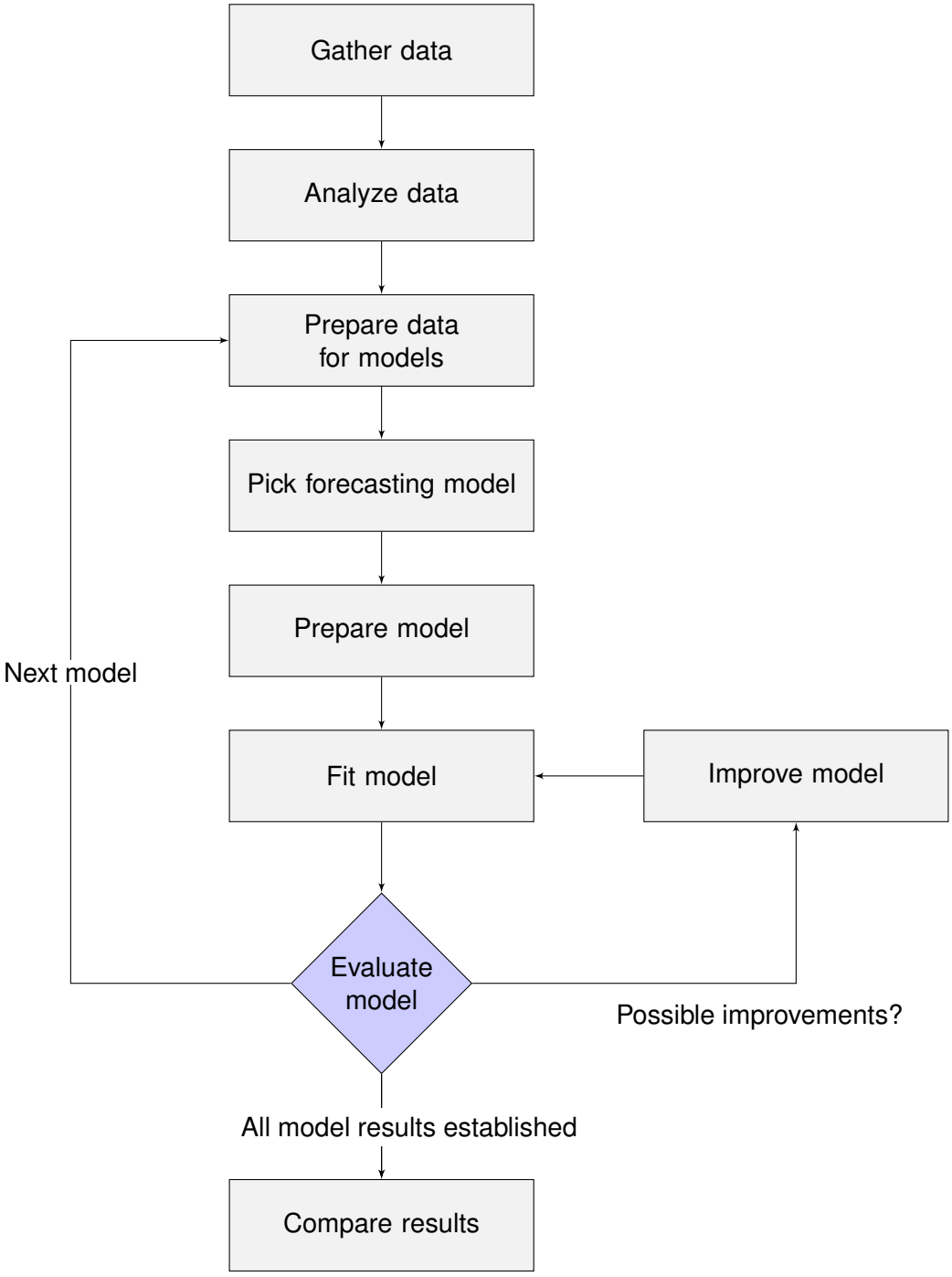


Figure 3.1: Visualization of the workflow throughout the thesis project.

found in statistical programming languages such as R. Additionally, it offers good support for manipulation of time series data which is the data format used in this project.

- **Scikit-learn** [49]. A library for manipulation of data sets and implementation of different machine learning algorithms.

- **Matplotlib** [50]. A 2D plotting library which has been used to create graphs that visualize the data in different ways.

- **TensorFlow** [51]. A library consisting of low level implementations of various machine learning algorithms, such as neural networks.

- **Keras** [52]. A library used for conveniently implementing various deep neural network architectures. It consists of an API that lies one abstraction layer above another library with low level implementations of neural networks (e.g. TensorFlow or Theano). In this project, TensorFlow was utilized as the back-end for Keras. The implementation of all neural networks in this project (FFNN, LSTM, CNN) used Keras.

3.2 Data Collection

The data needed for the project was collected using two of HERE Technologies APIs [53]. The traffic data was collected using the Traffic API [54] and the weather data was collected using the Destination Weather API [55].

All the data was stored using the cloud database MongoDB Atlas. There are different tiers to choose from when setting up a project in the cloud with different price ranges, including a free tier which this project used. The tier properties differs in network performance and storage capacity.

3.2.1 Traffic Data

The Traffic API from HERE was used to collect data of vehicle traffic in Gothenburg. This was performed with an HTTP request of traffic data within the area shown in Figure 3.2. The request returns the data in BSON format, where the relevant parameters can be saved. The parameters were separated into two different collections in the database, *Roads* and *Traffic*. The PyMongo library was used in Python to access and make insertions to the collections in the database.

The collection named *Roads* saves the static parameters of every road and its segments within the proximity. There are 245 roads and a total of 811 road segments creating 245 documents in the collection. Every road has a fixed number of segments containing information (see Figure 3.3) such as the name and an array of coordinates creating a shape of the road segment. Since these parameters are static, they only

need to be saved once. This collection was used to visualize a map of all the road segments, including a function to click on the segments to see their information (see Figure 3.4).

Figure 3.2: Proximity of the collected traffic data.

Figure 3.3: One document in the collection *Roads*.

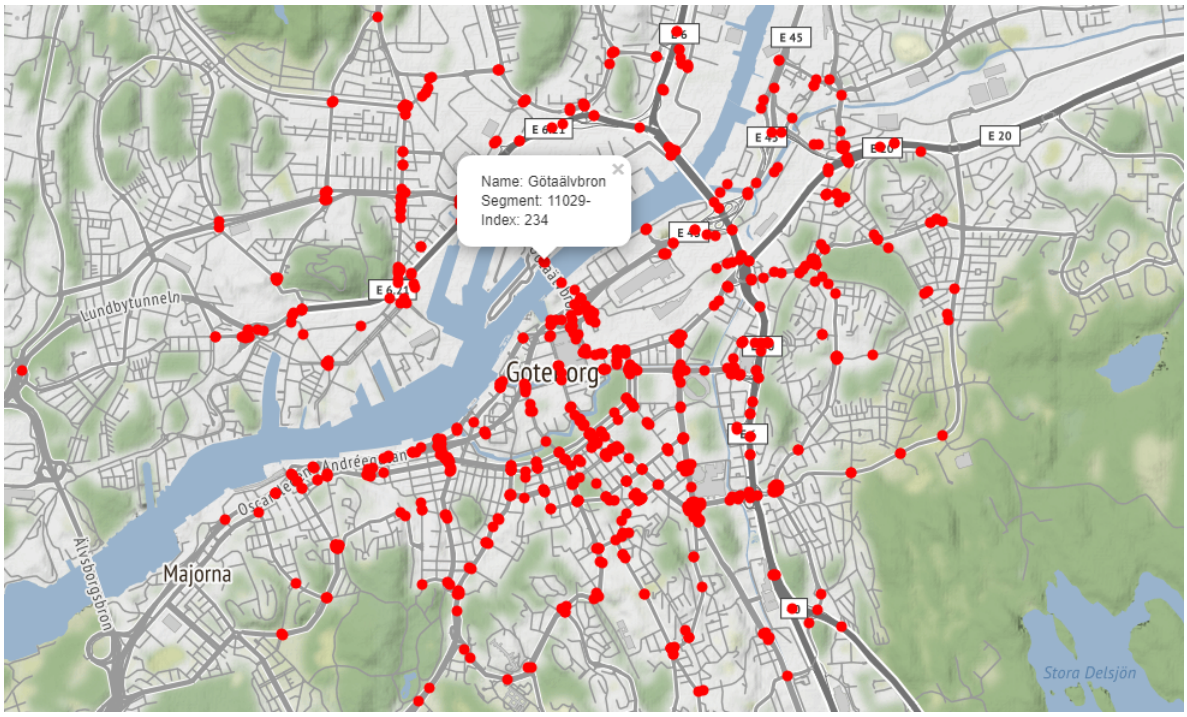


Figure 3.4: Map of all the road segments and the information of one segment.

The collection named *Traffic* saves the dynamic parameters of every road segment, where each of the 811 segments consists of two arrays (see Figure 3.5) creating 811 documents in the collection. The first array saves the jam factor (level of traffic congestion) for the segment which is a value between 0.0 and 10.0, where 0.0 indicates no traffic congestion and 10.0 indicates full traffic congestion. The second array saves the average vehicle speed in kilometers per hour. These two parameters were collected and appended to the arrays of each segment every 10 minutes.

To create a time series of the data from the *Traffic* collection, a third collection named *Timestamps* was created that saves the date and time every time data is collected (one time step every 10 minutes).

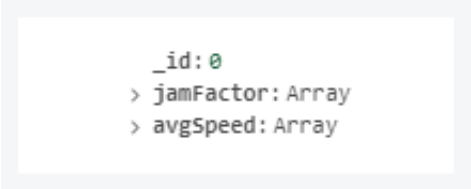


Figure 3.5: One document in the collection *Traffic*.

3.2.2 Weather Data

The Destination Weather API was used to collect weather data in Gothenburg. As with the traffic data, this was performed with an HTTP request. However, instead of using a proximity, the request was done by specifying a latitude and longitude of central Gothenburg. The assumption made was that the weather condition of

central Gothenburg is the same as in the proximity of the traffic data.

The parameters selected were: sky description, wind speed, temperature and precipitation (see Figure 3.6). The parameter for the sky description called *skyInfo* is an integer in the range of 1 to 34 representing a description of the sky. For instance, 1 indicates that the sky is sunny and 17 indicates that the sky is cloudy. Wind speed is saved as a value in kilometers per hour. Temperature is saved in degrees Celsius. Precipitation is an integer in the range of 0 to 77 representing a description of the precipitation. For instance, 0 indicates that there is no precipitation and 9 indicates that it is raining. Both the sky description and the precipitation are discrete enumerations. A higher value does not necessarily mean more clouds/rain.

The weather data is collected every time there is a new weather observation from the HTTP request, indicated by the date and time. Weather observations most often occur once every 30 or 60 minutes. To match the time series of the traffic data, the weather data was extended. This means that if there were 30 minutes between two weather observations, the first weather observation is duplicated twice to match the time series of one time step every 10 minutes. The assumption here is that the weather stays the same between two given weather observations.

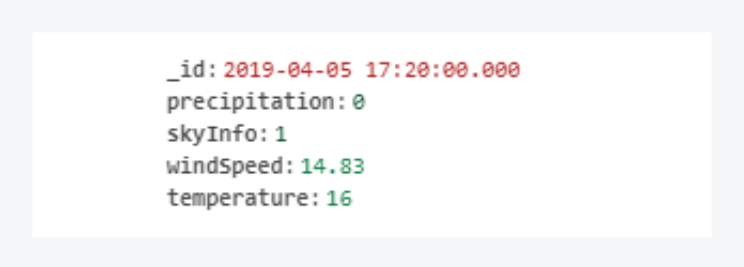


Figure 3.6: One document in the collection *Weather*.

3.3 Data Format

The time series of data that was gathered had 9202 time steps when it was used by the models to forecast. The time interval between each time step is 10 minutes, and thus correspond to roughly 9 weeks of data. As explained in the previous section, each time step contains four weather parameters, and one measurement of the traffic flow and average speed of 811 road segments. In total, each time step contains a feature vector of $4 + 811 + 811 = 1626$ features. The time series therefore has the dimensions 9202×1626, and can be described as shown in (3.1).

$$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{9202}\}, \quad \boldsymbol{x}_t \in \mathbb{R}^{1626} \tag{3.1}$$

The data can be retrieved from the database and formatted into a Pandas DataFrame object with the code shown below.

```
>> X = getDataFrame() # Custom function
```

```
>> print(X.shape())
(9202, 1626)
```

3.4 Data Analysis

This section aims to visualize the various data features in different ways to identify various patterns. Figure 3.7 shows the traffic flow of segment 58 (Falutorget) for a one week period (2019-03-11 to 2019-03-17). Each day corresponds to 144 time steps, i.e. 24 hours. The top plot shows the raw data, which is evidently quite noisy. The bottom plot is the exact same data, except some of the noise has been removed by using a rolling window of size 10. This makes it easier to see the temporal patterns throughout the time series. It also allows the forecasting models to find the patterns much more effectively. The first five days (Monday through Friday) seem to be following the expected traffic pattern as explained in Section 2.1. Saturday and Sunday, however, expectedly do not follow the same traffic pattern. This momentarily breaks the seasonality made up of the first five days, which immediately makes forecasting harder.

Figure 3.8 displays the traffic flow of the same period for segment 6 (Eriksbergsmotet), 31 (Surbrunnsgatan) and 355 (Ånäsmotet). It seems that segment 6 and 355 follow a similar pattern (except segment 6 has low weekend traffic). On the other hand, the traffic of segment 31 seems to be relatively random, and is therefore significantly harder to forecast. Now, while there are some segments like this with no clear seasonality, the average traffic flow for all 811 road segments does show the expected pattern (see the top plot of Figure 3.7). Therefore, it is likely that most of the 811 segments can be forecasted with good accuracy. Additionally, if weekends are removed from the time series, the forecasting accuracy would likely rise dramatically.

Figure 3.9 (bottom) shows the average traffic speed of all 811 segments. It seems that the traffic speed changes in the opposite direction of the jam factor. This makes sense since a lower jam factor (i.e. less traffic congestion), the faster the cars will be able to go. The traffic speed is therefore inversely correlated with the jam factor. Using the traffic speed as basis for predicting the traffic flow may therefore be a good idea.

Finally, Figure 3.10 plots the traffic flow of segment 58 for the last 20% of the time series. This part of the data will be used as the test set for evaluating the trained models. The top plot shows the original data. The fifth day, which is a Wednesday, seems to almost be following the traffic pattern that is seen during the weekends. The reason for this is because this Wednesday corresponds to the first of May, which happens to be a holiday in Sweden. Since the model does not encounter this pattern in the training set, it will be difficult to accurately predict the test set. The training set is therefore not informative enough to forecast the test set. Also, the training set includes a few Easter holidays which the test set does not. This may additionally lead to overfitting. The bottom plot of Figure 3.8 displays the test set with holidays

removed. Note that the first week now instead consists of only four weekdays, which may also lead to various issues. During the experiments, the forecasting models will be tried both with the holidays included and when they are removed, for comparison.

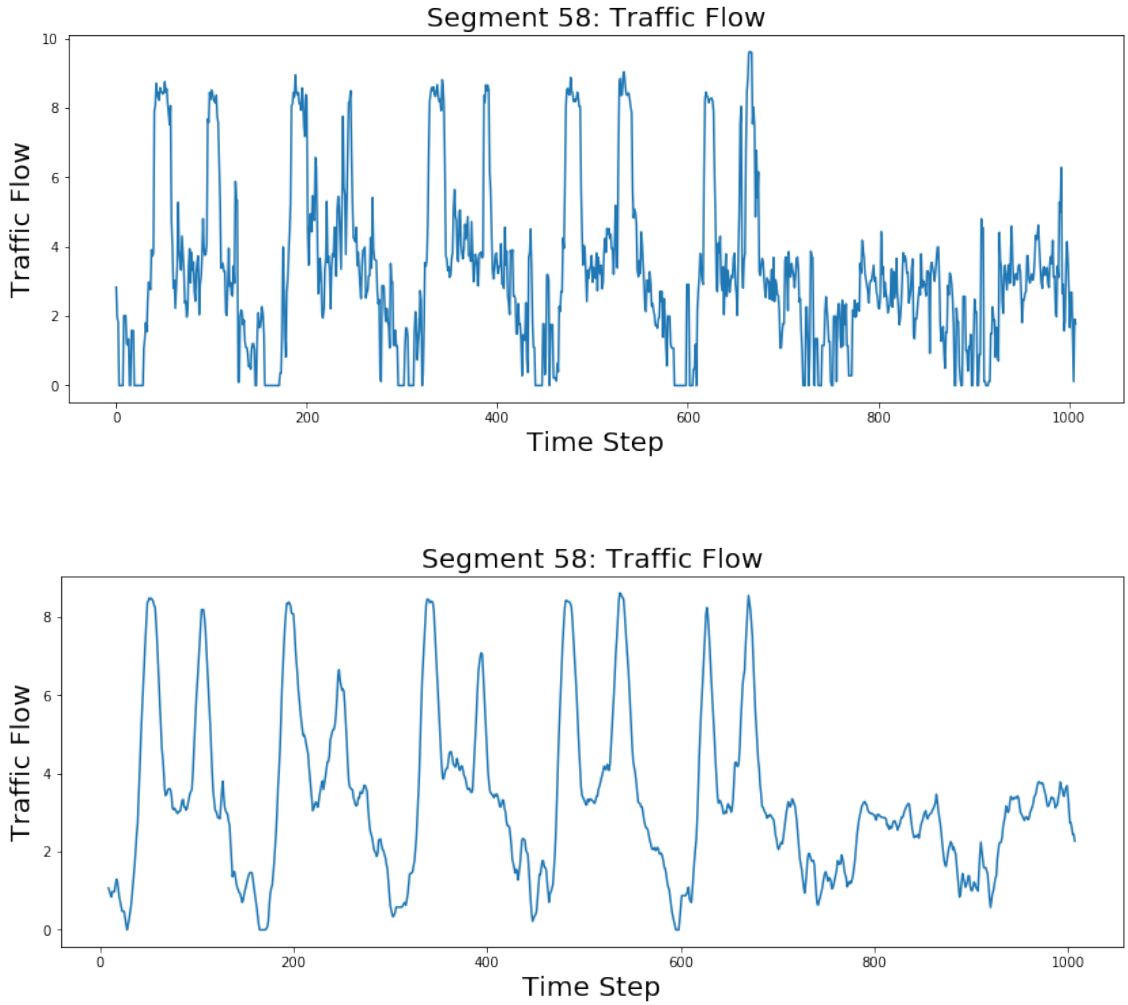


Figure 3.7: One week of traffic flow for segment 58 between 2019-03-11 and 2019-03-17. **Top**: Raw data. **Bottom**: Same data with some noise removed.

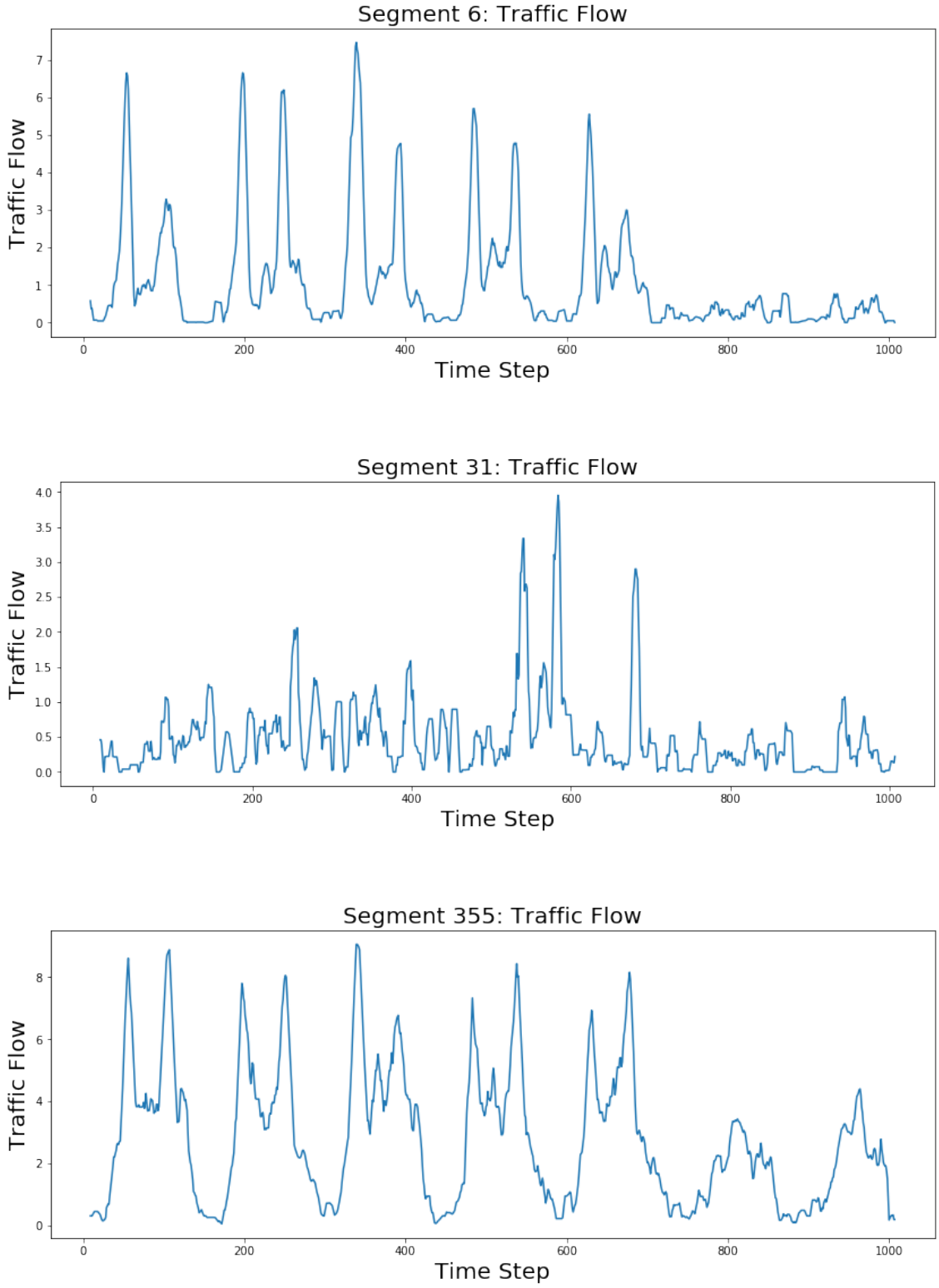


Figure 3.8: One week of traffic flow for segments 6, 31, 355 between 2019-03-11 and 2019-03-17.

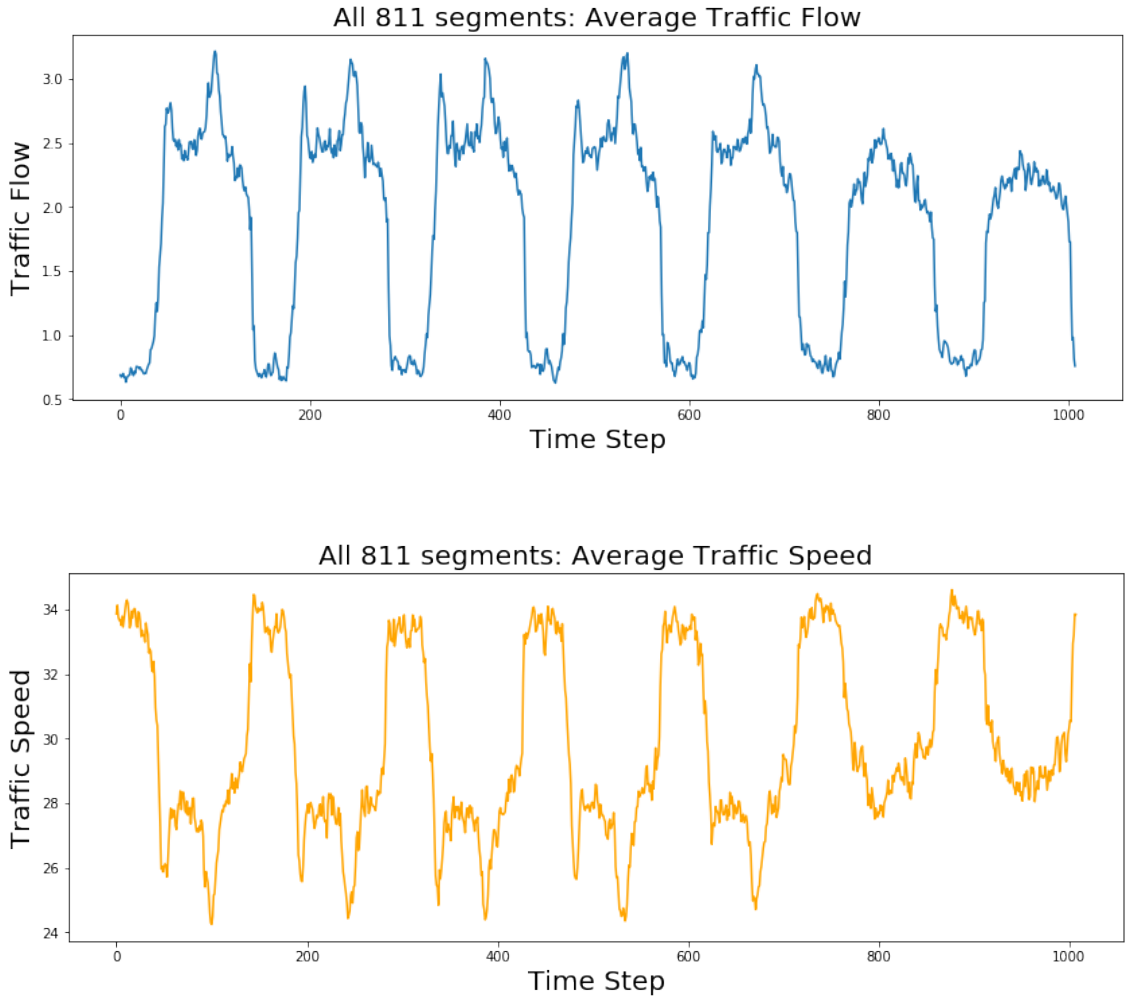


Figure 3.9: One week of average traffic flow (**top**) and average speed (**bottom**) from all 811 road segments between 2019-03-11 and 2019-03-17.

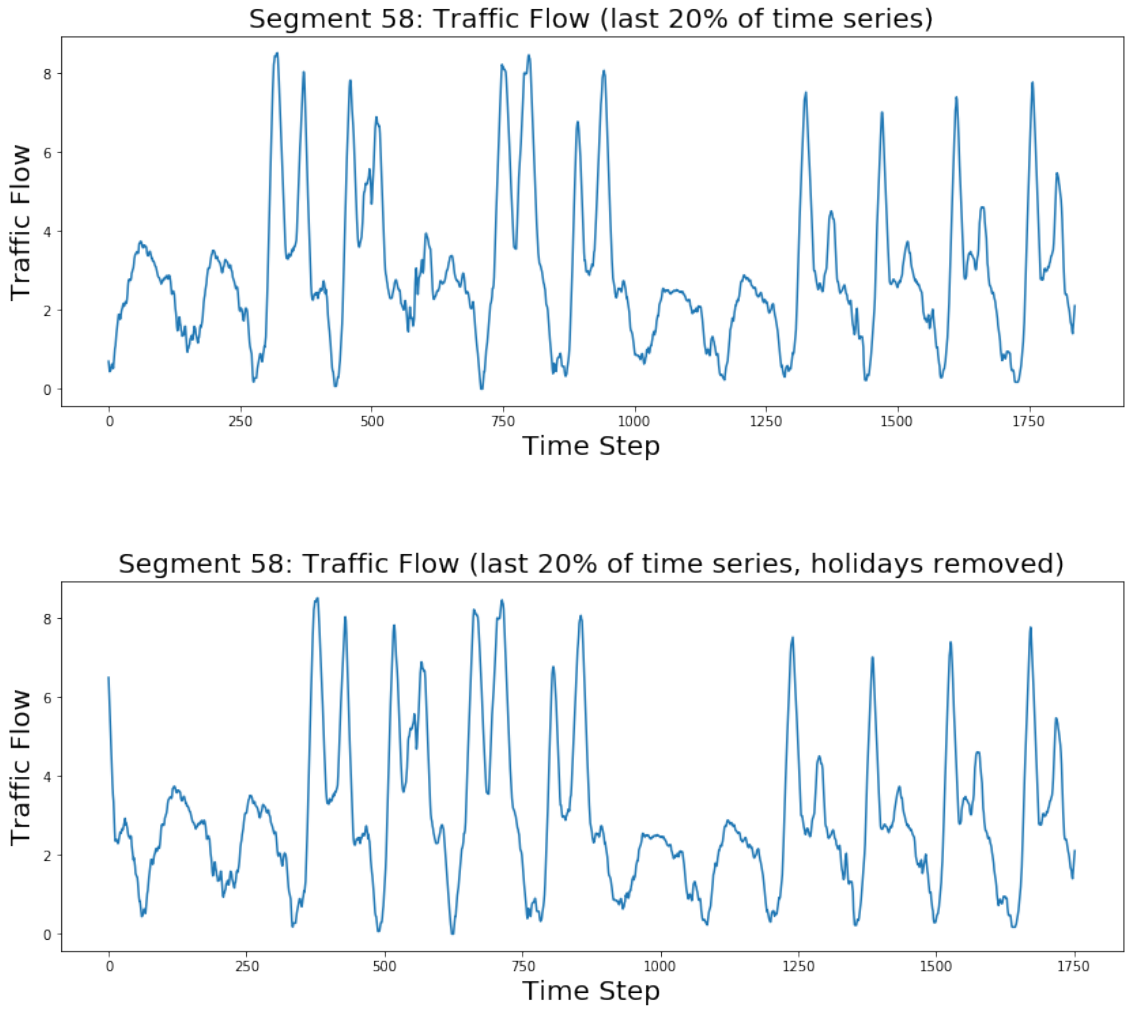


Figure 3.10: Top: The traffic flow of segment 58 for the last 20% of the time series (test set). **Bottom:** Equivalent plot but with holidays removed from entire time series (including 2019-05-1 in test set as can be seen).

3.5 Experiments

Three different experiments were conducted in this thesis and are explained below. For each forecasting experiment the same horizons were used. Section 3.5.1 begins by presenting the chosen forecasting horizons.

3.5.1 Forecasting Horizon

The forecasting horizons chosen are 10 minutes, 1 hour, 6 hours, 12 hours and 24 hours. This gives a good idea of how well the models can produce short-term, medium-term as well as long-term forecasts. The 10 minutes and 1 hour are considered short-term, 6 hours medium-term, and 12 and 24 hours long-term. All of which may be of interest when used in applications that utilize traffic forecasting.

3.5.2 Experiment (1): Univariate input forecast

The first experiment consists of forecasting the traffic flow of one segment. The forecast will be based only on historic traffic flow data from the same segment that is being predicted. The segment to be forecasted was arbitrarily chosen to be segment 58. The experiment will be tried on both the original dataset, and with the holidays removed.

3.5.3 Experiment (2): Univariate input forecast (without holidays)

The second experiment will be equivalent to experiment (1) except that all holidays will be removed from the data set.

3.5.4 Experiment (3): Multivariate input forecast (without holidays)

The third experiment also consists of forecasting the traffic flow of segment 58. However, the forecast will this time be based not just on the historic traffic flow of segment 58, but also other correlated features. This may include the traffic speed of segment 58, and/or traffic parameters of other nearby road segments. This means that spatial correlations are considered. Section 3.6.1 presents which features will be used. Note that holidays were removed for this experiment as well.

3.5.5 Expectations

The main expectation for all experiments is that the machine learning models will outperform the statistical approaches. In particular, the LSTM network is expected to provide the superior results as it is well suited for time sequence problems. In experiment (3) it is expected that the forecasting accuracy improves quite drastically when holidays are removed from the data set. Furthermore, it is highly plausible that experiment (2) will further improve the results when considering spatial correlations.

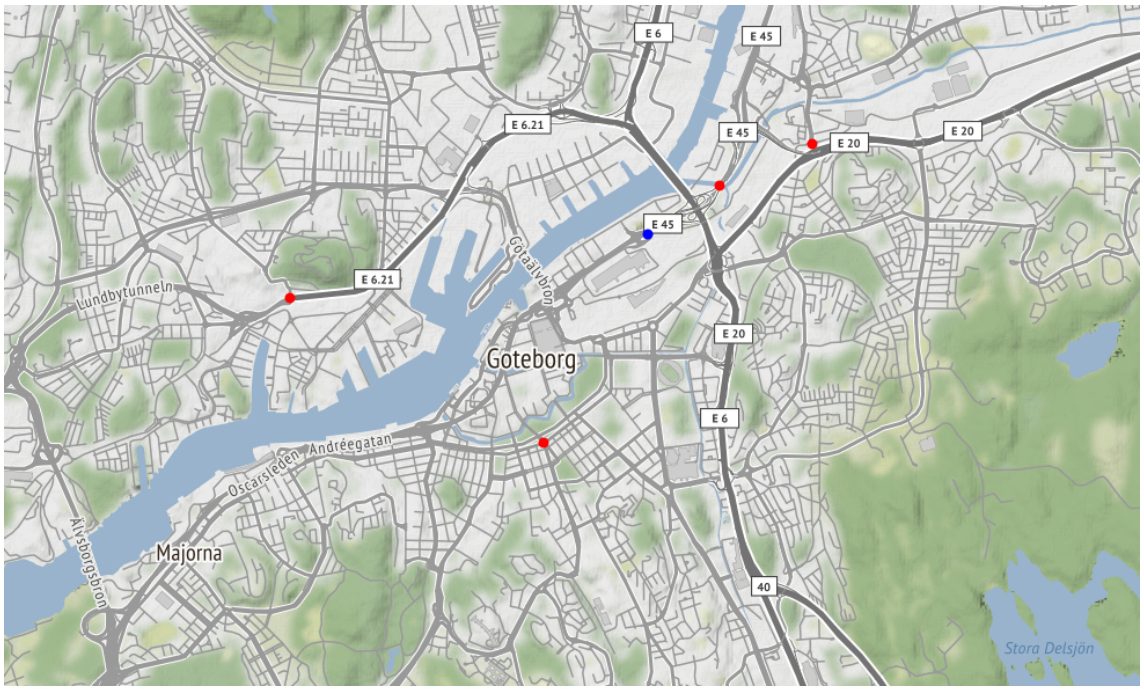


Figure 3.11: Map of segments. The blue dot corresponds to segment 58. The red dots correspond to segments 6, 189, 355 and 761. The traffic flow at the red dots are highly correlated with the traffic flow at the blue dot.

3.6 Data Preparation

This section walks through the various steps of preparing the data. This included feature engineering, training and test set split of the data, adapting the time series to a supervised learning context, and finally scaling the data.

3.6.1 Feature Engineering

Feature engineering was only necessary for the second experiment. This was achieved by evaluating the correlation with segment 58 and all other features, and filtering out the features that had a correlation over some threshold. This is visualized in the code below. The print statement shows some of the features that have a correlation of $abs(\pm\ 0.85)$ or higher. This resulted in 9 features out of the total 1626 that be will used as basis for forecasting the traffic flow at segment 58. As can be seen, the traffic speed of segment 58 is expectedly very (inversely) correlated with the flow. The segments with correlated traffic parameters to segment 58 was 6, 189 (Parkgatan), 355 and 761 (Gullbergsmotet). These are the segments shown in Figure 3.11 (red dots). Segment 58 is the blue dot in the same figure. Also, Figure 3.12 shows the correlations in a correlation matrix. Only a subset of the 9 features are shown in the matrix.

```
>> threshold = 0.85
>> X = X.iloc[:,1:] # Remove timestamp
>> segment_58 = X["flow_58"]
```

```
>> corr = X.corrwith(segment_58, method="spearman").abs()
>> correlated_features = corr[corr > threshold].index
>> X = X[correlated_features]
>> print(corr[corr > threshold])
flow_58        1.000000
flow_761       0.998387
speed_761     -0.993167
speed_58      -0.990854
flow_355       0.874526
speed_6       -0.855887
flow_6         0.855590
speed_355     -0.854793
flow_189       0.852866

>> print(X.shape)
(9202, 9)
```

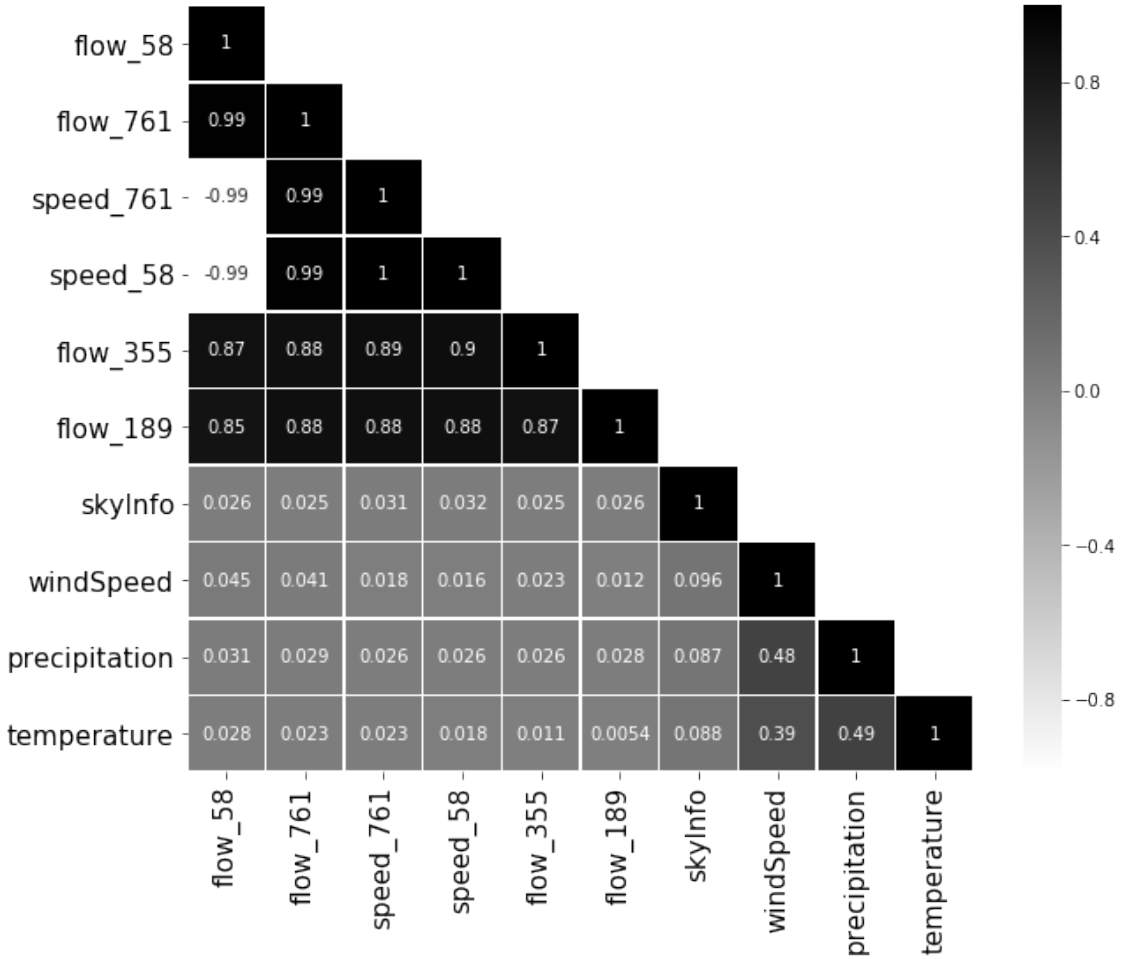


Figure 3.12: Correlation matrix between some of the features used in experiment (3). The weather parameters were not used, but are shown in the graph to illustrate their low correlation.

Note that the correlation with the various weather parameters was extremely low, as seen below. This is also illustrated in the correlation matrix in Figure 3.12.

```
skyInfo          0.026189
windSpeed        0.045012
precipitation    0.031231
temperature      0.028119
```

This may be due to the fact that the weather has remained relatively unchanged during the period that all the data was gathered. These features are therefore not used for forecasting.

3.6.2 Data Split

The time series was split into a training set (80%) and a test set (20%) as shown below.

```
>> split_ratio = 0.2
>> X = X.values # Convert to NumPy array
>> split = int(len(X) * (1-split_ratio))
>> training_set = X[:split]
>> test_set = X[split:]
>> print(training_set.shape(), test_set.shape())
(7350, 9)
(1838, 9)
```

3.6.3 Time Series Forecasting as a Supervised Problem

The next step is to format the data such that supervised learning can be applied for the machine learning models. This is done in accordance with what was explained in Section 2.3.5. Note that both the training set and test set will undergo the same transformation. The time series $\boldsymbol{X}$ in (3.2) visualizes what the training set and test set currently look like.

$$\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T\} \tag{3.2}$$

Now, they will both be transformed into $\boldsymbol{X'}$ as shown in (3.3).

$$\boldsymbol{X'} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \\ \boldsymbol{x}_2 & \boldsymbol{x}_3 & \cdots & \boldsymbol{x}_{n+1} \\ \boldsymbol{x}_3 & \boldsymbol{x}_4 & \cdots & \boldsymbol{x}_{n+2} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_{T-n-h+1} & \boldsymbol{x}_{T-n-h+2} & \cdots & \boldsymbol{x}_{T-h} \end{bmatrix} \tag{3.3}$$

Additionally, the target outputs $\boldsymbol{Y}$ will be created as shown in (3.3).

$$\mathbf{Y} = \{\boldsymbol{y}_{n+h}, \boldsymbol{y}_{(n+1)+h}, \boldsymbol{y}_{(n+2)+h}, \cdots, \boldsymbol{y}_T\} \tag{3.4}$$

Note that each row of $\boldsymbol{X'}$ corresponds to the inputs, and are mapped to each element in $\boldsymbol{Y}$. The number of rows in $\boldsymbol{X'}$ therefore matches exactly with the number of elements in $\boldsymbol{Y}$. The motivation for organizing the data this way is because the

machine learning models expect the dimensions to be (#samples, #timesteps, #features). $\boldsymbol{X'}$ follows that rule with dimensions (T-n-h+1, n, q_1). The dimensions of $\boldsymbol{Y}$ are (T-n-h+1, q_2). q_1 decides how many features are used as a basis for forecasting. q_2 decides how many features are being forecasted.

In experiments (1) and (2), $q_1 = 1$ and $q_2 = 1$ since the traffic flow of one segment is forecasted based on the traffic flow from the same segment. In experiment (3) $q_1 = 9$ and $q_2 = 1$ since the traffic flow of one segment is being forecasted based on 9 features.

The function **timeseriesToSupervised** achieves the desired transformation and is shown below.

```
def timeseriesToSupervised(data, n, h):
    x, y = list(), list()
    for i in range(len(data)-n-h+1):
        x.append(data[i:(i+n)])
        y.append(data[i+h+n-1])
    return np.array(x), np.array(y)
```

The parameters h and n are first chosen, then the transformation is done to both the training set and test set as shown below.

```
>> h = 1
>> n = 12
>> trainX, trainY = timeseriesToSupervised(training_set, n, h)
>> testX, testY = timeseriesToSupervised(test_set, n, h)
>> print("trainX: ", trainX.shape)
>> print("trainY: ", trainY.shape)
>> print("testX: ", testX.shape)
>> print("testY: ", testY.shape)
trainX:  (7338, 12, 9)
trainY:  (7338, 9)
testX:  (1826, 12, 9)
testY:  (1826, 9)
```

Note that *trainY* and *testY* must be modified such that the correct feature is being predicted, as it by default contains all 9 features. For all experiments, one segment is being predicted. The desired transformation is therefore done as shown below, where only one feature is selected as output. In this case the feature of interest (traffic flow of segment 58) is assumed to be at index 0.

```
>> import numpy as np
>> testY = np.reshape(testY[:,0], (testY[:,0].shape[0],1))
>> trainY = np.reshape(trainY[:,0], (trainY[:,0].shape[0],1))
>> print("trainY: ", trainY.shape)
>> print("testY: ", testY.shape)
trainY:  (7338, 1)
testY:  (1826, 1)
```

trainX and *trainY* will now be used during the learning phase. *testX* and *testY*

will be used to make forecasts on new data, such that the model performance can be evaluated.

3.6.4 Data Scaling

The ML models require the data to lie within the range (0, 1). The code below achieved this by scaling it accordingly.

```
from sklearn.preprocessing import MinMaxScaler

# Scaling 3D data requires some extra work:
scalers = {}
for i in range(trainX.shape[2]):
    scalers[i] = MinMaxScaler()
    trainX[:, :, i] = scalers[i].fit_transform(trainX[:, :, i])

for i in range(testX.shape[2]):
    testX[:, :, i] = scalers[i].transform(testX[:, :, i])


# The target values are 2D arrays, which is easy to scale:
scalerY = MinMaxScaler()

trainY = scalerY.fit_transform(trainY)
testY = scalerY.transform(testY)
```

3.7 Model Evaluation

As already mentioned, the data was scaled to the range (0, 1) for all ML models. For these models, the data must now be descaled using the same scaler that was originally used to scale the data (named *scalerY* below).

```
# Descale
predictions = scalerY.inverse_transform(predictions)
testY = scalerY.inverse_transform(testY)
```

Next, the predictions can be plotted against the real values to visualize the accuracy with graphs.

```
import matplotlib.pyplot as plt
start = 0
end = 2000
plt.plot(predictions[start:end], label="Test set predictions")
plt.plot(testY[start:end], label="Real data")
plt.legend()
plt.ylabel('traffic flow')
plt.xlabel('timestep')
plt.show()
```

And finally, the results can be presented using the accuracy metric MAE. The code for this is shown below.

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(predictions, testY)
print("Test MAE: %.6f" % mae)
```

4
Results

This chapter presents the results of the various experiments conducted. The forecasting accuracy of each model for each experiment is presented in tables, with MAE as the accuracy metric. For the ML models, the average MAE over 10 test runs are shown. Table 4.1 shows the results of experiment (1), which used the entire time series, including holidays. Table 4.2 shows the results of experiment (2), where holidays were removed from the time series. Table 4.3 shows the results of experiment (3), where spatial correlations were taken into account.

Out of the three experiments, the best results for each horizon are plotted in the figures below. In each figure, the forecasted traffic flow is plotted against the actual measurerd traffic flow for comparison. Figure 4.1 and 4.2 plots the results for the 10 minute and 1 hour horizons, where SVR was the superior model. Figure 4.3 and 4.4 plots the results for the 6 and 12 hour horizons, where FFNN was the superior model. Figure 4.5 plots the results for the 24 hour horizon, where CNN was the superior model. Figure 4.5 also includes the results using a CNN network with holidays included in the time series. This visualizes the problem of not removing the holidays.

MAE values in bold correspond to the model that gave the best result for a given horizon in each experiment. Underlined MAE values correspond to the model that gave the best result for a given horizon across all experiments.

	10 min (h = 1)	1 hour (h = 6)	6 hours (h = 36)	12 hours (h = 72)	24 hours (h = 144)
NM	0.1248	0.6540	1.9888	2.5473	1.0968
SNM	0.9456	0.9456	0.9456	0.9456	0.9456
SHA	0.8463	0.8463	0.8463	**0.8463**	**0.8463**
LSTM	0.0542	0.3019	0.9446	0.9229	0.9872
FFNN	0.0564	0.3424	**0.7752**	0.8494	0.9155
CNN	0.0640	0.3279	0.7867	0.9293	0.9791
SVR	**0.0462**	**0.2884**	0.8540	0.9666	1.0171
ARIMA	0.1620	0.5527	1.4659	2.6428	1.5584

Table 4.1: Forecasting results from **experiment (1)**.

	10 min (h = 1)	1 hour (h = 6)	6 hours (h = 36)	12 hours (h = 72)	24 hours (h = 144)
NM	0.1284	0.6809	2.0759	2.6492	0.9245
SNM	0.8953	0.8953	0.8953	0.8953	0.8953
SHA	0.7923	0.7923	0.7923	0.7923	0.7923
LSTM	0.0521	0.2971	0.9419	0.7337	0.7743
FFNN	0.0559	0.3478	0.6959	**0.6850**	**0.6925**
CNN	0.0598	0.3219	**0.6754**	0.8272	0.7500
SVR	**0.0472**	**0.2893**	0.7500	0.8611	0.9749
ARIMA	0.1651	0.5548	1.5634	2.8718	1.3869

Table 4.2: Forecasting results from **experiment (2)**. Holidays were removed from the time series.

	10 min (h = 1)	1 hour (h = 6)	6 hours (h = 36)	12 hours (h = 72)	24 hours (h = 144)
NM*	0.1284	0.6809	2.0759	2.6492	0.9245
SNM*	0.8953	0.8953	0.8953	0.8953	0.8953
SHA*	0.7923	0.7923	0.7923	0.7923	0.7923
LSTM	**0.0563**	0.3475	0.7584	0.9259	0.8784
FFNN	0.0591	0.3219	**0.6021**	**0.6249**	0.6503
CNN	0.0622	**0.3022**	0.6192	0.6313	**0.5913**
SVR	0.0587	0.3747	0.9334	1.0523	1.0135
ARIMA*	0.1651	0.5548	1.5634	2.8718	1.3869

Table 4.3: Forecasting results from **experiment (3)**. Spatial correlations were taken into account. Models with a (*) were not evaluated for multivariate inputs, and the best results from previous experiments are instead used for comparison.

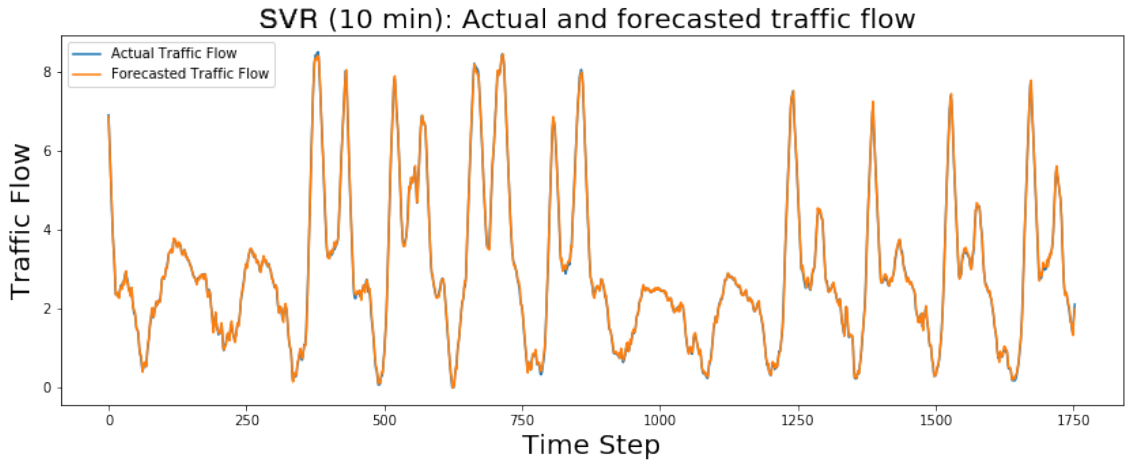


Figure 4.1: SVR 10 minute traffic flow forecasts plotted against the actual measured traffic flow.

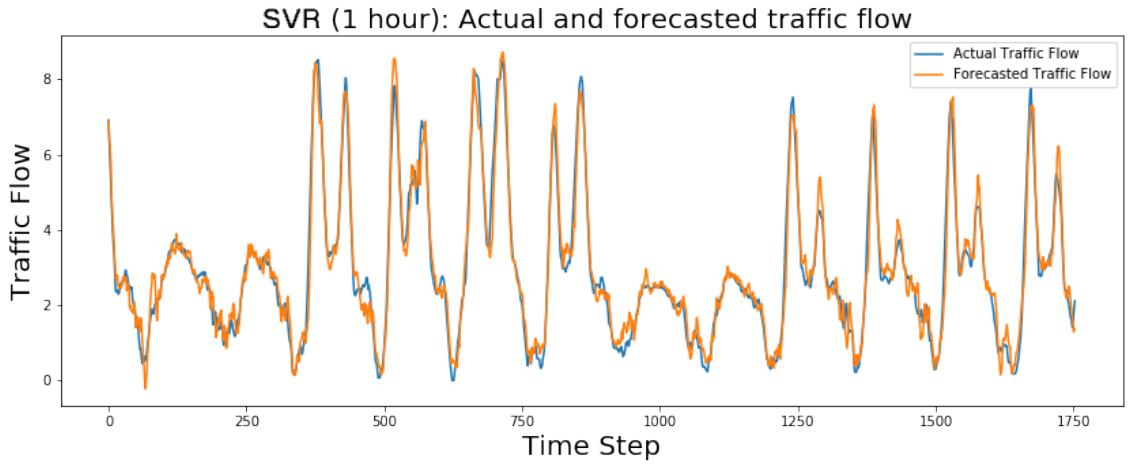


Figure 4.2: SVR 1 hour minute traffic flow forecasts plotted against the actual measured traffic flow.

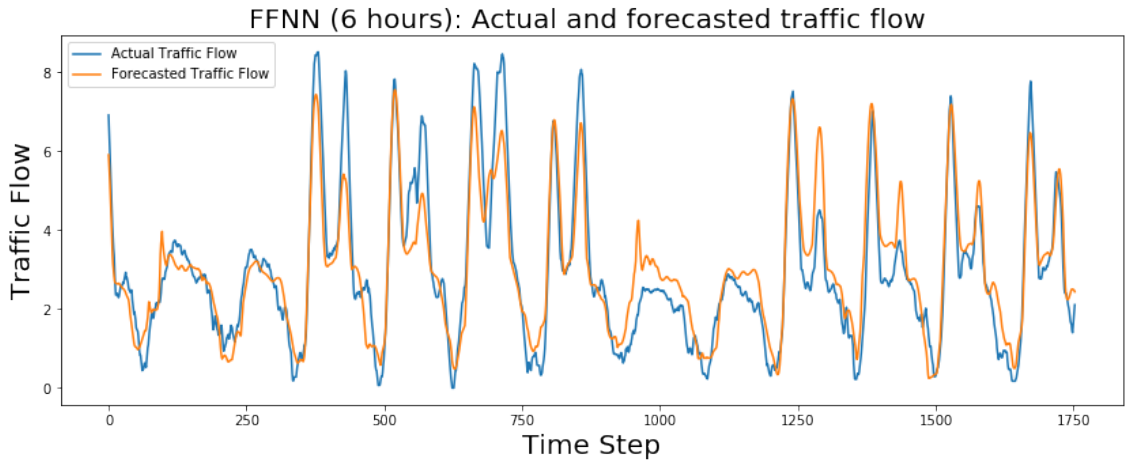


Figure 4.3: FFNN 6 hour traffic flow forecasts plotted against the actual measured traffic flow.

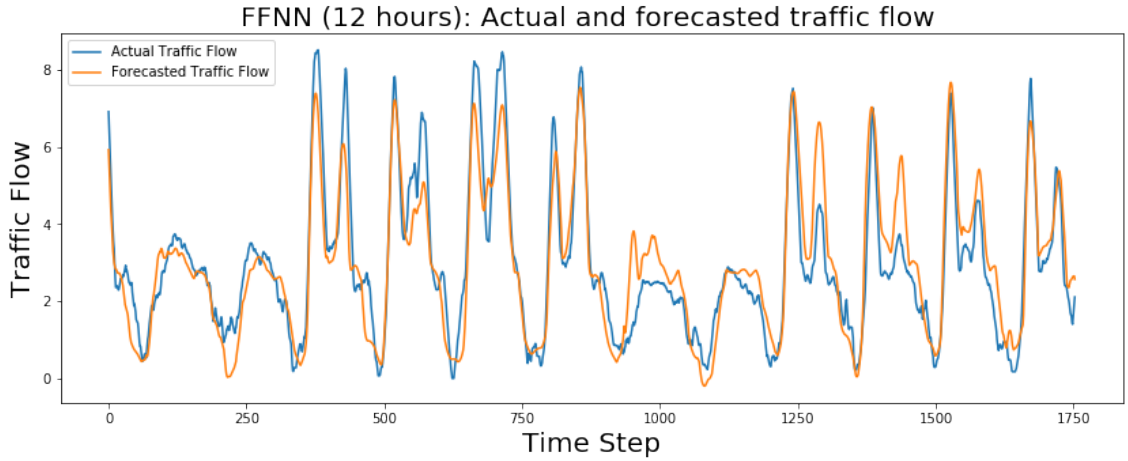


Figure 4.4: FFNN 12 hour traffic flow forecasts plotted against the actual measured traffic flow.

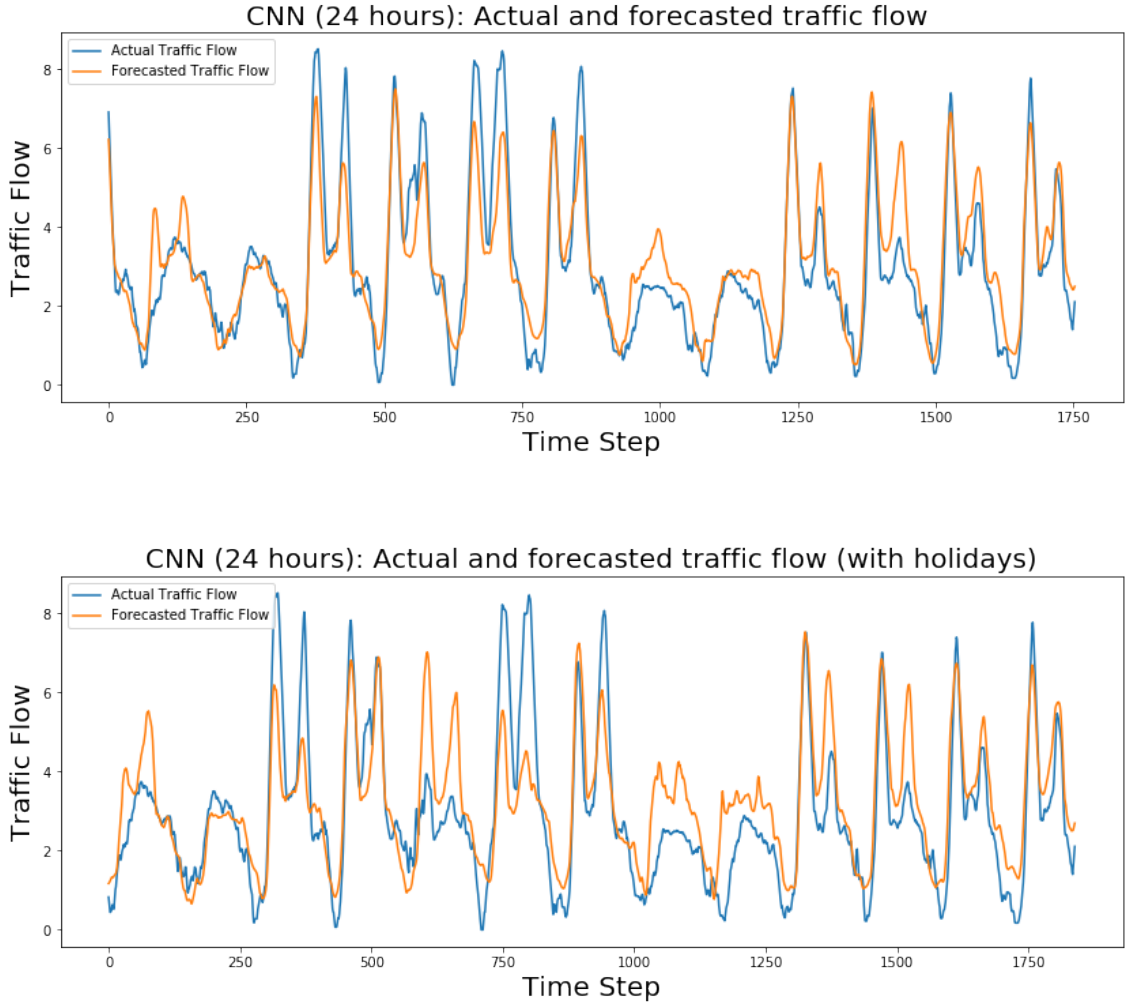


Figure 4.5: CNN 24 hour traffic flow forecasts plotted against the actual measured traffic flow. Top: Original time series used. Bottom: Holidays removed from time series.

5
Discussion and Conclusion

This chapter discusses the conducted experiments and brings up how well it met the expectations. Various improvements to further increase the forecasting accuracy will subsequently be discussed. Finally, Section 5.5 will conclude the thesis.

5.1 Experiments

Out of all the baseline methods, the Naive Method (NM) expectedly gets the best accuracy for the short-term forecasts. The traffic flow in the short-term future is most of the time quite similar to the current traffic flow. Since NM simply sets the forecast equal to the current time step, it consequently yields decent results. It also performs relatively well for 24 hour forecasts. The reason being that the traffic exactly 24 hours from now is likely to be similar to the current traffic. However, for medium-term forecasts NM gets the worst results out of all models. For medium and long-term forecasts, NM is outperformed by both SNM and SHA. This will likely always be the case for a time series with some seasonality. Out of the baseline methods, SHA performs the best. The ARIMA model does beat some of the baseline methods for short-term forecasts, but is significantly worse for medium and long-term forecasts. This shows that the ARIMA model is unable to learn the long-term horizons as they follow a more nonlinear relationship.

The chosen hyperparameters and input features used in the first experiment did not allow the ML models to beat SHA (and even SNM in some cases) for long-term forecasts. The ML models do however beat the baseline methods for short-term forecasts quite convincingly. For the second experiment with the holidays removed, the ML models gave better results than the baselines for the most part. In the third experiment, similar conclusions can be drawn with slightly improved results for the ML models. This indicates that a lot of experimentation is required in order to get the ML models to perform well and beat the baselines. In other words, more advanced feature engineering and hyperparameter optimization would likely make the ML models convincingly superior for all horizons.

Overall, SVR was the superior model for short-term forecasting and was achieved in the first experiment. Figure 4.1 and 4.2 plot these predictions, and they are evidently very accurate. The 10 minute forecast is almost identical to the actual traffic flow. This was of course expected since the traffic flow 10 minutes into the future

is very similar to the current traffic flow. All models consequently perform the best for this horizon, with the 1 hour horizon in second place. The FFNN model was superior for the 6 and 12 hour forecasting horizons, and was achieved in the third experiment. These are plotted in Figure 4.3 and 4.4. Finally, the CNN model gave the best results for the 24 hour forecasting horizon, and was achieved in the third experiment. This is plotted in Figure 4.5. Something interesting to note from these plots is how the models are able to predict the traffic flow for Saturdays. Remember that the look-back window was 1-2 days for medium and long-term forecasts. This means that the model looks at the flow from Thursday and Friday, and is able to realize that the traffic flow pattern of the next day follows that of a weekend. Even though it is not clearly visible by just looking at the graph, the models are able to recognize that the traffic flow on Fridays somehow differ from the other weekdays. From this information it is able to realize that a Saturday is coming up.

Something to note regarding SVR for experiment (3) is that it produces worse results than it did for the other two experiments. This means that the SVR model used in this project did not benefit from the extra information provided by correlated road segments. A similar conclusion could be drawn for the LSTM network. The other two ML models (FFNN and CNN), however, do display improved results in the third experiment. This would indicate that the spatial correlations in fact are of interest, but that the LSTM and SVR models used simply could not deal with the larger amounts of data.

The most unexpected outcome of the experiments was the disappointing results of the LSTM network. It seems that the other ML models were superior for all horizons. Upon inspection, it was realized that LSTM networks may not necessarily be the best option for time series forecasting. This was, as mentioned in Chapter 2, concluded by Gers *et al.* (2002) [40]. This does not mean that LSTM networks should be excluded entirely as an option. Other sources have mentioned that they have great potential if used right. For example, one could attempt to train the LSTM network with stationary data as mentioned by Brownlee [56]. He also suggests that more complex LSTM networks are required to properly learn the temporal and spatial dependencies. E.g. one could try many more stacked layers, more neurons, and subsequently train the model for thousands of epochs. This will however make the training phase extremely slow. As the hardware used in this project was very limited (Intel Core i7 3770k 3.5GHz CPU), this was not an option.

Figure 4.5 visualizes the 24 hour forecasts of the CNN model with holidays removed (top) and holidays included (bottom). This clearly illustrates the problem with keeping the holidays in the time series. It can be seen that the fifth day (i.e. first of May, which is a holiday in Sweden), follows an abnormal pattern. The model assumes that it follows the regular pattern for Wednesdays and predicts according to that (see orange line). The top plot shows the forecasts with the same model, but with holidays removed. As can be seen, this did indeed solve the problem. Another way of solving this may be to utilize a lot more data, such that the training set can learn all holidays throughout the year. The better solution, however, is probably to

keep separate models for holidays and normal week days. The reason being that the time series being forecasted should preferably contain a consistent daily seasonality. Additionally, a further improvement would have been to remove the weekends as well. The weekends are however not as big of a problem because they at least appear at regular intervals.

5.2 Data Collection

Despite collecting weather data consisting of four parameters, none of the parameters were used for the traffic forecasting. The parameters for the temperature and the wind speed proved to have very low correlation with the traffic data and was therefore not relevant for use. Due to the parameters for the sky description and the precipitation being discrete enumerations and not continuous values, the measurement of the correlation for these parameters turns out incorrect. With the data collected, it does not seem possible to accurately distinguish each respective description from the other in order to determine useful values. A potential solution would be to look into methods of measuring correlations between discrete and continuous variables, but this needs to be further investigated.

When working with large amounts of data and an expanding database, the network performance when using the free tier of MongoDB Atlas turns out to be too slow for making queries and insertions. The solution would be to pay for an upgraded tier with higher network performance.

5.3 Feature Engineering

The feature engineering did provide some extra features that improved the forecasting accuracy, but some limitations must be mentioned. The spearman correlations evaluated only captured relationships between features at the same time step. It would not find more complex temporal relationships between two features at different time steps. For example, high traffic flow at road segment X at time step x_t may be correlated with the traffic flow at road segment Y at some future time step x_{t+h}. Using the flow from road segment X to forecast the flow at road segment Y with horizon h would therefore be useful. The features used as predictors in experiment (3) likely included such correlations with segment 58, which the FFNN and CNN models captured. It is however possible that some segments with a lower spearman correlation could have been at least equally as useful. Finding these more complex correlations is more difficult and require shifting the time steps of the features being compared. Another option would be to arbitrarily feed a very large number of road segments as input, and see if the model can find useful correlations by itself. This would of course risk feeding it completely useless data which could increase overfitting and unnecessarily increase computation times.

In addition to this, other features could have been explored such as explicitly telling the model what day of the week is being forecasted. This would likely make it easier

for the model to distinguish between weekdays and weekends.

5.4 Future work

For future work it would be interesting to attempt the same experiments but examine more complex versions of the models used. For example by using more neurons and hidden layers in the neural network architectures. Doing this would however require better hardware, such that the training phase execution time does not become unfeasible. The hardware in mind would be some high end Graphics Processing Unit as they are well optimized for matrix operations, which is a large part of training neural networks. Even better would be to perhaps utilize a new technology released by Google in 2017 called Tensor Processing Units (TPU) [57]. TPUs were built specifically for training neural networks, and are available for usage in the Google Cloud [58].

Also, considering entirely new models would also be an option. One example is to utilize both CNN and LSTM networks in the same model. This has been tried in several previous projects with success [59], [60]. This works by using a CNN network to capture the spatial correlations, and letting the LSTM deal with the temporal dependencies. Another interesting idea was proposed by Ma *et al.* (2017) [61], where they forecast future traffic patterns based on images. In other words, they interpret the traffic speed at various locations of some road network as an image. A CNN network is then used to learn the patterns of the images.

Finally, for a future project, much more data would be needed as this would allow the model to learn all the traffic patterns over an entire year. This would likely improve the results because no matter where the test set is put in time, the training set will at some point have included similar patterns for the previous year. Additionally, finding more interesting predictors through more advanced feature engineering techniques as described in Section 5.3 would be interesting.

5.5 Conclusion

It is clear that machine learning has great potential when it comes to time series forecasting. This has been shown in this thesis as well as in other referenced literature. Existing statistical approaches should however not be underestimated. The baseline methods did in fact achieve decent results and are faster to evaluate compared to the ML techniques. When faced with a forecasting problem, whether its traffic forecasting or something else, the traditional approaches should always be tried first. If they do not perform as well as expected, one could try experimenting with machine learning. If this option is considered, a few things are important to keep in mind. Powerful hardware is crucial as this allows one to train very large and complex ML models at fast speeds. Increased performance due to hardware will in turn open up many doors for further improvement of the ML models. For one, it will speed up grid search optimizations which helps finding better hyperparameters.

Additionally, it will allow for more advanced feature engineering as more data can be fed as input to the model, without making the computation time unfeasible.

References

[1] The World Bank Group. Co2 emissions from transport (% of total fuel combustion). https://data.worldbank.org/indicator/en.co2.tran.zs, 2014. [Online; accessed 07-May-2019].

[2] World Health Organization. Number of road traffic deaths. https://data.worldbank.org/indicator/en.co2.tran.zs, 2013. [Online; accessed 07-May-2019].

[3] European Union. Directive 2010/40/eu of the european parliament and of the council of 7 july 2010 on the framework for the deployment of intelligent transport systems in the field of road transport and for interfaces with other modes of transport text with eea relevance. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32010L0040, 2010. [Online; accessed 07-May-2019].

[4] Peter J Bickel, Chao Chen, Jaimyoung Kwon, John Rice, Erik Van Zwet, and Pravin Varaiya. Measuring traffic. *Statistical Science*, pages 581–597, 2007.

[5] Adrien Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. F. Didot, 1805.

[6] GEORGE EP Box, Gwilym M Jenkins, and G Reinsel. Time series analysis: forecasting and control holden-day san francisco. *BoxTime Series Analysis: Forecasting and Control Holden Day1970*, 1970.

[7] Florin Schimbinschi, Xuan Vinh Nguyen, James Bailey, Chris Leckie, Hai Vu, and Rao Kotagiri. Traffic forecasting in complex urban networks: Leveraging big data and machine learning. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 1019–1024. IEEE, 2015.

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

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[10] & Athanasopoulos G. Hyndman, R.J. Forecasting: principles and practice, 2nd edition. https://otexts.com/fpp2/, 2018. [Online; accessed 07-May-2019].

[11] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[12] Todd E Clark and Michael W McCracken. Improving forecast accuracy by combining recursive and rolling forecasts. *International Economic Review*, 50(2):363–395, 2009.

[13] Souhaib Ben Taieb, Gianluca Bontempi, Amir F Atiya, and Antti Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition. *Expert systems with applications*, 39(8):7067–7083, 2012.

[14] Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*, volume 2. Springer, 2007.

[15] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] B Mehlig. Artificial neural networks. *arXiv preprint arXiv:1901.05639*, 2019.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[19] Lund Research Ltd. Spearman's rank-order correlation. `https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php`, 2018. [Online; accessed 17-May-2019].

[20] Thomas G Dietterich. Machine learning for sequential data: A review. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pages 15–30. Springer, 2002.

[21] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine learning strategies for time series forecasting. In *European business intelligence summer school*, pages 62–77. Springer, 2012.

[22] William Jay Conover and William Jay Conover. Practical nonparametric statistics. 1980.

[23] Marco Lippi, Matteo Bertini, and Paolo Frasconi. Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):871–882, 2013.

[24] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2015.

[25] Michael Nielsen. Neural networks and deep learning. [Online; accessed 13-March-2019].

[26] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[27] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[28] Yonghui Wu and Mike Schuster et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[29] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

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

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[33] MongoDB Inc. What is nosql? `https://www.mongodb.com/nosql-inline/`, 2019. [Online; accessed 05-May-2019].

[34] Albertas Krisciunas. Benefits of nosql. `https://www.devbridge.com/articles/benefits-of-nosql/`, 2014. [Online; accessed 06-May-2019].

[35] MongoDB Inc. Mongodb manual - glossary. `https://docs.mongodb.com/manual/reference/glossary/#term-collection`, 2019. [Online; accessed 07-May-2019].

[36] Rest API Tutorial. Json data types. `https://restfulapi.net/json-data-types/`, 2019. [Online; accessed 07-May-2019].

[37] MongoDB Inc. Mongodb architecture guide. `https://resources.mongodb.com/mongodb-architects/mongodb-architecture-guide/`, 2017. [Online; accessed 07-May-2019].

[38] MongoDB Inc. Mongodb atlas. `https://www.mongodb.com/cloud/atlas/`, 2019. [Online; accessed 07-May-2019].

[39] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621, 2010.

[40] 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.

[41] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3):e0194889, 2018.

[42] Jyun-Yan Yang, Li-Der Chou, Yu-Chen Li, Yu-Hong Lin, Shu-Min Huang, Gwojyh Tseng, Tong-Wen Wang, and Shu-Ping Lu. Prediction of short-term average vehicular velocity considering weather factors in urban vanet environments. In *2010 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3039–3043. IEEE, 2010.

[43] Fangce Guo, John W Polak, and Rajesh Krishnan. Comparison of modelling approaches for short term traffic prediction under normal and abnormal conditions. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1209–1214. IEEE, 2010.

[44] Python Software Foundation python language reference, version 3.6.7. `http://www.python.org`. Accessed: 2019-04-18.

[45] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[46] MongoDB Inc. Pymongo 3.8.0 documentation. `https://api.mongodb.com/python/current/`, 2019. [Online; accessed 05-May-2019].

[47] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006–. [Online; accessed 05-May-2019].

[48] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[49] F. Pedregosa and G. et al. Varoquaux. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[51] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[52] François Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[53] HERE Technologies. Here developer documentation. `https://developer.here.com/documentation`, 2019. [Online; accessed 08-May-2019].

[54] HERE Technologies. Traffic api. `https://developer.here.com/documentation/traffic/topics/what-is.html`, 2019. [Online; accessed 08-May-2019].

[55] HERE Technologies. Destination weather api. `https://developer.here.com/documentation/weather/topics/overview.html`, 2019. [Online; accessed 08-May-2019].

[56] Jason Brownlee. On the suitability of long short-term memory networks for time series forecasting. `https://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series-forecasting/`, 2017. [Online; accessed 16-May-2019].

[57] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

[58] Inc. Google. Cloud tpu. `https://cloud.google.com/tpu/`, 2019. [Online; accessed 05-May-2019].

[59] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Zhenhui Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[60] Jingqing Zhang. Short-term traffic prediction: Modelling temporal-spatial features in local highway networks with deep neural networks. 2018.

[61] Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, 17(4):818, 2017.

A

Appendix 1

A.1 Model Implementations

This section walks through the implementation of the various forecasting models. This was done in accordance with the explanations given in Section 2.4. For the machine learning models, the choice of hyperparameters will also be described. In deciding which hyperparameters to use, a grid search was performed. This was done using a finished tool in the library *scikit-learn*, called *GridSearchCV*. Additionally, the window size n must be chosen. A larger window size means more temporal context, and therefore possibly more accurate forecasts, but also a longer computational complexity. After some testing, it was realized that the window size must be chosen based on the horizon. A shorter horizon did not require nearly as long of a time window compared to longer horizons. The window sizes chosen for each horizon are therefore summarized in Table A.1.

Horizon	10 min (h = 1)	1 hour (h = 6)	6 hours (h = 36)	12 hours (h = 72)	24 hours (h = 144)
Window size	2 hours (n = 12)	6 hours (n = 144)	1 day (n = 144)	2 days (n = 288)	2 days (n = 288)

Table A.1: Chosen window sizes for each forecasting horizon used in all experiments.

The training phase of the various machine learning models does include various random events that may cause the accuracy to differ slightly from time to time. Solving this was done by evaluating each ML model ten times and presenting the average of these.

A.1.1 Naive Approach

This model sets the forecasted traffic flow equal to the current time step.

```
predictions = list()
# Walk-forward validation
for i in range(len(testY)):
    y_hat = test_set[i]
    predictions.append(y_hat)
```

A.1.2 Seasonal Naive Approach

This model sets the forecasted traffic flow equal to corresponding flow at the same time step the week before. With the weekends included in the dataset, the period of the seasonality is 1 week (1008 time steps). Thus, m is set to 1008.

```
data = training_set
predictions = list()
m = 1008 # Seasonality
# Walk-forward validation
for i in range(len(testY)):
    y_hat = data[len(data) - m + horizon]
    data = np.append(data, test_set[i])
    predictions.append(y_hat)
```

A.1.3 Seasonal Historical Average

This is similar to the Seasonal Naive approach, except it averages over all previous days for the same time step being forecasted.

```
data = training_set
predictions = list()
m = 1008 # Seasonality
# Walk-forward validation
for i in range(len(testY)):
    seasonal_sum = 0
    l = len(data)
    num_seasons = int(l / m)
    for j in range(1, num_seasons+1):
        seasonal_sum += data[l + horizon - 1 - j * m]
    y_hat =  seasonal_sum / num_seasons
    data = np.append(data, test_set[i])
    predictions.append(y_hat)
```

A.1.4 ARIMA

The ARIMA order used was empirically set to (1,1,0). The middle parameter decides the degree of differencing and is set to one. This means that the time series was differenced one time in order to achieve stationarity.

```
from statsmodels.tsa.arima_model import ARIMA
data = training_set
predictions = list()
for i in range(len(testY)):
    model = ARIMA(data[i:], order=(1,1,0))
    model_fit = model.fit(disp=0)
    output = model_fit.forecast(steps=h)
    yhat = output[0][h - 1]
    data.append(test_set[i])
    predictions.append(yhat)
```

A.1.5 FFNN

Table A.2 shows the chosen hyperparameters for the FFNN model. It can be seen that the number of neurons in experiment (3) increases. This is logical since the first two experiments produce forecasts based on one feature. Only 100 neurons in each layer was necessary to deal with that much data. For experiment (3), however, more features were considered and therefore more data had to be analyzed by the network. This consequently required the network to consist of more neurons. Also, a regularization technique called *dropout* was used to reduce overfitting (see code implementation).

	Exp. (1)	**Exp. (2)**	**Exp. (3)**
Number of hidden layers	3	3	3
Neurons in each layer	200	200	500
learning rate	0.01	0.01	0.01
Epochs	60	60	60
Batch size	30	30	60

Table A.2: FFNN hyperparameters.

Note that the class *Dense* represents a fully connected layer in the code below.

```
import tensorflow as tf
from keras import Sequential
from keras.layers import Dense, Dropout, Activation

# Flatten input (to support multivariate input)
n_input = trainX.shape[1] * trainX.shape[2]
trainX = trainX.reshape((trainX.shape[0], n_input))

n_input = testX.shape[1] * testX.shape[2]
testX = testX.reshape((testX.shape[0], n_input))

# Create multilayered FFNN model
model = Sequential()
model.add(Dense(100, activation='relu', input_dim=trainX.shape[1]))
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(100, activation='relu'))
model.add(Dense(trainY.shape[1]))
model.compile(loss="mae", optimizer='adam')
model.summary()

# Fit model
history = model.fit(trainX, trainY, epochs=60, verbose=1)

# Predict the test set
predictions = model.predict(testX)
```

A.1.6 LSTM

The LSTM network similarly required more neurons for experiment (3), and is shown in Table A.3. The number of epochs used was only 30 because the training time for LSTM networks is significantly longer.

	Exp. (1)	Exp. (2)	Exp. (3)
Number of hidden layers	2	2	2
Neurons in each layer	32	32	100
learning rate	0.01	0.01	0.01
Epochs	30	30	30
Batch size	60	60	100

Table A.3: LSTM hyperparameters.

```
import tensorflow as tf
from keras import Sequential
from keras.layers import Dense, Dropout, LSTM, Activation

# Create LSTM model
model = Sequential()
model.add(LSTM(32, input_shape=(trainX.shape[1],
                trainX.shape[2]), return_sequences=True))
model.add(LSTM(32))
model.add(Dense(trainY.shape[1]))
model.summary()
model.compile(loss="mae", optimizer='adam')

# Fit model
history = model.fit(trainX, trainY, epochs=30, verbose=1)

# Predict the test set
predictions = model.predict(testX)
```

A.1.7 CNN

Table A.4 shows the chosen hyperparameters for the CNN model. All CNN models used consisted of 3 hidden layers. One convolutional layer, one max pooling layer, and finally one fully connected layer. Various parameters chosen for the first two layers (such as filters and kernel size) are shown in the code below. The number of neurons in the fully connected layer is presented in Table 4.2.

```
import tensorflow as tf
from keras import Sequential
from keras.layers import Dense, Dropout, Conv1D, MaxPooling1D, Flatten,
    Activation

# Create CNN model
model = Sequential()
```

	Exp. (1)	Exp. (2)	Exp. (3)
Number of hidden layers	3	3	3
Neurons in fully connected layer	50	50	200
learning rate	0.01	0.01	0.01
Epochs	60	60	60
Batch size	30	30	60

Table A.4: CNN hyperparameters.

```
model.add(Conv1D(filters=16, kernel_size=3, activation='relu',
    input_shape=(trainX.shape[1], trainX.shape[2])))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(trainY.shape[1]))
model.compile(loss='mse', optimizer='adam')
model.summary()

# Fit model
history = model.fit(trainX, trainY, epochs=60, verbose=1)

# Predict the test set
predictions = model.predict(testX)
```

A.1.8 Support Vector Regression

No grid search optimization was performed for the SVR model. The parameters used are the ones shown in the code below.

```
from sklearn import svm
from sklearn.svm import SVR

# Flatten input data (to support multivariate input)
n_input = trainX.shape[1] * trainX.shape[2]
trainX = trainX.reshape((trainX.shape[0], n_input))

n_input = testX.shape[1] * testX.shape[2]
testX = testX.reshape((testX.shape[0], n_input))

# Create SVR model
clf = SVR(kernel="rbf", degree=2, C=100, epsilon=.01)

# Fit model
clf.fit(trainX, trainY)

# Predict the test set
predictions = clf.predict(testX)
```