Neural Network Structure for Traffic Forecasting

William Zhuk

December 2019

1 Introduction

Traffic prediction is a long-standing and important field both customer-facing industry (Maps & Guidance Software) as well as control applications (Intelligent Transportation Systems, abbreviated in literature as ITS). The task is to use current-time and historical data to predict future traffic, which then is used to re-route users or alter controls like lights and freeway entrance regulators.

Historically, moving-average (ARIMA variants) and statistical model driven approaches have been leaders in this area. Today, the large amount of public data from many traffic systems have brought data driven Machine Learning approaches to the forefront: LSTMs, node embeddings, and spectral graph decompositions play a major role in many algorithms.

Even more recently, Graph Convolutional Networks have become a large part of learning tasks that have easily accessible information graphs. These new approaches use a combination of LSTMs and graph convolutions to encode both temporal and spatial features to help with prediction.

In this project, we investigate the usage of LSTM cells within convolutions, and how much expressive power is gained by this approach instead of using the graph convolutions and LSTMs in sequence. Future work can expand on larger time horizon predictions, the difference in methods on larger datasets, and potential deeper enhancements to in-convolution LSTM-cells.

2 Related Work

2.1 Graph Convolutions

Graph convolutions have been introduced to many problem areas lately to allow for the application of many convolutional techniques learned in image processing into graph-based problems. Today, there are many different convolutions being used, and in this paper, I have made use of the "vanilla" GCN, GAT [5], and GIN [3] convolutions that are baked into pytorch-geometric.

The difference between these is the function per convolution that applies post-aggregation to generate the outputs before convolutions happen again. GCNs hold a single set of weights, GATs have both weights and an attention mechanism to increase expressiveness, and GINs have small feed-forward neural networks as universal approximators within the convolutions.

2.2 Traffic Graph Convolution

The TGC framework I am using was shown in Cui's paper [2], which also showed its relative performance to previous models on their datasets. This model was centered around a novel Free Flow Matrix, which calculated which stations were within range of causality for traffic based on traffic limits.

$$FFR_{ij} = \begin{cases} 1, S\Delta t - \text{Dist}_{ij} \geq 0 \\ 0, \text{otherwise} \end{cases} \tag{1}$$

where S is the speed of traffic flow, and Δt is the amount of time to look at the network. This mask allows for more selective convolutions that are guaranteed to not lose any precision for influences on the road that move below a certain speed. They then combine this with the graph's modified (to include self edges) adjacency matrix

$$\tilde{A} = A + I \tag{2}$$

$$\tilde{A}^k = \min((\tilde{A})^k, 1) \tag{3}$$

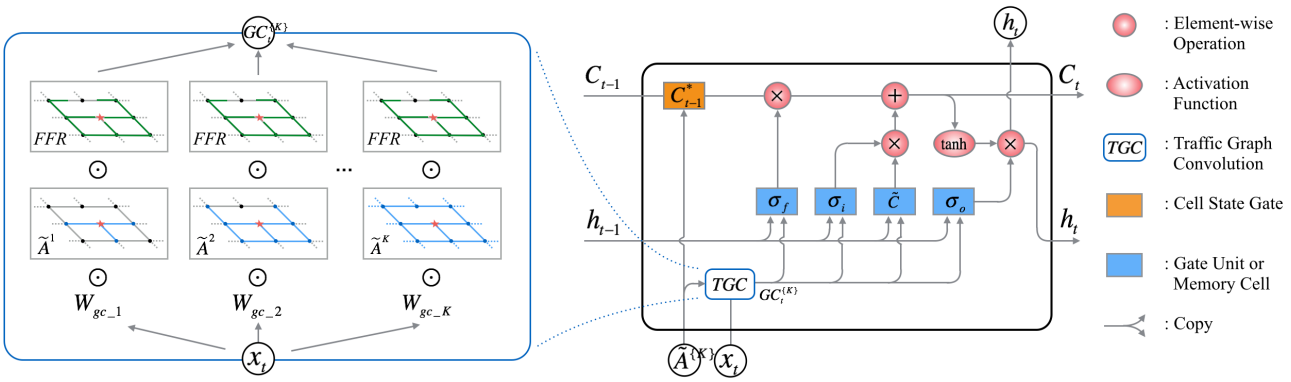

Figure 1: Cui's TGC-LSTM Model [2]

for a certain number k steps, and finally multiply the end result by weights according to distance. The operation can be seen in the left panel of Figure 1, using these results to select for and weight edges in the graph convolution operation.

This paper has the graph convolution before the LSTM cell, with the convolution's output for every time step as the input for LSTM cells. The exact methods for implementing this are reliant on the small size of the graph (the loop dataset used is 323 detectors total), and are implemented by hand with matrix operations.

2.3 T-GCN

A similar model is found in the T-GCN paper [4]. This model was tested on many prediction horizons, proving stable on longer horizons. This paper uses the adjacency matrix and stacked convolutions to create a topologically expressive model. Instead of looking at the road network and adding edges for every sensor that can influence another sensor within a certain time period (Cui's method above), the method here uses repeated convolutions on the original adjacency graph, bringing more distant node information after every convolution.

The paper also went into some depth about the model's power against Gaussian and Poisson noise the training distribution, showing that a model made via a series of convolutions that feed into an LSTM is robust under most real-world circumstances.

3 Dataset

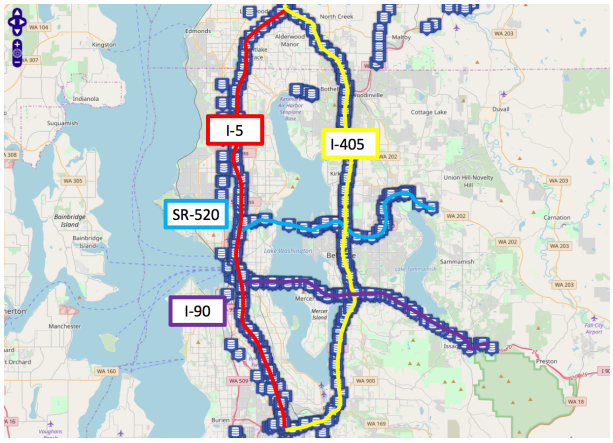

Figure 2: Loop Dataset

The dataset used for this paper is LOOP, a set of time-series speed data from road installed metal loop detectors in Seattle. The detectors are located almost entirely on four connected freeways (I-5, I-405, I-90, and SR-520). This has data for average speeds across 5 minute segments for a year, measured at 323 sensor stations.

The benefits of the Loop dataset is that all sensors are very close to equivalent, and will have similar noise profiles to each other. Since the area covered is also relatively small, many external factors like weather will apply to all nodes in the dataset roughly uniformly.

The downsides to the dataset is the lack of extra features beyond just average speeds, and the lack of positional data about the sensors. The dataset already has free-flow-matrices provided that used the distances to calculate which nodes are reachable within certain time frames, but a dataset with real distances would be much more powerful.

Despite the lack of distances, since the data has loop detectors listed by their highway

and mile-marker, some information can be re-extracted. When doing so, it became apparent that the FFR matrices provided are not correct. Locations with very similar mile counts on the same highway in the same direction frequently had no connections, and adding back those connections gave large differences in the number of edges in the array. To counteract this, I created some FFR matrices manually using some approximations for road intersections and tested the different possible matrices as a hyperparameter.

4 Free Flow Matrix

One of the main defining factors of Cui's TGC networks is the Free-Flow Matrix. This can influence the "neighborhoods", or equivalently the convolutional filter size used by the graph convolutional algorithm. Since traffic is a real-world phenomenon, free-flow matrices allow us to impose the physical limits on traffic propagation into the model.

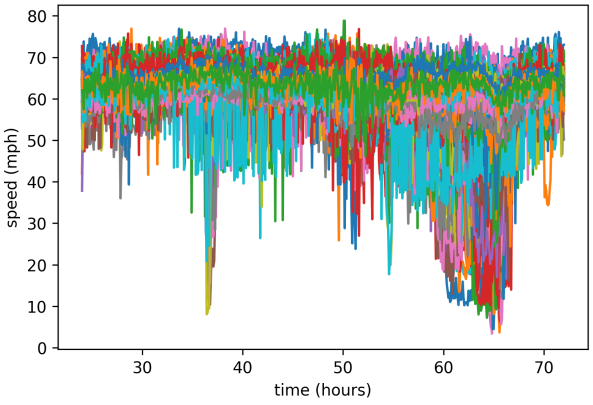

Figure 3: 2 Day Speed Snippet

In Cui's paper, the Free Flow Matrix is set this to the speed limit (60mph). However, when analyzing the data speeds are usually higher than this. The speeds over these datapoints vary a lot, with some reaching past 100mph. Looking at the LOOP dataset used in their paper, we can see that even the averaged traffic speeds are best upper bounded by 80 mph to be more reasonable.

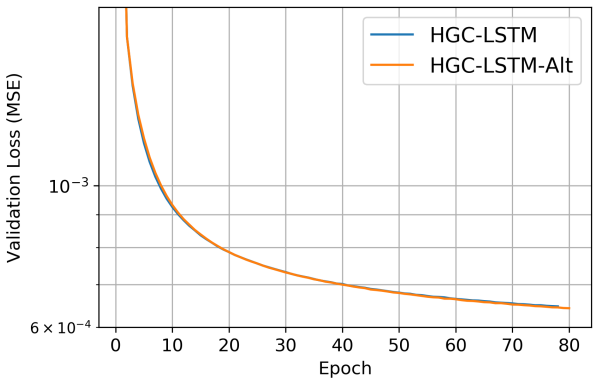

Figure 4: 60mph (default) vs 80mph (alt)

However, when using this new FFM for model training, our performance does not move. This hints that flow-speed of traffic may not be the correct formulation of the speed at which actual traffic flows. After further investigation, the bottleneck for this algorithm is the hop count that the TGC will look at. Changing the hop count changes model performance regardless of selected speed, and it seems the free-flow matrix plays an insignificant part in this operation. For the custom-made model by Cui, increasing K from gave large decreases in speed, and a large enough number caused the model to break due to some form of gradient exponentiation in backprop. Only once K was large enough could the multiplication by the FFR matrix have a considerable difference.

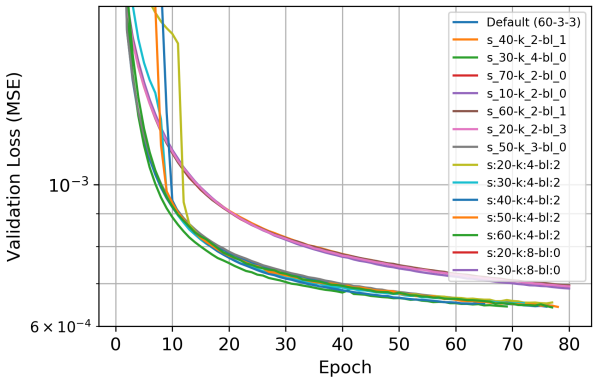

Figure 5: Changing the FFM (s & bl) has much less impact than changing k. The two bands formed are k = 2, and k = 3,4

Since traffic moves slowly through the network, the model still performs well regardless of the hop-count or FFR additions due to the underlying adjacency matrix in it.

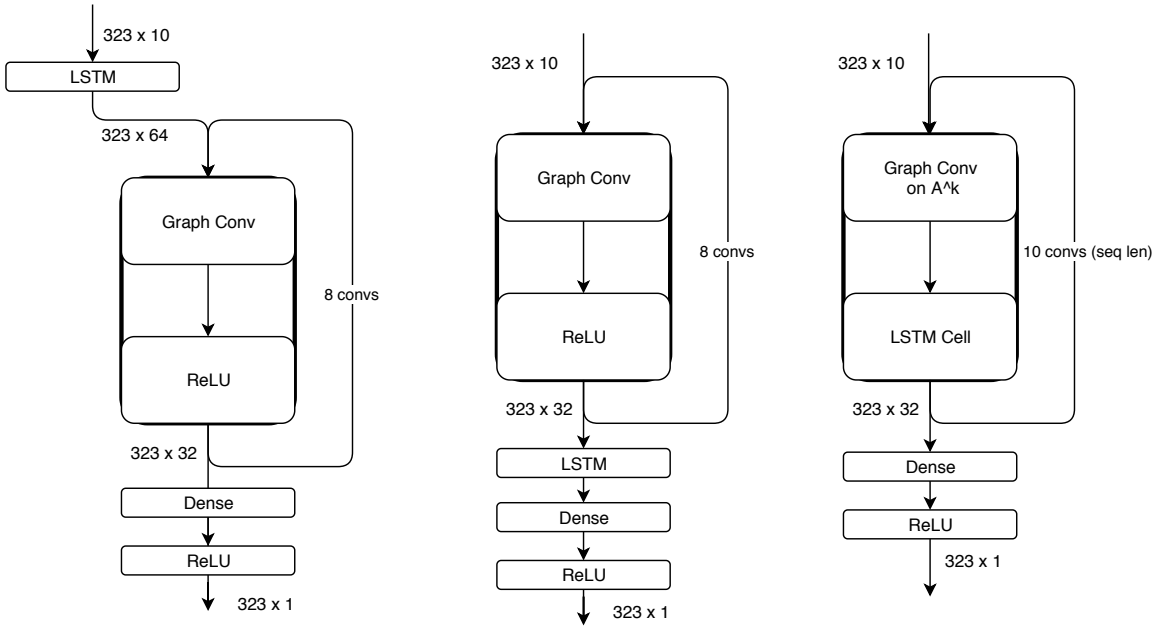

Figure 6: Examples of Architecture Variants for LSTM placement Many of the widths of the outputs of the LSTM and Convs are hyperparameters, as are the number of convolutions and type of convolution.

5 Train/Valid/Test Split

Cui's original paper with the TGC-LSTM model used an interesting splitting technique, where every ordered sequence of input generated data was created, shuffled, and then given to each set. This surprised me, as it meant that valid and test data had many sequence entries that overlap with train data, due to the random shuffling.

Train	Test
60	
58	58
54	54
57	57
53	53
51	51
40	40
38	38
44	44
50	50
	54

Figure 7: One example each from a hypothetical train and test set, with 90% of the sequence data overlapping.

After changing the split to be truly separate, and re-running existing models alongside my own, the difference was minor (a very small increase in valid/test data error). It seems that LSTMs did not overfit to the data, probably because of their sensitivity to the first input, which was always different in the three sets regardless of method.

This being said, all of my data comes from a train distribution on the first 70% of the timestamps, validation on next 20%, and test on the last 10%. This makes sure that none of the model is trained on data pulled form the validation or test set.

6 Comparison Models

The results seen from Cui's paper are custom-made and tailored to a small 323 node graph. However, to use this technique on larger graphs, pytorch's geometric framework [1] can save heavily on memory and computation cost. It will also serve as a way to validate the results in a meaningful comparison.

There are also some existing GNN convolution types within pytorch geometric that can be applied to this problem to validate that the need for an LSTM cell during repeated convolutions is necessary. To do this, I tested different convolutions that featured LSTMs either before or after in the network architecture.

The unfortunate downside to using pytorch-geometric was that for this small graph, the lack of graph batching slowed down the computation compared to the existing matrix-based custom solutions. Extensions to these methods that would flatten out both by graph batch and by nodes in the graph, preserving order, and unflatten afterwards would help with speedup.

7 Hyperparameter Search

Once I had my own model for graph convolutions, I started hyperparameter search on the new structures to see how closely I could approximate the existing methods. This was done on both GAT models and the new custom convolution model. The following is an example of the results for a search on the custom model.

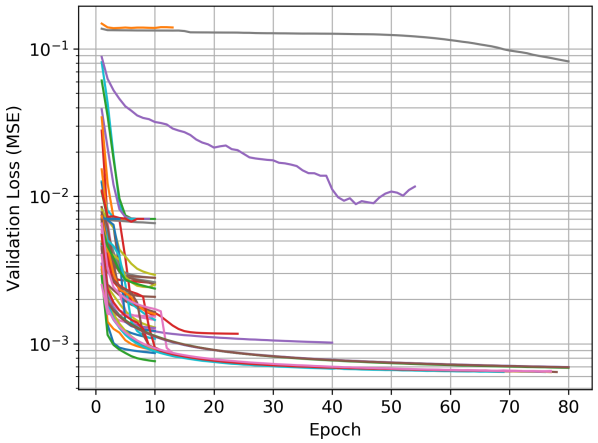

Figure 8: Custom Convolution Hyperparameter Search Results. Given the number of hyperparams, the associated features are not shown for space concerns and instead we just see results.

After the search, it became clear that the best models in each of the categories were performing roughly equivalently on the data. This means that a series approach: using LSTMs to capture the sequential information, and then spreading that around through graph convolutions, had roughly the same power and holding the LSTM inside of the convolutional operator.

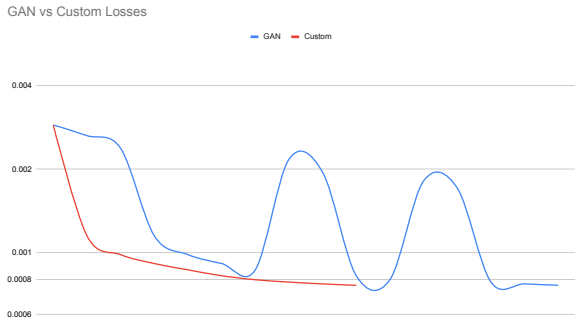

Figure 9: Losses of best of GAN and best Custom.

However, it is important to point out that the sudden spikes in some epochs during the series approach suggests it may be more brittle, especially since this is reflected in other hyperparameter values that perform well. This may be interesting to look into, as perhaps my implementation has some oversight.

8 Conclusion

Neural Networks built on top of both Graph Convolutions and Recurrent Networks are currently the best systems for short-term traffic prediction. The results found on the LOOP dataset found little to no difference in the effectiveness of similar networks that re-order when the spatial and temporal data intermingle. Networks with an LSTM after the graph convolutions performed equally well as networks that had LSTMs before the graph convolutions, and again equally with networks that had LSTM cells within the convolutions.

Equally surprising was that using convolutions on the edge matrix (A) versus the FFR-based matrices ($A^k \bigotimes FFR$) produced little difference when there were enough convolutions in the single-neighbor networks. It provides one extra point of evidence that repeated neighbor-based convolving is good at expressing topological features of graphs.

9 Code

The code is located at https://bitbucket.org/williamzhuk/wz-final-project-2019/src/master/
The original library was made by Zhiyong Cui, at https://github.com/zhiyongc/Graph_Convolutional_LSTM
There are modifications throughout the Code_V2 library, but to make it more manageable for both myself and the reader, you may ignore changes there and instead move into more recent Code_V3 library that I have built. I have made many changes throughout, but Cui's code can still be found (somewhat edited) in the model training file, in the models file above the newer convolutional models, and in the dataset preparation file.

References

[1] pytorch geometric. `https://pytorch-geometric.readthedocs.io`.

[2] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. High-order graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *arXiv preprint arXiv:1802.07007*, 2018.

[3] Jure Leskovec Stefanie Jegelka Keyulu Xu, Weihua Hu. How powerful are graph neural networks? 2018.

[4] Chao Zhang Yu Liu Pu Wang Tao Lin Min Deng Haifeng Li Ling Zhao, Yujiao Song. T-gcn: A temporal graph convolutionalnetwork for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems-2019*, 2018.

[5] Arantxa Casanova Adriana Romero Pietro Liò Yoshua Bengio Petar Veličković, Guillem Cucurull. Graph attention networks. *ICLR 2018*, 2017.