Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 220 (2023) 102–109

Procedia Computer Science

www.elsevier.com/locate/procedia

The 14th International Conference on Ambient Systems, Networks and Technologies (ANT)
March 15-17, 2023, Leuven, Belgium

Hourly forecasting of traffic flow rates using spatial temporal graph neural networks

Eline A. Belt[a,b,*], Thomas Koch[a,c], Elenna R. Dugundji[a,c]

[a] *CWI National Research Institute for Mathematics & Computer Science, 1098 XG Amsterdam, The Netherlands*
[b] *Vrije Universiteit Amsterdam, 1081 HV Amsterdam, The Netherlands*
[c] *Massachusetts Institute of Technology, Center for Transportation and Logistics, 1 Amherst St, Cambridge, MA 02142, USA*

Abstract

Traffic congestion forms a large problem in many major metropolitan regions around the world, leading to delays and societal costs. As people resume travel upon relaxation of COVID-19 restrictions and personal mobility returns to levels prior to the pandemic, policy makers need tools to understand new patterns in the daily transportation system. In this paper we use a Spatial Temporal Graph Neural Network (STGNN) to train data collected by 34 traffic sensors around Amsterdam, in order to forecast traffic flow rates on an hourly aggregation level for a quarter. Our results show that STGNN did not outperform a baseline seasonal naive model overall, however for sensors that are located closer to each other in the road network, the STGNN model did indeed perform better.

© 2023 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the Conference Program Chairs

Keywords: Highway Traffic; Traffic Flow Forecasting; Spatial Temporal Graph Neural Network; Seasonal Naive; Metropolitan Regional Congestion

In the past couple of decades the city of Amsterdam has grown rapidly. As a result, more people living in and around Amsterdam leads to more people that need to use the transportation network for daily activities. The number of people traveling on the road network is also influenced by different aspects such as the season, time of day, day of week and location. In this context, it is of great importance that accessibility in the region is maintained especially during the peak hour travel. Forecasting traffic flow rates can support intelligent management of transportation network.

The aim of this paper is to explore to what extent a Spatial Temporal Graph Neural Network (STGNN) framework can be used to predict the traffic flow rate in and around Amsterdam. In the following two sections, some related research will be discussed and the data that was provided for the metropolitan Amsterdam case study will be discussed. Subsequently, the methods used in this paper – the Seasonal Naive (baseline) model and the STGNN – will be

[*] Corresponding author. Tel.: +31 20 592 9333
E-mail address: e.a.belt@student.vu.nl

1877-0509 © 2023 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the Conference Program Chairs
10.1016/j.procs.2023.03.016

explained. Next the results of the two models will be compared and their results will be discussed. We conclude with a summary and provide recommendations for future research in this area.

1. Background

Previous research on the topic of forecasting traffic flow rates has been done using a wide range of techniques.

Lv et al. [6] proposed an early deep learning based traffic flow prediction method. They consider spatial and temporal correlations using a stacked auto encoder model which learns generic traffic flow features. The model is trained in a greedy layer wise fashion.

Qu et al. [7] used a deep neural network to forecast long term daily traffic flows using temporal correlation together with contextual factors such as day of week, weather, and season. To investigate the relationship of these factors, they trained a predictor using multi-layer supervised learning. A batch training method was proposed to reduce training times. A case study in Seattle shows that their method outperforms the conventional forecasting in terms of prediction accuracy.

Kang et al. [3] proposed a Long Short-Term Memory (LSTM) recurrent neural network to predict traffic flow based on inputs such as flow, speed and occupancy at each sensor. Furthermore they include the variables from sensors upstream and downstream to capture (partial) spatial effects. In this study they show that the inclusion of this spatial information improves prediction accuracy.

For a better model of spatial connections in traffic data, other studies incorporated a Graph Neural Network (GNN) into their network architecture. This is done to augment temporal structure in the data with spatial connections between different sensors that measure the data.

Much of the related research incorporates the GNN into their network as a new layer in its architecture, or modifies the structure of a existing network architecture to allow the GNN to be incorporated in the network. Cui et al. [2] proposed a Traffic Graph Convolutional LSTM recurrent neural network, where they keep the structure of the LSTM unit the same, and instead replace the input by graph convolutional features. Yu et al. [10] proposed a framework that consists of spatio-temporal blocks, where these blocks contain temporal gated convolution layers to model the temporal dependencies, together with a spatial graph convolutional layer that models the spatial dependencies of the data. Most recently Wang et al. [9] used a GNN as the first layer in a framework that they call a Spatial Temporal Graph Neural Network. All these papers achieve very good results on their traffic datasets with these methods that incorporate GNNs.

2. Case study

For this study, data from 48 traffic sensors in and around Amsterdam in the period from 2016 to 2020 have been provided. The relevant columns of the data are shown in table 1. The data contains measurements of the traffic flow rate (vehicle count) and the average speed per sensor location, per hour, per traffic lane and per vehicle category. This means we have very granular data. To simplify the data, we aggregate over all vehicle categories and traffic lanes. This results in data per location, per hour. An issue with the data however, is that there are two types of sensors: a so-called MONICA sensor, and a so-called MONIBAS sensor. MONIBAS sensors are built to complement any missing data from MONICA sensors. However, MONIBAS sensors measure multiple lanes, whereas MONICA sensors only measure one lane. Using the coordinates of the MONICA sensors, we merged the MONICA sensors that are at the same location to get measurements for multiple lanes as well. Here, we summed the traffic flow rates (vehicle counts) and averaged the speeds. Finally, we removed all sensors that only recorded data for a short period of time; less than one month. This resulted in a final dataset which contained 34 sensors, the locations of these sensors and their connections are shown in Fig. 1. These connections between sensors are present if the sensors are on the same side of the road and if they are direct neighbours of each other.

To create the training set, validation set and test set for the STGNN, the data concerning the year 2020 was used. This data was pre-processed so that it can be used in a STGNN. This was done by creating a new dataset where one entry consists of the features corresponding to a time window of size s and a target with the traffic flow rate for the next s' time periods, i.e., one entry contains the features corresponding to times $[t, ..., t+s]$ and traffic intensities of times $[t+s+1, ..., t+s+s']$. After this pre-processing step, 70% of this new dataset were used for training, 10% for

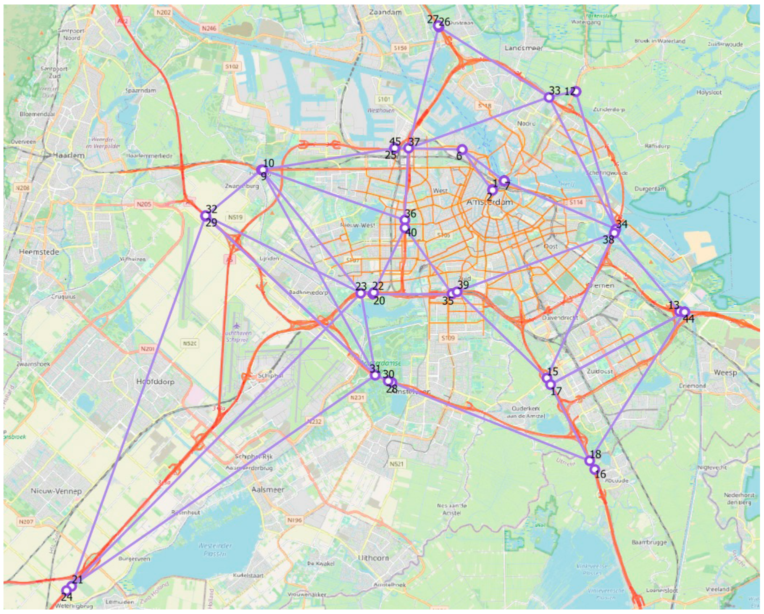

Fig. 1. Sensor locations and their connections

validation, and 20% for testing. To train the baseline model, the traffic flow rate in 2019 was used to predict the traffic flow rate in 2020.

Table 1. Column description of sensor data

Name	Description	Type
id_meetlocatie	Name of the sensor location	Text
start_meetperiode	Start of measurement	Datetime
eind_meetperiode	End of measurement	Datetime
waarnemingen_intensiteit	Number of observations of flow rate	Numeric
waarnemingen_snelheid	Number of observations of traffic speed	Numeric
gebruikte_minuten_intensiteit	Number of minutes in measurement period for traffic counts	Numeric
gebruikte_minuten_snelheid	Number of minutes in measurement period for traffic speeds	Numeric
data_error_snelheid	Number of errors in speed measurement	Boolean
data_error_intensiteit	Number of errors in flow rate measurement	Boolean
gem_intensiteit	Number of vehicles in class and on traffic lane	Numeric
gem_snelheid	Average speed of vehicles in class and on traffic lane	Numeric
totaal_aantal_rijstroken	Number of traffic lanes	Numeric
rijstrook_rijbaan	Number indicating the lane number	Numeric
voertuigcategorie	Vehicle category	Category
start_locatie_latitude	Latitude of sensor location	Numeric
start_locatie_longitude	Longitude of sensor location	Numeric

3. Methods

3.1. Seasonal Naive Model (Baseline)

The choice for the baseline model is the Seasonal Naive Model. It is a simple method that uses previous data to predict the future. For this model, the dataset is split into seasons. The Seasonal Naive Model looks at the value of data

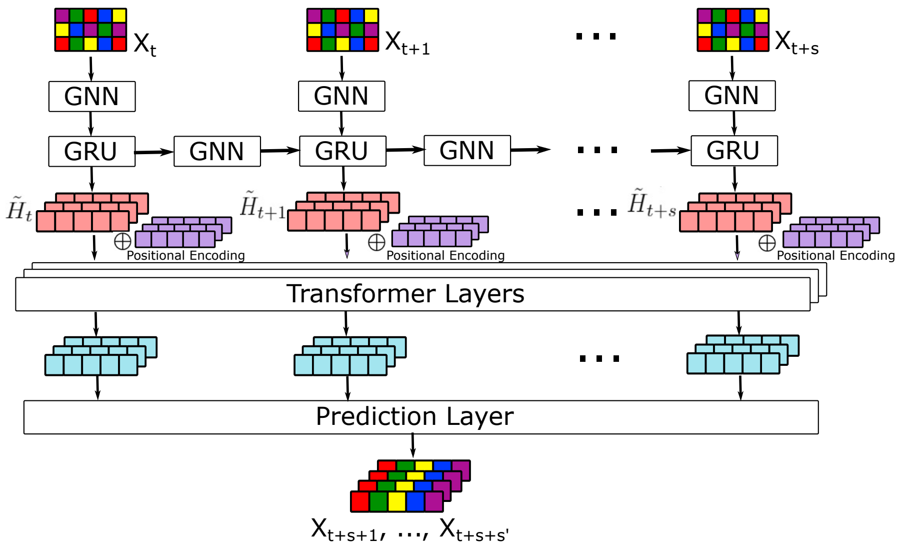

Fig. 2. Spatial Temporal Graph Neural Network framework, adapted from Wang et al. [9]

from the previous season to predict the value of the next season. An example could be to define a season as a week. This would mean that, in order to predict a value of next Friday, the value of the previous Friday is taken. For this project, the choice of season is a little bit more complicated because there are multiple seasonal trends. For example, not only a yearly trend is visible, but also a weekly trend, and a trend within a day. This makes it hard to define a season. To try to predict a future value of a specific sensor as accurately as possible, we use the data of that sensor of the year before, in the same month, weekday and hour. For example, if the value of Friday the 29th of January 2020 at 10:00 is to be predicted, then the average value of all the Fridays in January 2019 at 10:00 is used.

3.2. Spatial Temporal Graph Neural Network

The method that will be used to predict the traffic flow rate, while also taking the spatial dependencies between the sensors into account, is a Spatial Temporal Graph Neural Network [9]. This network has multiple layers that are made up of different types of neural networks. The first layer consists of multiple graph neural networks, one for each time step in the time window that is used as input. These GNNs will model the spatial dependency of the sensors. Then the output of these networks will be used as input for a recurrent neural network, which uses Gated Recurrent Units (GRUs) to model local temporal dependencies. The hidden layers of this GRU will then be used as input for transformer layers to model the global temporal dependencies. As a final step, the output of the transformer layers is used to make a prediction of the flow rate for a certain time interval, for all the sensors at the same time. The framework of this network is shown in Fig. 2 [9]. The rest of this section will describe the different layers of this STGNN in more detail.

3.3. Graph Neural Network

The GNN or Graph Convolutional Network (GCN) [4] models the spatial relations of the traffic network. For this, the traffic network can be represented by a directed graph $G = (V, E)$, where $V = \{v_1, ..., v_N\}$ is the set of nodes, which represent the N sensors, and $E = \{e_1, ..., e_M\}$ is the set of edges, which represent the roads that connect the sensors. From this graph G, an adjacency matrix $A \in \mathbb{R}^{N\times N}$ will be made. The elements of this adjacency matrix represent the geographical proximity of two sensors and it could be seen as a weighted adjacency matrix, with

$$A_{ij} = \begin{cases} \exp(-\frac{d_{ij}^2}{\sigma^2}), \text{ if } i \neq j \text{ and } \exp(-\frac{d_{ij}^2}{\sigma^2}) \geq \epsilon \\ 0, \quad \text{otherwise,} \end{cases} \tag{1}$$

where d_{ij} is the distance between sensor i and j, and σ^2 and ϵ are thresholds that control the sparsity of the adjacency matrix [10]. From this matrix A, the refined adjacency matrix $\tilde{A} = A + I_N$ can be created, with I_N the N-dimensional identity matrix [9], this adds self-loops to the graph G. From the refined adjacency matrix, the refined degree matrix can be determined and its elements can be calculated as follows: $\tilde{D} = \sum_j \tilde{A}_{ij}$. The final part of the data preparation is making the feature matrices, $X_t \in \mathbb{R}^{N \times d_{\text{in}}}$, for each time t (every hour) in the dataset, where d_{in} is the number of features which will be used in the model. The inputs of this GNN layer are the matrices $\tilde{A}, \tilde{D}$, and X_t, where X_t is the feature matrix, and the output of this layer is $X_{\text{out},t} \in \mathbb{R}^{N \times d_{\text{in}}}$, which is calculated for each time t as specified by Equation 2.

$$\text{GCN}(X_t) = X_{\text{out},t} = \text{ReLU}(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X_t W) \tag{2}$$

3.4. Gated Recurrent Unit

The Gated Recurrent Unit (GRU) [1],[9] is used to model the local temporal dependency. The GRU operation is applied to each input separately, but the parameters of the operation are shared for all the inputs of this layer. The operations of the GRU at time t are the following for each node v_i:

$$\begin{aligned} z_t &= \sigma_z(W_z\tilde{X}_{\text{out},t}[i,:] + U_z\tilde{H}_{t-1}[i,:] + b_z), \\ r_t &= \sigma_r(W_r\tilde{X}_{\text{out},t}[i,:] + U_r\tilde{H}_{t-1}[i,:] + b_r), \\ \hat{H}_t[i,:] &= \tanh\left(W_h\tilde{X}_{\text{out},t}[i,:] + U_h(r_t \odot U_h\tilde{H}_{t-1}[i,:]) + b_h\right), \\ H_t[i,:] &= (1 - z_t) \odot \tilde{H}_{t-1}[i,:] + z_t \odot \tilde{H}_t[i,;] \end{aligned} \tag{3}$$

where σ is the sigmoid function, $\odot$ is the element-wise multiplication, the matrices $W_z, W_r, W_h, U_z, U_r, U_h$ are the parameters to be learned, $\tilde{H}_{t-1} = \text{GCN}(H_{t-1})$, and $H_t[i,:]$ is the hidden representation of the current time step, which also serves as the output of the GRU layer.

3.5. Transformer Layer

To model the global temporal dependency, the transformer layer is used [8],[9]. This layer is also applied to each input seperately, just like for the GRU layer. The transformer layer is made up of a multi-head attention layer, a batch normalization layer, a shared feed-forward network, and another batch normalization layer. There is also a residual connection from the input of the transformer layer to the batch normalization layer, and from the output of the first batch normalization layer to the second batch normalization layer. The multi-head attention layer uses queries, keys, and values. The keys have dimension d_k, and the values have dimension d_v. Then the attention function is defined as

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}V\right) \tag{4}$$

where $Q, K \in \mathbb{R}^{T \times d_k}$ and $V \in \mathbb{R}^{T \times d_v}$ are the queries, keys, and values for all the nodes, respectively. Before the output of the GRU layer can be used in the transformer layer, it has to be transformed so that the sequences are no longer arranged by time, but by node. For this, the output of the GRU layer for each node v_i will be stacked row-wise, such that the matrix $H^{v_i} = (H_1[i,:], ..., H_T[i,:]) \in \mathbb{R}^{T \times d_{\text{in}}}$. To make sure that the transformer layer takes the relative position of each input into account, a positional encoding e_t will be added to matrices H^{v_i}, such that $H_t'^{v_i}[i,:] = H_t^{v_i}[i,:] + e_t$. This matrix H'^{v_i} will be used as input for the transformer layer. The positional encoding is defined as follows, with d_{model} being the number of dimensions used for the positional encoding:

$$e_t = \begin{cases} \sin(t/10000^{2i/d_{model}}), \text{ if } t = 0, 2, 4, ... \\ cos(t/10000^{2i/d_{model}}), \text{ otherwise.} \end{cases} \tag{5}$$

Then we define the queries, keys, and values for each node v_i as

$$Q^{v_i} = H'^{v_i}W^Q, \quad K^{v_i} = H'^{v_i}W^K, \quad V^{v_i} = H'^{v_i}W^V, \tag{6}$$

where $W^Q \in \mathbb{R}^{d \times d_k}$, $W^K \in \mathbb{R}^{d \times d_k}$, and $W^V \in \mathbb{R}^{d \times d_v}$ are matrices to be learned. The multi-head attention can then be defined as follows:

$$\begin{aligned} Multihead(H'^{v_i}) &= Concat(head_1, ..., head_S)W^O; \\ head_s &= Attention_s(H'^{v_i}W_s^Q, H'^{v_i}W_s^K, H'^{v_i}W_s^V) \end{aligned} \tag{7}$$

where W_s^Q, W_s^K, W_s^V, and W^O are matrices to be learned. The output of the transformer layer is $H_{\text{out}}^{v_i} \in \mathbb{R}^{T\times d}$

3.6. Prediction Layer

The output, $\{H_{\text{out}}^{v_i} | v_i \in V\}$, of the transformer layer is used as input for the prediction layer. This prediction layer uses a multi-layer feed-forward network to give the predictions of the traffic flow rate for future periods.

3.7. Implementation details

For the implementation of the STGNN, the features that were chosen to use as input (i.e. for matrix X_t), are the hour, weekday, and month of the start of the measuring period, the average flow rate (gem_intensiteit), and the average speed (gem_snelheid). Because the columns hour, weekday, and month are cyclical features represented as a number range, a transformation was performed that ensured that the first and last number in the range have a similar value. The transformation creates two new features out of one cyclical feature, and it is the following:

$$\begin{aligned} x_{\sin} &= \sin\left(\frac{2\pi x}{\max(x)}\right) \\ x_{\cos} &= \cos\left(\frac{2\pi x}{\max(x)}\right), \end{aligned} \tag{8}$$

where x represents the vector (column of a dataframe) that contains the cyclical feature. After this transformation, the features that will be used in the network are the transformation of the features hour, weekday, and month, and the average flow rate and average speed, so in total the model uses eight features. The number of sensors that were used as input is 34, which means the matrices $X_t \in \mathbb{R}^{34\times 8}$. The model will be trained using 24 hours as input to predict the next 7 days, which means it will predict the next 168 hours. To create the dataset that will be used for training, validation, and testing purposes, a sliding window of size 192 was used, for which the first 24 values will be used to determine the times for which a matrix X_t has to be added to the batch, and the last 168 values are used to add the targets y to the batch. This way, for each time t in our original dataset, a batch was created of inputs and targets. All the batches together form the dataset of which 70% will be used for training, 10% for validation, and 20% for testing.

4. Results

4.1. Seasonal Naive Model

To be able to compare the results of the seasonal naive model with those of the STGNN, the root mean square error (RMSE) of the predictions starting from 2020-10-20 00:00:00, and ending at 2020-12-31 23:00:00, is averaged over the different sensors. This results in a RMSE of 715 over all sensors.

4.2. Spatial Temporal Graph Neural Network

The network architecture was implemented with hidden dimensions of the network set to 64, the number of attention heads to 4, and the number of transformer layers set to 1. The hyperparameter tuning of the learning rate and weight decay was done using Ray Tune [5] and using the Adam optimizer. For both the learning and the weight decay, the specified range that Ray Tune could choose from was [1e-5, 1e-2]. The hyperparameters that gave the lowest RMSE during hyperparameter tuning were learning rate ≈ 0.0014, and weight decay ≈ 0.0004. The network was then trained using this learning rate and this weight decay, while using the standard Pytorch values for the other parameters. The network was trained for 20 epochs, for each epoch the mean and standard deviation of the RMSE were saved, and they are shown in Fig. 3. From this figure we can see that the validation RMSE is around 850 after 20 epochs, and the validation loss is around 1000. After the training was finished, the network was run on the test set, which resulted in a RMSE of 869 over all sensors.

5. Discussion

For most of the sensors, the RMSE of the spatial temporal graph neural network are not as low as the RMSE of the seasonal naive model. Some causes of that could be that it has not been trained for enough epochs, or that the

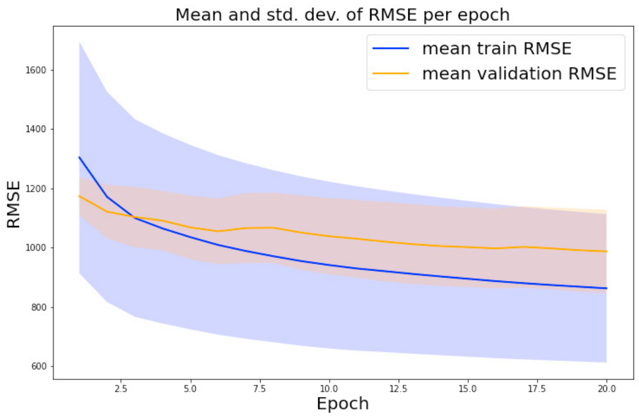

Fig. 3. RMSE of the training and validation set during training of the network

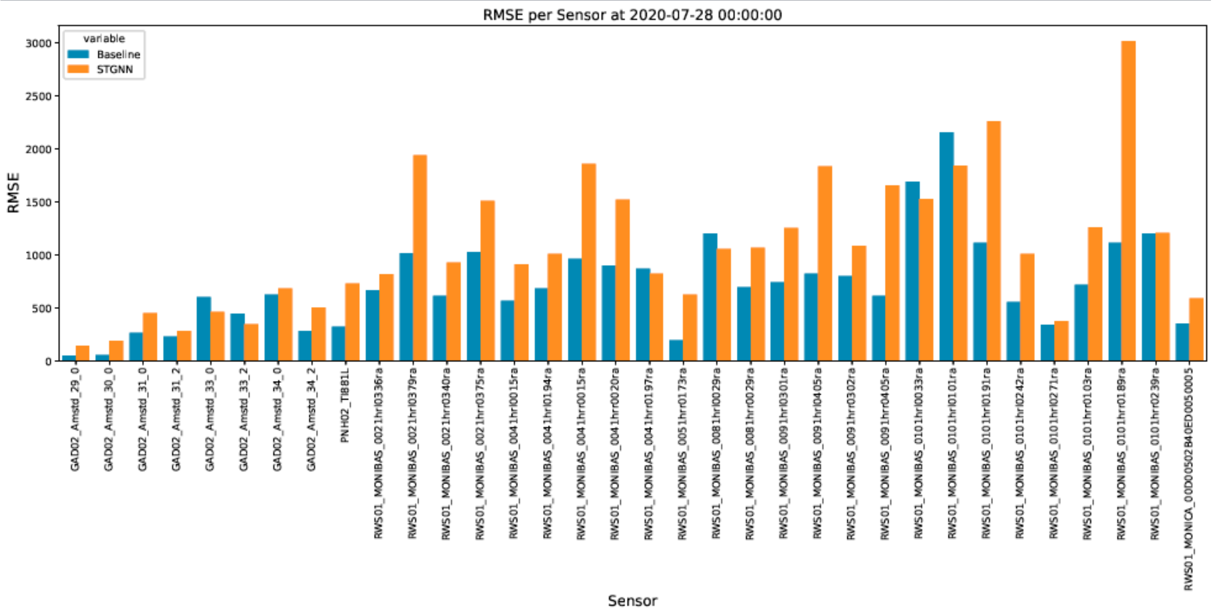

Fig. 4. RMSE of the baseline model and the STGNN on 2020-07-28 at midnight

dataset does not contain enough sensors for the network to have strong connections in the adjacency matrix, since a lot of sensors are very far apart. An indication that more sensors, and thus a more dense graph, could benefit this network is shown in Fig. 4. In this figure, the RMSE is shown for each sensor on a randomly chosen day, which is in this case 2020-07-28 at midnight. On the left side of this figure, for the sensors of the municipality of Amsterdam, the sensors starting with 'GAD', it can be seen that the STGNN has a similar RMSE to the baseline model, while for the other sensors, the STGNN has a RMSE that is a lot higher than that of the baseline model. This could be because the sensors of the municipality of Amsterdam are all close together in the center of Amsterdam and therefore have a strong connection in the adjacency matrix, while the other sensors are on highways and very far apart from each other and thus have a less strong connection in the adjacency matrix.

6. Conclusion

Related work indicated that incorporating a GNN in the network architecture showed very promising results in traffic prediction. Because of these results, the STGNN that was proposed in [9] was applied to a new dataset. The results of this framework on this new dataset were not as promising as those in the related literature. Comparing the results of the STGNN to a seasonal naive model showed that the STGNN did not outperform this simple model. However, for sensors that were closer together, like the ones in the center of Amsterdam, the STGNN seemed to perform more similar to the seasonal naive model, indicating that the framework might work better on a dataset which contains data from sensors that are located closer together.

7. Future study

Based on the results that were obtained in this paper, some interesting topics for future research are training the network for more epochs and/or including more training data. Also adding data from more sensors to the dataset would be useful to inspect if the STGNN model works better with a denser graph. Another interesting subject for future research could be to learn latent positional representations of the graph, which could also model some factors that influence the relationships between nodes that we are not aware of. This latent representation could be used to model pairwise relations between any two nodes and can be used instead of the refined adjacency matrix, as described in [9].

Acknowledgment

We would like to thank the Faculty of Science at the Vrije Universiteit Amsterdam for giving us the opportunity to initially work on the research during the Project Operations of Business Processes (POBP).

References

[1] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 .

[2] Cui, Z., Henrickson, K., Ke, R., Wang, Y., 2019. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. IEEE Transactions on Intelligent Transportation Systems 21, 4883–4894.

[3] Kang, D., Lv, Y., Chen, Y.y., 2017. Short-term traffic flow prediction with lstm recurrent neural network, in: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), IEEE. pp. 1–6.

[4] Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 .

[5] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J.E., Stoica, I., 2018. Tune: A research platform for distributed model selection and training. arXiv preprint arXiv:1807.05118 .

[6] Lv, Y., Duan, Y., Kang, W., Li, Z., Wang, F.Y., 2014. Traffic flow prediction with big data: a deep learning approach. IEEE Transactions on Intelligent Transportation Systems 16, 865–873.

[7] Qu, L., Li, W., Li, W., Ma, D., Wang, Y., 2019. Daily long-term traffic flow forecasting based on a deep neural network. Expert Systems with applications 121, 304–312.

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need. arXiv preprint arXiv:1706.03762 .

[9] Wang, X., Ma, Y., Wang, Y., Jin, W., Wang, X., Tang, J., Jia, C., Yu, J., 2020. Traffic flow prediction via spatial temporal graph neural network, in: Proceedings of The Web Conference 2020, pp. 1082–1092.

[10] Yu, B., Yin, H., Zhu, Z., 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv preprint arXiv:1709.04875 .