Forecasting Traffic with Graph Neural Networks

Olivier Filion
ofilion@andrew.cmu.edu

Michael Agaby
magaby@andrew.cmu.edu

Abstract

Sequential modelling and capturing graph dependencies using neural networks are two major areas of interest in modern deep learning. Accurate traffic speed prediction requires elements of both: traffic at a certain point in time depends on the traffic at the same spot at a previous time step, but is also correlated with the current and previous traffic of nearby roads, which can be modelled as a graph. In this paper, we predict traffic speed on the PeMS dataset using graph convolution and recurrent neural networks. We achieve a MAE of 2.74 on traffic speed predictions 15 minutes ahead, and 4.21 on traffic speeds 60 minutes ahead. We also discuss possible further improvements of the STGCN model on which we base our own.

1 Introduction

Recent advancements in deep learning have allowed for the tackling of various types of problems. One class of problems where such approaches have had success is in time series forecasting. Sequential problems arise in various fields such as natural language processing, financial trading, speech, video, and control theory. Traffic forecasting is one example of a sequential problem, where we would like to predict future traffic speed based off of past measurements. Since traffic is faced by almost all, predicting it is of major interest for governments and web mapping platforms. According to StatsCan, in 2016, Canadian commuters spent an average of 26 minutes travelling to work (Yaropud et al., 2019), and this number is likely to increase. Another survey by the UK's Office for National Statistics found that commuters have lower life satisfaction than non-commuters (Roxby, 2014), making the ability to accurately forecast traffic important from a societal perspective.

We approach this problem using graph convolution and recurrent neural networks. Graph neural networks are particularly well suited for this problem because of its spatial element (if traffic on a given road starts to slow down, it's likely neighbouring roads will also slow down). Recurrent neural network are also well suited because of the obvious temporal element of the problem (traffic at any point in time is heavily correlated with traffic five minute prior). The dataset that we work with is PeMSD7 (Chen et al., 2001). It consists of sensor data collected by Caltrans Performance Measurement System (PeMS) from District 7 of California, which has been curated and processed by Yu et al. (2018). The sensor network can be seen in figure 1. The dataset contains average traffic speed data for 228 stations for every 5 minutes on weekdays of May and June 2012. It also contains the distance between each of the stations. We forecast future average traffic speed given past speed, modelling it as a time series problem and incorporating spatial information using the distance between each sensor through graph convolution. The traffic network is defined as a graph, and the model predicts traffic over the graph. We evaluate the model using metrics such as mean absolute error (MAE), mean absolute percentage errror (MAPE) and root mean squared error (RMSE) and show that it performs well for the given task.

2 Background

In our midway report, we presented results from a simple LSTM encoder-decoder architecture. We present a diagram of that architecture in figure 2. We found that while this model has inferior

Preprint. Under review.

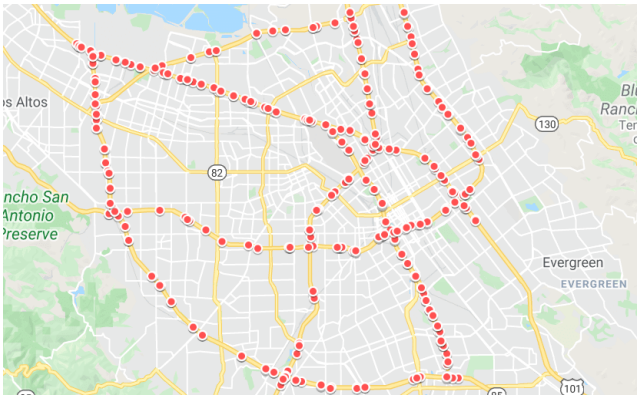


Figure 1: PeMS sensor network for district 7 of California, taken from Lu et al. (2020)

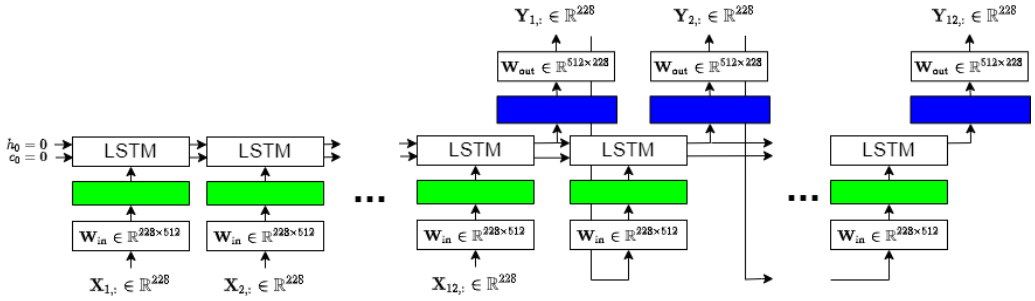


Figure 2: Diagram of the LSTM encoder-decoder architecture used in our midway report. Green and blue boxes represent batch normalization, dropout and ReLU activation.

performance compared to models that can incorporate spatial information like STGCN, the relative difference in error on long-term prediction compared to short term predictions was better. The LSTM encoder-decoder model had a test MAE of 4.17 on predictions 15 minutes away and 4.44 on predictions 60 minutes away (6.47% higher error). On the other hand, the ST-UNet architecture (Yu et al., 2019) had a test MAE of 2.15 on predictions 15 minutes away and 3.28 on predictions 60 minutes away (52.5% higher error). Our goal is to incorporate spatial information to our model to match the short term performance of state-of-the-art models while improving performance on longer term predictions.

3 Related work

3.1 Time-series forecasting

Several approaches to modelling time series have been suggested over the years. Early approaches were focused on autoregressive and moving average models. Box and Jenkins (1970) introduced the ARIMA (AutoRegressive Integrated Moving Average) model. ARIMA as well as its seasonal generalization SARIMA (Seasonal ARIMA) (Box and Jenkins, 1970) have been used extensively ever since. However, such approaches are limited by their stationary assumptions and inability to handle complex data (Yu et al., 2018).

In recent years, deep learning approaches have been applied to time series forecasting with great success. Recurrent neural networks (Rumelhart et al., 1986) introduced the concept of internal state (memory) to better deal with temporal sequences, allowing them to process variable length sequences. One of the limitations of early RNN's was their difficulty in learning long-term dependencies, in part due to the vanishing gradient problem (Pascanu et al., 2012). In 1997, a class of RNN's, the LSTM (Long Short Term Memory) (Hochreiter and Schmidhuber, 1997), were introduced to address some of the problems of early RNN's. In addition to a memory cell, LSTM cells also have special gates that allow them to learn what information from previous states to maintain and forget, as well as

what information to output. A simplified version of the LSTM, the GRU (Gated Recurrent Units) (Cho et al., 2014), was introduced in 2014, and has been shown to be approximately as good in most contexts (Chung et al., 2014). More recently, attention architectures such as the one proposed by Dzmitry Bahdanau (2014) have been used with great success in many sequence tasks. Attention allows the model to enhance certain aspects of the input, which in turn helps model long-term dependencies. Transformers, introduced by Vaswani et al. (2017) , eliminate the need for recurrent models by using multi-headed self attention layers.

3.2 Traffic forecasting

Due to the popularity of ARIMA based models, several researchers have used them to predict traffic with some success (Chen et al., 2018). There are certain difficulties that arise when predicting traffic speed. According to Yu et al. (2018) and Ge et al. (2020), the nonlinearity and complexity of the problem makes it difficult for traditional methods to predict accurately in the medium to long term. Kang et al. (2017) propose an LSTM model to predict traffic flow using flow, speed, and occupancy as inputs. They also include traffic from upstream and downstream as inputs to the model in an attempt to include spatio-temporal information and show that it improves accuracy. In order to fully utilize spatial features, some researchers have suggested using a convolutional neural network (CNN) (Du et al., 2018). Wu and Tan (2016) combine an LSTM with a 1-d CNN to predict short-term traffic. To fully utilize spatial information, Yu et al. (2018) model traffic networks as a graph and use a fully convolutional structure on the time axis. They also propose a new deep learning architecture called the spatio-temporal graph convolutional network to predict traffic flow.

Spatio-temporal graph convolutional networks (STGCN) were first introduce by Yu et al. (2018) as a way to combine spatial and temporal data to improve on earlier models. STGCN are build from ST-Conv blocks that alternate between spatial layers (graph convolutions) and temporal layers (gated convolutions). Yu et al. (2019) then introduce spatio-temporal U-Networks (ST-UNet) as an improvement on STGCNs for traffic prediction, by using spatio-temporal pooling and unpooling operations in a U-shaped network.

A limitation of models like STGCN is that their output layer is simply a linear layer that is only trained to predict the next step in the time series. This means that to obtain long-term forecasting, we must go pass the input through the whole model multiple times. Given that these models can get quite deep, this can be time consuming. In this project, we explore ways to improve the output operation so the model can be trained to predict several steps at a time, reducing the computation time required for long-term forecasting.

4 Methods

In this section, we briefly describe the different models that were used to tackle this problem.

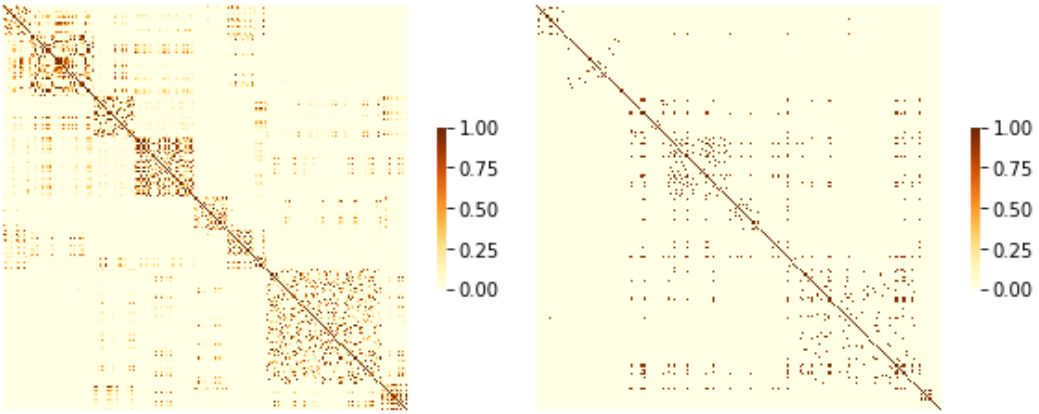
4.1 Data preprocessing

We start by doing a train/validation/test split where we use the last 2 weeks of data (10 days) for test, the 2 weeks before that (10 days) for validation and the rest of the data for training (24 days). We split the examples so none of them overlap the weekends (so the input and label sequences don't start on a Friday and end on a Monday).

In addition to the processing done by Yu et al. (2018), we standardize the data by subtracting the mean and dividing by the standard deviation before feeding it into the model. The data preprocessing step also requires building the graph. The dataset includes a list of distances between each sensor. We take that matrix of distances and calculate a weight matrix similarly to the preprocessing done by Yu et al. (2018),

$$w_{ij} = \exp\left(-\frac{d_{ij}}{\sigma^2}\right) \tag{1}$$

where d_{ij} is the distance between sensors i and j. Then, we build the graph by considering each sensor to be a node and adding an edge between two nodes i and j if $w_{ij} > \epsilon$. We use the same values for σ^2 and ϵ as Yu et al. (2018), which are $\sigma^2 = 10$ and $\epsilon = 0.5$. The average correlation coefficient between speeds of nodes with edges between them is 0.182 while the average for nodes without is



(a) Weight matrix

(b) Correlation between sensor traffic speeds

Figure 3: Comparing weight matrix and the correlation of the traffic speed for each pair of nodes. Correlations under 0.87 are set to 0 to distinguish between sets of extremely highly correlated nodes and others in the visualization.

0.0084. This justifies the method used for assigning edges, as the nodes with edges between them have on average much more correlation between their speeds than those without. Figure 3 shows a heatmap of the weights of the traffic edges as computed in equation 1 as well as the correlations between sensor traffic speeds. We observe that there are similar patterns in both; for example the large square of higher correlations on the bottom right corner.

4.2 Graph convolution

Graph convolutions allow neighbour nodes to share information. Each node receives information from all its neighbours, aggregates it, and updates its own representation. Kipf and Welling (2016) first introduced graph convolutional networks and their aggregation and update equations are (Yoon, 2022):

$$m_i = \frac{1}{|\mathcal{N}(i)| + 1} \sum_{i \in \mathcal{N}(i) \cup \{i\}} w_{ij} \mathbf{x}_j \tag{2}$$

$$\mathbf{x}_i \leftarrow \sigma(\mathbf{A} m_i) \tag{3}$$

where $\mathbf{A}$ is a learned linear transformation. Our first attempt on improving upon the basic LSTM encoder-decoder architecture is to add a graph convolution layer to the input of the encoder. We present the diagram of that model in figure 4. We add a skip connection (He et al., 2015) over the graph convolution to help training, as the model is fairly deep.

That model is trained by minimizing the mean-squared error of the output. We use Pytorch's (Paszke et al., 2019) implementation of the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 5×10^{-5}, a weight decay of 1×10^{-3}, a dropout rate of 50% and a batch size of 128. We ran the training loop for 65 epochs and kept the model with the best validation MAE on predictions 60 minutes away (epoch 51) to evaluate.

4.3 STGCN with linear output layer

The STGCN model proposed by Yu et al. (2018) is composed of two ST-Conv blocks and an output layer that consists of a temporal gated-convolution and a linear layer. Each ST-Conv block consists of two temporal layers, which in the original paper are taken to be temporal gated-convolutional layers, with a graph convolutional layer in between them.

Temporal gated-convolutions consist of a temporal convolution and a gated linear unit (GLU) to add non-linearity. We pass a sequence $\mathbf{x}$ with H_1 features per time step through the convolution and obtain a shorter sequence $\tilde{\mathbf{x}}$ (since there is no padding) with $2H_2$ features per time step. Then, the

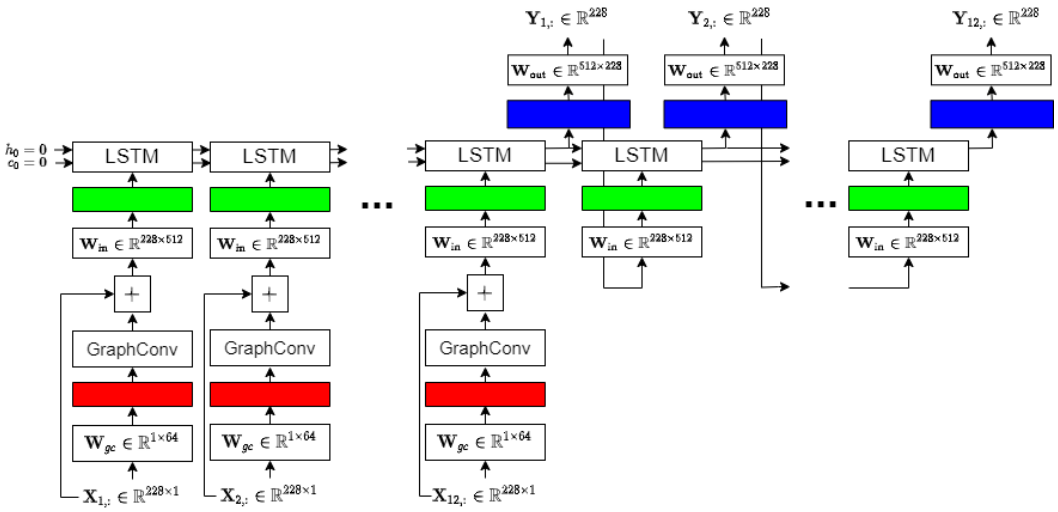


Figure 4: LSTM encoder-decoder with graph convolution. The hidden dimension of the LSTM cells is 512. The graph convolution output is dimension 1 for each node. The green, red, and blue boxes represent one layer of batch normalization, dropout and a ReLU activation.

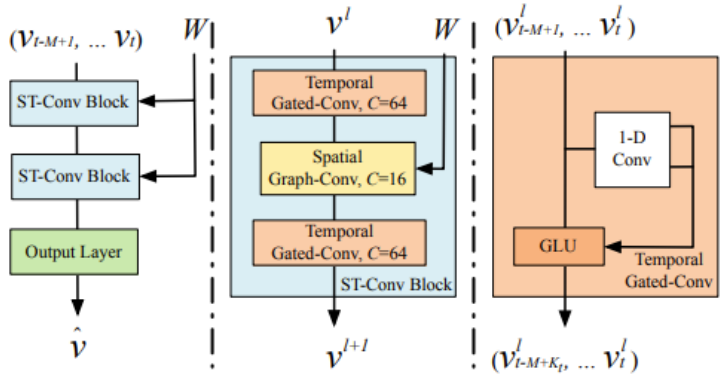


Figure 5: Diagram of the STGCN model, taken from Yu et al. (2018)

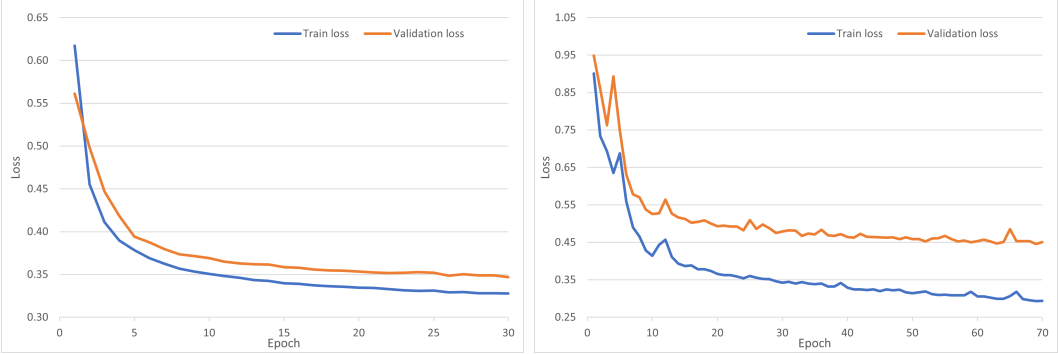
GLU works as follows:

$$\text{GLU}(\tilde{\mathbf{x}}) = \sigma(\tilde{\mathbf{x}}_2) \otimes (\tilde{\mathbf{x}}_2 + [\mathbf{x}, \mathbf{0}]) \tag{4}$$

where $\tilde{\mathbf{x}}_1$ is the first half of $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}_2$ is the second half. Our convolution uses a stride of 1 and a filter size of 3. An advantage of using a convolution instead of a LSTM is that it can be computed more efficiently as the operations are parallelizable. Also, for our specific task, it may work better since LSTM are designed to learn long term dependencies and convolutions focus on local dependencies. In traffic prediction, the speed 5 minutes ago is more important than the speed 40 minutes ago.

Additionally, we add layer normalization (Ba et al., 2016) within each block to help prevent overfitting. The inputs to each ST-Conv blocks are tensors of shape (B, N, T, F) where B is the batch size, N is the number of nodes, T is the number of timesteps, and F is the number of features. Note that F is taken to be 1 in the input to the first block (the only feature considered is the speed). The output is a tensor of shape (B, N, T, F') where F' is the number of output features. The output layer has input of shape (B, N, T, F) and outputs a tensor of shape (B, N, T), that is the predicted speed at each timestep for each sensor for the current batch.

Our first attempt at improving this architecture is to update the final linear layer in the output block to have an output of dimension equal to the output sequence length instead of 1, so we can train the model to output as many time steps in the future as required.



(a) STGCN with linear decoder

(b) STGCN with LSTM decoder

Figure 6: Loss progression of the two models based on the STGCN architecture during training

Model	MAE	MAPE (%)	RMSE
STGCN (Yu et al., 2018)	2.25 / 3.03 / 3.57 / —	5.26 / 7.33 / 8.69 / —	4.04 / 5.70 / 6.77 / —
ST-UNet (Yu et al., 2019)	2.15 / 2.81 / — / 3.38	5.06 / 6.79 / — / 8.33	4.03 / 5.42 / — / 6.68
LSTM encoder-decoder	4.17 / 4.21 / 4.30 / 4.44	10.2 / 10.3 / 10.6 / 10.9	6.78/ 6.96 / 7.16 / 7.39
LSTM + GraphConv	5.98 / 5.83 / 5.97 / 6.26	17.3 / 17.0 / 17.3 / 17.9	9.78 / 9.80 / 10.04 / 10.40
STGCN linear decoder	2.74 / 3.48 / 4.14 / 4.77	6.71 / 8.50 / 10.3 / 12.2	4.70 / 6.24 / 7.35 / 8.22
STGCN LSTM decoder	3.99 / 4.05 / 4.11 / 4.21	10.3 / 10.5 / 10.7 / 11.0	6.92 / 7.07 / 7.20 / 7.37

Table 1: Mean absolute error (MAE), mean absolute percentage error (MAPE) and root mean squared error (RMSE) for the models tested compared to state-of-the art architectures. All results are presented on predictions 15 / 30 / 45 / 60 minutes away.

We train this model with PyTorch's (Paszke et al., 2019) implementation of the Adam optimizer (Kingma and Ba, 2014) by minimizing the mean-squared error of the output. We use a learning rate of 1×10^{-3}, a batch size of 32, a weight decay of 1×10^{-4} and a dropout rate of 50%. The training loop is run for 30 epochs.

4.4 STGCN with LSTM decoder

In an attempt to improve the performance of the model on long-term predictions, we also tried replacing the linear layer in the output block by a LSTM decoder similar to the one in figures 2 and 4. We removed the batch normalization from the decoder for this model since in this case it was worsening the results. The model is trained in the same way as section 4.3, except we run it for 70 epochs to allow it to learn a more complex model.

Figure 6 presents the loss progression of the STGCN with linear decoder and STGCN with LSTM decoder during training.

4.5 Other methods explored

In addition to the methods listed so far, we also tried using attention based-models for this problem, as they have been very successful in other fields. In our first attempt, we placed a multi-headed self attention layer before the GLU layer in the temporal block, but found that it did not produce nearly as good results. We also tried replacing the GLU unit entirely by a transformer encoder, but similarly found that it did not achieve good results. The advantage of attention-based models is their ability to model long-term dependencies, however, this may not be very useful for traffic prediction, as the current traffic speed depends mostly on the speed at the previous time step (in addition to the speed of its neighbours). We did not have much success with these methods and will not be presenting results related to them.

5 Results

We present our results in table 1 where we compare the performance of our models to previous papers using MAE, RMSE and MAPE on predictions 15 / 30 / 45 / 60 minutes ahead. None of our models achieve better results than the STGCN by Yu et al. (2018) and the ST-UNet by Yu et al. (2019). As mentioned in our midway report, the LSTM encoder-decoder architecture achieves performance that doesn't deteriorate a lot for longer-term predictions but is significantly inferior to the state-of-the-art architectures. Adding the graph convolution layers to that model did not give the results we were expecting, as the model failed to learn how to take advantage of the spatial information and performed significantly worse than the normal basic LSTM encoder-decoder.

The two models based on the STGCN architecture offered better performance. As expected, the STGCN with a linear layer output performs very well on short-term predictions (2.74 MAE for 15 minutes). However, the performance falls off a lot for the 60 minutes predictions, where it gets a 4.77 MAE which is worse than what the LSTM encoder-decoder gets. The STGCN architecture with the LSTM decoder performs better on 60 minutes predictions but is worse than the one with the linear layer decoder on short term predictions.

In figure 6 we present the loss progression for train and validation sets for our STGCN with linear decoder and LSTM decoder models. We note that the loss progression is significantly less smooth for the LSTM decoder model, which may indicate the need for a lower learning rate in future experiments.

6 Discussion and analysis

6.1 Analyzing results

One surprising result was that the LSTM encoder-decoder baseline model performs significantly better than the LSTM+GraphConv model. As noted in the introduction, there is a strong spatial component to traffic prediction, since traffic speed at a given sensor will surely be highly correlated with traffic speed at nearby sensors. This intuition is supported by figure 3, where we observe that for nodes that have high weights (which represent sensors that are relatively nearby to one another), there tends to also be a higher correlation between traffic speed. As a result, we introduced the graph convolution layers to our encoder-decoder model in an attempt to capture this spatial information, but without much success. This could be due to the difficulty in training such a complicated model. We note that adding a skip-connection over the graph convolution layers did improve results slightly (the results we present are with this skip connection). The rationale for adding the skip connection is that the model starts to become fairly deep without it, which may lead to training issues such as vanishing gradient. Another reason for adding the skip connection, is that it gives the model the ability to completely ignore information from the graph convolution layer, as it can effectively skip over it (i.e. learn to set the weights to 0). Following this logic, we expected that in theory it should at least perform as well as the LSTM encoder-decoder and the fact that it doesn't suggest our training parameters need to be further tuned.

The STGCN and ST-UNet models do a much better job at incorporating spatial information using graph convolution as evidenced by their superior predictions. As noted earlier, the original STGCN model has an output sequence length of 1. As a result, in order to predict a certain amount if time steps ahead, the previous predictions must be passed through the network again, which can be quite inefficient given the complexity of the model. In order to improve on this, we tried updating the final linear layer in the output block to have an arbitrary output sequence length (taken to be 12). In table 1 we see that this model achieves good results, with a MAE of 2.74 and 4.14 for 15 and 45 minute away predictions, while the original STGCN achieves 2.25 and 3.57 respectively.

Another surprising result is that the STGCN with LSTM decoder performs significantly worse than the STGCN with linear output layer in short term predictions (3.99 MAE for 15 minutes vs 2.74), but slightly better on longer term ones (4.21 MAE for 60 minutes vs 4.77). We hypothesize that this could be due to the fact that a large amount of the loss during training is incurred from incorrect long-term predictions, and thus the model learns to perform relatively well on longer term predictions rather than shorter ones. Increasing the hidden dimension of the LSTM might be able to help alleviate this issue, as it would allow the model to store more information in its hidden and cell states which it will allow it to better model short and long term dependencies.

6.2 Limitations and future directions

Although figure 3 shows that there is a relationship between the weights that are computed (and thus the assignment of edges between nodes) and the correlation of traffic speed between pairs of nodes, there are also clearly pairs that have very strong correlations between them without having a large edge weight. A possible improvement on the method of calculating the weights, would be to incorporate those correlations, rather than only using the distance between sensors. There could be sensors that are in locations that might have highly correlated traffic for reasons other than distance. For example, two sensors that are relatively far apart, but that are on streets that intersect further down might have highly correlated traffic speeds. Or perhaps sensors that are both near business centers will have similar traffic speed patterns, for example high amounts of traffic during rush hour when workers are coming into or leaving work. Including this correlation information when building the graph might allow for the graph to more accurately model the traffic network.

Additionally, due to time constraints, we were limited in how much model and hyperparameter tuning that we could perform. As noted in table 1, the STGCN with linear decoder performs comparably with the original STGCN. It is possible that further hyperparameter tuning as well as more training could help close the gap, while maintaining the advantage of predicting multiple timesteps ahead.

As noted previously, the STGCN+LSTM model performs slightly better than the STGCN+linear decoder on long term predictions, but quite a bit worse on short term predictions. A possible improvement would be to combine the two and have a linear output layer which predicts a certain number of time steps ahead, followed by a LSTM decoder which predicts the rest. In this way, we might be able to combine the strength of the linear layer on short term predictions with the strength of the LSTM on longer term ones. Another direction for further work would be to look into even longer term predictions (e.g. 2 hours+) to see if the advantage of the STGCN+LSTM decoder continues to grow for even longer term predictions.

We would also like to continue exploring the use of attention models for this task. Finally, we also propose incorporating more recent GNN architectures into our model in order to improve performance.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

George E.P. Box and Gwilim M. Jenkins. 1970. *Time Series Analysis Forecasting and Control. 1e first*.

Chao Chen, Karl Petty, and Alexander Skarbadonis. 2001. Performance measurement system: Mining loop detector data. *Transportation Research Record*, 1748(1):96–102.

Jianbin Chen, Demin Li, Guanglin Zhang, and Xiaolu Zhang. 2018. Localized space-time autoregressive parameters estimation. *Applied Sciences*, 8(2).

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv:1409.1259*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*.

Shengdong Du, Tianrui Li, Xun Gong, and Shi-Jinn Horng. 2018. A hybrid method for traffic flow forecasting using multimodal deep learning. *arXiv:1803.02099*.

Yoshua Bengio Dzmitry Bahdanau, Kyunghyun Cho. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Liang Ge, Siyu Li, Yaqin Wang, Feng Chang, and Kunyan Wu. 2020. Global spatial-temporal graph convolutional network for urban traffic speed prediction. *Applied Sciences*, 10(4).

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Danqing Kang, Yisheng Lv, and Yuan-yuan Chen. 2017. Short-term traffic flow prediction with lstm recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Huakang Lu, Dongmin Huang, Youyi Song, Dazhi Jiang, Teng Zhou, and Jing Qin. 2020. St-trafficnet: A spatial-temporal deep learning network for traffic forecasting. *Electronics*, 9(9).

Razvan Pascanu, Thomas Mikolov, and Yoshua Bengio. 2012. On the difficulty of training recurrent neural networks. *arXiv:1211.5063*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Philippa Roxby. 2014. How does commuting affect wellbeing? [Online; accessed March 16, 2022].

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Yuankai Wu Wu and Huachun Tan. 2016. Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework. *arXiv:1612.01022*.

Tetyana Yaropud, Jason Gilmore, and Sébastien LaRochelle-Côté. 2019. Results from the 2016 census: Long commutes to work by car.

Minji Yoon. 2022. Introduction to graph neural networks. `https://minjiyoon.xyz/Paper/Minji_Yoon_intro_deep_graph_learning.pdf`. Accessed: 2022–04-27.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv:1709.04875*.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2019. St-unet: A spatio-temporal u-network for graph-structured time series modeling. *arXiv:1903.05631*.