T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction (REPRODUCE)

Haider Shoaib*
haider98@my.yorku.ca
York University
Toronto, Canada

Abdul Hamid Dabboussi*
abduldab@yorku.ca
York University
Toronto, Canada

ABSTRACT

In recent times, traffic congestion has been more of an issue, especially during certain rush hours on the highways from people commuting to work, school, etc., which in turn, affects productivity. Traffic prediction is an important metric to solve traffic congestion issues by forecasting traffic flow at certain times. Some prior research work have used community based methods, Graph Convolutional Neural Network (GCN) models, and Gated Recurrent Units (GRU) to predict traffic. A recent model which was proposed captures the spatial modeling of the GCN and temporal modeling of the GRU called the Temporal Graph Convolutional Neural Network (T-GCN) model. In this project, attempted to reproduce the results which the authors of the T-GCN model have reported in their paper. We discuss the important parts of the model, our journey, successes, failures, and benchmark our results against the original results of the paper.

KEYWORDS

traffic prediction, convolutional neural networks, network models, graphs, spatial dependency, temporal dependency

1 INTRODUCTION

1.1 Motivation

Traffic prediction has been a hot topic in the recent years in the field of ITS (Intelligent Transportation Systems), especially with the rise of technology and ease of travel with ride sharing applications such as Uber and Lyft. In San Francisco, Uber and Lyft contribute to 13.4% of all vehicle miles [2]. Traffic prediction is an important paradigm because it can help forecast real-time traffic information and optimize the traffic flow in certain areas which are highly congested.

In this project, our main goal is to reproduce the T-GCN (Temporal Graph Convolutional Network) model proposed by Zhao et al. [5]. This is a deep learning model which captures the temporal and spatial dependencies traffic data by combining a graph convolutional network (GCN) to learn the spatial dependency of the road network and a gated recurrent unit (GRU) to handle the extraction of temporal information from the time series traffic data.

1.2 Summary of T-GCN

As mentioned in the prior section, we will be focusing on the Temporal Graph Convolutional Network model for our project. T-GCN is a network model proposed by Zhao et al. [5], where they comprised this model from the concepts of GCN (Graph Convolutional Network) and GRU (Gated Recurrent Unit). The GCN covers the structure of the road network in order to model the spatial dependence, and the GRU covers the dynamic time-series properties of traffic data in order to model temporal dependence. This model was used to forecast results under short-term and long-term predictions, and can be used for more than just traffic prediction. What Zhao et al. brought in this model was a 1.5% to 57.8% decrease in prediction error when compared to existing models which can achieve similar types of results. Zhao et al. benchmarks their results against other models such as HA, SVR, and ARIMA [1].

Our goal in this project is to attempt to reproduce the T-GCN model and compare our results with the results Zhao et al. have reported. If our attempt at reproducing the model differs with the results of Zhao et al., we will discuss the possible limitations preventing us from achieving the expected results.

1.3 Related Work

1.3.1 DCRNN Model. DCRNN is a model proposed by Li et al. [3] which stands for Diffusion Convolutional Recurrent Neural Network. Li et al. proposed that traffic flow could be described as a diffusion process and used this model to capture spatial and temporal dependencies by performing bidirectional random walks on the graph, and using an encoder-decoder architecture respectively. They claimed that they observed consistent improvement of 12% - 15% over state-of-the-art baselines.

1.3.2 STGCN Model. Spatial-Temporal Graph Convolutional Netowrk (STGCN) was a novel method for time-series prediction problems proposed by Yu et al. [4] for the traffic domain. Similar to T-GCN, STGCN also attempts to campture the spatio-temporal dependencies of traffic flow. The model's architecture is made up of two convolutional blocks known as ST-Conv blocks because of their spatio-temporal ability and a fully connected output layer. Each ST-Conv block consists of two temporal gated convolution (TGC) layers with a gated linear unit (GLU) and a 1-D convolution layer with residual connection. The two TGC blocks handle the temporal dependencies and sandwich a spatial graph convolution layer (SGC) to extract spatial features from the road network.

1.4 Datasets

The two main datasets which Zhao et al. [5] utilize are the SZ-taxi and Los-Loop datasets. Both datasets consist of traffic speed data taken from different sensors during specific times. Both datasets include an $n \times n$ adjacency matrix which describes the road connectivity, and a feature matrix which include the speed changes over time for each road sensor.

The SZ-taxi dataset is the taxi trajectory of Shenzhen from Jan. 1 to Jan. 31, 2015. It includes 156 major roads and Zhao et al. aggregated the speed on the road every 15 minutes. The Los-Loop dataset

*Both authors contributed equally to this research.

is based off of the METR-LA dataset [3] which consists of speed data taken from 207 sensors across Los Angeles highways from the dates March 1st, 2012 to June 30th, 2012. Zhao et al. aggregated the speed on the road every 5 min.

For our project, we will be utilizing one these two datasets which Zhao et al. have provided to the public to benchmark our results against the results they have presented in their work. More specifically, we will use the Los-Loop dataset which consists of 207 nodes which represents the sensors, and 2016 timesteps.

1.5 Our Approach

Regarding the approach, we will implement the T-GCN model using PyTorch and train, validate, and test using Google Colab. Colab will provide us with the compute resources necessary for reproducing the results achieved by Zhao et al [5]. The central claim of the paper is achieving traffic speed forecasting measured by metrics including RMSE, MAE, and R^2. We will be reproducing a subset of the results (i.e. only one time period rather than the four time periods the authors reported) from Table 1 in section 4.4 of the original paper. Essentially, the forecasting problem is modeled as learning a mapping function f on network G and feature set X such that the traffic information in the coming T periods of time can be calculated as follows:

$$\left[X_{t+1}, \ldots, X_{t+T}\right] = f\left(G; \left(X_{t-n}, \ldots, X_t\right)\right) \tag{1}$$

2 EXPERIMENTS

2.1 Reproducibility Attempt

In terms of the reproducibility attempt, the authors of the T-GCN paper provided some high-level figures and important equations as shown in **Figures 1 and 2**. These figures helped us get an idea of how the model operates and helped us write the TGCN cell portion as described in **Figure 2**. Equations 2-5 are the equations which are used in the T-GCN cell which helped us achieve the output. The explanation of these equations can be found in the orignal paper, but assentially they are the update and reset gates and the outputs, where $f(A, X_t)$ is the graph convolution process where $f(X, A) = \sigma\left(\hat{A}Relu\left(\hat{A}XW_0\right)W_1\right)$.

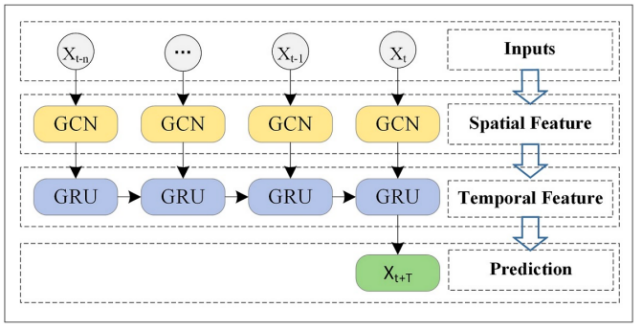

Figure 1: High-level overview of the T-GCN model

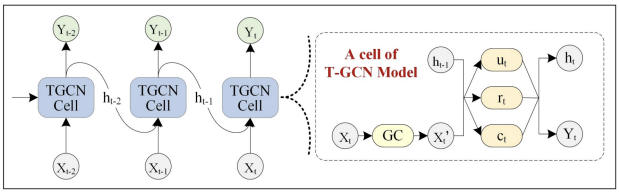

Figure 2: Spatio-temporal prediction process

2.1.1 TGCN Calculations. The following equations represent the process of the TGCN cell which the authors of the original paper have provided.

Update gate at time t:

$$u_t = \sigma\left(W_u\left[f(A, X_t), h_{t-1}\right] + b_u\right) \tag{2}$$

Reset gate at time t:

$$r_t = \sigma\left(W_r\left[f(A, X_t), h_{t-1}\right] + b_r\right) \tag{3}$$

Memory content at time t:

$$c_t = tanh\left(W_u\left[f(A, X_t), (r_t * h_{t-1})\right] + b_c\right) \tag{4}$$

Output at time t:

$$h_t = u_t * h_{t-1} + (1 - u_t) * c_t \tag{5}$$

Therefore, with this information at hand, we attempted to recreate the T-GCN model, with the help of prior PyTorch knowledge as well as referring to different online tutorials to help us craft the code together to combine GCN and GRU for the T-GCN model. In section 3, we discuss our journey of where we started, where we ended up, and the ups and downs along the way. Continuing on, we discuss the model and metrics used, and present our results compared against the original and bench-marked results.

2.1.2 Model and Metrics. Since T-GCN is a regression model, the following five metrics were used to evaluate the prediction performance between the predicted value $\hat{y}_i$ and the real traffic information value y_i:

(1) Root Mean Squared Error:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2} \tag{6}$$

(2) Mean Absolute Error:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{7}$$

(3) Accuracy:

$$Accuracy = 1 - \frac{\|Y - \hat{Y}\|_F}{\|Y\|_F} \tag{8}$$

(4) Coefficient of Determination:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2} \tag{9}$$

(5) Explained Variance Score:

$$Var = 1 - \frac{Var\{Y - \hat{Y}\}}{Var\{Y\}} \tag{10}$$

2.1.3 Loss Function. It is obvious that the goal of estimating traffic speed is a regression problem where the primary objective is to get a prediction as close to the real speed as possible. In other words, the main objective is to minimize the error. This can be achieved by minimizing the difference between the real speed Y and the estimator $\hat{Y}$. In other words, this is the MSE (mean squared error) loss function. Additionally, an L_2 regularization term is added to constrain the learning space by imposing a penalty, essentially minimizing the chance of over-fitting. λ would be added as a hyper-parameter to control the penalty. The final loss function to be used can be described as follows

$$loss = \|Y - \hat{Y}\| + \lambda L_2. \tag{11}$$

2.2 Coding and Testing Details

In terms of the coding itself, we had some trouble setting up the data, in which we discovered in the source code, the train and test split as well as the data setup was done in a very specific manner. Therefore, we are using the exact source code for this task. Furthermore, we use the exact source code to calculate the laplacian of the adjacency matrix A which is just a mathematical operation.

For the training and test sets, we use the original configuration where 80% of the data is used as training and the remaining 20% for testing.

2.3 Results

Table 1 below shows a comparison between the performance of our reproduced T-GCN and a series of baseline models and other advanced traffic forecasting models that have been widely used for traffic forecasting. As mentioned earlier, we used the Los-Loop dataset in our training and testing process. An important thing to note here is that we used the 5 minutes aggregation for our own results which is what we are benchmarking our results to, rather than at various intervals like in the original paper.

In terms of the hyperparameters, in table **Table 1** we used the same setup which the original results used. The learning rate is 0.001, batch size of 64, and 3000 epochs. Furthermore, the Adam optimizer is used with a weight decay of 1.5×10^{-3}.

To insure rigor, the set of compared models includes statistical models along with basic neural networks such as HA, ARIMA, SVR, GCN, and GRU where the results are taken from the T-GCN paper.

- **HA** Historical Average is a technique that models traffic flow as a time-series process and uses the average weight from all previous data points to make a forecast.
- **ARIMA** Auto-Regressive Integrated Moving Average model with (p, d, q) Kalman filter of (3, 0, 1). ARIMA is a statistical time-series forecasting model.
- **SVR** Support Vector Regression model with a penalty term $C = 0.1$ and no kernel.

As seen in **Table 1**, our results for T-GCN is blank. This is due to the fact that our reproduced T-GCN model was not learning with the same hyperparameters as the original paper such as the hidden dimension size and learning rate. Essentially what was happening was that the loss would hover around the same values after each epoch, even when going through up to 1000 epochs. Therefore, we could not obtain any practical results when using these specific hyperparameters. In section 3, we discuss our journey and results we obtained when changing the hyperparameters which allowed for our model to actually train.

Table 1: Performance comparison of our reproduced T-GCN and other baseline traffic prediction models in literature.

Model	MAE	RMSE	Accuracy	R^2	Var
HA [3]	4.0145	7.4427	0.8733	0.7121	0.7121
ARIMA [3]	7.6832	10.0439	0.8275	*	*
SVR [3]	3.7285	6.0084	0.8977	0.8123	0.8146
GCN [4]	5.3525	7.7922	0.8673	0.6843	0.6844
GRU [3]	3.0602	5.2182	0.9109	0.8576	0.8577
Original T-GCN	3.1802	5.1264	0.9127	0.8634	0.8634
Our T-GCN	-	-	-	-	-

3 DISCUSSION

3.1 The Journey

The training process did not go as smoothly as we were hoping. While attempting to replicate the results with the number of hidden units set to 64, the number that the original paper claims to have showed the best results, we realized that the model struggled to learn. First, trying to accomplish that with the original learning rate of 0.001, the model outright refused to learn and the training loss would not change. By decreasing the learning rate to 0.01 and keeping 64 hidden units, we were able to see slight improvement. The training loss started to change but the decrease was almost minimal with high fluctuations.

We decided to stop the training process at 300 epochs because with such a high number of learnable parameters the training process was taking many hours and Google Colab would not allow us to train for that long. **Figure 3** shows how the training loss struggled to decrease over the first 300 epochs.

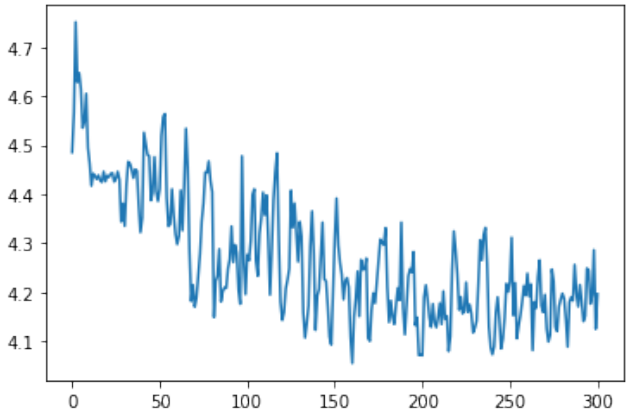

Figure 3: Training loss with 64 hidden units

After this training run for 300 epochs, testing on the validation set confirmed our suspicions that model had not learned much. **Table 2** shows the resulting performance metrics. Not only is the accuracy lower than expected, but the main tell tails were the extremely low explained variance and R^2 scores.

Table 2: Performance of our reproduced T-GCN with 64 hidden units, 300 epochs and lr=0.01.

Model	MAE	RMSE	Accuracy	R^2	Var
Our T-GCN	25.373	2.639	0.7804	0.1794	0.1813

From there, we aimed to decrease the complexity of the model (i.e. number of learnable parameters) to insure a smoother learning process. We found that using just three hidden units allowed us to train much quicker and the training loss decreased nicely as shown in **Figure 4**. The results achieved on the validation set were also much more promising. We saw the validation MAE and RMSE drop to nearly half. Accuracy went up considerably to nearly the same value as the original paper. Most importantly however, was the polar opposite explained variance and R^2 scores which went from being below 0.2 to over 0.7 as shown in **Table 3**.

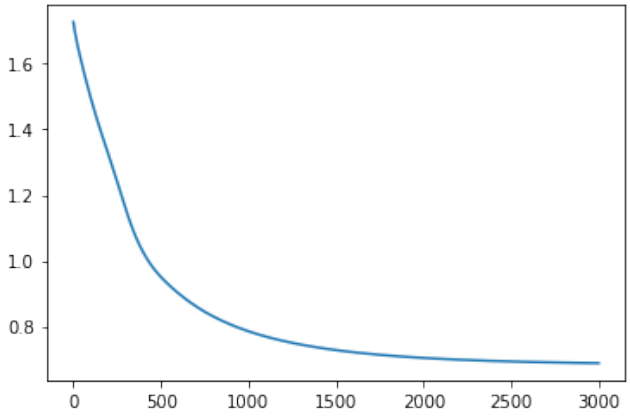

Figure 4: Training loss with 3 hidden units

Table 3: Performance of our reproduced T-GCN with 3 hidden units, 3000 epochs and lr=0.001.

Model	MAE	RMSE	Accuracy	R^2	Var
Our T-GCN	13.235	1.3486	0.8883	0.7762	0.7762

Even though we were able to achieve accuracy, R^2, and explained variance score very close to the original paper's reported results, our loss metrics (namely MAE and RMSE) are drastically different with MAE multiple times higher and RMSE multiple times lower than the original results. After trying different libraries (torch.nn, torchmetrics.functional) and our own functions based on the equations in the paper, we realized that we get different numbers with different methods. We noticed that the factor causing the difference was the *mean* operation in MAE and RMSE. Depending on the shape of the output prediction tensors the model produces, different libraries might take the "number of total elements" differently when calculating the mean (e.g. number of rows vs number of total elements etc.). The difference, however, will always be a constant factor so it would not affect training or comparing results of different hyper-parameters. That said, we were unable to get MAE and RMSE within the same order of magnitude as the paper.

We are not exactly sure why our results are different from the paper's claimed results, however after analyzing the source code of the paper provided by the authors, we have a theory as to why our results are different. When analyzing their training code, they have several methods such as a method named "forward" which is not clear to us where it is used. Furthermore, they utilize PyTorch Lightning techniques to train the model which may have underlying methods to help train the model better which the traditional PyTorch methods to train models do not consider. Due to these unclear parts in the source code, we suspect that this may be the reason why our model is not learning anything with the same hyperparameters reported in the paper.

In the future, we plan to learn how to use TensorFlow, since the authors of T-GCN provide the implementation using TensorFlow, which might be more reproducible.

4 TEAM MEMBER CONTRIBUTIONS

In terms of team member contributions, both of us worked together simultaneously on developing the main T-GCN model code which was based off of the equations 2-5. The sections below detail the work we did individually. It is important to note that not every task is mentioned here, as we would have done it together simultaneously.

4.1 Haider

Haider set-up the GitHub repository for the project and created the Google Colab notebook. He also downloaded the datasets provided in the source code and wrote the code to import the data from GitHub, and utilized the source code methods to obtain the train and test sets. Furthermore, he analyzed the T-GCN model code after it was written by both team members and found some major flaws in the reproduced code, such as incorrect reshaping which resulted in the incorrect output dimension sizes. These issues were critical to fix so that the outputs of the model would be in the correct shape and form. Finally, Haider added more meaningful comments to the T-GCN model code so that it was easy to understand for us to review when troubleshooting.

4.2 Abdul Hamid

Abdul Hamid took on the responsibility of training the model. He had to fix major bugs in the training code which were resulting in errors when we attempted to train. After having the training and validation code ready. Abdul Hamid trained the model, testing different hyperparameters and tried to optimise the results achieved while debugging reasons for failed training attempts. He also had to re-write loss functions manually because as mentioned in section

3.1, built in torch functions for MAE and RMSE were producing incomprehensible results.

4.3 Links

4.3.1 GitHub. https://github.com/haidershoaib98/TGCN-Reproduce

4.3.2 YouTube. https://youtu.be/onoDasO4g1o

REFERENCES

[1] Mohamed S. Ahmed and Allen R. Cook. 1979. Analysis of freeway traffic timeseries data by using box-jenkins techniques. *Transportation Research Board* 722 (1979), 1–9.

[2] Andrew J. Hawkins. 2019. *Uber and Lyft finally admit they're making traffic congestion worse in cities.* https://www.theverge.com/2019/8/6/20756945/uber-lyft-tnc-vmt-traffic-congestion-study-fehr-peers

[3] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations (ICLR '18).*

[4] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* (2017).

[5] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2020. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems* 21, 9 (2020), 3848–3858. https://doi.org/10.1109/TITS.2019.2935152