Traffic forecasting based on Diffusion Convolutional Recurrent Neural Network

CS 512 (Fall 2022) Project final report

Fan Wu*
University of Illinois at Urbana-Champaign
fanw8@illinois.edu

Rong Li*
University of Illinois at Urbana-Champaign
rongl2@illinois.edu

Yifei Liu*
University of Illinois at Urbana-Champaign
yifeil6@illinois.edu

1 Introduction

Spatial-temporal forecasting is challenging because of the complex dependency on temporal and spatial dimensions. Traffic forecasting is one canonical example of such tasks.

The goal of traffic forecasting is to predict the future traffic speeds of a sensor network by using given historic traffic speeds and road networks. Such tasks are difficult since sensors on the road network contain complex spatial correlations.

The future traffic speed is influenced more by the downstream traffic other than the upstream traffic. Hence the spatial structure in traffic is non-Euclidean and directional.

In addition, for the temporal domain, the traffic time series can be non-stationary because of traffic accidents and rush hours which make the model hard to predict in long term. This also increases the complexity of the problem.

There are many approaches designed to solve traffic forecasting problems. Such as ARIMA: Auto-Regressive Integrated Moving Average model with Kalman filter[6] but this method relies on the assumption that the time series is stationary which violates the real-world traffic data. The deep learning models which consider the spatial correlation with Convolutional Neural Networks[1] or graph convolution[2] also have a limitation in the spatial structure.

To solve the traffic forecasting problem, this paper implements a DCRNN model (Diffusion Convolutional Recurrent Neural Network)[5] which captures both spatial and temporal dependencies among time series using diffusion convolution. This DCRNN model has experimented with a real-world dataset that constructing from

2 Related Works

With the rapid development of deep learning, the deep neural network models have received attention because they can capture the dynamic characteristics of traffic data well and achieve the best results at present.

Researchers tend to model the spatial dependency of GCNs on graph-structured data. However, the merging function of the graph convolutional network is fixed which limits the flexibility of the model.

CGRNN [3]: CGRNN models the spatial dependency with Laplacian graph convolution operation and models the time dependency with RNN. It augments RNN by learning a gating mechanism, which is calculated based on the summarized global information, to reweight observations in different timestamps. CGRNN also has a multi-graph structure for non-Euclidean data and global contextual information when modeling temporal dependencies.

STGCN [7]: STGCN models the spatial dependency like CGRNN. To fully utilize spatial information, STGCN models the traffic network by a general graph instead of treating it separately. It also employs a fully convolutional structure on the time axis to handle the inherent deficiencies of recurrent networks. STGCN is composed of several spatiotemporal convolutional blocks, each of which is formed with two gated sequential convolution layers and one spatial graph convolution layer in between.

T-GCN [8]: T-GCN combining the graph convolutional network and gated recurrent unit. The graph convolutional network is used to capture the topological structure of the road network to model spatial dependence. The gated recurrent unit is used to capture the dynamic change of traffic data on the roads to model temporal dependence. T-GCN leverages 1st order diffusion convolution as a merging function of the spatial dependency model and leverages a fully connected network instead of RNN to model temporal dependency.

DCRNN [5]: DCRNN models the spatial dependency with diffusion convolution network and models the temporal dependency using a Recurrent Neural Network (RNN) variant: Gated Recurrent Units. DCRNN captures the spatial dependency using bidirectional random walks on the graph, and the temporal dependency using the encoder-decoder architecture with scheduled sampling.

The DCRNN model implemented in this paper is a variation of the original DCRNN design. The overview of the model and algorithm are discussed in the next section3. A detailed implementation approach and analysis of the predicted result are given in the later section 4, 5, 6.

3 Algorithm Overview

The goal of the DCRNN algorithm is to predict the future traffic speeds of a sensor network based on the historical condition of the network. Mathematically speaking, we want to model a prediction function $h(X)$ that $X_{future} = h(X_{nowandpast})$. In DCRNN, $h(X)$ is the combination of diffusion convolution, an RNN variant(GRU), and encoder-decoder architecture. The goodness of the model is measured based on MAE, RMSE, and MAPE.

To be specific, firstly, diffusion convolution in the algorithm considers both upstream and downstream traffic on a directed graph, so it is defined as:

$$DC(X, f_\theta) = \sum_{k=0}^{K-1} (\theta_{k,1}(D_O^{-1}W)^k + \theta_{k,2}[Reverse(D_O^{-1}W)]^k)X$$

where X is the graph signal. Reverse(X) denotes the reversed direction diffusion process, D_O is the out-degree diagonal matrix, W is the weighted adjacency matrix, and $\theta_{k,1} + \theta_{k,2} = 1$. This closed

*Both authors contributed equally to this work. Authors are ordered alphabetically.
†Both authors contributed equally to this work. Authors are ordered alphabetically.

form converges to a stationary distribution and is similar to the random work with restart(RWR). Theory prepared, we will apply diffusion convolution combined with an activate function, for example, RELU, to create a diffusion convolutional layer, which can be used to capture spatial features of the traffic flow.

Secondly, GRU architecture will capture temporal features of the traffic network. We will combine the GRU with the diffusion convolutional layer, so the equations for modified GRU (MGRU) is shown below:

Update gate: $Z_t = \sigma_z(DC_z(X_t, h_{t-1}))$

Reset gate: $R_t = \sigma_r(DC_r(X_t, h_{t-1}))$

New memory cell: $\tilde{h_t} = tanh(DC_c(X_t, (Z_t \odot h_{t-1})))$

Hidden state: $h_t = (1 - Z_t) \odot \tilde{h_t} + Z_t \odot h_{t-1}$

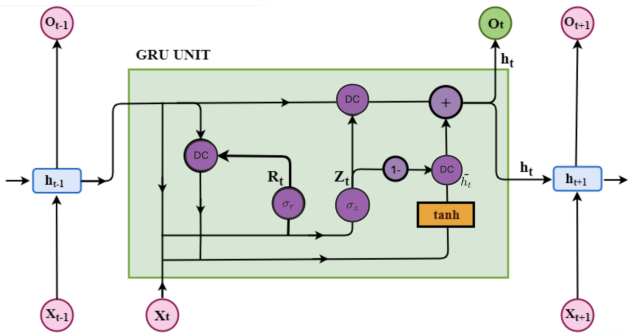


Figure 1: MGRU

The diffusion convolution was implement by using either Laplacian Matrix or Random-Walk Markov Matrix.

In the project, we implement symmetrically normalized Laplacian. It is defined as the difference between the degree matrix and the adjacency matrix of the graph. The degree matrix is a diagonal matrix that contains the degree of each node on the diagonal, and the adjacency matrix is a square matrix that contains information about which nodes are connected to which other nodes in the graph. The symmetrically normalized Laplacian is defined as $L = D^{-1/2} * (D - A) * D^{-1/2}$, where D is the degree matrix and A is the adjacency matrix.

For Random-Walk Markov Matrix. It is defined as the transition matrix of a Markov chain, where the probability of transitioning from one node to another is proportional to the number of edges connecting them in the graph. let $G = (V, E)$ be a graph, where V is the set of nodes and E is the set of edges. The Random-Walk Markov Matrix for G is defined as the matrix P, where the entry $P_{i,j}$ is the probability of transitioning from node i to node j in one step of the random walk. This probability is defined as $P_{i,j} = 1/degree(i)$ if (i, j) is an edge in E, and 0 otherwise, where $degree(i)$ denotes the number of edges incident on node i.

4 Data preparation

This section is going to give a brief view of the construction of the original dataset and data preprocessing steps.

4.1 Dataset generation

As mentioned in 1, The new database is built by extracting the recorded sensor data in the Bay area from the PeMS system. The time period we pick is between Jan 2022 to Jun 2022.

4.1.1 Build dataset from raw data :

The raw data contains basic information about around 2,4000 sensors in the bay area. (such as timestamp, sensor id, total flow, average speed, station length, and so on.)

A new entry is added every 5 minutes for each sensor, as shown in 2. Entries in the raw data are not ordered by timestamp yet, since the raw data files are not read in time order. This problem is fixed in the data cleaning and feature engineering steps.

```
df_Jan  = pd.read_csv('all_Jan.txt', delimiter=',', header = None)
df_Jan
```

	0	1	2	3	4	5	6	7	8	9	...
0	01/19/2022 00:00:00	400000	4	101	S	ML	0.415	40	100	19.0	...
1	01/19/2022 00:00:00	400001	4	101	N	ML	0.265	50	100	30.0	...
2	01/19/2022 00:00:00	400002	4	101	S	ML	0.310	50	0	174.0	...
3	01/19/2022 00:00:00	400006	4	880	S	ML	0.340	40	100	74.0	...
4	01/19/2022 00:00:00	400007	4	101	N	ML	0.365	50	0	107.0	...
...	...	...	...	...	...	...	...	...	...	...	...
35629939	01/14/2022 23:55:00	426457	4	84	S	ML	0.400	30	0	162.0	...
35629940	01/14/2022 23:55:00	426458	4	84	S	ML	0.435	30	0	162.0	...
35629941	01/14/2022 23:55:00	426459	4	84	S	FR	NaN	10	0	NaN	...
35629942	01/14/2022 23:55:00	426460	4	84	S	OR	NaN	20	0	NaN	...
35629943	01/14/2022 23:55:00	426461	4	84	S	ML	0.360	40	0	40.0	...

35629944 rows × 52 columns

Figure 2: Raw data example

In raw data, we pick three features to form our new dataset: timestamp, sensor_id, and average speed in 5 mins. In our model, we only concern with the average speed to predict the traffic load. Because the amount of sensors in the bay area is large, to narrow down the area covered, we pick 319 sensors in a specific area to observe the traffic load. The sensors we picked are mostly the same as [5] so that we can make a comparison of the model results between different time periods.

However, during the formation of the dataset, we found that there are sensors that didn't return data starting in Mar 2022, probably losing the connection to the PeMS system, to keep the consistency among the data, these sensors are removed.

The DCRNN model also needs a graph of the location of all sensors. This data is also generated from the sensor metadata in the PeMS system.

4.1.2 Deal with NaN value : During the 4.1.1 step, NaN values are found in the raw data, they are considered as outliers. Since simply remove nodes contain NaN value, or remove time rows which contain NaN value will destroy the structure in both spatial and spectral domain, NaN values are filled with values from pervious time slot, which is data recorded in the past 5 mins.

All of the dataset files can be accessed through the project github repo. 'Dataset. ipynb' includes the code and steps to form the dataset.

sensor_id	400001	400017	400030	400040	400045	400052	400057	400059	400065	400069	...
Timestamp											
01/01/2022 00:00:00	71.8	68.1	68.1	67.9	68.1	68.8	68.2	68.2	66.7	69.8	...
01/01/2022 00:05:00	71.1	68.0	68.1	67.4	68.0	68.7	68.0	68.2	66.4	69.1	...
01/01/2022 00:10:00	70.9	68.0	68.1	67.7	68.0	68.5	68.0	67.3	66.5	69.7	...
01/01/2022 00:15:00	71.3	67.8	68.2	67.6	67.8	69.0	67.9	67.7	66.1	68.4	...
01/01/2022 00:20:00	70.7	67.8	68.0	67.3	67.8	68.0	67.9	67.6	66.6	66.9	...
...	...	...	...	...	...	...	...	...	...	...	...
01/31/2022 23:35:00	71.1	61.9	67.5	68.4	67.7	66.9	65.6	66.8	65.3	68.6	...
01/31/2022 23:40:00	70.9	61.9	68.5	67.2	67.7	65.6	65.4	67.2	66.6	70.2	...
01/31/2022 23:45:00	70.2	62.1	68.3	67.6	67.7	65.6	65.9	67.1	65.7	70.0	...
01/31/2022 23:50:00	70.1	61.8	67.4	68.8	67.8	67.0	65.9	67.9	66.0	69.1	...
01/31/2022 23:55:00	70.9	62.0	68.3	67.1	67.5	66.9	66.1	66.6	66.3	67.3	...

8928 rows × 320 columns

Figure 3: Data after feature engineering.

	sensor_id	Latitude	Longitude
0	400001	37.364085	-121.901149
1	400017	37.253303	-121.945440
2	400030	37.359087	-121.906538
3	400040	37.294949	-121.873109
4	400045	37.363402	-121.902233
...	...	...	...
320	413845	37.422887	-121.925747
321	413877	37.321613	-121.899642
322	413878	37.324641	-121.888603
323	414284	37.323066	-121.896538
324	414694	37.315051	-121.913497

325 rows × 3 columns

Figure 4: Sensor data.

4.2 Data preprocessing

For the spectral domain, the DCRNN model needs the input data to be fed in the correct format. The input data should continually provide the average speed for each sensor in a time sequence.

For the spatial domain, we need to generate a graph convolutional network based on the spatial dependence within the road nodes.

4.2.1 Generate spectral domain data :

Since the input data depends on time series, the input data need a new feature that indicates time. Simply using the index value Datetime type is difficult for the model to understand, so we quantify time into numbers and add time data to each sensor.

Figure 5, gives an example of the structure of input data. The first element represents average speed, and the second element represents time. In this case, time is at 00:00:00, and figure 6 is at 00:05:00 which matches the expectation.

The speed data shown in both figure are not transformed data. Later when the input data is loaded into the training model, all the speed data passed through transform function to improve performance and run-time efficiency.

```
array([[[7.18000000e+01, 0.00000000e+00],
        [6.81000000e+01, 0.00000000e+00],
        [6.81000000e+01, 0.00000000e+00],
        ...,
        [6.83000000e+01, 0.00000000e+00],
        [7.14000000e+01, 0.00000000e+00],
        [6.78000000e+01, 0.00000000e+00]],
```

Figure 5: Ex1: Time data after quantification

```
[[7.11000000e+01, 3.47222222e-03],
 [6.80000000e+01, 3.47222222e-03],
 [6.81000000e+01, 3.47222222e-03],
 ...,
 [6.82000000e+01, 3.47222222e-03],
 [7.13000000e+01, 3.47222222e-03],
 [6.84000000e+01, 3.47222222e-03]],
```

Figure 6: Ex2: Time data after quantification

4.2.2 Generate spatical domain data :

For the space scale, the weighted adjacency matrix of the road graph is created based on the figure 4 where each sensor location represents one road, and the road network distance between sensors in the traffic networks. [4][5].

The adjacency matrix is normalized and given a threshold value of k. If the distance(weight) between two nodes is less than k, then it is set to 0.

5 Modeling

As mentioned in 3, we use the DCRNN model to infer the vehicle speed at different places in the future. Specifically, we use past 60 minutes speeds detected in the 319 sensors and predicts the speeds of following 60 minutes. There are three main parts in our DCRNN model, Gated Recurrent Unit (GRU) with convolutional layer, Encoders, and Decoders. By applying GRU and the convolutional layer, we can capture both time features and spatial features.

5.1 Gated Recurrent Unit (GRU)

The GRU cell shown in Figure 7 contains input, reset gate, update gate and hidden state.

Input can be ‘graph signal‘ tensor from historical data or predicted data from the last time step, which is of size (batch size, number of nodes, number of features).

The reset gate and update gate here is a neuron network that can be a fully connected layer or convolutional diffusion layer. The right-most architecture is for generating a candidate hidden state, which is a neuron network using a convolutional diffusion layer. To be more specific, the convolutional layer contains calculating the Random-Walk Markov Matrix or Laplacian matrix on the adjacency matrix depending on our parameter configuration.

5.2 Encoder

Stack the GRU cell mentioned in the above section 5.1 together, which process all the historical graph signals we feed into the model.

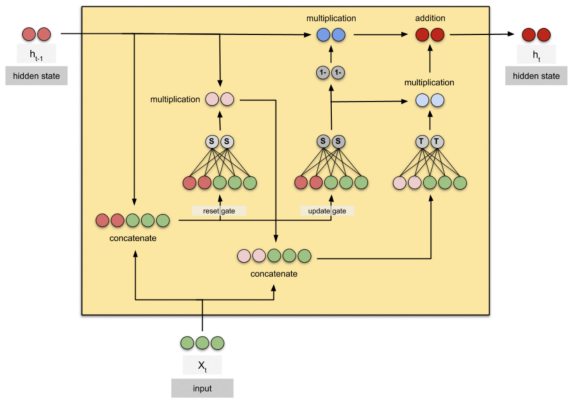


Figure 7: GRU architecture

It takes the initial hidden state and the graph signal at time t as inputs.

5.3 Decoder

Stack the GRU cell mentioned in the above section 5.1 together, which makes predictions on future graph signals. It takes the input from the output of the last encoder and an initial start symbol.

We applied 'Curriculum Learning' in Decoder forward. It trains a model on a sequence of increasingly difficult tasks as the number of batches increasing to help with model improving. The difficulty increasing rate is controled by 'cl decay step' in exponential rate.

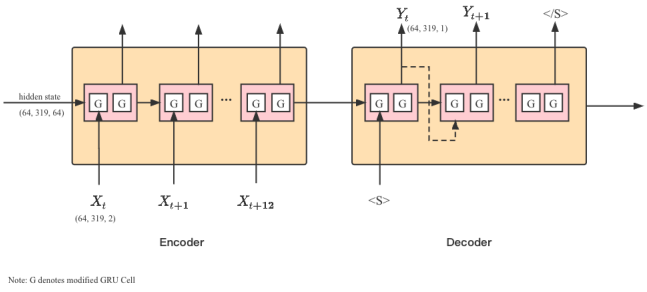


Figure 8: Encoder Decoder architecture

Overall, as shown in the Figure 8, we used two Modified GRU layers to process each inputs, stacked 12 GRUs in the Encoder and stacked 12 GRUs in the Decoder. The input shape of hidden state is (batch size, number of nodes, rnn units) = (64, 319, 64) and the shape of input data is (batch size, number of nodes, input feature dimension) = (64, 319, 2), in which 2 features are speed and time. The output shape of the decoder is (batch size, number of nodes, output feature dimension) = (64, 319, 1), in which the only output is speed.

6 Training

6.1 Training Script

Given the number of data samples, model training is run on Google Colab GPU to speed up the training process (see Appendix 1). Since the Google Colab session will expire within a few hours, we decided to build up our training process so that we save our model as checkpoints periodically in terms of training epochs inside Google Drive so that we can always recover from one of the best models in the future even if the training process was shut due to the session expiration.

6.2 Hyper Parameters

We use Masked MAE loss to evaluate the model. We started with the hyperparameters used in the original paper, which was using Adam with a base learning rate of 0.01, epsilon 0.3, and step-wise learning rate decay that decreases the learning rate at the $20, 30, 40, 50^{th}$ epoch with a learning rate decay rate at 0.1.

However, the model suffers from overfitting under this hyperparameter setting. The loss value over epochs is shown in 9. We then tune our hyperparameters in order to add additional regulations to prevent overfitting. We added an early stop with patience 3 and delta 0 so that if the validation loss increases for continuous three epochs the training stops. Observing that the validation loss stops decreasing around epoch 30, we changed the learning rate decay steps to decrease at the $8, 16, 32^{th}$ epoch.

We reduced the initial learning rate to 0.007 and decrease the Adam epsilon to 1e-4. We also compared two filter types. The original paper used "Dual Random Walk" and we decided to compare the Laplacian filter with the Dual Random Walk filter. The results for the tuned model and further improvements will be described in detail in 6.5 and 7

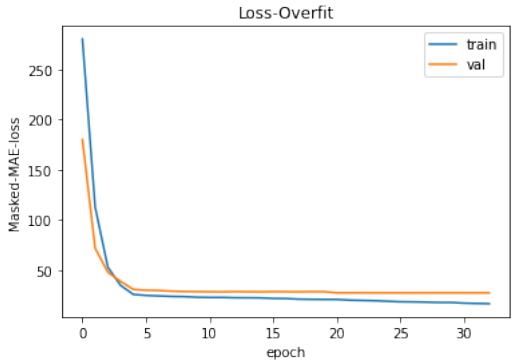


Figure 9: Loss Overfit

6.3 Model tuning

Model tuning is the process of adjusting the parameters of a machine learning model to optimize its performance on a specific dataset. This is done by using techniques such as cross-validation and grid search to evaluate the model's performance on different combinations of parameters. The goal of model tuning is to improve the model's accuracy, and reliability and prevent overfitting.

6.4 Normalize Data

Data normalization is treated as a regulation method and we compared the results of the model trained on normalized data and un-normalized data. Figure 10 shows the prediction and label (ground truth) of the vehicle speed and it is obvious that though the trend of the prediction matches the label, there is a huge gap in-between. We have also observed that the loss value of the model trained on the un-normalized data was significantly bigger than that trained on the normalized data, where the former converged to 28 (Figure11) and the latter converged to 3. More analysis of the model trained on normalized data will be presented in Figure6.5

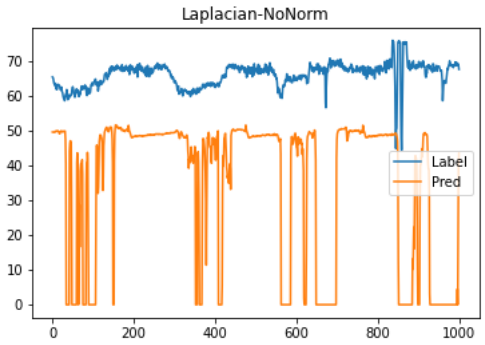


Figure 10: Lap-NoNorm

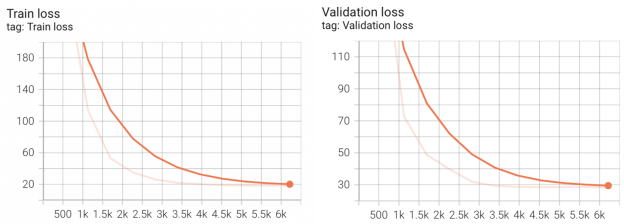


Figure 11: Lap-NoNormLoss

6.5 Results

As discussed in 6.2 and 6.4, we compared two filter types on both normalized data and un-normalized data. This section will mainly focus on presenting the results of two filter types trained on normalized data.

6.5.1 Laplacian Filter The results for using the Laplacian Matrix in the diffusion convolution layer. As shown from the training loss plot (Figure13), it is clear that using the Laplacian suffers from overfitting issues. Due to the time limitation, we managed to train our model with 15 epochs of data, and the model was underfitting as the validation loss is still decreasing. On the other hand, the training loss is increasing. This could also be a signal that the model was underfitting. Even trained with limited data, the result was still satisfactory (Figure12). We can see that the predicted result shows periodic characteristics, which makes sense due to peak and low periods.

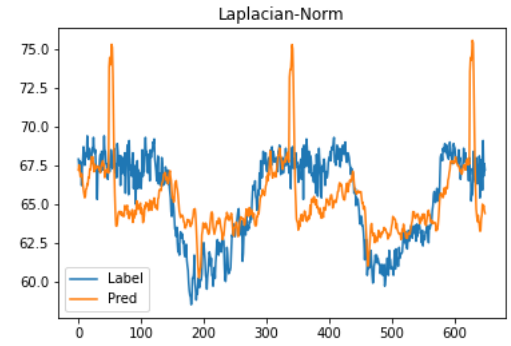


Figure 12: Lap-Normed

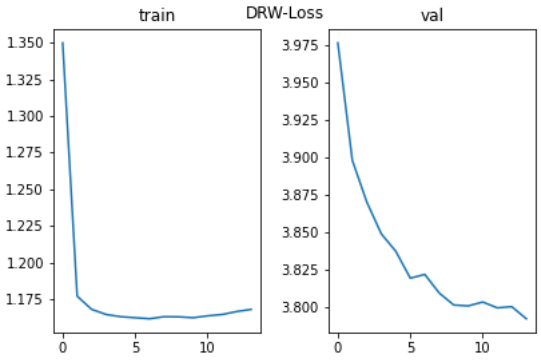


Figure 13: Lap-loss

6.5.2 Dual Random Walk Filter The results for using Random-Walk Markov Matrix in the diffusion convolution layer: The validation loss converges to around 3.4 after 5 epochs. Due to the time limitation, we managed to train our model with 5 epochs. As shown in Figure14, the absolute difference between the ground truth speed and the real speed can be seen as less than 10 omitting outliers. Compared to the Laplacian filter-type training process, it can be seen (Figure15) that using the DRW filter can prevent the model from overfitting. On the other hand, the training loss is increasing while the validation loss is decreasing. It can be potentially caused due to limited epochs trained.

6.5.3 Analysis The second model trained 2 times slower than the first model. Because it used random walk twice. We finally achieved 1.18,3.79 of masked MAE for training and validation in the first one and 3.10, 4.43 of masked MAE in the other. Note that, the MAE has much room for improvement because we only train for 15 and 6 epochs respectively due to the time limit. The gap between training loss and test loss depends on the filter type we used in the diffusion convolution layer. But we tried to first standardize the columns then it ran 1.5 times faster, which is due to the data being in the same scale.

We found the benchmark of this model in the paper showing that we can achieve a masked MAE of 3.60 for DCRNN model using the

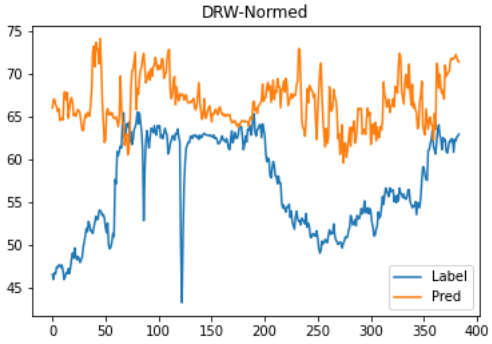


Figure 14: DRW-Normed

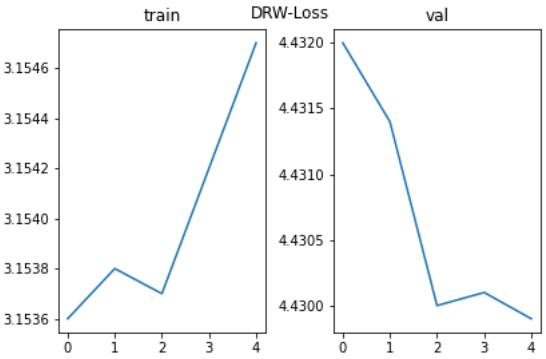


Figure 15: DRW-Loss

Random-Walk Markov Matrix in the diffusion convolution layer, which means that if the actual speed is x mile/h, the predicted speed is within the range of (x - 3.6, x + 3.6) miles/h.

7 Future Improvements

There is still room for improvement compared with the benchmark in the paper. For example, we can increase the number of training epochs to reduce the masked MAE loss, because the second model using Random-Walk Markov Matrix had not converged yet. Also, by adding more features to the input data and doing some data augmentation might help. Moreover, introducing Rank Influence Factor Matrix [4] also helps.

The two models were both underfitting, training with more epochs (50 more with early stopping) will yield a better result.

References

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. https://doi.org/10.48550/ARXIV.1312.6203

[2] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf

[3] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. 2019. Spatiotemporal Multi-Graph Convolution Network for Ride-Hailing Demand Forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, 01 (Jul. 2019), 3656–3663. https://doi.org/10.1609/aaai.v33i01.33013656

[4] Yujun Huang, Yunpeng Weng, Shuai Yu, and Xu Chen. 2019. Diffusion Convolutional Recurrent Neural Network with Rank Influence Learning for Traffic Forecasting. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 678–685. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00096

[5] 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)*.

[6] Marco Lippi, Matteo Bertini, and Paolo Frasconi. 2013. Short-Term Traffic Flow Forecasting: An Experimental Comparison of Time-Series Analysis and Supervised Learning. *IEEE Transactions on Intelligent Transportation Systems* 14, 2 (2013), 871–882. https://doi.org/10.1109/TITS.2013.2247040

[7] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting. *CoRR* abs/1709.04875 (2017). arXiv:1709.04875 http://arxiv.org/abs/1709.04875

[8] Ling Zhao, Yujiao Song, Min Deng, and Haifeng Li. 2018. Temporal Graph Convolutional Network for Urban Traffic Flow Prediction Method. *CoRR* abs/1811.05320 (2018). arXiv:1811.05320 http://arxiv.org/abs/1811.05320

8 Contributions

1. Data sourcing: Obtain the data between January 2020 and August 2020 from PeMS. (Fan Wu)
2. Data cleaning: Perform exploratory data analysis and aggregate data into a fixed time window. (Fan Wu)
3. Feature engineering: Pick features like speed, latitude, and longitude to create a bidirectional graph for later usage. (Fan Wu)
4. Modeling: Create the DCRNN model using diffusion convolutional layer, sequence-to-sequence architecture, and scheduled sampling techniques. (Rong Li)
5. Training & Evaluation: Load dataset, training model on Colab, test and evaluate the result. (Yifei Liu)

Overall contribution for each member(in percentage):
Fan Wu(30%),Rong Li(40%), Yifei Liu(30%)

Appendix

Pages below are the training and evaluation code running on the Google Colab.

[ ]:
```
from google.colab import drive
drive.mount('/content/gdrive/', force_remount=True)
```

[ ]:
```
!git clone https://github.com/ClimbWorm/CS512-22fall.git project
%cd project/Dataset
!unzip pems_all_2022_nonnan.h5.zip
%cd ..
```

[ ]:
```
!mkdir -p /content/gdrive/MyDrive/CS512/checkpoint
```

[ ]:
```
!git init
```

[ ]:
```
!git remote add origin https://github.com/ClimbWorm/CS512-22fall.git
```

[ ]:
```
!git pull origin main
```

Macro Configs

[ ]:
```
# Configs
# %cd project
from scripts.utils import EarlyStopper, DEVICE
ES = EarlyStopper(patience=3, min_delta=0)
CP_PATH = "/content/gdrive/MyDrive/CS512/checkpoint"
HORIZON = 12
INPUT_DIM = 2
OUTPUT_DIM = 1
SEQ_SIZE = 12
BATCH_SIZE = 64
CL_DECAY_STEPS = 2000
LR = 0.007
EPS = 1e-4
STEPS = (8, 16, 24, 32)
GRU_ARGS = {
    "max_diffusion_step": 2,
    "cl_decay_steps": CL_DECAY_STEPS,
    "filter_type": "laplacian",
    # "filter_type": "dual_random_walk",
```

```
    "num_nodes": 319,
    "num_rnn_layers": 2,
    "rnn_units": 64
}
```

Load dataset

[ ]:
```
from scripts.utils import load_dataset
train, val, test, std, mean = load_dataset("Dataset/pems_all_2022_nonnan.h5")
```

Train model

[ ]:
```
# train
from scripts.utils import gen_adj_mat
import pandas as pd
with open("Dataset/sensor_id.txt", "r") as f:
    sensor_ids = [int(sid) for sid in f.read().strip().split(",")]
dist = pd.read_csv("Dataset/distances_bay_2017.csv")
_, adj_mat = gen_adj_mat(dist, sensor_ids)
```

[ ]:
```
# from scripts.scheduler import TrainScheduler
scheduler = TrainScheduler(adj_mat, train, val, test, std, mean,␣
↪input_dim=INPUT_DIM, output_dim=OUTPUT_DIM, horizon=HORIZON,␣
↪seq_size=SEQ_SIZE,
                           num_sensors=len(sensor_ids), cp_path=CP_PATH,␣
↪batch_size=BATCH_SIZE,
                           cl_decay_steps=CL_DECAY_STEPS, gru_args=GRU_ARGS,
                      trained_epoch=23)
```

[ ]:
```
scheduler.train(early_stop=ES)
```

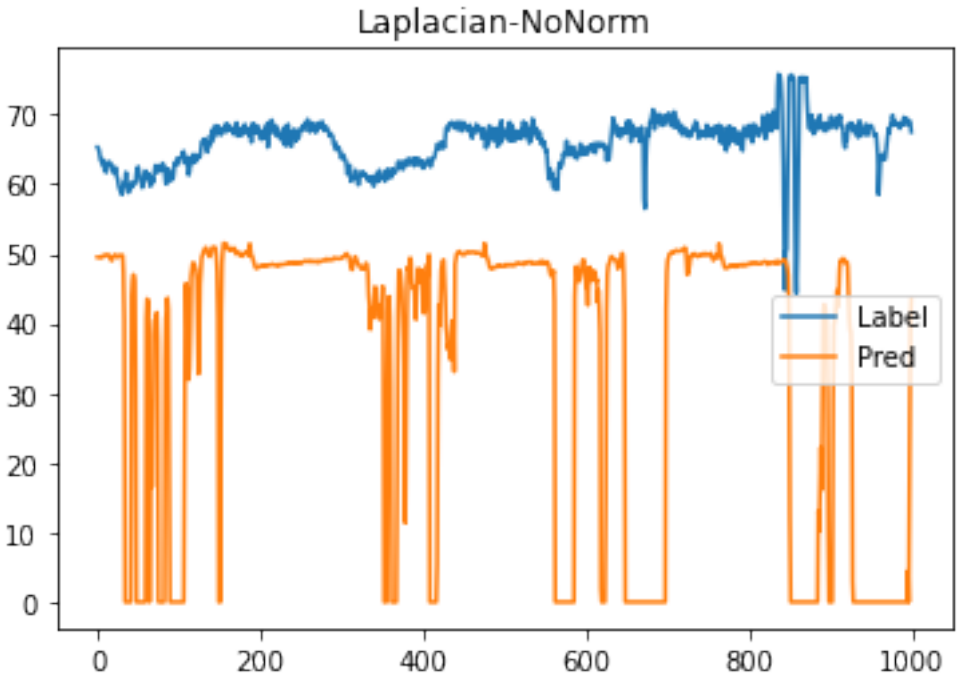
[ ]:
```
label, pred = scheduler.predict("val")
def transform(data, std, mean):
    return (data - mean) / std
pred = transform(pred, scheduler.std, scheduler.mean)
```

```
Predict on val: 81it [00:58,  1.38it/s]
```

[ ]:
```
import matplotlib.pyplot as plt
plt_pred = pred[-1][2000:3000, 222]
plt_pred[plt_pred < 0] = 0
plt.plot(label[-1][2000:3000, 222], label="Label")
plt.plot(plt_pred, label="Pred")
plt.legend()
plt.title("Laplacian-NoNorm")
plt.savefig("./Laplacian-NoNorm.png")
from google.colab import files
files.download("./Laplacian-NoNorm.png")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



[ ]:
```
%load_ext tensorboard
%tensorboard --logdir="./runs"
```

[ ]:
```
!zip -r runs.zip ./runs
```

```
  adding: runs/ (stored 0%)
  adding: runs/Dec12_06-24-28_a44d66a0ec58/ (stored 0%)
  adding: runs/Dec12_06-24-28_a44d66a0ec58/events.out.tfevents.1670826270.a44d66
a0ec58.115.0 (deflated 68%)
```

[ ]:
```
from google.colab import files
files.download("runs.zip")
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```