Technical University of Munich

Chair of Transportation Systems Engineering

Master's Thesis

Deep Learning Techniques for Short - Term Traffic Forecasting

Artyom Kholodkov

Supervised by:

Dr. Tao Ma

Chairholder: Professor Dr. Constantinos Antoniou

September 26, 2019

Abstract

Accelerated evolution of intelligent transport systems (ITS) promotes the need to offer advanced methods of traffic forecasting. Intelligent predictive models are trained using the historical data obtained from the loop-detectors installed under road pavement. At the same time, increased availability of data made it possible to apply deep-learning techniques to provide traffic flow forecasting on road links and in networks.

In this thesis, the data collected from loop detectors in Nicosia, Cyprus is used to provide the short-term univariate time-series forecasting of traffic flow on links. First, the data is properly preprocessed to make it appropriate for deep learning algorithms. Data is collected over three consecutive months leading to the shortage of training batches that are fed into the neural networks for time – series analysis. To eliminate the issue of data shortage, the data was extended to constitute enough training batches. Then, the traffic data is aggregated into 5 minuted intervals and reshaped into the multidimensional tensors to create the training batches used by deep learning framewroks, and the process of preprocessing and reshaping is described. Four most promising models are compared in terms of speed of convergence and accuracy - conventional recurrent deep neural networks, deep neural networks with LSTM cells, deep neural networks with GRU cells and WaveNet.

The results demonstrate that there is not a significant difference between the models in terms of accuracy. However, the WaveNet architecture converges much faster than other models making it the most promising solution for short-term traffic forecasting on links.

Declaration of authorship

I hereby confirm that this thesis is my own work. All the materials and sources used in this thesis are referenced and acknowledged. This is an unpublished version of my thesis for the purpose to meet partial requirement of the degree of Master of Sciense at Technical Univresity of Munich. The views and opinions expressed in this thesis are those of the author.

Munich, September 26, 2019

Artyom Kholodkov

Table of Contents

List of tables

List of figures

Chapter 1. Introduction

1.1 Problem statement

The rapid improvement of computer hardware and increased data availability have led to accelerated development of a subset of Artificial Intelligence (AI) and machine learning – deep learning. Recently, deep learning architectures have been applied to the areas that were not possible in the past – computer vision, speech recognition, natural language processing, machine translation, and many more.

When researchers talk about AI, they imply a wide variety of methods that make machines smart. Machine Learning and Deep Learning are considered to be the subsets of artificial intelligence and, despite of being quite similar, share several of key differences. Pioneer in the field of artificial intelligence Artur Samuel defined the term machine learning as: "Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed." Machine learning algorithms use structured training data to produce the desired output using the input that is not included in the training set. Being a subset of machine learning, deep learning performs the same task using the training data that should not necessarily be structured or labeled. Deep learning algorithms are obtained by composing multiple layers of non-linear modules where each module transforms the data at one level into a representation at a higher and more abstract level. With enough such transformations, very complex functions can be learned (LeCun, Bengio, Hinton, 2015). Hence, deep learning techniques might be crucial in traffic forecasting due to the high nonlinearity and complexity of traffic flow (Yu, Yin, Zhu, 2018). Figure 1.1 presents the relationship between Artificial Intelligence, Machine Learning and Deep Learning.

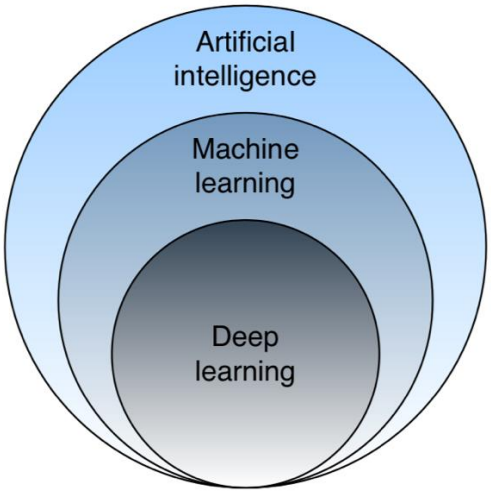


Figure 1.1: Difference between artificial intelligence, machine learning and deep learning

Advancements in deep learning did not pass by the transportation industry. Lately, the transportation sector across the globe faces the problems of an increased travel demand, CO-2 emissions, safety concerns and environmental degradation. These problems emerge from the steady growth of rural and urban traffic because of the growing population and the tendency to relocate to cities. Solving the transportation issues might become a challenge when the travel behavior is hard to predict using the conventional statistical methods and mathematical rules (Abduljabbar, Dia, Liyanage, 2019).

The rapid development of intelligent transport systems (ITS) facilitates the need to propose advanced methods of traffic forecasting. Intelligent predictive models are trained using the historical data obtained from the loop-detectors installed under road pavement. Those data are used as an input to various AI algorithms for real-time short-term or long-term forecasting (Abduljabbar, Dia, Liyanage, 2019). Several research articles propose multiple approaches to achieve short-term traffic prediction. In most of the cases, the researchers suggest the usage of recurrent neural networks with LSTM cells, a combination of recurrent and convolutional neural networks to capture the spatial dependencies of traffic in networks. However, little attention is paid to short-term traffic forecasting on links considering the inaccessibility of a large amount of data. Moreover, researchers tend to concentrate on specific methods without paying attention to comparing different models and tuning hyperparameters to achieve the best forecast possible. Therefore, this thesis is motivated by the necessity to identify the most

promising deep learning techniques to provide the best forecasting capabilities on road links in the face of a lack of data.

1.2 Objectives and research questions

The objective of this thesis is to determine which deep learning technique is preferable for the short-term traffic forecasting on links considering the shortage of data (approximately 500 thousand observations in each direction). Since the raw data are usually aggregated into 5 minutes intervals and reshaped into a three-dimensional array, it leads to the shortage of the training batches that are fed into the neural networks for time – series analysis.

The mathematical representation of four different deep learning models is provided. Then, the prediction accuracies of the models are compared, and the benefits and drawbacks of these models are discussed. The main objective of this research is associated with a number of secondary research questions that will be discussed further:

- Which features have to be extracted from raw data to provide the time series forecasting?
- How to clean and preprocess the raw data?
- How data aggregation is performed?
- Which methods should be applied to reshape the raw data to be used in deep learning algorithms?
- How to overcome the issue of lack of the training batches necessary for the model training?
- Which trained model provides the least loss and leads to better forecasting?
- How to tune the hyperparameters of each deep learning model such as the number of hidden layers, the number of neurons in each layer, and the learning rate?

1.3 Expected contributions

This thesis is expected to have the following practical and methodological contributions:

- Practical contributions:
    - This study might be useful to traffic managers and other stakeholders for the

purpose of short-term traffic prediction.
    - The coding techniques used in case study section might revised changed by practitioners to fit specific needs and used in ITS software.
- Methodological contributions:
    - The following study provides an insight on how to preprocess raw data and reshape it into tensors containing features and targets, which is useful for further research as the process is always similar for recurrent networks.
    - The study also presents the mathematical background behind the deep learning techniques used for traffic forecasting, allowing to acquire a thorough understanding of the traffic prediction problem.

1.4 Research framework

The following study consists of four main stages: preliminary stage, methodology stage, case study and discussion and conclusion. All stages follow one after another and organized in a linear fashion.

The preliminary stage consists of formulating the study problem, objectives, and research question. A literature review is presented in which the researchers examine the problem of short-term traffic forecasting, methodologies used in these works are considered, critical comments on these articles are given, and the necessity of applying these methodologies in the case study are discussed.

Methodology stage provides an overview of deep learning algorithms used in case study and their mathematical representation, advantages and disadvantages of these algorithms are discussed. It also provides an insight on data preparation and reshaping into multi-dimensional arrays with targets and features.

In case study the description of the study area is provided, as well as characteristics and location of loop detectors in the road network in Malta. The results of preprocessing, cleansing and aggregating the real data collected from three different loop detectors in different directions are demonstrated. Then, the resulting data are used in deep learning algorithms described in the

methodology section to provide the short term flow forecasting according to directions. The performance of all models in terms of final predictions is demonstrated.

In the discussion and conclusion stage, the careful comparison of the results of forecasting provided by all available models are discussed. Based on these results, the best model is identified, and its advantages and limitations are discussed. The stage is summarized by the potential prospect of further research. Figure 1.2 illustrates the given framework:

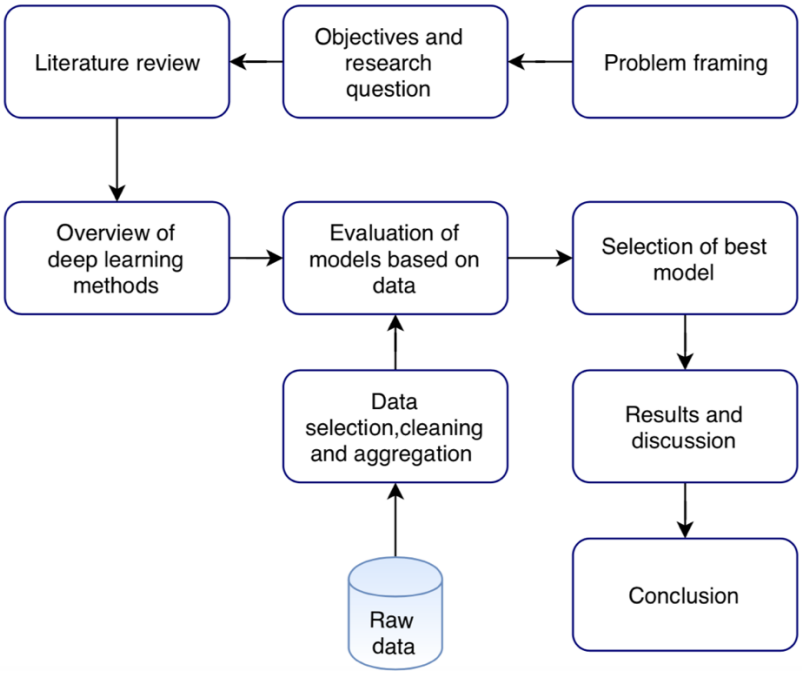


Figure 1.2: Research framework

1.5 Report structure

The rest of this paper is organized in the following manner. Chapter 2 provides the literature review of the research articles on the topic of contemporary methods of deep learning in traffic forecasting. In chapter 3, the methodology is proposed, including the description of deep learning algorithms and their mathematical form as well as their benefits and limitations, data cleansing, preprocessing, and aggregation methods. Chapter 4 presents the evaluation of deep – learning models on real data leading to the final results of the forecasting. Chapter 5 discusses the results achieved in the case study are analyzed, and the best model in terms of the least cost

and highest accuracy is determined. The conclusion sums up the results of the study and elaborates on potential further research.

Chapter 2. Literature review

In this chapter, the literature review is provided to acquire a profound understanding of the problem and develop the methods leading to efficient short-term traffic forecasting techniques. First of all, the overview of the available forecasting methods is provided. Then, a mixture of LSTM and convolutional networks is considered to capture the spatial-temporal components in networks and what components of this architecture can be derived to adjust the proposed methodology to forecasting on links. After this, similar architecture is considered to capture both long-term and short-term patterns in time series data, and conclusion about the applicability of this technique is discussed. Then, there is an analysis of recurrent neural networks with a different type of cells and discussion about their role in short term traffic forecasting. Finally, the chapter is summed up by giving the representation of the WaveNet neural network intended for traffic forecasting.

2.1 Overview of predictive methods for short-term traffic forecasting

Since the comparison of deep learning techniques with other methods is provided in most of the scientific literature, it is necessary to give a general overview of available predictive systems for short-term traffic forecasting. Van Lint and Hoogendoorn conducted an comprehensive study giving an overview of predictive models for short-term traffic forecasting. According to the authors, it is feasible to split the forecasting techniques into three broad categories: parametric, non-parametric and naive (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

1. **Non-parametric models**

Non-parametric models imply that the number of parameters of a model is flexible and not fixed, and the model structure and parameters need to be prepared using the available data. Usually, in comparison with the other two methods, the amount of data should be substantial. The advantage of such models is that the complex non-linear relationships in traffic variables can be found. On the other hand, the drawback of these models is that unseen cases and outliers might influence the model's structure as it is determined from data (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007). The list of non-parametric models is as follows:

K-nearest neighbour. k-Nearest Neighbor method implies the search for the k events in a historical database which are the closest to the current traffic condition. The studies demonstrate

that it is a fast method that might exceed naïve prediction models but is less accurate than advanced methods (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

Fuzzy logic. Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any number between 0 and 1. The model is based on the concept of partial truth where the truth value ranges between completely true and completely false. The model might produce promising results. However, backpropagation neural networks might produce better predictions (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

Neural networks. Neural networks are considered the most popular models for traffic prediction as they are capable of modelling complex non-linear relationships. Different variations of neural networks exist depending on the training procedure, techniques of preprocessing input data, internal structure, including spatial or temporal patterns explicitly in the models. The traditional neural network is backpropagation neural network which consists of an input layer, one or more hidden layers and an output layer. The backpropagation algorithm implies that the input is fed into the input layer, then the error is calculated based on one or several outputs using an appropriate objective function, and the gradient vector is determined by propagating the error backwards through the network, the parameters are adjusted. The research articles suggest that a conventional neural network might produce consistent results in terms of the traffic forecasting, but extensions are needed to provide better accuracy (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

Bayesian networks. A Bayesian network is a probabilistic graphical model that describes a set of variables and their conditional dependencies through an acyclic graph. In Bayesian networks, the data from nearby links are informative to the current link under examination. The method might provide a reasonable accuracy but are inferior to the neural networks van Hinsbergen, (van Lint, Van Zuylen, Sanders, 2007).

2. Parametric models

Parametric models indicate that the parameters of the model need to be found using data, the structure of the model, and the number of parameters are strictly predetermined. The advantage of such models is that the unseen cases and incidents can be captured. Another advantage is the

necessity of fewer data. Some models provide good accuracy and are less demanding in terms of the computational effort (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

Traffic simulation models were developed early in 1956 by Beckmann, McGuire and Winsten. Traffic is assigned using the origin-destination matrix using the concept of Network Equilibrium (so-called Wardrop's First Principle). The disadvantage of traffic simulation models is that their accuracy is hard to measure and the performance is never compared to the other models. These models can be further classified into macroscopic, microscopic and mesoscopic (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007). In macroscopic simulation models, only global variables of a road network are considered, such as the densities, mean speeds and flows. Macroscopic models are performed through either static or dynamic assignment. In a static model, fluctuations in departure times are not taken into account; and an equilibrium is found using the entire origin-destination flows. On the contrary, dynamic models consider the variations in demand (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007). Microsimulation models simulate the individual vehicles through the network as well as their interaction. One frequently used method is Cellular Automata, in which a road is represented as an array of cells. These models can use either O-D matrices or turn fractions for predictions. A mesoscopic model is a mixture of macroscopic and microscopic models (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

Time series models predict a variable as a function of its past observations and an error. To be able to apply the time series models, the data should satisfy the conditions of stationarity. In comparison to the traffic simulation models, instead of the traffic theory, time series models use statistical functions. Usually, these models have high accuracy and low computational effort making them a promising solution (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

In **linear regression**, the prediction function is considered to be a linear combination of its covariates, in which parameters indicate how much one covariate contributes to the outcome (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007). Due to its simplicity, the model is not computationally expensive and in some cases, can produce good results.

ARIMA or Box-Jenkins model is a standard statistical technique that can be used for

prediction. The opinion of researchers on the performance of these models varies from negative to positive. To improve the performance of such models, different variations of Arima are suggested such as Kohonen ARIMA, SARIMA, VARMA and STARMA, ARIMAX, and Exponential Smoothing. Some of these models report high prediction accuracy (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

Another type, a **Kalman filter**, estimates the future state from the determined state in the previous time step and the current measurement. The results are fluctuating (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007).

3. Naive methods

To sum up, the final set of methods are **naive methods**. Naive methods assume the absence of any model assumption. These techniques are widely used in practice due to their simplicity and low computational effort. These methods might be **instantaneous, historical averages and clustering** (van Hinsbergen, van Lint, Van Zuylen, Sanders, 2007). We will not consider these because of their low accuracy.

It is evident that considering all of the available models for forecasting is impossible. Moreover, the model that is typically used in deep learning is a neural network. That is why it has been decided to pay closer attention to four different variations of neural networks. According to the overview, these model seems to be the most appropriate for short-term traffic forecasting in terms of accuracy and popularity.

2.2 LSTM and convolutional architecture to capture spatial-temporal component

Similarly to spatial-temporal features of video and audio, traffic flow data have a numerous characteristics in both time and space aspects. For instance, in the space domain, traffic flow patterns in some location are more likely to have stronger dependencies in nearby locations. In the time domain, the traffic flow in the past has a long-term impact on current traffic (Yuankai, Huachun, 2016).

Yuankai and Huachun propose a deep neural architecture to combine CNN and LSTM to capture both time and space domains of the traffic flow. An 1-dimension CNN is used to capture spatial characteristics of traffic flow, and two neural networks based on LSTM are applied to determine the short-term variability and periodicities of traffic flow. To be exact, a deep convolution neural network is used to determine the space features of traffic flow data. After this, LSTMs are applied to acquire features of both short-term time displacement and long-term periodicity. Then, these spatial-temporal traits are fed into a linear regression layer to make future predictions. (Yuankai, Huachun, 2016).

The network structure that Yuankai and Huachun use is presented in Figure 2.1 (Yuankai, Huachun, 2016).

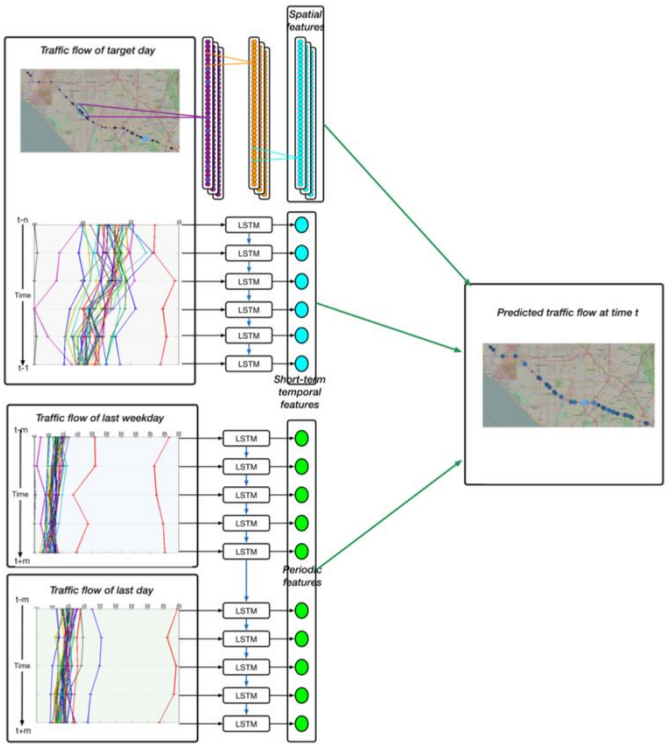


Figure 2.1: Neural network consisting of a 1D CNN (capture spatial features), two LSTM RNNs (capture short-term and periodic features) and followed by a fully connected layer to fuse all features to forecast traffic flow at target time point t, Yankai, Huachun, 2016

The network tries to forecast traffic flow in p locations $\{s_i\}_{i=1}^{p}$ in (t,t+1,⋯ ,t+h), where h is the

prediction horizon. The historical traffic flow data in p locations $\{s_i\}_{i=1}^{p}$ in (t − n, t − 1) is used as inputs for generating prediction in (t, t + 1, · · · , t + h). The historical data is put together to get a data matrix (Yuankai, Huachun, 2016).

$$\mathbf{S} = \begin{bmatrix} S_1 \\ S_2 \\ . \\ . \\ . \\ S_p \end{bmatrix} = \begin{bmatrix} s_1(t-n) & s_1(t-n+1) & \cdots & s_1(t-1) \\ s_2(t-n) & s_2(t-n+1) & \cdots & s_2(t-1) \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ s_p(t-n) & s_p(t-n+1) & \cdots & s_p(t-1) \end{bmatrix}.$$

Conventional forecasting models have two main drawbacks in capturing the temporal pattern of traffic flow. First of all, traditional RNNs are challenging to train if the traffic flow series has long time lags, which means the diminishing performance if n is too large. Second of all, it is hard to find the optimal time window of size n, as the correlation between different points of time-series data is affected by several complex factors such as weather, speed or an unpredictable accident. The LSTM aims to overcome these issues and captures the time-series pattern of traffic flow. The LSTM is used to generate time features at each point to build a sophisticated traffic flow forecasting model (Yuankai, Huachun, 2016).

Similar to the traditional RNNs, an LSTM structure is composed of one input layer, one or several hidden layers and one output layer. The distinctive feature of LSTM is a memory cell designed to overcome the gradient vanishing problem of conventional RNNs. A memory cell contains four main parts: an input gate, an output gate, a neuron with a self-recurrent connection and a forget gate (Figure 2.2) (Yuankai, Huachun, 2016).

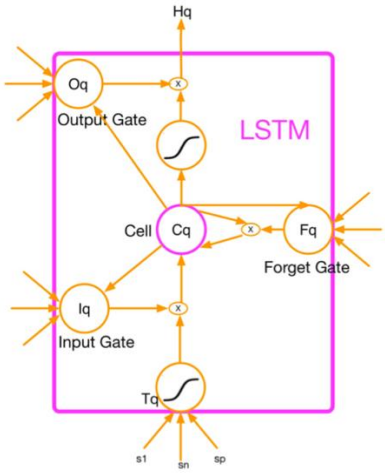


Figure 2.2: LSTM structure, Yankai, Huachun, 2016

The experimental part is conducted in the following manner: spatial features, short-term temporal features and periodic features are generated based on the combination of CNN and LSTM to train different Lasso models. Then, there is a comparison between the results of Lasso models with different variations of features (Yuankai, Huachun, 2016).

The prediction method proposed by Yuankai and Huachun seems to be a promising choice. However, unlike in the conducted research, the data given in the case study does not include the spatial component and adheres only the temporal pattern. This leads to the conclusion that only the LSTM part of the proposed architecture can be used to provide short-term traffic forecasting effectively.

2.3 LSTM and convolutional architecture to capture short – term and long – term dependencies

Another research conducted by Lai and Yang follows a similar approach. The authors argue in some real-world applications, the data consists of long-term and short-term patterns, for which traditional forecasting methods are ineffective. To overcome this issue, they propose a unique deep learning framework called Long- and Short-term Time-series network (LSTNet). It includes a convolutional component that instead of capturing the spatial pattern deals with the long-term dependencies of time-series data (Lai, Yang, 2018).

Multivariate time series forecasting frequently tries to find out how to obtain and leverage the dependencies among various variables. Mainly, some applications often require a fusion of short-term and long-term repeated patterns. In particular, the study attempts to determine the hourly occupancy rate of a freeway. The time-series data has two repeating patterns, daily and weekly. The former pattern represents the morning and evening peaks, while the latter reveals the workday and weekend patterns. The authors claim that an appropriate model should be able to capture both patterns to make accurate predictions. They argue that traditional approaches such as autoregressive methods cannot capture both patterns at the same time. The main goal of their research is to overcome these limitations by trying to utilize contemporary deep-learning methods (Lai, Yang, 2018).

The purpose of proposed LSTNet architecture is to leverage both the convolutional layer to explore the local dependency patterns among multi-dimensional input variables and the recurrent layer to obtain multiple long-term dependencies (Lai, Yang, 2018).

In their paper, the authors perform the time-series forecasting. In particular, given a series of time-series values $Y = \{y_1, y_2, \ldots, y_T\}$ where $y_T \in R^n$, and n corresponds to the variable dimension, they try to predict a series of future values. To predict $y_{T+h}$where h is the desirable horizon ahead of the current time step, they assume $Y = \{y_1, y_2, \ldots, y_T\}$ are prepared. Similarly, to predict the value at next time stamp y_{T+h+1}, they assume $\{y_1, y_2, \ldots, y_T, y_{T+1}\}$ are available. Then, they formulate the input matrix at time step T as X_T=$\{y_1, y_2, \ldots, y_T\} \in R^{n \times T}$(Lai, Yang, 2018).

At the fundamental level, the model includes four components: convolutional, recurrent, recurrent-skip layer, and fully-connected layer (Lai, Yang, 2018).

The first layer of LSTNet is a convolutional network without pooling, which aims to extract short-term patterns in the time dimension and the dependencies between variables. The convolutional layer consists of various filters of width ω and height n (the height corresponds to the number of variables). The output of the convolutional layer is fed into the recurrent component and recurrent-skip component. The recurrent part is a recurrent layer with the Gated Recurrent Unit (GRU) using the RELU activation function. The recurrent layers with GRU and

LSTM units are created to learn the historical information to capture the long-term dependencies. Because of the gradient vanishing problem, GRU and LSTM are not able to capture the long-term correlations. The authors suggest mitigating this issue using a recurrent-skip component which handles the recurrent pattern. As an example, traffic shows a clear pattern daily. To be able to capture the seasonality, a recurrent structure with temporal skip-connections is developed to extend the time span and to ease the optimization process. Finally, due to the non-linear nature of the convolutional and recurrent networks, the disadvantage of the neural network is that the scale of outputs is not sensible to the scale of inputs. The scale of input signals continually changes in a non-periodic manner, which diminishes the accuracy of the neural network models. To avoid this issue, the authors suggest decomposing the final prediction layer into a linear structure which alleviates the scaling issue. In the LSTNet architecture, they adopt the conventional Autoregressive (AR) model as the linear element (Lai, Yang, 2018). The structure if the proposed model is presented in figure 2.3 (Lai, Yang, 2018).

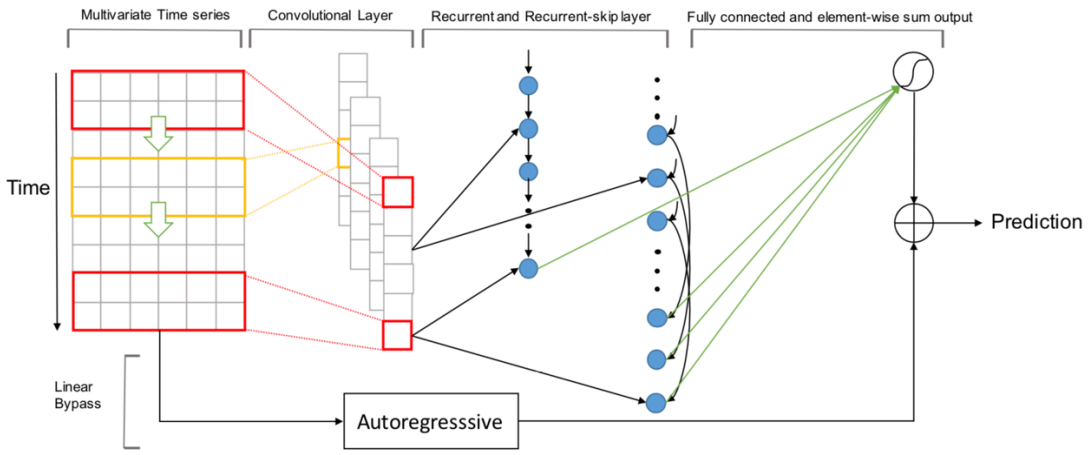


Figure 2.3: Overview of the Long- and Short-term Time-series network (LSTNet), Lai, Yang, 2018

The approach suggested by Lai and Yang seems to be a quite reasonable solution, except that the data provided in the case study has an only short-term pattern repeating daily. To leverage this pattern, we might try to use the different variations of the recurrent architecture proposed by the authors. As suggested by the authors, to alleviate the scaling issue, we might use a linear structure in the final layer, such as a dense layer.

2.4 Recurrent neural networks with LSTM, GRU cells for short-term traffic prediction

In 2017 paper Azzouni and Pujolle propose using an LSTM RNN framework for predicting the traffic matrix in networks. They argue that the decisions made by the traffic operators depend on the traffic flow conditions. However, contemporary network devices cannot handle real-time monitoring; thus, network operators cannot react adequately to the alterations in the traffic state. Having an actual and timely traffic matrix is necessary for traffic accounting, short-time traffic scheduling, network design, long-term capacity planning, and network anomaly detection (Azzouni, Pujolle, 2017).

The authors claim that conventional filly-connected neural networks provide only limited temporal forecasting operating on a fixed-size window of traffic matrix sequence. Additionally, they are not suitable to handle historical dependencies. By contrast, recurrent neural networks or deep recurrent neural networks constitute cycles that feed the activations of a recurrent layer from a previous time step as inputs to the next time step (Azzouni, Pujolle, 2017).

However, training conventional recurrent neural networks with the gradient-based back-propagation through time (BPTT) is insufficient due to the vanishing gradient and exploding gradient problems. As a result, the given input on the hidden layers either decays or blows up exponentially when moving through the network recurrent connections. This problem makes it difficult to train the sequences that are 5 to 10 discrete time steps long (Azzouni, Pujolle, 2017).

The problem statement is as follows. The authors denote N to be the number of nodes in the network. Then N-by-N traffic matrix denoted by Y is created, where each entry y_{ij} represents the traffic volume from node i to node j. The third time dimension is added to the matrix to constitute N-by-N-by-T tensor S such as an entry s_{ij}^{t} corresponds to the traffic volume from node i to node j at time step t, and T is the total number of time slots. To predict the traffic matrix Y^t (denoted by $\widehat{Y_t}$) via a series of historical data sets ($Y^{t-1}, Y^{t-2}, Y^{t-3}, \dots, Y^{t-T}$) (Azzouni, Pujolle, 2017).

The data is fed into the LSTM in network the following way. The matrix Y is transformed into a traffic vector X^t of size $(N \times N)$ by concatenating N rows from top to bottom. x_n entry

corresponds to the original y_{ij} in the following manner $n = i \times N + j$. Now we predict the traffic vector X^t (denoted by $\widehat{X^t}$) via a series of historical traffic vectors $(X^{t-1}, X^{t-2}, X^{t-3}, \dots, X^{t-T})$ (Azzouni, Pujolle, 2017).

Real-time prediction of traffic matrix requires constant feeding and learning. Over time, the total number of time-slots become too big, resulting in high computational complexity. To cope with this problem, the authors introduce the notion of learning window (indicated by W) which shows a fixed number of previous time-slots to predict the traffic vector Xt at the current time step t (figure 2.4) (Azzouni, Pujolle, 2017).

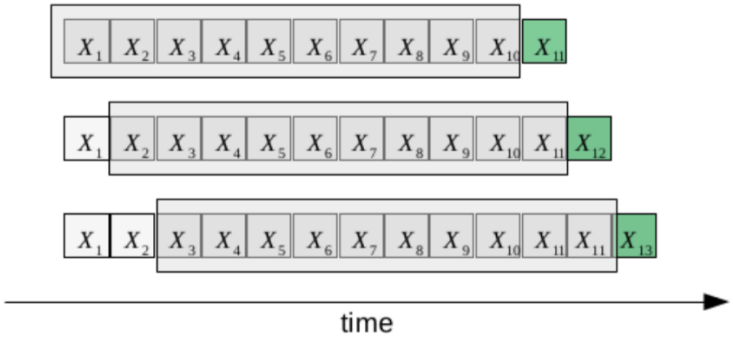


Figure 2.4: Sliding learning window, Azzouni, Pujolle, 2017

To assess the performance of the model, the Mean Squared Cost function is used to estimate the prediction accuracy. MSE is denoted as $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, where y_i corresponds to an observed value, $\hat{y}_i$ is a preicted values and N is the number of observations (Azzouni, Pujolle, 2017).

The results of the prediction of the traffic matrix using recurrent neural networks are compared against the traditional linear models such as ARMA, ARAR and HoltWinters algorithm. The results confirm that NNs outperform conventional linear models which cannot meet the accuracy requirements (Azzouni, Pujolle, 2017).

The framework proposed by Azzouni and Pujolle seems to be a promising solution. However, the traffic tensor S should be adapted in accordance with the traffic flow on links instead of the

networks. Additionally, the problem of predicting traffic flow is a regression problem making the usage of Mean Squared Error a feasible solution. Moreover, the input sequences used in the case study are about 50 instances long, which might lead to the vanishing/exploding gradient problem. Hence, the comparison of LSTM and conventional cells is necessary. As the authors suggest, to reduce the computational complexity, the usage of the learning window is preferable. In the case study, we may use a similar approach, predicting a certain number of future instances using the last learning sequence. Then, the process is repeated for a new sequence in real-time.

In another study conducted by Fu and Zhang, the authors are trying to determine which model provides the best performance - LSTM or GRU. Besides, the comparison with Arima is given. The methodological approach is quite similar to the previous study. In their research, the authors select the traffic flow over the past 30 minutes, which is a time sequence of 6 data points, to predict the traffic flow in the next 5 minutes (Fu, Zhang, 2016). The thirst three weeks of the traffic data are allocated to the training set and the last week is assigned to the validation set. The accuracy is evaluated using MSE and MAE. Experimental results demonstrate that LSTM and GRU networks have better performance than ARIMA model and GRU have a little better performance than LSTM. Meanwhile, in 84 % of the cases, GRU networks managed to demonstrate a slightly better performance than LSTM (Fu, Zhang, 2016).

The results obtained by Fu and Zhang suggest that it is reasonable to use GRU recurrent networks to provide a comparison with LSTM and ordinary RNN cells.

Chapter 3. Methodology

3.1 Recurrent neural networks

Recurrent neural networks are a specific type of neural networks that deal with sequential data. Such data pattern envisages the observations according to the chronological order of time. There are various examples of sequential data among which there are speech and language, stock prices, audio waves and many more. Likewise, traffic flow falls within the definition of sequential data being time-series data that includes temporal component. A distinctive characteristic that determines the RNNs is that their cells possess the memory state making them very useful for prediction of the time-dependent targets. Therefore, it means that this type of networks is likely to be valuable in traffic forecasting.

3.1.1 Mathematical representation of recurrent neural networks

Conventional artificial neural networks assume the flow of activations from the input layer to the output layer in one direction. A recurrent neural network is quite similar to conventional neural networks with the exception that the activations also point backwards. Both input and output of an RNN are typically sequences, although they also can be vectors. An input sequence can be denoted as $(x^{(1)}, x^{(2)}, \cdots, x^{(T)})$ where each element is a vector. Then, a target sequence can be denoted as $(y^{(1)}, y^{(2)}, \cdots, y^{(T)})$. A training set consists of several (input sequence, target sequence) pairs, although both inputs and targets may be single data points (Lipton, Berkowitz, 2015). In the case of traffic flow prediction, we might have the so-called sequence-to-vector or sequence-to-sequence models. A sequence-to-vector model implies that while unrolling the recurrent neural network through time, the output is returned only at the last time step. Similarly, a vector-to-vector model assumes that the output is returned at every given time step. The unrolling process is illustrated in figure 3.1 (Geron, 2019).

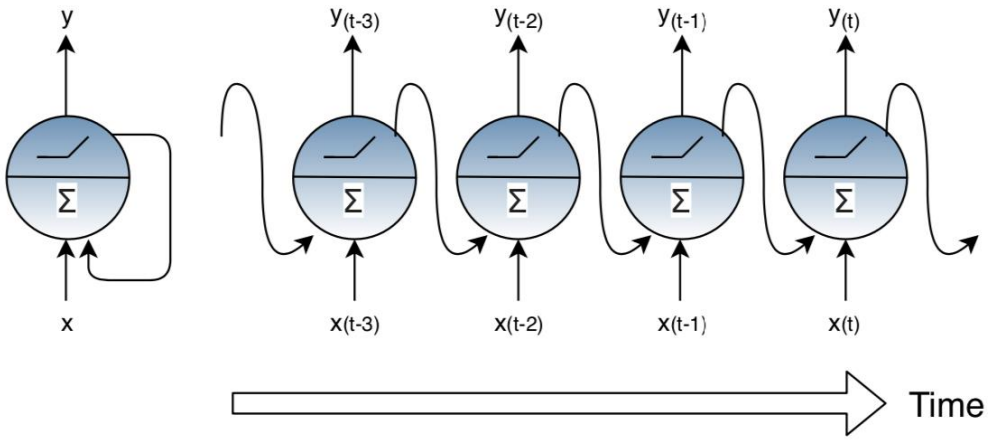


Figure 3.1: A recurrent neuron (left), unrolled through time (right), Geron, 2019

Furthermore, it is conceivable to combine several recurrent cells in a single layer, each of which receives the same input vector x_t together with the output vector $y_{t-1}$from the previous timestep. A representation of a layer consisting of several cells is presented in figure 3.2 (Geron, 2019).

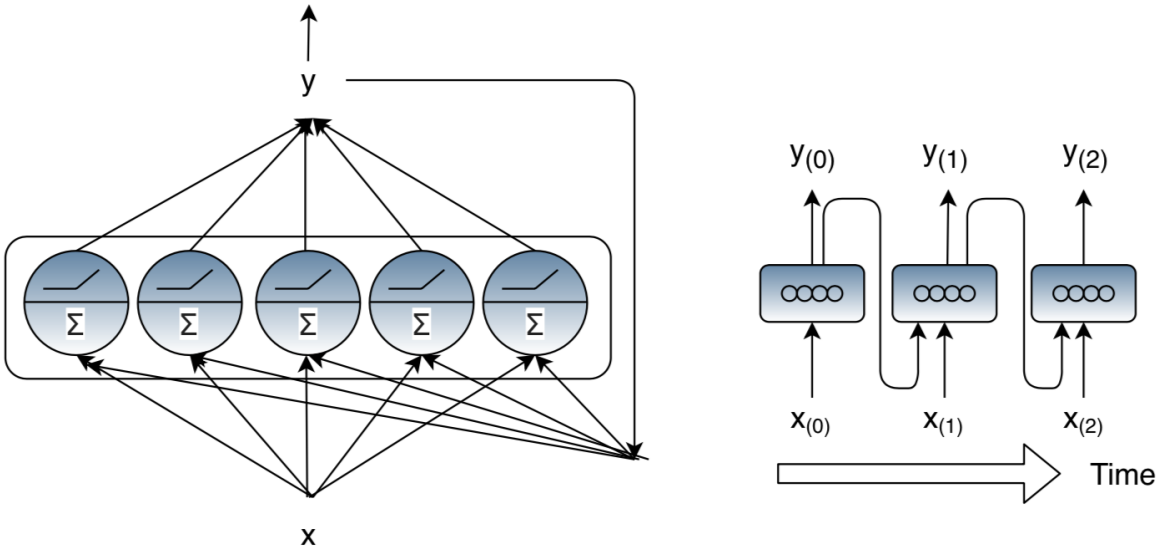


Figure 3.2 A layer of recurrent neurons (left), unrolled through time (right), Geron, 2019

Each recurrent neuron shares two types of weight: one associated with an input x_t and another with the output from the previous time step y_{t-1}. The output at the next time step is obtained by transforming the weighted sum of the input at the current time step and the output from the previous time step plus bias term using the activation function ϕ (which is usually rectified linear activation unit RELU) (Geron, 2019).

$$y_t = \phi(x_{(t)}^T \cdot w_x + y_{(t-1)}^T \cdot w_y + b)$$

When we take into consideration all instances in a mini-batch, the output at the current time step holds the vectorized form (Geron, 2019).

$$Y_t = \phi\left(X_t \cdot W_x + Y_{(t-1)} \cdot W_y + b\right) = \phi\left(\left[X_t \;\; Y_{(t-1)}\right] \cdot W + b\right) \text{ where } W = \begin{bmatrix} W_x \\ W_y \end{bmatrix}$$

- Y_t is a matrix of shape $m \times n_{neurons}$ containing the layer's output at time step t for each instance in a batch (where m is the number of instances)
- X_t is a matrix of shape $m \times n_{inputs}$ containing the inputs for all instances (n_{inputs} is a number of input features, 1 for univariate forecasting and more if we have several traffic parameters to predict)
- W_x is a matrix of shape $n_{inputs} \times n_{neurons}$ containing weights for the inputs at time step t
- W_y is a matrix of shape $n_{neurons} \times n_{neurons}$ containing weights for the outputs at the previous time step t-1
- b is a vector of shape $n_{neurons}$ that contains the bias terms for a recurrent layer
- $\begin{bmatrix} W_x \\ W_y \end{bmatrix}$ is a vertical concatenation of two weight matrices into a single weight matrix W of shape $(n_{inputs} \times n_{neurons}) \times n_{neurons}$
- $\left[X_t \;\; Y_{(t-1)}\right]$ is a concatenation of the input at current time step and the output at the previous time step in a horizontal direction

Based on the mathematical representation above, we can conclude that each recurrent cell has a form of memory since every output Y_t is a function of all the inputs $(X_{(0)}, X_{(1),\ldots}X_{(2)})$ starting

from t=0. This makes recurrent neural networks efficient in terms of ability to provide the output based on the previously processed elements of a sequence which is beneficial in time-series traffic forecasting (Geron, 2019).

3.1.2 Training recurrent neural networks

Training a neural network means finding the parameters of a model that minimize the objective function. To train the recurrent neural network, one has to unroll it through time, receive the output, calculate the error using the objective function, then propagate the gradients of the cost function backwards through the unrolled network using the chain rule of derivatives and update the weights. The process of training can be illustrated in figure 3.3 (Geron, 2019):

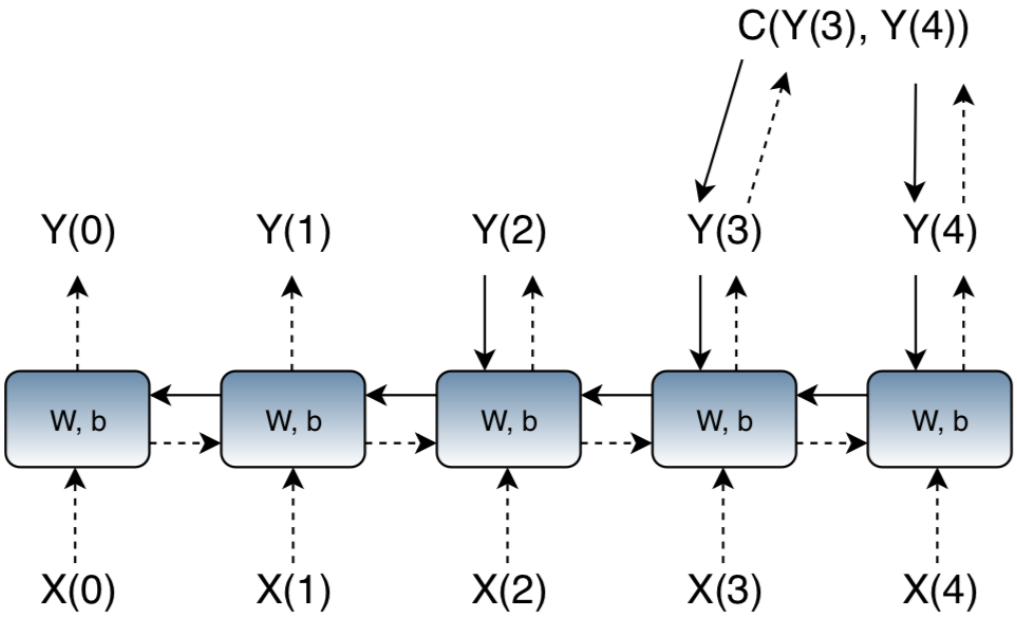


Figure 3.3: Backpropagation through time, Geron, 2019

It is essential to keep in mind that when we propagate backwards, we only propagate through the outputs used by the cost function. In a visual representation above, since we have a sequence-to-sequence model and the objective function accepts only the outputs at the last two-time steps, we propagate backwards through only these last outputs (Geron, 2019).

3.1.3 Recurrent deep neural networks

To be able to capture the nonlinearity of traffic flow, it is a common practice to organize multiple recurrent layers into a deep network. At each time step, a layer's output is passed to the next time step as well as to the next layer at the current time step. Deep recurrent neural networks come in different structures and shapes, and might have various types of cells. The simplest structure of a deep recurrent neural network with simple RNN cells is presented in figure 3.4 (Geron, 2019):

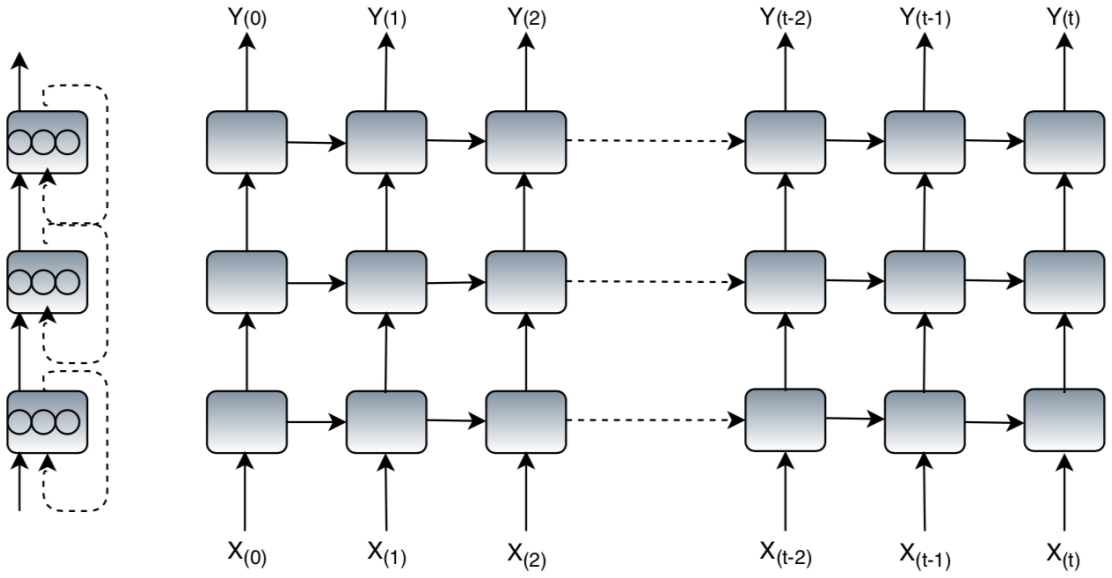


Figure 3.4: Deep RNN (left), unrolled through time (right), Geron, 2019

3.1.4 Adaptive moment optimization (Adam)

In deep learning, the purpose of training optimizers is to adjust the parameters of a model to achieve the optimal solution that leads to the minimization of an objective function. A neural network might have an immense number of parameters resulting in a non-convex nature of a cost function and making it hard for the conventional optimization algorithms such as gradient descent to achieve an optimal solution. To overcome this issue, several optimization algorithms

to train neural networks have been proposed among which the most popular is adaptive model estimation or Adam. The same optimizer is used in the case study section of this thesis.

To better understand the concept behind Adam, we need to address several other algorithms from which Adam is derived. Essentially, adaptive model estimation combines the ideas behind the so-called Momentum optimization and RMSProp (Geron, 2019).

The main problem behind numerous variations of Gradient Descent is that all of them take the same step down the "slope" regardless of the steepness of that slope. Conventional Gradient descent algorithm cannot navigate ravines or in other words, the areas where the surface of a cost function is more steeply in one dimension than in another. In such a scenario, this algorithm oscillates across the slopes while making hesitant progress along the bottom towards the local minima. At each iteration, the momentum optimization subtracts the local gradient from the momentum vector m, multiplies it by the learning rate and updates the weights by adding this momentum vector. To prevent momentum from growing too large, a momentum term is added (Ruder, 2017; Geron, 2019).

$$m \leftarrow \beta m + \eta \nabla_\theta J(\theta)$$

$$\theta \leftarrow \theta - m$$

Another variation of Momentum optimization is Nesterov Accelerated Gradient whose only difference is that it measures the local gradient of the cost function not at the local position but slightly ahead making the estimation slightly more accurate (Ruder, 2017; Geron, 2019).

$$m \leftarrow \beta m + \eta \nabla_\theta J(\theta + \beta m)$$

$$\theta \leftarrow \theta - m$$

Another modification of conventional Gradient Descent is AdaGrad. This algorithm makes it possible to detect the steepest slope of the cost function beforehand and correct its trajectory on the way. It does so by scaling the gradient vector along the steepest dimensions. First, it collects the square of the gradients into vector s. Second, it scales the gradient vector down by a factor of $\sqrt{s + \epsilon}$ (Ruder, 2017; Geron, 2019).

$$s \leftarrow s + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s + \epsilon}$$

Finally, the disadvantage of AdaGrad is that it might slow down next to a minimum and never converge. RMSProp fixes this by accumulating the gradients from the most recent iterations. It is achieved by using exponential decay (Ruder, 2017; Geron, 2019).

$$s \leftarrow \beta s + (1 - \beta) \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \oslash \sqrt{s + \epsilon}$$

These days, the most popular optimization algorithm that has been proven to be most efficient to train the neural networks is Adam. It consolidates the keys concepts behind momentum optimization and RMSProp. Similar to Momentum optimization, it keeps track of exponentially decaying average of past gradients. Moreover, identical to RMSProp, it keeps track of exponentially decaying averages of past squared gradients. The mathematical representation of Adam algorithm can be expressed using the following formulas (Ruder, 2017; Geron, 2019):

$$m \leftarrow \beta_1 m + (1 - \beta_1) \nabla_\theta J(\theta)$$

$$s = \beta_2 s + (1 - \beta_2) \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$m = \frac{m}{1 - b_1^T}$$

$$s = \frac{s}{1 - b_2^T}$$

$$\theta = \theta - \eta m \oslash \sqrt{\hat{s} + \epsilon}$$

The great advantage of Adam has to do with its adaptive nature making it possible to avoid tuning a learning rate too much (Ruder, 2017). It is feasible to use this algorithm with a default value of a learning rate equal to 0,001.

3.1.5 Data reshaping and training process

When it comes to training a neural network, data reshaping is a crucial process. Usually, after data cleansing, preparation and aggregation are performed, we get a table the first column of which contains an array with target values while the remaining columns hold the arrays of values corresponding to the input features (also called independent variables). When it comes to processing time-series sequences, the rule is somewhat different. Instead of having a separate column containing an array with target values, the targets are obtained from the columns corresponding to the input features themselves. To achieve this, a one-dimensional array with input feature values has to be reshaped into a three-dimensional tensor. Any tensor is defined as an n-dimensional matrices. For instance, a zero-dimensional tensor is a scalar, a one-dimensional tensor is a vector, and a two-dimensional tensor is a matrix. A tensor holding the traffic data is three-dimensional and has the shape [batch_size, n_steps, diminsion], where the first dimension corresponds to a number of training batches, the second dimension - to a sequence length, and third - to the number of features. Visual representation of a tensor containing the reshaped traffic data is presented in figure 3.5:

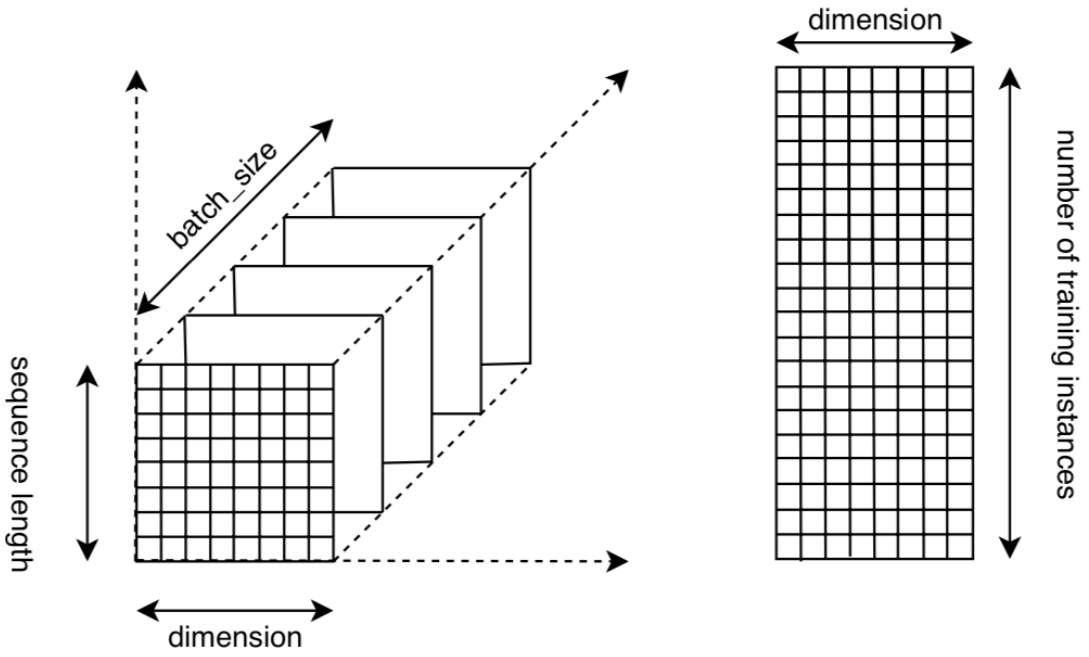


Figure 3.5: Visual representation of reshaped traffic data

After a tensor containing the traffic data is created, we need to split it into two separate tensors, one of which contains features and another - targets. To achieve this, we have to divide the second dimension into two sequences. The first sequence contains the number of training instances we want to predict (it might also be a single instance if the goal is to predict a single value) counted from the bottom of the initial sequence, and the second sequence corresponds to the remaining top instances. Then, the first sequence is assigned to the tensor with targets, and the second sequence is assigned to the tensor with features. As a result, the shape of a tensor with features is [batch_size, n_steps, dimension]. Tensor with targets has a shape [batch_size, forecast, dimension]. The process of splitting the initial tensor into two tensors containing features and targets is illustrated in figure 3.6:

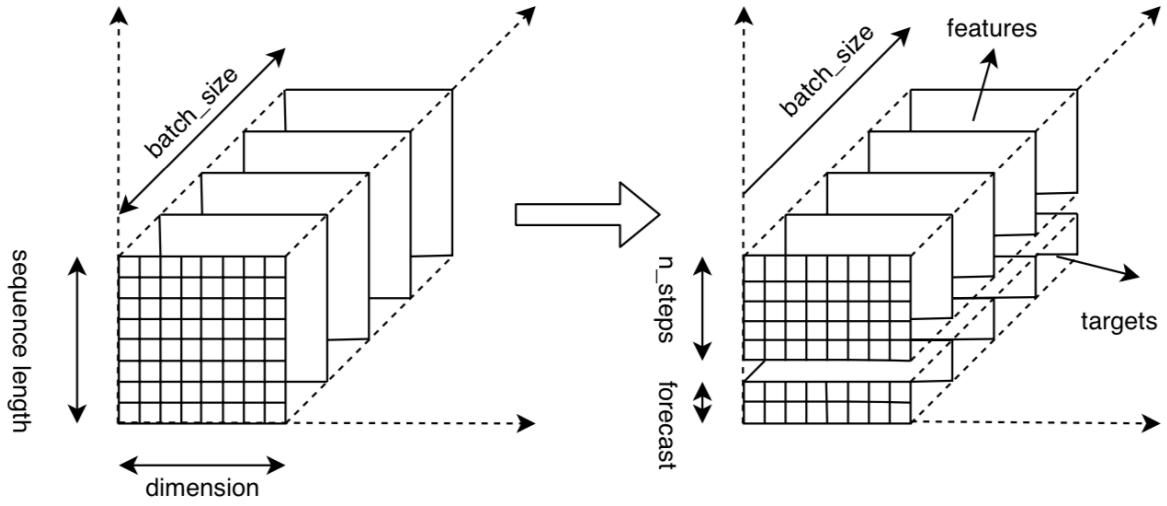


Figure 3.6: Splitting traffic tensor into features and targets

Sequence-to-vector model envisages the output of a recurrent layer at the last time step. When a sequence-to-sequence model is used, the output is returned at every time step. This means that the backpropagation algorithm will use the gradients flowing through the outputs speeding up and stabilizing the training process (Geron, 2019). Considering the aforementioned, in case of the univariate time-series forecasting the shape of the targets is [batch_size, n_steps, forecast]. Nevertheless, only the last element of the second dimension is used for forecasting.

At each iteration, several training batches (equal to the batch size) containing the traffic matrix are fed into a deep learning algorithm. After several matrix operations, the neural network outputs the tensors of a shape identical to a shape of a target. Using the appropriate cost function, the error is calculated and propagated backwards to compute the gradient. Then, Adam algorithm uses the found vector to update the parameters of a model. For each iteration, this process is repeated until every batch is involved in the calculation of a gradient vector.

3.1.6 Training algorithm for univariate forecasting

As it has been observed in the previous chapters, at each time step, a recurrent layer returns a tensor of shape $m \times n_{neurons}$. Sequence-to-vector model envisages that only the output at the last time step is returned. This makes it possible to feed the output of the last recurrent layer to a dense layer to make a prediction ten-time steps in the future. In case of a sequence-to-sequence model, we need to wrap a dense layer in a timedistributed layer to be able to compute the error at every timestep. The training process for sequence-to-vector model is described as follows:

Procedure: predict a sequence of traffic data 50 minutes ahead (10 observations of data aggregated in 5 minutes)

Input data: a tensor with input features of shape **[batch_size, n_steps, dimension]**, a tensor with **targets [batch_size, forecast, 1],** number of epochs **T**, learning rate $\boldsymbol{\eta}$ equal to 0.001, **Adam** optimizer, momentum vector **m**, vector **s** containing square of the gradients, hyperparameter $\boldsymbol{\beta_1}$ equal to 0.9, hyperparameter $\boldsymbol{\beta_2}$ equal to 0.999, smoothing term $\boldsymbol{\epsilon}$ equal to 10^{-8}, hyperparameter **batch** (number of batches used to compute the gradient) is equal to 32, cost function – **MSE,** error **E,** activation function $\boldsymbol{\phi}$ – tanh, **n_neurons**, **n_layers**.

Result: an output $\boldsymbol{target}$ of shape **[1, forecast]**, where forecast is the number of traffic instances to be forecasted into the future.

Algorithm:

Randomly initialize $\theta = \begin{bmatrix} W_x \\ W_y \end{bmatrix}$ using normal distribution, assign zeros to vector b, assign zeros to m ans s, initialize E with 0.

For 1 to T:

For 1 to batch_size/batch:

For 1 in batch:

Recurrent layer:

$Y_t = \phi\left(X_t \cdot W_x + Y_{(t-1)} \cdot W_y + b\right)$

$X_t \cdot W_x$ = (n_steps x dim) x (dim x n_neurons) = (n_steps x n_neurons)

$Y_{(t-1)} \cdot W_y$ = (n_steps x n_neurons) x (n_neurons x n_neurons) = (n_steps x n_neurons)

$Y_t = \phi\left(X_t \cdot W_x + Y_{(t-1)} \cdot W_y + b\right)$ = (n_steps x n_neurons)

Output at last step = (1 x n_neurons)

Dense layer:

batch_target = (1 x n_neurons) x (n_neurons x forecast) = (1 x forecast)

compute E = $\frac{1}{forecast}\sum_{i=1}^{forecast}(target_i - batch\_target_i)^2$

Compute $\nabla_\theta J(\theta)$ by propagating E using backpropagation through time

Compute:

$m \leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta J(\theta)$

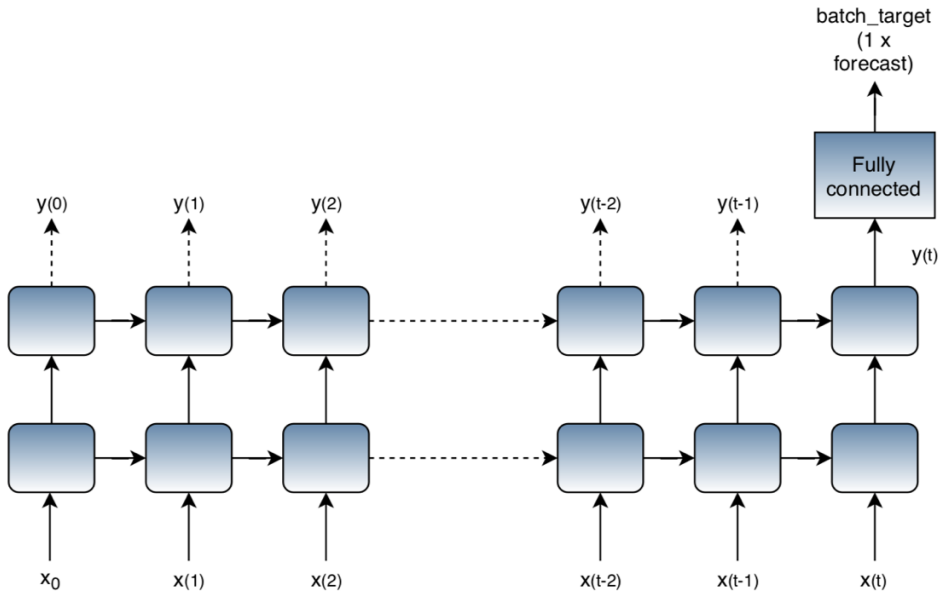
$s \leftarrow \beta_2 s + (1-\beta_2)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$

$m = \frac{m}{1-b_1^T}$

$s = \frac{s}{1-b_2^T}$

$\theta \leftarrow \theta - \eta m \oslash \sqrt{s+\epsilon}$

The structure of a network corresponding to a sequence-to-vector traffic model in case of two recurrent layers is illustrated in figure 3.7:



Figire 3.7: Sequence-to-vector traffic model

3.2 Recurrent neural networks with LSTM and GRU cells

Recurrent neural networks have been proven to be very successful at sequence forecasting. Backpropagation through time is the algorithm used for training the RNNs; however, it has a significant limitation. The temporal evolution of the path integral over all error signals depends on the magnitude of the weights exponentially. This implies that the error flowing backwards might completely vanish or blow up (Gers, Schmidhuber, Cummins, 1999). That results in the elimination of the first inputs of a sequence. Many methods exist to fight the so-called vanishing/exploding gradient problem, among which one of the most popular is LSTM (Long Short-Term Memory) cells. Another variation of LSTM that is becoming increasingly popular is GRU (The Gated Recurrent Unit). GRU is considered to be a simplified version of LSTM but performs as efficiently (Greff, Srivastava, 2017).

3.2.1 Mathematical representation of LSTM cells

A conventional LSTM cell consists of several elements: three gates (input, output, and forget), block input, a single cell, an output activation function, and peephole connections. The output of the block is connected to the block input and all of the gates. A visual representation of an LSTM cell is given in figure 3.8 (Greff, Srivastava, 2017):

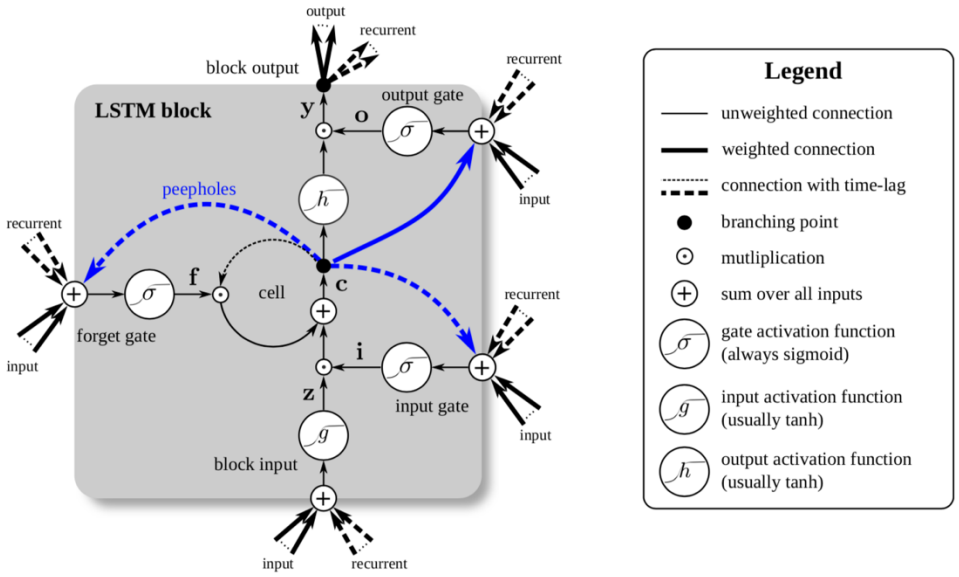


Figure 3.8: Long Short-Term memory block, Greff, Srivastava, 2017

If we denote x^t to be the input vector at time t, N be the number of LSTM blocks and M the number of inputs, then we get the following weights of an LSTM layer (Greff, Srivastava, 2017):

- Input weights: $W_z, W_i, W_f, W_o \in R^{N \times M}$
- Recurrent weights: $R_z, R_i, R_f, R_o \in R^{N \times M}$
- Peephole weights: $R_z, R_i, R_f, R_o \in R^N$
- Bias weight: $b_z, b_i, b_f, b_o \in R^N$

where input weights are the weight matrices for each of the four layers to connect to the input vector at time t x^t, recurrent weights are the weight matrices for each of the four layers to connect to the previous short term state y^{t-1}, bias weights are the bias terms for each of the four layers. The purpose of the peephole connections is to add the previous long-term state c^{t-1} to as an input to the controllers of the input and forget gates, while the current long-term state c^t is added as an input to the controller of the output gate (Greff, Srivastava, 2017).

Then, the vectorized formulas for an LSTM layer during the forward pass can be written in the following way (Greff, Srivastava, 2017):

$$\overline{z^t} = W_z x^t + R_z y^{t-1} + b_z$$

$$z^t = g(\overline{z^t})$$

$$\overline{i^t} = W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i$$

$$i^t = \sigma(\overline{i^t})$$

$$\overline{f^t} = W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f$$

$$f^t = \sigma(\overline{i^t})$$

$$c^t = z^t \odot i^t + c^{t-1} \odot f^t$$

$$\overline{o^t} = W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o$$

$$o^t = \sigma(\overline{o^t})$$

$$y^t = h(c^t) \odot o^t$$

where σ, g and h are the non-linear activation functions. The logistic sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ is used as gate activation function and the hyperbolic tangent $g(x) = h(x) = \tanh(x)$ is used as the block input and utput activation function. Pointwise multiplication of two vectors is $\odot$ (Greff, Srivastava, 2017).

To update the parameters of the model, the calculated error is propagated backwards using the backpropagation through time algorithm. The algorithm has the following mathematical representation (Greff, Srivastava, 2017):

$$\delta y^t = \Delta^t + R_z^T \delta z^{t+1} + R_i^T \delta i^{t+1} + R_f^T \delta i^{f+1} + R_0^T \delta o^{t+1}$$

$$\delta \overline{o^t} = \delta y^t \odot h(c^t) \odot \sigma'(\overline{o^t})$$

$$\delta c^t = \delta y^t \odot o^t \odot h'(c^t) + p_0 \odot \delta \overline{o^t} + p_i \odot \delta \overline{i^{t+1}} + p_f \odot \delta \overline{f^{t+1}} + \delta c^{t+1} + \delta f^{t+1}$$

$$\delta \overline{f^t} = \delta c^t \odot c^{t-1} \odot \sigma'(\overline{f^t})$$

$$\delta \overline{i^t} = \delta c^t \odot z^t \odot \sigma'(\overline{i^t})$$

$$\delta \overline{z^t} = \delta c^t \odot i^t \odot g'(\overline{z^t})$$

The deltas for the input are only necessary if there is a layer below (Greff, Srivastava, 2017):

$$\delta x^t = W_z^T \delta \overline{z^t} + W_i^T \delta \overline{i^t} + W_f^T \delta \overline{f^t} + W_o^T \delta \overline{o^t}$$

Then, the gradients of the weights are calculated as follows (Greff, Srivastava, 2017):

$$\delta W_\star = \sum_{t=0}^{T} < \delta \star^t, x^t >$$

$$\delta R_\star = \sum_{t=0}^{T-1} < \delta \star^{t+1}, y^t >$$

$$\delta b_\star = \sum_{t=0}^{T} \delta \star^t$$

$$\delta p_i = \sum_{t=0}^{T-1} c^t \odot \delta \overline{i^{t+1}}$$

$$\delta p_f = \sum_{t=0}^{T-1} c^t \odot \delta \overline{f^{t+1}}$$

$$\delta p_f = \sum_{t=0}^{T-1} c^t \odot \delta \overline{o^t}$$

where $\langle \star_1, \star_2 \rangle$ is the outer product of two vectors.

Conceptually speaking, LSTM cell can learn to recognize a certain input with the help of an input gate, store this input in the long-term state, learn to preserve it for as long as necessary with the help of the forget gate and extract it on demand. This ability of LSTM cells makes them suitable for capturing the long-term patterns in time-series data (Geron, 2019).

3.2.2 Mathematical representation of GRU cells

GRU was introduced by Cho et al. in 2014. It is similar to LSTM, but simpler in terms of implementation. The typical structure of GRU cells is given in Figure 3.9 (Cho, Merrienboer, Gulcehre, 2014):

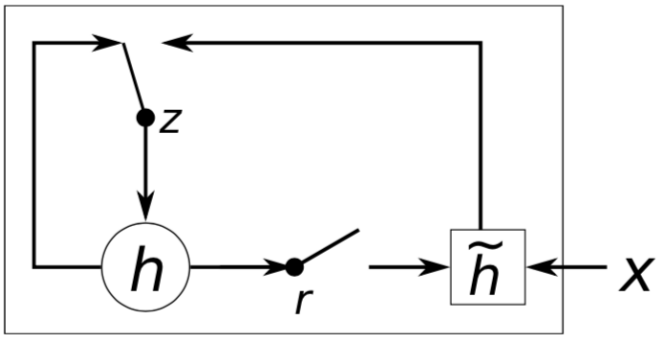


Figure 3.9: An illustration of the proposed hidden activation function. The update gate z selects whether the hidden state is to be updated with a new hidden state h̃. The reset gate r decides whether the previous hidden state is ignored, Merrienboer, Gulcehre, 2014

The principal simplifications are as follows (Geron, 2019):

- Both state vectors are joined into a separate vector.
- An individual gate controller manages both the forget gate and the input gate. If the input gate is open, and the forget gate is closed, it means that the gate controller outputs 1. On the contrary, if the opposite happens, the gate controller outputs 0. This means that, before allocating the memory, the location where it must be stored is erased first.
- A GRU cell has no output gate, and the state vector is output at every time step.

The following equations represent how to compute a state of a cell at every time step (Geron, 2019):

$$z_{(t)} = \sigma(W_{xz}^T \cdot x_{(t)} + W_{hz}^T \cdot h_{(t-1)})$$

$$r_{(t)} = \sigma(W_{xr}^T \cdot x_{(t)} + W_{hr}^T \cdot h_{(t-1)})$$

$$g_{(t)} = tanh(W_{xg}^T \cdot x_{(t)} + W_{hg}^T \cdot (r_{(t)} \otimes h_{(t-1)}))$$

$$h_{(t)} = (1 - z_{(t)}) \otimes tanh(W_{xg}^T \cdot h_{(t-1)} + z_{(t)} \otimes g_{(t)})$$

3.3 WaveNet neural network for sequence processing

In 2016 the DeepMind researchers introduced the neural architecture called WaveNet. Even though the initial intention of this architecture was to generate the raw audio waveforms, it is perfectly suitable for any one-dimensional sequence processing, including time-series traffic data. The idea behind the architecture is based on stacked one-dimensional convolutional layers with a doubled dilation rate at every layer. The network was proven to provide state-of-the-art performance at processing large sequences outperforming LSTMs and GRUs.

3.3.1 Representation of WaveNet architecture

In the research paper (van den Oord, Simonyan, Kalchbrenner, 2016), the authors propose a new generative model working directly on the raw audio waveform, which is a sequence of data points. However, the sequence can have any form, including one-dimensional traffic time-series traffic data.

The joint probability of a sequence $x = \{x_1, \dots, x_T\}$ is factorized as a product of conditional probabilities in the following way (van den Oord, Simonyan, Kalchbrenner, 2016):

$$p(x) = \prod_{t=1}^{T} p(x_t | x_1, \dots, x_{t-1})$$

Therefore, it indicates that each x_t is conditioned on the sample at the previous time steps. The model outputs a categorical distribution over the next value x_t using a softmax layer, and it is intended to maximize the log-likelihood of the data. Tuning the hyperparameters is performed

by comparing the log-likelihoods of different models (van den Oord, Simonyan, Kalchbrenner, 2016).

Instead of casual convolutions, WaveNet architecture uses the dilated convolutions. In a dilated convolution the filter is implemented over an area greater than its length by skipping input values with a particular step. Such structure is comparable to a convolution with a larger filter by dilating it with zeros. Dilated convolutions allow the network to perform on a larger scale compared to conventional convolutions (van den Oord, Simonyan, Kalchbrenner, 2016). Stacked dilated convolutions let networks to have vast receptive fields using a few layers while preserving the input resolution and resulting in computational efficiency. Convolutions with dilations 1,2,4 and 8 are represented in figure 3.10 (van den Oord, Simonyan, Kalchbrenner, 2016):

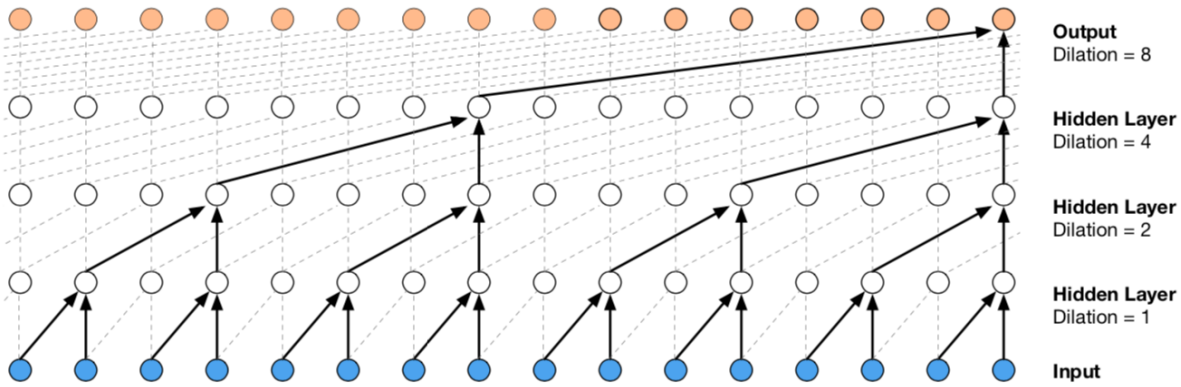


Figure 3.10: Visualization of a stack of dilated convolutional layers, van den Oord, Simonyan, Kalchbrenner, 2016

The network uses a gated activation unit (van den Oord, Simonyan, Kalchbrenner, 2016):

$$z = \tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x)$$

Where $*$ is a convolution operator, $\sigma(\cdot)$ denotes a sigmoid function $\odot$ is an element-wise multiplication operator, f and g are filter and gate, respectively, k is the index of a layer, and W is dentoes convolution filter (van den Oord, Simonyan, Kalchbrenner, 2016).

The network uses residual and parameterized skip connection to quicken convergence and allow training deeper models (figure 3.11) (van den Oord, Simonyan, Kalchbrenner, 2016).

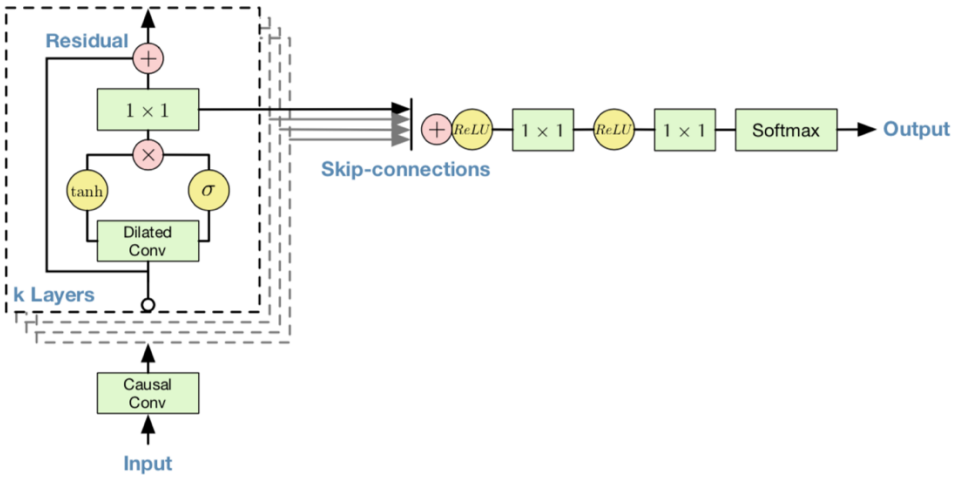


Figure 3.11: Overview of the residual block and entire architecture, van den Oord, Simonyan, Kalchbrenner, 2016

Chapter 4. Case study

4.1 Description of the study area

Data for this project was collected from the urban network in Nicosia, Cyprus. In total, 15 loop detectors with different characteristics were installed in the study area, among which only three were presented in the available data. Collection of disaggregated data was performed in 2015 over three months from September, 18 to November, 18. The total number of individual vehicle records among all 15 detectors is approximately 410 thousand a day (Dimitrou, Stylinou, 2018). The map of the area and the characteristics of the loop detectors are shown in figure 4.1 (Dimitrou, Stylinou, 2018) and table 4.1 (Dimitrou, Stylinou, 2018) respectively:

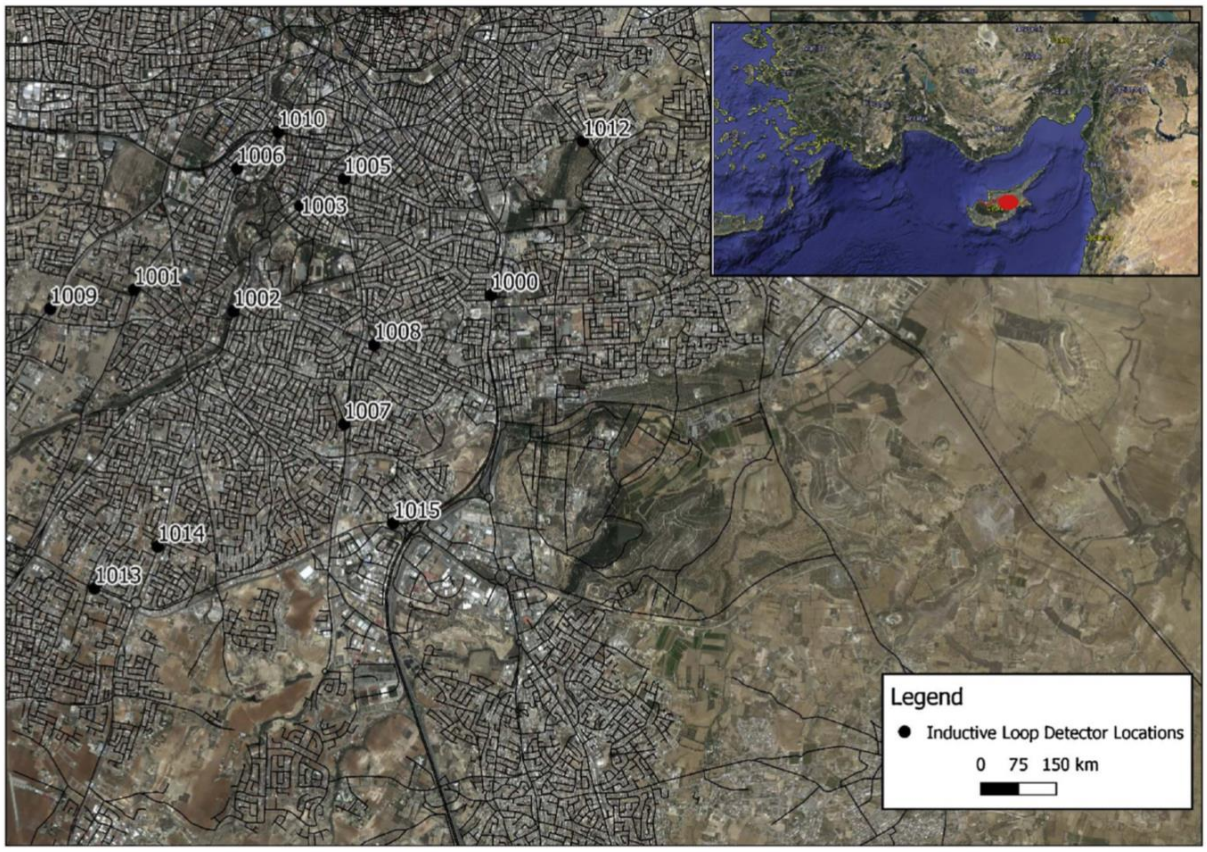


Figure 4.1: Inductive Loop Detecor Locations, Dimitrou, Stylinou, 2018

ILD Number	Carriageway Design	Number of Lanes covered by ILD	Lane Width (m)	Distance from upstream major Intersection (m)		Distance from downstream major Intersection (m)		Distance from nearest upstream minor Intersection (m)		Distance from nearest downstream minor Intersection (m)	
1000	Dual (a)	4	3.50	S	255	S	255	P	155	P	115
1001	Single	2	3.15	S	350	S	635	P	63	P	32
1002	Dual (b)	4	3.30	S	400	S	280	P	86	P	20
1003	Single	2	3.15	R	410	S	130	P	200	P	10
1005	2 + 1 (a)	3	3.30	S	520	S	340	P	35	P	8
1006	Single	2	3.50	S	85	S	150	P	50	P	80
1007	Single	2	3.30	S	315	S	430	–	–	–	–
1008	2 + 1 (b)	3	3.30	S	160	S	140	–	–	–	–
1009	Dual (a)	4	3.30	S	115	S	550	–	–	–	–
1010	Dual (a)	4	3.30	S	140	S	66	–	–	–	–
1012	Single	2	3.15	S	247						
1013	Dual (a)	4	3.30	R	670	R	380	P	65	–	–
1014	Single	2	3.15	R	550	S	1200	P	100	P	50
1015	Dual (a)	4	3.30	S	650	R	145	P	40	–	–

Dual (a): Dual Carriageway with built median island.
Dual (b): Dual Carriageway with painted median.
2 + 1 (a): One lane per direction with an additional dedicated right turning lane.
2 + 1 (b): One lane per direction with an additional dedicated right turning lane and built median island.
S: signalized intersection.
R: Roundabout.
P: Priority Intersection.

Table 4.1: Inductive loop detector location characteristics, Dimitrou, Stylinou, 2018

Depending on the carriageway design, each loop detector covers either one or two lanes in each direction. The total number of lanes covered is 46, while the three loop detectors presented in the case study cover the total of 10 lanes. For instance, detectors 2016 and 2015 serve a total of 8 lanes: two in the eastbound and two in the westbound direction each. At the same time, detector 2014 serves only two lanes: one in the southbound direction, and another in the northbound (Dimitrou, Stylinou, 2018).

4.2 Data preparation

To be able to train the deep neural networks, training data must adhere to the standards of uniformity, consistency and should be free of errors, outliers and missing values. Otherwise, the model might be inaccurate and lead to the wrong predictions. However, the real world raw data rarely meets these demands and needs to be processed in accordance with the requirements of a model. In this project, the process of data preprocessing takes place in six consecutive steps that is illustrated in Figure 4.3.

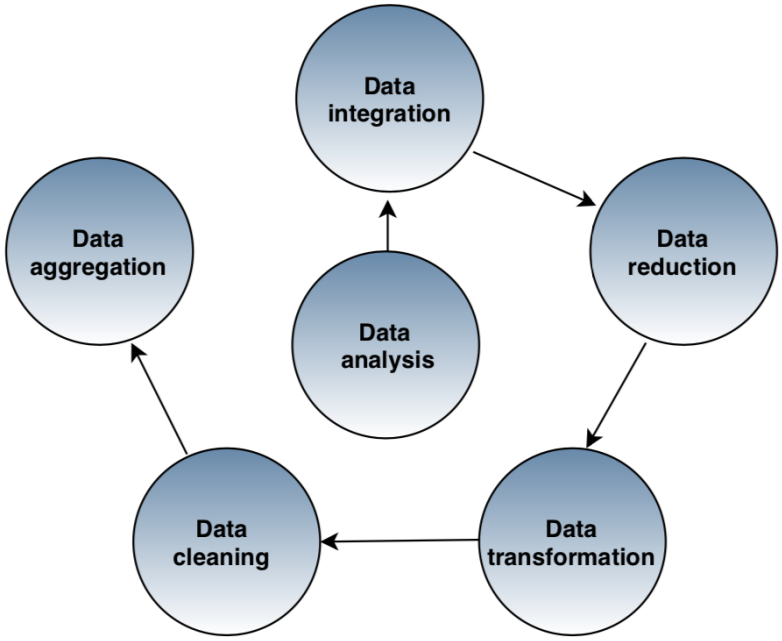


Figure 4.2: Data preprocessing steps

1. Data analysis involves the general inspection of the parameters of the data, value types and units. The raw data for each loop detector contains 43 parameters. The data structure is identical for every loop detector regardless of the number of lanes covered or technical characteristics. The structure of data for every loop detector is presented in the following table 4.2:

Parameter name	Value type	Unit measure	Description
SITE_NUMBER	Numerical	-	Number of a loop detector
SITE_ID	Numerical	-	ID of a site
SERIAL_NUMBER	Numerical	-	Serial number of an individual vehicle
DATE	Numerical	Date (day of a month)	Date of a vehicle passing
TIME	Numerical	Time (seconds)	Time of a vehicle passing

LANE	Numerical	-	Lane of passing of a vehicle
DIRECTION	Categorical	-	Direction of a lane
SPEED	Numerical	Km/h	Individual vehicle speed
CLASS_INDEX	Numerical	-	Inex of class
CLASS	Numerical	-	Type of a vehicle (1: passenger car, 2: HGV)
LENGTH	Numerical	Centimeters (sm)	Length of an individual vehicle
VALIDITY	Numrical	Seconds (sec)	Duration of a vehicle passing over a detecor
STRADDLE	-	-	-
GROSS	-	-	-
HEADWAY	Numerical	Milliseconds (ms)	Temporal headway (the time difference between the fronts of two consecutive vehicles)
GAP	Numerical	Milliseconds (ms)	Temporal gap (the time difference between the back of the following vehicle and the front of the preceding vehicle)
LEGAL_STATUS	-	-	-
CHASSIS_CODE	Numerical	-	Code of a chassis correspondinf to the vehicle type
AX_WT1- AX_WT25	-	-	-

Table 4.2: Data description

SITE_NUMBE	SITE_ID	SERIAL_NUM	DATE	TIME	LANE	DIRECTION	SPEED	CLASS_INDE	CLASS	LENGTH	VALIDITY	STRADDLE	GROSS	HEADWAY	GAP	LEGAL_STATI	CHASSIS_CO	AX_WT1	AX_WT2
1015	3051	23126913	9/18/15 00:00	12/30/99 00:00	1	EASTBOUND	84	81	2	340	0	0	0	600000	5740	0	0		
1015	3051	23126914	9/18/15 00:00	12/30/99 00:00	4	WESTBOUNI	61	81	2	400	0	0	0	600000	7840	0	0		
1015	3051	23126915	9/18/15 00:00	12/30/99 00:00	3	WESTBOUNI	71	81	2	350	0	0	0	600000	10000	0	0		
1015	3051	23126916	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	75	81	2	120	0	0	0	600000	10000	0	0		
1015	3051	23126917	9/18/15 00:00	12/30/99 00:00	4	WESTBOUNI	69	80	1	120	0	0	0	3000	4450	0	0		
1015	3051	23126918	9/18/15 00:00	12/30/99 00:00	1	EASTBOUND	72	81	2	440	0	0	0	10900	10000	0	0		
1015	3051	23126919	9/18/15 00:00	12/30/99 00:00	1	EASTBOUND	65	81	2	400	0	0	0	1500	1960	0	0		
1015	3051	23126920	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	59	81	2	450	0	0	0	10400	10000	0	0		
1015	3051	23126921	9/18/15 00:00	12/30/99 00:00	3	WESTBOUNI	52	80	1	120	0	0	0	12400	10000	0	0		
1015	3051	23126922	9/18/15 00:00	12/30/99 00:00	4	WESTBOUNI	50	81	2	410	0	0	0	10000	10000	0	0		
1015	3051	23126923	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	55	81	2	450	0	0	0	1500	1450	0	0		
1015	3051	23126924	9/18/15 00:00	12/30/99 00:00	3	WESTBOUNI	50	81	2	320	0	0	0	2000	2280	0	0		
1015	3051	23126925	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	59	81	2	430	0	0	0	2300	2700	0	0		
1015	3051	23126926	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	57	81	2	410	0	0	0	1100	1020	0	0		
1015	3051	23126927	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	55	81	2	390	0	0	0	1600	1590	0	0		
1015	3051	23126928	9/18/15 00:00	12/30/99 00:00	1	EASTBOUND	86	81	2	420	0	0	0	13400	10000	0	0		
1015	3051	23126929	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	61	81	2	360	0	0	0	3000	3860	0	0		
1015	3051	23126930	9/18/15 00:00	12/30/99 00:00	1	EASTBOUND	82	81	2	450	0	0	0	1500	2480	0	0		
1015	3051	23126931	9/18/15 00:00	12/30/99 00:00	4	WESTBOUNI	66	80	1	110	0	0	0	10400	10000	0	0		
1015	3051	23126932	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	62	80	1	120	0	0	0	1500	2190	0	0		
1015	3051	23126933	9/18/15 00:00	12/30/99 00:00	3	WESTBOUNI	78	81	2	390	0	0	0	9500	10000	0	0		
1015	3051	23126934	9/18/15 00:00	12/30/99 00:00	3	WESTBOUNI	74	81	2	430	0	0	0	2000	3410	0	0		
1015	3051	23126935	9/18/15 00:00	12/30/99 00:00	3	WESTBOUNI	73	81	2	360	0	0	0	600	220	0	0		
1015	3051	23126936	9/18/15 00:00	12/30/99 00:00	4	WESTBOUNI	55	81	2	430	0	0	0	4500	7290	0	0		
1015	3051	23126937	9/18/15 00:00	12/30/99 00:00	3	WESTBOUNI	61	81	2	330	0	0	0	1500	2140	0	0		
1015	3051	23126938	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	73	81	2	420	0	0	0	7200	10000	0	0		
1015	3051	23126939	9/18/15 00:00	12/30/99 00:00	4	WESTBOUNI	69	81	2	330	0	0	0	3700	5020	0	0		
1015	3051	23126940	9/18/15 00:00	12/30/99 00:00	1	EASTBOUND	75	81	2	440	0	0	0	11300	10000	0	0		
1015	3051	23126941	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	88	81	2	380	0	0	0	6100	10000	0	0		
1015	3051	23126942	9/18/15 00:00	12/30/99 00:00	2	EASTBOUND	81	81	2	410	0	0	0	9900	10000	0	0		
1015	3051	23126943	9/18/15 00:00	12/30/99 00:01	3	WESTBOUNI	76	81	2	410	0	0	0	31200	10000	0	0		

Table 4.3: Example of raw data for loop detector 1015

Data analysis leads to a conclusion that the raw data for all three loop detectors is mainly uniform according to the data types and consistent across all days. However, several parameters, such as 'ax_wt1' - 'ax_wt25' don't have any values at all, and they are excluded from data. Moreover, other variables like 'legal_status', 'chassis_code' and 'gross' contain only zero values and are additionally eliminated. In addition to this, the columns date and time don't correspond to panda's time value type. The values of the 'date' column have a form 'day/month/year 00:00:00', while the 'time' column contains the values of a form '12/30/1999 hours/minutes/seconds'. Also, some of the rows corresponding to vehicle observations were missing entirely.

2. Since the raw data is divided into several files corresponding to a particular day, the process of data integration into a single file is necessary. Overall, the data is integrated into three separate files for each loop detector containing the vehicle records for three full months. The data unification is performed using the python programming language and the ‘pandas’and 'os' framework.

3. As the purpose of the case study is to perform the relevant short term traffic flow forecasting on links according to directions, the derivation of necessary parameters from the raw data is required. The following parameters necessary for determining the traffic flow are extracted from the raw data: 'date', 'time', 'lane', 'direction', 'speed', and 'headway'.

4. As mentioned in the data analysis section, 'time' and 'date' columns of the raw data do not correspond to panda's time value type. To be compliance with the format, we need to remove the last six and the first ten characters of the values corresponding to 'date' and 'time' columns respectively. Then, the columns are transformed into a single one with the resulting format of type 'day/month/year hours/minutes/seconds'.

	TIME	LANE	DIRECTION	SPEED	HEADWAY
0	##########	1	EASTBOUND	84	600000
1	##########	4	WESTBOUNI	61	600000
2	##########	3	WESTBOUNI	71	600000
3	##########	2	EASTBOUND	75	600000
4	##########	4	WESTBOUNI	69	3000
5	##########	1	EASTBOUND	72	10900
6	##########	1	EASTBOUND	65	1500
7	##########	2	EASTBOUND	59	10400
8	##########	3	WESTBOUNI	52	12400
9	##########	4	WESTBOUNI	50	10000
10	##########	2	EASTBOUND	55	1500
11	##########	3	WESTBOUNI	50	2000
12	##########	2	EASTBOUND	59	2300
13	##########	2	EASTBOUND	57	1100
14	##########	2	EASTBOUND	55	1600

Table 4.4: Data after transformation step

5. Most of the deep learning algorithms cannot work with missing values which makes it necessary to fill them in. As mentioned in the data analysis section, some of the rows in the raw data corresponding to vehicle observations are missing. The solution to overcome this issue is to calculate the median of each attribute in the data set and fill the missing values using the median value. The data cleaning is performed with the help of Imputer class of Scikit - Learn package.

6. The last step is to aggregate the data according to directions. Aggregation is performed in 5 minutes intervals.

First of all, we need to derive the traffic flow attribute using available data. We start with the harmonic speed, which defined as the harmonic mean of speeds passing a point during a period of time:

$$\overline{v_s} = \frac{N}{\sum_{n=1}^{N} \frac{1}{v_n}} \text{ (km/h)}$$

Where N is the number of vehicles that pass the observation section over the period of time Δt (5 minutes). In the case of 5 minutes aggregation, N would correspond to the number of vehicles that fit within 5-minute intervals.

The next step is to aggregate the temporal headway $\overline{h_t}$ (sec), which is defined as the mean of temporal headway attribute divided by 1000 to convert it into seconds. Then, we derive the space headway h_s from the temporal headway using the formula:

$$\overline{h_s} = \overline{v_s} * \overline{h_t} \text{ (m)}$$

Traffic density is defined as the number of vehicles that exist on a road segment Δs at a given point in time t. Density and space headway are related to each other in the following way:

$$k = \frac{1}{\overline{h_s}} \text{ (veh/km)}$$

Traffic flow is defined as the number of vehicles that cross the observation point within one hour. It can be found using the fundamental equation of traffic flow:

$$q = k * \overline{v_s}$$

The visual representation of aggregated data is given in table 4.5:

TIME	SPEED	HEADWAY	SPACE_HEADWAY	DENSITY	FLOW
9/18/15 00:00	64.0201947	17.690099	314.5898843	3.17874175	203.503666
9/18/15 00:05	64.300307	7.07738095	126.4104911	7.91073582	508.662742
9/18/15 00:10	65.3258654	7.99315068	145.0443016	6.89444527	450.385604
9/18/15 00:15	64.3556912	9.6640625	172.7603951	5.7883637	372.514147
9/18/15 00:20	65.036012	7.11904762	128.609574	7.7754709	505.685619
9/18/15 00:25	58.5449908	11.9392157	194.1614647	5.15035258	301.527344
9/18/15 00:30	66.5641999	9.86181818	182.3455657	5.48409278	365.044248
9/18/15 00:35	67.0549406	12.9705882	241.5950064	4.1391584	277.55102
9/18/15 00:40	67.162298	18.49375	345.0229858	2.89835762	194.660358
9/18/15 00:45	63.7705051	24.2952381	430.366557	2.32360062	148.177185
9/18/15 00:50	64.9070142	18.0583333	325.5868049	3.07137754	199.353946
9/18/15 00:55	68.8665132	25.436	486.5801747	2.05515977	141.531687
9/18/15 01:00	64.2833661	26.5454545	474.0086588	2.1096661	135.616438
9/18/15 01:05	66.4449182	28.2789474	521.9423183	1.91592052	127.303183
9/18/15 01:10	63.4250367	38.51875	678.6258702	1.47356599	93.4609768

Table 4.5: Example of aggregated data for loop detector 1015 in the westbound direction

The resulting data is seasonal due to peak and off-peak conditions and do not have a trend. The plots of traffic data for all loop detectors in all directions is presented in the following figures .

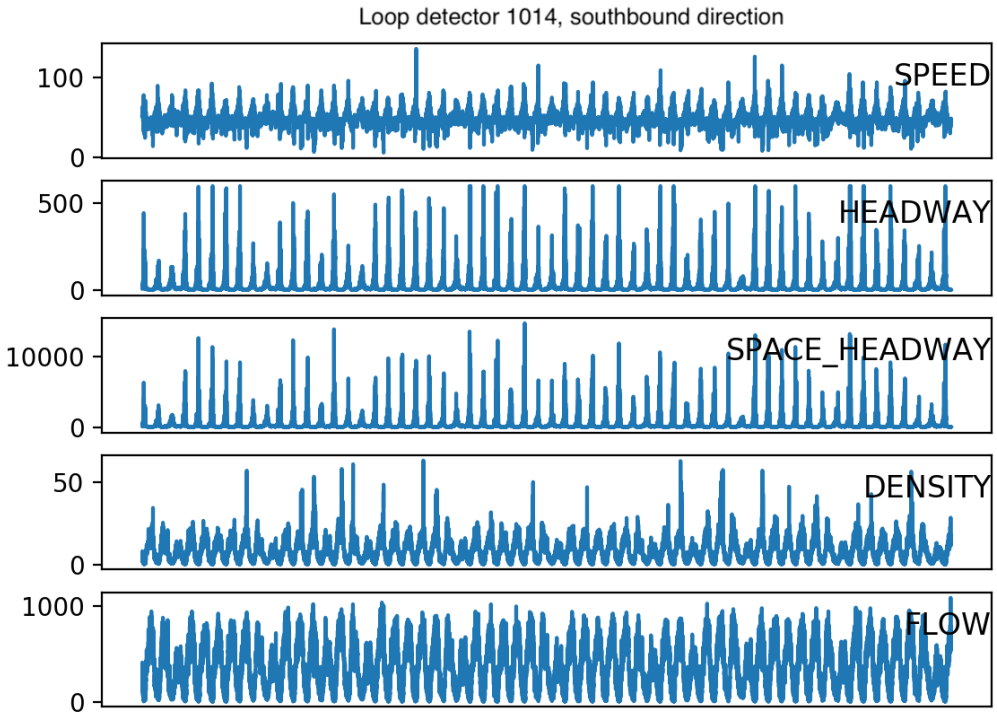


Figure 4.3: Data plot for loop detector 1014 in southbound direction

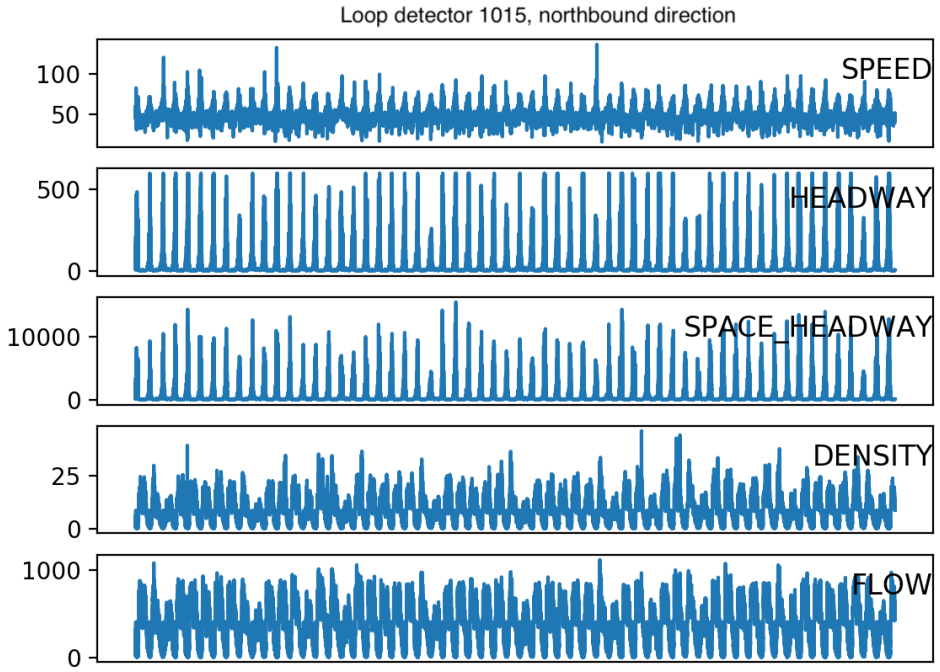


Figure 4.4: Data plot for loop detector 1014 in northbound direction

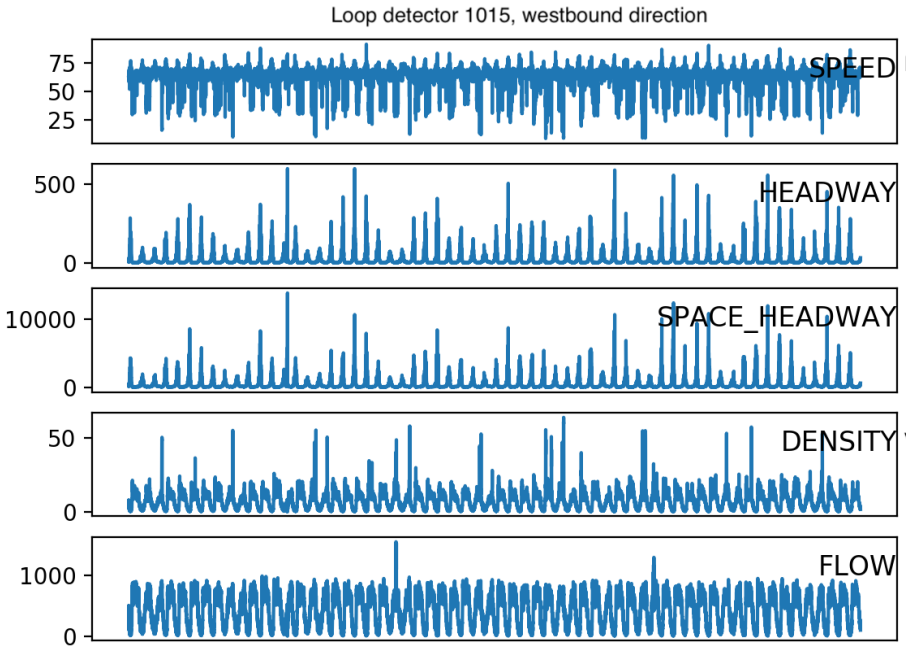


Figure 4.5: Data plot for loop detector 1015 in westbound direction

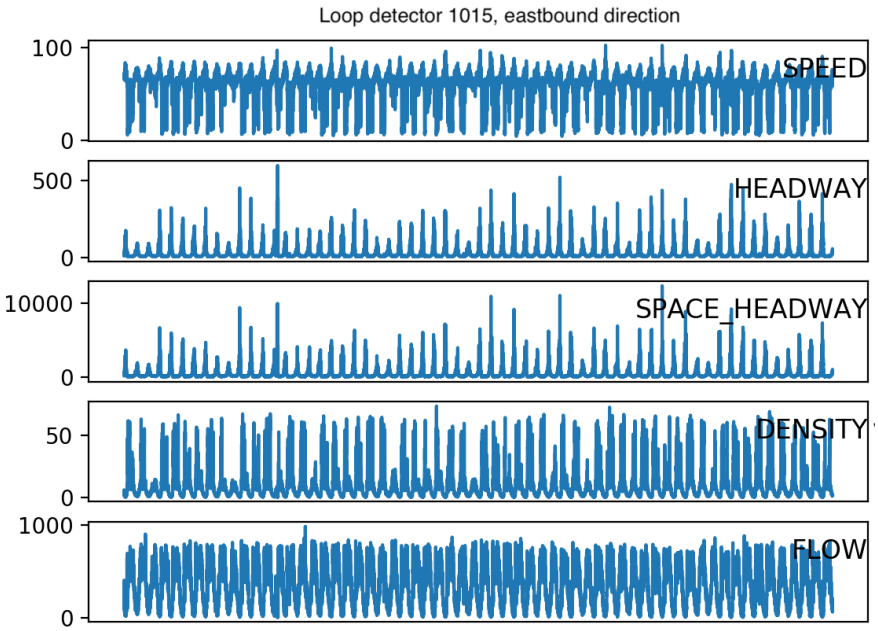


Figure 4.6: Data plot for loop detector 1015 in eastbound direction

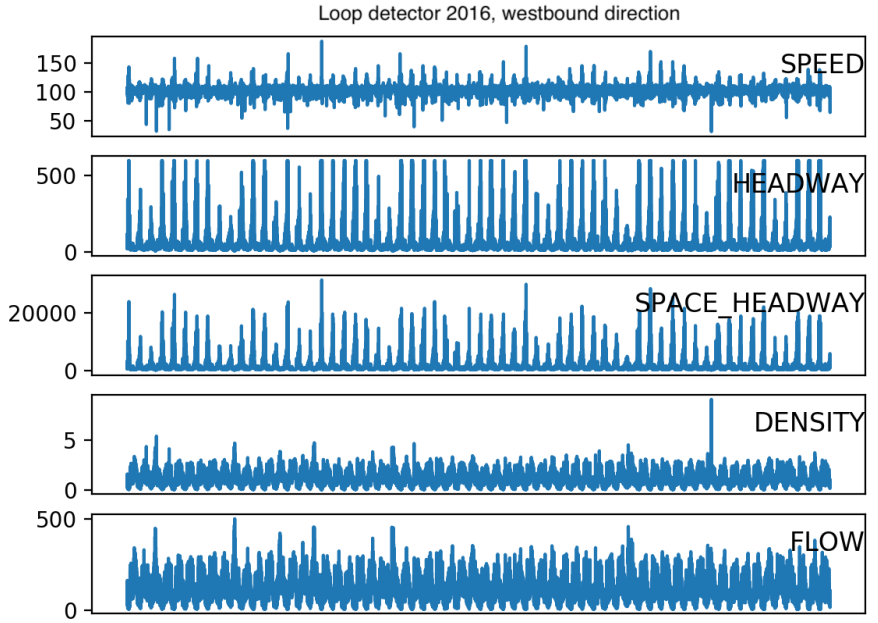


Figure 4.7: Data plot for loop detector 1016 in westbound direction

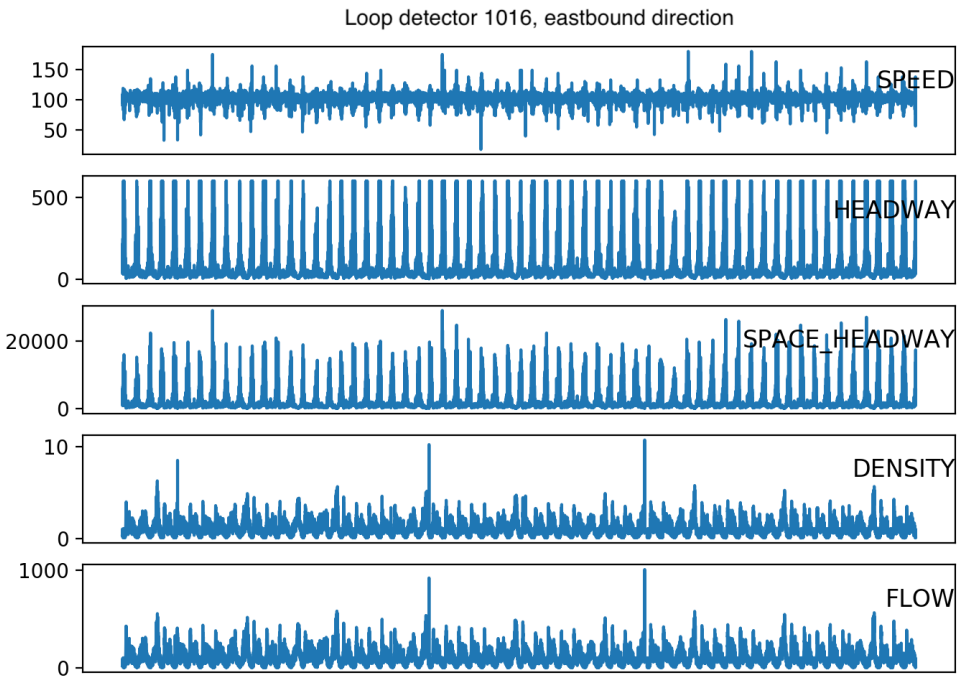


Figure 4.8: Data plot for loop detector 1016 in eastbound direction

4.3 Traffic flow prediction procedure using neural networks

In this section, we present the strategy to perform the traffic flow forecasting on links using neural networks.

The first step is to determine whether the data in each direction has outliers. We create a boxplot to determine whether the data has outliers and the need for their deletion is present. The outliers detection is performed using z-score of data. The z-score is defined as the number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured. Then, the outliers are deleted, and the data is smoothed using the Infinitive Impulse Response filter for one-dimensional time series array. Outliers detection and data smoothing are performed using SciPy python library. The examplde of data smoothing is presented in figure 4.9

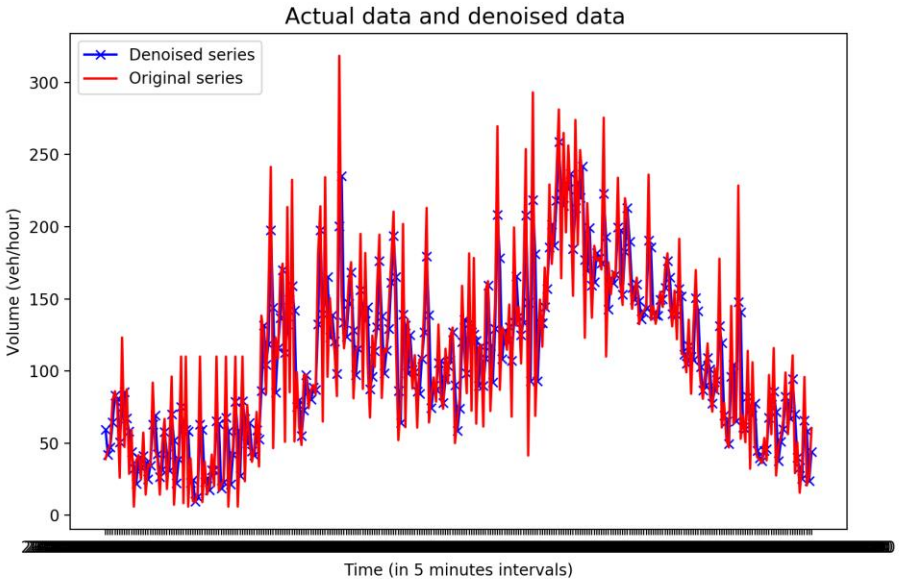


Figure 4.9: Example of actual data and denoised data using IRF filter

We are aiming to predict the traffic flow over the future 50 minutes. To be able to identify the best model, we perform a comparison based on three factors: final MSE, the number of iterations to converge, the speed of going through a single iteration. Then, the benefits and drawbacks of every model are discussed, and the preferable model is selected. As discussed in the methodology section, the procedure is conducted using a sequence-to-sequence traffic model, meaning that the error is calculated at every time step, and the traffic tensor only at the last time step is extracted.

When we transform a one-dimensional array with data into the three-dimensional traffic tensor, we are dealing with the lack of training batches. For instance, for a vector with data containing 17855 observations, after the transformation, we get a tensor of shape (297, 60, 1) where the first dimension corresponds to the number of training batches. That is not enough, and to alleviate the problem, we come up with a solution to extend the initial one-dimensional sequence. As we can see from the plots in the data analysis section, the data is seasonal and repeats daily. This means that a one-dimensional array with data can be extended 35 times to constitute 10000 training batches as shown in Figure 4.10.

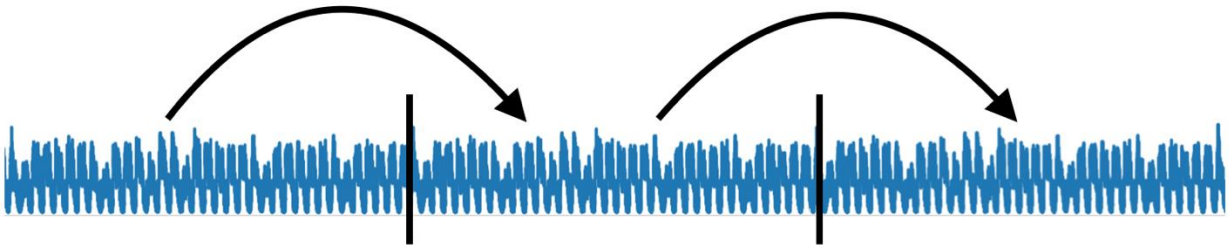


Figure 4.10: Process of extending the training data

Hyperparameter tuning is a necessary process to achieve the best accuracy. For instance, if we have a lack of layers in our network, the validation error might prevail over the training error, and the model won't perform well on new data. We define three sets of hyperparameters: number of hidden layers, number of neurons in each layer and the learning rate. We determine the space of hyperparameters that the model is going to use: the number of hidden layers is between 0 to 3, the number of neurons in each hidden layer lies within the range from 0 to 20, the learning rate is between 0,05 and 0,005. This implies that we have 840 possible combinations of models with all hyperparameters. Verifying the performance of every single model would take tremendous time and computational effort, especially considering the lack of computational power. To overcome this issue, the hyperparameters tuning is performed randomly applying k-fold cross-validation using RandomizedSearchCV class of SciKit Learn. For every single direction, the resulting number of hidden layers turns out to be 1, the number of neurons in each layer varies around 20, and the learning rate is typically close to 0.01. For the sake of time-saving, it has been decided to keep the aforementioned combination of hyperparameters for each model in every direction, as it proved to be successful for some directions.

Before feeding the data into the training algorithm, data scaling is necessary; otherwise, the algorithm performs poorly. The data is transformed on a scale from 0 to 1. After the predictions are made, the predicted values can be rescaled backwards. Each value is scaled according to the following formula:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

We split the data in each direction into three parts: train set, which is 70 % of all data, validation set - 20 % and test set - 10%. We train our model on the train set and determine its performance on the validation set to make sure that there is not any significant difference in accuracy on train and validation sets. The software used for training are python programming language, Keras deep learning library and Google's collaborative platform for cloud computing. Finally, we use the last instances of our test set to visualize the real values against predictions for all loop detectors in all directions using every variation of the neural architecture.

4.4 Process of models training

We provide the forecasting for every loop detector in every direction. First of all, we extract a one-dimensional array with traffic flow. Then, for every architecture, the data is sclaed, extended and, according to the proposed methodology, reshaped into a three-dimensional tensor containing the batches of traffic flow vectors. Data reshaping is performed using NumPy python library. For loop detector 1016, the resulting shape of the extended tensor in both directions is (10545, 60, 1). Then, the resulting tenor with training data is split into three parts: 70 % of bathes are assigned to the training set, 20 % - to the validation set, and 10% - to the test set. For every set, the acquired tensor is reshaped into two tensors: one with features and another with targets. The first 50 data points of every batch go to the tensor with the features and remaining 10 points go to a tensor with targets. Hence, the final number of tensors for train, validation and test sets is 6. For instance, the tensor with training features has a shape (7381, 50, 1), while a tensor with training targets has a shape (7381, 50, 10) from which, according to sequence-to-sequence model, only the vector corresponding to the last element of the second dimension is obtained.

At each iteration, we calculate the error both on the test set and the validation set to make sure that the model will perform well on new data. When the model is converged, we derive a single training feature from the test set to predict an array of 10 values (50 minutes forecast) and visualise the actual values against the predicted values.

For this chapter, the visualisation is performed for loop detector 1016 only, and the remaining plots are given in the Appendix.

4.5 Models estimation and forecasting results for all loop detectors

To start with, we test the architecture of **deep neural network consisting of conventional RNN cells**. The first layer of the proposed architecture is the layer with 20 RNN cells; the hidden layer contains 20 RNN cells, and the last layer is a dense layer with ten cells wrapped in a time distributed layer.

We compile our model using the MSE cost function $J(\theta) = \frac{1}{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ and Adam optimizer as proposed in methodology. We train the model on 700 iterations applying the early stopping technique with the patience of 10 (meaning that if the model doesn't improve either on the test or validation sets for 10 consecutive iterations, the training terminates and the best parameters are saved). Then, the plot of training against validation error is created, as well as the plot of real values against the predicted values on the test set.

The forecasting performance is compared for every available model. For loop detector 1016, the learning curves and the real against predicted values for both eastbound and westbound directions are plotted in the following figures:

Eastbound direction is shown in Figure 4.11.

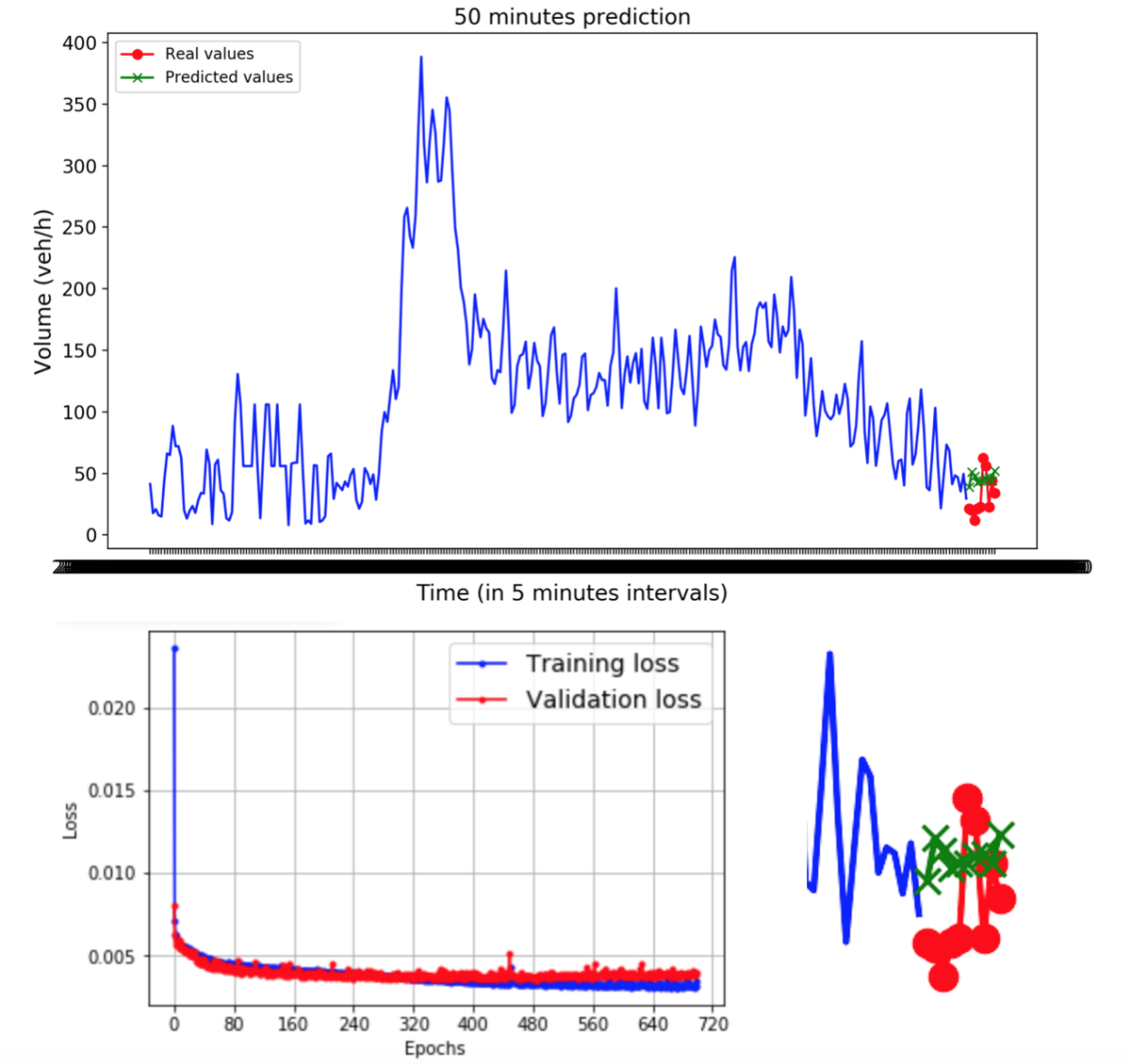


Figure 4.11: Learning curve and plot of predicted against real values for loop detector 1016 in eastbound direction, conventional RNN architecture

As we can see, the proposed model provides quite reasonable predictions. The resulting loss is - 0.0040, validation loss - 0.0037.

The second neural architecture is the neural networks with LSTM cells. The structure is the same as the previous model. LSTM training and validation losses, as well as prediction are demostratrd in figure 4.12.

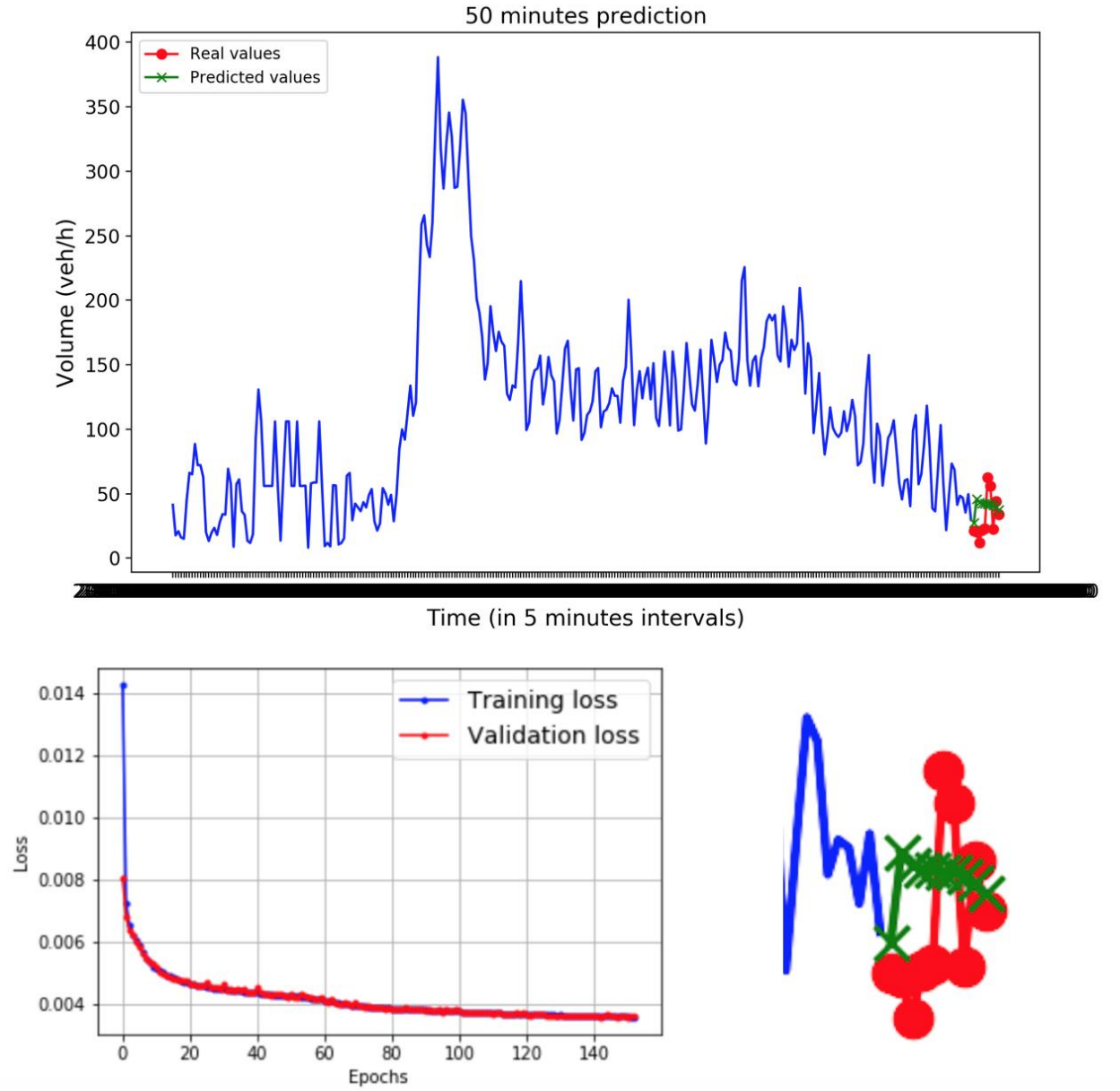


Figure 4.12: Learning curve and plot of predicted against real values for loop detector 1016 in eastbound direction, LSTM architecture

The resulting loss of LSTM model is - 0.0028, validation loss - 0.0027.

The second neural architecture is the neural networks with GRU cells. The structure is the same as the previous two models. GRU training and validation losses, as well as prediction are demostratrd in figure 4.13.

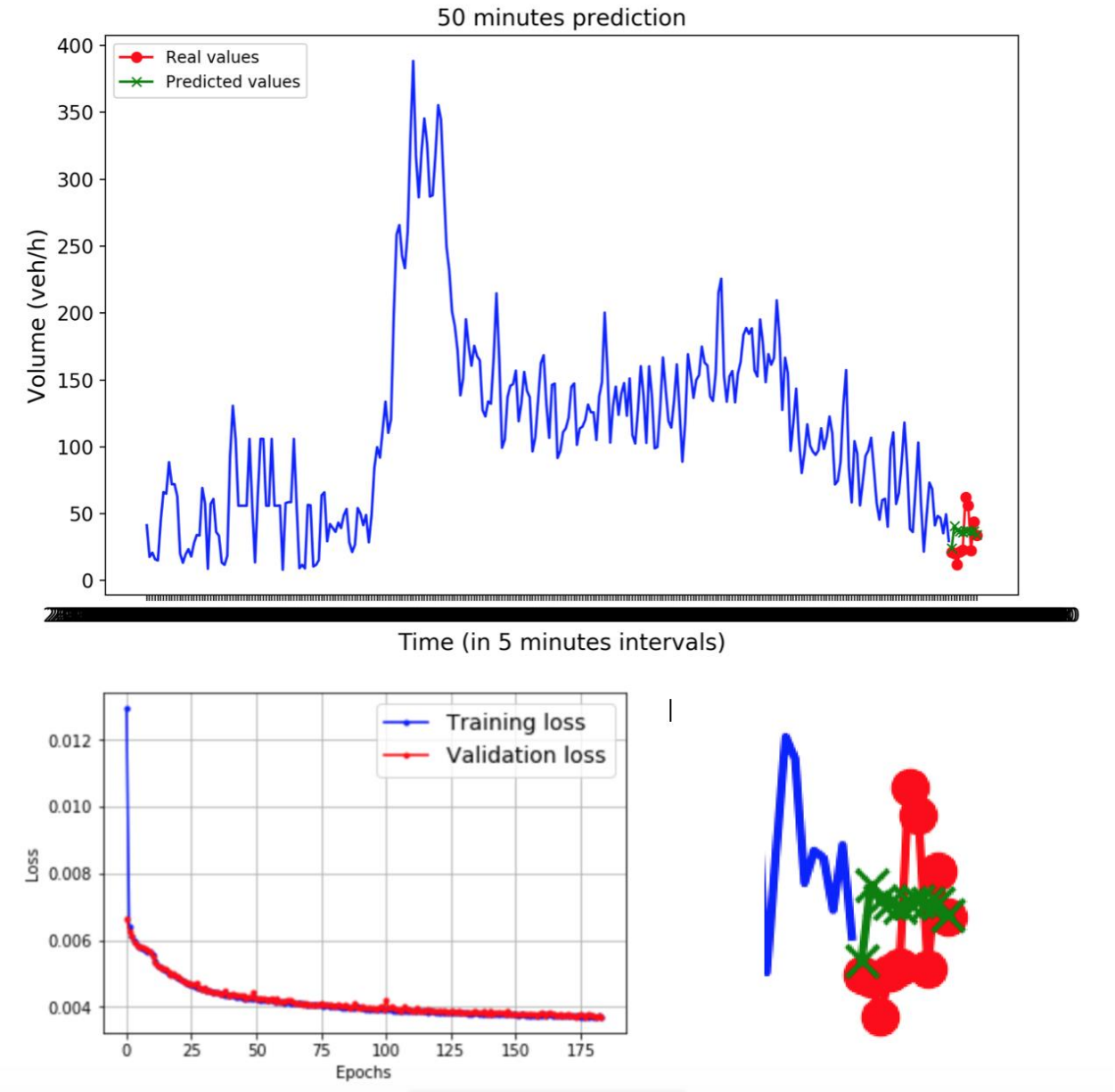


Figure 4.13: Learning curve and plot of predicted against real values for loop detector 1016 in eastbound direction, GRU architecture

The resulting loss of GRU model is - 0.0028, validation loss - 0.0027.

The WaveNet model starts with an input layer that accepts a 1-dimensional array of shape [None, 1]. After, a 1-D convolutional layer, using causal padding is added to ensure that the convolutional layer will not make wrong predictions. Then, similar pairs of layers using growing dilation rates: 1, 2, 4, 8, and again 1, 2, 4, 8 are added. Finally, the output layer is a convolutional layer with ten filters of size one without an activation function. (Geron, 2019) The structure of WaveNet architecture used in the case study is presented in figure 4.14 (Geron, 2019).

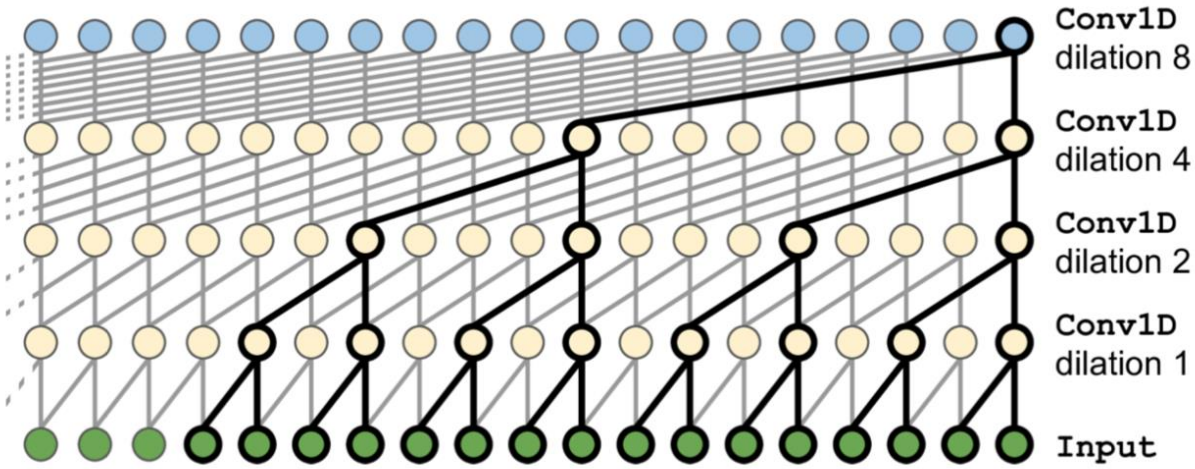


Figure 4.14: WaveNet Architecture, Geron, 2019

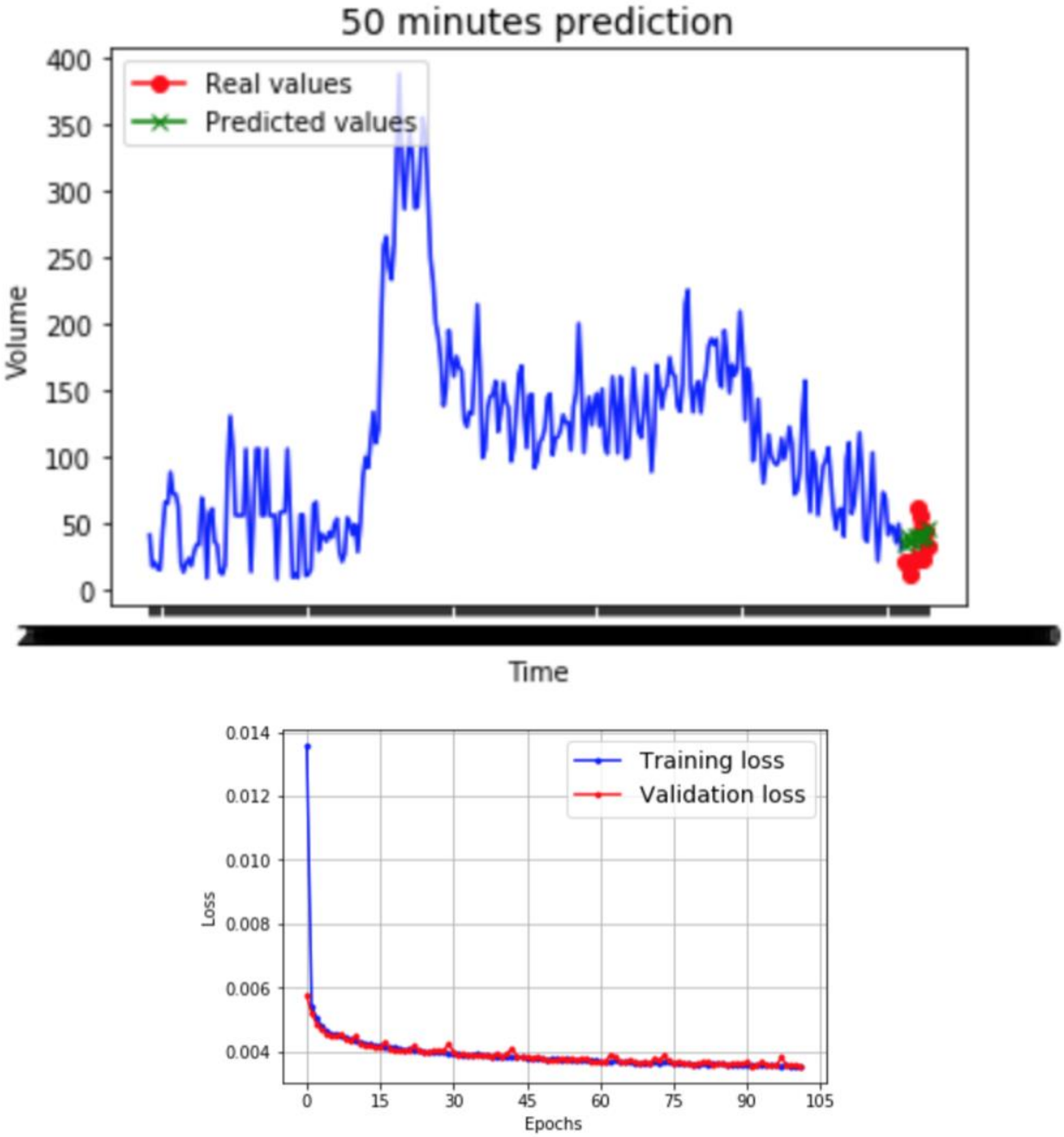


Figure 4.15: Learning curve and plot of predicted against real values for loop detector 1016 in eastbound direction, WaveNet architecture

The resulting loss of WaveNet model is - 0.0028, validation loss - 0.0028.

The remaining plots for every loop detector in every direction are given in the Appendix.

The final values of training and validation errors on scaled data for every loop detector in every direction, as well as the required number of iterations and training time per iteration, are presented in the tables 4.6 – 4.9:

Parameter	Direction	RNN	LSTM	GRU	WaveNet
MSE (train ;validation)	west	0.0060; 0.0061	0.0038; 0.0038	0.0041; 0.0041	0.0040; 0.0040
	east	0.0040; 0.0037	0.0028; 0.0027	0.0029; 0.0029	0.0029; 0.0028
Iterations	west	140	360	430	290
	east	154	130	185	100

Table 4.6: Final MSE and required number of iterations for loop detector 1016

RNN	LSTM	GRU	WaveNet
8.7	13.85	10.03	4.5

Table 4.7: Training time for one full iteration for every architecture (in seconds)

Parameter	Direction	RNN	LSTM	GRU	WaveNet
MSE (train ;validation)	west	0.0020; 0.0020	0.0013; 0.0013	0.0013; 0.0013	0.0014; 0.0015
	east	0.0059; 0.0062	0.0037; 0.0037	0.0029; 0.0029	0.0045; 0.0045

Iterations	west	168	500	600	225
	east	72	130	340	110

Table 4.8: Final MSE and required number of iterations for loop detector 1015

Parameter	Direction	RNN	LSTM	GRU	WaveNet
MSE (train ;validation)	south	0.0055; 0.0056	0.0038; 0.0040	0.0046; 0.0045	0.0042; 0.0043
	north	0.0062; 0.063	0.0052; 0.0052	0.0056; 0.0057	0.0058; 0.0059
Iterations	south	98	230	130	150
	north	180	190	130	100

Table 4.9: Final MSE and required number of iterations for loop detector 1014

Chapter 5. Discussion and conclusion

To answer the research question which deep-learning technique is preferable for short-term traffic forecasting, we have to compare the results on scaled data obtained during the estimation.

For loop detector 1016 in the westbound direction, the least MSE is demonstrated by the LSTM model, being almost two times more precise than conventional RNN. However, in comparison with the remaining two models, the difference is not substantial. For the east direction, the WaveNet architecture became the most efficient in terms of the number of iterations required for convergence. For the west direction, the RNN converges faster. However, in terms of MSE score, the WaveNet is better than RNN. Also, it needs the least time to go through a single epoch, being 2.2 times more efficient than GRU and 3.1 times than LSTM.

For loop detector 1015, the situation is quite the same. RNN is inferior in terms of accuracy to the other three models which demonstrate approximately the same performance in comparison to each other. Additionally, the RNN architecture needs fewer iterations to converge, but this advantage is diminished by the fact that the WaveNet needs less time per one iteration making the training time approximately equal.

Finally, for loop detector 1014 in both directions, the RNN architecture provides the worst performance among four available models. LSTM delivers the best accuracy, followed by WaveNet and GRU. In the northbound direction, the situation is the same with the only difference being that GRU is inconsiderably more accurate than WaveNet.

Comparison of all models for all loop detectors in every direction is provided in the following table:

Detector	Direction	RNN	LSTM	GRU	WaveNet
1016	west	4	1	3	2
	east	4	1	3	2
1015	west	4	1	2	3
	east	4	2	1	3
1014	south	4	1	3	2
	north	4	1	2	3

Table 5.1: The rank of models in terms of accuracy

From the table above, we can conclude that the RNN architecture is not considered further as it provides the worst MSE score. In terms of accuracy, LSTM architecture is the best as it outperforms the remaining two models in most of the cases. WaveNet and GRU are quite similar often taking second and third place. Considering that the same data preprocessing techniques are performed on data in every direction, we can conclude that the performance of a particular model may depend on the statistical characteristics of data such as variance, standard deviation and noisiness.

On the other hand, by carefully examining the average residuals per instance on the unscaled data, we can conclude that for every prediction, the difference in the residual per instance among all three models rarely exceeds two vehicles per hour. For the ITS tasks, such infelicity is very insignificant meaning that the speed of convergence plays a decisive role in choosing the best model. In that aspect, the WaveNet architecture is an undoubtedly preferable choice as it needs 4.5 seconds on average to go through a single training epoch outperforming the other models more than two times.

The rapid development of intelligent transport systems (ITS) and increased availability of data collected from loop detectors and other devices made it possible to apply deep-learning techniques to provide traffic flow forecasting on links and in networks. In this thesis, an attempt was made to determine the best deep-learning method to provide the short-term traffic forecasting on links considering the limitation in the size of data.

First of all, the relevant literature review was conducted which leads to developing the most promising deep-learning techniques. First, the study provides an overview of predictive methods for short-term traffic forecasting, and the conclusions are made concerning the most promising among them. Then, it presents a review of the combination of LSTM and convolutional architecture to capture short – term, long – term and spatial-temporal dependencies in traffic data. Finally, the literature review is summed up by giving a revision of LSTM and GRU methods and their comparison.

The methodology section describes the deep-learning methods used in the case study and provides their mathematical representation. Four architectures are considered - deep networks with conventional RNN cells, LSTM cells, GRU cells and WaveNet.

Data used in the case study was collected from three loop detectors installed in the urban network in Nicosia, Cyprus. The case study describes the data attributes and provides the necessary data preprocessing steps such as analysis and cleaning. Then, the data is aggregated into 5 minutes intervals according to directions. In each direction, the resulting number of observations is approximately 17 thousand.

After data extension and scaling are performed, the data gets reshaped and fed into four different deep-learning networks. Following the hyperparameter tuning, the number of hidden layers is established to be 1, the number of cells in each layer is 20, and the learning rate is equal to 0.001.

The results of training do not determine a preferable model in terms of accuracy. However, in terms of convergence speed, WaveNet is a preferred choice as it converges much faster than other models.

- The lack of computational power doesn't allow to verify the best set of hyperparameters to deliver the most reliable forecasting possible. It is preferable to train the models using computers with high computational power to consider the larger space of hyperparameters.

- Data extension techniques proposed in the case study do not allow to capture the long-term patterns. The data collected over three consecutive months is not enough to provide enough training batches for training algorithms signifying that the artificial extension of data is necessary.

- The high correlation of data attributes made it pointless to perform the multivariate time series forecasting. More attributes that are less correlated with each other should be collected to explain the dependent variable better.

- Additionally, the dataset did not include any attributes related to the structural characteristics of the study area. Hence, it was impossible to create the origin-destination matrix for the network-wide traffic prediction proposed in the literature review.

References

Abduljabbar, R. and Dia, H. (2019). Applications of Artificial Intelligence in Transport: An Overview. Sustainability, 11 (189), pp. 1 – 24. Available at: https://www.mdpi.com/2071-1050/11/1/189 [Accessed 25 Sept. 2019].

Azzouni, A. and Pujolle, G. (2017). A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction. In: 2011 IEEE 7th International Conference On Intelligent Computer Communication And Processing. [online], pp. 1-6. Available at: https://arxiv.org/abs/1705.05690 [Accessed 25 Sept. 2019].

Cho, K. and Merriënboer, van B. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. arXiv.org, [online], pp. 1-15. Available at: https://arxiv.org/abs/1406.1078 [Accessed 25 Sept. 2019].

Dimitriou, L. and Stylianou, K. (2018). Assessing rear-end crash potential in urban locations based on vehicle-by-vehicle interactions, geometric characteristics and operational conditions. Accident Analysis and Prevention, [online] 118, pp. 221-235. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0001457518300861 [Accessed 25 Sept. 2019].

Fu, R. and Zhang, Z. (2016). Using LSTM and GRU neural network methods for traffic flow prediction. In: 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC). [online] Wuhan, China: IEEE, pp 1-5. Available at: https://ieeexplore.ieee.org/document/7804912 [Accessed 25 Sept. 2019].

Geron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and Tensorflow. 2nd Edition. Boston: O'Reilly, chapter 15.

Gers, F. and Schmidhuber, J. (1990). Learning to forget: continual prediction with LSTM. In: ICANN International Conference on Artificial Neural Networks. [online] Edinburgh: IEE, pp. 850-855. Available at: https://pdfs.semanticscholar.org/e10f/98b86797ebf6c8caea6f54cacbc5a50e8b34.pdf [Accessed 25 Sept. 2019].

Greff, K. and Srivastava, K. (2017). LSTM: A Search Space Odyssey. IEEE Transactions on Neural Networks and Learning Systems, [online] 28 (10), pp. 2222 – 2232. Available at: https://arxiv.org/pdf/1503.04069.pdf [Accessed 25 Sept. 2019].

Lai, G. and Yang, Y. (2018). Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. [online] New York: ACM , pp 1-11. Available at: http://www.citethisforme.com/harvard-referencing [Accessed 25 Sept. 2019].

LeCun, Y. and Bengio, Y. (2015). Deep learning. Nature, [online]521, pp. 436 – 441. Available at: https://www.researchgate.net/publication/277411157_Deep_Learning [Accessed 25 Sept. 2019].

Lipton, Z. and Berkowitz, J. (2015). A Critical Review Of Recurrent Neural Networks for Sequence Learning. arXiv.org, [online] pp. 1-38. Available at: https://arxiv.org/abs/1506.00019 [Accessed 25 Sept. 2019]

Ruder, S. (2017). An overview of gradient descent optimization algorithms. arXiv [online] pp. 1-14. Available at: https://arxiv.org/pdf/1609.04747.pdf [Accessed 25 Sept. 2019].

Van der Oord, A. and Simonyan, K. (2016). WaveNet: A Generative Model For Raw Audio. arXiv.org, [online], pp. 1-15. Available at: https://arxiv.org/pdf/1609.03499.pdf [Accessed 25 Sept. 2019].

Van Hinsbergen, C.P.IJ., van Lint, J.W.C., Sanders F.M. (2007). Short Term Traffic Prediction Models. In: ITS World Congress. [online] Beijing: TU Delft & Vialis Traffic bv, p. 4. Available at: https://www.researchgate.net/publication/228882378_Short_Term_Traffic_Prediction_Models

Yu, B. and Yin, H. (2018). Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. pp 1-2. Available at: https://arxiv.org/abs/1709.04875# [Accessed 25 Sept. 2019].

Yuankai, W. and Huachun, T. (2016). Short-term traffic flow forecasting with spatial-temporal

correlation in a hybrid deep learning framework. arXiv.org, [online] pp. 1-14. Available at: https://arxiv.org/abs/1612.01022

References of figures

Azzouni, A. and Pujolle, G. (2017). Sliding learning window. [image] Available at: https://arxiv.org/abs/1705.05690 [Accessed 25 Sept. 2019].

Cho, K. and Merriënboer, van B. (2014). An illustration of the proposed hidden activation function. The update gate z selects whether the hidden state is to be updated with a new hidden state h̃. The reset gate r decides whether the previous hidden state is ignored. [image] Available at: https://arxiv.org/abs/1406.1078 [Accessed 25 Sept. 2019].

Dimitriou, L. and Stylianou, K. (2018). Inductive Loop Detecor Locations . [image] Available at: https://www.sciencedirect.com/science/article/abs/pii/S0001457518300861 [Accessed 25 Sept. 2019].

Geron, A. (2019). A layer of recurrent neurons (left), unrolled through time (right). [image] Available at: https://github.com/ageron/handson-ml2/blob/master/15_processing_sequences_using_rnns_and_cnns.ipynb

Geron, A. (2019). A recurrent neuron (left), unrolled through time (right). [image] Available at: https://github.com/ageron/handson-ml2/blob/master/15_processing_sequences_using_rnns_and_cnns.ipynb

Geron, A. (2019). Backpropagation through time. [image] Available at: https://github.com/ageron/handson-ml2/blob/master/15_processing_sequences_using_rnns_and_cnns.ipynb

Geron, A. (2019). Deep RNN (left), unrolled through time (right). [image] Available at: https://github.com/ageron/handson-ml2/blob/master/15_processing_sequences_using_rnns_and_cnns.ipynb

Geron, A. (2019). WaveNet Architecture. [image] Available at: https://github.com/ageron/handson-ml2/blob/master/15_processing_sequences_using_rnns_and_cnns.ipynb

Greff, K. and Srivastava, K. (2017). Long Short-Term memory block. [image] Available at: https://arxiv.org/pdf/1503.04069.pdf [Accessed 25 Sept. 2019].

Lai, G. and Yang, Y. (2018). Overview of the Long- and Short-term Time-series network (LSTNet). [image] Available at: http://www.citethisforme.com/harvard-referencing [Accessed 25 Sept. 2019].

Van der Oord, A. and Simonyan, K. (2016). Overview of the residual block and entire architecture. [image] Available at: https://arxiv.org/pdf/1609.03499.pdf [Accessed 25 Sept. 2019].

Van der Oord, A. and Simonyan, K. (2016). Visualization of a stack of dilated convolutional layers. [image] Available at: https://arxiv.org/pdf/1609.03499.pdf [Accessed 25 Sept. 2019].

Yuankai, W. and Huachun, T. (2016). LSTM structure. [image] Available at: https://arxiv.org/abs/1612.01022

Yuankai, W. and Huachun, T. (2016). Neural network consisting of a 1D CNN (capture spatial features), two LSTM RNNs (capture short-term and periodic features) and followed by a fully connected layer to fuse all features to forecast traffic flow at target time point t. [image] Available at: https://arxiv.org/abs/1612.01022

References of tables

Dimitriou, L. and Stylianou, K. (2018). Inductive loop detector location characteristics. [table] Available at: https://www.sciencedirect.com/science/article/abs/pii/S0001457518300861 [Accessed 25 Sept. 2019].

Appendix

The code used in this thesis is partially based on the code used in the book by Aurelien Geron Hands-on machine learning with Scikit-Learn, Keras, and Tensorflow. 2nd Edition. Available at: https://github.com/ageron/handson-ml2


```
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
import sklearn
import sys
import io

n_steps = 50
forecast = 10

data_west = pd.read_csv('imputer_1016_west_5.csv')

#Z SCORE, OUTLIERS DELETION
flow = data_west['FLOW']
sns.boxplot(x=np.array(flow))
z = np.abs(stats.zscore(data_west['FLOW']))
data_west_o = np.array(data_west.FLOW)
print(len(data_west_o))

#EXTEND DATA
array_to_concatinate = data_west_o[288:]
for iter in range (35):
    data_west_o = np.concatenate([data_west_o,array_to_concatinate])

#SCALE AND RESHAPE DATA
scaler = MinMaxScaler()
array = data_west_o.reshape(-1, 1)
array_scaled = scaler.fit_transform(array)
flow_reshaped = array_scaled[:(len(array_scaled) - (len(array_scaled) % (n_steps+forecast

#TRAIN SET, VALIDATION SET, TEST SET
test = int(0.7 * flow_reshaped.shape[0])
valid = int(0.9 * flow_reshaped.shape[0])

X_train, y_train = flow_reshaped[:test, :n_steps], flow_reshaped[:test, n_steps:, 0] #fir
X_valid, y_valid = flow_reshaped[test:valid, :n_steps], flow_reshaped[test:valid, n_steps
X_test, y_test = flow_reshaped[valid:, :n_steps], flow_reshaped[valid:, n_steps:, 0]

#NEW MODEL TO FIT HYPERPARAMETERS
def build_model(n_hidden=1, n_neurons=20):
    model = keras.models.Sequential()
    model.add(keras.layers.SimpleRNN(n_neurons, return_sequences=True, input_shape=[None,
    for layer in range(n_hidden):
        model.add(keras.layers.SimpleRNN(n_neurons, return_sequences=True))
    model.add(keras.layers.SimpleRNN(n_neurons, return_sequences=False))
    model.add(keras.layers.Dense(10))
    optimizer = keras.optimizers.Adam()
    model.compile(loss="mse", optimizer=optimizer)
    return model

keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)

from scipy.stats import reciprocal
from sklearn.model_selection import RandomizedSearchCV

param_distribs = {
"n_hidden": [0, 1, 2],
"n_neurons": np.arange(1, 21)
"learning_rate": reciprocal(3e-4, 3e-2),"
}

rnd_search_cv = RandomizedSearchCV(keras_reg, param_distribs, n_iter=10, cv=3)
rnd_search_cv.fit(X_train, y_train, epochs=250,
```

```
                    validation_data=(X_valid, y_valid), callbacks=[keras.callbacks.EarlySto
print(rnd_search_cv.best_params_)
model = rnd_search_cv.best_estimator_.model
model.save("deep_rnn_west_50_10_hyper_new.h5")
```

Figure 7.1: Example code of hyperparameters tuning


```
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
import sklearn
import sys
import io

n_steps = 50
forecast = 10

def plot_learning_curves(loss, val_loss):
    plt.figure()
    plt.plot(np.arange(len(loss)), loss, "b.-", label="Training loss")
    plt.plot(np.arange(len(val_loss)), val_loss, "r.-", label="Validation loss")
    plt.gca().xaxis.set_major_locator(mpl.ticker.MaxNLocator(integer=True))
    plt.legend(fontsize=14)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.grid(True)

#LOAD DATA
data_west = pd.read_csv('denoised_data_1015_east.csv')
data_west_o = np.array(data_west.FLOW)

#EXRTEND DATA
array_to_concatinate = data_west_o
for iter in range (35):
    data_west_o = np.concatenate([data_west_o,array_to_concatinate])

#SCALE AND RESHAPE DATA
scaler = MinMaxScaler()
array = data_west_o.reshape(-1, 1)
array_scaled = scaler.fit_transform(array)
flow_reshaped = array_scaled[:(len(array_scaled) - (len(array_scaled) % (n_steps+forecast

#TRAIN SET, VALIDATION SET, TEST SET
test = int(0.7 * flow_reshaped.shape[0])
valid = int(0.9 * flow_reshaped.shape[0])

X_train, y_train = flow_reshaped[:test, :n_steps], flow_reshaped[:test, n_steps:, 0] #fir
X_valid, y_valid = flow_reshaped[test:valid, :n_steps], flow_reshaped[test:valid, n_steps
X_test, y_test = flow_reshaped[valid:, :n_steps], flow_reshaped[valid:, n_steps:, 0]

# NEW MODEL TO FIT HYPERPARAMETERS
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(forecast)
])

model.compile(loss="mse", optimizer="adam")
early_stopping_cb = keras.callbacks.EarlyStopping(patience=15, restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=700,
                    validation_data=(X_valid, y_valid))
model.save("deep_rnn_1015_east.h5")
plot_learning_curves(history.history["loss"], history.history["val_loss"])

#1 HOUR FORECAST
model = keras.models.load_model("deep_rnn_1015_east.h5")

y_real_resacled = scaler.inverse_transform(y_test[-1,:].reshape(-1, 1)) #rescale last y_t
y_pred = model.predict(X_test[-1, :].reshape(-1, n_steps, 1)) #perdict last y_test based
y_pred_rescaled = scaler.inverse_transform(y_pred).reshape(-1, 1) #rescale prediction, tr
```

Figure 7.2: Example code for data preprocessing and model training for RNN

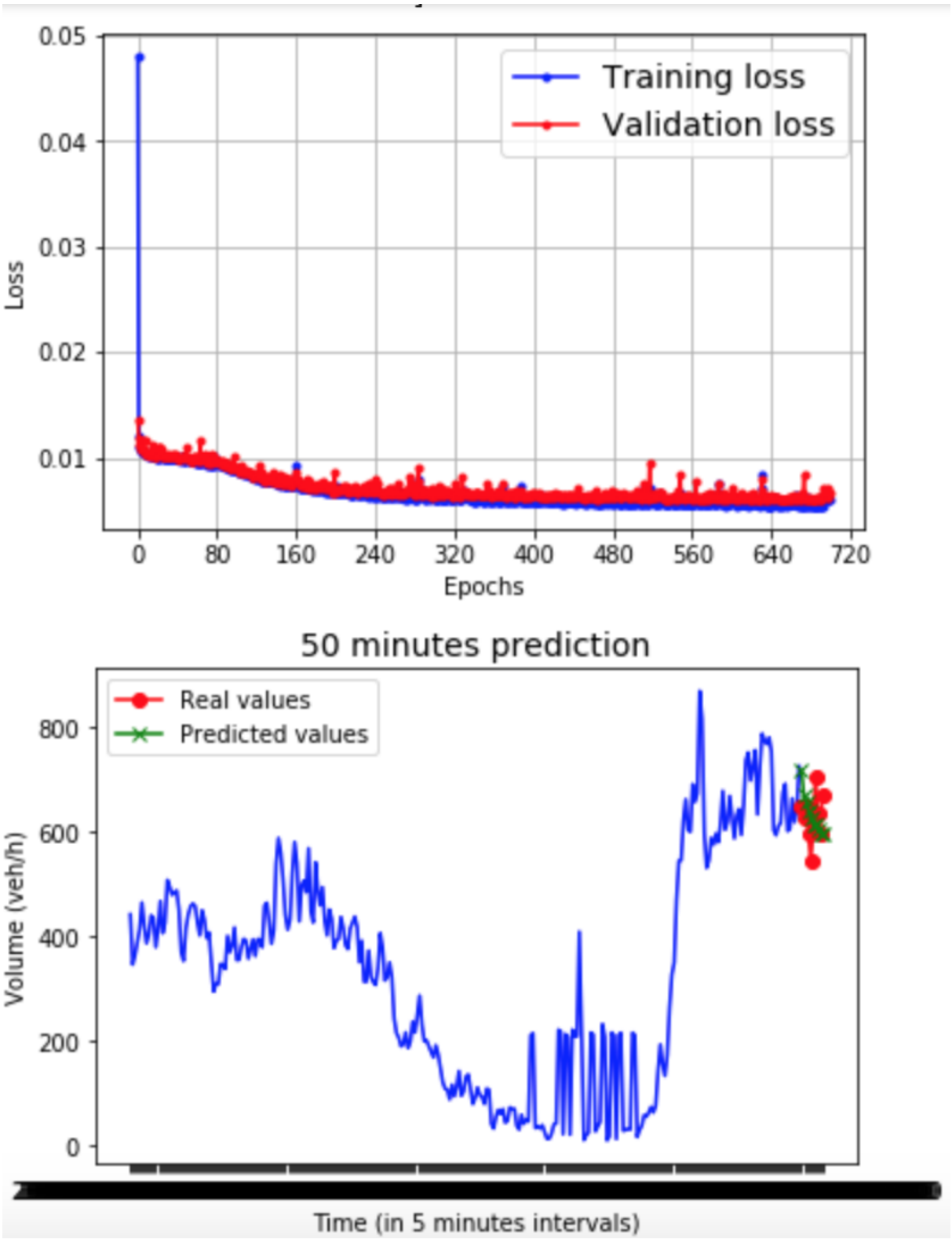


Figure 7.3: Learning curve and plot of real values agains prediction, RNN, loop detector 1014, northbound direction

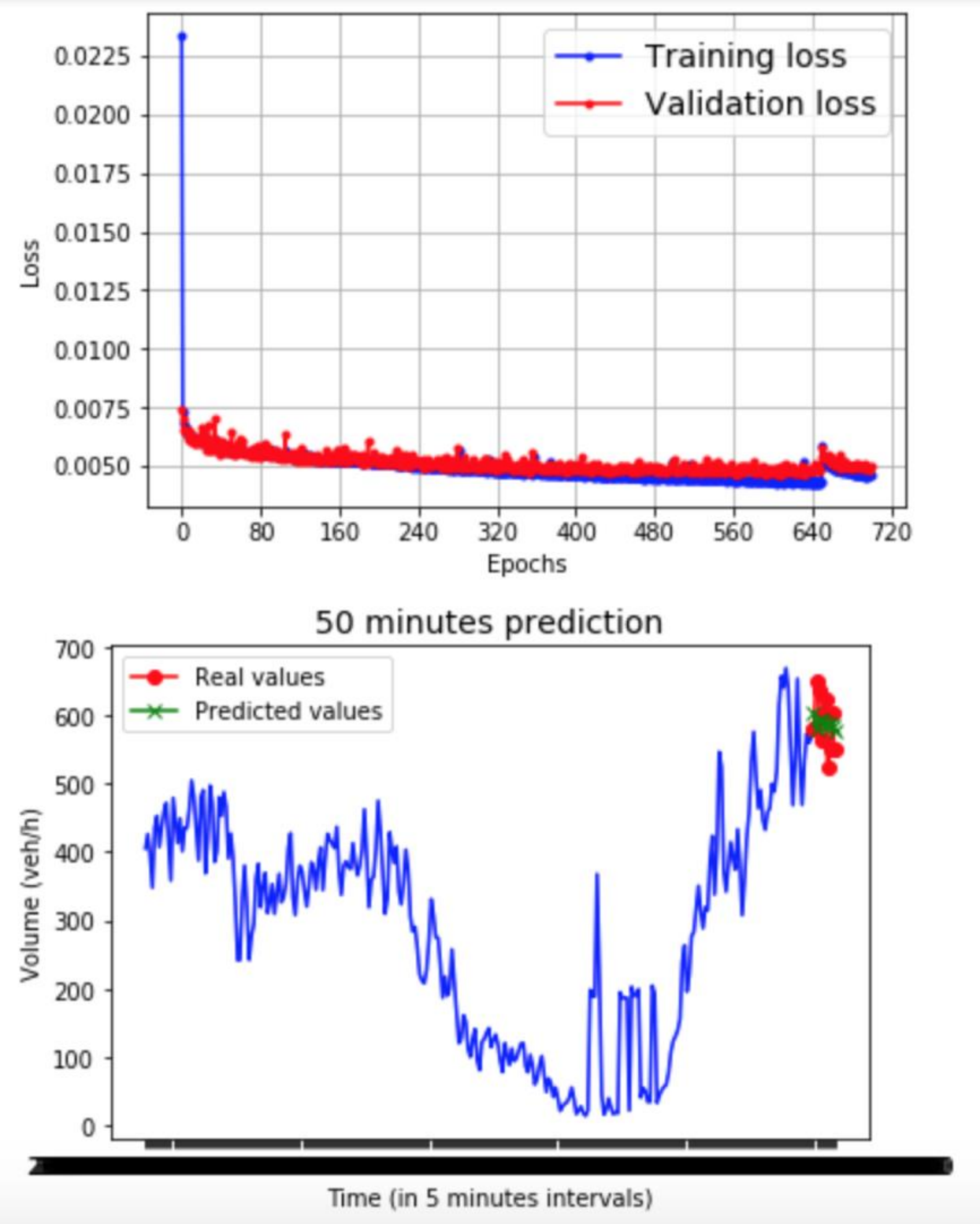


Figure 7.4: Learning curve and plot of real values agains prediction, RNN, loop detector 1014, southbound direction

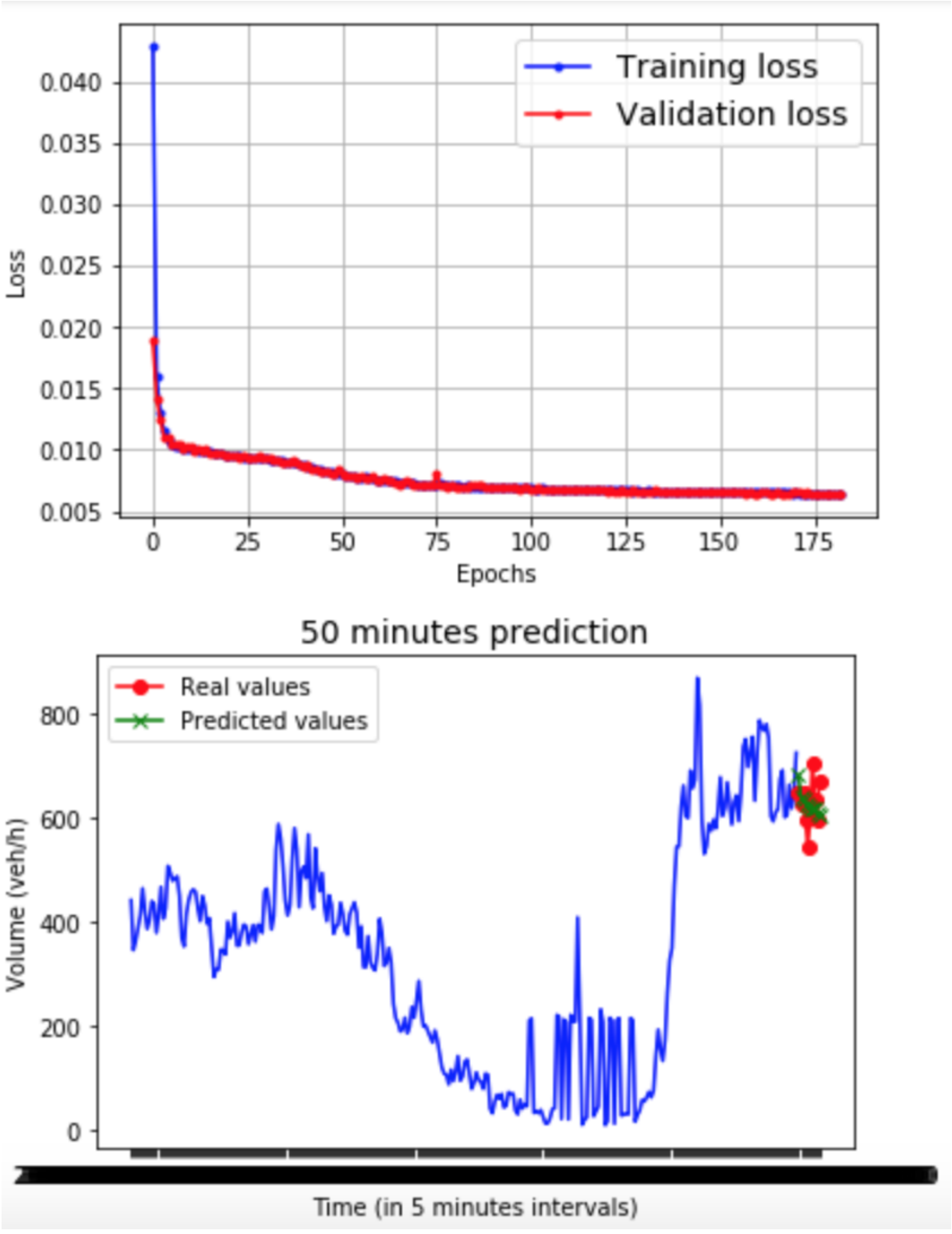


Figure 7.5: Learning curve and plot of real values agains prediction, LSTM, loop detector 1014, northbound direction

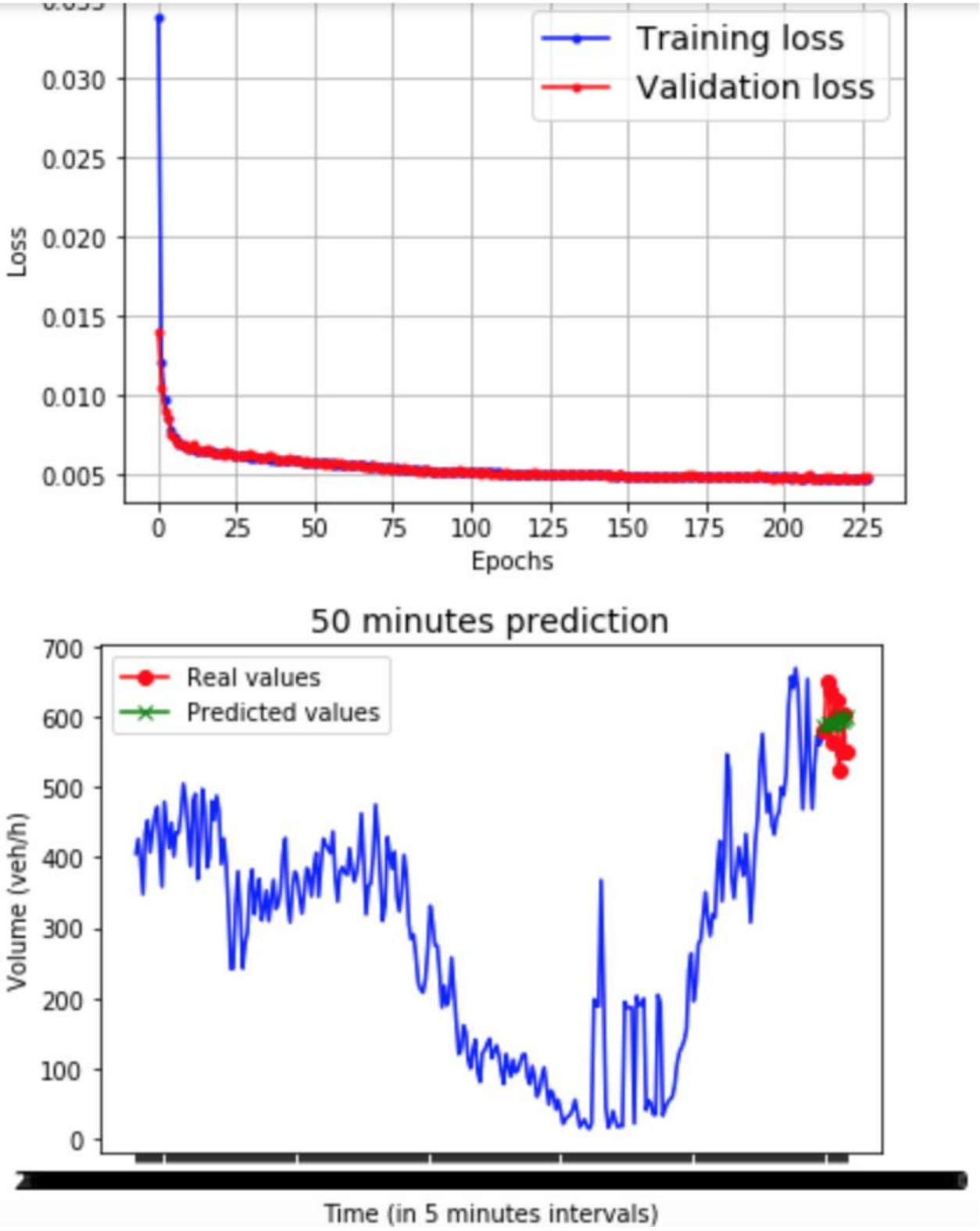


Figure 7.6: Learning curve and plot of real values agains prediction, LSTM, loop detector 1014, southbound direction

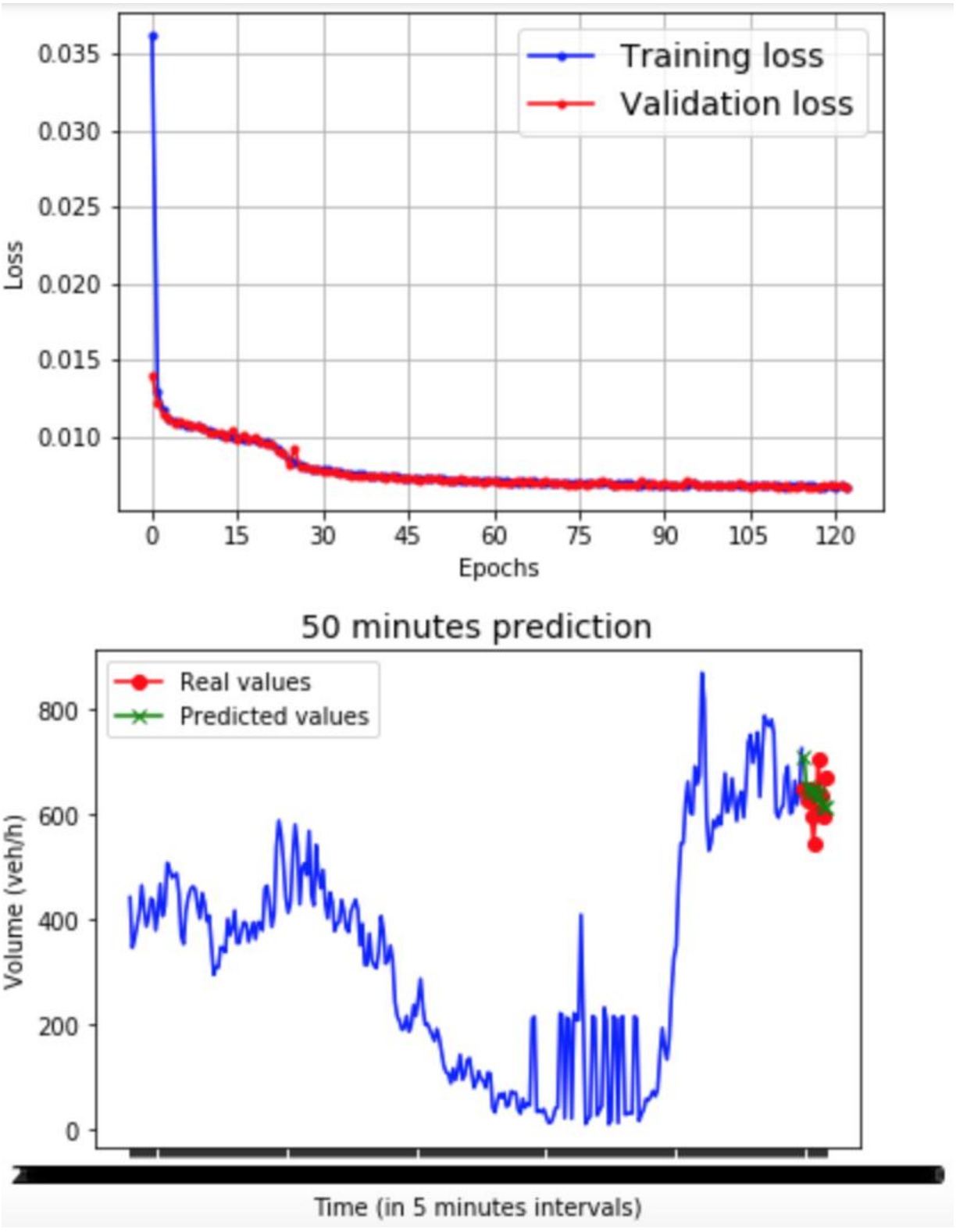


Figure 7.7: Learning curve and plot of real values agains prediction, GRU, loop detector 1014, northbound direction

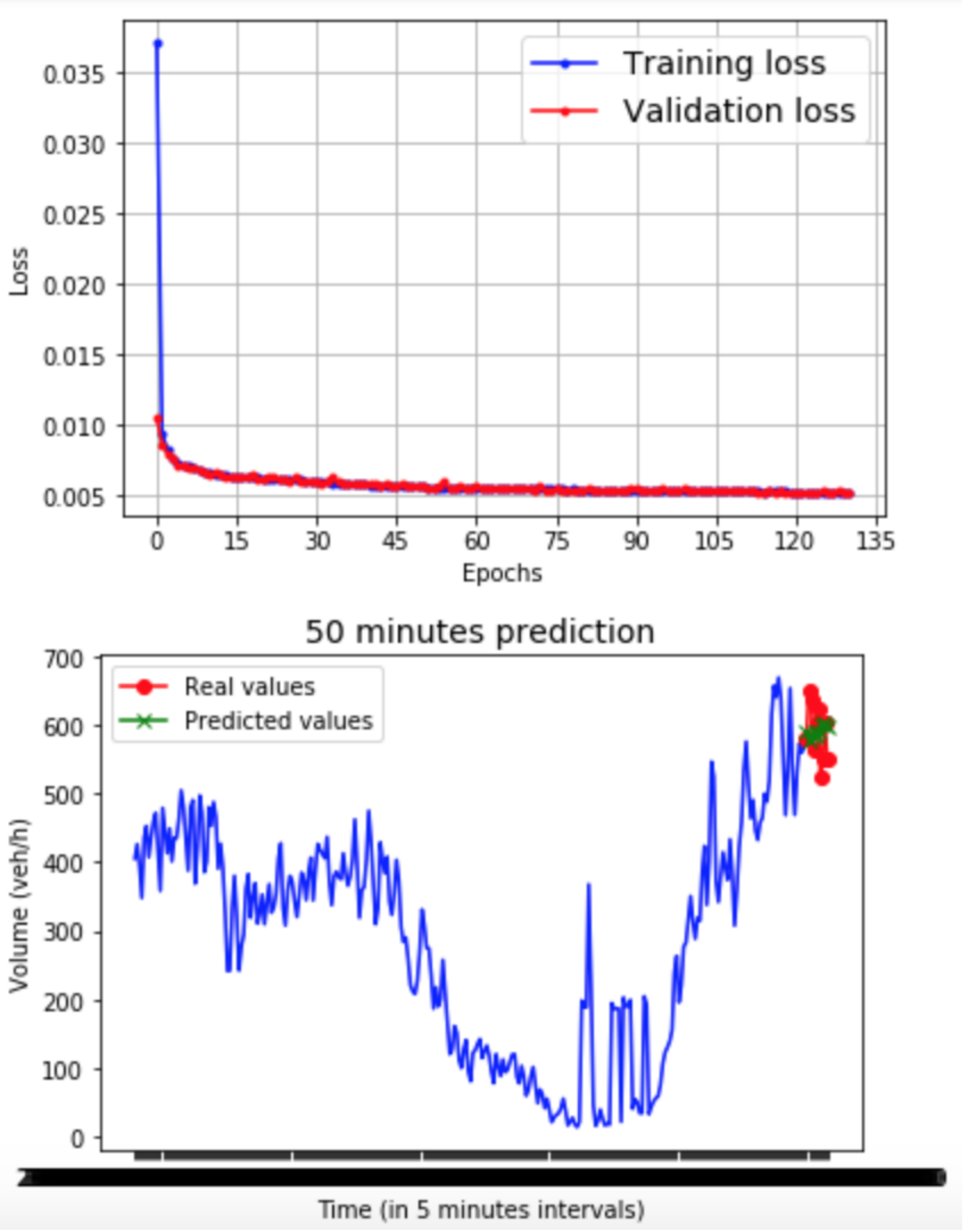


Figure 7.8: Learning curve and plot of real values agains prediction, GRU, loop detector 1014, southbound direction

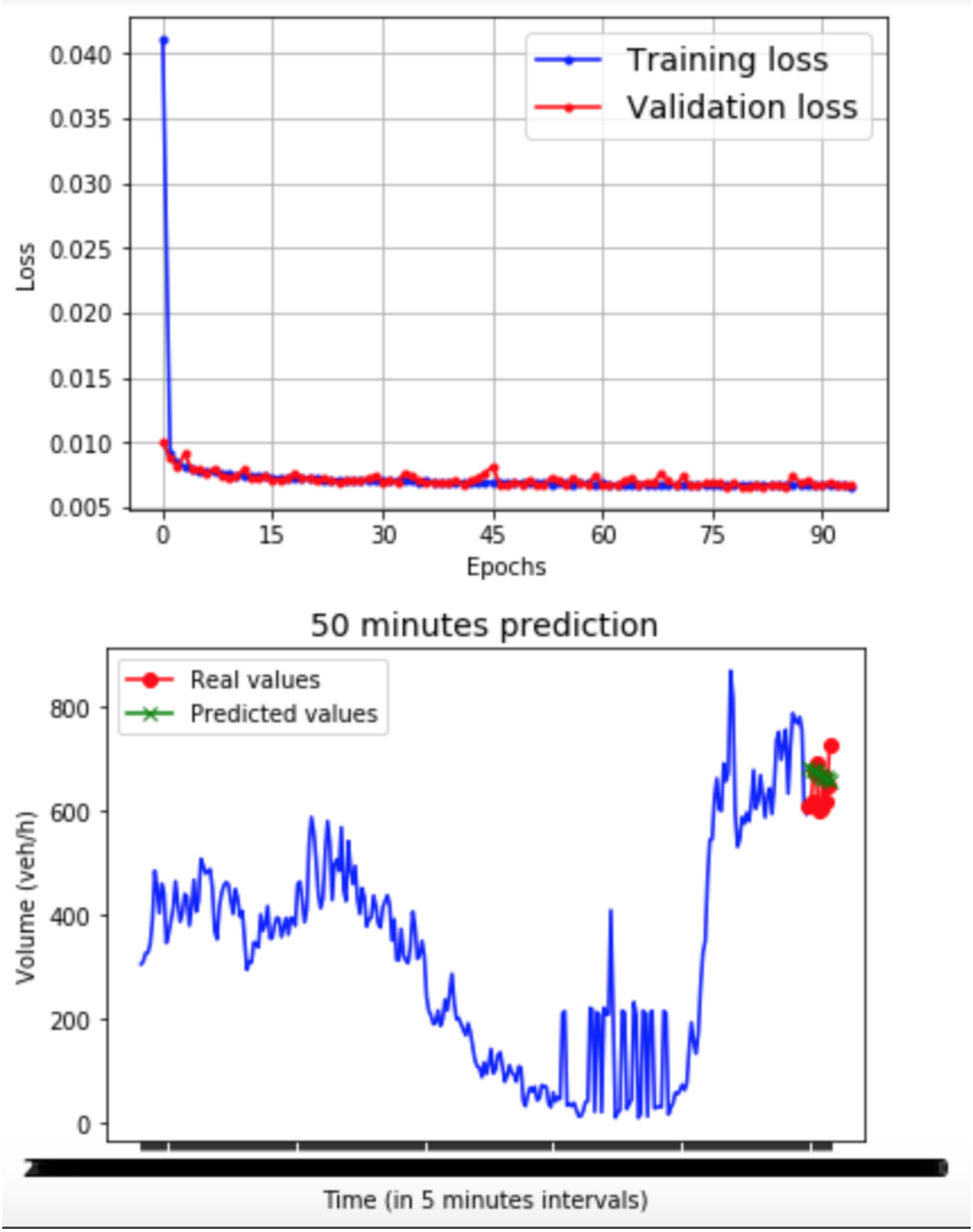


Figure 7.9: Learning curve and plot of real values agains prediction, WaveNet, loop detector 1014, northbound direction

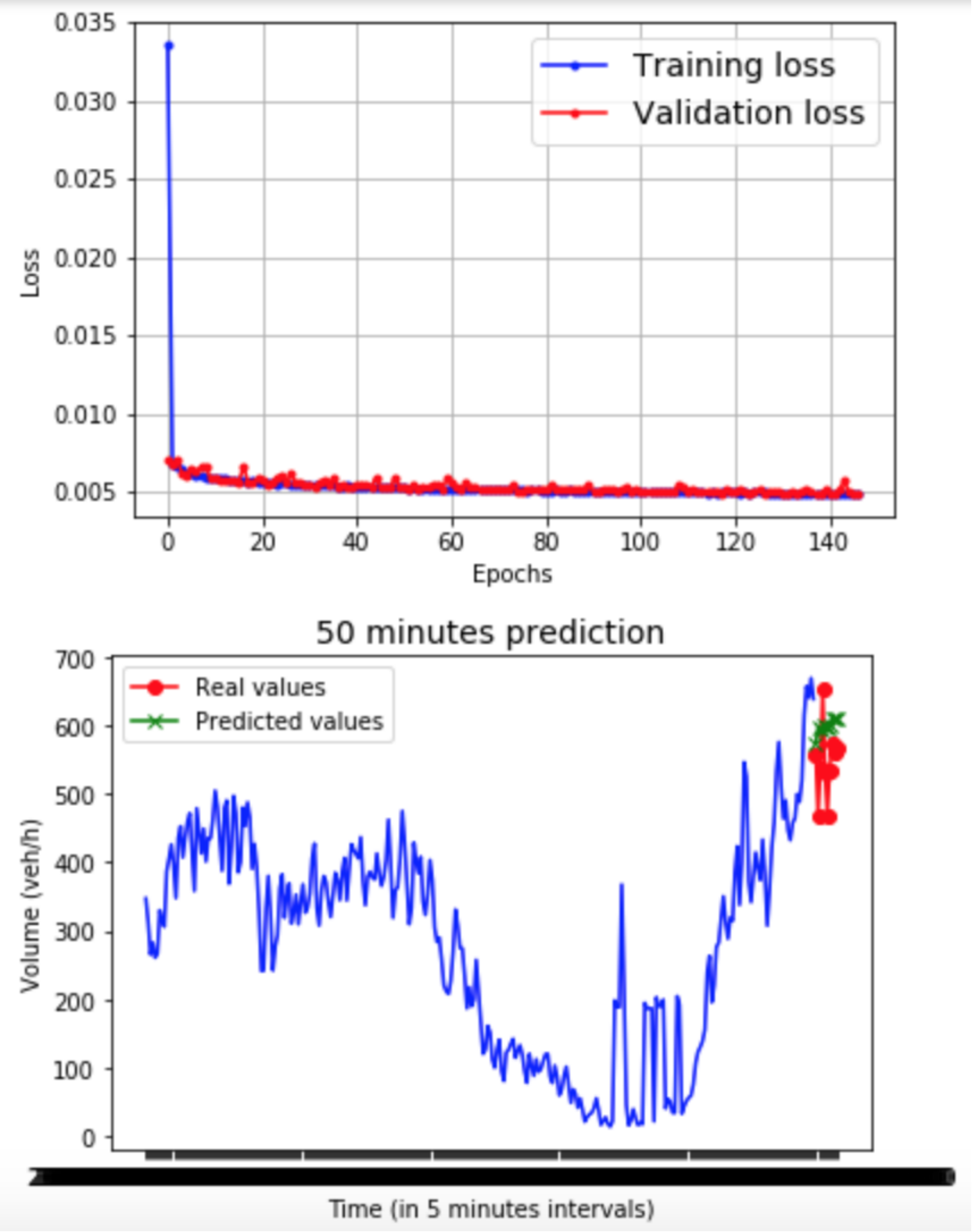


Figure 7.10: Learning curve and plot of real values agains prediction, WaveNet, loop detector 1014, southbound direction

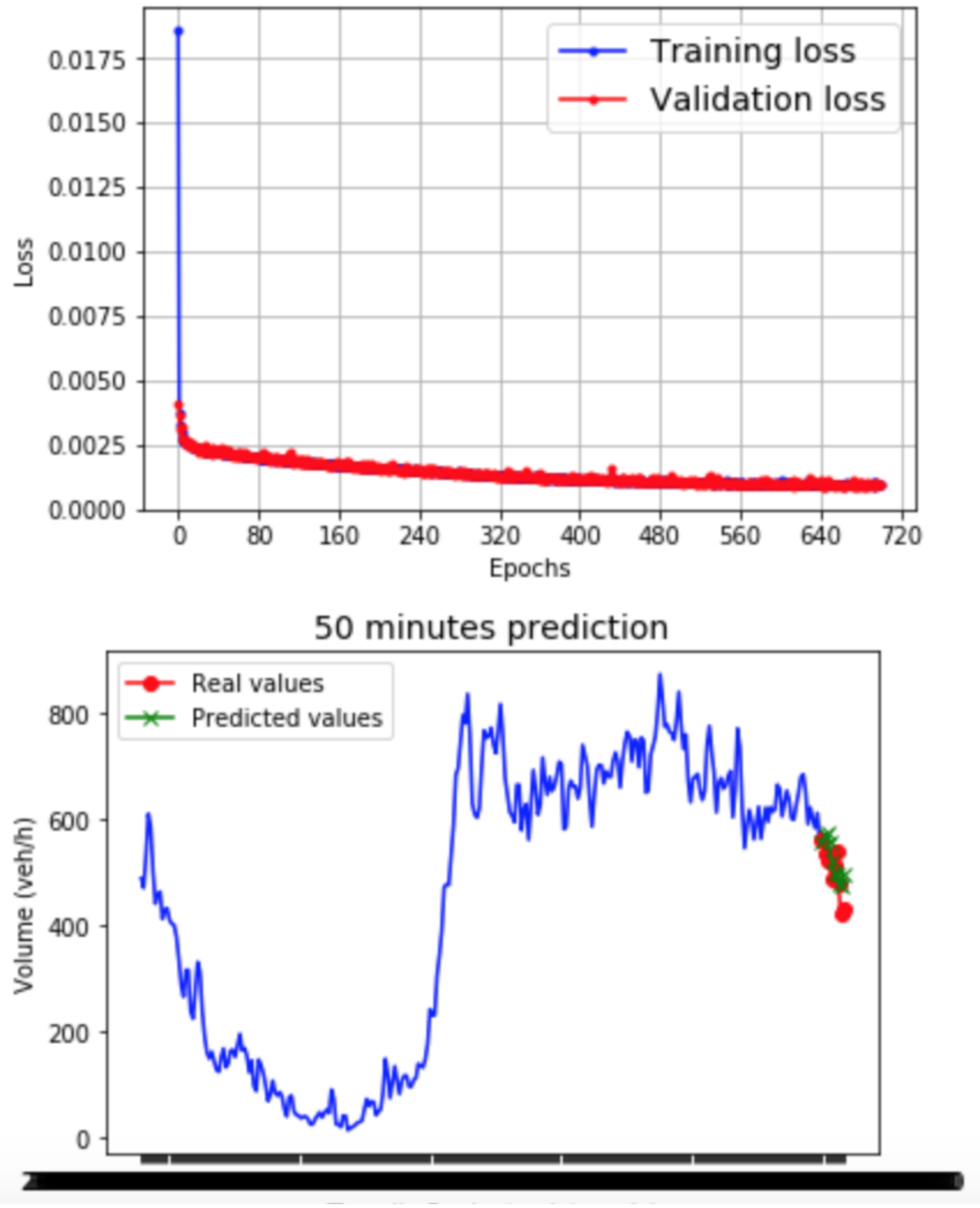


Figure 7.11: Learning curve and plot of real values agains prediction, RNN, loop detector 1015, westbound direction

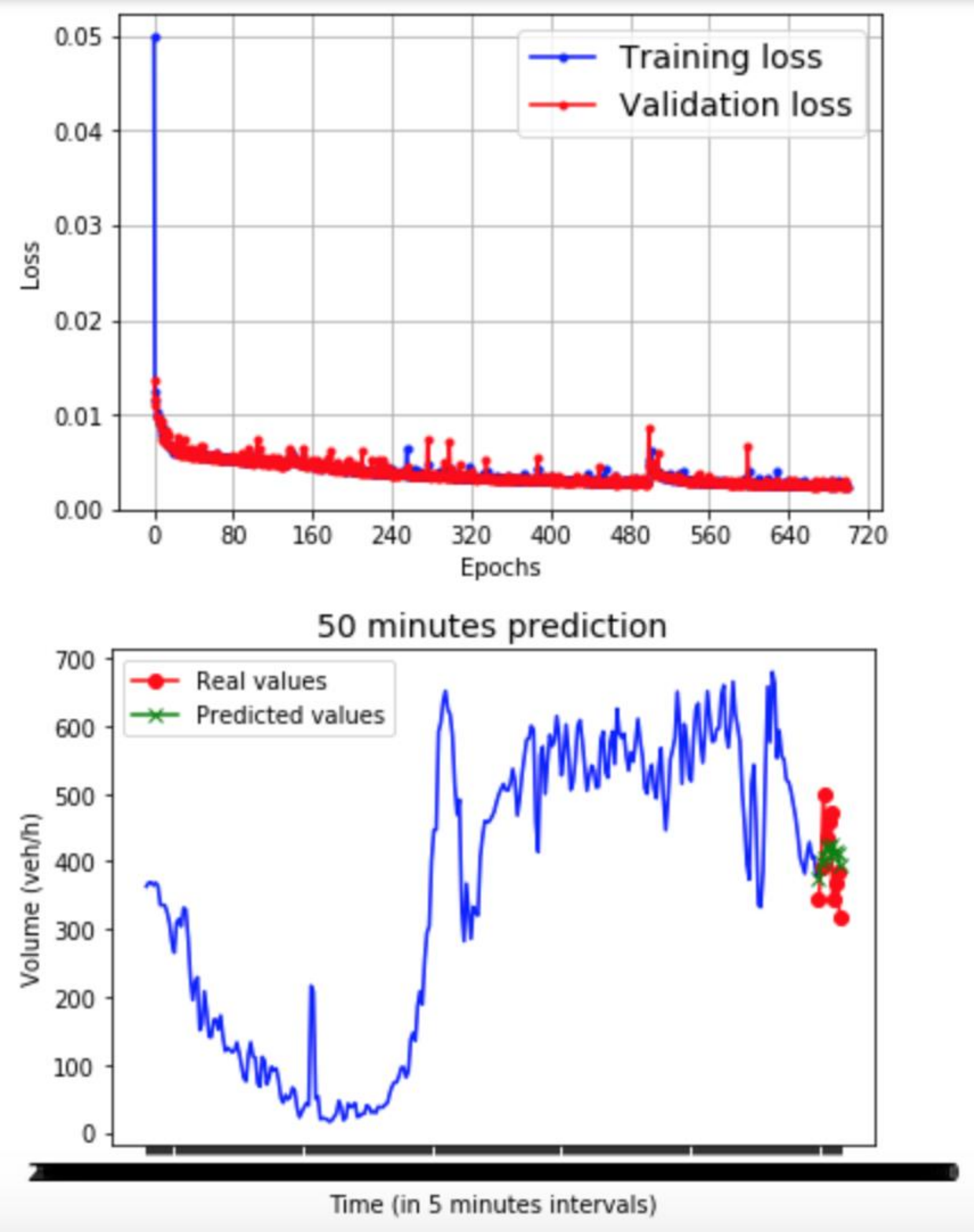


Figure 7.12: Learning curve and plot of real values agains prediction, RNN, loop detector 1015, eastbound direction

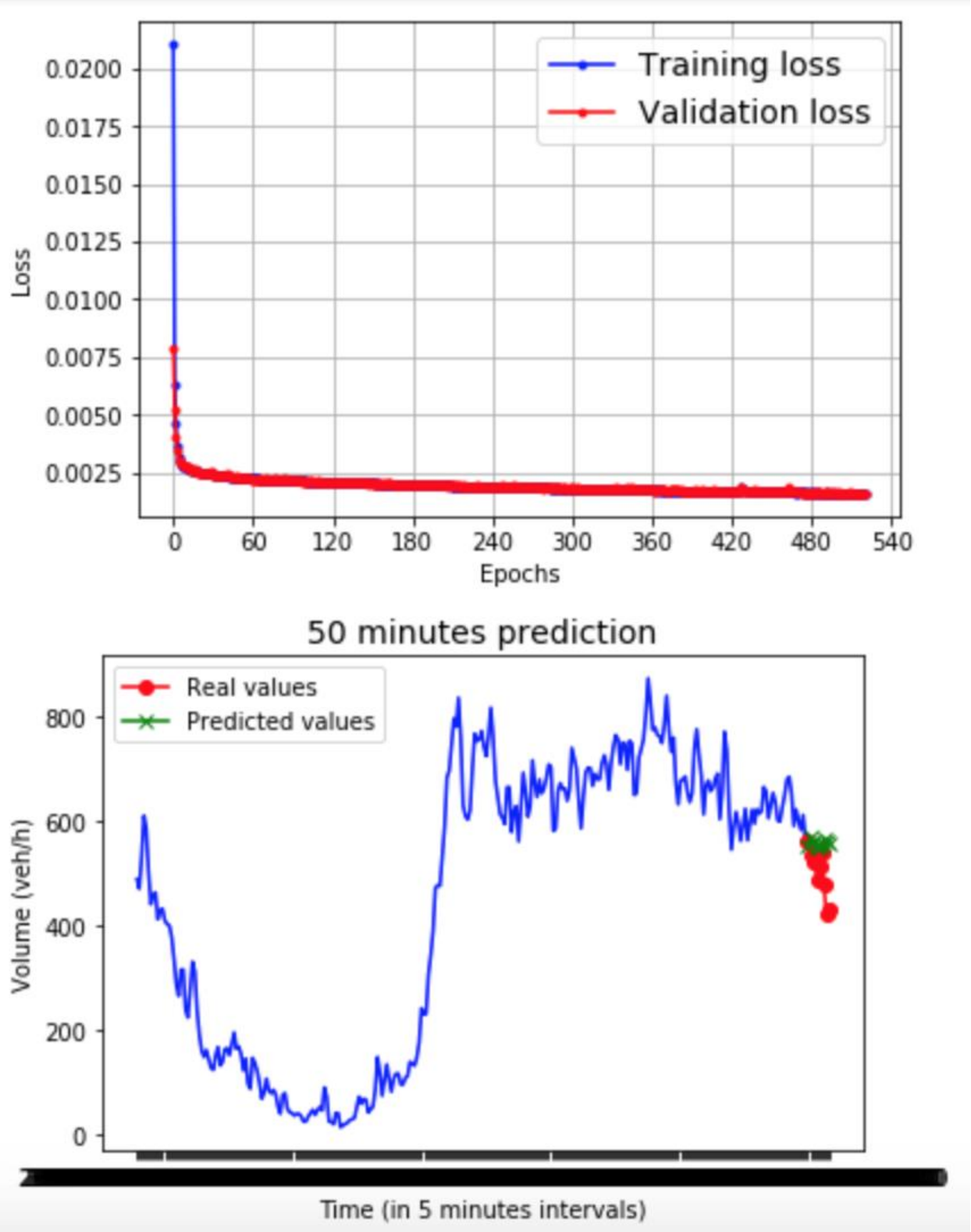


Figure 7.13: Learning curve and plot of real values agains prediction, LSTM, loop detector 1015, westbound direction

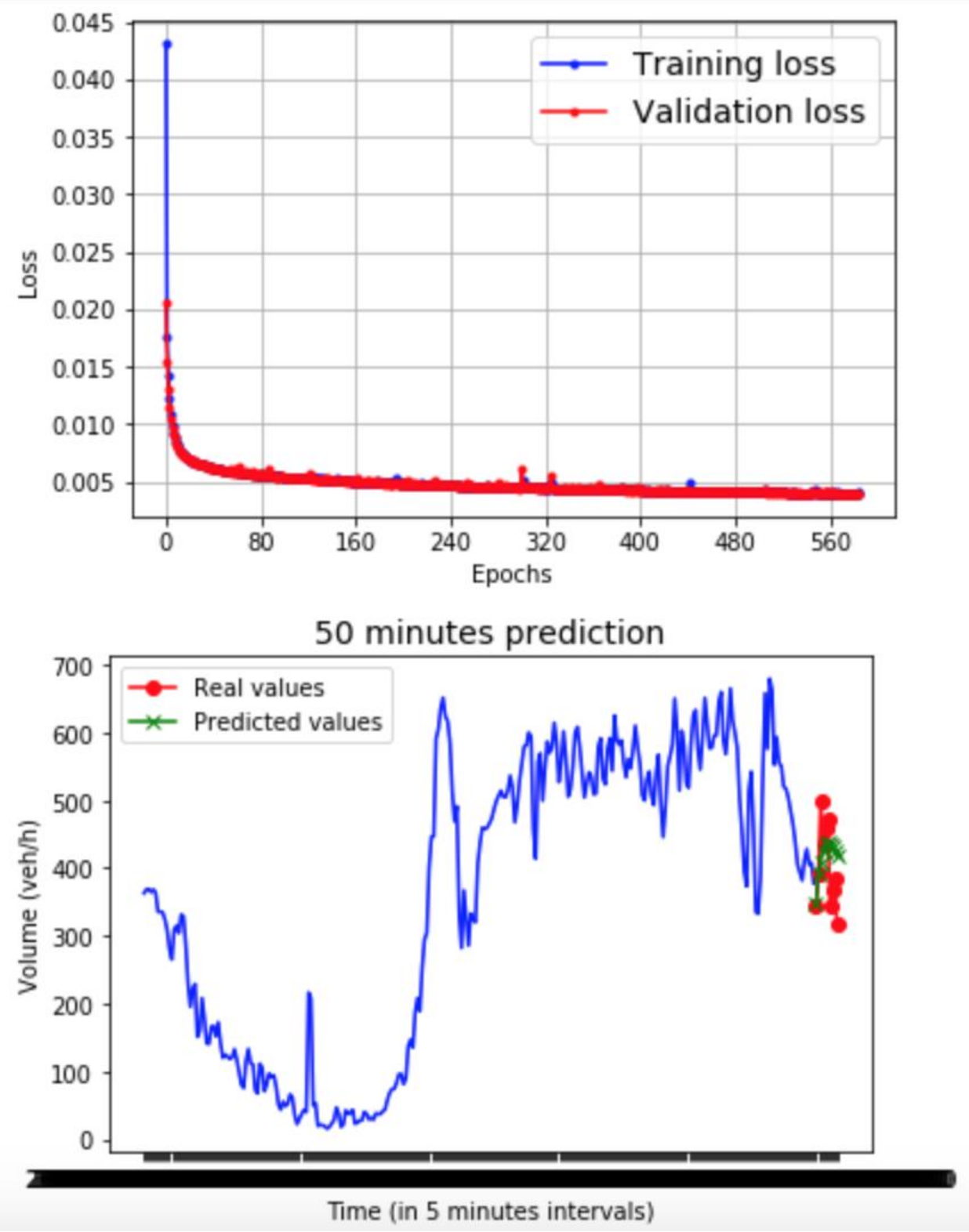


Figure 7.14: Learning curve and plot of real values agains prediction, LSTM, loop detector 1015, eastbound direction

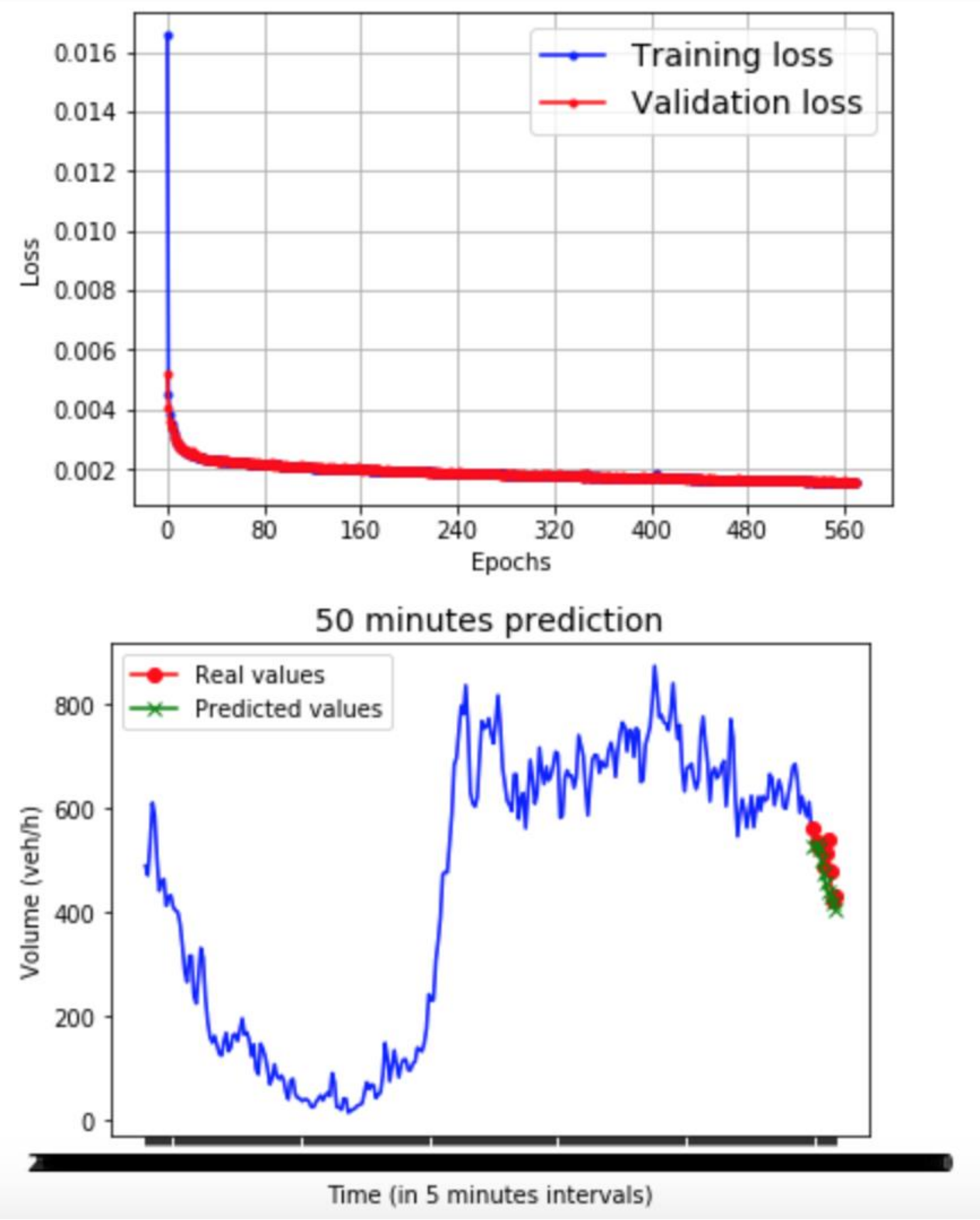


Figure 7.15: Learning curve and plot of real values agains prediction, GRU, loop detector 1015, westbound direction

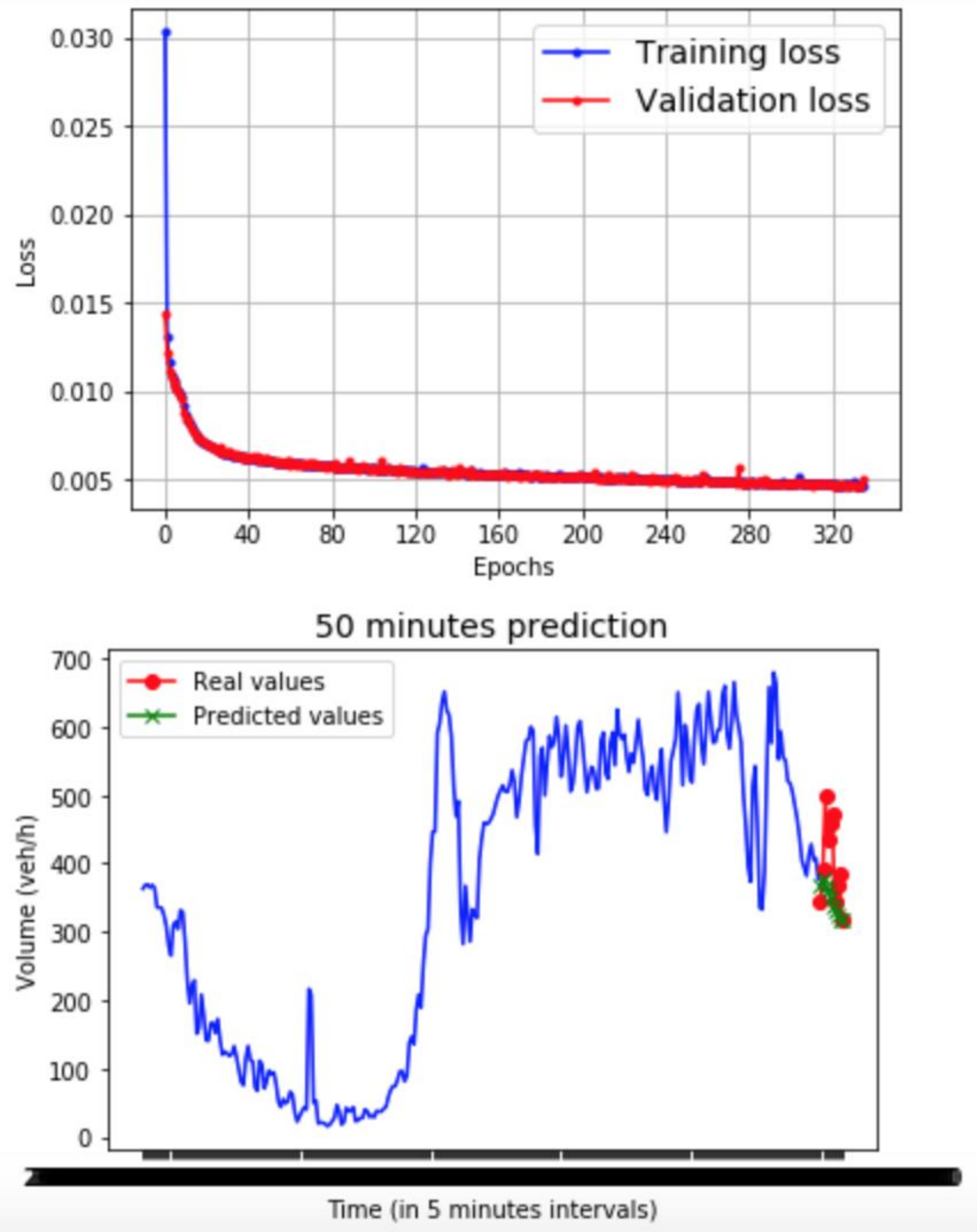


Figure 7.16: Learning curve and plot of real values agains prediction, GRU, loop detector 1015, eastbound direction

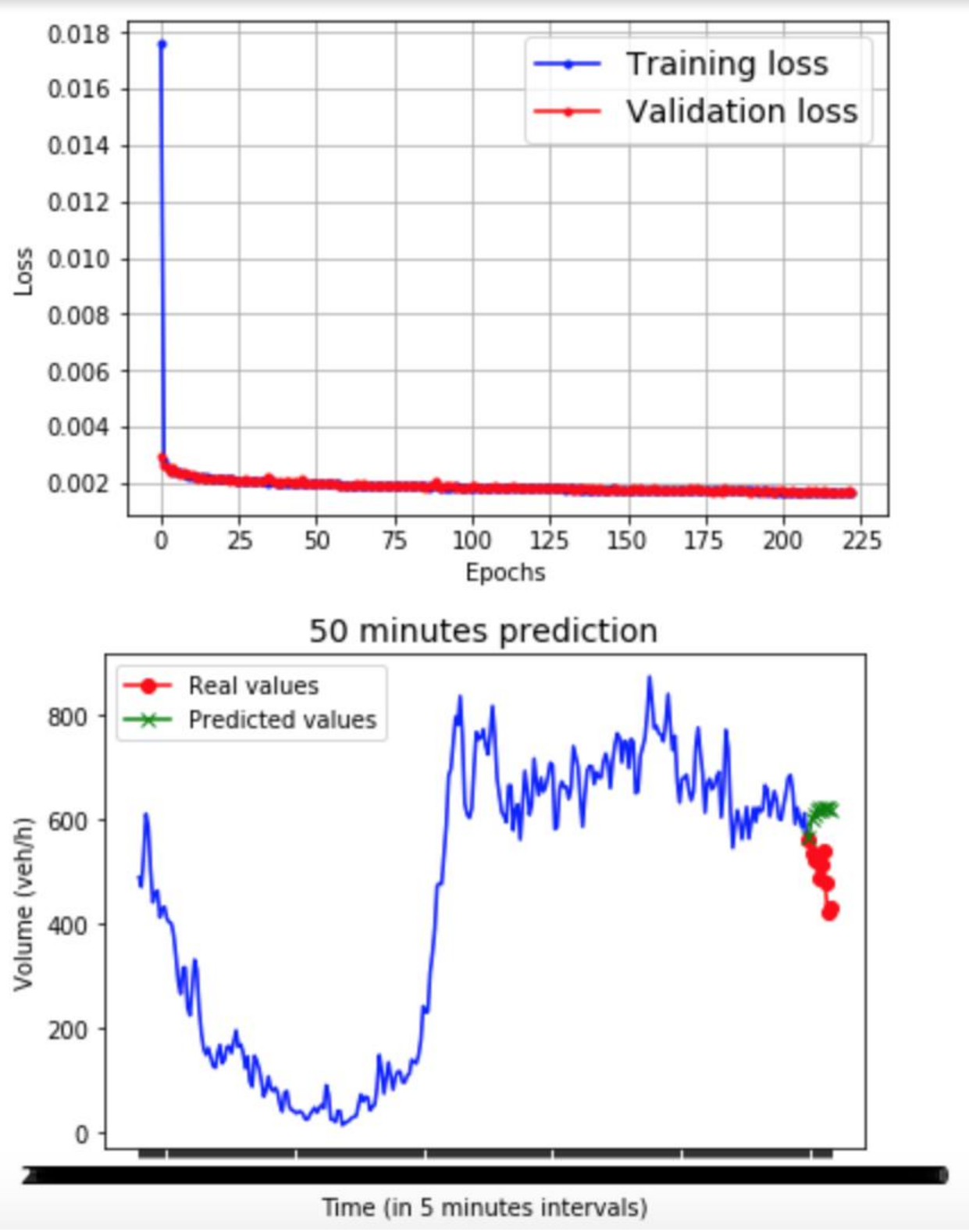


Figure 7.17: Learning curve and plot of real values agains prediction, WaveNet, loop detector 1015, westbound direction

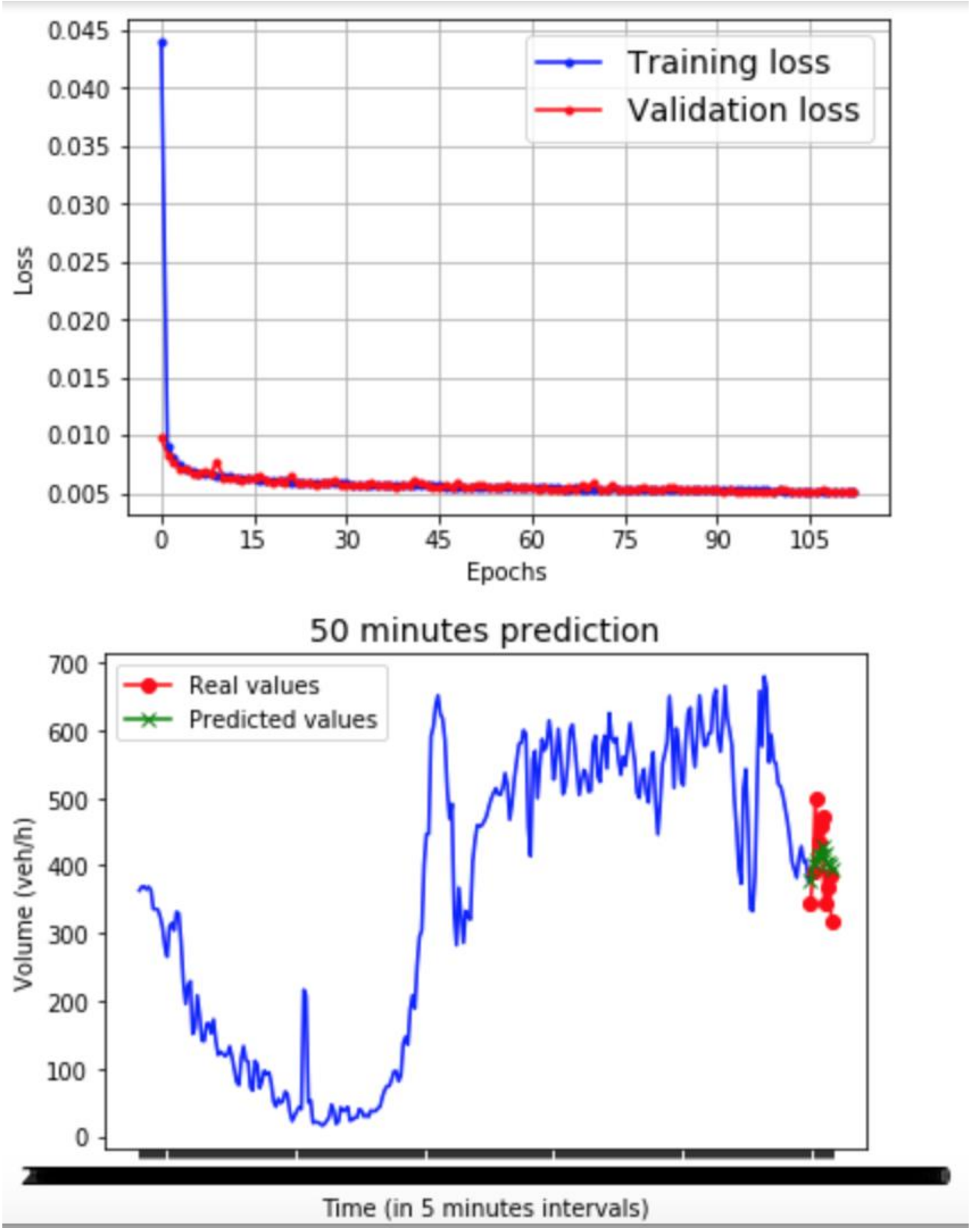


Figure 7.18: Learning curve and plot of real values agains prediction, WaveNet, loop detector 1015, eastbound direction