Reproducing and Benchmarking Neural Network Architectures for Traffic Prediction

Luke Andreesen

University of Chicago

Abstract

In recent years, graph neural networks - systems that take graph structures as inputs - have become increasingly popular, and a wide array of architectures have been proposed. A popular application and benchmark for these networks is traffic forecasting, as it requires the network to handle a dynamic graph wherein node features change with time. This paper focuses on implementing the STGCN architecture proposed by (Yu et al., 2018).

Project Objective

The objective of this project is to reproduce the Spatio-Temporal Graph Convolutional Network architecture proposed by (Yu et al., 2018), and evaluate it on a traffic dataset. In addition to implementing the STGCN model, I sought to test it against other architectures on the same dataset - however, due to time constraints, I was only able to implement the single ST-GCN model. The goal of this project was broadly to learn about extracting features from and making predictions on graphs, particularly dynamic graphs, due to their applications in topics ranging from drug discovery to social network analysis.

Specifically, in this paper we seek to predict traffic speeds over a set of certain future time intervals given recent time interval data. We only consider traffic speed in this study, though other metrics of traffic flow including volume and density could utilized in future studies.

Methods

Data

In this study. we utilize the METR-LA dataset, which can be found here. The data set contains 34272 traffic speed observations for each of 207 traffic loop sensors, with each observation separated by 15 minute intervals. Along with the feature set, the data includes an adjacency matrix that maps each of the sensors to each other (Figure 1.) Before

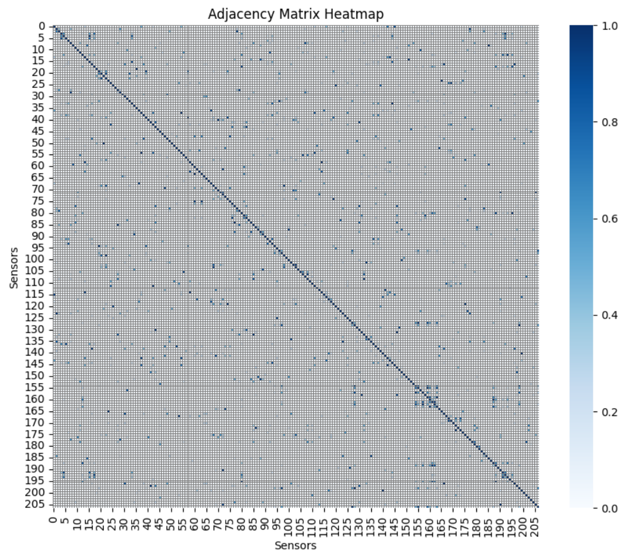

Figure 1. Adjacency Matrix for Sensor Locations - darker represents closer distance.

training, the adjacency matrix was normalized using the following normalization process:

$$\mathcal{A} \equiv D^{-1/2} A D^{-1/2}$$

where A is the adjacency matrix and $D = \text{diag(d)}$ for d(i) the degree of node i. (Williamson, 2016)

A custom data-loading system was implemented to prepare the data. First, we performed Z-score normalization on the features - speed - to standardize the data for training. Next, data was split 80/10/10 into training, testing, and validation sets. We then used a sliding window technique - as demonstrated here to build samples that consisted of two sequences: one of shape (N, 12, 207, 1), and one of shape (N, 9, 207, 1), where N is the number of observations. The dataset with 12 in the second dimension contains samples of 12 time intervals of sequential speed observations (each separated by 15 minutes). The other set has 9 time intervals which are to serve as the predictions. The length 12 input sequence and length 9 output sequence were used in (Yu et al., 2018). Training data was shuffled randomly before the training process.

Procedure

After evaluating a number of models that have been applied to traffic forecasting, I decided on the Spatio-Temporal Graph Network architecture proposed by (Yu et al., 2018). According to their benchmarking, this model consistently outperformed other models on traffic forecasting accuracy, including LSTM-based models and auto regressive models. Due to the popularity of the paper and its success, I decided to implement the architecture proposed by its authors, with the goal of replicating their specifications as accurately as possible in order to achieve similar results.

Training

Our model was trained to closely mimic the specifications described by (Yu et al., 2018). We utilized a learning rate of 10^{-3}, a batch size of 50, and trained for 50 epochs. The PyTorch RMSProp optimizer was selected for training, and a scheduler was used to implement a learning rate decay at a rate of 0.7 after every 5 epochs. We selected MSE loss as the loss criterion function. Again, each of these specifications closely aligns with those used in the paper we base this research on. Training was conducted using an NVIDIA A100 GPU, accessed via Google Colab.

Testing

(Yu et al., 2018), and other traffic forecasting research efforts, typically rely on three main metrics to compare performance between models: Mean Absolute Error (MAE), Mean Absolute Percentage Errors (MAPE), and Root Mean Squared Errors (RMSE). In this study, we similarly compute each of these three metrics in our evaluation of the model's performance.

Model

As mentioned earlier, our model was a Spatio-Temporal Graph Convolutional Network replicating the architecture proposed by (Yu et al., 2018). The original diagram of the model from the paper can be found in *Figure 2.*

This architecture relies on so-called ST-Conv blocks, which contain two Gated Convolutional Layers meant to capture temporal features, with a spatial graph convolutional layer in between them to extract spacial features.

Gated CNN

Temporal features were extracted using a gated CNN architecture, which consists of a single 1-D convolutional layer

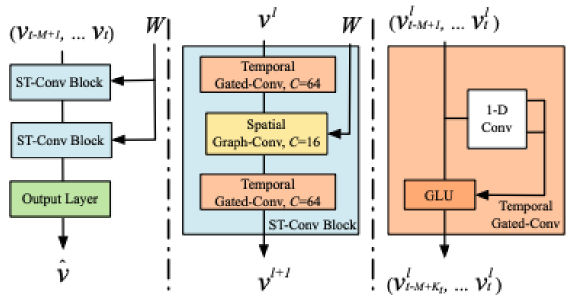

Figure 2. STGCN Architecture via (Yu et al., 2018)

with an input channel for the single feature - speed - and 64 x 2 output channels. This dual set of 64 output channels is then fed to a Gated Linear Unit (GLU), which we imported from PyTorch, which splits the two channels into 64 dimensions each. The Gated CNN layer can be described by the equation below (Yu et al., 2018)

$$\Gamma_{\star\tau} Y = P \odot \sigma(Q) \in \mathbb{R}^{(M-K_1+1)\times C_o}$$

P and Q are the two inputs to the GLU gates; M is the sequence length, K_t is kernel width, and C_0 is the number of channels for the sequence (Yu et al., 2018).

Spatial Graph Convolution

While the paper we base this model off of describes the use of Chebyshev Convolutional Layers, we opted to use a traditional graph convolutional layer (GraphConv), provided by the PyTorch Geometric library. We did not implement the GCNN layer itself, which was originally described in (Morris et al., 2019) The utilized GCNN layer can be described by the following equation:

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j$$

where $e_{j,i}$ denotes the edge weight from source j to target node i. Edge indices and weights were extracted using PyTorch Geometric's `dense_to_sparse` function.

Spatio-Temporal Convolution Block

The Gated GNN's and Spatial Graph Convolution structures were combined to extract temporal and spatial features of our dynamic graph, respectively. The combined unit is referred to as an ST-Conv Block in (Yu et al., 2018). As single ST-Conv Block simply consists of two Gated Convolution blocks, as described above, with a single Spatial Graph Convolution layer in between. Additionally, we created a linear layer between the spatial block and second temporal block to serve as convolutional filter parameters, per the approach described in (Kipf & Welling, 2017). To compensate for overfitting, dropout layers with a rate of 0.35 were applied after each of the three convolutional layers in the ST-Conv Block structure. It is worth noting that (Yu et al., 2018) do not mention the use of dropout layers. The two temporal convolutional layers consisted of 64 output channels, while the spatial layer had 16 - again, these are to specification of the original proposed architecture. An ST-Conv Block can be represented by the following formulation:

$$v^{l+1} = \Gamma_1^l *_\mathcal{T} \mathrm{ReLU}(\Theta^l *_\mathcal{G} (\Gamma_0^l *_\mathcal{T} v^l)),$$

(Yu et al., 2018)

ST-GCN

The final ST-GCN architecture is simply two sequential layers of ST-Conv Blocks, followed by a single linear output layer.

Results

We present the results of our model as compared to the results from (Yu et al., 2018) in Table 1. First, it is worth noting that the original authors training and tested using the PeMSD7 dataset, rather than METR-LA used in this study. Nonetheless, due to similarity between the datasets - both consist of speed data gathered from a variety of sensors in California - we feel that the comparison between results is fair. The original authors computed metrics for short term, medium term, and long term periods of 15/30/45 minutes future intervals; since our dataset worked in 15 minute intervals rather than 5, we cannot directly compare these results. However, both approaches predicted traffic for 9 future intervals, and we have simply averaged the metrics across the full duration of predictions for both our study and for the results given by (Yu et al., 2018).

The original implementation of the ST-GCN by its authors significantly exceeds the accuracy of our model by all metrics, but our model performs well relative to many benchmarked models found in the original paper. It is likely that additional time spent on fine-tuning, as well as slight implementation differences between our model and the original account for these slightly different results. Nonetheless, our model performs well, as evident by Figure 3, which displays strong performance of our model in terms of predicted speeds versus actual recorded speeds for four randomly selected sensors from our testing dataset.

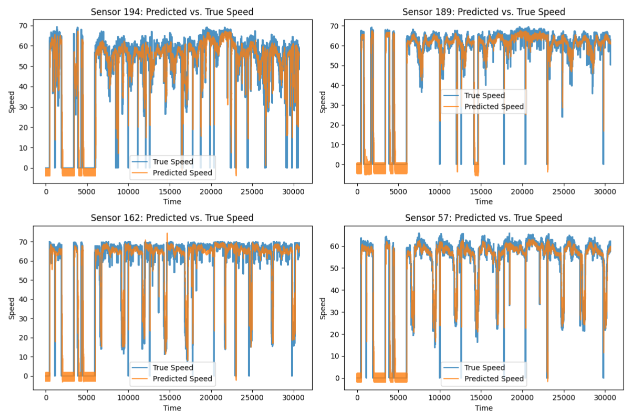

Figure 3. Predicted speeds (orange) versus actual speeds (blue) for four randomly selected sensors from testing.

Table 1. Averaged metrics comparison between this paper's implementation (Custom) and original research results (Yu et al.

MODEL	MAE	MAPE(%)	RMSE
YU ET AL.	3.05	7.25	5.62
CUSTOM	4.47	10.80	9.23

Conclusion

Overall, this research exercise was successful in its goal of learning about unique neural network architectures for making predictions on dynamic graph data. Though it lagged behind the original implmenetation, our model performed impressively in testing. We are confident that additional time spent on fine-tuning the model's parameters and structure would narrow this performance gap, but was beyond the scope of this research due to time limitations.

Acknowledgments

It is worth reiterating that the goal of this project was to replicate and test a system originally proposed by (Yu et al., 2018) - though I made modifications to the structure, I do not claim creation of it.

Additionally, below I have included links to two implementations of ST-GCN's in PyTorch. one of which was implemented by an original author of the paper. While that vast majority of work on my implementation was my own, I acknowledge referencing the following implementations for clarification on certain matters. Citations to the source for code that was influenced by these implementations is included directly in my accompanying Jupyter Notebook. A few particular matters that I struggled to overcome and referenced these implementations for include adjacency matrix normalization, dimension handling, and the use of theta layer in STConvBlock. These references provided clarify on matters that I struggled to interpret directly from the paper. Additionally, I have included a link to PyTorch Geometric, from which I utilized the GraphConv layer and a function to convert the adjacency matrix to edge weights and indices.

I've also included a useful resource for downloading and understanding the METR-LA dataset.

https://github.com/VeritasYin/STGCN_IJCAI-18

https://github.com/FelixOpolka/STGCN-PyTorch/blob/master/stgcn.py

https://github.com/liyaguang/DCRNN

https://pytorch-geometric.readthedocs.io/en/2.5.3/index.html

References

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014602. URL https://doi.org/10.1609/aaai.v33i01.33014602.

Williamson, D. P. Lecture 7, orie 6334 spectral graph theory. Lecture notes, September 13 2016. Scribed by Sam Gutekunst.

Yu, B., Yin, H., and Zhu, Z. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 3634–3640. AAAI Press, 2018. ISBN 9780999241127.