NTNU

Norwegian University of
Science and Technology

Traffic flow forecasting with deep learning

Per Øyvind Kanestrøm

Master of Science in Informatics
Submission date: January 2017
Supervisor: Anders Kofod-Petersen, IDI
Co-supervisor: Torkil Aamodt, Bekk
Tomas Levin, Statens Vegvesen

Norwegian University of Science and Technology
Department of Computer Science

Abstract

In recent years there has been a vast increase in available data with the advancement of smart cities. In the domain of Intelligent Transportation Systems (ITS) this modernisation can positively effect transportation networks, thus cutting down travel time, increase efficacy, and reduce environmental impact from vehicles.

Norwegian Public Roads Administration (NPRA) is currently deploying a new vehicle detector system named *Datainn* on all public roads in Norway. *Datainn* sends metadata on all detected vehicles in real time. This includes information about speed, gap between vehicles, weight, and classification of vehicle type.

Many machine learning approaches has been researched in literature on how to forecast traffic flow information. One such approach is that of using Artificial Neural Networks (ANNs). In this research ANN based methods have been explored. This was done by first performing a state-of-the-art Structured Literature Review (SLR) on ANN methods in literature.

From the review, Stacked Sparse Autoencoder (SSAE) model was compared with recent advances of Long Short-Term Memory (LSTM) and Deep Neural Network (DNN) on four different prediction horizons. The data foundation was the new *Datainn* system using traffic data from a highway around Norway's capitol, Oslo. Further, the model performance was assessed with extended feature vectors including more metadata from *Datainn*.

The results found that the LSTM model always outperformed DNN and SSAE, although in general the performance characteristics was somewhat similar. Extending the feature vector with more variables had a negative effect on DNN, while resulting in better performance for Recurrent Neural Network (RNN) on long-term (60 minute) forecasting horizons. For SSAE it had a slight positive effect, but not enough get better results than RNN or DNN.

Sammendrag

I de senere årene har det vært en enorm økning i tilgjengelige data med framskrittene av smarte byer. I feltet for Intelligente Transport System (ITS) kan denne moderniseringen positivt påvirke transportnettverk, og dermed kutte ned reisetid, øke effektiviteten og redusere miljøbelastningen fra kjøretøy.

Statens vegvesen er i gang med å ta i bruk Datainn, et nytt system for å detektere kjøretøy på offentlige veier i Norge. Datainn sender metadata på alle oppdagede kjøretøy i sanntid. Dette inkluderer informasjon om fart, avstand mellom kjøretøy, vekt, og klassifisering av kjøretøytype.

Mange maskinlærings metoder har vært utforsket i litteraturen om hvordan å forutsi trafikkflyt. Et eksempel på dette er med Artifical Neural Networks (Anns). I denne forskningen har fokuset vært på slike metoder. Dette ble gjort ved å først utføre et state-of-the-art strukturert litteratur gjennomgang om tidligere brukt ANN metoder i litteraturen.

Fra gjennomgangen, ble Stacked Sparse Autoencoder (SSAE) modellen sammen kombinert med nyere modeller som Long Short-Term Memory (LSTM) og Deep Neural Network (DNN) utforsket på fire forskjellige prediksjon horisonter. Datagrunnlaget var det nye Datainn systemet med trafikkdata fra en motorvei rundt i Norges hovedstad, Oslo. Videre ble modellenes ytelse vurderet med en utvidete input vektorer fra metadata i Datainn.

Resultatene viser at LSTM modellen alltid er bedre enn DNN og SSAE, selv om den generelle ytelseskarakteristikken var nokså like. Å utvidet input vektor med flere variabler hadde en negativ effekt på DNN, men en positiv effekt på resultene til LSTM modellen for langsiktig (60 minutt) prediksjon. For SSAE modellen hadde det en svak positiv effekt, men ikke nok til å få bedre resultat enn RNN eller DNN.

Preface

This work was conducted as the my Masters degree at NTNU. I would like to thank my supervisor Anders Kofod-Petersen for his help and feedback. I would also like to thank the Norwegian Public Roads Administration and my co-supervisor Tomas Levin for granting me data access and domain knowledge. Lastly, a big thanks to my co-supervisor Torkil Aamodt for giving invaluable insights, feedback, and helping as a discussion partner.

Per Øyvind Kanestrøm
Trondheim, 15th January 2017

Contents

List of Figures

List of Tables

List of Abbreviations

AE Autoencoder 12, 24, 28, 58

AI Artificial Intelligence 1, 57

ANN Artificial Neural Network . . . . . i, 2, 3, 9, 12, 22–24, 57, 58, 69

ARIMA Autoregressive Integrated Moving Average 2, 18, 22–24, 57

BN Bayesian Network 24

BP Back Propagation 24

CCF Cross-Correlation Function 22

CDF Cumulative distribution function 36

CNN Convolutional Neural Network 59

DBN Deep Belief Network 24, 25, 57

DNN Deep Neural Network . . . . . i, 24, 30, 38, 43, 45, 49, 50, 55–58

DPNN Deep Process Neural Network 24

EDF Empirical distribution function 36, 49, 50, 55, 57

FFNN Feed Forward Neural Network 12, 13, 27, 28, 30, 43, 50, 57

GMM Gaussian Mixture Model 22

GNN Generalized Neural Network 24

HA Historical Average 27, 37, 50, 55, 57

HPNN Hybrid Process Neural Network 22

ITS Intelligent Transportation Systems . . . . . i, 1, 5, 56, 58, 68

KLD Kullback–Leibler Divergence 28, 29

KNN K-Nearest Neighbor 12, 22

KSOM Kohonen Self-Organizing MAP 23

LogReg Logistic Regression 23, 24

LSTM Long Short-Term Memory . . . . . i, 15, 16, 31, 56–58

ML Machine Learning 2, 23

MLP Multi-layer Perceptron 22, 23

MPRM Multi Factor Pattern Recognition Model 22

MRE mean relative error 35, 36, 44, 55

MSE mean square error 30

MTL Multi Task Learning 25

NAN not a number 29

NLP Natural Language Processing 14, 57

NPRA Norwegian Public Roads Administration . . . . . i, 1, 2, 6, 58

NRW Naïve Random Walk 28, 37, 49, 50, 55, 57

OLWSVR online learning weighted support-vector regression 23

PCA Principal Component Analysis 22

PNN Process Neural Network 24

ReLU Rectified Linear Unit 30, 57

RMSE root mean square error 35, 44, 55

RNN Recurrent Neural Network i, 14–16, 25, 27, 31, 37, 38, 43, 45, 49, 50, 55–59

RTDCE Roadside Traffic Data Collection Equipment . . . 6, 7, 32, 38, 39, 41

SAE Stacked Autoencoder 2, 24, 27, 37, 55, 57

SARIMA Seasonal Autoregressive Integrated Moving Average 22

SLR Structured Literature Review . . . . . i, 3, 5, 19, 27, 31, 57

SNN Spiking Neural Network 25, 57

SOM Self-Organizing MAP 22

SQL Structured Query Language 41

SSAE Stacked Sparse Autoencoder . . . . . i, 28, 31, 43, 45, 50, 55–58, 75

SVR support-vector regression 22–24

SWT Stationary Wavelet Transform 23

WNN Wavelet Neural Network 22, 24

Chapter 1

Introduction

In this section the study is presented with its background and motivation, and how the research is conducted.

1.1 Background and Motivation

In recent years there has been a vast increase in available data with the advancement of smart cities. The big question is how to put this data to good use. The field of ITS is one where research on the new data combined with Artificial Intelligence (AI) has started to show interesting results.

ITS is about how to provide innovative and advanced services relating to modes of transportation and traffic management, and how to enable users to make smarter choices when using transportation networks. This is in direct effect to how effective the infrastructure in urban smart cities are. It is not simple to optimize for faster transportation, reduced environmental impact, and fewer accidents. There are many factors at play in how the daily traffic changes. Is there a football game in the local stadium? Are there active build sites that cause traffic redirection? The list of possible situations altering the traffic flow goes on.

Having a system that can reasonably predict these changes in traffic is of great value to government, industry, and citizens. Commision of the European Communities [2001] has stated that ITS can reduce travel time by up to 20%. This will not only reduce costs for society, but also have a positive effect on the environment.

The closest solution to a production system in Norway today is a site[1] that gives travel time predictions and congestion information for specific road segments in a few selected urban cities. This system is built by the NPRA and the

[1] `http://reisetider.no`

data source is Norway's AutoPASS system. AutoPASS is primarily made for toll stations and uses chips on vehicles to register a drive by.

There are a few issues with this system:

- Privacy concerns because AutoPASS uses chip identification that is unique for each vehicle.
- Can only collect data on road segments where AutoPASS stations are deployed.
- Travel time is acquired when a vehicle passes two AutoPASS stations. As there are many stops in between AutoPASS stations advanced filtering must be done to avoid data from those who have not driven straight from the start to the destination.

The NPRA is currently deploying a new system named Datainn that uses inductive coils under the road as detectors. These sensors register each bypassing vehicle and sends the raw data to their data centres in real time.

The study from Barros et al. [2015] gives a comparison of model and data driven approaches for traffic prediction. In production systems the most found models are Autoregressive Integrated Moving Average (ARIMA) and derivatives, or a hybrid of other techniques. Inductive coil detectors produce lots of data and it is found that data driven models works best in that case.

The study also suggested that it is worth looking at combining the data driven models with weather information. Models based on ANNs are also mentioned as promising. Schimbinschi et al. [2015] gives further insight that ARIMA models should be avoided in favor of ANN based techniques as ARIMA performs smoothing on the data that might affect the spatiotemporal relationships in the data. In their experiments ANN outperforms the other models. Another interesting approach to solving the spatial problem is done by Lv et al. [2014]. They have data from over 15000 different detectors in California and train by performing sliding window over all the detectors at the same time. The model that handles this size is a deep learning model known as Stacked Autoencoder (SAE). By combining all sensors in a single dataset the model will inherently learn the spatial relationship between the detectors.

Solving the problem of predicting traffic flow is thus not only beneficial to society, but the amount of data involved can be used to better understand and advance the field of Machine Learning (ML). New models based on data from Datainn might also help the NPRA phase out the AutoPASS system. Based on these observations my research aims at looking on modern machine learning models for traffic prediction and how to apply them on the new detectors from the NPRA. I will also investigate how these models can be further improved and how they are affected by introducing variables from other data sources.

1.2 Goals and Research Questions

RQ1 What is state of the art in traffic congestion prediction?

RQ2 What neural network based techniques have been used in traffic congestion prediction?

RQ3 How can neural networks best be used for traffic congestion prediction?

RQ4 What changes to the data and data sources have impact on the learners' ability in traffic congestion prediction?

1.3 Research Method

First, a state of the art review based on SLR was performed to answer the two first research questions. Then the process of this thesis was iterative. The phases in each iteration started with the design and creation of a model based on a stated hypothesis. Experiments was then conducted on the model, comparing it to the baseline and models from previous iterations. These experiments was then observed and analysed quantitatively. A new hypothesis was then drawn based on the results. Thereafter, the hypothesis was tested on the next iteration. Conclusions on these iterations was then made.

1.4 Contributions

The contributions presented in this thesis is a state of the art review of traffic flow forecasting with ANN models. Based on the review a few select models are then implemented and tested on traffic data from Oslo, Norway. Then, the performance of models are evaluated with extra features besides traffic flow.

1.5 Thesis Structure

In Chapter 2 the relevant theory for this thesis will be explained. The resulting SLR will also be presented. Chapter 3 will present the different models used. Chapter 4 presents how the experiments are performed and their results. Last Chapter 5 present the evaluation of the results, discuss them, and present ideas for further work.

Chapter 2

Background Theory

First, in Section 2.1 the relevant theory and concepts for this thesis is introduced and explained. Lastly, Section 2.2 presents the Structured Literature Review (SLR) process and the findings from the completed process.

2.1 Background Theory

This section presents the necessary theory and background for this thesis. First, the domain of Intelligent Transportation Systems (ITS) is introduced in Section 2.1.1. Then, in Section 2.1.4, the necessary machine learning concepts used are described. This includes the general concepts and the basics of the models implemented. Further, the concept of data preprocessing is described in Section 2.1.5. Lastly, in Section 2.1.6 the concept of time series forecasting is explained.

2.1.1 Traffic theory

In this section, the variables traffic flow and density is introduced. The concept of free flow is also explained.

Traffic flow

Traffic flow (q) is defined as the number of vehicles per time unit in a reference point.

$$q = \frac{n}{\Delta T} \tag{2.1}$$

Traffic density

The traffic density (k) is the number of vehicles present per length of the road. For a given road this means that we have k_c defined as the critical density. This is when the road has peak traffic flow. When the density goes further up the traffic flow will decrease and at some point reach jam density k_j. Then max congestion have peaked.

Free flow

When density k is less than k_c the traffic is said to be in a free flow state. Traffic flow can also be viewed as $flow = speed * density$. In that regard, the traffic state can also be understood by the mean speed of the reference point.

2.1.2 Collecting traffic data

There are many ways to collect traffic data. It can either be located at the vehicle as an installed GPS sensor or in a phone. The other way is to detect passing vehicles on a given point. For the last method there are many approaches which vary from country to country. In this thesis the data is from Norwegian Public Roads Administration s (NPRAs) new traffic data collection system *Datainn*.

How Datainn collects data

On strategic locations around Norway the NPRA has installed traffic measurement equipment. Each such location is called a station and has a given unique id defined as the measure point number. To detect passing vehicles on the stations, units known as Roadside Traffic Data Collection Equipment (RTDCE) are used. RTDCE works by having inductive coils arranged in loops under the road. These loops read the signature of metal passing over them. Continually this signature is used to classify the passing vehicles. If a vehicle is classified, then the classification results with the current time stamp is stored and transmitted over 3G to the Datainn servers. This system is also used for bicycle traffic. The information attained by the RTDCE on each vehicle is shown in Table 2.1.

Table 2.1: Variables from Datainn

Variable	**Explanation**
equipment_local_timestamp	Time stamp for when vehicle bypassed the RTDCE unit.
datainn_utc_timestamp	Time stamp for when data is stored at Datainn server.

Table 2.1: Variables from Datainn (continued)

Variable	Explanation
lane_number	Corresponds to the lane the vehicle bypassed
speed_(km/h)	Speed of the vehicle
speed_quality	Quality of the speed measurement
length_(m)	Measured length of the vehicle
weight_(kg)	Weight of the vehicle (not yet implemented)
time_gap_back_to_front_(s)	Gap from last passing vehicle
event_number	Monotonically increasing counter for each passing vehicle that is unique for each RTDCE unit
vehicle_type	Classification of the vehicle type (motorcycle, car, truck, etc.)
vehicle_type_quality	Quality of the vehicle type classification
measure_point_number	Identification number for the station on which the vehicle was registered
contains_all_required_fields	True if speed, lane, length, and gap is present

The *speed_quality* variable is used as an estimate of speed measurement quality. In reality each RTDCE comes with inductive loops in pairs. The measurement from both loops are used to assess vehicle information, and *speed_quality* is the difference in measured speed from both loops. Using the *speed_quality* value naively as a measurement for quality can be wrong as the vehicle can be accelerating or decelerate between them.

2.1.3 Definitions

Rush hour

In Fig. 2.1 the historical average from traffic data in Oslo, Norway on one RTDCE is plotted for each day in the week over each hour.

According to the hourly traffic flow in the city it is possible to define which hours can be considered rush hour. When the traffic is in a free flow state it is typical for forecasting models to give worse prediction as the traffic characteristics are random. For that reason these definitions will be used filter out hours when prediction has no purpose. Below are the definitions of rush hour for Oslo city.

- Morning rush hour: $06:00 \sim 09:00$
- Non-rush hour: $09:00 \sim 12:00$
- Evening rush hour: $14:00 \sim 18:00$

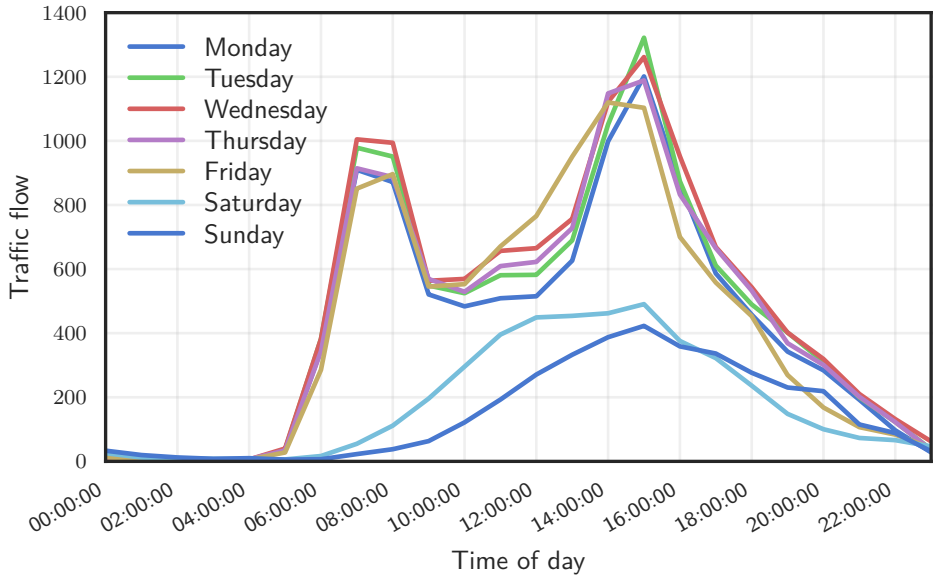


Figure 2.1: Average hourly traffic flow for each day from station 300016.

Forecasting horizons

When forecasting time series it is usual to define a set of horizons on which one focuses on. These definitions vary from study to study but is usually within the three classes defined below. This study is mostly focused on short-term prediction on 15 minute horizon.

- Short-term forecast: time span from 5 minutes to 30 minutes.
- Medium-term forecast: from 30 minutes to a few hours.
- Long-term forecast: from one day to several days.

2.1.4 Machine Learning

The definition of machine learning by Russell and Norvig [2009] is "to adapt to new circumstances and to detect and extrapolate patterns". This gives information about what we want to achieve, but a more formal quote from Mitchell [1997] shown below gives exact definition on how to learn.

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance

at tasks in T, as measured by P, improves with experience E.

In this thesis the program is trying to learn from historical traffic data (E) the task of predicting future time horizon (T) measured in performance by how well it approximates the future traffic flow (P).

There are many approaches to model the program in the definition above. In this section Artificial Neural Network (ANN) and Deep Learning is explained.

Artificial Neural Networks

ANN is a computational unit or network based on approximation of how biological nervous systems work. These consist of multiple connected neural cells signaling each other with electric impulses over synapses. For an in depth explanation on how nervous systems work the reader is referred to Garibay [2010]. The first published work on understanding and using the nervous system, introduced as the perceptron, was in 1958 by Rosenblatt [1958].

In software the perceptron is a set of input values $x_1, \ldots, x_n$ and output activation value y. The sum of a neuron's weighted input signals $x_j w_{ij}$ is defined as the pre-activation a_i as shown in Eq. (2.2).

$$a_i = \sum_{j=1}^{N} w_{ij} x_j - \vartheta_i \tag{2.2}$$

To calculate the activation value an activation function $\Phi(\cdot)$ shown in Eq. (2.3) is used. The input is the pre-activation value with a threshold ϑ_i known as bias. A bias value is used since the output value from the activation function can be above zero for input 0 and thus be considered activated.

$$y_i = \Phi(a_i) = \Phi(\sum_{j=1}^{N} w_{ij} x_j - \vartheta_i) \tag{2.3}$$

There are many different activation functions $\Phi(\cdot)$. Some of the most used are the binary, linear, hyperbolic tangent (tanh), and sigmoid. For this thesis the tanh and sigmoid functions are used. The sigmoid function shown in Eq. (2.4) has the property that for all $\mathbb{R}$ the range is $\{y \in \mathbb{R} : 0 < y < 1\}$.

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{2.4}$$

While the tanh function shown in Eq. (2.5) has the property that the range is $\{y \in \mathbb{R} : -1 < y < 1\}$.

$$tanh(x) = \frac{1 - \mathrm{e}^{-2x}}{1 + \mathrm{e}^{-2x}} \tag{2.5}$$

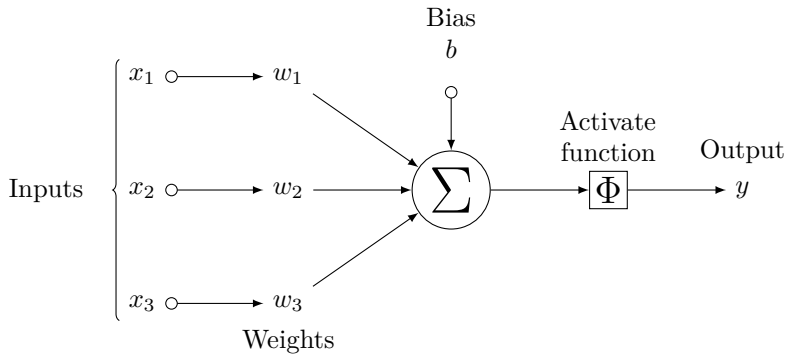


Figure 2.2: Graphical visualization of a perceptron

Training neural networks

In general two steps are performed when training neural networks. The first step is the feedforward pass where the training data is passed into the network and an error estimate is calculated based on the output. Then the error estimate is used in the backwards pass to alter the weights in the network to perform better on the training set. There are numerous different algorithms for the backwards pass. Most of them employ the concept known as gradient descent. The main idea is to find the minima of a function. This is done by calculating the gradient from an error function and taking steps that are proportional to the negative of the gradient.

Given an error function $f(\cdot)$ that is a multi-variable and differentiable in a point a then the fastest way to decrease $f(\cdot)$ is by stepping in the negative direction of the gradient $\Delta f(a)$.

$$a = a - \gamma \Delta f(a) \tag{2.6}$$

After performing the gradient decent calculation several steps, the function $f(\cdot)$ will converge. When $f(\cdot)$ is a convex function, gradient descent will be guaranteed to find the global minimum. As real world problems are not guaranteed to be convex, there will most probably be possible local minima gradient descent can converge to. For this reason the variable γ known as the step size or learning rate is used. It determines how far each step will go down the gradient. Another factor is that each step taken down the gradient is not necessary stable meaning that it can result in taking a step up in another direction.

Training on data is done either in the batch, minibatch, or stochastic way. Batch training is on the entire data set, while minibatch is on a subset of the

data set. Stochastic training is done iteratively on every sample. What type of training one can before depends on the size of the data and training algorithm. To have a sense of how much training a model has performed it is usual to count in epochs where one epoch is having trained on all data samples ones.

There are many variations on gradient descent. Some of these are

- Stochastic Gradient Descent
- Momentum
- Adagrad
- RMSProp

One issue when training neural networks is to avoid overfitting. Overfitting happens when the trained model describes the random noise or error in the training data while performing badly on the testing data. There are many tricks that can help avoid this problem. By applying a technique known as regularization on the model weights and/or bias then higher values will increase the overall loss of the model. In effect this will help avoid overfitting to certain values. Alone this is most often not enough. Therefore, it is often in conjecture that the data is split in different sets. Conceptually the goal is to have an independent data set in which to test the models on. By doing this one can see how the model generalizes on data which it has not been trained on. If the error on the training samples are low but high in the test samples then it is safe to say that there are problems with overfitting.

Concerning the process of training models repeatedly over the same data set one will reach the point where the error function has converged. Stopping the training process when the difference in the error results have started to be less than a threshold or has started to increase again is useful. However one does not known if the model is in a state where it will not generalize over the test data. One way to measure the generalization of the model while training is to use early stopping as shown in Algorithm 1. This is done by splitting the training data further into a smaller set of validation data that is not trained on. On a given interval while training the validation data is used to get the current generalization error. If the training process is improving the generalization error and the training error then it is safer to say that the model is trained into a state that has not been overfitted.

Learning paradigms

There are two different learning paradigms to machine learning algorithms. First there is supervised learning. Supervised means that it learns a representation trained on some sorts of labels connected to the training data. A typical example is of classifying images which has classes depending on what they can. The other

Algorithm 1: Training with early stopping

Data: Data
Result: Trained model
1 Split data into training, validation and test set;
2 Train model on the training set. Every N steps run the model on the validation set and get the error;
3 If the validation set error has not improved after E steps then quit;
4 Use the weights from the run with the best validation set result;

type is unsupervised learning. Unsupervised means that the representation is constructed only on the input data. One example of this is K-Nearest Neighbor (KNN) clustering that creates clusters based on the data in each sample. Another subclass of unsupervised learning is semi-unsupervised learning which Autoencoder (AE) is a good example. AE uses its own input data as "labels" to reconstruct.

Neural network architectures

There are many different ways to construct ANNs. How the network is constructed gives different trade-off between training time, ease of training, memory requirements, and what problems they are able to solve.

Deep Learning The term deep learning can be considered as ANNs where the model architecture is deep and uses techniques to avoid overfitting. There definition deep used for a deep model architecture is that the network has multiple levels of representation that corresponds to different abstractions of the input features.

Logistic regressor Perceptrons that are using a logistic function like the sigmoid or tanh activation functions are usually referred to as logistic regressors.

Feed Forward Neural Network A Feed Forward Neural Network (FFNN) is a multilayer perceptron where each neuron strictly depends on the output of neurons from the layer below. Usually it is trained in a supervised fashion to classify or predict a value.

When the hidden layer representation of a FFNN is used in another architecture it is usually referred to as a fully connected layer.

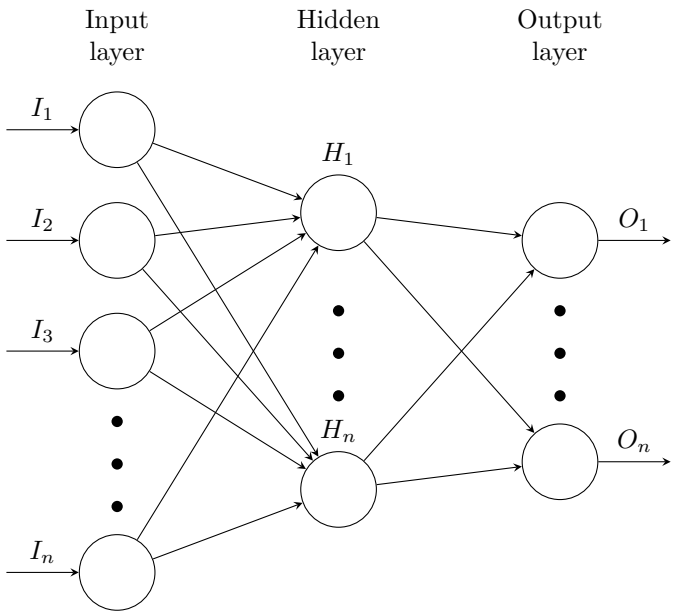


Figure 2.3: Feed Forward Neural Network architecture

Autoencoder An autoencoder is designed by changing the output layer of a FFNN to instead represent the original input features. Conditionally the input neurons and output neurons needs to be the same size. By design an autoencoder does not use the labels and is therefore a kind of semi-unsupervised learner. As shown in Figure 2.4 the weights to the hidden layer is called the encoder and the weights from the hidden layer to the output layer is called the decoder. The encoding layer will after training represent some alteration of the input space. Further this can be used as the decomposition of the feature space into another layer.

The mathematical definition of an autoencoder is shown in Eq. (2.7). Given a set of m training examples $\{X^0, \ldots, X^{m-1}\} \in \mathcal{D}^{\mathcal{M}} = X^i$ each sample contains $\forall\{X_0^i, \ldots, X_{n-1}^i\} \in \mathcal{D}^{\mathcal{M}}$. Then each X^i is the input to the encoder function $y(X^i)$. The decoder function is $z(y(X^i))$. For each layer $f(\cdot)$ and $g(\cdot)$ represents an activation function, and b and c the bias.

$$\begin{aligned} y(x) &= f(W_1 x + b) \\ z(x) &= g(W_2 y + c) \end{aligned} \tag{2.7}$$

Notably, one of the issues with autoencoders are the possibility that the in-

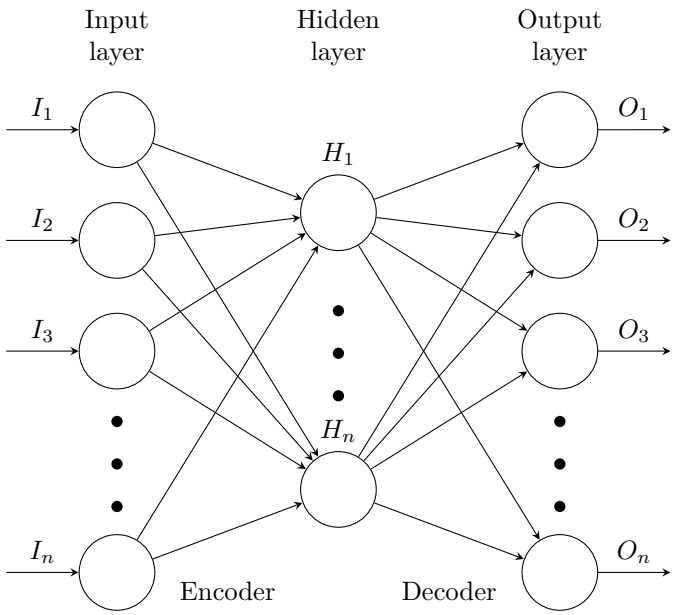


Figure 2.4: Autoencoder architecture

herent representation learned in the encoder layer is the identity function. For that reason it is important that the hidden encoder layer has fewer nodes than inputs nodes. Purposefully, this will avoid the possibility that the hidden layers can represent the identity function. A way to combat this issue is to add a sparsity constraint to the encoding layer. A sparsity constraint will give a higher loss correction for values far away from a given constraint value.

Stacked Autoencoder As explained in the last section the encoding layers can be further used as input to other layers. These layers can be a type of classifier or regressor, or another layer of an autoencoder. Usually, this is done by taking the output of $f_i(\cdot)$ as the input to the next $y_i(\cdot)$ layer. Each layer is trained separately in order by greedy layer-wise training.

Recurrent Neural Network Recurrent Neural Networks (RNNs) is a neural network topology for processing sequential data. They have been made popular from within the community of Natural Language Processing (NLP). This comes from the fact that RNN work as a generalization across variable length sequential input. Generally, the concept of RNN comes from dynamic system. Equation (2.8)

is dynamic system which has the recurrent computation of the state $f(s_{t-1}$.

$$s_t = f(s_{t-1}) \tag{2.8}$$

To calculate the dynamic system equation the recurrent computation is unfolded into a set of steps by repeatedly applying the inner f value. For RNN an unfolded graph is shown in Fig. 2.5. The computation has the form of X as input vectors and Y as output vectors, where H is the internal state vector.

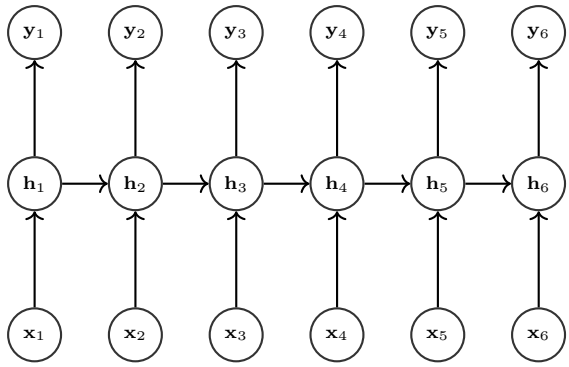


Figure 2.5: Unfolded Recurrent Neural Network architecture

More accurately, RNN works by having internal state vectors h where h_0 is a zero vector. Each new computation follows Eq. (2.9). W_{hh}, W_{xh}, and W_{hy} are the three matrices that computes the internal state and output. Φ is the activation function used.

As can be seen from the equation, the weight matrices are shared over each time step, while the h vector keeps track of inner state.

$$\begin{aligned} h_t &= \Phi(W_{hh}h_{t-1} + W_{xh}x_t) \\ y_t &= W_{hy} * h_t \end{aligned} \tag{2.9}$$

For each step t_0 to t_n the hidden state is passed along one way, computing the next hidden the state. This process works well for predicting the next step t_{n+1}, but in the process long term context is lost. One popular alteration to RNN that addresses the issue of context is the Long Short-Term Memory (LSTM) first introduced by Hochreiter [1997].

Conceptually, it works in the same way as was shown in Eq. (2.9), except that the hidden state computation is based on a new cell state C. Each cell decides whether or not it should update the cell state or keep it as is. This is the long-term

part of the model name. A visual representation of the LSTM cell is shown in Fig. 2.6

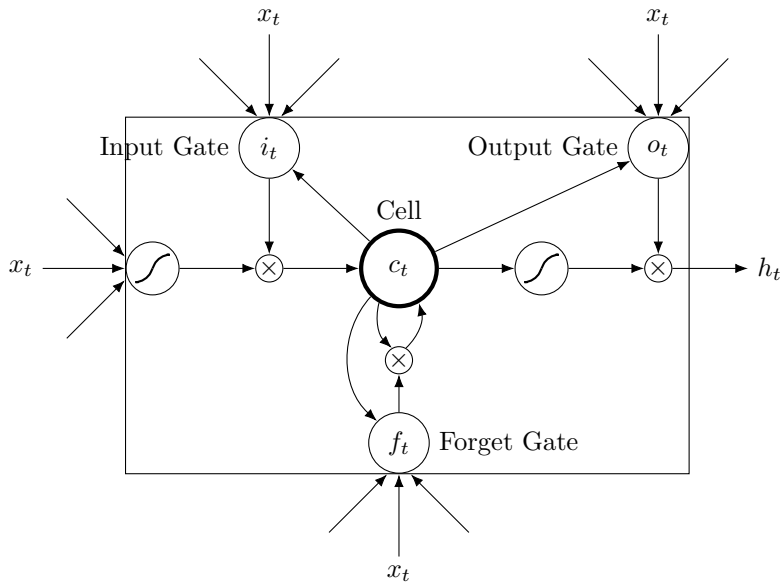


Figure 2.6: Long Short-Term Memory cell

Equation (2.10) shows all the computational steps to go from input x_t to hidden state h_t. Firstly, U is the input matrix weights and W the recurrent matrix weights. f_t is the result of computing the "forget gate layer". Then i_t is computed as the "input gate layer" that represents how much to scale the actual state update. Subsequently, the new candidate values $\tilde{C}_t$ are computed. These represent what can be added to the next state. Next cell state is then computed as the combination of what to forget in the previous state C_{t-1} and the new scaled candidate values $i_t * \tilde{C}_t$. Lastly, the hidden state h_t is computed similarly to RNN, except that its also scaled according to the new cell state C_t.

$$
\begin{aligned}
f_t &= \sigma(U_f x_t + W_f h_{t-1} + b_f) \\
i_t &= \sigma(U_i x_t + W_i h_{t-1} + b_i) \\
\tilde{C}_t &= tanh(U_C x_t + W_C h_{t-1} + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma(U_o x_t + W_o h_{t-1} + b_o) \\
h_t &= o_t * tanh(C_t)
\end{aligned}
\tag{2.10}
$$

2.1.5 Data preprocessing

One of the challenges in machine learning is how to prepare the raw data before trying to fit different models on it. Doing this step correctly requires a good understanding of the problem domain and how the algorithms work.

Firstly, the data itself contains a lot of noise. There are many signal processing filtering techniques that can be applied. One such technique is the median filter. For one discreet signal of N values it goes through every value and the m neighbors besides it, known as the window size. The values in every window is sorted and the median one is selected as the positional value.

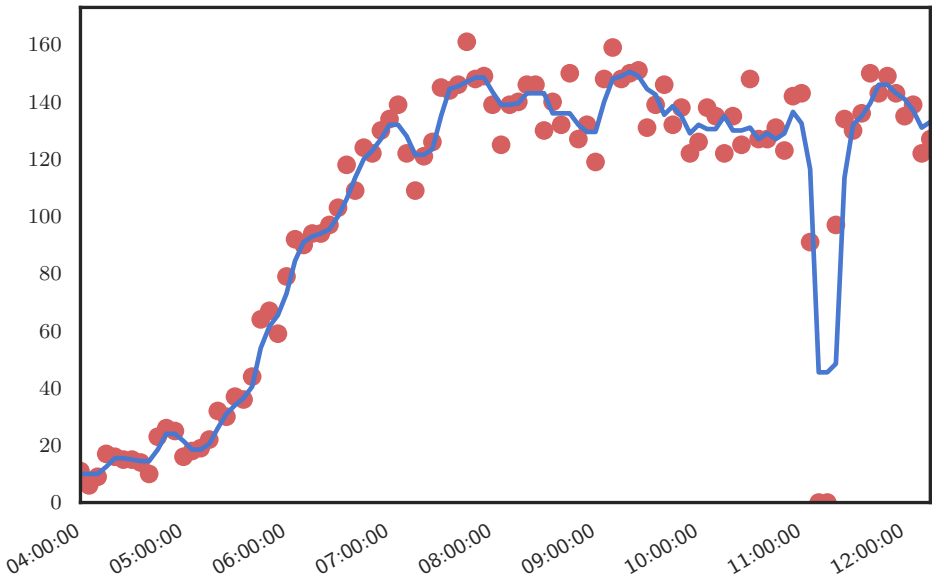


Figure 2.7: Plot of station 300225 lane 4 on 2016-04-26. Red dots are the raw data and the blue line is the results after using a centered median filter with a window size of 4.

Figure 2.7 shows a plot comparing the raw data for one station and the results after using a median filter on it. Some information is lost on the extreme low peaks, but overall there is less noise in the traffic flow information.

Next, the data must be prepared in a way so the used model can handle it. As an example Section 2.1.4 gives an overview over different activation functions and their ranges. If the output of the model is limited based on the inner workings of the model, then the input data must conform to this restriction. In other words

the data needs to be scaled appropriately. This must be done on every input features X_i. One way to do this is to scale the features to have a minimum and maximum value. Equation (2.11) show how to do that.

$$\begin{aligned} x_{std} &= \frac{x - x_{min}}{x_{max} - x_{min}} \\ x' &= x_{std} * (x_{max} - x_{min}) + x_{min} \end{aligned} \tag{2.11}$$

Machine learning algorithms might also take assumptions on the distribution of the data itself. In that case feature standardization must also be done. Given a feature x, the mean $\bar{x}$, and standard deviation σ then Eq. (2.12) does that.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{2.12}$$

Another question to ask is about the domain of the data. If it is subject to many outliers or noise this might affect the performance of the algorithm. Accordingly, this must either be filtered away before use, scaled appropriately, or be used on a model that can handle such data.

2.1.6 Forecasting

In the context of machine learning to forecast is to perform prediction of time series h time steps into unobserved data based on previous observations. In formal terms given sequential observation $X^0, \ldots, X^n$ the task is to estimate some unknown X^{n+h}. The parameter h for horizon denotes the gap between last observation and the point in time to estimate.

More robust models like Autoregressive Integrated Moving Average (ARIMA) and its derivatives needs parameters denoting its seasonality. In the case of traffic data there is a somewhat clear seasonality regarding the different periods of a year. Problems are that traffic is dependent on the ever changing dynamics of urban environments.

Data representation

There are two different ways to represent the data to forecast. Either it can be the same as the input features. Meaning that the predicted value $\hat{Y}_{T+h}$ is the traffic flow for time $T + h$. The other way to represent the predicted value is the change from last observed value Y_T. Predicted traffic flow on time $T + h$ is then $Y_T + \hat{Y}_{T+h}$.

2.2 Structured Literature Review Protocol

This section discusses the results from the completed SLR process. A detailed description of the entire process is found in Appendix A. The website used to find papers is IEEE Xplore[1], a digital library for scientific and technical content. In short the SLR process consists of three steps. First the identification of research was done by selected a set of predefined keywords to help find relevant papers. The search statement was built up based on the terms shown in Table 2.2. This resulted in over 1800 papers and to further reduce this set it was found that many of the papers were from irrelevant publishers. After narrowing the publishers to those relevant, there were 500 papers.

Table 2.2: Search terms from the first SLR step

Concerns	**Search terms**
Domain	Traffic
Problem	Queue Congestion Flow Prediction Forecasting Estimation
Techniques	Deep learning Neural networks Machine learning Big data

Second step is to select the primary studies. In this step the papers were filtered based on the set of inclusion criteria shown in Appendix A.2. After this step, the amount of papers was reduced to 114.

Third step is assess the quality of the primary study papers. This was done by assessing the papers based on their quality through three phases. For the first two phases the inclusion and quality criteria shown in Appendix A.3 were used. Phase three used the quality criteria shown in Appendix A.3.3. All criteria got a score and if the papers score was bad then it was not included in the next phase. In the first phase only the abstracts were read. Then the second phase includes the conclusion and test results, and lastly the third phase use the entire paper. The set of reviewed papers is shown in Table 2.3 and consists of 22 papers.

The result of SLR can also be considered conclusions for RQ1 and RQ2. Partial answers is also given to RQ3 and RQ4 which will further be used in Chapter 4.

[1] `http://ieeexplore.ieee.org/Xplore/home.jsp`

Table 2.3: Final set of papers from the SLR process

Papers	**Title**
Tu et al. [2016]	Mapping Temporal Variables Into the NeuCube for Improved Pattern Recognition, Predictive Modeling, and Understanding of Stream Data
Schimbinschi et al. [2015]	Traffic forecasting in complex urban networks: Leveraging big data and machine learning
Fusco et al. [2015]	Short-term traffic predictions on large urban traffic networks: Applications of network-based machine learning models and dynamic traffic assignment models
Oh et al. [2015]	Urban Traffic Flow Prediction System Using a Multifactor Pattern Recognition Model
Hou et al. [2015]	Traffic Flow Forecasting for Urban Work Zones
Huang et al. [2014b]	Deep Architecture for Traffic Flow Prediction: Deep Belief Networks With Multitask Learning
Lv et al. [2014]	Traffic Flow Prediction With Big Data: A Deep Learning Approach
Huang et al. [2014a]	Deep process neural network for temporal deep learning
Moussavi-Khalkhali et al. [2014]	Leveraging Machine Learning Algorithms to Perform Online and Offline Highway Traffic Flow Predictions
Dunne and Ghosh [2013]	Weather Adaptive Traffic Prediction Using Neurowavelet Models
Jeong et al. [2013]	Supervised Weighting-Online Learning Algorithm for Short-Term Traffic Flow Prediction
Chan et al. [2012]	Neural-Network-Based Models for Short-Term Traffic Flow Forecasting Using a Hybrid Exponential Smoothing and Levenberg–Marquardt Algorithm

Table 2.3: Final set of papers from the SLR process (continued)

Papers	Title
Nguyen and Quek [2012]	Traffic prediction using a Generic Self-Evolving Takagi-Sugeno-Kang (GSETSK) fuzzy neural network
Affonso et al. [2011]	Traffic flow breakdown prediction using feature reduction through Rough-Neuro Fuzzy Networks
Yang et al. [2010]	Prediction of short-term average vehicular velocity considering weather factors in urban VANET environments
Guo et al. [2010]	Comparison of modelling approaches for short term traffic prediction under normal and abnormal conditions
Gu and Yu [2010]	Study on Short-Time Traffic Flow Forecasting Methods
Zhu and Zhang [2009]	A Layered Neural Network Competitive Algorithm for Short-Term Traffic Forecasting
Hu et al. [2008]	Hybrid Process Neural Network based on Spatio-Temporal Similarities for Short-Term Traffic Flow Prediction
Liu et al. [2006]	Research on Forecasting Model in Short Term Traffic Flow Based on Data Mining Technology
Guan et al. [2005]	A practical model of dynamic forecasting of urban ring road traffic flow
Guozhen Tan et al. [2004]	Traffic flow prediction based on generalized neural network

2.2.1 State of the art review

The literature in traffic prediction can be divided into many segments depending on what type of problems they try to solve. The underlying infrastructure from where the data is gathered also vary from every data source, thus there are vastly different techniques based on the limits or possibilities. Some have access to gap information, or have added other non-vehicle variables as weather, accidents, and day time. Another factor is what they are trying to predict. Some try to predict mean velocity of vehicles or mean traffic flow. Others focus on long term or short term forecasting. There are also different approaches to how the data is used.

Certain methods only focus on one station while others incorporate upstream and downstream stations, or look at the big picture and tries to inherently model the spatiotemporal relationships between stations.

The different approaches can be divided as being model or data driven. Decidedly, this study is focused on data driven models. Additionally, data driven models can further be defined by the statistical definition as either parametric or nonparametric. In this definition parametric models makes a priori assumptions about the data distribution. Notable models are ARIMA, Seasonal Autoregressive Integrated Moving Average (SARIMA), and Kalman Filters which has been used extensively with great success. However, the focus in this study is on ANN which can be considered nonparametric. Consequently, this refers to ANNs which is trying to estimate the underlying function of the data. Other nonparametric models are support-vector regression (SVR), decision trees, and KNN.

Variable reduction

One problem within traffic prediction is the large number of variables with inherently complex relationships to model. Liu et al. [2006] used genetic algorithms to successfully reduce the needed variables for Wavelet Neural Network (WNN).

The study from Moussavi-Khalkhali et al. [2014] used Principal Component Analysis (PCA) to handle many parameters like occupancy, flow, speed, etc. Their simple Multi-layer Perceptron (MLP) model perform better after this process. In the conclusion they recommend looking at deep architectures and auto-encoders.

Oh et al. [2015] worked on tackling the issue that variables vary in their value range. The suggested approach was Multi Factor Pattern Recognition Model (MPRM) that normalized the input values. Their proposed model used Gaussian Mixture Model (GMM) for clustering and Levenberg-Marquardt back-propagation algorithm for learning.

Hou et al. [2015] focused on traffic in work zones with both long-term and short-term forecasting. Together with traffic flow at one station, they used an upstream and downstream station as well. In addition to this they included the workday, hour in day, and speed limit. The results however showed that these extra parameters had little significance for the models' accuracy.

Spatiotemporal

Another way to approach the issue of having many variables is to model the inherent spatiotemporal relationship between them. Hu et al. [2008] looked at the issue in how spatiotemporal stations affect each other. By performing Self-Organizing MAP (SOM) clustering with Cross-Correlation Function (CCF) they found, for one station, the other most spatiotemporal similar stations. They theorized an Hybrid Process Neural Network (HPNN) model and test the assumption that the

more spatiotemporal stations give better accuracy than the closest upstream and downstream ones. This assumption was found to be true and the authors suggest that some stations are effected by the fact that they are closer to ramps that bias their correlation. Their HPNN model also performed well with upstream and downstream stations, but best with the most spatiotemporal ones.

Zhu and Zhang [2009] uses Kohonen Self-Organizing MAP (KSOM) to cluster data and test different models within the clusters. The proposed method performs better than ARIMA. The model is only tested on one station with a 10 minute horizon.

With a dataset from Melbourne, Australia spanning six years, Schimbinschi et al. [2015] has used this to test a few important assumptions regarding traffic prediction with big data and machine learning. The tests were done with Logistic Regression (LogReg), ANN, and classification trees. The results show that increasing window size does increase accuracy. This was also proved by getting the same effect after having removed the biggest source to variance in the data; weekends. Clustering stations by proximity further improved accuracy. This shows that the spatial information is more influential than the temporal. It was also found that prediction accuracy has a significant decrease when using fewer data.

Jeong et al. [2013] addresses the issue that historical data is less significant than newer data when prediction future states. The proposed method is online learning weighted support-vector regression (OLWSVR). The proposed method works much better than regular SVR and a simple Machine Learning (ML) with four neurons in the hidden layer.

Extending models with other variables

Other research has focused on weather data. Yang et al. [2010] applies a simple MLP to see the prediction accuracy when introducing weather information. The results show a marginal increase in prediction accuracy with weather data.

Dunne and Ghosh [2013] takes into account the effect of rainfall when predicting traffic flow. The model uses Stationary Wavelet Transform (SWT) to perform neurowavelet prediction. Their model outperforms a standard ANN model substantially.

Guo et al. [2010] looks at how traffic behaves under normal versus abnormal conditions (accidents, etc.). This is done by making three separate input models that respectively considers: Current stations, current stations with historical data, and current station with historical data and error feedback. These three input models are then tested under two different conditions; normal and abnormal traffic where the abnormal traffic is traffic data from when a traffic accident happened. Perhaps unsurprisingly, results show that historical data negatively impacts prediction under abnormal conditions. However, using error feedback loops positively helps in these cases.

Fuzzy

The traffic prediction domain has also been tested with Neuro-fuzzy Networks, a variation of which use ANN to train the rules. Affonso et al. [2011] tries to use Rough Neuro Fuzzy Network on MLP and RBF to measure the impact when performing reduction on the rule set.

Nguyen and Quek [2012] found great results using Self-Evolving Takagi-Sugeno-Kang (GSETSK) Fuzzy Neural Network.

Artificial Neural Network

Guozhen Tan et al. [2004] perform early experiments with Generalized Neural Network (GNN) on traffic flow prediction with upstream and downstream stations in five minute aggregated windows. Guan et al. [2005] makes a practical attempt to use ANN to forecast traffic in Beijing.

Gu and Yu [2010] shows that chaotic neural networks outperforms traditional Back Propagation (BP) ANN on road intersection exits.

Chan et al. [2012] generalizes ANN by using hybrid exponential smoothening. Their results show that the generalization power of ANN is increased when the lumpiness in the data is removed. It was also proved that this method helped more complex ANN like WNN and BNN.

Fusco et al. [2015] found that ANN and Bayesian Network (BN) had similar accuracy characteristics.

Deep Learning

Lv et al. [2014] proposes Stacked Autoencoder (SAE) to address the issue of spatiotemporal relationship. For 15 minute traffic flow prediction they found that the model with three layers where each layers consists of $[400, 400, 400]$ neurons performed best. The prediction layer used LogReg. To train this deep learning architecture they used greedy layer-wise training. The hidden layers are first trained unsupervised, then the prediction layer was trained supervised.

Huang et al. [2014a] builds on deep learning techniques to propose Deep Process Neural Network (DPNN) which is a combination of the principles of building and training AE combined with Process Neural Network (PNN). When compared against Deep Neural Network (DNN) they found that DPNN has better accuracy, convergence time, and training time.

Huang et al. [2014b] proposed using Deep Belief Network (DBN) for unsupervised feature learning. This is similar to SAE and the approach taken by Lv et al. [2014]. For regression a sigmoid layer is applied atop of the unsupervised hidden layers. This model was compared against many models including ARIMA, ANN, and SVR. The DBN architecture outperformed all the other models in all

tests including increasing prediction windows up to one hour. Multi Task Learning (MTL) was also tested atop the unsupervised DBN model. This clustering approach helped increase the models' ability for generalization and performed better.

Tu et al. [2016] adds graph mapping to temporal data with NeuCube Spiking Neural Network (SNN) architecture. NeuCubes were originally designed for brain data. With the addition of mapping, thew new model is proved to work in different domains, including traffic flow prediction. This model addresses the fact that spatiotemporal traffic data is inherently complex and changes over time (similar to RNN, and self-organizing models).

Chapter 3

Architectures/Models

This sections outlines the implementation details about the models used in the experiments. First, in Section 3.1, the models and the background for using these models in the experiments are explained. Then in Section 3.2 the implementation details on language choice, frameworks, and data pipeline are explained.

3.1 Models

In this section the implemented models are explained. The four models are Historical Average (HA), Feed Forward Neural Network (FFNN), Stacked Autoencoder (SAE), and Recurrent Neural Network (RNN). HA is used as the simple baseline algorithm showing the predictive effectiveness when the model is not making any assumption besides the previous data. FFNN architecture is selected since it showed promising predictive results when applied to big data sets in Schimbinschi et al. [2015]. For the more complex models SAE architecture from Lv et al. [2014] is selected. Last an RNN architecture is implemented. From the Structured Literature Review (SLR) results in Section 2.2 no previous implementation of RNN models were found in the context of traffic data.

3.1.1 Historical Average

The HA model is used as a baseline for the other models. HA calculates the mean traffic flow for every station on every day and hour in all the weeks. Simplistically, the error deviation from this model on the test set does also give a certain notion of the temporal difference in the train/test data split.

The historical data is denoted as $y_1, \ldots, y_T$, then $\check{y}_{T+h|T}$ is the estimate of y_{T+h} based on $y_1, \ldots, y_T$ as shown in Eq. (3.1).

$$\check{y}_{T+h|T} = \overline{y} = \frac{(y_1 + \cdots + y_T)}{T} \tag{3.1}$$

3.1.2 Naïve Random Walk

The Naïve Random Walk (NRW) algorithm is the second baseline algorithm and is perhaps one of the simplest models for forecasting. By Eq. (3.2) it is stated as taking the last observed value Y_T as the future value $\hat{Y}$, where h is the prediction horizon.

$$\hat{Y}_{T+h} = Y_T \tag{3.2}$$

All the different data representations can easily be misused by the models to either predict the identity function for the last observed value (thus having the same predictive power as NRW), or always predicting the value for zero change (also having the same predictive power as NRW). By comparing the other models with NRW more confidence will be gained as to not having this issue.

3.1.3 Feed Forward Neural Network

The FFNN model implemented is explained in Section 2.1.4. It is a simple architecture with one hidden layer and the sigmoid activation function.

3.1.4 Stacked sparse autoencoder

In this section the Stacked Sparse Autoencoder (SSAE) model is introduced. It is based on the SSAE model from Lv et al. [2014] with some alterations as explained below. A more detailed explanation of the core theory is found in Section 2.1.4.

For training the model the separate stacked Autoencoder (AE) layers are trained in a semi unsupervised fashion to recreate the features. This is done in a greedy layer-wise fashion. Then the prediction layer is trained in a supervised fashion. The prediction layer is a fully connected layer with the sigmoid activation function. Conditionally, the sigmoid layer is used because it has a range of $\{y \in \mathbb{R} : 0 < y < 1\}$, which is important in regards to how the data is represented as is discussed in Section 4.3.1

In the AE layers the encoder $f(\cdot)$ and decoder $g(\cdot)$ functions from Eq. (2.7) are also the sigmoid function.

The loss function representing the error used for calculating the gradients is the l2-loss in the reconstruction function $L2(X^i, Z')$ and Kullback–Leibler Divergence (KLD) divergence from $y(x^i)$ as shown in Eq. (3.3).

$$loss = L2(X^i, Z') + \gamma \sum_{j=1}^{H_D} KL(\rho||\check{\rho}_j) \tag{3.3}$$

KLD is used as the sparsity constraint on the encoding layer. H_D is the number of neurons in the hidden layer. $\check{\rho}$ is calculated as $\check{\rho}_j = \frac{1}{N}\sum_{i=1}^{N} y_j(x^i)$. The property given by using the KLD divergence is that $KL(\rho, \check{\rho}) = 0$ if $\rho = \check{\rho}$. If $\check{\rho}$ approaches either 0 or 1 then KL will diverge to ∞ as plotted in Fig. 3.1 This ensures that the activations in the encoding layer is kept as sparse as possible. It is important to note that KLD is only defined for the domain $\{y \in \mathbb{R} : 0 < y < 1\}$. Subsequently, this is why the sigmoid function is used in the encoding and decoding layers.

$$L2(X^i, Z') = \frac{1}{2}\sum_{i=1}^{N} ||x^i - y(x^i)||^2 \tag{3.4}$$

$$KL(\rho||\check{\rho}) = \rho log\frac{\rho}{\check{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\check{\rho}_j} \tag{3.5}$$

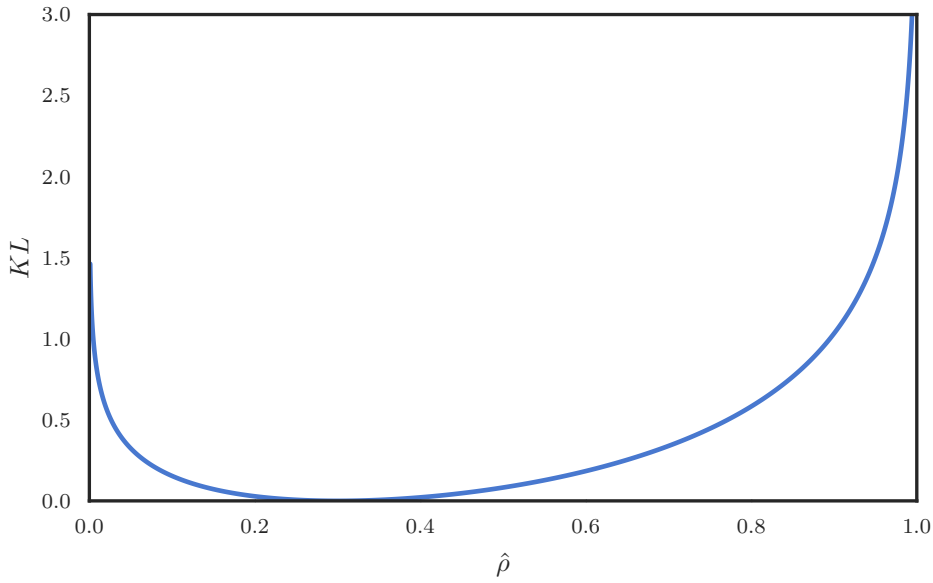


Figure 3.1: Plot of $KL(0.3, \check{\rho})$.

One issue with the KLD implementation is that $\check{\rho}$ values of 0 will give not a number (NAN) values in the gradient and effectively "kill" the neuron. By adding

a small noise value of 10^{-9} in the denominator of the log expressions in Eq. (3.5) this is avoided.

These autoencoders are then stacked in layers. Given a three layered autoencoder the last encoding layer will be computed as $y^3(y^2(y^1(\cdot)))$.

To perform prediction on future states the last encoding layer is feed into the prediction layer with the activation function. This is the same model as explained in Section 3.1.3. Loss is calculated as the mean square error (MSE) of the prediction layer result Y' and the actual values Y.

$$MSE(Y, Y') = \frac{1}{N}\sum_{j=1}^{N}(y_j - y'_j)^2 \tag{3.6}$$

In Lv et al. [2014] the weights are initialized with random values. In this model Xavier initialization is used. Xavier initialization was first introduced in Glorot and Bengio [2010] and is a normal distribution based on the count of neurons in and the neurons out of a feed forward network. For the distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, μ and σ^2 is shown in Eq. (3.7).

$$\begin{aligned} \mu &= \sqrt{\frac{6.0}{in + out}} \\ \sigma^2 &= \sqrt{\frac{3.0}{in + out}} \end{aligned} \tag{3.7}$$

The greedy layer wise training of the models is done as shown in Algorithm 2. Every autoencoder layer is trained one by one where l^0 use the features as input and the next l^1 use layer l^0 as input. After the pretraining is done then the prediction layer is trained on the entire network minimizing MSE.

3.1.5 Deep Neural Network

The Deep Neural Network (DNN) model is an extension to the FFNN model. While having more then one hidden layer, it sets itself apart from FFNN by using Rectified Linear Unit (ReLU) as activation function and dropout between every layer. The advantages of using ReLU is that its an linear activation function as apposed to sigmoid which is nonlinear and can easily blow up the activation values. The equation for ReLU is shown in Eq. (3.8).

$$f(x) = max(0, x) \tag{3.8}$$

Dropout, first introduced by Hinton et al. [2012], is a regularization technique. Each neuron in a layer with dropout has a specified chance of not being “dropped”.

Algorithm 2: Training algorithm for SSAE

Data: Training samples X, features Y, and hidden layers l
Result: Trained model
let γ be the sparsity weight;
let ρ be the sparsity term;
Initialize the weights with Xavier initialization;
let *input* be the features X;
foreach *layer in* l **do**
    Calculate $z' = z_{layer}(input)$;
    Calculate $loss(z', Y)$;
    Minimize the loss and repeat from step 6 until convergence;
    let *input* be $layer'$;
Train the prediction layer with MSE until convergence;

A dropped neuron is set to zero. The idea is that the network is forced to learn observe the data as more noise than it actually is. The internal representation obtained after training is more general, and in turn perform better on unobserved data.

3.1.6 Recurrent Neural Networks

From the SLR performed in Section 2.2 no model where found that was based on RNN. RNNs has been successfully applied to prediction tasks on sequential data. Given that traffic data also has the characteristics of being sequences where each step is dependent on the last, an Long Short-Term Memory (LSTM) model is included in the set of deep learning models. The RNN model with LSTM cells are explained in Section 2.1.4.

TODO

3.2 Implementation

All development is done the programming language *Python 3.5*[1]. The main advantage with using the *Python* language is that it is a interpreted dynamically typed language with great interoperability with low level *C* code. Even though interpreted languages often has bad runtime performance, especially when it comes to number crunching, *Python* has become widely popular within the field of data

[1] `https://www.python.org/`

science. This is can be attributed to the vast ecosystem of *Python* interface for numerical computation implemented in *C*.

For exploration and analysis of data the *Pandas*[2] library for tabular data is used. *Pandas* has efficient tools to work with time series making it possible to resample, pivot, and plot.

To perform data preprocessing and grid searches on the models the machine learning library *scikit-learn*[3] is used.

The models are implemented with *Tensorflow*[4]. The library is developed by *Google* and first introduced 9 November 2015 with the whitepaper by Abadi et al. [2015]. There are many alternatives to *Tensorflow* for deep learning but the advantage is that the main interface for model development is in *Python* and that all computation are built up as a graph that can be evaluated on CPU and GPU. This makes it possible to use high level *Python* code to built new low level numeric computations on top of the normal constructs. On the downside the library can still be considered new and the api, at time of writing, is still in flux.

Tensorflow also ships an interface known as *skflow* that has an interface similar to the models in *scikit-learn*. This makes it easy to build pipelines with transformations and performing grid search. This interface had some issues when trying to perform layer-wise training. To support this feature it was necessary to write a new *scikit-learn* model interface which had the features necessary for the models in this thesis.

One of the advantages of using *Tensorflow* is that it has great features for logging the training data results and visualise it with the built in *Tensorboard* software. *Tensorboard* is a website run locally which can visualize different parameters in the model for each separate step. It can also show all the layers in the model so it can be visually inspected.

The raw data delivered from Datainn was too large to work on the complete set in memory. For this reason the raw data was copied over to the database engine *PostgreSQL* [5]. *PostgreSQL* was chosen because it has features for working with time series. The data was aggregated within a specific interval and grouped on each Roadside Traffic Data Collection Equipment (RTDCE) unit and lane. This process takes too much time do be done on each run of a model so the results are cached on disk.

By combining the data caching over *PostgreSQL* and the model interface of *scikit-learn* it was possible to define a JSON formatted file as an experiment which specifies all model parameters, data parameters, and horizon prediction parameters. This made it simpler to perform new experiments with small alterations while

[2] `http://pandas.pydata.org/`
[3] `http://scikit-learn.org/`
[4] `https://www.tensorflow.org/`
[5] `https://www.postgresql.org/`

keeping the original information about previous experiments.

Chapter 4

Experiments and Results

This chapter introduces how the experiments are performed with their respective results. Firstly, Section 4.1 describes the performance metrics and how they are evaluated. Then, Section 4.2 describes the experiments performed. Section 4.3 describes the setup of the experiments. Section 4.4 introduces the environment in which the experiments are performed on. Lastly, the results from the experiments are introduced in Section 4.5.

4.1 Experimental Plan

In this section the metrics used to evaluate models and methods to compare between models are introduced.

4.1.1 Performance metrics

Models are compared by evaluating their prediction error as given by two error estimates: The mean relative error (MRE) from Eq. (4.1) gives a relative error for a given model, and root mean square error (RMSE) from Eq. (4.2) for measurement of accuracy between models. A problem with MRE is that it is undefined for target values of 0. Given that the measurements are calculated on traffic flow then the only data of interest is where traffic flow is above zero. For that reason the division error can be regarded as not a big issue.

$$MRE(y, y') = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - y'_i|}{y_i} \tag{4.1}$$

$$RMSE(y, y') = \sqrt{\sum_{i=1}^{n}(y_i - y_i')^2} \tag{4.2}$$

It is also relevant to know how the different models perform on each station between each other. One way to see this is by plotting the empirical distribution function.

The accuracy of the models is also shown within rush hour times as defined in Section 2.1.3.

4.1.2 Evaluation

The error metrics explained in Section 4.1.1 gives error over each station and lane. To correctly assess the accuracy between models the Empirical distribution function (EDF) is applied. First this section introduce how this is done. Finally, it is explained how the error estimates are compared when the result data is filtered on different criteria.

Empirical error distribution

All models output multivariate variables representing the traffic flow for each station. The information which differentiate the separate models are how well they predict the flow for every station and lane. Error variations are highlighted with EDF. EDF is Cumulative distribution function (CDF) with the empirical measure of a sample. For a measure $x \in X$ it tells how much, from 0 up to the total distribution of 1, how much of the data is distributed with a value under x. In this case, the sample is MRE over the prediction result for each station.

Comparing error distributions

In the data set applied, there are many variations based on the given time, time of the year (holidays), events, and other unknown factors. It is important to investigate how the models behaves on these factors. In order to highlight some of these factors, the data result set is filtered to only contain data within specified criteria before the error metrics are computed.

One important factor to note is that the models are more prone to errors in night time. Another is to filter the data on a certain threshold of traffic flow volume.

4.2 Experiments

This section introduces the experiments performed and their rational for doing them. First the experiment concerns with how the model behaves with different prediction horizons and what lag that gives the best results. The second experiment will see how the results are affected by using more variables besides the flow data. Lastly a wide and deep model is tested. It is a combination of the Stacked Autoencoder (SAE) architecture and Recurrent Neural Network (RNN).

Decidely, the models used are SAE, RNN, Naïve Random Walk (NRW), and Historical Average (HA). HA and NRW are the naive models that must be outperformed if the more complex models are too be considered useful. Each of those represent two different aspects of what is important in a forecasting algorithm By getting a better result than NRW a model has clearly got some meaningful understanding of the data. For HA the importance of beating it is a baseline for having any merit at all.

4.2.1 Experiment: Prediction horizon

The forecasting horizon is the time lag from last observed value to the time steps one is trying to forecast. In this study is focused on forecasting horizons of short-term (15 minutes), mid-term (30 minutes), mid-long-term (45 minutes), and long-term (60-minutes). Hopefully, by testing model performance on the different horizons, insights into the model performance characteristics will be gained.

4.2.2 Experiment: Introducing other variables

In this experiment the method of adding extra variable on the feature vector is explored. This is the same method as from Guo et al. [2010] where weather data was used. But, in this research, the traffic information from *Datainn* is used instead. The relevant variables exposed from *Datainn* per vehicle are shown in the list below. Within each aggregated interval the mean and standard deviation is computed.

- Vehicle speed
- Vehicle gap

Other metadata that can be considered relevant for every aggregated time step is:

- Hour of the day
- Day of the week
- Day of the year

The question remains which of these are actually related to the traffic flow. This is computed by the Spearman correlation. This correlation takes no assumption about the distribution of the data, but assumes that it is ordinal and the different variables are monotonically related.

The correlation computation takes all variables which includes every Roadside Traffic Data Collection Equipment (RTDCE) station. This correlation grid is thus $|variables| * |variables|$ in size. Therefore, the visualization of the Spearman correlation shown in Fig. 4.1 only contains each vehicle variable. The correlation values are the mean over every station.

The correlation heatmap clearly shows that relevant variables that should be used in the model are the ones listed below.

- Hour of the day
- Day of the week
- Average gap
- Variance gap

Negative score implies that the variable will negatively affect the values. In this regard a higher gap between vehicles means that the traffic flow is more likely to be lower.

In the last experiment results, outlined in Section 4.5.1, that mid-term and long-term forecasting gave the most interesting result. Therefore, those were the only forecasting horizon explored. Only RNN, Deep Neural Network (DNN), and DNN results were explored.

4.3 Experimental Setup

In this section the setup details are described. Section 4.3.1 describes the specifics of how the data set is used. Lastly, Section 4.3.2 describes the hyperparameter tuning performed on the models.

4.3.1 Data set

The data set used is explained in Section 2.1.2. There are three dumps that was taken from the data set. The data contained in each dump is explained in the list.

Dump 1 From 2016-02-01 00:00:01.79+00 to 2016-09-11 03:47:24.979+00 with 27 stations.

Dump 2 From 2016-01-29 00:00:00.679+00 to 2016-11-03 06:52:09.25+00 with 26 stations.

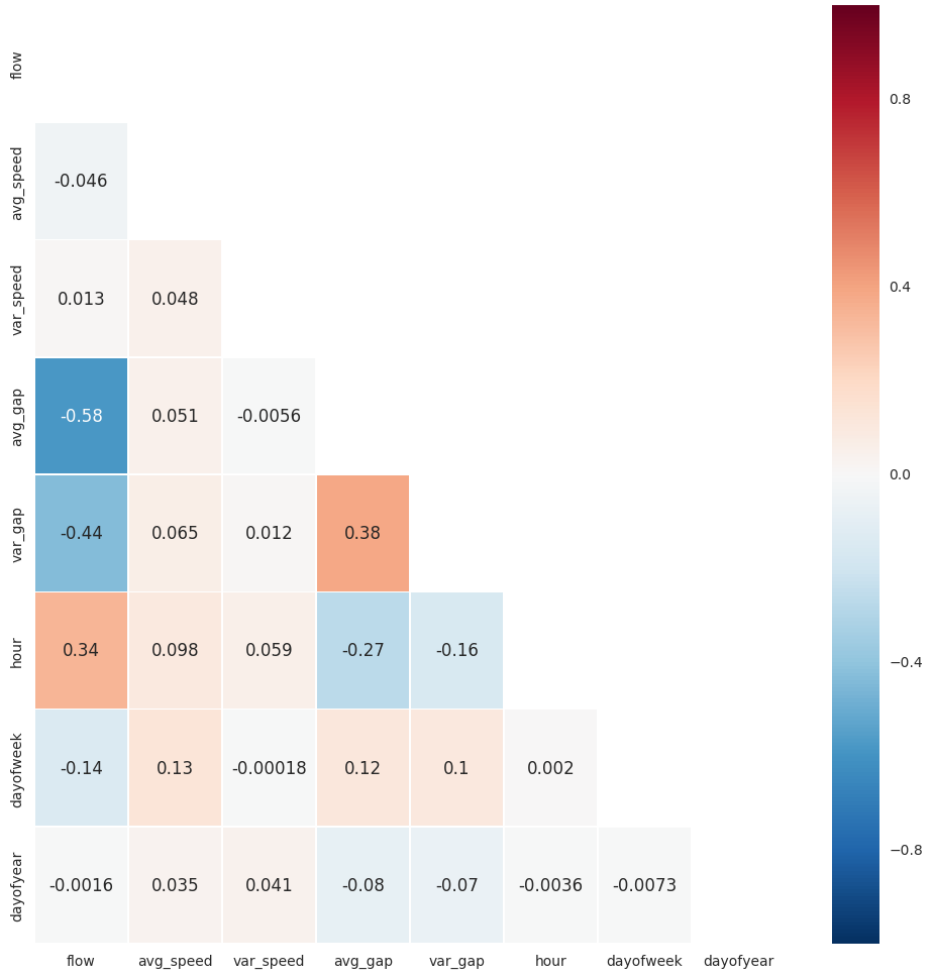


Figure 4.1: Spearman correlation heatmap

Dump 3 From 2016-01-28 22:13:17.31+00 to 2016-11-18 22:59:57.72+00 with 26 stations.

Dump 1 has mostly been used with model experimentation and data exploration. Some of the stations included were found to be virtual; they do not contain any physical RTDCE, but represent an aggregation of other stations. For that reason another dump was requested without virtual stations. Dump 2 showed

issues when it was found that traffic was missing at certain times. Therefore a third and final dump was acquired.

Choice of stations

The amount of stations using the new *Datainn* system is increasing. To make sure the stations used had enough data they were filtered by their activation date. Then the resulting stations were filtered on their location. They had to be within a rectangle area that covers most of Oslo. This gave the stations shown within the rectangle in Fig. 4.2. Most of the stations are concentrated around what is known as ring three which is the main road around Oslo city.

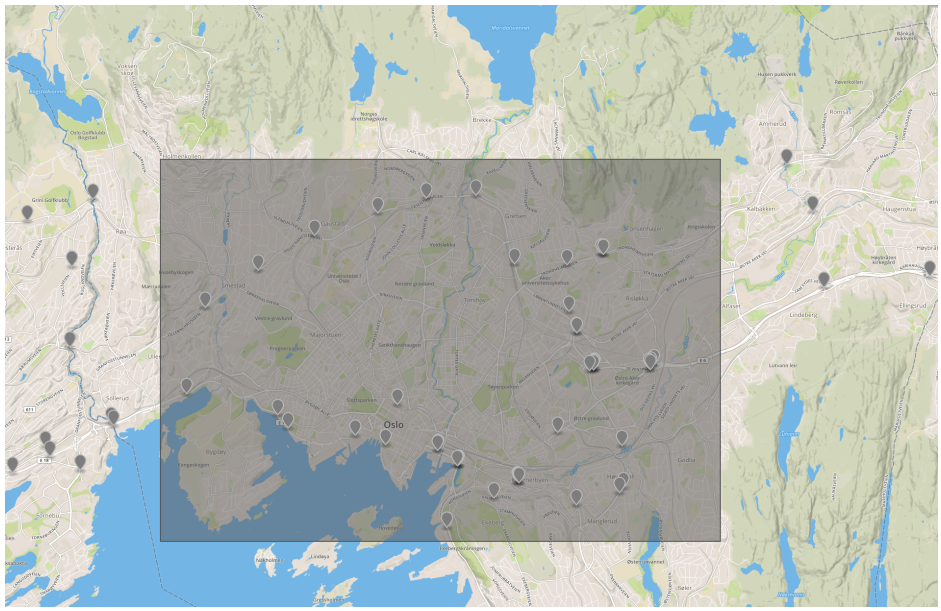


Figure 4.2: Stations in Oslo that is in dump 3. Screen capture from `http://geojson.io/`.

Storing the data

For the first data dump received from Datainn the data was formatted as newline delimited json where each row is one vehicle. In total there was 173,033,934 rows in the specified time period. Each row contains information about which stations it corresponds too. This is 7.7GB of zipped (gzip) json and approximately 140GB

unzipped. Before use the data has to be grouped by station and then aggregated in specific time intervals. Doing this in memory on the computer in use is not possible because of lack of memory.

To solve the issue of large data sets, it was decided that it was easier to store the data in a database. Decidedly, the database needed to handle time series data, not have a large data storage overhead, and be able to import data from csv. For these reasons *PostgreSQL* was selected. The files were converted to CSV and copied into the database. For further details look in Appendix B.2.

Cleaning the data

There are different suppliers of RTDCE with certain differences in quality and type of errors they produce. Therefore the data must be cleaned for such errors. The data that is filtered away is listed below:

- When length is equal to 29
- Vehicles that are classified as 11 (unknown)

The reason for filtering away vehicles that has a length of 29 is that some of the RTDCE suppliers use that value to show that there is an error in the registered event. Classification as 11 means that the RTDCE unit also had issues with classifying vehicle type.

Next, by performing small experiments it was found that there were specific stations with problematic data. The stations in question were 300047, 300224, 300142, 300145, and 300147. These stations have data in the training set, but no data in test set. While performing experiments that included these stations it was found that the models gave residual noise from the test data. The noise had a pattern respresenting actual traffic data, only with small values. Other stations that had holes in both training and test data did not have this issue. Conclusively, the models can learn the concept of stations that have missing data. As this study is not focused on the issue of missing data, the stations missing data in the test set were not used.

Aggregating data

From *Datainn* the data is row based were each row is one vehicle drive by event. The models used assumes that each row corresponds to one point in time and that each feature is one lane in one station or some other data relevant to that point in time. For this reason the raw data set needs to be aggregated into unique bins for the specified time interval, station, and lane. An Structured Query Language (SQL) query on the database aggregats the data into this format. A more in depth explanation of how this is done is presented in Appendix B.3.

The time interval to aggregate the data into varies between research. Some must adhere to the lower bound of resolution on the data used. Usually, it varies between 5 and up to 15 minutes. In this thesis 5 minutes were used because traffic jams in Norway usually dont last over too much time. If the time interval is too large then there can be an issue of loosing traffic congestion information.

In those situations were there was no traffic flow on a given point in time, the traffic flow value was set to zero. For that reason, this was done to make sure that each row in time increases by the specified time interval.

Splitting

To minimize the issue of overfitting as discussed in Section 2.1.4 the data is split in a training set, validation set, and testing set. The validation set is used for training with early stopping. Intentionally, the training set consists of the first 59% of data, the validation set of the next 13%, and test set of the last %33. As the results on the test set gives a certain degree of confidence on the generalization of the model, the test set percentage is chosen to be somewhat high. There are many variations and no exact answers to what numbers that should be selected for train, validation, and test sets. One of the disadvantage's with leaving so much data to the test set is that time series are changing over time. Seasonal changes from the training set will increase the further one gets through it.

Preprocessing the data

Section 2.1.5 discussed some techniques and the necessities of preprocessing the data. These techniques are applied to the *Datainn* data used for experimenation. Firstly, the data goes through a centered median filter with a window size of 4.

Then, since the layers in the models use the sigmoid function, the data needs to be within the range of the sigmoid function. Accordingly, Eq. (2.12) and Eq. (2.11) is used on the filterd data in that order.

Data format

Each distinct data set D_t is then split into feature vectors X_t and target vectors Y_{t+h}, where h is the prediction horizon. Next, the Y_t vectors are tranformed in representing change from last observed value. This process is explained in Section 2.1.6.

Then feature vectors X_t represent traffic flow for every station and other relevant variables for each time step t. Further, the features set is expanded with the window size r into feature matrice X'_t. In X'_t dimension zero has the features $\{X_{t-r}, \cdot, X_t\}$.

Lastly, the X'_t and Y_{t+h} vectors is split into batchs of batche size b. Now the dimensions of the feature matrices are $[b, r, |X_t|]$. For Feed Forward Neural Network (FFNN), Stacked Sparse Autoencoder (SSAE), and DNN the two last dimensions are flattened to the matrice $[b, r * |X_t|]$.

4.3.2 Selecting the hyperparameters

As shown in Section 2.1.4, some machine learning algorithms can be described as a set of mathematical formulas with parameters that is learned on the data through a training process. The parameters that are not learned through training are known as the hyperparameters of the model. When training the model, these parameters must be decided upon a priori. In many models, especially those used in this study, finding optimal hyperparameters can not be considered an exact science. The problem of choosing the optimal hyperparameters is known as hyperparameter optimization. This can either be done through previous experience or experimentation. One experimental approach is performing grid search. This is an exhaustive search on a subset of the hyperparameter space. Most parameters are defined for infinite set of discrete or continuous values and it is therefore not feasible to search the entire hyperparameter space. This method can thus be considered an approximation to finding the best parameters.

In this study hyperparameter optimization is done with the module GridSearchCV from *Scikit-learn* introduced in Section 3.2. Each grid search is done by training on the training set and cross validated with the validation set introduced in Section 4.3.1. Detailed results from hyperparameter search is found in Appendix C. Every parameter denoted with * is based on grid search results.

Firstly, in Table 4.1 the hyperparameters and the corresponding values for SSAE are presented. Then the hyperparameters for RNN, DNN, and FFNN are presented in Table 4.2.

4.4 Environment

Three different machines have been used. One personal laptop for off campus working, one desktop station for development, and one GPU server for heavy model training workloads.

Laptop Machine running *Kubuntu 16.10*. It has one 4 core *Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz*, 16 GB RAM, and *NVIDIA GeForce GTX 860M* with driver version 367.57.

Desktop station Machine borrowed from NTNU running *Ubuntu 16.04* with 16GB RAM and one 8 core *Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz*.

Table 4.1: SSAE hyperparameters for forecasting horizon experiments

Hyperparameters	**15 minutes**	**30 minutes**	**45 minutes**	**60 minutes**
Φ	Sigmoid	Sigmoid	Sigmoid	Sigmoid
AE Φ	Sigmoid	Sigmoid	Sigmoid	Sigmoid
ρ	0.09	0.09	0.09	0.09
γ	3	3	3	3
Learning rate	0.001	0.001	0.001	0.001
Optimizer	RMSProp	RMSProp	RMSProp	RMSProp
Neurons in layer*	1400	400	400	1000
Number of layers*	2	4	4	3
Batch size	1024	1024	1024	1024

Table 4.2: Hyperparameters for RNN, DNN, and FFNN. Not relevant parameter values are shown as "-".

Hyperparameters	**RNN**	**DNN**	**FFNN**
Φ	Sigmoid	Sigmoid	Sigmoid
Learning rate	0.001	0.001	0.001
Optimizer	RMSProp	RMSProp	RMSProps
Neurons in layer	1000	1000	-
Number of layers	3	3	-
Dropout	-	0.95	-
Batch size	1024	512	1024

Server Machine borrowed from NTNU running *Ubuntu 14.04*. It has one 8 core *Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz*, 12GB RAM, and *NVIDIA GeForce GTX 745* with driver version 367.57.

AWS EC2 Instance running *Amazon Linux AMI* on *p2.xlarge*. It has one *NVIDIA Tesla K80* GPU with 12GB GPU memory and 64GB RAM.

4.5 Experimental Results

In this section the experimental results from Section 4.2 is presented. All of the results presented in this section is the MRE and RMSE over every station on all the test data. A more in depth exploration of these results are presented in Section 5.1 and Section 5.2.

4.5.1 Experiment: Prediction horizon

In Table 4.3 the 15 minutes forecasting horizon results are presented. Next the 30 minutes forecasting horizon in Table 4.3. Then in Table 4.5 the 45 minutes forecasting horizon results are presented. Lastly, the 60 minutes forecasting horizon results are found in Table 4.6

Table 4.3: MRE and RMSE aggregated forecasting results with 15 minute forecasting horizon.

Error		**DNN**	**FFNN**	**HA**	**NRW**	**RNN**	**SSAE**
mre	mean	0.2641	0.4748	0.3388	0.2860	**0.2483**	0.2901
	std	0.1280	0.2660	0.1298	0.1224	0.1444	0.1366
	min	0.0973	0.1459	0.1596	0.1223	0.1033	0.1028
	25%	0.1599	0.2349	0.2545	0.1860	0.1698	0.1638
	50%	0.2534	0.4137	0.3071	0.2628	0.2096	0.2644
	75%	0.3053	0.6694	0.3992	0.3328	0.2734	0.3879
	max	0.6927	1.2266	0.9201	0.6733	0.9994	0.6904
rmse	mean	**6.5889**	8.8914	10.5951	7.6483	7.5440	6.8822
	std	2.6972	3.8278	5.0222	3.3673	4.1360	2.9851
	min	0.4277	0.4721	0.6400	0.4322	0.3249	0.3528
	25%	5.1146	6.6620	7.7134	5.8255	5.5873	5.4194
	50%	6.7511	8.9703	10.7072	7.6517	7.2716	7.1290
	75%	8.3813	11.7552	12.8302	9.7683	9.1450	8.4822
	max	12.3442	18.2138	23.9770	14.7042	21.7621	13.2489

4.5.2 Experiment: Introducing other variables

Section 4.2.2. Only RNN, DNN, and SSAE was explored. The same hyperparameters from Section 4.3.2 were applied on the models with the extra features hour of day, day of week, average gap, and variance in the gap. Too better highlight how the models performed with the extra features, the results are presented alongside the results from Section 4.5.1. Model names highlighted with * are using the extra features. For each model, the best mean result is highlighted in bold.

First, in Table 4.7 the 30 minutes forecasting horizon results are presented. Then the long-term 60 minutes forecasting horizon results in Table 4.8.

Table 4.4: MRE and RMSE aggregated forecasting results with 30 minute forecasting horizon.

Error		**DNN**	**FFNN**	**HA**	**NRW**	**RNN**	**SSAE**
mre	mean	0.2983	0.5730	0.3388	0.3585	**0.2696**	0.3110
	std	0.1471	0.3694	0.1298	0.1344	0.1456	0.1554
	min	0.1130	0.1699	0.1596	0.1718	0.1180	0.1166
	25%	0.1855	0.2941	0.2545	0.2487	0.1912	0.1861
	50%	0.2604	0.4653	0.3071	0.3338	0.2369	0.2887
	75%	0.3584	0.7507	0.3992	0.4399	0.2979	0.3843
	max	0.7699	1.9678	0.9201	0.7998	1.1083	0.9003
rmse	mean	**7.4286**	10.7957	10.5951	10.2867	8.0241	7.6657
	std	3.1011	4.8913	5.0222	4.8400	4.2761	3.4073
	min	0.4423	0.5089	0.6401	0.4291	0.3369	0.4119
	25%	5.7436	8.4515	7.7136	7.6425	5.8422	5.5293
	50%	7.7808	11.0299	10.7056	10.1987	7.7453	7.8342
	75%	9.2916	14.0491	12.8300	13.4575	10.1722	10.2149
	max	15.1435	20.1995	23.9781	19.7730	21.7272	16.5223

Table 4.5: MRE and RMSE aggregated forecasting results with 45 minute forecasting horizon.

Error		**DNN**	**FFNN**	**HA**	**NRW**	**RNN**	**SSAE**
mre	mean	0.3466	0.6708	0.3388	0.4381	**0.2828**	0.3452
	std	0.1771	0.4660	0.1298	0.1561	0.1430	0.1683
	min	0.1256	0.2127	0.1596	0.2198	0.1335	0.1306
	25%	0.2033	0.3542	0.2545	0.3183	0.2051	0.2116
	50%	0.2899	0.5297	0.3071	0.4043	0.2461	0.3102
	75%	0.4604	0.8415	0.3992	0.5485	0.3241	0.4314
	max	0.9511	3.0395	0.9202	1.0472	1.1019	0.8948
rmse	mean	**7.7903**	12.4444	10.5949	12.7097	8.2042	8.1570
	std	3.2684	5.8327	5.0222	6.1333	4.3663	3.5270
	min	0.4159	0.4822	0.6401	0.4542	0.3396	0.4206
	25%	6.1762	9.4286	7.7137	9.3384	5.8169	6.2176
	50%	7.8519	12.5498	10.7018	12.7641	7.9044	8.5284
	75%	9.7822	16.2596	12.8300	17.0287	10.3192	10.8573
	max	16.9273	24.4469	23.9791	24.6436	21.1953	17.8198

Table 4.6: MRE and RMSE aggregated forecasting results with 60 minute forecasting horizon.

Error		DNN	FFNN	HA	NRW	RNN	SSAE
mre	mean	0.3483	0.7645	0.3389	0.5235	**0.2978**	0.3658
	std	0.1649	0.5686	0.1298	0.1809	0.1477	0.1783
	min	0.1376	0.2669	0.1596	0.2653	0.1472	0.1417
	25%	0.2262	0.4052	0.2545	0.3879	0.2071	0.2177
	50%	0.3027	0.5603	0.3071	0.4714	0.2620	0.3297
	75%	0.4151	0.9656	0.3993	0.6677	0.3443	0.5001
	max	0.9993	4.0492	0.9202	1.2422	1.0796	0.9634
rmse	mean	**8.5214**	14.2305	10.5949	15.0497	8.5575	8.6406
	std	3.6643	6.5770	5.0223	7.3668	4.5314	3.8684
	min	0.4513	0.5108	0.6401	0.4747	0.3384	0.3639
	25%	6.7355	10.3321	7.7138	10.8479	6.0355	6.3065
	50%	8.8403	14.9595	10.7003	15.3142	8.4349	8.9021
	75%	10.7421	18.6253	12.8302	20.3527	10.6345	11.3757
	max	17.8644	27.8113	23.9803	29.2737	22.5929	19.5142

Table 4.7: MRE and RMSE aggregated forecasting results with 30 minute forecasting horizon.

Error		DNN	DNN*	RNN	RNN*	SSAE	SSAE*
mre	mean	**0.2983**	0.3113	**0.2696**	0.2711	0.3110	**0.3054**
	std	0.1471	0.1562	0.1456	0.1448	0.1554	0.1446
	min	0.1130	0.1220	0.1180	0.1108	0.1166	0.1179
	25%	0.1855	0.1875	0.1912	0.1832	0.1861	0.1919
	50%	0.2604	0.2725	0.2369	0.2374	0.2887	0.2877
	75%	0.3584	0.3971	0.2979	0.3152	0.3843	0.3691
	max	0.7699	0.8252	1.1083	0.9847	0.9003	0.7632
rmse	mean	**7.4286**	7.5893	**8.0241**	8.0312	7.6657	**7.3935**
	std	3.1011	3.3504	4.2761	4.2164	3.4073	3.0893
	min	0.4423	0.4428	0.3369	0.3326	0.4119	0.4234
	25%	5.7436	5.9435	5.8422	5.6625	5.5293	5.5874
	50%	7.7808	7.9319	7.7453	7.7748	7.8342	7.7005
	75%	9.2916	9.3744	10.1722	9.8270	10.2149	9.5871
	max	15.1435	15.8012	21.7272	21.5373	16.5223	15.5591

Table 4.8: MRE and RMSE aggregated forecasting results with 60 minute forecasting horizon.

Error		**DNN**	**DNN***	**RNN**	**RNN***	**SSAE**	**SSAE***
mre	mean	**0.3483**	0.3662	0.2978	**0.2763**	0.3658	**0.3590**
	std	0.1649	0.1736	0.1477	0.1485	0.1783	0.1807
	min	0.1376	0.1454	0.1472	0.1222	0.1417	0.1345
	25%	0.2262	0.2366	0.2071	0.1892	0.2177	0.2073
	50%	0.3027	0.3292	0.2620	0.2499	0.3297	0.3315
	75%	0.4151	0.4594	0.3443	0.3186	0.5001	0.4831
	max	0.9993	1.0946	1.0796	1.0997	0.9634	0.8953
rmse	mean	**8.5214**	9.2288	8.5575	**8.2944**	8.6406	**8.1595**
	std	3.6643	4.0991	4.5314	4.4434	3.8684	3.6133
	min	0.4513	0.4406	0.3384	0.3370	0.3639	0.3886
	25%	6.7355	6.9324	6.0355	5.9781	6.3065	5.9251
	50%	8.8403	9.3587	8.4349	8.2904	8.9021	8.4299
	75%	10.7421	11.6991	10.6345	10.3885	11.3757	10.6990
	max	17.8644	20.2268	22.5929	22.2886	19.5142	17.5592

Chapter 5

Evaluation and Conclusion

Section 5.1 presents the evaluation of the experimental results in Section 4.5. Further, the results are discussed in Section 5.2. Section 5.3 presents the conclusion of this study. Then

5.1 Evaluation

In this section the evaluation of the experimental results from Section 4.2 are presented. First, in Section 5.1.1 the experiment outlining forecasting results on the models for different horizons is evaluated. Then, in Section 5.1.2 the results from extending the feature vector with new variables are evaluated.

5.1.1 Experiment: Forecasting horizon

In this section, the experiments for forecasting horizons is evaluated. To evaluate and better understand the results from Section 4.5.1, the Empirical distribution function (EDF) method is applied to the results. In Section 4.1.2 EDF is explained in detail. Compared to the numerical results in Section 4.5.1 the EDF plots gives a more in depth view to how each model performed.

15 minutes

The short-term (15 minutes) EDF forecasting results are plotted in Fig. 5.1. Apparently, for all the data, most of the model have comparable, to slightly better results to Naïve Random Walk (NRW). While that is impressive results, there are slight differences when only looking at relevant data. In the case of morning rush hour, then Recurrent Neural Network (RNN), Deep Neural Network (DNN), and

Stacked Sparse Autoencoder (SSAE) beats NRW. For evening rush hours, and when traffic flow is over certain peaks, then the model performances is somewhat similar.

30 minutes

For the 30 minute forecasting results in Fig. 5.2 there are more variations in the results. Feed Forward Neural Network (FFNN) performance drops significantly in all cases. Also, for NRW the results has begun being worse than Historical Average (HA).

Interestingly, RNN starts to show a significant difference from the other models on all the data. The pattern on error difference between morning rush hour and the other glsedf results are consistant with 15 minutes forecating. Except that the difference betwen more advanced models (RNN, DNN, and SSAE) and the other have diverged more. Another distinction is the model performances when traffic flow is above 100. Surprisingly, DNN show best overall results with SSAE, while RNN lags behind.

45 minutes

In the case of 45 minute forecasting horizon in Fig. 5.3 there is significant change in the model performances for all the data and morning rush hour. Only RNN is able to get better results than HA. For evening rush hour and data from certain thresholds the same pattern as previous is repeated. All the advanced models have performance bundled together.

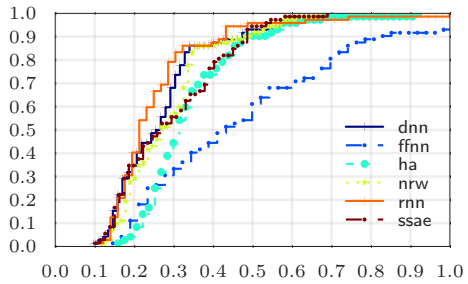
60 minutes

Lastly, the 60 minutes forecasting horizon EDFs are presented in Fig. 5.4. Except for a slight increase in forecasting error, these results give the same insights as for 45 minute forecasting horizon

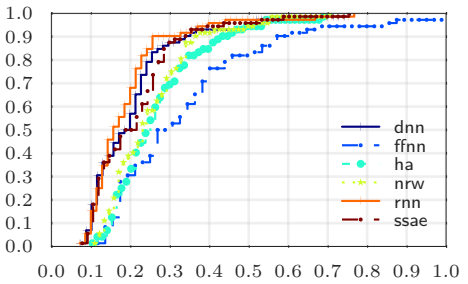
5.1.2 Experiment: Introducing other variables

In this section the experiment from Section 4.2.2 regarding extra features in the feature vector is evaluated. This experiment was applied to the for RNN, DNN, and SSAE model on mid-term (30 minutes) and long-term (60 minutes) forecasting horizons. The results were presented in Section 4.5.2.

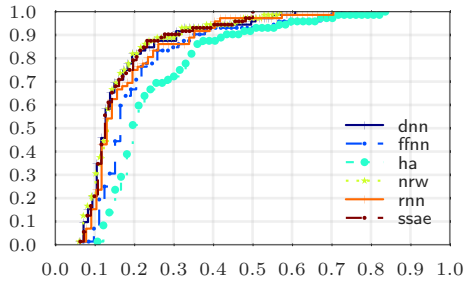
Based on the preliminary analysis by using spearman correlation on the feature vectors, it was found that there were correlation between traffic flow and the variables representing hour of day, day of the week, average gap between vehicles, and variance in the gap between vehicles. Surprisingly, the results did not show



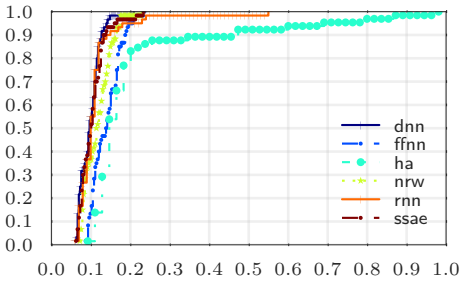
(a) Error on all test data



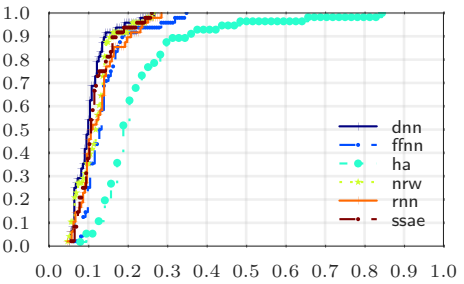
(b) Error in time of day between 06:00 to 10:00.



(c) Error in time of day between 14:00 to 17:00.

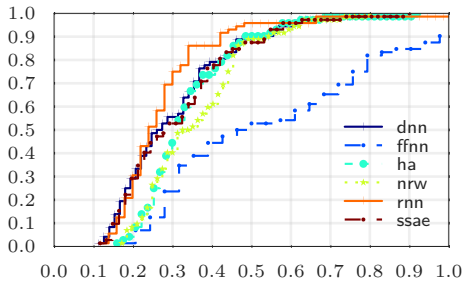


(d) Error when traffic flow is above 50.

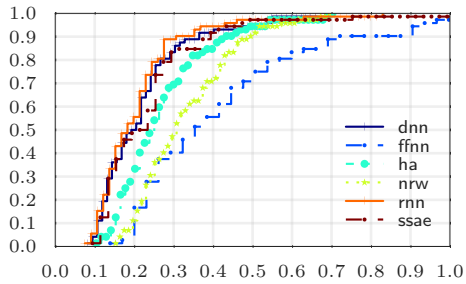


(e) Error when traffic flow is above 100.

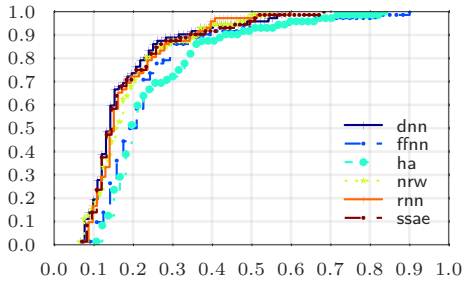
Figure 5.1: Empirical CDF of the MRE on the forecast result with 15 minute horizon.



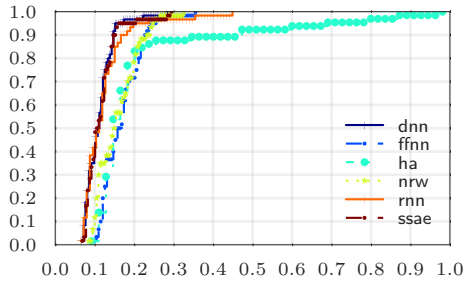
(a) Error on all test data



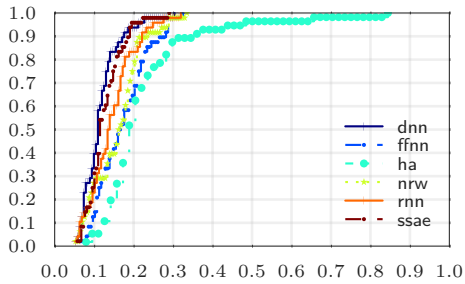
(b) Error in time of day between 06:00 to 10:00.



(c) Error in time of day between 14:00 to 17:00.

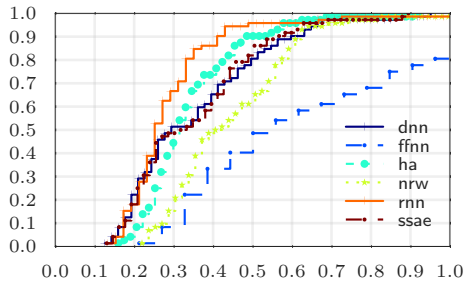


(d) Error when traffic flow is above 50.

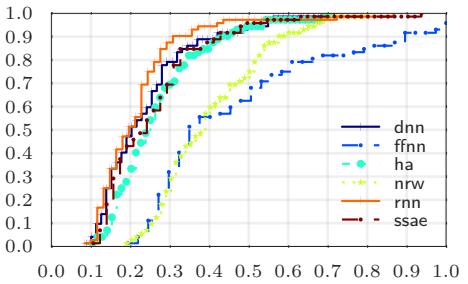


(e) Error when traffic flow is above 100.

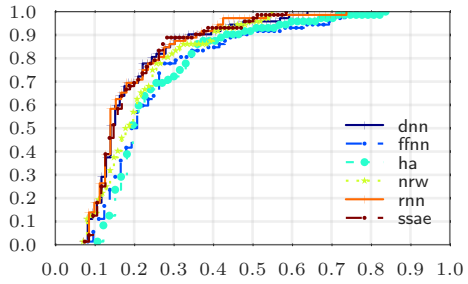
Figure 5.2: Empirical CDF of the MRE on the forecast result with 30 minute horizon.



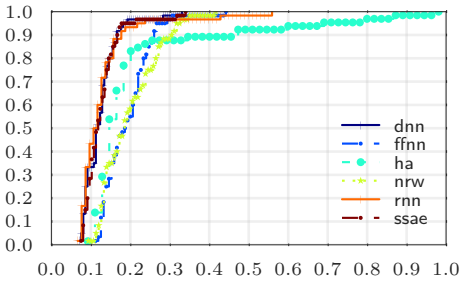
(a) Error on all test data



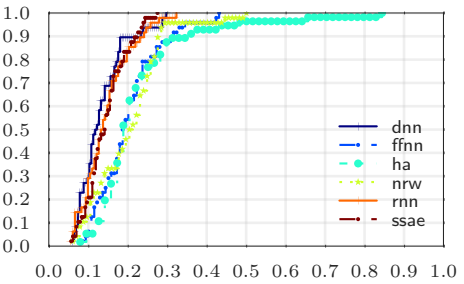
(b) Error in time of day between 06:00 to 10:00.



(c) Error in time of day between 14:00 to 17:00.

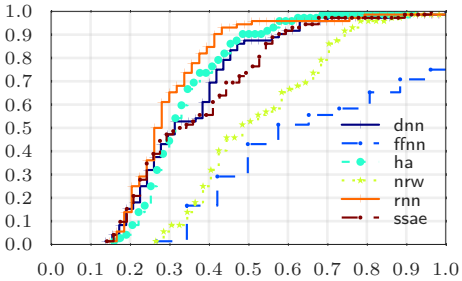


(d) Error when traffic flow is above 50.

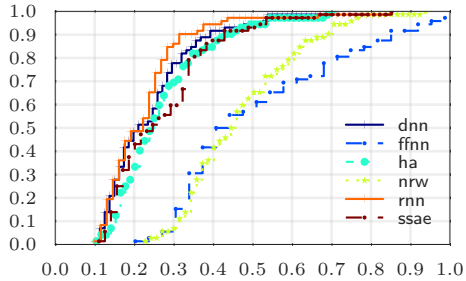


(e) Error when traffic flow is above 100.

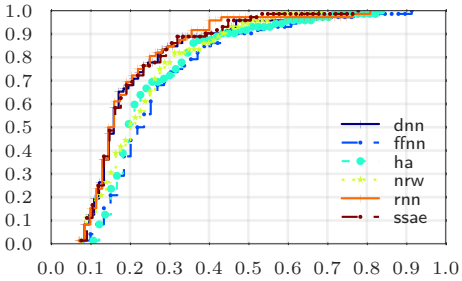
Figure 5.3: Empirical CDF of the MRE on the forecast result with 45 minute horizon.



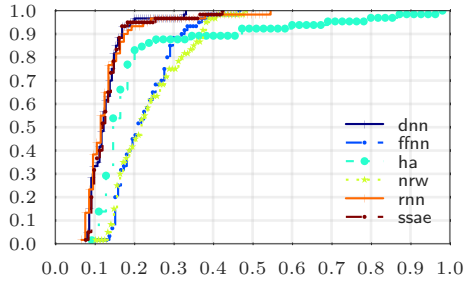
(a) Error on all test data



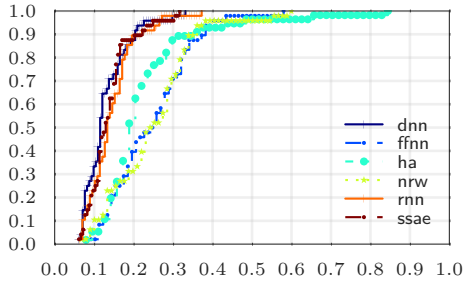
(b) Error in time of day between 06:00 to 10:00.



(c) Error in time of day between 14:00 to 17:00.



(d) Error when traffic flow is above 50.



(e) Error when traffic flow is above 100.

Figure 5.4: Empirical CDF of the MRE on the forecast result with 60 minute horizon.

much promise. For mid-term forecasting horizon only SSAE got improved prediction results. RNN and DNN results got worse, and yet better results than the improved SSAE results.

Subsequently, long-term forecasting displayed a related pattern to mid-term forecasting. The only discrepancy was with RNN which got 0.02 lower mean relative error (MRE) mean score.

5.2 Discussion

Consistently, from sec:4:exp:horizon it was found that RNN always gave the best mean MRE on all the station and DNN gave the best root mean square error (RMSE). To get a better understanding of the error distribution over the stations Section 5.1 applied and plotted EDF.

From the EDFs it was found that, the only model able to give better forecasting results on all the data was RNN. Further, the test data set results was filtered based rush hour and peak traffic flow volume thresholds.

Looking at the different test data filters revealed overall similar patterns in the performance of RNN, DNN, and SSAE over the different forecasting horizons. In some cases DNN got better results than RNN for traffic flow over 100. Since this was not a pattern over all the prediction horizons this might be the case of one badly trained instance of RNN.

The analysis also showed that error EDF gap between the advanced models (RNN, DNN, and Stacked Autoencoder (SAE)) and HA increased with a higher traffic volume threshold filter. This pattern is also found in Lv et al. [2014]. While Lv et al. [2014] had a threshold for traffic flow volume over 450 for 15 minutes, the highest threshold in this research was 100 for 5 minutes. Given that the traffic data used in that research was from California, and it can be assumed that traffic volume in Norway is lower, then this was a reasonable change. Filtering on rush hour times was also introduced to be sure that the data with the most traffic was included.

Interestingly, morning rush hour and evening rush hour have vastly different EDF error patterns. NRW results are much worse in the morning than the evening. Decidedly, this shows that morning rush hour has more variance. The advanced models results in respect to this also worse in the morning rush hour.

Since RNN has better results when the test data is not filtered, but comparable results when looking at filtered data, the model shows a better understanding of traffic flow characteristics when there is less traffic. This might be attributed to the fact that RNN is a recurrent model.

By extending the feature vector with further traffic variables, the prediction results for SSAE slightly improved, while RNN got improved results for long-term forecasting. DNN prediction result worsened for both mid-term and long-term

forecasting. These results are somewhat comparable to Yang et al. [2010] that found found slight performance increases by extending the model with weather data. Despite not having success on DNN the results show that with more feature engineering on the data can be applicable to get better prediction results. To make sense of why extra features helped SSAE and RNN, but not DNN is not easy. It might be addressed to the fact that the greedy layerwise training of the SSAE autoencoder layers combined with the sparsity constraint, helps preserve the input vector information while improving the representation of the traffic flow. For the RNN models, the Long Short-Term Memory (LSTM) cells are trained to understand the context through the "forget gates" and "input gates". The Ct vector in the LSTM cell is modified only when the weights in the gates deem it appropriate. Such mechanism's is not found in the DNN model.

5.2.1 Limitations

To make the research doable given the available time, decisions about data processing had to be made based on intuition and analysis. Some of these aspects are on the choices of aggregating interval on the raw data, keeping each lane from every station as separate features, and how many time steps that is in each feature vector. It was also, in contrast to other research, decided to not filter away weekends. Lastly, the training data set consists of the first 60% of the data. Since the data set trained on is from January to December, the trained model have not seen the seasonal changes in the test set. These are all decisions that can be further explored to get better model performance.

Further, all the models have been tested on forecasting multiple stations for each time step. To get a further intuition at how applicable these models are, then they should be further assessed against models forecasting each station separately.

Next is the issue of how the models are trained. Because of time constraints, DNN and RNN did not get thorough hyperparameter searches. Each models results should also have been assessed over multiple runs.

Lastly, the domain of Intelligent Transportation Systems (ITS) should have a few sets of data that researches could reuse to make it easier to compare research.

5.3 Conclusion

From Section 1.2 the following research questions were outlined:

RQ1 What is state of the art in traffic congestion prediction?

RQ2 What neural network based techniques have been used in traffic congestion prediction?

RQ3 How can neural networks best be used for traffic congestion prediction?

RQ4 What changes to the data and data sources have impact on the learners ability in traffic congestion prediction?

In order to answer research question 1 and 2 the Structured Literature Review (SLR) in Section 2.2 was performed. From the review it was found that a lot of different models have been explored. One key issue was clear; most of the different research was not done on the same dataset. Necessarily, this is a key problem when assessing the merit of the different models. The more classical forecasting algorithm Autoregressive Integrated Moving Average (ARIMA) and its derivatives are always mentioned as a choice depending on the forecasting horizon and accuracy needed. In general, because of the data comparison issue it is hard to assess what models can be constituted as the state-of-the-art. Although, more recent works has shown an increase in interest for deep learning based models.

From the SLR different variations on neural network models was found. Firstly, some models combining other Artificial Intelligence (AI) methods were used. Besides more standard FFNN, Neuro-fuzzy networks that is a variation of Fuzzy Logic with Artificial Neural Network (ANN) had been applied with success. On the deep learning side of neural networks, methods like Deep Belief Network (DBN), SAE, SSAE, and Spiking Neural Network (SNN) have been applied.

From the SLR a lot of research was found to focus on the issue of spatiotemporal data. Schimbinschi et al. [2015] found that spatial information have more influence on the result than temporal data. While Hu et al. [2008] found that he got better results by using station that were closer in actual data similarity. For these reasons it was assessed, to try and answer research question 3, that models using multiple features from many different stations was of interest. Once such model was SSAE from Lv et al. [2014]. Further, more recent advance with RNN with LSTM cells was included in the set of models. LSTM has shown good results on temporal data, especially in the field of Natural Language Processing (NLP). Last of the more advanced models were DNN with dropout and Rectified Linear Unit (ReLU) activation function. HA, NRW, and FFNN were included as baseline.

One of the main differences compared with previous work was that the traffic data was from Norway's capitol, Oslo. Other research has used traffic data from cities with higher traffic volume compared to Oslo.

It was found that the error distribution pattern with SSAE from Lv et al. [2014] was to a certain degree reproducible. Unfortunately, the models in this research was not tested on the same dataset as the one used in Lv et al. [2014] as some details were missing to reproduce the same data set. To get a better understanding of the error distribution over the stations, EDF was used on the test data filtered on criteria as rush hour and traffic volume thresholds. Conclusively, the results

showed that RNN outperformed every model in most cases. More interestingly, RNN had much better results with all the test data. This might be attributed to that RNN is the only model that inherently models data over time.

Lastly, the experiment in Section 4.2.2 was conducted to answer research question 4. The experiment found differing result on the performance of RNN, DNN, and SSAE. Extending the feature vector with information about each time stamp and vehicle gap negatively impacted the performance of DNN, while it had a positive effect for SSAE and RNN for long term forecasting. In Section 5.2 it was discussed that this might be an effect based on how those models work.

During the process of writing this thesis, a lot of different choices were made in how the data was processed. These details are outlines in Section 4.3.1. Based on this experience, more detailed research focused on how these choices effect model performance should be done.

Conclusively, this study found that deep learning based methods are applicable to traffic data from *Datainn* and Norwegian Public Roads Administration (NPRA). Since traffic flow on highways are inherently coupled with average vehicle speed, these methods can prove useful for new models designed to replace the *Autopass* system used by NPRA to predict travel time.

5.4 Contributions

There are two main contributions presented in this work. First, the contribution of this work presents a literature review focused on ANN based models applied to the domain of ITS.Last, this work has compares the performance of RNN, DNN, and SSAE on traffic flow forecasting in Norway. To the authors knowledge, this is the first work where RNN with LSTM cells has been applied to the domain of ITS.

5.5 Future Work

From the discussion of limitations on Section 5.2.1 in this research, there are more topics left to explore on the choices of preprocessing and data representation.

Subsequently, previous work has demonstrated that extending the feature vector with weather data should be further explored for deep learning and ITS data.

Furthermore, the RNN results showed great promise and should be further researched. While SSAE results did not show promise, there are other variations on Autoencoder (AE) that might be worth researching for ITS

While this work represents traffic flow prediction with aggregated data, systems like *Datainn* has made it possible to use individual traffic events. This approach,

unlike aggregated data, can avoid the issue of loosing time sensitive details in the traffic data. Possible models to explore are Convolutional Neural Networks (CNNs) or dynamic length RNNs.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Man, D., Monga, R., Moore, S., Murray, D., Shlens, J., Steiner, B., Sutskever, I., Tucker, P., Vanhoucke, V., Vasudevan, V., Vinyals, O., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467v2*, page 19.

Affonso, C., Sassi, R. J., and Ferreira, R. P. (2011). Traffic flow breakdown prediction using feature reduction through Rough-Neuro Fuzzy Networks. In *The 2011 International Joint Conference on Neural Networks*, pages 1943–1947. IEEE.

Barros, J., Araujo, M., and Rossetti, R. J. F. (2015). Short-term real-time traffic prediction methods: A survey. In *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 132–139. IEEE.

Chan, K. Y., Dillon, T. S., Singh, J., and Chang, E. (2012). Neural-Network-Based Models for Short-Term Traffic Flow Forecasting Using a Hybrid Exponential Smoothing and Levenberg–Marquardt Algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):644–654.

Commision of the European Communities (2001). European transport policy for 2010: time to decide.

Dunne, S. and Ghosh, B. (2013). Weather Adaptive Traffic Prediction Using Neurowavelet Models. *IEEE Transactions on Intelligent Transportation Systems*, 14(1):370–379.

Fusco, G., Colombaroni, C., Comelli, L., and Isaenko, N. (2015). Short-term traffic predictions on large urban traffic networks: Applications of network-based ma-

chine learning models and dynamic traffic assignment models. In *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 93–101. IEEE.

Garibay, I. (2010). *Dario Floreano and Claudio Mattiussi (eds): Bio-inspired artificial intelligence: theories, methods, and technologies*, volume 11.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256.

Gu, Y. and Yu, L. (2010). Study on Short-Time Traffic Flow Forecasting Methods. In *2010 International Conference on Logistics Engineering and Intelligent Transportation Systems*, pages 1–4. IEEE.

Guan, W., Cai, X., Wei Guan, Xiaolei Cai, Guan, W., and Cai, X. (2005). A practical model of dynamic forecasting of urban ring road traffic flow. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2005:1050–1055.

Guo, F., Polak, J. W., and Krishnan, R. (2010). Comparison of modelling approaches for short term traffic prediction under normal and abnormal conditions. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1209–1214. IEEE.

Guozhen Tan, Wenjiang Yuan, and Hao Ding (2004). Traffic flow prediction based on generalized neural network. In *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pages 406–409. IEEE.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, pages 1–18.

Hochreiter, S. (1997). Long Short-Term Memory. *Neural Computation*.

Hou, Y., Edara, P., and Sun, C. (2015). Traffic Flow Forecasting for Urban Work Zones. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1761–1770.

Hu, C., Xie, K., Song, G., and Wu, T. (2008). Hybrid Process Neural Network based on Spatio-Temporal Similarities for Short-Term Traffic Flow Prediction. In *2008 11th International IEEE Conference on Intelligent Transportation Systems*, pages 253–258. IEEE.

Huang, W., Hong, H., Song, G., and Xie, K. (2014a). Deep process neural network for temporal deep learning. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 465–472. IEEE.

Huang, W., Song, G., Hong, H., and Xie, K. (2014b). Deep Architecture for Traffic Flow Prediction: Deep Belief Networks With Multitask Learning. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2191–2201.

Jeong, Y.-S., Byon, Y.-J., Castro-Neto, M. M., and Easa, S. M. (2013). Supervised Weighting-Online Learning Algorithm for Short-Term Traffic Flow Prediction. *IEEE Transactions on Intelligent Transportation Systems*, 14(4):1700–1707.

Kofod-petersen, A. (2014). How to do a Structured Literature Review in computer science. pages 1–7.

Liu, B.-s., Li, Y.-j., Yang, H.-t., Sui, X.-s., and Niu, D.-f. (2006). Research on Forecasting Model in Short Term Traffic Flow Based on Data Mining Technology. In *Sixth International Conference on Intelligent Systems Design and Applications*, volume 1, pages 707–712. IEEE.

Lv, Y., Duan, Y., Kang, W., Li, Z., and Wang, F.-Y. Y. (2014). Traffic Flow Prediction With Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):1–9.

Mitchell, T. M. (1997). *Machine Learning*.

Moussavi-Khalkhali, A., Jamshidi, M., and Chair, L. B. E. (2014). Leveraging Machine Learning Algorithms to Perform Online and Offline Highway Traffic Flow Predictions. In *2014 13th International Conference on Machine Learning and Applications*, pages 419–423. IEEE.

Nguyen, N. N. and Quek, C. (2012). Traffic prediction using a Generic Self-Evolving Takagi-Sugeno-Kang (GSETSK) fuzzy neural network. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE.

Oh, S.-d., Kim, Y.-j., and Hong, J.-s. (2015). Urban Traffic Flow Prediction System Using a Multifactor Pattern Recognition Model. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2744–2755.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach, 3rd edition*.

Schimbinschi, F., Nguyen, X. V., Bailey, J., Leckie, C., Vu, H., and Kotagiri, R. R. R. R. (2015). Traffic forecasting in complex urban networks: Leveraging big data and machine learning. In *2015 IEEE International Conference on Big Data (Big Data)*, number January, pages 1019–1024. IEEE.

Tu, E., Kasabov, N., and Yang, J. (2016). Mapping Temporal Variables Into the NeuCube for Improved Pattern Recognition, Predictive Modeling, and Understanding of Stream Data. *IEEE transactions on neural networks and learning systems*, PP(99):1–13.

Yang, J.-Y., Chou, L.-D., Li, Y.-C., Lin, Y.-H., Huang, S.-M., Tseng, G., Wang, T.-W., and Lu, S.-P. (2010). Prediction of short-term average vehicular velocity considering weather factors in urban VANET environments. In *2010 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3039–3043. IEEE.

Zhu, J. and Zhang, T. (2009). A Layered Neural Network Competitive Algorithm for Short-Term Traffic Forecasting. In *2009 International Conference on Computational Intelligence and Software Engineering*, pages 1–4. IEEE.

Appendices

Appendix A

Structured Literature Review Protocol

This section describes a structured literature process based on Kofod-petersen [2014]. The papers are gathered from IEEE Xplore[1].

A.1 Identification of research

When building the search term one needs to identify the concepts in which one needs information based on the research questions explained in Section 1.2. Table A.1 shows the concepts used for the search terms.

Table A.1: Search terms used for identification of research

Concerns	Search terms
Domain	Traffic
Problem	Queue Congestion Flow Prediction Forecasting Estimation
Techniques	Deep learning Neural networks Machine learning Big data

Based on these concepts a search string was build were each concept are AND statements and the terms within a concept are OR statements. The result is shown below:

```
traffic AND
```

[1] `http://ieeexplore.ieee.org/Xplore/home.jsp`

```
(queue OR congestion OR flow OR prediction OR forecasting OR
    estimation) AND
("deep learning" OR "neural networks" OR "machine learning" OR "
    big data")
```

Listing A.1: Search statement

The given search statement gave over 1800 results over different publications. Looking at the publication titles on the result gave hint to those of relevance. Therefore, to reduce the list to a comprehensible amount, the search string was extended with:

```
("Publication Title":"big data" OR
"Publication Title":"intelligent systems" OR
"Publication Title":"computational intelligence" OR
"Publication Title":"machine learning" OR
"Publication Title":"data science" OR
"Publication Title":"neural")
```

Listing A.2: Search statement for publication titles

A.2 Selection of primary studies

The search was performed on 02. March 2016 and resulted in 500 papers. The amount of papers was too much to reasonably handle. Selection of these papers was thus done by the criteria described in the list below based on the papers respective titles.

Inclusion criteria 1 The study main concern is Intelligent Transportation Systems (ITS)

Inclusion criteria 2 The study utilizes machine learning techniques.

Inclusion criteria 3 The study predicts traffic variables

The paper selection filtering process resulted in 114 probably relevant papers.

A.3 Study quality assessment

This step is performed in three phases. Every paper will be reviewed in accordance to the criteria described in Appendix A.3. The first two inclusion criteria is regarded as primary criteria and the remaining as secondary. The inclusion criteria is rated as either fail, not answerable, or approved. The quality criteria can get a score from 0 to 1.

Inclusion criteria 1 The main concern of the study is prediction of traffic data.

Inclusion criteria 2 The study is a primary study presenting empirical results

Inclusion criteria 3 The study focuses on Artificial Neural Network (ANN) models.

Inclusion criteria 4 The study describes the models in a way clearly reproducible.

Inclusion criteria 5 The study describes traffic prediction in context to other data sources e.g. weather data near stations.

Quality criteria 1 There is a clear statement of the aim of the research

Quality criteria 2 The study is put into context of other studies and research

A.3.1 Phase 1: Abstract inclusion criteria screening

Only the abstracts was read in this phase. Reading the abstract gives some idea to whether the different criteria are fulfilled or not. Getting a decisive answer was not prioritized. The criteria that had to be fulfilled was the first three. This phase reduced the amount of papers to 94.

A.3.2 Phase 2: Full text inclusion criteria screening

In this phase the papers were read backwards. By reading the conclusions, test results, and model explanation conclusions were made whether they fulfilled the inclusion criteria. The primary criteria had to be fulfilled and the secondary could not fail in more than one. The quality criteria had to get at least more than 1.5 in score.

After this step was completed there was 21 approved papers.

A.3.3 Phase 3: Full text quality screening

The papers left in this phase had a full text read. They were reviewed according to the quality criterion described in Appendix A.3.3. Each quality criterion is answered from 0 to 1.

Quality criterion 1 Are design decisions justified?

Quality criterion 2 Is the method thoroughly explained?

Quality criterion 3 Is the experiment procedure thoroughly explained?

Quality criterion 4 Is it clearly stated what the method/algorithm has been compared against?

Quality criterion 5 Has the performance and performance metrics been discussed?

Quality criterion 6 Have the results been thoroughly analysed?

Quality criterion 7 Are the experiments quantitative?

Quality criterion 8 Are the findings supported?

A.3.4 End result

Table A.2: The papers and their respective scores after phase 3

Paper	QC1	QC2	QC3	QC4	QC5	QC6	QC7	QC8
Tu et al. [2016]	1	1	1	1	1	1	1	1
Schimbinschi et al. [2015]	1	1	1	1	0.5	1	1	1
Fusco et al. [2015]	1	0.5	1	1	1	1	1	0.5
Oh et al. [2015]	1	1	1	1	1	1	1	1
Hou et al. [2015]	1	1	1	1	1	1	1	1
Huang et al. [2014b]	1	1	1	1	1	1	1	1
Lv et al. [2014]	1	1	1	1	0	1	1	1
Huang et al. [2014a]	1	1	1	1	1	1	1	1
Moussavi-Khalkhali et al. [2014]	0.5	1	1	1	0.5	0.5	1	1
Dunne and Ghosh [2013]	1	1	1	1	1	1	1	1
Jeong et al. [2013]	1	1	1	1	1	1	1	1
Chan et al. [2012]	1	1	1	1	1	1	1	1
Nguyen and Quek [2012]	1	1	1	1	0.5	0.5	1	1
Affonso et al. [2011]	1	0.5	0	1	0.5	0.5	1	1
Yang et al. [2010]	1	1	1	1	1	0.5	1	1
Guo et al. [2010]	1	1	1	0.5	1	1	1	1
Gu and Yu [2010]	0.5	1	0.5	0.5	0.5	1	1	1
Zhu and Zhang [2009]	1	1	1	1	0.5	0.5	1	1
Hu et al. [2008]	1	1	1	1	1	1	1	1
Liu et al. [2006]	0.5	1	0.5	1	0.5	0.5	1	1
Guan et al. [2005]	1	1	1	1	1	0.5	1	1
Guozhen Tan et al. [2004]	1	1	1	1	0	0.5	1	1

Appendix B

Data

In this appendix all the metadata from the data set is presented and the respective code to preprocess it from the database.

B.1 Data dumps from Datainn

Table B.1 lists all the dumps taken from the Datainn system with the respective date interval and format. Each station contained in the data dumps is listed in Table B.2.

Table B.1: Data dumps and meta data

Dump	**Date interval**	**Format**
Dump 1	2016-02-01 00:00:01.79+00 - 2016-09-11 03:47:24.979+00	JSON
Dump 2	2016-01-29 00:00:00.679+00 - 2016-11-03 06:52:09.25+00	CSV
Dump 3	2016-01-28 22:13:17.31+00 - 2016-11-18 22:59:57.72+00	CSV

Table B.2: Data dumps and stations

Dump	**Stations**
Dump 1	300055, 300108, 300142, 300152, 300107, 300160, 300140, 300072, 300153, 300164, 300165, 300224, 300030, 300163, 300162, 300166, 300155, 300167, 300051, 300093, 300151, 300049, 300016, 300154, 300156, 300139, 300047
Dump 2	-

Table B.2: Data dumps and content (continued)

Dump	Stations
Dump 3	300001, 300016, 300030, 300047, 300083, 300093, 300099, 300139, 300140, 300141, 300142, 300144, 300145, 300147, 300148, 300151, 300152, 300153, 300154, 300160, 300162, 300163, 300166, 300224, 300225, 300233

B.2 Importing data to PostgreSQL

PostgreSQL supports file content importing from CSV. The first data dump was given in JSON format and had to be preprocssed to CSV.

Finally when the data is ready to be copied into the database table a named pipe file is created by the command `mkfifo namedpipe`. This is done since the files are zipped, and unzipping them will take too much space. Subsequently in a *PostgreSQL* console one must run Listing 1. After that is done then CSV data can be piped into the named pipe as so `zcat filepath/filename.csv.gz > namedpipe`.

B.3 Aggregate data

Before the data is aggregated based on an interval start s, end e, and interval gap Δ the interval end is recalculated to make sure that there is no remainder that does not fit the Δ. Given Eq. (B.1) the interval used to get the aggregated data from *PostgreSQL* is (s, e').

$$e' = \left\lfloor \frac{e-s}{\Delta} \right\rfloor * \Delta + s \tag{B.1}$$

```
COPY oslo_traffic(
    region_id,
    county_id,
    speed,
    vehicle_number,
    timestamp,
    qspeed,
    eventd_date_time,
    gap,
    measure_point_number,
    vehicle_type,
    event_number,
    lane,
    vehicle_type_quality,
    contains_all_required_fields,
    length
)
FROM '/home/jovyan/work/namedpipe'
WITH csv header;
```

Listing 1: SQL for copying CSV data

```
SELECT
    oslo_traffic.measure_point_number||'-'||oslo_traffic.lane
        AS mpn,
    windows.start::TIMESTAMP WITH TIME ZONE as timestamp,
    count(oslo_traffic) as flow,
    avg(oslo_traffic.speed) as avg_speed,
    var_samp(oslo_traffic.speed) as var_speed,
    avg(oslo_traffic.gap) as avg_gap,
    var_samp(oslo_traffic.gap) as var_gap
FROM (
    -- generates a list from start to end with delta interval
    SELECT
        generate_series(start, end - delta, delta) AS start
) AS windows
    LEFT JOIN
        oslo_traffic
    ON
        timestamp >= windows.start AND timestamp < windows.start
            + delta
WHERE
    oslo_traffic.deleted ISNULL
    AND contains_all_required_fields IS TRUE
    AND length != 29
GROUP BY oslo_traffic.measure_point_number, oslo_traffic.lane,
    windows.start
ORDER BY oslo_traffic.measure_point_number, oslo_traffic.lane,
    windows.start
```

Listing 2: SQL code that aggregates time series within intervals per station and lane. Correct table name, start interval, end interval, and interval delta must be set.

Appendix C

Hyperparameters

In this appendix the results from the hyperparameter searches are presented.

C.1 Experiment: Forecasting horizon

Hyperparameter searches were performed on the Stacked Sparse Autoencoder (SSAE) model for each forecasting horizon. To restrict the search space the only parameters searched over was the amount of layers and how many neurons each layer had. Search method was grid search which means that the model is trained and evaluated the crossproduct over all search parameter times. The parameter values search on are listed in Table C.1.

Table C.1: Grid search parameters that were searched over and the values used for each parameter.

Neurons in layer	Number of layers
2	400
3	800
4	1000
	1400

Subsequently, the other parameter values were selected based on previous experiments and simple tests. These values are listed in Table C.2.

For training, the early stopping algorithm was used. For each 20th step the test error was evaluated and if the test error had not improved over 500 checks, then that was used as the final result.

Table C.2: Default parameters for SSAE.

Parameter	Value
Activation function in prediction layer	Sigmoid
Activation function in AE	Sigmoid
ρ	0.09
γ	3
Learning rate	0.001
Optimizer	RMSProp

C.1.1 15 minutes

For 15 minute forecast horizon a batch size of 1024 was used. Results are shown in Table C.3.

Table C.3: SSAE grid search results for 15 minute forecasting horizon.

Neurons in layer	Number of layers	Test score
1400	2	-0.001152
1400	3	-0.001157
800	2	-0.001158
400	3	-0.001166
1400	4	-0.001179
800	3	-0.001193
400	4	-0.001197
1000	3	-0.001199
800	4	-0.001201
1000	4	-0.001210
400	2	-0.001229
1000	2	-0.001287

C.1.2 30 minutes

For 30 minute forecast horizon batch size of 512 was used. Results are show in Table C.4.

Table C.4: SSAE grid search results for 30 minute forecasting horizon.

Neurons in layer	Number of layers	Test score
400	4	-0.001048
800	2	-0.001055
1400	3	-0.001056
1400	4	-0.001065
400	2	-0.001070
1000	4	-0.001070
1000	3	-0.001084
1000	2	-0.001095
1400	2	-0.001110
800	3	-0.001114
400	3	-0.001121
800	4	-0.001130

C.1.3 45 minutes

For 45 minute forecast horizon batch size of 512 was used. Results are show in Table C.5.

Table C.5: SSAE grid search results for 45 minute forecasting horizon.

Neurons in layer	Number of layers	Test score
400	4	-0.001439
800	4	-0.001447
800	2	-0.001471
400	3	-0.001484
1400	3	-0.001488
400	2	-0.001493
1400	4	-0.001504
800	3	-0.001508
1000	2	-0.001521
1400	2	-0.001536
1000	4	-0.001552
1000	3	-0.001558

C.1.4 60 minutes

For 60 minute forecast horizon batch size of 512 was used. Results are show in Table C.6.

Table C.6: SSAE grid search results for 60 minute forecasting horizon.

Neurons in layer	Number of layers	Test score
1000	3	-0.001580
800	4	-0.001590
400	4	-0.001596
800	2	-0.001599
1400	4	-0.001600
800	3	-0.001606
400	2	-0.001609
1400	3	-0.001632
400	3	-0.001641
1000	4	-0.001666
1000	2	-0.001668
1400	2	-0.001670