CHALMERS

Highway tollgates traffic prediction using a stacked autoencoder neural network

Master's thesis in Computer Science - Algorithms, Languages and Logic

OSKAR KÄRRMAN
LINNEA OTTERLIND

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2018

MASTER'S THESIS IN COMPUTER SCIENCE - ALGORITHMS, LANGUAGES AND LOGIC

Highway tollgates traffic prediction using a stacked autoencoder neural network

OSKAR KÄRRMAN
LINNEA OTTERLIND

Department of Mechanics and Maritime Sciences
Vehicle Safety
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2018

Highway tollgates traffic prediction using
a stacked autoencoder neural network
OSKAR KÄRRMAN
LINNEA OTTERLIND

Master's thesis 2018:52
Department of Mechanics and Maritime Sciences
Vehicle Safety
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Chalmers Reproservice
Göteborg, Sweden 2018

Highway tollgates traffic prediction using
a stacked autoencoder neural network
Master's thesis in Computer Science - Algorithms, Languages and Logic
OSKAR KÄRRMAN
LINNEA OTTERLIND
Department of Mechanics and Maritime Sciences
Vehicle Safety
Chalmers University of Technology

Abstract

Traffic flow prediction is an important area of research with a great number of applications such as route planning and congestion avoidance. This thesis explored artificial neural network performance as travel time and traffic volume predictors. Stacked autoencoder artificial neural networks were studied in particular due to recent promising performance in traffic flow prediction, and the result was compared to multilayer perceptron networks, a type of shallow artificial neural networks. The Taguchi design of experiments method was used to decide network parameters. Stacked autoencoder networks generally did not perform better than shallow networks, but the results indicated that a bigger dataset could favor stacked autoencoder networks. Using the Taguchi method did help cut down on number of experiments to test, but choosing network settings based on the Taguchi test results did not yield lower error than what was found during the Taguchi tests.

Keywords: stacked autoencoder, multilayer perceptron, neural network, traffic prediction, traffic flow, taguchi

Acknowledgements

We are immensely thankful to our supervisor Selpi who helped and supported us throughout the project. We also want to thank Harald Otterlind who lent us a computer we could use for neural network training. Finally we want to thank all friends and family for the support, and the people at SAFER for providing a great place for us to work on the thesis.

Oskar Kärrman & Linnea Otterlind

Contents

1 Introduction

This thesis aims to investigate artificial neural networks as traffic flow predictors using data provided by the Knowledge Discovery and Data Mining (KDD) Cup 2017. The KDD Cup is an annual competition about data mining and machine learning topics.

1.1 Problem description

Tollgates are commonly used to collect tolls and can often be found at expressways in many parts of the world. Tollgates without automatic electric toll collection are a source of large amounts of traffic congestion, which can easily overwhelm traffic management authorities [6]. To counter the effects of congestion one could for example deploy temporary toll collectors or adjust traffic lights at nearby intersections. However, this can only be done efficiently if traffic flow can be predicted to a reasonable extent.

In this thesis we investigate whether a stacked autoencoder neural network could be a satisfactory solution to the following problems, and compare the results to a traditional shallow neural network.

In a closed system further explained in Chapter 3 and pictured in Figure 3.1 containing three intersections (A, B, C) and three tollgates (1, 2, 3)

1. **Estimate the average travel time from designated intersections to tollgates.**
   For every 20-minute time window in 08:00 to 10:00 and 17:00 to 19:00 between the 18th and 24th of October 2016, estimate the average travel time of vehicles for a specific route, given traffic and weather data collected between the 19th of June and the 17th of October 2016.

   - Routes from intersection A to tollgates 2 and 3.
   - Routes from intersection B to tollgates 1 and 3.
   - Routes from intersection C to tollgates 1 and 3.

2. **Predict average tollgate traffic volume.**
   For every 20-minute time window in 08:00 to 10:00 and 17:00 to 19:00 between the 18th and 24th of October 2016, predict the entry and exit traffic volumes at tollgates 1, 2 and 3, given traffic and weather data collected between the 19th of September and the 17th of October 2016.

The objectives are to:

1. Implement and test a stacked autoencoder neural network as well as a shallow neural network for the above specified problems.
2. Find the optimal settings of each network and for each task using the Taguchi method of designing tests.
3. Compare the performance of the two networks to address the two tasks using the optimal settings.

1.2 Related work

Accurate traffic flow prediction has many useful applications such as route planning and congestion avoidance, and has therefore been researched extensively for many years. Many different models have been studied and compared, including autoregressive integrated moving average (ARIMA), neural networks, historical average and nearest neighbour. These four models were compared in *Traffic Flow Forecasting: Comparison of Modeling Approaches* published in 1997 [14], which describes nearest neighbour as the preferred method. It was also concluded that using neural networks often resulted in under estimations of the traffic flow. However, more recent studies suggest that certain types of neural networks can be used for traffic flow prediction with great success [5, 18, 19].

In a study by Ishak and Alecsandru [5], four different neural network architectures were tested by predicting average 5-minute speeds. These architectures were the widely studied multi-layer perceptron (MLP) network, a modular network consisting of multiple parallell multi-layer perceptrons, a hybrid principal component analysis (PCA) network using PCA for feature reduction and MLP for predictions, and a co-active neuro-fuzzy inference system (CANFIS) which integrates neural networks with fuzzy inference systems. The architectures were

compared by measuring the average absolute relative error (AARE) and the root mean square error (RMSE). Training data was collected using loop detectors on a freeway segment in Florida. Input was divided into short term and long term memory, where short term was considered data observed 10 minutes prior to the prediction and long term was historical data. The study showed that long term memory was more important for better performance as the prediction horizon increased. None of the network architectures outperformed all others for every prediction horizon (5, 10, 15 and 20 minutes), however the CANFIS architecture proved to be the more preferable choice for 10 and 20 minute horizons.

Lv et al. [8] explored a stacked autoencoder with logistic regression neural network for traffic flow, trained and tested on data from freeway systems in California. Predictions for 15, 30, 45 and 60 minutes were made and compared with other techniques by calculating the mean absolute error (MAE), mean relative error (MRE), and RMSE. The stacked autoencoder network outperformed the other techniques, which were: standard back-propagation neural network, random walk, support vector machine, and radial basis function neural network (RBFNN).

Another study conducted in 2016 by Yang et al. [18] compared three existing neural network architectures and a novel architecture called stacked autoencoder Levenberg–Marquardt (SAE-LM). The existing architectures tested were a hybrid exponential smoothing approach with the Levenberg-Marquardt algorithm (EXP-LM), a network based on the particle swarm optimization algorithm (PSONN), and a radial basis function neural network (RBFNN). Comparisons were made by predicting traffic flow and calculating the mean absolute percentage error (MAPE) and the variance absolute percentage error (VAPE). To find the best parameters for the SAE-LM model the Taguchi method of designing tests was used and proved successful. The results clearly showed that SAE-LM outperformed the other architectures using real world data collected from the M6 freeway in the U.K. However, the improved performance came at cost as the SAE-LM architecture had the longest computational time (training and forecasting) of all tested networks. The total computational time for the SAE-LM architecture was about 12% longer than for the second slowest network (PSONN), and about 62% longer than for the fastest network (RBFNN).

Recent studies using neural networks also used Taguchi's design of experiments to figure out what parameters to use. Pontes et al. [11] used the Taguchi method when designing a radial basis function neural network for predicting surface roughness in the turning process of hardened steel. The parameters tested for the architecture were the number of radial units on the hidden layer, the algorithm for spread factor calculation of radial units, and the algorithm for calculation of the center location of radial functions. The authors concluded that the Taguchi method was efficient for tuning network parameters in the context of the study.

Another study by Packianather et al. [10] explores the Taguchi method when designing a multilayered feed forward neural network trained using standard back-propagation. The network was trained to classify defects in birch wood veneer, and the design parameters chosen for the Taguchi tests were the learning rate, momentum, number of hidden neurons in the first hidden layer, and the number of neurons in the second hidden layer. The most significant factors were found to be the learning rate and number of neurons in the first hidden layer.

In a case study by Sukthomya and Tannock [15, 16] Taguchi's design of experiments was used to identify the optimal settings in a multilayer perceptron network. The network was trained with data from a superplastic forming process for manufacturing a wide-chord fan blade, and the network would then model the actual manufacturing process. The parameters to optimize with the Taguchi method were the learning algorithm, number of layers, transfer function, proportion of validation data, and a *hint*. The hint given to the network was input representing the average level of the output from the network. This was added to help the network deal with a shifting mean value in the data which occurred when the tooling in the manufacturing process was changed or adjusted.

1.3 Scope

This thesis is limited to investigating the use of stacked autoencoder neural networks and shallow neural networks for traffic flow predictions, as specified in Section 1.1. The solution could potentially be used for traffic flow prediction in similar systems with small modifications, however, the performance of the networks is not guaranteed when applied to a different set of data. The networks are implemented in the programming language Python using the Tensorflow library provided by Google, no other similar software will be used or tested. The KDD Cup 2017 provided the data necessary for the given tasks, and the thesis is limited to the use and availability of this data.

1.4 Thesis outline

Chapter 2 describes the relevant theory needed to understand the techniques used in the implementation and testing. The chapter describes artificial neural networks, how to train them, different architectures, and how to process input data to neural networks. The two architectures described are a common shallow feed forward neural network, along with a stacked autoencoder. Section 2.2 contains a description of the Taguchi method of experiments which is used in this project to decide network parameter levels.

Chapter 3 describes how the data used for traffic prediction is structured, what kind and how much data is available.

Chapter 4 details how the neural networks in this project are designed, how predictions are made, and what different methods are used and evaluated when filling in missing data while training the networks. This is followed by specific implementation details, how to use the Taguchi method to evalute network parameters, and finally which network parameters are evaluated.

Chapter 5 presents the results obtained for the network parameter evaluation, followed by the results from tests performed after choosing network parameters.

Chapter 6 discusses the results and what factors could have been important when producing those results.

Finally, the thesis outcome is concluded in Chapter 7.

2 Background

This chapter introduces the concepts and theories the thesis relies upon. Section 2.1 explains what an artificial neural network (ANN) is, as well as some example networks, such as a shallow feed forward neural network. This is followed by an explanation of stacked autoencoders and how they operate. Consecutively, it is explained how to prepare input data to potentially achieve better results. Finally, Section 2.2 contains an explanation of what the Taguchi method is as well as why it is useful in the area of ANNs. General descriptions of neural networks, autoencoders, and sparsity are summarized from [9]. Additional explanations of autoencoders, sparsity, and stacked autoencoders are summarized from [8]. Data representation for neural networks along with how to normalize the data is common knowledge in neural network research, hence no specific source was used for Section 2.1.4 describing this. The description of the Taguchi method is summarized from [10] and [11].

2.1 Artificial neural networks

An artificial neuron is a mathematical model attempting to mimic the process of a biological neuron in the brain. It takes an arbitrary number of inputs, multiplies them by a weight matrix and adds a bias, before applying a so called activation function and outputting a value, as shown in Figure 2.1.

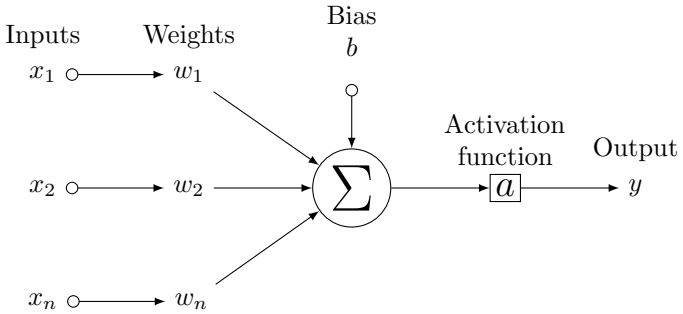

Figure 2.1: Operations in an artificial neuron.

The mathematical process of an artificial neuron can thus be described as in equation 2.1, where $\boldsymbol{x}$ is the input vector, $\boldsymbol{w}$ is the weight vector, b is the bias, a is the activation function and y is the output of the neuron.

$$y = a\left(\sum(\boldsymbol{w}\boldsymbol{x}) + b\right) \tag{2.1}$$

There are many different kinds of activation functions used in artificial neurons, for example the identity function and softsign. Softsign is a bit more complex than the identity function, which simply returns its input, and will furthermore scale its input into the range $(-1, 1)$ as shown in equation 2.2.

$$softsign(x) = \frac{x}{1 + |x|} \tag{2.2}$$

An artificial neural network aims to mathematically simulate the biological neural network of the brain and can be used to solve classification problems or to make predictions. It consists of a number of artificial neurons connected by weighted links. The links between neurons are weighted to emulate how not all connections between them are of equal importance. There are many different strategies available to use when deciding what activation functions to use and how to connect neurons in a neural network. In this thesis, only the shallow feed forward and stacked autoencoder neural network types will be discussed in depth.

2.1.1 Training an artificial neural network

There are two main ways to train a neural network: supervised and unsupervised. When using unsupervised learning the network is given input and is left to see correlations and groupings in that data without anyone telling it what is ultimately correct. In supervised learning however, one gives the ANN some input and a corresponding output, the network goes through all of its calculations, and yields its own output. The network then compares that output to the expected output it was given, calculating how big of an error it made

according to a predefined function, called the loss, error, or cost function. There are many different ways of measuring the error, two examples are the mean absolute percentage error (MAPE) and mean squared error (MSE) specified in equation 2.3 and 2.4 respectively, where $\boldsymbol{y}$ are the actual values, $\hat{\boldsymbol{y}}$ are the predicted values, and $n = |\boldsymbol{y}| = |\hat{\boldsymbol{y}}|$.

$$MAPE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{2.3}$$

$$MSE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.4}$$

Given the size of the error, the weights and biases of the network are adjusted to get closer to the real output. The adjustment of weights and biases is made by applying an optimization algorithm to the chosen loss function.

An optimization method that is commonly used to optimize neural networks is the so called gradient descent method [13]. There are a few variations of gradient descent but the basic method consists of calculating and analyzing the gradient of the loss function with respect to its parameters. The parameters are updated in the opposite direction of the gradient to follow the slope of the loss function and reach a minimum. The speed at which this is done is determined by a predefined learning rate. Thus if the learning rate is increased the global minimum might be found sooner but we also risk missing it altogether, getting stuck in a local minimum instead. If the learning rate is decreased it will take longer to find the minimum but it is more likely that the global minimum is not missed.

Gradient descent uses the same learning rates for every training data point used, even though it may happen that some samples are very close to a minimum and others are very far from it. Another optimization method which has identified and solved this is called Adam, derived from adaptive moment estimation [7]. Adam adapts the learning rates of each parameter so that if the gradient is steep the learning rate is high and when the gradient flattens out the learning rate drops to prevent overshooting the minimum. This means that the global minimum can potentially be found faster and without high risk of missing it.

Error functions and optimization methods are commonly used to solve both classification and regression problems. Classification is when the network is trying to determine which class a given input belongs to. The network has as many output nodes as there are different classes and each node represents one class and the networks confidence (from 0 to 1) that the given input belongs to that class. Another way to explain classification is to say that the network learns to group the data. In regression on the other hand, the network is asked to output one or several real values given some input data. Regression is used when using a network to make predictions based on some previous data.

When training a neural network to classify or predict, all available data will be divided into training, validation and test data. The training data is used to train the network, i.e. the network is given some input from this block of data and after calculating an output, the corresponding actual output is provided and weights and biases are adjusted accordingly. The validation data is used to check for and prevent so called over training or over fitting. One could imagine that if the network trains long enough on the same data it will become incredibly adept at classifying or predicting in that span. However, outside the scope of that data the network will most likely not be able to make any accurate estimations, it was only ever told to fit its curve to the data it was given. This can be prevented using the validation data by periodically calculating the error on the validation data and assure that it is improving. When the error on the validation data starts to increase while the error of the training data still decreases, the network has started to over fit and training should cease, to keep the network's ability to generalize. The test data is introduced only after training of the network has finished, and is used to measure the performance of the network. In the test data there are inputs and corresponding output but, as with the validation data, the network is never given the outputs. The network outputs its own classifications or predictions which are then compared to the correct output to measure the performance of the network.

2.1.2 Feed forward neural network

A common type of neural network is the feed forward neural network. Feed forward means that there are no circular connections in the network, any input data moves only forward through neurons until it reaches the output nodes. This type of network is often divided into distinct layers: the input layer, an arbitrary number of hidden layers, and an output layer. A typical neural network with one hidden layer is pictured in Figure 2.2.

The input moves from the input layer through to the first hidden layer, then through the subsequent hidden layers and finally through the output layer.

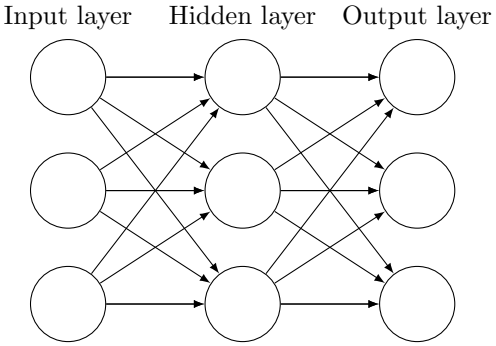

Figure 2.2: A typical feed forward neural network.

A neural network is called *shallow* if it only has one hidden layer, and conversely *deep* if it has more than one hidden layer.

2.1.3 Stacked autoencoder

An autoencoder is a shallow artificial neural network trained to reproduce its input, and is used in unsupervised learning to discover interesting features of the input. It consists of two steps; encoding and decoding. Encoding transforms the input vector $\boldsymbol{x}$ to the hidden vector $\boldsymbol{h}$ by applying the function h as depicted in equation 2.5 where f is an activation function.

$$h(\boldsymbol{x}) = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{h} \tag{2.5}$$

Decoding then transforms the hidden vector $\boldsymbol{h}$ into the output vector $\boldsymbol{z}$ by applying the function depicted in equation 2.6, where g is an activation function. If $\boldsymbol{x}$ consists of real values it is common to omit the activation function g [17].

$$z(\boldsymbol{h}) = g(\boldsymbol{W'}\boldsymbol{h} + \boldsymbol{b'}) = \boldsymbol{z} \tag{2.6}$$

The output of the autoencoder $\boldsymbol{z}$ is then compared to the input vector $\boldsymbol{x}$ and loss is commonly calculated using *squared error* if the input consists of real values, or *cross-entropy loss* if the input consists of binary values [17].

If an autoencoder has a hidden layer of equal or bigger size than the input layer, the autoencoder could potentially learn the identity function and become useless as no information has been learned from the input. Different methods exists to prevent this, one of them is imposing a *sparsity constraint* on the network [9]. The sparsity constraint penalizes the network if the average activation of neuron j over the training set is far from a specified *sparsity parameter*. A sparsity parameter of 0.1 encourages neuron j to be active close to 10% over all inputs in the training set. The sparsity constraint is achieved by adding an additional parameter when calculating the loss which increases in value when the average activation of neurons in the network deviates significantly from the sparsity parameter. Weight of the sparsity constraint compared to the original loss function is tuned by a *sparsity constant* and the full loss equation is described in equation 2.7, where L is the original loss function, c is the sparsity constant, H_D is the total number of hidden neurons, S is the sparsity function, p is the sparsity parameter and $\hat{p}_j$ is the average activation of neuron j.

$$L(\boldsymbol{x}, \boldsymbol{z}) + c\sum_{j=1}^{H_D} S(p, \hat{p}_j) \tag{2.7}$$

The sparsity function S should approach 0 as $\hat{p}_i$ approaches p, and increase when $\hat{p}_i$ deviates significantly from p. The Kullback–Leibler (KL) divergence has those features and can be used as a sparsity constraint, defined as in equation 2.8, and the loss function along with KL divergence as the sparsity constraint is described in equation 2.9.

$$KL(p||\hat{p}_j) = p\log\frac{p}{\hat{p}_j} + (1-p) * \log\frac{1-p}{1-\hat{p}_j} \tag{2.8}$$

$$L(\boldsymbol{x}, \boldsymbol{z}) + c \sum_{i=1}^{H_D} KL(p||\hat{p}_j) \tag{2.9}$$

A stacked autoencoder (SAE) consists of multiple autoencoders added sequentially where the output of the previous autoencoder becomes the input to the next, creating a deep network. If an additional predictor is added on top of the SAE, the network can be used for prediction. Models based on SAE has previously been used successfully in traffic flow prediction [18], cancer detection [2] as well as general machine learning benchmarks [17].

To train a SAE network, autoencoders are trained layerwise starting with the autoencoder closest to the input (the first autoencoder). After the first autoencoder is sufficiently trained to reproduce its input, the output layer of the first autoencoder is removed and the output of the hidden layer instead becomes the input to the second autoencoder. When training the second autoencoder, input is given to the first autoencoder with locked weights and biases. The target of the second autoencoder becomes the output of the first autoencoder's hidden layer and training proceeds in the same manner. Subsequent autoencoders are added in the same way: the output of the hidden layer from autoencoder l is the input to autoencoder $l+1$. When enough autoencoders has been added, a predictor can be used to complete the network. At this point, weights can be unlocked and additional fine tuning using supervised learning can be applied.

2.1.4 Input preparation

The data input into a neural network can use any arbitrary number representation, however the performance of the network can change depending on that representation. For example, for the typical problem of classifying handwritten digits from 100 by 100 pixel images in black and white, the network could have 10 000 $(100 * 100)$ input nodes, each representing a pixel. The value of each input would be decided by the amount of black or white used in the corresponding pixel, a percentage, which could for example be represented by a number from zero to one, or equally by a number from zero to 100. What representation to choose for the data is evaluated on a case-by-case basis, but previous studies can be used to assess pros and cons with different representations.

For data divided into classes (nominal data), multiple methods are possible to use when feeding it into a network. The naive way would be to have only one input node representing what class the input belongs to, so if the data contains n classes we would order the classes, assign each class a number, and input an integer e.g. $[1, n]$ depending on the class. Since the network does not know that the input represents nominal data and not a real valued number, the network might incorrectly infer that class 1 and 2 are more alike than class 1 and 5 due to the values being closer. Another way to input nominal data is to use a 1-of-n technique, also called one-hot. This technique uses a vector of n input nodes where n is the number of classes in the nominal data. All classes are ordered and assigned a number $[1, n]$, and when data of class m is used as input, input node m is set to be active while all other input nodes are set to be inactive. An exception to this is if the number of classes is 2 (binary data). If so, only one input node is used, and one of the classes is described by setting the input node to active while the other class sets the input node to inactive.

Real valued inputs are often *normalized* before being fed to the network, setting the average of every input close to 0, and makes sure that different data points are relatively close in magnitude. If different inputs has a vastly different magnitude, they will have different relative significance when calculating the error, which affects the rate at which the network will learn. A common technique for normalizing real valued data is calculating the *standard score* (standardization), which is used when the data is approximately normally distributed. To standardize the data, one first calculates the mean and standard deviation over all data points on that specific attribute. Then the mean is subtracted from each data point, and the result is divided by the standard deviation. This produces the standardized value v_n as described in equation 2.10 where v is the original value, μ is the calculated mean and σ is the calculated standard deviation.

$$v_n = \frac{v - \mu}{\sigma} \tag{2.10}$$

2.2 Taguchi design of experiments

Taguchi's design of experiments is a method developed by Genichi Taguchi that aims to design tests in manufacturing processes to identify the parameter levels to get the best and most stable performance [12]. All tests are planned and performed without analyzing the results in between, in contrary to trial and error

methods where results are analyzed before the next test is performed. The method uses orthogonal arrays to structure the tests, and when completed the results are statistically analyzed to reveal what combination of parameter levels should be used.

To perform a test, parameters and different levels of these parameters are chosen. The parameters are assumed to be independent, but the tests can give useful information even with dependent parameters [10]. These are then fitted into an appropriate orthogonal array, where an orthogonal array is denoted L_i where i is the number of tests that needs to be performed. Orthogonal arrays for all possible number of features and levels does not exist [1], so to choose which orthogonal array to use for testing one should look for already published arrays that contains at least as many features and levels as needed. An array with more features and/or levels than needed can be used with modifications, where the additional features and levels should be filled with dummy variables. After all tests are performed, the result of every test is coupled with the parameter levels used. Analysis of the mean values, variance and other attributes can then be performed at parameter level to see what different parameters contribute to the result.

The number of tests performed when structured in an orthogonal array is drastically reduced compared to testing all possible combinations, while still having properties to ease the result analysis. These attributes includes testing of all possible pairwise combinations of variables, along with equal distribution of all pairwise combinations. A typical orthogonal array is the L_4 array depicted in Table 2.1, where each row is a test and each column describes what level that specific feature parameter should be at in the test. The L_4 array contains three features with two levels each. If all combinations were to be tested, 8 (i.e., 2^3) tests would be needed instead of the 4 tests that are needed in the L_4 design. The difference in needed tests increases as the parameters and levels increases, and a test with 13 parameters with 3 levels each would need $3^{13} = 1\,594\,323$ tests to be performed if all possible combinations should be tested, while only needing 27 tests using a L_{27} array.

Table 2.1: The L_4 orthogonal array.

Test	**F1**	**F2**	**F3**
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

3 Provided data

The Chinese traffic system from which the data has been collected is specified as follows:

There are three different intersections (A, B, C), and three different tollgate locations (1, 2, 3). Intersection A connects to tollgates 2 and 3, while intersection B and C connect to tollgates 1 and 3. In tollgate locations 1 and 3 one can both enter and exit the highway, while tollgate location 2 is only an entry on to the highway. Intersection and tollgate locations are further connected by roads, which are in turn made up of several road segments. The road segments are connected in three places. The road system is depicted in Figure 3.1.

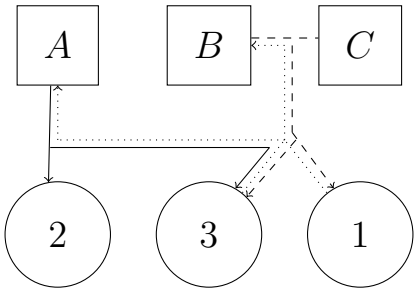

Figure 3.1: Road segment connections in the road system.

The available data has been collected from the 19th of July to the 24th of October 2016 and includes weather data as well as information about which vehicles pass each intersection or tollgate location at what time. This makes it possible to calculate average travel times between intersections and tollgates as well as the traffic volume at each tollgate location for each 20 minute period.

In between the 19th of July and the 24th of October there are two Chinese holidays that may influence the traffic condition. These holidays occur from the 15th to 17th of September and from the 1st to 7th of October, respectively, as pictured in Figure 3.2. The figure also describes what parts of the data is used for training, validation, and testing.

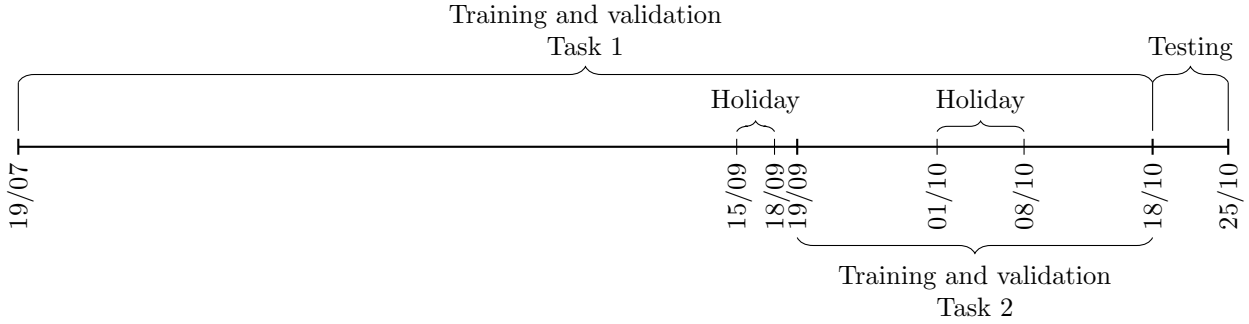

Figure 3.2: The time line of Task 1 and Task 2 data, showing holidays and distribution of training, validation and test data. The time line is not made to scale.

As has been mentioned in Section 1.1 predictions are to be made for the hours 8:00-10:00 and 17:00-19:00. The data collected from the 18th to the 24th of October is selected as test data, as shown in Figure 3.2. For this week the data of the two hours before the critical hours, 6:00-8:00 and 15:00-17:00, will be the only available data to the networks for the primary prediction. A secondary prediction which uses more than the data in 6:00-8:00 and 15:00-17:00 hours is also made.

3.1 Task 1 - travel time

For Task 1 traffic data collected from the 19th of July to the 17th of October is available for training and validation of the networks, as shown in Figure 3.2. In other words, there is data collected from 91 consecutive days, meaning that there should be 91 data points for each of the 72 (= 24 hours∗3 intervals/hour) 20-minute

intervals. This also means that for each of the six intersection-tollgate pairs, there are 6 552 (= 91 days∗72 intervals/day) distinct 20-minute intervals. However, for some of the intervals data is missing.

Figure 3.3 shows that there is a clear increase in missing data points during night time, most are missing from 20-minute intervals between 19:00 and 07:00. For B:1, C:1 and C:3 almost all 91 data points are missing in each of the 20-minute intervals between 00:40 and 04:40. Figure 3.3 also shows that intersection A to tollgate 2 is missing the least number of data points. However, as can be seen in Table 3.1 this intersection-tollgate pair is still missing data for almost 9% of the 20-minute intervals, which is a significant amount.

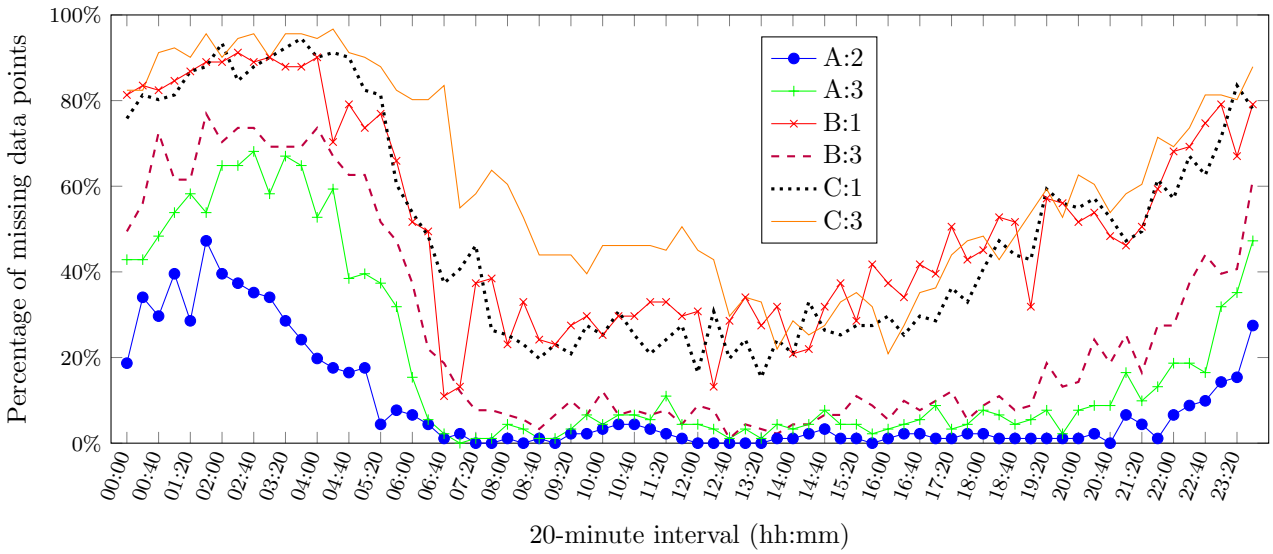

Figure 3.3: The percentage of missing points of each intersection-tollgate pair and 20-minute interval during all days for Task 1.

As can be seen in Table 3.1, C:1 is missing almost 50% of the total number of data points, B:1 is missing more than 50% and C:3 is missing more than 60% of its data. In total there should be 39 312 traffic data points available for Task 1, 6 552 points for each of the six intersection-tollgate pairs, however, 14 168 points or 36% of the points are missing.

Table 3.1: The number of existing and missing data points of each intersection-tollgate pair in the training and validation data set for Task 1.

Intersection:Tollgate	**Existing**	**Missing**	**Missing (%)**
A:2	5965	587	8.96%
A:3	5304	1248	19.04%
B:1	3206	3346	51.07%
B:3	4803	1749	26.69%
C:1	3290	3262	49.79%
C:3	2576	3976	60.68%

Two separate Chinese holidays occur in between the 19th of June and the 24th of October, as previously mentioned and pictured in Figure 3.2. The 1st to the 7th of October 2016 was a Chinese national holiday week. However, this does not seem to affect the data significantly, as can be seen in Figure 3.4. Only the intersection-tollgate pair A:2 has been depicted here, because it is missing the least number of points. The plots of the data from the 28th of September to the 10th of October for the rest of the pairs can be found in Appendix B. The same thing holds true for the holiday from the 15th to 17th of September, none of the pairs show significant changes during this time. Plots from the 12th to the 20th of September, for each of the pairs, can be found in Appendix B.

For the week of the test data, from the 18th to the 24th of October, the total number of 20-minute intervals is 504 (= 72 intervals/day∗7 days). From Table 3.2 one can tell that a significant part of the test data is missing as well. However, when restricting the 20-minute intervals to the critical eight hours of each day, 06:00-10:00

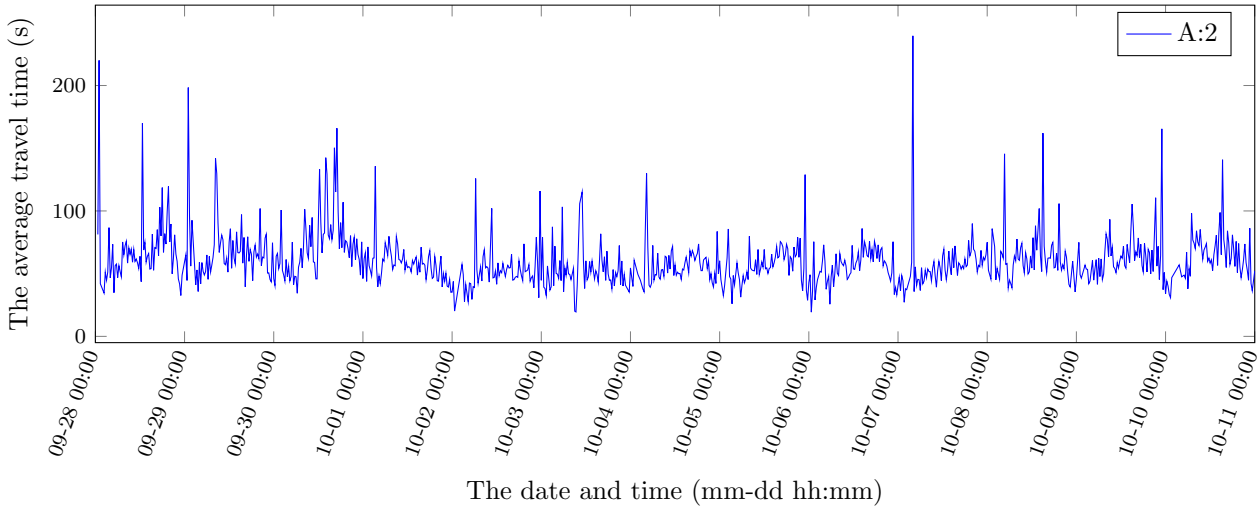

Figure 3.4: Task 1 data, from intersection A to tollgate 2, from the 28th of September to the 10th of October, showing the holiday week and three regular days before and after for reference.

and 15:00-19:00, there are only 168 (= 7 days∗8 hours∗3 intervals/hour) intervals and the number of missing data points is as shown in Table 3.3

Table 3.2: The number of existing and missing data points of each intersection-tollgate pair in the test week.

Intersection:Tollgate	**Existing**	**Missing**	**Missing (%)**
A:2	463	41	8.13%
A:3	431	73	14.48%
B:1	331	173	34.32%
B:3	371	133	26.39%
C:1	324	180	35.71%
C:3	248	256	50.79%

Table 3.3: The number of existing and missing data points of each intersection-tollgate pair in the critical eight hours (06:00-10:00 and 15:00-19:00) of each day in the test week.

Intersection:Tollgate	**Existing**	**Missing**	**Missing (%)**
A:2	167	1	0.60%
A:3	168	0	0.00%
B:1	148	20	11.90%
B:3	161	7	4.17%
C:1	145	23	13.69%
C:3	120	48	28.57%

During the whole test week there should be 3 024 data points in total, 504 points for each pair. 856 data points or 28.3% of these are missing. For only the eight critical hours there should be 1 008 data points in total, 168 points for each pair. Here 99 data points or 9.82% are missing.

3.2 Task 2 - traffic volume

For Task 2 data collected from the 19th of September to the 17th of October is available for training and validation of the networks, as shown in Figure 3.2. The available data has been collected during only 21 consecutive days, meaning that for each of the 72 20-minute intervals during a day, there should be 21 data

points. This means that for each of the five tollgate-direction pairs there are 2 088 (= 21 days∗72 intervals/day) distinct 20-minute intervals. However, as with Task 1, data is not available for all of the intervals.

The number of data points available for training and validation of Task 2 is less than a third of the number of data points available for Task 1. However, as can be seen in Figure 3.5, the data of Task 2 is more complete. The only tollgate-direction pair that is missing a significant amount of data is tollgate 2, through which it is only possible to enter the highway. This tollgate-direction pair is missing 17.43% of its data points while the remaining four pairs are all missing less than 0.2% of theirs, as can be seen in Table 3.4.

We can also see that for the only tollgate-direction pair that is missing a significant number of data points, most of them are missing during night time. Between 02:00 and 04:40 more than 70% of the data points are missing for tollgate 2.

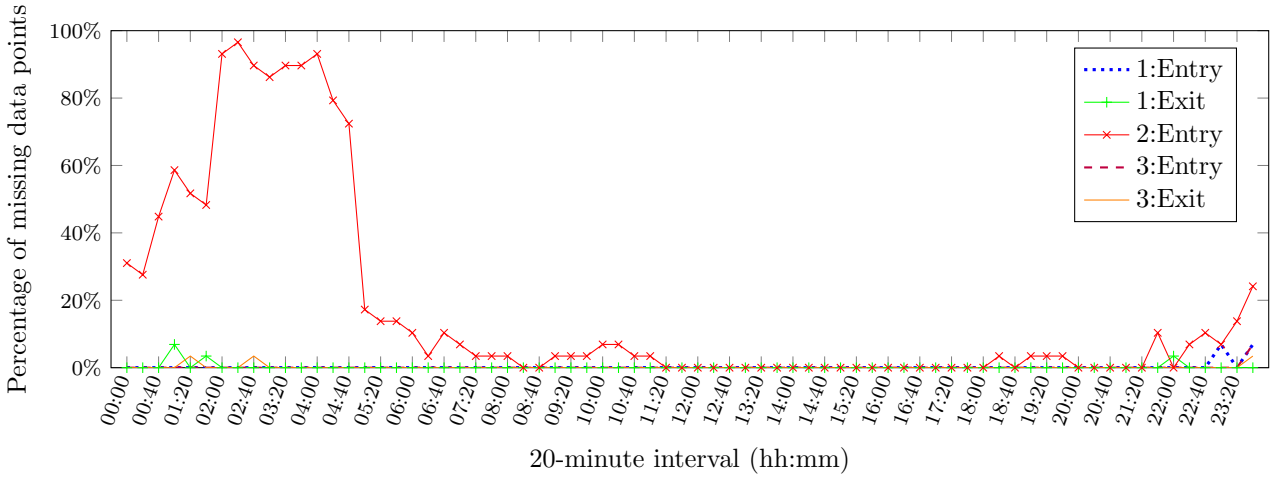

Figure 3.5: The percentage of missing points of each tollgate-direction pair and 20-minute interval during a day for Task 2.

Table 3.4: The number of existing and missing data points of each tollgate-direction pair in the training and validation data set of Task 2.

Tollgate:Direction	**Existing**	**Missing**	**Missing (%)**
1:Entry	2084	4	0.19%
1:Exit	2084	4	0.19%
2:Entry	1724	364	17.43%
3:Entry	2086	2	0.10%
3:Exit	2085	3	0.14%

From the 19th of September to the 17th of October only one Chinese holiday takes place. Since the 15th to 17th of September falls outside of these dates, the Chinese holiday week from the 1st to the 7th of October is the only holiday that could potentially influence the data.

As can be seen in Figure 3.6 Task 2 is heavily affected by the holiday week. The somewhat regular pattern before and after the week is completely disrupted and changed during the week. Only tollgate 3 with direction Exit has been plotted, showing a much lower volume of traffic for this week. However, all tollgate-direction pairs except for 3:Entry show significant changes during that same week.

Tollgate 2 with direction Entry and tollgate 1 with direction Exit both experience a decline in traffic volume during the week, just as 3:Exit. Tollgate 1 with direction Entry is the only pair that shows an increase in traffic volume during the holiday week. The plots for the rest of the pairs can be found in Appendix B.

For Task 2 the test data is collected from the same period of time as for Task 1, from the 18th to the 24th of October. This means that for each tollgate-direction pair there should be 504 20-minute intervals. The distribution between existing and missing points in this time interval is as shown in Table 3.5.

Constricting the data points to the chosen eight hours, 06:00-10:00 and 15:00-19:00, yields the distribution described in Table 3.6, where for each tollgate-direction pair there should be 168 data points.

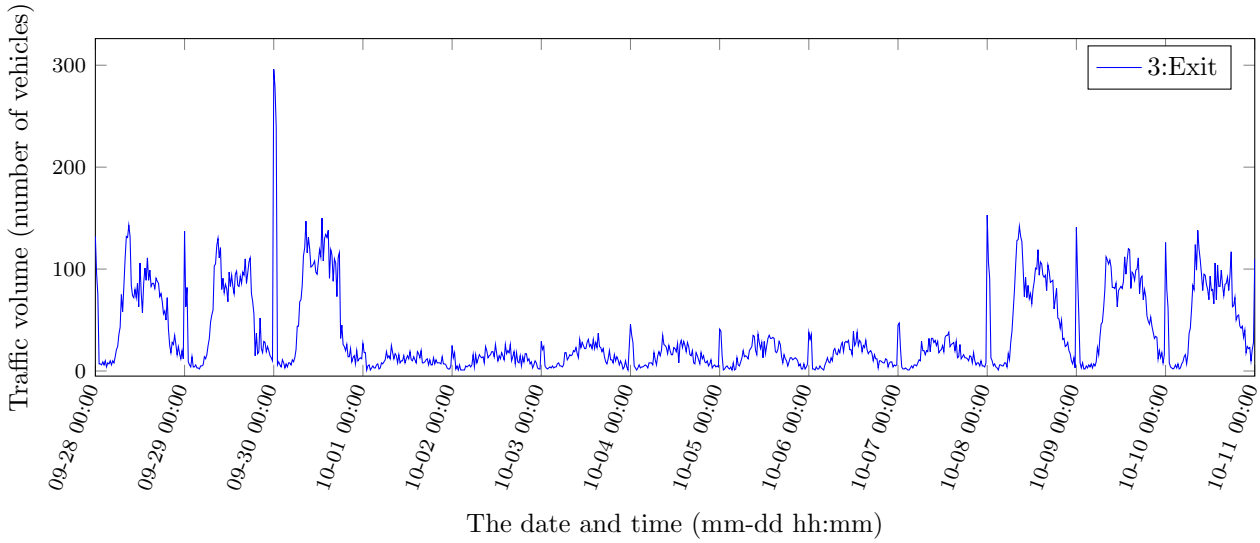

Figure 3.6: Task 2 data, tollgate 3 with direction Exit, from the 28th of September to the 10th of October, showing the holiday week and three regular days before and after for reference.

Table 3.5: The number of existing and missing data points of each tollgate-direction pair in the test week.

Tollgate:Direction	**Existing**	**Missing**	**Missing (%)**
1:Entry	504	0	0.00%
1:Exit	504	0	0.00%
2:Entry	430	74	14.68%
3:Entry	504	0	0.00%
3:Exit	503	1	0.20%

The Task 2 test data set is again more complete than the Task 1 data set. With 2 520 available data points during the whole test week, 504 points for each pair, Task 2 is only missing 75 points or 2.98% of the points the entire week. Additionally, the Task 2 data set is missing no data points during the eight critical hours of the day of the test week as shown in Table 3.6.

3.3 Weather

In addition to the traffic data, weather data is collected every third hour from the 1st of July to the 24th of October. However, since traffic data is only available from the 19th of July and the 19th of September for Task 1 and 2 respectively, the weather data before the 19th of July is ignored. The weather data includes seven different variables, listed in Table 3.7 with their respective units of measurement.

Since the weather data has been collected every third hour there should be eight data points per day. From the 19th of July to the 24th of October there are 98 days, thus there should be 784 (= 98 days∗8 points/day) three hour data points in total. Out of these 784 data points, eight are partly corrupted and ten are missing completely. The 10th of October is missing completely from the data, as well as the last three hours of the 29th of September and the first three hours of the 30th of September. For the partly corrupted points, it is the wind direction that contains the value 999 017 instead of a valid value in $[0, 360)$ degrees.

Table 3.6: The total number of existing and missing data points of each tollgate-direction pair in the critical eight hours (06:00-10:00 and 15:00-19:00) of each day in the test week.

Tollgate:Direction	**Existing**	**Missing**	**Missing (%)**
1:Entry	168	0	0.00%
1:Exit	168	0	0.00%
2:Entry	168	0	0.00%
3:Entry	168	0	0.00%
3:Exit	168	0	0.00%

Table 3.7: The available weather parameters and their respective units of measurement.

Variable	**Unit**
Pressure	hPa
Sea pressure	hPa
Temperature	°C
Relative humidity	%
Precipitation	mm
Wind speed	m/s
Wind direction	°

4 Method

This chapter describes the method used to design, develop and evaluate different neural network designs. The design of the different networks are first described in Section 4.1. Different methods to fill in missing data and prepare input is then explored in Section 4.2, followed by network implementation details in Section 4.3. The design of methods to optimize network parameters is finally described in Section 4.4.

4.1 Network designs

Two different types of neural networks were designed for comparison, one containing a stacked autoencoder (SAE), and a shallow neural network (SNN). For the networks containing an SAE, an additional hidden layer was added after the SAE before the output layer, creating the complete predictor. The structure of the SAE networks predicting travel time is illustrated in Figure 4.1, and the structure of the SNNs predicting travel time is illustrated in Figure 4.2.

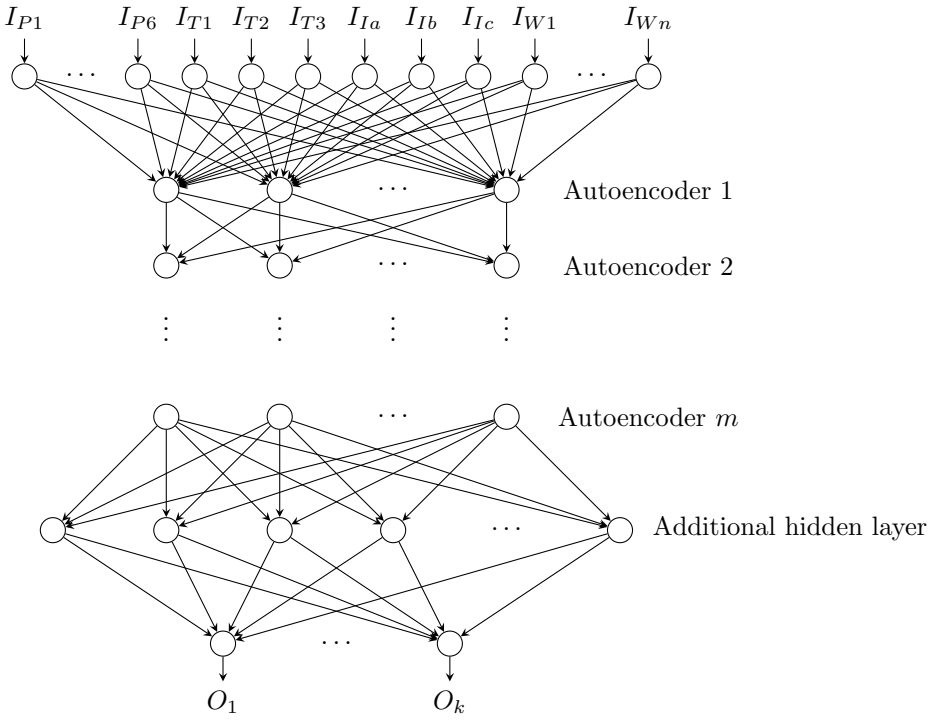

Figure 4.1: The general structure of the stacked autoencoder neural networks.

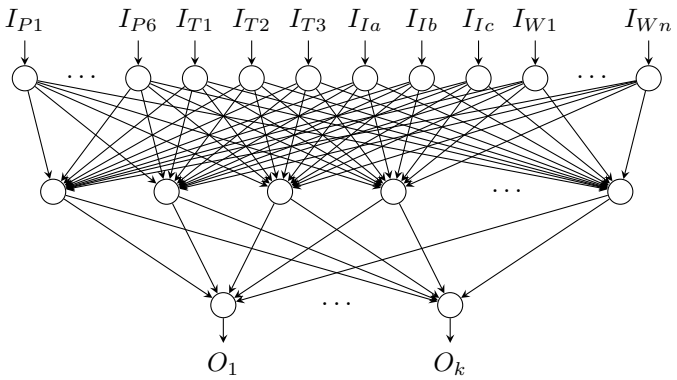

Figure 4.2: The general structure of the shallow neural networks.

The input layer for a network predicting travel time consisted of six nodes $I_{Pi}, i = 1..6$ with values from the six 20-minute time spans prior to the first 20-minute span to be predicted, three one-hot nodes describing the tollgate I_{T1}, I_{T2}, I_{T3}, three one-hot nodes describing the intersection I_{Ia}, I_{Ib}, I_{Ic}, and zero to seven nodes containing weather values $I_{Wi}, i = 1..7$. The input node for wind direction treats the direction as a real valued number. To ease the implementation, no additional effort was made to indicate that the number is periodic. To evaluate whether weather information helped improve the prediction, different networks with different weather data as input were designed. For a network predicting volume, the input layer consisted of the same input nodes as for travel time, except for the nodes describing intersections which were replaced with one binary node describing the direction from the tollgate I_{Di}.

The output layer consisted of one, two, three or six output nodes, each node describing a prediction for a 20-minute span. The first output node O_1 described the value for the first 20-minute span following the input values. The following output nodes O_i described the value for the 20-minute span following the previous output node O_{i-1}. A network with six output nodes predicted the full two hours, while networks containing fewer output nodes made multiple predictions where the output of the first prediction was standardized and used as input to the next prediction. An example of this process is illustrated in Figure 4.3, where a network with three output nodes predicts the two hour span $t + i, i = 0, 20, 40, 60, 80, 100$.

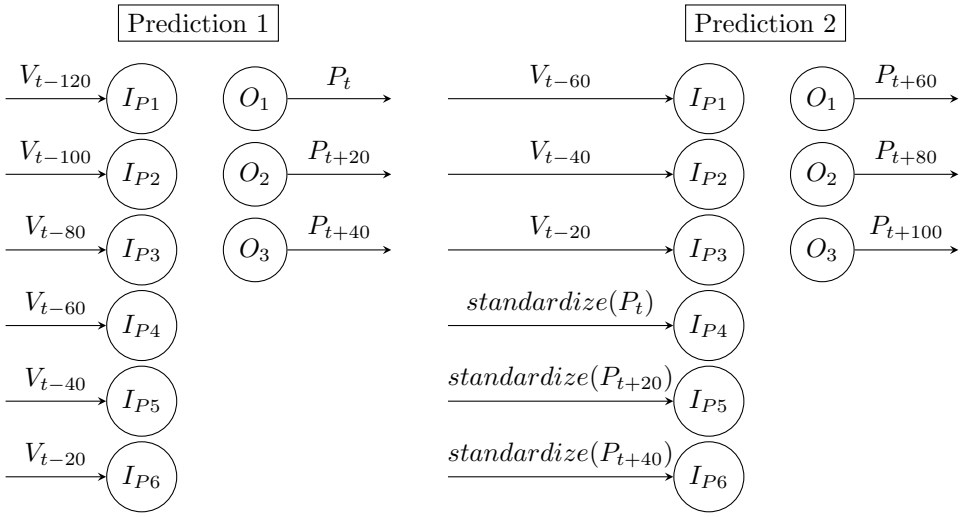

Figure 4.3: How two hour predictions were made using networks with three output nodes.

The SAE networks had an SAE with varying amounts of layers. Every layer in an SAE had the same amount of nodes with every node using the softsign activation function in the encoder. The softsign activation function was chosen due to the performance in [18]. While pretraining, the decoder used the softsign activation function with exception for the first autoencoder. The identity function was used so the first autoencoder could easily recreate values outside the $(-1, 1)$ span that the softsign function has. To calculate the loss when pretraining the SAE, mean squared error was used as the loss function, which is different than the error function used for evaluation. A different error function can be chosen while pretraining since the input and output of autoencoders represent abstract features instead of real values, and mean squared error is commonly used when calculating regression loss in autoencoders. The additional hidden layer after the SAE used the identity function as the activation function as this generally performed better in trial and error testing. Supervised finetuning used MAPE as the loss function.

The SNNs had a hidden layer with varying amount of nodes, and used softsign as the activation function. Comparisons were made between softsign and the identity function. Softsign performed better in general, albeit resulted in big errors with specific network parameters. MAPE was used as the loss function.

4.2 Data preparation

Due to the relatively small dataset that was provided, it was decided to artificially fill in missing data points and evaluate if this improved performance of the networks. Since weather data was almost complete as described in Chapter 3, no evaluation of methods to fill in the weather data was done, and linear interpolation was used to

fill in the missing data in every case. Where traffic data was missing, five naive methods to fill in missing data points in traffic data were chosen as follows:

Mean: Fill in with mean value of all existing values in that 20-minute time span for that specific tollgate-intersection or tollgate-direction pair.

CloseMean: Fill in with mean value of previous 20-minute time span and next 20-minute time span for that specific tollgate-intersection or tollgate-direction pair.

LinearInterpolation: Fill in value using linear interpolation on the closest previous and next 20-minute values.

Lowest: For travel time data, fill in with the lowest travel time found in the dataset for that specific tollgate-intersection pair. For volume data, fill in with a volume of 1.

Random: Fill in with a value randomly chosen from a Gaussian distribution where the mean and standard deviation is calculated from all existing data points for that 20-minute time span for that specific tollgate-intersection or tollgate-direction pair.

The first method *Mean* calculated the mean value for all 20-minute segments for all different tollgate-intersection or tollgate-direction pairs over all data used for training and validation. This value was then used for every missing data point in that segment.

The second method *CloseMean* calculated the mean value of the previous and next 20-minute segment for the specified tollgate-intersection or tollgate-direction pair. Two exceptions were made for the first and last data points respectively. If the first segment in the dataset was missing, the mean was instead calculated by taking the mean of the next two 20-minute segments. If these segments were not available, the *Mean* method was instead used to fill in the first data point. Likewise, if the last segment in the dataset was missing, the mean was calculated by taking the mean of the previous two 20-minute segments. Segments were calculated in order, meaning that the previous 20-minute segment would always be available, with the exception for the very first segment.

The third method *LinearInterpolation* used linear interpolation to fill in the missing value. The interpolant was calculated using the closest previous available value, along with the closest next available value. In cases where no previous or next value was available, the *Mean* method was instead used to fill in the data point. Linear interpolation is described in equation 4.1, where v_{prev} and v_{next} are the values for the previous and next available values respectively, t_{prev} and t_{next} are the timestamps for the previous and next available values respectively, t is the timestamp for the value to be filled in, and v is the value filled in.

$$v = (t - t_{prev}) \frac{v_{next} - v_{prev}}{t_{next} - t_{prev}} + v_{prev} \tag{4.1}$$

The forth method *Lowest* was based on the assumption that a missing data point described a 20-minute segment where no cars passed the detectors. For travel time data, this method found the lowest travel time for that specific tollgate-intersection pair, assuming that no congestion would allow the travel time to be as low as possible. For volume data, no cars passing the detectors described a volume of 0, but to preserve integrity when calculating the MAPE the lowest positive non-zero integer 1 was instead chosen to fill in the data. This method was not used in evaluation as it did not perform well for travel time prediction.

The fifth method *Random* assumed that the values were distributed according to a Gaussian distribution at that 20-minute segment for that specific tollgate-intersection or tollgate-direction pair. The mean for the distribution was calculated by calculating the mean for every 20-minute segment and tollgate-intersection or tollgate-direction pair, over all values available for those segments in the training and validation data. The standard deviation for the distribution was calculated over the same values as the mean. A random value picked from that distribution was then used to fill in missing data points.

A threshold for when to exclude traffic data completely while training was also added. This was specified as a percentage of missing input and output values, where if the amount of missing values was above the threshold, the aggregated data point would be excluded. The threshold was implemented to evaluate if a network would perform different if part of the training data was artificially created instead of collected naturally. An example would be a network with three output nodes and a threshold of 33%, one of the tested values. The total number of nodes would be nine, as there were six input nodes describing the previous two hour span. If there was

missing values for three or fewer nodes, these would be filled in by the specified method. Otherwise, the data would not be filled in and that aggregated data point would not be used when training.

Before training the network, traffic and weather input data was standardized. For traffic values, the mean and standard deviation was calculated over all training and validation values, and the same values were used to standardize the input data when doing a prediction. Mean and standard deviation for weather data was calculated the same way, calculating one mean and standard deviation for every different weather attribute.

4.3 Implementation

The neural networks and the testing suite were implemented in Python using the Tensorflow library. The initial SAE network implementation was based on a blog post by Chris Green [3] and the corresponding code provided at GitHub [4]. Raw data was stored in a SQLite database to make sorting and aggregation easier using prepared SQL statements. Standardization along with artificially filling in missing data was performed at runtime before training the network, as it was easier to implement and the additional computation added an insignificant increase in running time. All networks used the Tensorflow implementation of the Adam optimizer.

Training data was passed to the network in batches, where each entry in a batch was chosen uniformly at random from the available entries. Since every batch contained entries chosen randomly from all available entries, a training epoch didn't guarantee that every entry was used exactly once. After every epoch, validation error was calculated, and if the validation error was lower than for all previous epochs, a Tensorflow checkpoint was created. The checkpoint was a file containing all network variables, allowing the network variables to be set to that specific state at a later point in time. If the validation error didn't decrease for a specific amount of epochs, the network was reset to the latest saved checkpoint before calculating the error on the test data. A visualization of the training process is depicted in Figure 4.4.

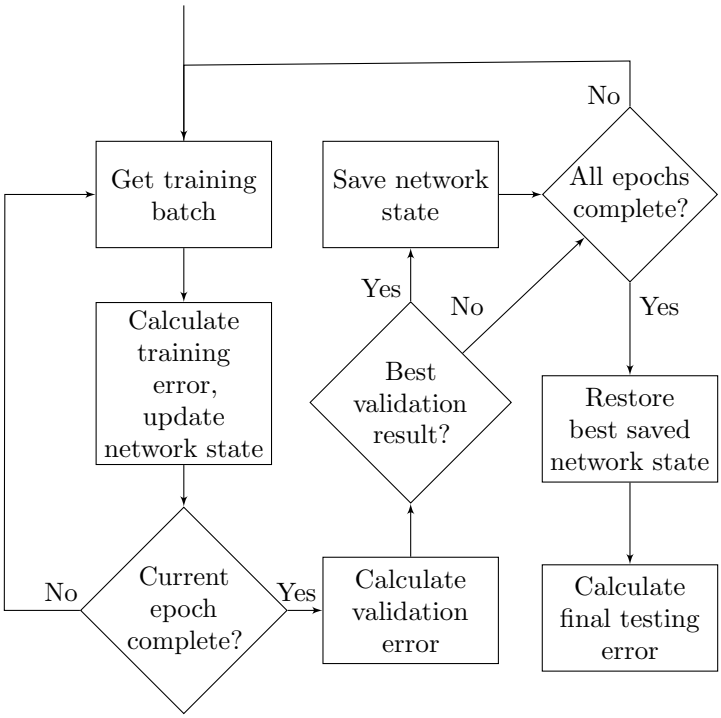

Figure 4.4: Network training flowchart.

Two different testing errors were calculated using MAPE. The data was calculated over two dimensions: intersection-tollgate pairs or tollgate-direction pairs for travel time prediction and volume prediction respectively, along with all 20-minute time spans. The MAPE calculation used for two dimensions is described in equation 4.2, where N is the number of pairs (intersection-tollgate or tollgate-direction), T is the number of 20-minute

time spans, $\boldsymbol{y}$ are the actual values, and $\hat{\boldsymbol{y}}$ are the predicted values.

$$MAPE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{100}{N} \sum_{n=1}^{N} \left(\frac{1}{T} \sum_{t=1}^{T} \left| \frac{y_{tn} - \hat{y}_{tn}}{y_{tn}} \right| \right) \tag{4.2}$$

The primary error, which was used for evaluating the networks, was calculated using the test data on the time spans specified in the problem description; every day between 08:00-10:00 and 17:00-19:00 where missing values were dropped from the calculation. A secondary error was calculated on the test data similar to the primary, but the time range of test data was the same as when training the network. This means that the secondary error value was the same as the primary value for networks trained using only 08:00-10:00 and 17:00-19:00 data. The secondary error value was used to identify if the network performed significantly better on data outside the smaller time span.

For SAE networks, every layer in the SAE was trained in sequence and Tensorflow checkpoints were used to save the network state with the lowest validation. When a layer was trained to over fitting, the network state was restored to the checkpoint before training the next layer. To calculate the KL-divergence when applying the sparsity constraint, the average activation of neurons were scaled to $(0, 1)$ using the function $y = \frac{1+x}{2}$, since activation using softsign gives values in $(-1, 1)$.

4.4 Parameter optimization

To optimize the network parameters, tests were designed based on Taguchi's design of experiments method. During development it was discovered that training a network to the point of overfitting was a relatively fast (from minutes to 24 hours) process due to the small dataset. There were a lot of network parameters available for testing in the SAE networks: 22 for travel time prediction and 23 for volume prediction networks. Taking training duration into account, it was decided to use the L_{64} orthogonal array for SAE network parameter optimization, allowing 21 different features at four levels each. For SNNs, 16 and 17 parameters were available for testing in travel time predictions and volume predictions respectively. It was decided to use the L_{64} orthogonal array for SNN parameter optimization due to allowing four feature levels for parameters, while other considered designs (L_{16}, L_{27}) did not allow sufficient levels. The first and second half of the L_{64} array is depicted in Appendix A, Table A.5 and Table A.6 respectively.

Features and level values were chosen based on results of trial-and-error testing during development of the networks. For SAE networks features and feature levels were the same for both travel time prediction and volume prediction. 21 features were chosen, and the full set of features and feature levels for SAE networks are described in Table 4.1.

The SNNs had a different feature set than the SAE networks, and the networks predicting volume had one additional feature that the travel time networks did not have. 16 features were chosen for travel time prediction, but the L_{64} orthogonal array was still used to test sufficient features and levels. Since volume is measured with integers, the output of volume networks is rounded, and different methods for rounding was evaluated in volume SNNs. This added a 17th feature for SNNs predicting traffic volume. Feature numbers were chosen to ease the testing in the testing suite, unused features contained dummy values that did not impact network performance. The full set of features and feature levels for SNNs are described in Table 4.2.

Table 4.1: Features and feature levels evaluated for stacked autoencoder neural networks.

Feature	**Description**	**Level 1**	**Level 2**	**Level 3**	**Level 4**
1	Number of output nodes	1	2	3	6
2	Learning rate	0.001	0.01	0.1	1
3	Percentage of data for training	80%	85%	90%	95%
4	Time of day to include in training data	All hours	06-21	06-12, 15-21	08-10, 17-19
5	Percent of missing values allowed	0%	33%	66%	100%
6	Fill function to use for missing values	Mean	CloseMean	Random	LinearInterpolation
7	Using pressure as an input	true	true	false	false
8	Using sea pressure as an input	true	true	false	false
9	Using wind direction as an input	true	true	false	false
10	Using wind speed as an input	true	true	false	false
11	Using temperature as an input	true	true	false	false
12	Using relative humidity as an input	true	true	false	false
13	Using precipitation as an input	true	true	false	false
14	Exclude holidays in the training data	true	true	false	false
15	Number of autoencoders	3	4	5	6
16	Number of nodes in every autoencoder	12	24	36	48
17	Number of nodes in last hidden layer	24	96	384	768
18	Sparsity constant value	0	0.01	0.1	1
19	Sparsity parameter	0.1	0.2	0.4	0.6
20	Batch size pretraining	100	500	1000	2000
21	Batch size finetuning	100	500	1000	2000

Table 4.2: Features and feature levels evaluated for shallow neural networks.

Feature	**Description**	**Level 1**	**Level 2**	**Level 3**	**Level 4**
1	Number of output nodes	1	2	3	6
2	Learning rate	0.001	0.01	0.1	1
3	Percentage of data for training	80%	85%	90%	95%
4	Time of day to include in training data	All hours	06-21	06-12, 15-21	08-10, 17-19
5	Percent of missing values allowed	0%	33%	66%	100%
6	Fill function to use for missing values	Mean	CloseMean	Random	LinearInterpolation
7	Using pressure as an input	true	true	false	false
8	Using sea pressure as an input	true	true	false	false
9	Using wind direction as an input	true	true	false	false
10	Using wind speed as an input	true	true	false	false
11	Using temperature as an input	true	true	false	false
12	Using relative humidity as an input	true	true	false	false
13	Using precipitation as an input	true	true	false	false
14	Exclude holidays in the training data	true	true	false	false
15	Number of nodes in the hidden layer	24	96	384	768
16	Batch size	100	500	1000	2000
(17)	Rounding method	Round	Round	Ceiling	Floor

5 Results

This chapter describes the results obtained during the course of the project. In Section 5.1 the results of the Taguchi tests are summarized and in Section 5.2 the final results of the SAE networks and SNNs are presented. Figure 5.1 depicts how experiments were conducted and results obtained. The process was repeated four times: two for both network architectures in combination with the two different tasks.

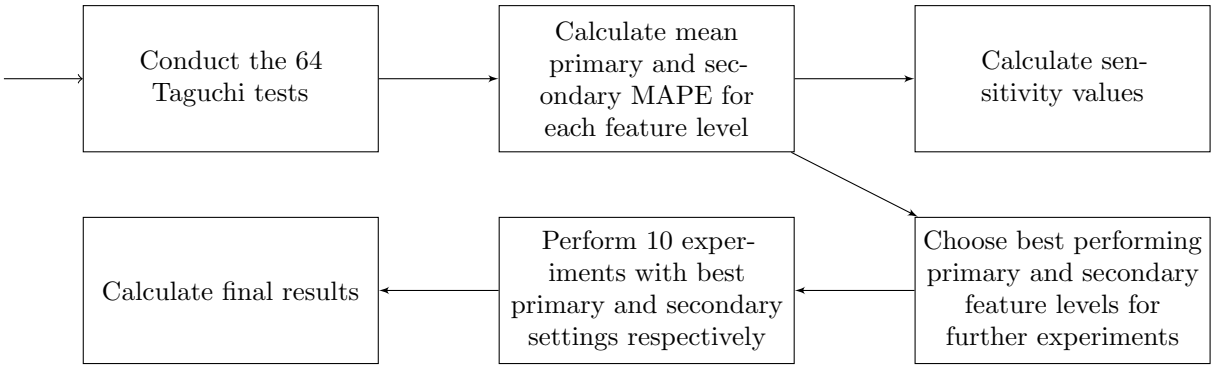

Figure 5.1: How experiments were conducted and results obtained.

5.1 Taguchi tests

In total the Taguchi method was applied four times, with 64 tests each. Once for each combination of prediction, travel time or volume, and neural network, SAE neural network (SAENN) or SNN. Features and feature levels are as described in Section 4.4, Table 4.1 and Table 4.2. The mean MAPE has been calculated for each Feature and level in each of the Tables 5.1, 5.2, 5.3, 5.4, 5.6, 5.7, 5.8 and 5.9. For example, the primary MAPE of all task 1 SAE Taguchi tests with feature 1 set to level 1 has the mean value found in Table 5.1 **F1**:**L1**. The best performing levels of each feature in terms of primary and secondary MAPE have been marked in bold.

5.1.1 Travel time prediction

For travel time prediction the primary and secondary MAPE results of the SAENN Taguchi tests are as described in Tables 5.1 and 5.2 respectively. The primary and secondary MAPE results of the SNN Taguchi tests are as described in Tables 5.3 and 5.4 respectively.

For features 7 through 14, which denote whether certain weather data or holiday data is included, levels 1 and 2 represent the same value, true, and levels 3 and 4 represent the same value, false. Consequently the mean MAPE of levels 1 and 2 and levels 3 and 4 have been calculated respectively.

Feature 19, sparsity parameter, is dependent on feature 18, the sparsity constant. For this reason only tests where the sparsity constant was not zero (i.e., not level 1) were considered when calculating the average MAPE results for feature 19. When the sparsity constant is set to zero, the sparsity parameter does not affect the network in any way. The same reasoning is used when calculating the results of feature 6, the fill function, which is dependent on feature 5, the percent of missing values allowed in each data point. When feature 5 is set to zero percent feature 6 does not affect the network in any way. Therefore, when calculating the MAPE for feature 6 only tests where feature 5 was not set to level 1 were considered.

In the primary MAPE results of the SAE network there is no value below 21%, while for the secondary MAPE the best results are all below 21%. The grand mean of the primary MAPE, the mean of all recorded primary MAPE results, is 21.92% and the grand mean of the secondary MAPE is 20.83%.

For the SNN none of the primary MAPE values are less than 21% while a majority of the best secondary MAPEs are. The grand mean of the primary MAPE is 22.11% and the grand mean of the secondary MAPE is 21.13%.

Table 5.1: Stacked autoencoder feature levels and their corresponding impact on the results on the primary MAPE for travel time prediction.

	L1	L2	L3	L4
F1	22.08%	22.10%	**21.66%**	21.84%
F2	**21.84%**	22.10%	21.86%	21.88%
F3	21.84%	22.18%	22.15%	**21.52%**
F4	23.45%	21.46%	21.55%	**21.22%**
F5	21.80%	**21.41%**	21.84%	22.64%
F6	23.53%	21.51%	21.59%	**21.21%**
F7	**21.87%**		21.97%	
F8	**21.80%**		22.04%	
F9	**21.75%**		22.09%	
F10	**21.77%**		22.07%	
F11	**21.85%**		21.99%	
F12	**21.88%**		21.96%	
F13	**21.49%**		22.35%	
F14	**21.85%**		21.99%	
F15	**21.34%**	22.21%	22.02%	22.11%
F16	**21.52%**	21.83%	22.25%	22.09%
F17	22.22%	**21.54%**	22.15%	21.77%
F18	**21.54%**	21.81%	22.05%	22.28%
F19	22.38%	**21.47%**	22.68%	21.65%
F20	**21.23%**	22.23%	22.56%	21.67%
F21	22.12%	21.87%	21.96%	**21.74%**

Table 5.2: Stacked autoencoder feature levels and their corresponding impact on the results on the secondary MAPE for travel time prediction.

	L1	L2	L3	L4
F1	21.35%	20.75%	**20.52%**	20.68%
F2	21.30%	20.87%	**20.49%**	20.65%
F3	21.18%	20.99%	20.78%	**20.35%**
F4	23.23%	**18.57%**	20.28%	21.22%
F5	21.13%	**20.09%**	20.69%	21.39%
F6	22.62%	20.21%	20.16%	**19.91%**
F7	**20.68%**		20.97%	
F8	**20.59%**		21.06%	
F9	**20.56%**		21.09%	
F10	**20.55%**		21.10%	
F11	**20.74%**		20.91%	
F12	**20.82%**		20.83%	
F13	**20.20%**		21.46%	
F14	**20.78%**		20.87%	
F15	**19.96%**	21.13%	21.17%	21.04%
F16	**20.20%**	20.64%	21.43%	21.04%
F17	21.23%	**20.45%**	21.16%	20.46%
F18	**20.32%**	20.73%	21.19%	21.06%
F19	21.48%	**20.36%**	21.71%	20.43%
F20	**19.90%**	21.25%	21.62%	20.54%
F21	20.72%	20.83%	21.05%	**20.70%**

Table 5.3: Shallow neural network feature levels and their corresponding impact on the results on the primary MAPE for travel time prediction.

	L1	L2	L3	L4
F1	**21.55%**	22.00%	21.92%	22.99%
F2	**21.18%**	21.93%	22.32%	23.03%
F3	21.54%	23.09%	22.37%	**21.45%**
F4	23.24%	**21.41%**	21.61%	22.21%
F5	21.84%	**21.42%**	22.41%	22.78%
F6	24.25%	21.41%	21.99%	**21.18%**
F7	**21.86%**		22.37%	
F8	22.16%		**22.07%**	
F9	22.13%		**22.10%**	
F10	22.14%		**22.08%**	
F11	22.32%		**21.91%**	
F12	22.18%		**22.05%**	
F13	**21.76%**		22.47%	
F14	**22.03%**		22.20%	
F15	**21.42%**	22.32%	21.82%	22.90%
F16	22.08%	22.10%	22.25%	**22.03%**

Table 5.4: Shallow neural network feature levels and their corresponding impact on the results on the secondary MAPE for travel time prediction.

	L1	L2	L3	L4
F1	**20.37%**	21.47%	20.68%	22.01%
F2	**20.05%**	20.98%	21.55%	21.95%
F3	**20.30%**	21.93%	21.48%	20.82%
F4	23.02%	**18.87%**	20.44%	22.21%
F5	20.82%	**20.75%**	21.21%	21.75%
F6	23.39%	20.63%	20.69%	**20.24%**
F7	**20.81%**		21.46%	
F8	**21.09%**		21.18%	
F9	**21.02%**		21.25%	
F10	21.14%		**21.12%**	
F11	21.22%		**21.05%**	
F12	21.37%		**20.90%**	
F13	**20.70%**		21.57%	
F14	**21.00%**		21.27%	
F15	**20.44%**	21.25%	20.83%	22.00%
F16	21.20%	**20.98%**	21.32%	21.03%

In order to make it possible to analyze each feature's impact on the network performance, a sensitivity study was performed on the results. For each feature the grand mean of the corresponding primary or secondary MAPE was subtracted from each level's MAPE value, the results of which were squared before being summed to one value. The resulting values are presented in Table 5.5, where higher values indicate that the corresponding feature impacts the network performance to a greater extent. The three highest values are shown in bold and the three lowest values are underlined, for each network, SAE or SNN, and MAPE, primary (1) or secondary (2).

From Table 5.5 it can be seen that for the SAE network, features 4, 6 and 13 have the biggest impact on the performance, both according to primary and secondary MAPE. Feature 4 describes the time of day to include in the training data, feature 6 describes what fill function is used and feature 13 describes whether or not precipitation was used as input to the network. The fill function used has the biggest impact on the primary MAPE while what time of the day to include in the data has the biggest impact on the secondary MAPE. For both the primary and secondary MAPE, feature 7 and feature 21 are in the three lowest sensitivity values. Feature 7 describes whether or not pressure was used as an input to the network and feature 21 describes the batch size used for finetuning of the network. For the primary MAPE the features with the lowest impact also include feature 2, what learning rate was used to train the network. In addition to feature 7 and 21, feature 14 has one of the lowest impacts for the secondary MAPE. Feature 14 describes whether or not holidays were included as input data.

For the SNN features 2, 4 and 6 have the biggest impacts on performance according to the sensitivity study both for primary and secondary MAPE. As already mentioned feature 2 describes the learning rate used, feature 4 describes what time of day were included in the data and feature 6 describes what fill function was used. Feature 6 has the biggest impact on the primary MAPE while feature 4 has the biggest impact on the secondary MAPE. The least influential features are also shared between both the primary and secondary MAPE; features 8, 14 and 16. Feature 8 describes whether or not to use sea pressure as input to the network, feature 14 describes whether or not to include holidays in the input data and feature 16 describes what batch size was used during training.

Table 5.5: Sensitivity study results for travel time prediction.

	Stacked autoencoder networks		**Shallow networks**	
	Sensitivity 1, 10^{-5}	**Sensitivity 2,** 10^{-5}	**Sensitivity 1,** 10^{-5}	**Sensitivity 2,** 10^{-5}
F1	1.33	3.99	11.35	16.76
F2	<u>0.46</u>	3.69	**17.85**	**20.22**
F3	2.91	3.77	17.80	15.46
F4	**31.87**	**113.20**	**20.29**	**103.49**
F5	7.98	9.69	10.90	6.31
F6	**33.55**	**48.94**	**59.71**	**63.48**
F7	<u>0.36</u>	<u>1.59</u>	2.76	4.30
F8	1.56	4.60	<u>2.06</u>	<u>2.64</u>
F9	2.12	7.31	2.68	3.81
F10	1.68	6.05	3.73	4.51
F11	1.08	3.56	3.61	5.01
F12	9.71	9.57	2.74	4.81
F13	**12.76**	**21.23**	5.39	7.65
F14	0.95	<u>2.48</u>	<u>0.64</u>	<u>1.92</u>
F15	4.67	10.02	12.23	13.39
F16	3.06	8.38	<u>0.29</u>	<u>0.74</u>
F17	3.07	5.45		
F18	3.02	4.50		
F19	10.70	15.77		
F20	10.46	17.51		
F21	<u>0.74</u>	<u>0.74</u>		

5.1.2 Volume prediction

The primary and secondary MAPE results of the SAENN Taguchi tests for volume prediction are as shown in Tables 5.6 and 5.7. The primary and secondary MAPE results of the SNN Taguchi tests for volume prediction are shown in Tables 5.8 and 5.9. As with travel time prediction, features 7 through 14 are consolidated as two levels represent the same value. Consequently the average MAPEs of levels 1 and 2 and levels 3 and 4 have been calculated respectively.

For feature 17 of the SNN, rounding method used, levels 1 and 2 represent the same method. Therefore the mean MAPEs of levels 1 and 2 have been calculated jointly.

Alike volume prediction testing, feature 6, fill function, is dependent on feature 5, percent of missing values allowed. The same is true for feature 19, sparsity parameter, which is dependent on feature 18, sparsity constant. Therefore only tests where feature 5 and feature 18 was not set to level 1 were considered when calculating the average MAPE results of feature 6 and 19 respectively.

For the primary MAPE of the SAE network tests only four of the best results are below 22%. The grand mean of all recorded primary MAPE values is 23.36%. Only five of the best results of the secondary MAPE are below 26% and four of those are more than 25%, while one is 21.85% for feature 4, the time of day to include in the data. The grand mean of the secondary MAPE is 26.97%.

Among the best primary MAPE results of the SNN Taguchi tests, none is below 21% and all are less than 25%. The grand mean of the primary MAPE is 24.98% Only one of the best secondary MAPE results is below 25%, and only one exceeds 30%. The grand mean of the secondary MAPE is 30.29%.

Table 5.6: Stacked autoencoder feature levels and their corresponding impact on the results on the primary MAPE for volume prediction.

	L1	L2	L3	L4
F1	23.48%	**22.43%**	23.75%	23.75%
F2	24.46%	23.85%	**21.81%**	23.31%
F3	22.67%	**21.23%**	24.13%	25.40%
F4	25.90%	23.28%	22.39%	**21.85%**
F5	**21.97%**	24.17%	23.51%	23.77%
F6	23.60%	**23.07%**	24.65%	23.94%
F7	24.06%		**22.65%**	
F8	24.11%		**22.60%**	
F9	23.37%		**23.35%**	
F10	**23.34%**		23.37%	
F11	23.71%		**23.00%**	
F12	24.03%		**22.68%**	
F13	23.50%		**23.21%**	
F14	23.53%		**23.18%**	
F15	23.13%	23.29%	**22.38%**	24.62%
F16	**22.30%**	23.75%	23.72%	23.65%
F17	23.71%	23.52%	23.15%	**23.05%**
F18	23.18%	23.48%	23.98%	**22.79%**
F19	23.24%	24.93%	**22.50%**	22.99%
F20	23.28%	24.02%	23.27%	**22.85%**
F21	24.41%	23.90%	**22.20%**	22.91%

Table 5.7: Stacked autoencoder feature levels and their corresponding impact on the results on the secondary MAPE for volume prediction.

	L1	L2	L3	L4
F1	27.06%	**26.21%**	27.60%	27.04%
F2	28.26%	26.76%	**25.60%**	27.28%
F3	26.21%	**25.00%**	27.63%	29.06%
F4	35.74%	24.80%	25.52%	**21.85%**
F5	**25.61%**	27.50%	27.75%	27.04%
F6	26.90%	**26.67%**	28.64%	27.51%
F7	27.77%		**26.18%**	
F8	27.79%		**26.16%**	
F9	27.09%		**26.86%**	
F10	27.19%		**26.76%**	
F11	27.08%		**26.87%**	
F12	27.61%		**26.34%**	
F13	27.20%		**26.74%**	
F14	**26.35%**		27.60%	
F15	26.89%	27.02%	**26.14%**	27.85%
F16	**26.91%**	27.03%	26.94%	27.01%
F17	27.52%	**26.62%**	26.79%	26.96%
F18	27.42%	27.14%	27.06%	**26.28%**
F19	26.71%	27.65%	**25.93%**	27.01%
F20	26.89%	**26.73%**	26.93%	27.34%
F21	27.77%	27.66%	**26.10%**	26.36%

Table 5.8: Shallow neural network feature levels and their corresponding impact on the results on the primary MAPE for volume prediction.

	L1	**L2**	**L3**	**L4**
F1	27.18%	24.74%	**24.00%**	24.01%
F2	23.47%	24.18%	**21.85%**	30.43%
F3	23.91%	**23.46%**	26.59%	25.97%
F4	28.97%	24.20%	**22.69%**	24.05%
F5	**23.45%**	24.13%	25.53%	26.81%
F6	26.18%	26.26%	25.98%	**23.55%**
F7	**24.81%**		25.15%	
F8	26.38%		**23.58%**	
F9	25.35%		**24.61%**	
F10	25.30%		**24.66%**	
F11	25.00%		**24.97%**	
F12	25.39%		**24.57%**	
F13	**24.91%**		25.05%	
F14	25.39%		**24.58%**	
F15	25.10%	25.59%	**23.63%**	25.61%
F16	26.43%	24.70%	**24.05%**	24.74%
F17	25.65%		**24.18%**	24.44%

Table 5.9: Shallow neural network feature levels and their corresponding impact on the results on the secondary MAPE for volume prediction.

	1	**2**	**3**	**4**
1	30.80%	30.56%	31.78%	**28.00%**
2	28.45%	27.30%	**26.97%**	38.42%
3	**28.81%**	31.18%	29.52%	31.63%
4	43.28%	27.00%	26.81%	**24.05%**
5	**27.83%**	29.66%	32.90%	30.75%
6	33.62%	31.71%	30.82%	**28.26%**
7	**29.98%**		30.59%	
8	30.86%		**29.72%**	
9	31.77%		**28.80%**	
10	**29.98%**		30.59%	
11	30.86%		**29.71%**	
12	31.36%		**29.21%**	
13	**30.05%**		30.52%	
14	30.63%		**29.95%**	
15	29.13%	29.93%	**28.69%**	33.40%
16	31.82%	29.41%	30.64%	**29.28%**
17	31.40%		**28.89%**	29.45%

Table 5.10: Sensitivity study results for volume prediction.

	Stacked autoencoder networks		**Shallow networks**	
	Sensitivity 1, 10^{-4}	**Sensitivity 2,** 10^{-4}	**Sensitivity 1,** 10^{-4}	**Sensitivity 2,** 10^{-4}
F1	1.18	0.99	6.83	7.82
F2	**3.86**	**3.69**	**42.41**	**89.49**
F3	**9.79**	**9.29**	7.06	5.38
F4	**9.70**	**110.01**	**22.65**	**230.43**
F5	2.77	2.76	6.72	13.50
F6	2.17	3.16	6.10	**17.58**
F7	3.56	2.65	2.82	8.56
F8	2.41	2.66	**8.63**	4.71
F9	0.32	0.69	1.40	9.28
F10	0.43	1.13	0.72	3.26
F11	0.76	0.34	3.66	11.27
F12	2.76	1.64	2.76	5.94
F13	0.56	0.23	0.22	1.17
F14	0.67	1.75	1.70	2.85
F15	2.61	1.48	2.60	13.72
F16	1.49	0.01	3.12	4.24
F17	0.29	0.45	2.21	5.61
F18	0.76	0.71		
F19	3.37	1.61		
F20	0.71	0.20		
F21	2.95	2.24		

The same kind of sensitivity study has been performed for volume prediction as for travel time in Section 5.1.1. The results of this study are shown in Table 5.10. The study shows that features 2, 3 and 4 have the biggest impact both on primary and secondary MAPE for the SAE network. These features describe the learning rate used, percentage of data used for training and the time of day to include in the data, respectively. The SAE network features with the smallest impact are 9, 10 and 17 for the primary MAPE. These correspond to using wind direction as input, using wind speed as input and the number of nodes used in the last hidden

layer. Features 13, 16 and 20 have the smallest impact on the secondary MAPE. Feature 13 describes whether or not to use precipitation as input data, feature 16 describes the number of nodes used in every autoencoder and feature 20 describes what batch size to use during pretraining.

For the SNN the results are not unanimous, only features 2 and 4, denoting the learning rate and time of day to include in data, are among the three most influential features of both MAPEs. The third feature for the primary MAPE is 8, whether or not to use sea pressure as input. The third feature of the secondary MAPE is 6, what fill function to use. The least influential features of the SNN are 9, 10 and 13 for the primary MAPE. These features describe whether or no to use wind direction, wind speed and precipitation respectively, as input data. The least influential features of the secondary MAPE are 10, 13 and 14, where 14 describes whether or not to exclude holidays in the input data.

5.2 Performance

After performing the Taguchi tests and analyzing the results, the optimal settings for each neural network were specified. Primary and secondary MAPE did not produce the same optimal settings, therefore eight different set of settings were selected to be used for further experiments. One set of settings per task, type of neural network and MAPE. Each of the settings were then tested ten times, by training the corresponding network using those settings and collecting primary and secondary MAPE results. This in turn yields 16 independent results, eight per task.

5.2.1 Travel time prediction

Given the results of the travel time Taguchi tests the optimal settings for the SAE networks and the SNNs are as specified in Table 5.11. The primary settings are derived from the primary MAPE results of the Taguchi tests and the secondary settings are derived from the secondary MAPE results. In the SAENN two features differ and in the SNN four features differ between primary and secondary settings.

After performing ten tests each with the settings specified in Table 5.11, the results are as specified in Tables 5.12 and 5.13. Table 5.12 shows the results of each of the four settings measured using primary MAPE while Table 5.13 shows the results of each of the settings using secondary MAPE. These tables show that the secondary settings yielded better results than the primary settings on all accounts except one, even when measured using the primary MAPE. According to the primary MAPE the SNN using the secondary settings yielded the overall best results. Only the SNN median for the primary settings and the SAENN standard deviation for the secondary settings were better. According to the secondary MAPE the SAENN using secondary settings performed best overall. Only the standard deviation of the SNN using secondary settings is a tiny bit better.

5.2.2 Volume prediction

Given the results of the volume Taguchi tests the optimal settings for the SAE networks and the SNNs are as specified in Table 5.14. Four features differ between the SAENNs primary and secondary settings: using wind speed as input, exclude holidays, nodes in last hidden layer, and pretraining batch size. Five features differ between the primary and secondary settings for the SNNs: number of output nodes, percentage of data for training, time of day to include, using wind speed as input, and batch size.

After performing ten tests each with each of the four settings specified in Table 5.14, the results are as specified in Tables 5.15 and 5.16. The best average, minimum, maximum and standard deviation values are shown in bold. Table 5.15 shows the results of each of the four settings measured using primary MAPE while Table 5.16 shows the results of each of the settings using secondary MAPE. Because the secondary settings for the SNNs only use data from 08:00-10:00 and 17:00-19:00 during the day the secondary MAPE results are effectively the same as the primary MAPE results for these settings. The SNNs performed better than the SAENNs in general, with the secondary settings yielding the best results over all.

Table 5.11: Final settings of the SAENNs and SNNs for task 1, travel time prediction.

	SAENNs		**SNNs**	
	Primary setting	**Secondary setting**	**Primary setting**	**Secondary setting**
Number of output nodes	3		1	
Learning rate	0.001	0.1	0.001	
Percentage of data for training	95%		95%	80%
Time of day to include in training data	08-10, 17-19	06-21	06-21	
Percent of missing values allowed	33%		33%	
Fill function to use for missing values	Interpolation		Interpolation	
Using pressure as an input	True		True	
Using sea pressure as an input	True		False	True
Using wind direction as an input	True		False	True
Using wind speed as an input	True		False	
Using temperature as an input	True		False	
Using relative humidity as an input	True		False	
Using precipitation as an input	True		True	
Exclude holidays in the training data	True		True	
Number of nodes in the hidden layer			24	
Batch size			2000	500
Number of autoencoders	3			
Number of nodes in every autoencoder	12			
Number of nodes in last hidden layer	96			
Sparsity constant value	0			
Batch size pretraining	100			
Batch size finetuning	2000			

Table 5.12: Primary MAPE results of SAENN and SNN primary and secondary settings for task 1.

	Stacked autoencoder networks		**Shallow networks**	
	Primary sett.	**Secondary sett.**	**Primary sett.**	**Secondary sett.**
Mean	20.71%	20.53%	20.40%	**20.39%**
Median	20.67%	20.53%	**20.39%**	20.43%
Minimum	20.23%	20.26%	20.22%	**20.16%**
Maximum	21.28%	20.69%	20.65%	**20.64%**
Standard dev.	0.0036	**0.0012**	0.0015	0.0017

Table 5.13: Secondary MAPE results of SAENN and SNN primary and secondary settings for task 1.

	Stacked autoencoder networks		**Shallow networks**	
	Primary sett.	**Secondary sett.**	**Primary sett.**	**Secondary sett.**
Mean	20.71%	**17.72%**	17.91%	18.13%
Median	20.67%	**17.75%**	17.89%	18.11%
Minimum	20.23%	**17.52%**	17.70%	17.97%
Maximum	21.28%	**17.91%**	18.09%	18.32%
Standard dev.	0.0036	0.0012	0.0014	**0.0012**

Table 5.14: Final settings of the SAENNs and SNNs for task 2, volume prediction.

	SAENNs		**SNNs**	
	Primary setting	**Secondary setting**	**Primary setting**	**Secondary setting**
Number of output nodes	2		3	6
Learning rate	0.1		0.1	
Percentage of data for training	85%		85%	80%
Time of day to include in training data	08-10, 17-19		06-12, 15-21	08-10, 17-19
Percent of missing values allowed	0%		0%	
Fill function to use for missing values	CloseMean		Interpolation	
Using pressure as an input	False		True	
Using sea pressure as an input	False		False	
Using wind direction as an input	False		False	
Using wind speed as an input	True	False	False	True
Using temperature as an input	False		False	
Using relative humidity as an input	False		False	
Using precipitation as an input	False		True	
Exclude holidays in the training data	False	True	False	
Number of nodes in the hidden layer			384	
Batch size			1000	2000
Rounding method			Ceiling	
Number of autoencoders	5			
Number of nodes in every autoencoder	12			
Number of nodes in last hidden layer	768	96		
Sparsity constant value	1			
Sparsity parameter	0.4			
Batch size pretraining	2000	500		
Batch size finetuning	1000			

Table 5.15: Primary MAPE results of SAENN and SNN primary and secondary settings for task 2.

	Stacked autoencoder networks		**Shallow networks**	
	Primary sett.	**Secondary sett.**	**Primary sett.**	**Secondary sett.**
Mean	18.31%	18.47%	18.55%	**15.95%**
Median	18.35%	18.34%	18.61%	**15.87%**
Minimum	17.47%	17.88%	18.10%	**15.53%**
Maximum	19.58%	19.69%	18.94%	**16.38%**
Standard dev.	0.0061	0.0056	**0.0026**	0.0029

Table 5.16: Secondary MAPE results of SAENN and SNN primary and secondary settings for task 2.

	Stacked autoencoder networks		**Shallow networks**	
	Primary sett.	**Secondary sett.**	**Primary sett.**	**Secondary sett.**
Mean	18.31%	18.47%	22.52%	**15.95%**
Median	18.35%	18.34%	22.53%	**15.87%**
Minimum	17.47%	17.88%	22.12%	**15.53%**
Maximum	19.58%	19.69%	22.85%	**16.38%**
Standard dev.	0.0061	0.0056	**0.0026**	0.0029

6 Discussion

The results from the testing in Section 5.2 shows that stacked autoencoder networks performed worse than the shallow neural networks in both travel time prediction and volume prediction for the primary task of predicting during the 8-10 and 17-19 hours. For volume prediction, the SAE networks performed considerably worse than the SNN, while the travel time prediction was closer between the two networks. The bad performance of the SAE networks could be related to the relatively small dataset used for training, as other studies have achieved good performance using networks based on SAEs for traffic prediction ([8, 18]). This could also possibly explain the significantly greater difference in performance between the two tasks, as the network structure for the two different predictions were very similar while the two datasets differed considerably in size, the dataset for volume prediction containing less than a third of the amount of data points compared to the travel time dataset.

While Lv et al. [8] and Yang et al. [18] achieved good performance using SAE networks, the networks in their studies differ from the SAE networks studied in this thesis. The study by Lv et al. used a SAE with a logistic regression layer as the final predictor layer, while our study used an identity function predictor layer. The SAE network studied by Yang et al. was trained with the Levenberg-Marquardt algorithm, instead of the Adam optimizer used in this thesis. It is possible that the SAE networks investigated in this thesis was outperformed due to these differences, and other SAE networks would perform better than the SNN networks.

Using the Taguchi method to help choose parameter levels did cut down a lot of time compared to doing trial-and-error style testing. However, the result could potentially be better using other parameter levels than suggested by the result from the Taguchi tests. In the beginning of the project we noticed that the MAPE had a non-trivial variation when training and testing the same network multiple times, which prompted us to measure mean values of the tests. We did not investigate the cause of this variation, but it could potentially be an effect of the random weight initialization. One could potentially compare different feature levels by their best performance instead of mean performance to receive a different result. It is also notable that the single best results overall was achieved during the Taguchi tests and not in any of the final tests. The best result achieved for travel time prediction was a primary MAPE of 19.99% with a SAENN during Taguchi tests, and for volume prediction the best result was a primary MAPE of 15.41% with a SAENN during Taguchi tests. This could indicate that the different parameters tested were too dependent, and some parameter settings together resulted in an improvement.

The sensitivity analysis gave some great insight in which parameters influenced the result most, but it should be noted that this does not necessarily mean that the feature with the highest sensitivity is what contributes the most to the best result. A parameter can have a high sensitivity value from one of the levels performing vastly worse than the other levels. An example of this could be if the result for the four different feature levels were 20.3%, 20.5%, 20.7% and 30%. The big difference between the last level and the other three would result in a high sensitivity, but that wouldn't mean that the feature is what contributed most to the best result, just that it is important to not choose the wrong level when performing additional tests. With this in mind, analyzing the sensitivity can still give some interesting information.

For travel time prediction, feature 4 (time of day to include) and 6 (fill function) had the highest sensitivity in both the SAE networks and the SNNs. Time of day was expected to have quite a bit of impact on the result, since it drastically changes how much data is available for training and what patterns the data contains, as the pattern is very different in different time spans. Using all 24 hours of the day performed worst in all cases, which makes sense since the night intuitively has another traffic pattern, along with missing data as described in Chapter 3. Sensitivity being higher on the secondary MAPE than the primary MAPE was also expected since the secondary MAPE is calculated over all included times, hence the result is dependent on what patterns in the data is easiest to predict and if they are included.

The third feature with one of the highest sensitivity values of the SAE network for travel time prediction was feature 13, whether or not to use precipitation as input data. The remaining weather data features all have significantly lower sensitivity values. This is not particularly surprising since rain will affect road grip, visibility and thus the behavior of drivers. It may also cause some motorists to choose not to drive at all, thus affecting the traffic pattern considerably. For the SNN the third of the features with the highest sensitivity was feature two, the learning rate used. In general, the learning rate had a greater impact on the SNNs than the SAENNs for both tasks. All weather data features had low sensitivity for the SNN which could imply that for the SNN the network settings were more important than data related settings. It also seems that the SAENNs were mostly impacted by the quality and quantity of data.

For volume prediction, the features with highest sensitivity values for the SAENNs were features 2, 3 and 4

(i.e., the learning rate, percentage of data to use for training, and time of day to include). Feature 2 had a significantly lower sensitivity value indicating that the SAENNs were overall more sensitive to data quality and quantity than network settings. For the SNNs the two most influential features were features 2 and 4 (learning rate and time of day to include), followed by sea pressure for primary MAPE and fill function for secondary MAPE which had significantly lower sensitivity values. The weather data settings had less of an impact on the SAENNs for volume prediction. This could be due to the amount of data being lower for volume prediction than for travel time prediction, which would mean that additional weather data doesn't contribute as much.

For the secondary MAPE of the SNNs the time of day included in the data had a considerably higher sensitivity value than features 2 and 6. This could partly be due to the first level having a mean MAPE of 43%, compared to levels 2, 3 and 4, having a mean MAPE of 27%, 27% and 24% respectively. Since the secondary MAPE is evaluated using the same time span as the training data, the first level uses all 24 hours of the day for evaluation as well as training. Volume prediction has a low amount of data, which could mean that the SNN does not have enough data to learn the pattern occurring during the night, hence the secondary MAPE is significantly lower when evaluating without using the night data.

Looking at the primary MAPE result for the final tests with fixed parameter levels, the best results were in general not obtained with networks using settings chosen to optimize primary MAPE performance, but instead of networks using settings for optimizing secondary MAPE performance. It is not clear why this happens, but it could be due to the test data having a different pattern than the training and validation data, and secondary settings performing better in general while primary settings performs well on the 8-10, 17-19 time span in the training and validation data.

The quantity and quality of training data has a big impact on the performance of neural networks. Using a stacked autoencoder was perhaps not ideal for this purpose, as the dataset was relatively small. For travel time prediction there was a decently long time span from which data was collected, however there was also many data points missing from this time. Most data is missing during the nighttime, but several of the intersection-tollgate pairs are missing a significant amount of data during the whole day. In contrast, for volume prediction the quantity of the data was only a third of the amount of data for task 1, but almost complete. Efforts was made to fill in the data, but the results from the Taguchi testing implies that our chosen methods to fill in the missing data was not sufficient for improving network performance considerably. Hypothetically both the SAENNs and the SNNs would have produced better predictions given a greater and more complete set of data for each task. With the amount of data that we were given the neural networks that we designed over trained fairly quickly. The run time for training each neural network varied between a few minutes and 24 hours depending on network settings. In general the stacked autoencoder neural network took longer to train fully than the shallow neural network. The average run time for the SNNs during Taguchi tests was about five minutes, while the SAENNs had an average run time of about one hour and 28 minutes.

There is an abundance of different settings and approaches used to design neural networks. Because of the time constraint of the study, we were not able to explore each and everyone of these. A few were picked hoping that they would yield the better results. For example, a single SAENN or SNN could have been designed for each intersection-tollgate or tollgate-direction pair. However, as mentioned above the amount of data was limited, dividing it up further would most likely not yield better results. Additionally, two hours is a relatively long time span to make a prediction for. In the cases where fewer than 6 output nodes were used, previous predictions were assumed to be correct while predicting the rest of the time windows. This means that there could be a snowballing effect on the results. If the first prediction is less accurate it is more likely that the following predictions are as well since they rely on less accurate information.

Since the data was from the KDD Cup 2017 competition, the results of this thesis can be somewhat compared to the results submitted to the competition. The best competition result for travel time prediction had a MAPE of 17.48%, and the best result for volume prediction had a MAPE of 12.03% This is considerably better than the result of this thesis with the best travel time prediction result having a MAPE of 19.99%, and volume predicting having a MAPE of 15.41%. The results obtained in this thesis would have placed 180 out of 368 and 20 out of 346 in travel time prediction and volume prediction respectively. This is however not directly comparable, as the competition result was evaluated by predicting the week following the week used for network evaluation in this thesis. The data for the last week was never published, hence couldn't be used for evaluation in this thesis. The competition evaluation also used denoising on submitted predictions, and the specific details of the denoising was never published.

Comparing this work with previous studies where similar tasks were addressed using similar methods, there are a few key differences. Contrary to other studies this work used a stacked autoencoder in combination with a small and somewhat incomplete data set. Additionally, fairly many weather measurements were used when

training the neural networks and making predictions. Furthermore, the effects on prediction performance when including data from different time spans were studied more in depth.

In this project we discovered that Tensorflow is a very powerful and easy to use tool when implementing simple neural network structures such as the shallow neural network. However, it is somewhat more difficult to use when implementing something more complex, such as a stacked autoencoder. We did not need to spend a lot of time implementing the shallow neural network, while the stacked autoencoder implementation took considerably longer. If we had had more previous experience with stacked autoencoders and Tensorflow, the time needed for implementation would probably have been considerably shorter.

7 Conclusion

The result shows that neural networks can be used to quite accurately predict traffic travel time and volume despite relatively small datasets. A stacked autoencoder neural network did not generally perform better than a simple shallow neural network when averages were compared. However, the quantity of data seemed to affect the stacked autoencoder neural networks to a greater extent and the difference between the two network designs' performances was smaller when the bigger dataset was used. This indicates that with a greater dataset a stacked autoencoder neural network could outperform a shallow neural network.

Using the Taguchi approach for design of experiments can cut down the time spent deciding neural network settings, but does not guarantee that optimal settings are chosen. In this study, the best results were obtained while performing the structured Taguchi tests, not when a combination of network settings had been chosen based on the results of the Taguchi tests.

Future studies should be made with a similar approach as in this thesis, but with bigger datasets to see if this would favor stacked autoencoder neural networks over shallow neural networks. Even though the Taguchi method did not prove to be optimal for this project, more experimentation with the method should be done, as other studies have used it successfully to optimize neural network parameters.

References

[1] Andries E Brouwer, Arjeh M Cohen, and Man VM Nguyen. Orthogonal arrays of strength 3 and small run sizes. *Journal of Statistical Planning and Inference* **136**.9 (2006), 3268–3280.

[2] Rasool Fakoor, Faisal Ladhak, Azade Nazi, and Manfred Huber. "Using deep learning to enhance cancer diagnosis and classification". *Proceedings of the Workshop on Role of Machine Learning in Transforming Healthcare at the 30th International Conference on Machine Learning*. 2013.

[3] Chris Green. *Diving Into TensorFlow With Stacked Autoencoders*. URL: `http://cmgreen.io/2016/01/04/tensorflow_deep_autoencoder.html` (visited on 2017-07-23).

[4] Chris Green. *MNIST Digit Classification Using Stacked Autoencoder And TensorFlow*. URL: `https://github.com/cmgreen210/TensorFlowDeepAutoencoder` (visited on 2017-07-23).

[5] Sherif Ishak and Ciprian Alecsandru. Optimizing traffic prediction performance of neural networks under various topological, input, and traffic condition settings. *Journal of Transportation Engineering* **130**.4 (2004), 452–465.

[6] Teruaki Ito and Tomoyuki Hiramoto. A general simulator approach to ETC toll traffic congestion. *Journal of Intelligent Manufacturing* **17**.5 (2006), 597–607.

[7] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] 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 (2015), 865–873.

[9] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes* **72**.2011 (2011), 1–19.

[10] MS Packianather, PR Drake, and H Rowlands. Optimizing the parameters of multilayered feedforward neural networks through Taguchi design of experiments. *Quality and reliability engineering international* **16**.6 (2000), 461–473.

[11] Fabrício José Pontes, Anderson Paulo de Paiva, Pedro Paulo Balestrassi, João Roberto Ferreira, and Messias Borges da Silva. Optimization of Radial Basis Function neural network employed for prediction of surface roughness in hard turning process using Taguchi's orthogonal arrays. *Expert Systems with Applications* **39**.9 (2012), 7776–7787.

[12] Ranjit K Roy. *Design of experiments using the Taguchi approach: 16 steps to product and process improvement*. John Wiley & Sons, 2001.

[13] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[14] Brian L Smith and Michael J Demetsky. Traffic flow forecasting: comparison of modeling approaches. *Journal of transportation engineering* **123**.4 (1997), 261–266.

[15] Wimalin Sukthomya and James Tannock. The optimisation of neural network parameters using Taguchiś design of experiments approach: an application in manufacturing process modelling. *Neural Computing & Applications* **14**.4 (2005), 337–344.

[16] Wimalin Sukthomya and James Tannock. The training of neural networks to model manufacturing processes. *Journal of Intelligent Manufacturing* **16**.1 (2005), 39–51.

[17] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* **11**.Dec (2010), 3371–3408.

[18] Hao-Fan Yang, Tharam S Dillon, and Yi-Ping Phoebe Chen. Optimized structure of the traffic flow forecasting model with a deep learning approach. *IEEE transactions on neural networks and learning systems, in press* (2016).

[19] Weizhong Zheng, Der-Horng Lee, and Qixin Shi. Short-term freeway traffic flow prediction: Bayesian combined neural network approach. *Journal of transportation engineering* **132**.2 (2006), 114–121.

A Appendix A

This appendix contains the L_{64} orthogonal array and results from testing using the Taguchi method described in Section 4.4.

Table A.1: Results of the first 32 tests using stacked autoencoder networks

Test	Task 1 MAPE 1	Task 1 MAPE 2	Task 2 MAPE 1	Task 2 MAPE 2
1	21.27%	20.09%	25.08%	38.16%
2	22.03%	24.28%	32.81%	37.49%
3	27.50%	31.63%	21.82%	31.78%
4	22.49%	23.56%	20.53%	32.48%
5	21.86%	18.64%	22.42%	24.76%
6	21.63%	18.59%	23.97%	25.06%
7	20.94%	18.19%	21.05%	21.31%
8	21.76%	18.85%	22.92%	21.70%
9	23.43%	21.50%	25.49%	29.19%
10	21.58%	20.50%	21.63%	25.30%
11	20.74%	19.17%	20.42%	23.92%
12	21.44%	20.00%	20.04%	24.20%
13	22.77%	22.77%	22.55%	22.55%
14	21.72%	21.72%	22.65%	22.65%
15	21.10%	21.10%	28.27%	28.27%
16	21.03%	21.03%	24.06%	24.06%
17	23.85%	23.17%	19.92%	22.18%
18	22.73%	21.52%	19.13%	21.90%
19	22.04%	20.41%	22.77%	27.31%
20	21.88%	20.51%	21.62%	25.10%
21	23.31%	23.31%	20.96%	20.96%
22	21.36%	21.36%	19.75%	19.75%
23	20.29%	20.29%	19.78%	19.78%
24	20.70%	20.70%	23.05%	23.05%
25	24.27%	24.07%	25.09%	37.77%
26	21.08%	19.66%	26.36%	35.64%
27	23.92%	22.76%	26.90%	34.46%
28	21.34%	19.96%	26.84%	36.96%
29	19.99%	17.45%	22.92%	22.93%
30	24.54%	20.72%	21.79%	22.43%
31	21.62%	18.43%	21.69%	24.48%
32	20.69%	17.73%	20.38%	24.63%

Table A.2: Results of the last 32 tests using stacked autoencoder networks

Test	Task 1 MAPE 1	Task 1 MAPE 2	Task 2 MAPE 1	Task 2 MAPE 2
33	20.91%	20.91%	20.56%	20.56%
34	20.72%	20.72%	25.37%	25.37%
35	20.11%	20.11%	27.97%	27.97%
36	20.63%	20.63%	25.83%	25.83%
37	20.00%	18.62%	20.56%	23.85%
38	20.84%	19.06%	20.33%	23.70%
39	22.90%	21.83%	24.48%	27.46%
40	20.05%	19.91%	30.44%	31.58%
41	22.54%	20.07%	21.16%	23.16%
42	22.15%	18.71%	18.60%	22.81%
43	21.79%	18.39%	23.36%	24.30%
44	20.58%	18.19%	24.28%	23.10%
45	27.06%	28.61%	26.56%	33.16%
46	21.56%	20.81%	22.20%	31.22%
47	22.57%	21.44%	21.77%	40.09%
48	22.12%	20.24%	26.58%	37.42%
49	21.77%	19.30%	27.55%	27.60%
50	20.14%	17.84%	23.22%	26.87%
51	20.83%	18.13%	30.78%	33.10%
52	20.49%	17.89%	26.35%	28.47%
53	29.15%	29.12%	27.65%	36.88%
54	22.43%	21.25%	30.77%	38.42%
55	24.09%	22.49%	29.76%	36.07%
56	22.35%	21.73%	23.73%	33.86%
57	20.74%	20.74%	15.41%	15.41%
58	21.23%	21.23%	16.20%	16.20%
59	21.32%	21.32%	18.14%	18.14%
60	21.54%	21.54%	18.97%	18.97%
61	21.40%	19.96%	23.32%	24.05%
62	21.05%	19.79%	23.18%	25.07%
63	20.68%	19.40%	23.02%	27.15%
64	20.23%	19.15%	21.95%	26.30%

Table A.3: Results of the first 32 tests using shallow neural networks

Test	Task 1 MAPE 1	Task 1 MAPE 2	Task 2 MAPE 1	Task 2 MAPE 2
1	20.78%	20.05%	31.17%	42.70%
2	21.11%	19.82%	34.13%	43.46%
3	20.72%	20.03%	22.01%	32.36%
4	20.95%	20.22%	20.91%	33.56%
5	21.63%	18.48%	24.31%	28.29%
6	20.61%	17.88%	22.96%	27.93%
7	20.76%	18.74%	24.23%	22.11%
8	20.78%	18.37%	21.43%	20.89%
9	23.03%	21.16%	26.52%	29.04%
10	21.94%	20.67%	21.74%	24.25%
11	20.78%	19.33%	24.41%	24.57%
12	21.43%	20.88%	23.03%	25.62%
13	23.44%	23.44%	36.75%	36.75%
14	21.18%	21.18%	38.52%	38.52%
15	23.62%	23.62%	38.92%	38.92%
16	22.03%	22.03%	23.90%	23.90%
17	24.03%	23.12%	20.35%	25.67%
18	21.47%	20.24%	20.58%	24.38%
19	22.22%	20.90%	22.28%	27.89%
20	21.09%	20.19%	21.74%	28.50%
21	23.75%	23.75%	20.03%	20.03%
22	20.70%	20.70%	21.67%	21.67%
23	21.62%	21.62%	23.90%	23.90%
24	20.49%	20.49%	24.37%	24.37%
25	23.12%	24.65%	25.73%	39.20%
26	21.41%	24.27%	26.17%	49.24%
27	22.11%	21.77%	25.97%	36.83%
28	21.26%	21.67%	25.08%	38.57%
29	20.97%	18.99%	26.61%	27.40%
30	20.93%	18.33%	27.14%	28.56%
31	23.52%	21.08%	25.42%	30.59%
32	23.32%	21.79%	38.78%	42.25%

Table A.4: Results of the last 32 tests using shallow neural networks

Test	Task 1 MAPE 1	Task 1 MAPE 2	Task 2 MAPE 1	Task 2 MAPE 2
33	21.19%	21.19%	21.81%	21.81%
34	20.01%	20.01%	21.44%	21.44%
35	21.06%	21.06%	22.12%	22.12%
36	20.31%	20.31%	23.91%	23.91%
37	20.45%	19.33%	19.68%	25.45%
38	20.19%	19.04%	21.20%	25.69%
39	20.18%	18.98%	19.10%	22.11%
40	20.32%	19.49%	21.47%	23.59%
41	22.99%	19.61%	19.94%	23.78%
42	21.21%	18.44%	20.58%	24.40%
43	21.12%	18.25%	20.82%	25.33%
44	20.73%	18.16%	22.01%	23.17%
45	31.66%	32.05%	34.19%	82.02%
46	22.34%	21.44%	32.12%	49.83%
47	24.93%	22.97%	37.19%	54.02%
48	21.97%	20.56%	26.34%	39.81%
49	21.95%	19.57%	23.95%	27.92%
50	20.17%	17.82%	25.51%	28.67%
51	20.90%	18.20%	21.91%	25.92%
52	20.90%	18.14%	21.67%	24.85%
53	30.34%	31.56%	33.33%	38.77%
54	22.81%	23.13%	35.92%	38.21%
55	24.18%	22.20%	26.81%	36.64%
56	22.09%	21.95%	26.51%	37.17%
57	24.53%	24.53%	17.62%	17.62%
58	22.42%	22.42%	16.11%	16.11%
59	23.82%	23.82%	17.27%	17.27%
60	25.17%	25.17%	16.56%	16.56%
61	23.90%	22.11%	27.23%	30.20%
62	23.02%	21.76%	27.91%	32.03%
63	20.56%	19.60%	23.18%	31.66%
64	21.05%	20.19%	22.64%	28.32%

Table A.5: First 32 tests with corresponding feature levels of the L_{64} orthogonal array.

Test	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**	**F13**	**F14**	**F15**	**F16**	**F17**	**F18**	**F19**	**F20**	**F21**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	1	1	1	1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	1	1	1	1	1	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	1	2	2	2	2	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
6	1	2	2	2	2	2	2	2	2	1	1	1	1	4	4	4	4	3	3	3	3
7	1	2	2	2	2	3	3	3	3	4	4	4	4	1	1	1	1	2	2	2	2
8	1	2	2	2	2	4	4	4	4	3	3	3	3	2	2	2	2	1	1	1	1
9	1	3	3	3	3	1	1	1	1	3	3	3	3	4	4	4	4	2	2	2	2
10	1	3	3	3	3	2	2	2	2	4	4	4	4	3	3	3	3	1	1	1	1
11	1	3	3	3	3	3	3	3	3	1	1	1	1	2	2	2	2	4	4	4	4
12	1	3	3	3	3	4	4	4	4	2	2	2	2	1	1	1	1	3	3	3	3
13	1	4	4	4	4	1	1	1	1	4	4	4	4	2	2	2	2	3	3	3	3
14	1	4	4	4	4	2	2	2	2	3	3	3	3	1	1	1	1	4	4	4	4
15	1	4	4	4	4	3	3	3	3	2	2	2	2	4	4	4	4	1	1	1	1
16	1	4	4	4	4	4	4	4	4	1	1	1	1	3	3	3	3	2	2	2	2
17	2	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
18	2	1	2	3	4	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3
19	2	1	2	3	4	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
20	2	1	2	3	4	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1
21	2	2	1	4	3	1	2	3	4	2	1	4	3	3	4	1	2	4	3	2	1
22	2	2	1	4	3	2	1	4	3	1	2	3	4	4	3	2	1	3	4	1	2
23	2	2	1	4	3	3	4	1	2	4	3	2	1	1	2	3	4	2	1	4	3
24	2	2	1	4	3	4	3	2	1	3	4	1	2	2	1	4	3	1	2	3	4
25	2	3	4	1	2	1	2	3	4	3	4	1	2	4	3	2	1	2	1	4	3
26	2	3	4	1	2	2	1	4	3	4	3	2	1	3	4	1	2	1	2	3	4
27	2	3	4	1	2	3	4	1	2	1	2	3	4	2	1	4	3	4	3	2	1
28	2	3	4	1	2	4	3	2	1	2	1	4	3	1	2	3	4	3	4	1	2
29	2	4	3	2	1	1	2	3	4	4	3	2	1	2	1	4	3	3	4	1	2
30	2	4	3	2	1	2	1	4	3	3	4	1	2	1	2	3	4	4	3	2	1
31	2	4	3	2	1	3	4	1	2	2	1	4	3	4	3	2	1	1	2	3	4
32	2	4	3	2	1	4	3	2	1	1	2	3	4	3	4	1	2	2	1	4	3

Table A.6: Last 32 tests with corresponding feature levels of the L_{64} orthogonal array.

Test	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11**	**F12**	**F13**	**F14**	**F15**	**F16**	**F17**	**F18**	**F19**	**F20**	**F21**
33	3	1	3	4	2	1	3	4	2	1	3	4	2	1	3	4	2	1	3	4	2
34	3	1	3	4	2	2	4	3	1	2	4	3	1	2	4	3	1	2	4	3	1
35	3	1	3	4	2	3	1	2	4	3	1	2	4	3	1	2	4	3	1	2	4
36	3	1	3	4	2	4	2	1	3	4	2	1	3	4	2	1	3	4	2	1	3
37	3	2	4	3	1	1	3	4	2	2	4	3	1	3	1	2	4	4	2	1	3
38	3	2	4	3	1	2	4	3	1	1	3	4	2	4	2	1	3	3	1	2	4
39	3	2	4	3	1	3	1	2	4	4	1	1	3	1	3	4	2	2	4	3	1
40	3	2	4	3	1	4	2	1	3	3	2	2	4	2	4	3	1	1	3	4	2
41	3	3	1	2	4	1	3	4	2	3	1	2	4	4	2	1	3	2	4	3	1
42	3	3	1	2	4	2	4	3	1	4	2	1	3	3	1	2	4	1	3	4	2
43	3	3	1	2	4	3	1	2	4	1	3	4	2	2	4	3	1	4	2	1	3
44	3	3	1	2	4	4	2	1	3	2	4	3	1	1	3	4	2	3	1	2	4
45	3	4	2	1	3	1	3	4	2	4	2	1	3	2	4	3	1	3	1	2	4
46	3	4	2	1	3	2	4	3	1	3	1	2	4	1	3	4	2	4	2	1	3
47	3	4	2	1	3	3	1	2	4	2	4	3	1	4	2	1	3	1	3	4	2
48	3	4	2	1	3	4	2	1	3	1	3	4	2	3	1	2	4	2	4	3	1
49	4	1	4	2	3	1	4	2	3	1	4	2	3	1	4	2	3	1	4	2	3
50	4	1	4	2	3	2	3	1	4	2	3	1	4	2	3	1	4	2	3	1	4
51	4	1	4	2	3	3	2	4	1	3	2	4	1	3	2	4	1	3	2	4	1
52	4	1	4	2	3	4	1	3	2	4	1	3	2	4	1	3	2	4	1	3	2
53	4	2	3	1	4	1	4	2	3	2	3	1	4	3	2	4	1	4	1	3	2
54	4	2	3	1	4	2	3	1	4	1	4	2	3	4	1	3	2	3	2	4	1
55	4	2	3	1	4	3	2	4	1	4	1	3	2	1	4	2	3	2	3	1	4
56	4	2	3	1	4	4	1	3	2	3	2	4	1	2	3	1	4	1	4	2	3
57	4	3	2	4	1	1	4	2	3	3	2	4	1	4	1	3	2	2	3	1	4
58	4	3	2	4	1	2	3	1	4	4	1	3	2	3	2	4	1	1	4	2	3
59	4	3	2	4	1	3	2	4	1	1	4	2	3	2	3	1	4	4	1	3	2
60	4	3	2	4	1	4	1	3	2	2	3	1	4	1	4	2	3	3	2	4	1
61	4	4	1	3	2	1	4	2	3	4	1	3	2	2	3	1	4	3	2	4	1
62	4	4	1	3	2	2	3	1	4	3	2	4	1	1	4	2	3	4	1	3	2
63	4	4	1	3	2	3	2	4	1	2	3	1	4	4	1	3	2	1	4	2	3
64	4	4	1	3	2	4	1	3	2	1	4	2	3	3	2	4	1	2	3	1	4

B Appendix B

This appendix contains additional data plots referenced in Chapter 3.

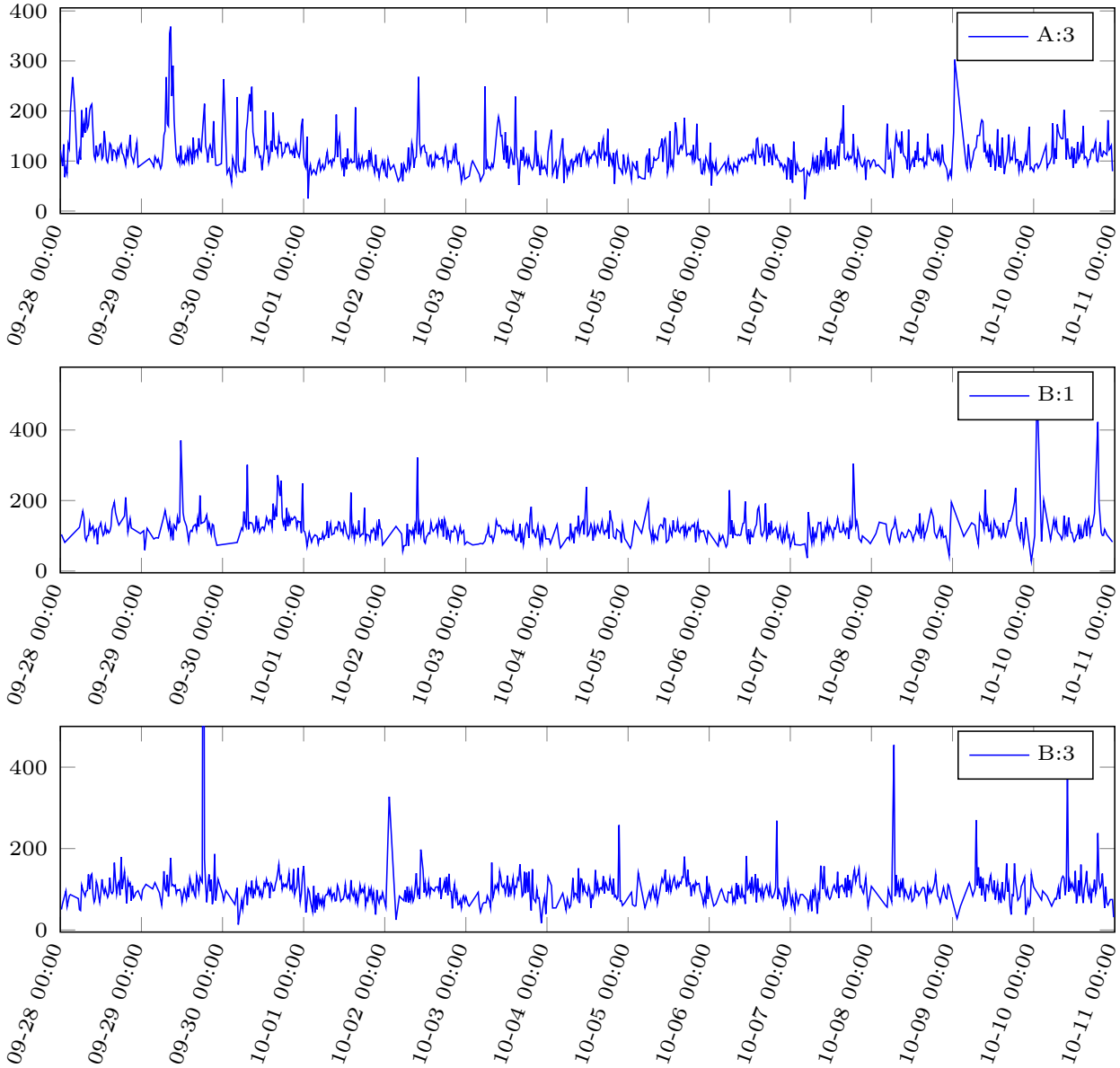

Figure B.1: Task 1 data, from intersection to tollgate, between the 28th of September and the 11th of October. Y-axis: average travel time (s), X-axis: date and time (mm-dd hh:mm)

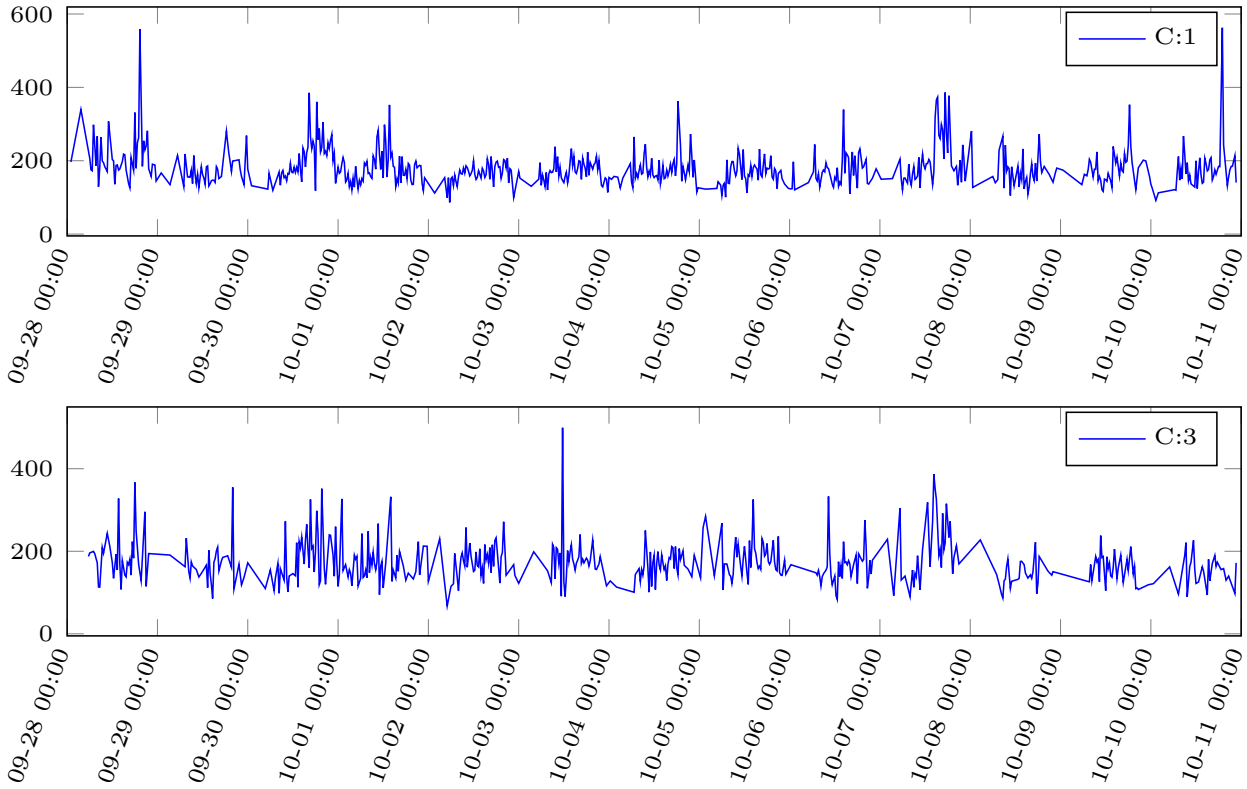

Figure B.2: Task 1 data, from intersection to tollgate, between the 28th of September and the 11th of October. Y-axis: average travel time (s), X-axis: date and time (mm-dd hh:mm).

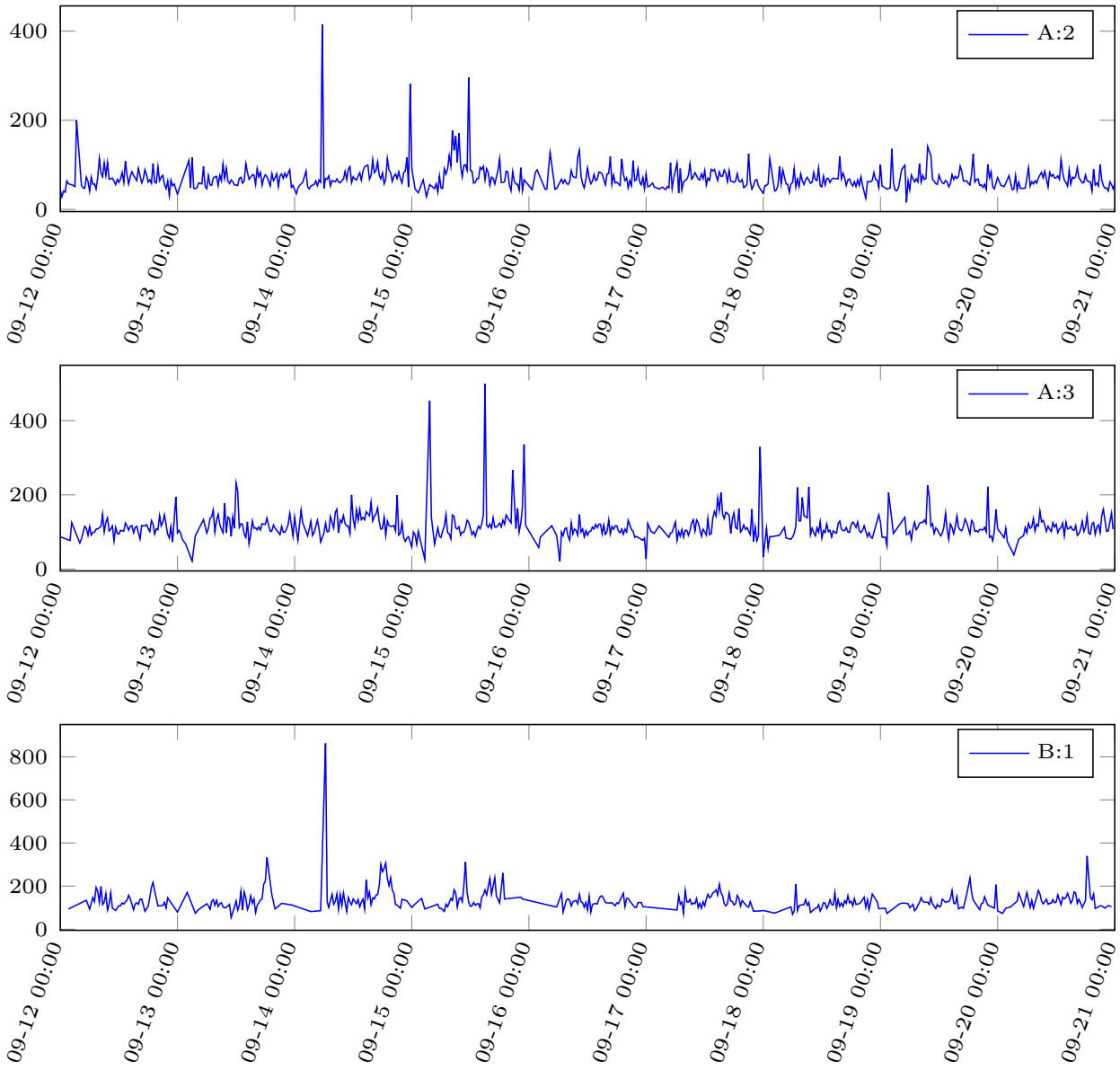

Figure B.3: Task 1 data, from intersection to tollgate, between the 12th and 20th of September. Y-axis: average travel time (s), X-axis: date and time (mm-dd hh:mm).

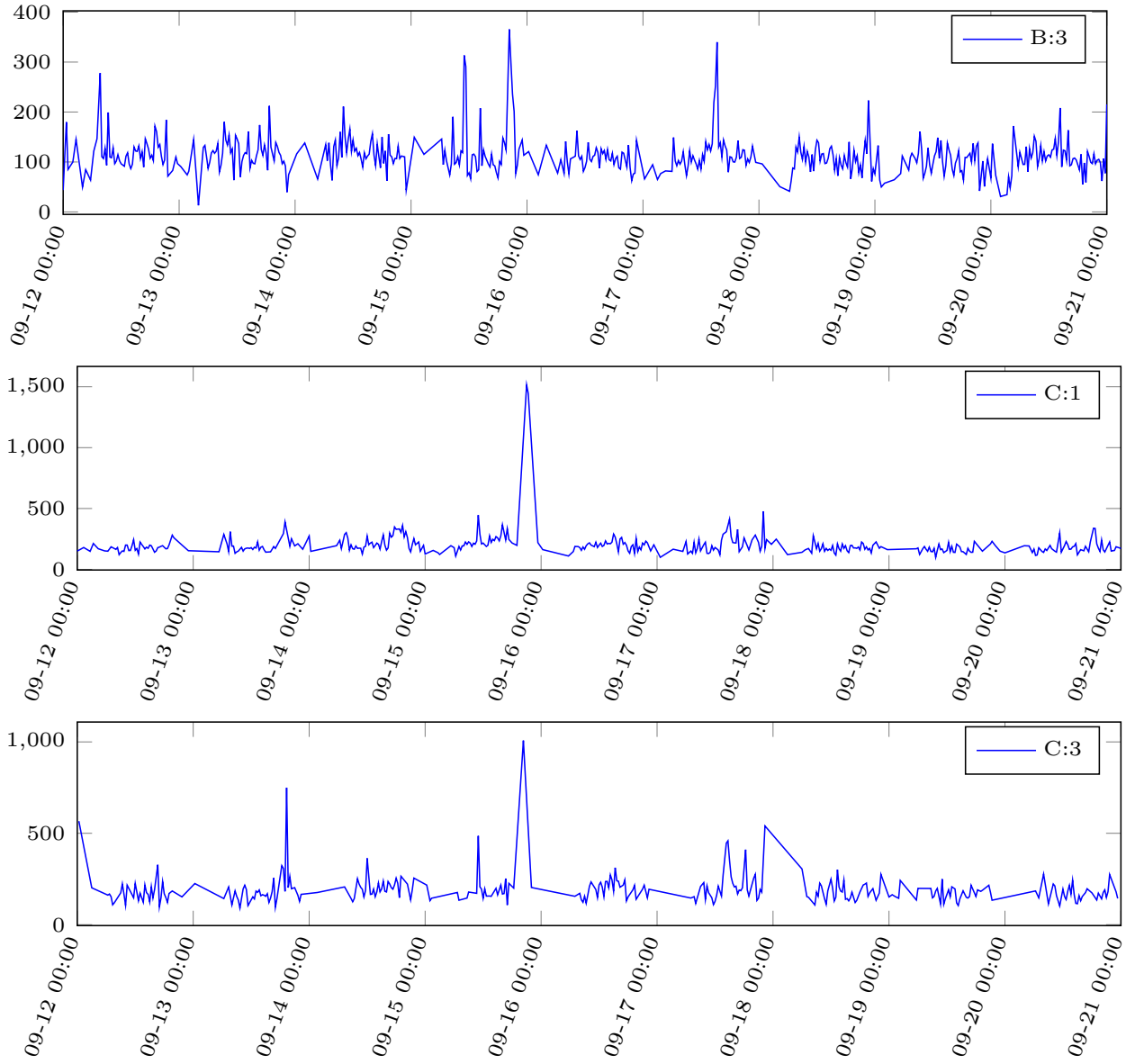

Figure B.4: Task 1 data, from intersection to tollgate, between the 12th and 20th of September. Y-axis: average travel time (s), X-axis: date and time (mm-dd hh:mm).

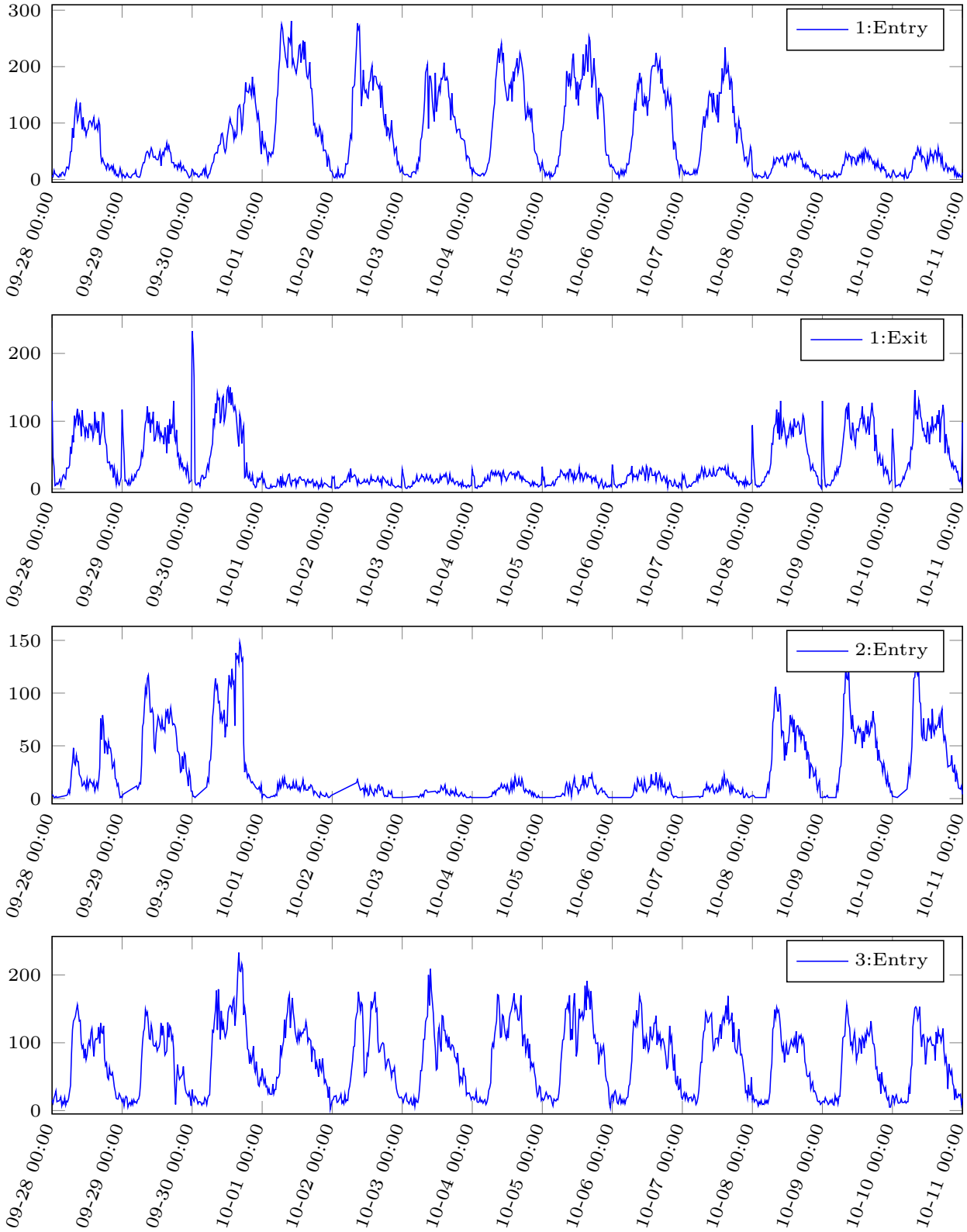

Figure B.5: Task 2 data, from intersection to tollgate, between the 28th of September and the 11th of October. Y-axis: traffic volume (number of vehicles), X-axis: date and time (mm-dd hh:mm).