Comparison of Prominent Temporal Graph Neural Network Models in Traffic Forecasting

Team:
Eric Le
Jason Wu
John Liu

Professor:
Yanning Shen

- Affiliation: University of California, Irvine
- Date: March 14, 2022

Table of Contents

I. Introduction

1. Motivation

Traffic congestion is a source of frustration that almost everybody has experienced within their lives. As California residents, we are all aware of the traffic congestion issues occurring in the greater Los Angeles area. This congestion increases yearly due to an increase of jobs and hot tourist spots for visitors across many countries. Due to its impact on increasing fuel cost, fuel wastefulness, as well as its adverse environmental impacts, traffic forecasting has become a prevalent research topic, especially in recent years.

To address this issue and similar problems, researchers have developed many different time series forecasting models and algorithms. In the context of traffic forecasting these models are essential in the development and implementation of smart vehicle and transportation systems, as well as better traffic management and control systems and practices; all of which provide solutions to the previously mentioned impacts of traffic congestion.

The crux of such models involve machine learning applications through Temporal Graph Neural Neural Networks (TGNN). Unlike conventional Graph Neural Networks (GNN), TGNNs are able to capture how the graph's node features change over time. Through the use of these TGNNs, researchers have developed many models that can be applied to the context of traffic forecasting.

2. Prior Works

A comparison of prominent recurrent TGNN models has been made in other forecasting contexts. The predictive performance of such models were evaluated using mean squared error as the evaluation criteria.
The datasets used for these comparisons were the following:

- Chickenpox Hungary: The officially reported cases of chickenpox in Hungary.
    - Nodes: countries
    - Edges: direct neighborhood relations
- Twitter Tennis RG: Major tennis tournaments from 2017.
    - Nodes: the number of mentions received
    - Edges: structural properties
- PedaMe London: The number of weekly bike package deliveries by Pedal Me in London.
    - Nodes: geographical units
    - Edges: proximity based mutual adjacency relationships
- Wikipedia Math: Wikipedia pages about popular mathematics topics.
    - Nodes: mathematics topics
    - Edges: links from one page to another

	Chickenpox Hungary		Twitter Tennis RG		PedalMe London		Wikipedia Math	
	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative	Incremental	Cumulative
DCRNN [32]	1.124 ± 0.015	1.123 ± 0.014	2.049 ± 0.023	2.043 ± 0.016	1.463 ± 0.019	1.450 ± 0.024	**0.679 ± 0.020**	**0.803 ± 0.018**
GConvGRU [51]	1.128 ± 0.011	1.132 ± 0.023	2.051 ± 0.020	2.007 ± 0.022	1.622 ± 0.032	1.944 ± 0.013	0.657 ± 0.015	0.837 ± 0.021
GConvLSTM [51]	1.121 ± 0.014	1.119 ± 0.022	2.049 ± 0.024	2.007 ± 0.012	1.442 ± 0.028	1.433 ± 0.020	0.777 ± 0.021	0.868 ± 0.018
GC-LSTM [10]	1.115 ± 0.014	1.116 ± 0.023	2.053 ± 0.024	2.032 ± 0.015	**1.455 ± 0.023**	1.468 ± 0.025	0.779 ± 0.023	0.852 ± 0.016
DyGrAE [54, 55]	1.120 ± 0.021	1.118 ± 0.015	**2.031 ± 0.006**	2.007 ± 0.004	**1.455 ± 0.031**	1.456 ± 0.019	0.773 ± 0.009	0.816 ± 0.016
EGCN-H [39]	**1.113 ± 0.016**	**1.104 ± 0.024**	2.040 ± 0.018	**2.006 ± 0.008**	1.467 ± 0.026	1.436 ± 0.017	0.775 ± 0.022	0.857 ± 0.022
EGCN-O [39]	1.124 ± 0.009	1.119 ± 0.020	2.055 ± 0.020	2.010 ± 0.014	1.491 ± 0.024	**1.430 ± 0.023**	0.750 ± 0.014	0.823 ± 0.014
A3T-GCN[68]	1.114 ± 0.008	1.119 ± 0.018	2.045 ± 0.021	2.008 ± 0.016	1.469 ± 0.027	1.475 ± 0.029	0.781 ± 0.011	0.872 ± 0.017
T-GCN [65]	1.117 ± 0.011	1.111 ± 0.022	2.045 ± 0.027	2.008 ± 0.017	1.479 ± 0.012	1.481 ± 0.029	0.764 ± 0.011	0.846 ± 0.020
MPNN LSTM [38]	1.116 ± 0.023	1.129 ± 0.021	2.053 ± 0.041	2.007 ± 0.010	1.485 ± 0.028	1.458 ± 0.013	0.795 ± 0.010	0.905 ± 0.017
AGCRN [4]	1.120 ± 0.010	1.116 ± 0.017	2.039 ± 0.022	2.010 ± 0.009	1.469 ± 0.030	1.465 ± 0.026	0.788 ± 0.011	0.832 ± 0.020

Fig 1.1 TGNN models performance

From the results obtained from above, we see the various TGNN models perform differently across the different datasets. As such, we decided to test some of these models in a traffic forecasting context for this study.

3. Problem

Due to the sheer amount of TGNN models, it can be difficult to determine which specifically captures the context of traffic forecasting in Los Angeles most appropriately. Some models have been specifically developed for traffic prediction contexts, while others were developed to be used for more flexible applications.

In this study, we choose to analyze three different recurrent TGNN models in traffic speed predictions. The selected models all proved to be most effective either in traffic forecasting or another time series forecasting context. The selected models also offer a wide variety of different spatial and temporal layers and vary in how the spatial and temporal aspects of a road network are processed.

The selected models are the: Diffusion Convolutional Recurrent Neural Network (DCRNN), Evolve Graph Convolution Network (EGCN) and Graph Convolutional Long Short Term Memory (GCLSTM) models. We set out to compare these three models under the METR-LA dataset to find the most consistent and suitable model for traffic predictions in Los Angeles.

II. Model Descriptions

1. Diffusion Convolutional Recurrent Neural Network

The DCRNN model captures the spatial dependency with a diffusion convolution layer, and captures the temporal dependency using a scheduled sampling encoder-decoder architecture.

The random walks characterize the diffusion process that relates to the traffic flow of the graph. The stationary distribution of the diffusion process can be seen below, in which the distribution is represented as a weighted combination of infinite random walks.

$$\mathcal{P} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k \left(\boldsymbol{D}_O^{-1}\boldsymbol{W}\right)^k$$

The resulted diffusion convolution operation (1) is then used to build a diffusion convolutional layer (2):

1. $$\boldsymbol{X}_{:,p} \star_{\mathcal{G}} f_{\boldsymbol{\theta}} = \sum_{k=0}^{K-1} \left(\theta_{k,1}\left(\boldsymbol{D}_O^{-1}\boldsymbol{W}\right)^k + \theta_{k,2}\left(\boldsymbol{D}_I^{-1}\boldsymbol{W}^\intercal\right)^k\right)\boldsymbol{X}_{:,p} \quad \text{for } p \in \{1, \cdots, P\}$$

2. $$\boldsymbol{H}_{:,q} = \boldsymbol{a}\left(\sum_{p=1}^{P} \boldsymbol{X}_{:,p} \star_{\mathcal{G}} f_{\Theta_{q,p,:,:}}\right) \qquad \text{for } q \in \{1, \cdots, Q\}$$

This layer is what learns the representations of our road network data.

A modified Gated Recurrent Unit (GRU) is used as the temporal model, which replaces the matrix multiplication within the GRU with diffusion convolution.

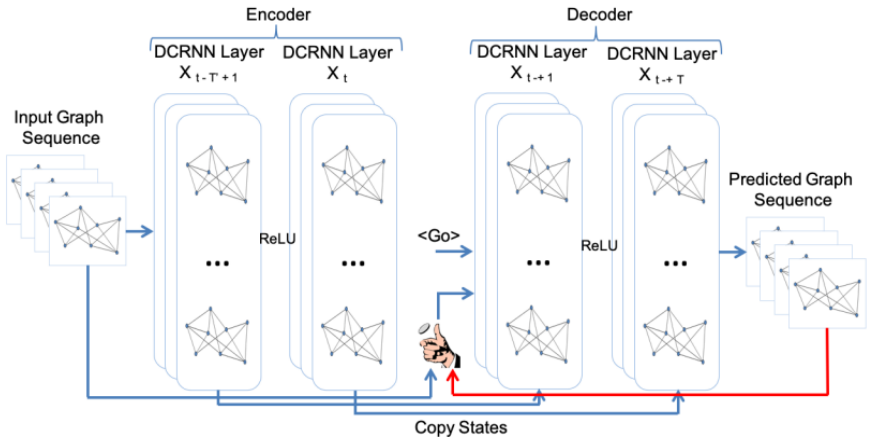


Fig 2.1 Illustration of DCRNN Model

In the above figure, we see that historical traffic data is fed into an encoder whose final states are used to initialize the decoder, which then makes the predictions based on the previous model output.

2. Evolve Graph Convolutional Network

EGCN captures the dynamism underlying a graph sequence by using a recurrent model to evolve the GCN parameters. In other words, EGCN adapts the GCN model along the temporal dimension without making use of node embeddings. Researchers suggested two versions to evolve the GCN weights, treating them with different roles in the RNN. The first one is EGCN-O, whose GCN parameters are input/outputs of the recurrent architecture; and the second one is EGCN-H, whose GCN parameters are hidden states of a recurrent architecture that takes node embeddings as input.

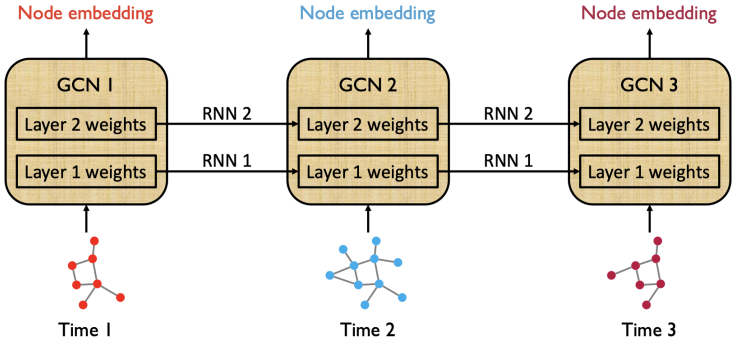


Fig 2.2 Illustration of EGCN Model

Below are the two versions of the ECGN model. In each form, the left is a recurrent architecture; the middle is the graph convolution unit; and the right is the evolving graph convolution unit. Red regions denote information input to the unit and the blue regions denote output information. The mathematical notation *W* means GCN parameters and *H* means node embeddings. Time progresses from left to right, whereas neural network layers are built up from bottom to top.

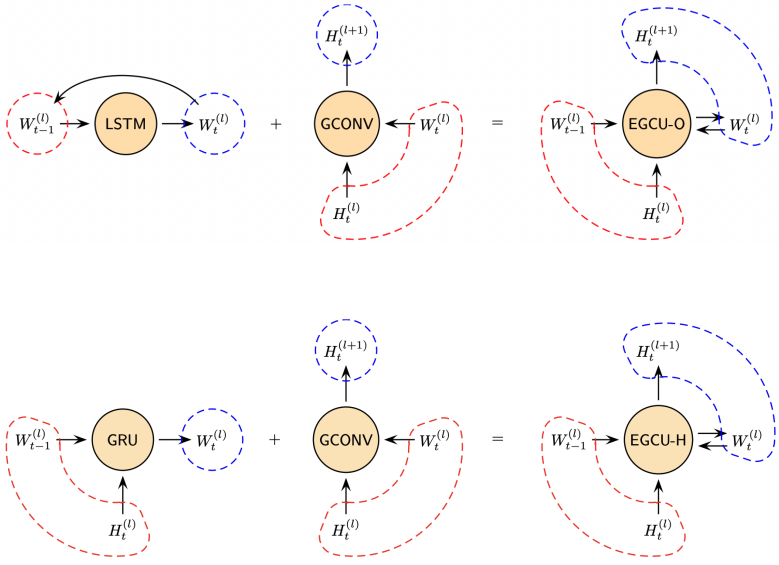


Fig 2.3 EvolveGCN-O (upper) vs EvolveGCN-H (lower)

EGCN-O can be implemented by using a straightforward extension of the standard Long Short Term Memory (LSTM) from the vector version to the matrix version.

function $H_t = f(X_t)$
    Current input X_t is the same as the past output H_{t-1}
    $F_t = \text{sigmoid}(W_F X_t + U_F H_{t-1} + B_F)$
    $I_t = \text{sigmoid}(W_I X_t + U_I H_{t-1} + B_I)$
    $O_t = \text{sigmoid}(W_O X_t + U_O H_{t-1} + B_O)$
    $\widetilde{C}_t = \tanh(W_C X_t + U_C H_{t-1} + B_C)$
    $C_t = F_t \circ C_{t-1} + I_t \circ \widetilde{C}_t$
    $H_t = O_t \circ \tanh(C_t)$
end function

With the above function f, we now completely specify the recurrent architecture:

$$W_t^{(l)} = \text{LSTM}(W_{t-1}^{(l)}) := f(W_{t-1}^{(l)}).$$

Fig 2.5 Implementation of the ECGN-O

The implementation for ECGN-H requires using a standard GRU with two extensions: 1. Extending the inputs and hidden states from vectors to matrices, and 2. matching the column dimension of the input with that of the hidden state.

The matrix extension uses the GRU to process each column of the GCN weight matrix.

function $H_t = g(X_t, H_{t-1})$
    $Z_t = \text{sigmoid}(W_Z X_t + U_Z H_{t-1} + B_Z)$
    $R_t = \text{sigmoid}(W_R X_t + U_R H_{t-1} + B_R)$
    $\widetilde{H}_t = \tanh(W_H X_t + U_H (R_t \circ H_{t-1}) + B_H)$
    $H_t = (1 - Z_t) \circ H_{t-1} + Z_t \circ \widetilde{H}_t$
end function

Fig 2.6 EGCN-H matrix extension

To fulfill the second requirement, researchers summarize all the node embedding vectors into *k* representative ones, which requires a parameter vector that is independent of the time index, but may vary for different graph convolution layers.

function $Z_t = summarize(X_t, k)$
    $y_t = X_t p / \|p\|$
    $i_t = \text{top-indices}(y_t, k)$
    $Z_t = [X_t \circ \tanh(y_t)]_{i_t}$
end function

Fig 2.7 ECGN-H summarization

With the above functions g and $summarize$, we now completely specify the recurrent architecture:

$$W_t^{(l)} = \text{GRU}(H_t^{(l)}, W_{t-1}^{(l)}) := g(summarize(H_t^{(l)}, \#col(W_{t-1}^{(l)}))^T, W_{t-1}^{(l)}),$$

where $\#col$ denotes the number of columns of a matrix and the superscript T denotes matrix transpose. Effectively, it summarizes the node embedding matrix $H_t^{(l)}$ into one with appropriate dimensions and then evolves the weight matrix $W_{t-1}^{(l)}$ in the past time step to $W_t^{(l)}$ for the current time.

Fig 2.8 Implementation of EGCN-H

3. Graph Convolutional Long Short Term Memory

The GCLSTM model uses both the GCN and LSTM frameworks to capture spatial and temporal dependencies. The GCLSTM model uses GCN to learn the network structure in both the hidden and cell states to capture spatial dependencies and uses a LSTM framework to learn the temporal features. The overall models consist of an encoder-decoder architecture, in which the encoder encapsulates both the GCN and LSTM framework.

GCLSTM embeds a GCN after redefining its Laplacian matrix to make it handle directed edges. The ensuing graph convolution seen below then utilizes chebyshev polynomials K to approximate convolutions on the directed network.

$$g_\theta = \sum_{k=0}^{K} \theta_k T_k(\tilde{L})$$

The model then first throws away past cell information based on the forget gate below.

$$f_t = \sigma(A_t W_f + GCN_f^K(\tilde{A}_{t-1}, h_{t-1}) + b_f)$$

Then the cell states are updated (left) and the output information (right) is determined with the following gates.

$$\bar{c}_t = tanh(A_t W_c + GCN_o^K(\tilde{A}_{t-1}, h_{t-1}) + b_c),$$

$$i_t = \sigma(A_t W_i + GCN_c^K(\tilde{A}_{t-1}, h_{t-1}) + b_i), \qquad o_t = \sigma(A_t W_o + GCN_o^K(\tilde{A}_{t-1}, h_{t-1}) + b_0),$$

$$c_t = f_t \odot GCN_c^K c_{t-1} + i_t \cdot \bar{c}_t. \qquad h_t = o_t \odot tanh(c_t).$$

The decoder is a fully connected network layer that converts features back to its original space and outputs a predicted network. Because this model holds long term information within its cell layer and filters out no longer relevant information, it is commonly used for long term predictions.

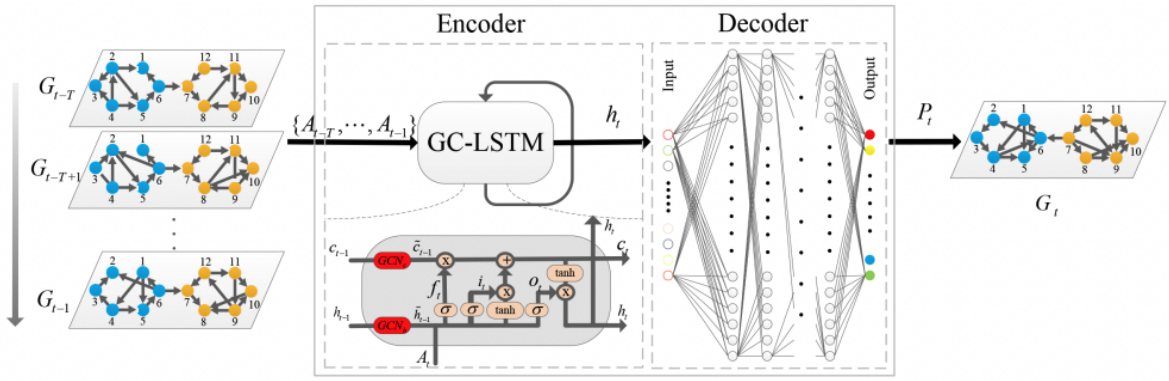


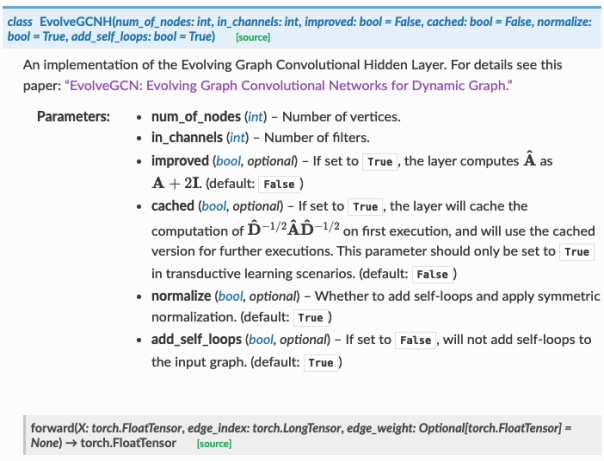
Fig 2.9 Illustration of GC-LSTM model

III. Experiment and Results

1. Implementation

Library Usage

In order to test the above models, we utilized the Pytorch Geometric Temporal Python library. This library allows us to abstract the models at a very high level, while also allowing us to tune the model's parameters in order to produce the best predictions.



class EvolveGCNH(num_of_nodes: int, in_channels: int, improved: bool = False, cached: bool = False, normalize: bool = True, add_self_loops: bool = True) [source]

An implementation of the Evolving Graph Convolutional Hidden Layer. For details see this paper: "EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graph."

Parameters:

- **num_of_nodes** (*int*) – Number of vertices.
- **in_channels** (*int*) – Number of filters.
- **improved** (*bool, optional*) – If set to `True`, the layer computes $\hat{\mathbf{A}}$ as $\mathbf{A} + 2\mathbf{I}$. (default: `False`)
- **cached** (*bool, optional*) – If set to `True`, the layer will cache the computation of $\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$ on first execution, and will use the cached version for further executions. This parameter should only be set to `True` in transductive learning scenarios. (default: `False`)
- **normalize** (*bool, optional*) – Whether to add self-loops and apply symmetric normalization. (default: `True`)
- **add_self_loops** (*bool, optional*) – If set to `False`, will not add self-loops to the input graph. (default: `True`)

forward(X: torch.FloatTensor, edge_index: torch.LongTensor, edge_weight: Optional[torch.FloatTensor] = None) → torch.FloatTensor [source]

Fig 3.1 A sample TGNN layer object from Pytorch Geometric Temporal

Using these implementations, we are able to train and evaluate the models under a common data set.

Data Set and Graph Construction

The METR-LA dataset is used to evaluate the performance of the models. This data set contains traffic speed measurements on north Los Angeles highways from 207 loop sensors. The measurements are taken in 5 minute intervals measured between March to June of 2012. This data set was published alongside the DCRNN paper and is widely used as a standard for traffic forecasting evaluation.

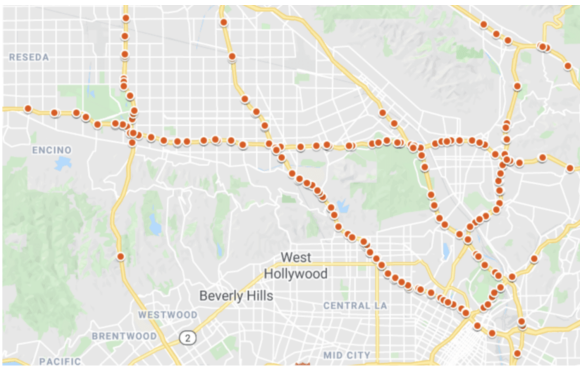


Fig 3.2 Visualization of the 207 loop sensor locations

The loader method from Pytorch Geometric Temporal lets us construct and instantiate a spatially static graph with dynamically changing – at constant time intervals – temporal features. The 207 sensors translate into 207 nodes, with the distance to the next adjacent sensor translating to weighted directed edges. Time of day and speed are the two nodal features of the TGNN, being a function of each time-step indicated by the 5 minute intervals. In the construction of this graph, the speed values are normalized using a z-score transformation in order for the models to work more appropriately.

2. Training and Testing

Of the 34249 samples, we split the data into 80 percent train and 20 percent test data. We then train each model for 200 epochs and evaluate them over a 864 time-step period – three days –, using Means Squared Error (MSE), Root Means Squared Error (RMSE) and Means Absolute Error (MAE) as our evaluation criteria. We evaluate the model's performances over all sensors, as well as three different specific sensors with varying levels of variance between measurements.

Due to the amount of time needed to train each of the models for three different time-steps over the resulting 27413 training samples, for each epoch we only train the model for the first 2000 training samples. In plotting the model's predictions versus the ground truth, we undo the z-score normalization in order to visualize and evaluate the actual speeds. However, for the upcoming prediction results, note that they are based on the normalized values.

3. Results

Prediction Performance

All Sensors

	3 Steps (15 Minute Prediction)			6 Steps (30 Minute Prediction)			12 Steps (1 Hour Prediction)		
Model	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE
DCRNN	0.0561	0.2076	**0.1059**	**0.0428**	0.1804	0.1078	0.0363	0.1697	0.1014
EGCNO	0.1385	0.3607	0.1917	0.1211	0.3412	0.1907	0.1170	0.3363	0.1875
EGCNH	0.1308	0.3598	0.1899	0.1273	0.3542	0.1907	0.1263	0.3384	0.1895
GCLSTM	**0.0503**	**0.2021**	0.1442	0.0460	**0.1739**	**0.0769**	**0.0255**	**0.1447**	**0.0879**

Sensor 122 / 207 Few Peaks and Valleys in Speed

	3 Steps (15 Minute Prediction)			6 Steps (30 Minute Prediction)			12 Steps (1 Hour Prediction)		
Model	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE
DCRNN	**0.0378**	**0.1577**	**0.1078**	**0.0281**	**0.1273**	0.1309	0.0369	0.1333	0.1698
EGCNO	0.1625	0.3745	0.1851	0.1234	0.3511	0.1936	0.1220	0.3652	0.1868
EGCNH	0.1254	0.3381	0.1824	0.1160	0.3408	0.2061	0.1460	0.3364	0.2051
GCLSTM	0.0388	0.1881	0.1940	0.0495	0.1438	**0.1237**	**0.0255**	**0.1017**	**0.1447**

Sensor 69 / 207 Many Peaks and Valleys in Speed

	3 Steps (15 Minute Prediction)			6 Steps (30 Minute Prediction)			12 Steps (1 Hour Prediction)		
Model	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE
DCRNN	**0.0547**	**0.1975**	**0.1375**	**0.0352**	**0.1400**	0.1765	0.0372	0.1601	0.2015
EGCNO	0.1366	0.3267	0.2239	0.0988	0.3234	0.2275	0.0921	0.3322	0.2142
EGCNH	0.1052	0.3257	0.2093	0.1067	0.3197	0.2205	0.1176	0.3155	0.2240
GCLSTM	0.0752	0.2303	0.1847	0.0531	0.1503	**0.1642**	**0.0314**	**0.1151**	**0.1864**

Sensor 18 / 207 Reletively Steady Speed

	3 Steps (15 Minute Prediction)			6 Steps (30 Minute Prediction)			12 Steps (1 Hour Prediction)		
Model	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE
DCRNN	0.0316	**0.1392**	0.1217	**0.0303**	**0.1087**	0.1216	0.0297	0.1232	0.1317
EGCNO	0.1052	0.2792	0.1982	0.0473	0.2376	0.1920	0.0470	0.2048	0.2345
EGCNH	0.0520	0.2243	0.2283	0.0412	0.2271	0.2386	0.0494	0.2151	0.2545
GCLSTM	**0.0309**	0.1628	**0.1167**	0.0330	0.1179	**0.1207**	**0.0182**	**0.1054**	**0.1267**

Fig. 3.3 evaluations over all and individual sensors

Convergence

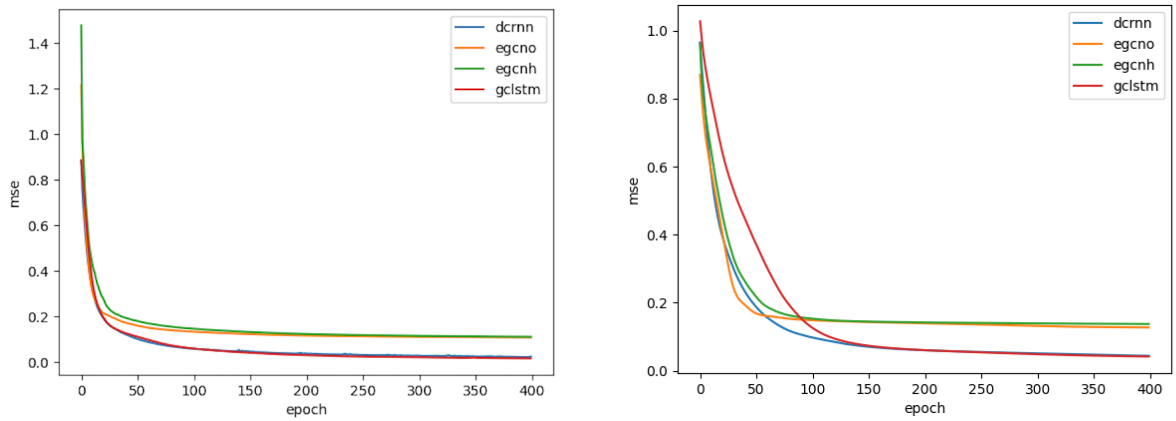


Fig. 3.4 convergence for 1 hour (left) and 15 minute (right) predictions

Visualizations

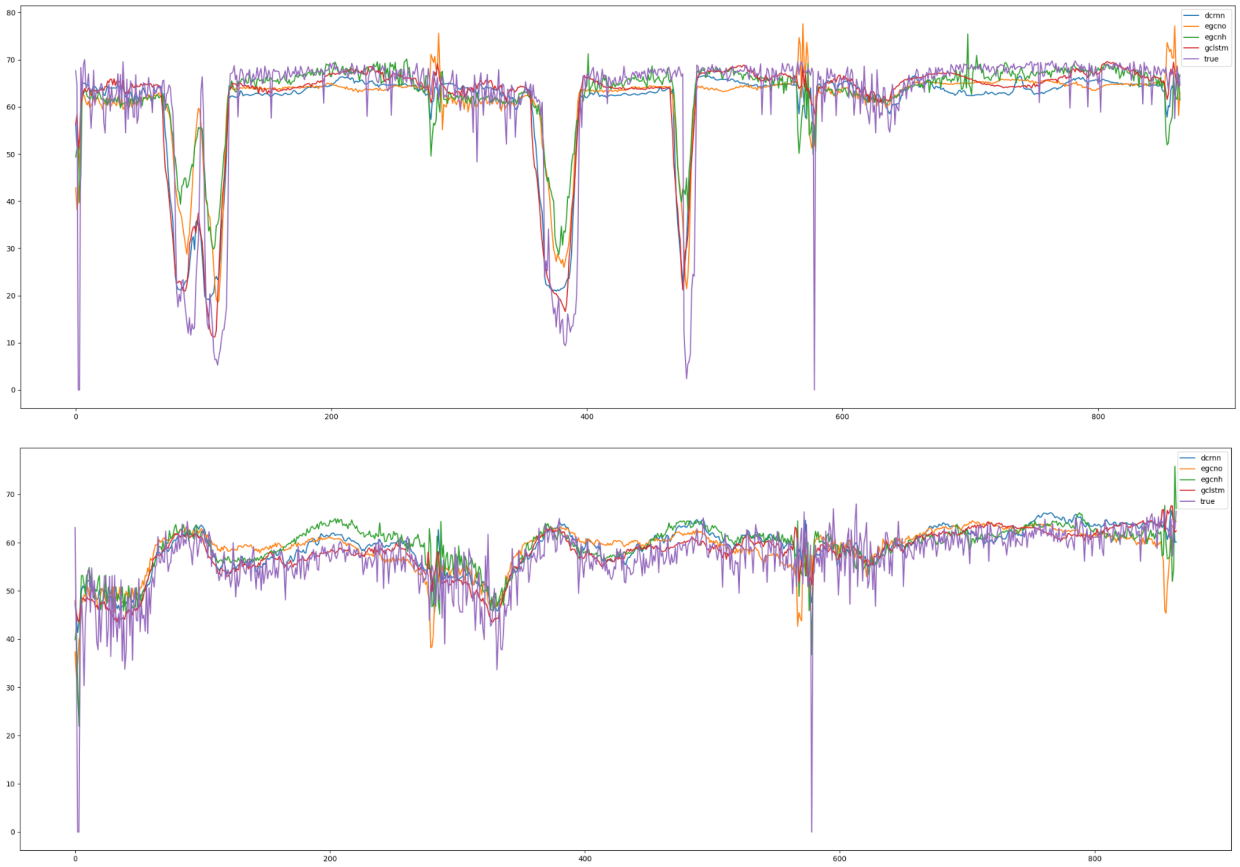


Fig. 3.5 1 hour predictions vs ground truth for sensors 122 (upper) and 69 (lower)

Analysis

As we see from the above results, the DCRNN and GCLSTM model are much more dominant in traffic predictions over the EGCN models. When there are many large fluctuations in speed; we see that DCRNN is most accurate in short term predictions while GCLSTM becomes the most accurate as the predictions become further ahead. This is indicative to GCLSTM's ability to store long term information while simultaneously deciding what old information it can filter out. GCLSTM is also consistently the most accurate with consistent traffic speeds. Across all sensors GCLSTM appears to be the most accurate across all time intervals.

Both the DCRNN and GCLSTM models converge at around the same time, with both EGCN models also converging similarly to each other. We can see that the GCLSTM model starts to converge more slowly the shorter the predictions become, while DCRNN converges similarly across all time interval predictions, indicating its robustness.

From the visual results we see all models are able to generally follow the peak and valley trends of the ground truth. The EGCN models struggle to capture deep valley trends the worst. Interestingly, we expected EGCNH to be more consistently accurate than EGCNO, as its incorporation of additional node embeddings should reflect better results when said embeddings are informative – in this case they are –. Perhaps we underestimated the importance of the graph structure, in which according to its paper, EGCNO's performance is commonly indicative of. Also reflected by the error metric results, we see that the DCRNN and GCLSTM models follow the ground truth the best.

Overall, the DCRNN model provides an extremely robust and consistent prediction performance, speaking to its reputation as a common comparison for newer models. Analogous to the DCRNN paper's experiments, this can be attributed to its use of bidirectional random walks which gives it the flexibility of capturing the physical traffic streams.

The GCLSTM model however, is able to make predictions even more accurately across all time intervals when considering all of the sensors. In reference to its paper, it was similarly able to outperform other state-of-the art models in applications involving dynamic graphs. The predictive performance can be attributed to its embedding of GCN into LSTM cells to better integrate information, rather than simply stacking spatial and temporal layers sequentially. Thus, out of the selected batch of models, the GCLSTM model seems to be the best for traffic forecasting on Los Angeles highways.

IV. Conclusion and Future Direction

In this report, we address choosing a most suitable model in Los Angeles traffic forecasting amongst the vast variety of TGNN based models. We chose some of the more prominent models that have been evaluated against each other in other forecasting applications before that offered a variety of different temporal and spatial layer implementations in order to solve forecasting problems such as that of traffic speed predictions. We evaluated such models under a widely used traffic dataset in order to find which is most suitable for forecasting city highway traffic, concluding that the GCLSTM model seems to be the best out of the batch we selected.

Although traffic prediction has been pretty accurate and demonstrated great results in the past years, there are still many challenges that haven't been fully investigated that can improve it even further. We did not take into account any road accidents and weather conditions which are important factors to consider for traffic forecasting. Also, in this study we only consider models with recurrent temporal layers; there is an entire family of attention based models that can be evaluated. Furthermore, we test our models over a spatially static road network; so a survey can be done to analyze them over dynamic ones. Finally, highway speed forecasting only produces insightful information pertaining to highways, which are not a majority of road networks. There is room for research in analyzing different models over suburban road networks as well.

V. References

EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs.
https://arxiv.org/pdf/1902.10191.pdf

GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Network Link Prediction.
https://arxiv.org/pdf/1812.04206.pdf

DIFFUSION CONVOLUTIONAL RECURRENT NEURAL NETWORK: DATA-DRIVEN TRAFFIC FORECASTING
https://arxiv.org/pdf/1707.01926.pdf

GRAPH NEURAL NETWORKS FOR TRAFFIC FORECASTING.
https://arxiv.org/ftp/arxiv/papers/2104/2104.13096.pdf

PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models.
https://arxiv.org/pdf/2104.07788.pdf

Traffic Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting.
https://arxiv.org/ftp/arxiv/papers/1802/1802.07007.pdf