香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

School of Data Science

ERG3020: Web Analytics and Intelligence

Course Project Report

A Graph Prediction Method based on LSTM and Personalized PageRank: LSTM-PageRank

---

Author:

Wang Juntao 118010294

Lin Jiejie 117010154

Wen Zhanghao 118010322

May, 2021

Abstract

Graph networks has received significant attention because of its numerous applications on real-life problems. This paper provides a concise overview of studies on road network pertinent to current main graph prediction methods, concluding their superiority and limitations. Inspired by previous studies, we build a new algorithm model which combines the personalised PageRank and LSTM algorithm in predicting the time-varying flows in the road network. We simulated 365 days' traffic flow data in a graph model with some rules. By controlling the scales of parameters of LSTM and LSTM part of LSTM-PageRank at the same level, we trained these two models to do prediction on the simulating data. The experimental results show that LSTM-PageRank model can perform better than LSTM, however, without adding additional parameters. All the improvements are based on the propagation scheme, PPNP.

Keywords: *LSTM-PageRank, LSTM, personalized PageRank, GCN, personalized propagation of neural prediction, flow data, graph, neural network.*

Contents

Chapter 1

Introduction

1.1 Background

Graph networks are ubiquitous in real world and are useful in solving a series of real-life problems. They have wild applications in social network connection, commodities recommendation, web pages recommendation, traffic optimization, etc. Some groups of these graph networks have the characteristics that not only contains the links between nodes that form the edges in the graph, but also features that vary from time to time at each node. Focusing on these problems with random walks of features from time to time and from nodes to nodes, we found that the road network is a perfect case for us to study with.

The road network in the traffic system of a city has two important features. One is its topology feature that composes the physical structure forming links between roads, and the other is its time-varying feature in traffic flow (Guo et al. 2016). To forecast the traffic states in roads at different times, we need to both construct the linked network by its topology feature and make predictions on its time-varying traffic flow. The goal of traffic forecasting is to predict future traffic states in the traffic network given the physical roadway network and a sequence of historical traffic states (Cui et al. 2019). In this paper we combine the PageRank and LSTM algorithm in predicting the time-varying

flows in the road network.

1.2 Literature Review

Previous studies on traffic forecasting used PageRank algorithms to estimate the importance at each road and Neural Network (NN) based algorithms to make predictions on the states in the network.

The PageRank algorithm, proposed by Page et al. (1999), is the algorithm of measuring the importance of website pages based on the in-links and out-links of each node. Previous works have shown how PageRank could be used as an efficient way of urban traffic optimization. According to Pop and Dobre (2012), PageRank algorithm can be efficiently applied to urban traffic road networks and demonstrates that the usage of this method to compute the centrality of each segment of road can improve the traffic and reduce congestion. However, the PageRank algorithm is limited for which, it is hard to capture the time-varying features of the road network and it can not make predictions on the flows of each road.

Accurate forecasting of time-varying traffic flow is a challenging project for its time-space complexity. Deep learning models have shown their powerful abilities in capturing nonlinear spatiotemporal effects for traffic forecasting (Polson and Sokolov 2017). With the rise of Neural Network (NN), many NN-based methods have been adopted for traffic forecasting like the works done by Park and Rilett (1999). Besides, Recurrent Neural Networks (RNNs) are especially suitable to capture the temporal evolution of traffic flow because of the self-circulation mechanism, which grants RNN the capability of learning complex time series (Wang et al. 2019). However, these models only consider the temporal variation of traffic state and neglect spatial dependence. On top of that, many scholars have introduced convolutional neural networks (CNNs) in their models to characterize spatial dependence remarkably. In addition to CNN, graph convolutional network (GCN) is also utilized in traffic forecasting and solved limitations of CNN on non-Euclidean structures (Zhu et al. 2021). However, one of the deficiency in GCN model is that GCN's

performance necessarily deteriorates for a high number of layers or aggregation steps, with the problems of increasing training parameter and oversmoothing (Klicpera et al. 2018).

1.3 Our Project

To solve these issues, we first notice that the GCN is similar to PageRank in that they both converge to the random walk's limit distribution. On top of that, studies done by Klicpera et al. (2018) have shown that an optimized PageRank algorithm can solve the oversmoothing problem of GCN by separating the neural network from the propagation scheme.

Inspired by previous studies, in this paper we are interested in building new algorithms to make accurate forecast on road network, which could be applied to the graphs with features similar to road network. Noticing that compared to conventional RNNs, the Long Short-Term Memory (LSTM) network is able to capture the features of time series within longer time span and thus lead to better performance in traffic forecasting (Zhao et al. 2017), we try to utilize LSTM in our algorithm to optimize traffic forecasting results combining with PageRank.

Chapter 2

Model

2.1 LSTM

LSTM is a variant of RNN, which solves the short-term memory problem of RNN. It can handle the relationship between inputs better. LSTM adds input gates, output gates and forget gates to its repeating unit module to control the flow of information in and out (see Figure 2.1), thereby achieving the function of retaining important information and ignoring unimportant information (Hochreiter and Schmidhuber 1997).

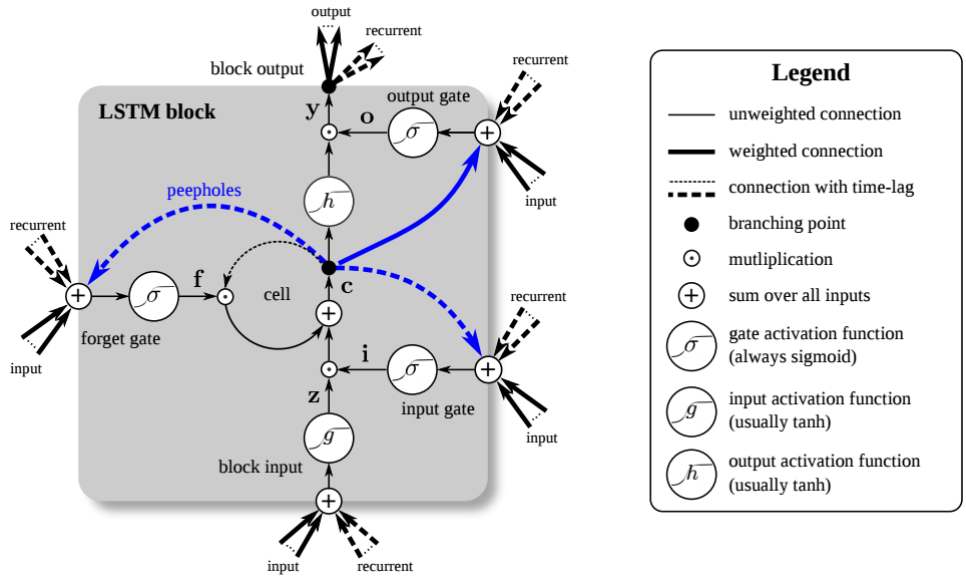

Figure 2.1: LSTM repeating unit module

Define the outputs of three gates as $I^{(t)}$, $F^{(t)}$ and $O^{(t)}$,

$$I^{(t)} = \sigma(U_i x^{(t)} + W_i h^{(t-1)} + Bias_i) \tag{2.1}$$

$$F^{(t)} = \sigma(U_f x^{(t)} + W_f h^{(t-1)} + Bias_f) \tag{2.2}$$

$$O^{(t)} = \sigma(U_o x^{(t)} + W_o h^{(t-1)} + Bias_o) \tag{2.3}$$

where the input $x^{(t)}$ and hidden state of last time $h^{(t-1)}$ times weight matrix of each gate separately. Besides hidden state, LSTM has a cell state which is used to be propagated along LSTM chain. The cell state $c^{(t)}$ at time slot t is defined as

$$c^{(t)} = \tilde{C}^{(t)} \circ I^{(t)} + c^{(t-1)} \circ F^{(t)} \tag{2.4}$$

where

$$\tilde{C}^{(t)} = \tanh(Ux^{(t)} + Wh^{(t-1)} + Bias_h) \tag{2.5}$$

The cell state $c^{(t)}$ goes through activation function tanh and times the value of output gate $O^{(t)}$ by Hadamard product, becomes hidden state at current time step $h^{(t)}$. By projecting hidden state, we can get the output $o^{(t)}$.

$$h^{(t)} = \tanh(c^{(t)}) \circ O^{(t)} \tag{2.6}$$

$$o^{(t)} = \sigma(Vh^{(t)} + Bias_{output}) \tag{2.7}$$

Therefore, the whole feedforward process of LSTM is

$$\begin{aligned} c^{(t)} =& \tanh(Ux^{(t)} + Wh^{(t-1)} + Bias_h) \circ \sigma(U_i x^{(t)} + W_i h^{(t-1)} + Bias_i) \\ &+ c^{(t-1)} \circ \sigma(U_f x^{(t)} + W_f h^{(t-1)} + Bias_f) \\ h^{(t)} =& \tanh(c^{(t)}) \circ \sigma(U_o x^{(t)} + W_o h^{(t-1)} + Bias_o) \\ o^{(t)} =& \sigma(Vh^{(t)} + Bias_{output}) \end{aligned} \tag{2.8}$$

2.2 Personalized PageRank

2.2.1 PageRank

PageRank Algorithm is developed by Page et al. (1999) to rate the importance of web pages objectively and mechanically using the link structure of the web. It uses hyperlinks between web pages to assign weights for each page. By constructing the matrix for transition probability, and by iteration operations, PageRank could derive the limit distribution of the web pages. As a model that contains both directional and numerical information, it could work well in long-time time-series forecasting problem combining with LSTM.

In our problem, the road network is similar to the web network that PageRank is primarily developed for. Specifically, the roads in the road network can be regarded as the nodes and the transaction of cars from one node to another forms the links between nodes. The centrality of a road is the importance of pages in the PageRank algorithm, which can be a measure of popularity of the road on different paths.

2.2.2 Personalized Propagation of Neural Predictions

Previous study done by Klicpera et al. (2018) has proposed a propagation structure in neural network based on personalized PageRank.

The original PageRank derived the stationary distribution via $\pi_{pr} = A_{rw}\pi_{pr}$, with $A_{rw} = AD^{-1}$, where A is the adjacent matrix and D is the degree matrix. Personalized PageRank adds self-loops to the primitive one by introducing the teleport (or restart) probability $\alpha \in (0, 1]$ and the teleport vector i_x on root x. The adaptation form of PageRank, or personalized PageRank, has a new equation for the stationary distribution $\boldsymbol{\pi}_{ppr}(\boldsymbol{i}_x) = (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\boldsymbol{\pi}_{ppr}(i_x) + \alpha\boldsymbol{i}_x$, which can be transformed to the form:

$$\boldsymbol{\pi}_{ppr}(i_x) = \alpha\left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\right)^{-1}\boldsymbol{i}_x \tag{2.9}$$

By replacing $\boldsymbol{i}$ with the identity matrix $\boldsymbol{I}_x$, we can derive fully personalized PageRank matrix $\boldsymbol{\Pi}_{ppr} = \alpha \left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\right)^{-1}$. The elements in the matrix are the influence score of one node to another.

To utilize the scores above, they proposed the Personalized Propagation of Neural Predictions (PPNP) model to generate final predictions. The model equation of PPNP is

$$\boldsymbol{Z}_{\text{PPNP}} = \sigma\left(\alpha\left(\boldsymbol{I}_n - (1-\alpha)\hat{\tilde{\boldsymbol{A}}}\right)^{-1}\boldsymbol{H}\right), \quad \boldsymbol{H}_{i,:} = f_\theta\left(\boldsymbol{X}_{i,:}\right) \tag{2.10}$$

where $\boldsymbol{X}$ is the feature matrix, f_θ is a neural network with parameter θ, and $\boldsymbol{H}$ is the output of NN. The propagation process in shown in Figure 2.2.

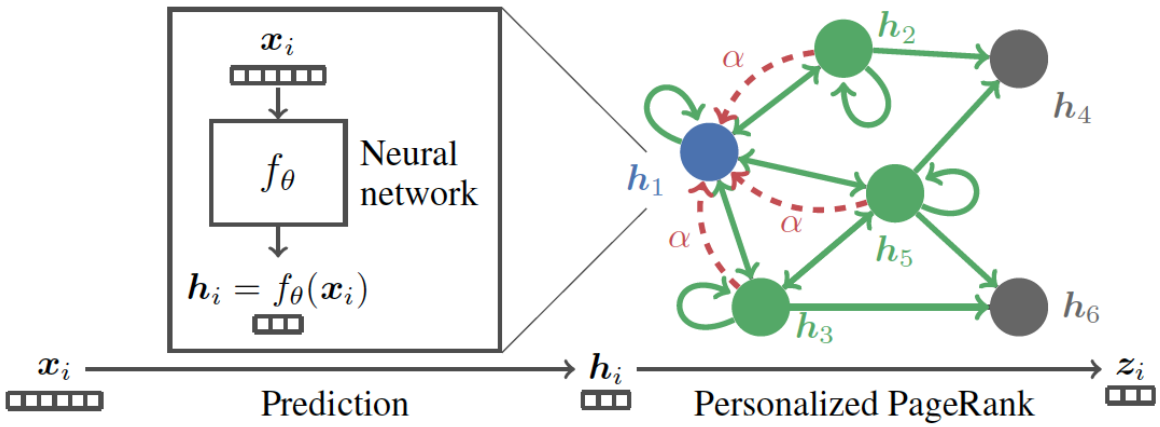

Figure 2.2: Personalized propagation of neural prediction

2.3 LSTM-PageRank

Based on the works of PPNP model, we propose the LSTM-PageRank model in this paper. We utilize the idea in PPNP to use the output derived from NN and then apply the output matrix in personalized PageRank model. Different from previous work, we make further exploration on using LSTM to make the prediction on the output of Neural Network.

The output matrix $\boldsymbol{H}$ in formula 2.10 is now derived from LSTM process with the

formula 2.8. By multiplying the value of hidden state $h^{(t)}$ with the weight matrix V, $Vh^{(t)}$ is the final output that composes the elements in the matrix $\boldsymbol{H}$. Therefore, we derive the whole feed forward process of LSTM-PageRank in the formula 2.11. And the framework of our model is shown in Figure 2.3.

$$\begin{aligned} c^{(t)} =& \tanh(Ux^{(t)} + Wh^{(t-1)} + Bias_h) \circ \sigma(U_i x^{(t)} + W_i h^{(t-1)} + Bias_i) \\ &+ c^{(t-1)} \circ \sigma(U_f x^{(t)} + W_f h^{(t-1)} + Bias_f) \\ h^{(t)} =& \tanh(c^{(t)}) \circ \sigma(U_o x^{(t)} + W_o h^{(t-1)} + Bias_o) \\ o^{(t)} =& \sigma\left(\alpha\left(\boldsymbol{I}_n - (1-\alpha)\hat{\hat{\boldsymbol{A}}}\right)^{-1} Vh^{(t)} + Bias_{output}\right) \end{aligned} \tag{2.11}$$

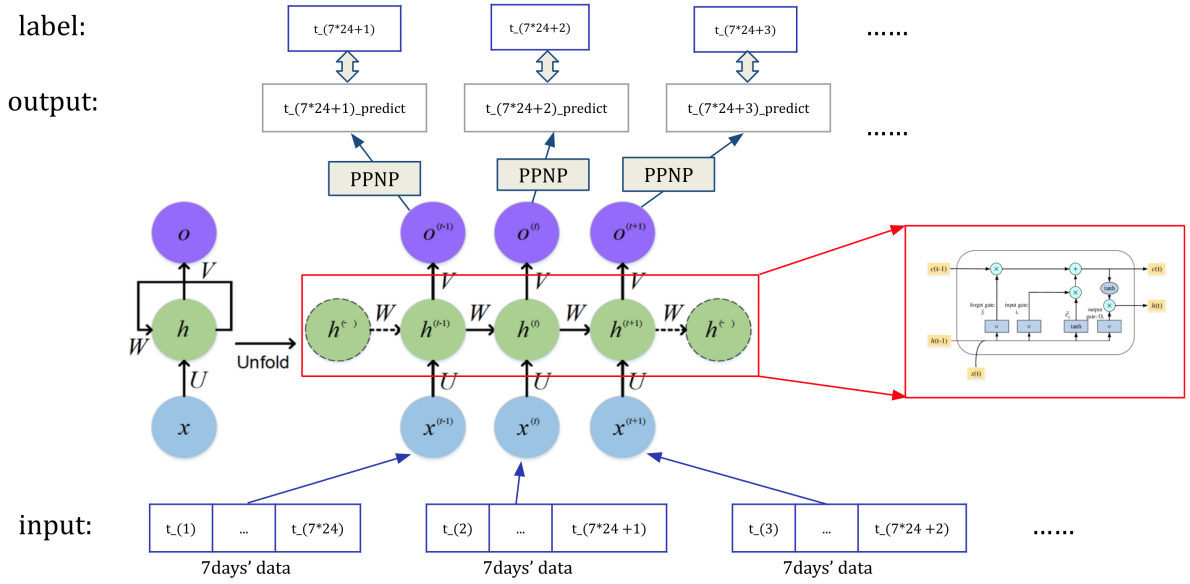

Figure 2.3: Framework of LSTM-PageRank

Chapter 3

Simulation Experiment

3.1 Experimental Data Generation

Due to the lack of certain kind of data, we decided to generate experimental data by simulating in the context of traffic which can be modeled as a graph.

3.1.1 Data Simulation Design

Traffic data of a specific region are needed for the project. For the car movement happened at each time t, we have made several assumptions to keep data robust:

1) The region is a rectangle, and it is divided into 9 parts.

2) The region is considered as an internal conservation, whose car number inside would be fixed as 1000. It could use the following equation to express:

$$\sum_{i=1}^{9} n_i^t = 1000 \tag{3.1}$$

where n_i^0 defines the car number of region i at time t. For the initial time, 1000 cars are randomly distributed into this region.

3) At each time t, internal movements happen, which means that cars in any part could move to other adjacent part. The car exchange only happens between region

parts that are edge-crossed, but not diagonal one.

4) The car movement happens once an hour in a day. The moving volume range is changeable in different time interval in a day.

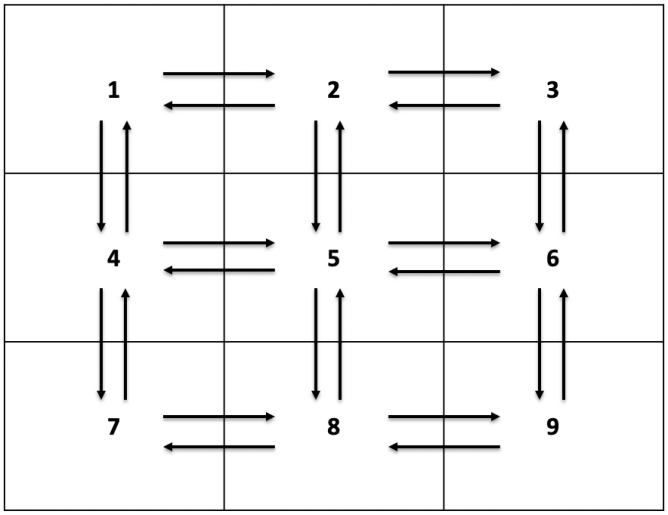

Figure 3.1: Car movements in region parts.

The region movement is illustrated in Figure 3.1. The number of cars enter from region part i to j at time t is defined as O_{ij}^t and it $\in [-10, 0]$, and the number of cars exit from region part i to j at time t is defined as I_{ij}^t and it $\in [0, 10]$. The range of them is correlated with time interval in a day, which is shown in Table 3.1.

time interval	car volume range
$[0, 6)$	$[0, 2]$
$[6, 12)$	$[2, 5]$
$[12, 18)$	$[5, 8]$
$[18, 24)$	$[8, 10]$

Table 3.1: Car volume range's correlation with time interval in a day.

It should be noted that I_{ij}^t=O_{ji}^t. The following equation is used to obtain each region part's car number at time t:

$$n_i^t = n_i^{t-1} + I_{ij} + O_{ij} \tag{3.2}$$

To symbolize the function, we use $delta_{ij}^t$ to define the net inflow number of internal movement. So the equation becomes:

$$n_i^t = n_i^{t-1} + delta_{ij}^t \tag{3.3}$$

where $delta_{ij}^t \in [-10, 10]$.

In all, there should be two data tables generated. One is a table listing each time unit's car number at each region parts, another is a table listing 12 columns of $delta_{ij}^t$.

3.1.2 Code Implementation

There are three parts mainly to implement: data initialization, car movement in a time unit, and data output as visual table file. The detailed code is shown in Appendix B.1.

In the first step, random library's sample method is used to randomly create 8 different numbers in the interval of (0, total). Then start and end is added to the array and sorted. By using the listing comprehension, each interval between two neighbored number is calculated, and car number array at time 0 is obtained.

To make data easier to store, two dictionaries are created: data_n and data_delta. Their keys are integer value of time index, and values are array of certain values. The data_n is for storing car distribution at each time unit, whose value is an array of 9 part's car number. The data_delta is to store region's internal movement, whose value is an array of $delta_{ij}^t$. The array has length of 12, whose elements are chosen to be $delta_{12}$, $delta_{23}$, $delta_{14}$, $delta_{25}$, $delta_{36}$, $delta_{45}$, $delta_{56}$, $delta_{47}$, $delta_{58}$, $delta_{69}$, $delta_{78}$, $delta_{89}$. Contents in data_delta's array at time 0 are full of 0. The data_delta array's visual relation is shown in Figure 3.2

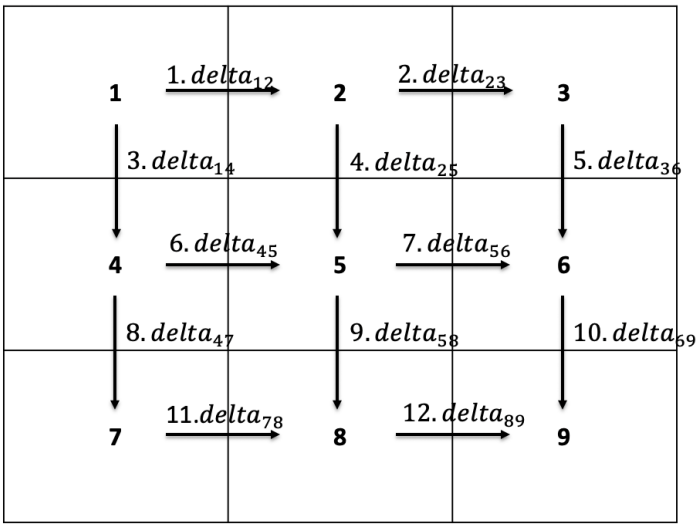

Figure 3.2: Region part's internal movement data sequence in array data_delta.

The initialized data at time 0 are already prepared, then car internal movement calculation should be done in each iteration. As nine region part has 12 connection lines and each line has two-sided direction, to avoid mixing wrong and to make clear, nine region parts are partitioned into three conditions by their connection parts' number. In this three conditions, we consider each part as outflow central. The first condition is for part 5 only. It is located in the middle and would transfer car with four parts. The second condition is for part 1, 3, 7, 9, which have two neighbors to transfer. The last condition is for part 2, 4, 6, 8. For all conditions, the transferred car number $delta_{ij}$ would be randomly decided first by rand_array() function. As the outflow region part could not outflow car volume larger than its capacity, rand_array() function would consider whether region part's car number n_i first:

1) If $n_i = 0$, then this region part is unable to outflow to other neighbors. The $delta_{ij}$ would be 0.

2) If n_i is larger than 10, then it will consider the time now, and to do the random selection in its specific range.

3) If n_i is smaller than 10, then it will randomly choose a number that is in the range of $[0, n_i]$.

After deciding the value of outflow volume, basing on its direction, the outflow value would be added to certain net inflow array element shown in Figure 3.2, and the overall model is attached in Figure A.3 which reveals transition matrix mentioned in Chapter.3.2 as well. In each iteration, we would use the value of $data\_n[t-1]$ and $data\_delta[t]$ to get new data array $data\_n[t]$.

After all have been done, we come to the final step. Csv package is imported to export *data_n* and *data_delta* dictionaries as a csv file.

3.2 Result

We first simulate experimental data along 365 days (i.e. $365 * 24$ time steps(hours)). Then we generate the transition matrix $\hat{\tilde{\boldsymbol{A}}}$ by counting the frequency of hopping between nodes. The details of how to generate the transition matrix $\hat{\tilde{\boldsymbol{A}}}$ and the value of transition matrix will be shown in Appendix A.1. The code of transition matrix generation can be found in Appendix B.2.

Using the transition matrix we generated, we do a pre-experiment to choose a suitable restart probability $\alpha \in (0, 1)$. More precisely, we try $\alpha \in \{0.1, 0.2, ..., 0.8, 0.9\}$ and find the most suitable restart probability in this set is 0.3.

Then we run the LSTM and LSTM-PageRank algorithm on the data. We use 7 days' data (i.e. 168 hours' data) to predict next one hour data. The data is normalized by MinMaxScaler, where

$$\tilde{y_i} = \frac{y_i - min(y)}{max(y) - min(y)} \tag{3.4}$$

The data is divided into three parts: 70% as training data, 20% as validating data and 10% as testing data. Some hyperparameters are shown in Table 3.2.

Hyperparameters	LSTM	LSTM-PageRank
Epochs	1000	1000
Learning rate	0.02	0.02
Hidden size of LSTM	3	3
Number of LSTM layers	1	1
L2-regulization α	1e-5	1e-5
Restart probability α	-	0.3

Table 3.2: Hyperparameters of LSTM and LSTM-PageRank.

We use mean absolute percentage error(MAPE) as the error metric, the formula of MAPE is

$$\text{MAPE} = \frac{1}{N_j} \sum_{i=1}^{N_j} \frac{|y_i - \hat{y}_i|}{y_i} \tag{3.5}$$

where y_i is the true value, $\hat{y}_i$ is the predicting value. However, once the true value y_i is small, the absolute percentage error will be very large to make MAPE be much larger than the true level. Thus, we also choose median absolute percentage error(MedAPE) as the error metric, which is defined by

$$\text{MedAPE} = Median \left\{ \frac{|y_i - \hat{y}_i|}{y_i} \mid i = 1, 2, ... \right\} \tag{3.6}$$

The result of 365 days is shown in Table 3.3. And all the plots can be found in Appendix A.2. The code implementation of models and experiments can be seen in Appendix B.3.

As you can see, LSTM-PageRank has a better perfermance than LSTM, from the aspects of no matter MAPE or MedAPE.

Node	MAPE		MedAPE	
	LSTM-PageRank	**LSTM**	**LSTM-PageRank**	**LSTM**
1	0.1059	0.15685	0.10311	0.15786
2	0.17418	0.25426	0.17129	0.25679
3	0.47144	0.7036	0.40282	0.64642
4	0.09486	0.18167	0.04973	0.09254
5	0.07083	0.16045	0.07303	0.15368
6	1.02053	1.06514	0.86291	1.00815
7	0.36096	0.60889	0.19763	0.36129
8	0.03971	0.04774	0.0298	0.03932
9	0.14242	0.2812	0.07706	0.21888
Average	**0.2756477778**	**0.3844222222**	**0.2185977778**	**0.3261033333**

Table 3.3: Result (365 days)

Conclusion

In this paper, we proposed a graph prediction method based on LSTM and personalized PageRank. We are inspired by PPNP proposed by Klicpera et al. (2018). Thus we try to construct framework of the combination of LSTM and PPNP, LSTM-PageRank. This model utilized the decoupling idea of PPNP, divided the whole graph flow data prediction to flow data prediction by LSTM and graph propogation by personalized PageRank. Compared to traditional PageRank, personalized PageRank can use the information from a large scale and adjustable neighborhood for prediction of each node by the restart probability.

Due to the lack of data, we simulated experimental data in the context of road network. We, to some extent, simulated the time characteristics of traffic data flow. We generated 365 days' data and applied LSTM and LSTM-PagrRank on the data. We use MAPE and MedAPE as the error metric. By setting the same hyper parameters, we control the ability of LSTM and LSTM part of LSTM-PageRank at an approximately same level. As shown by the results, LSTM-PageRank performs better with a suitable restart probability on our data than LSTM.

For future work, we may want to apply LSTM-PageRank on data with more scales of granularity or more scales of length, to see how better (or worse) LSTM-PageRank performs than LSTM. It would be also interesting to combine more complex framework with LSTM-PageRank e.g. attention, bi-direction. And there may be more sophisticated and useful propagation scheme other than personalized PPNP.

References

Cui, Zhiyong, Henrickson, Kristian, Ke, Ruimin, and Wang, Yinhai (2019). "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting". In: *IEEE Transactions on Intelligent Transportation Systems* 21.11, pp. 4883–4894.

Guo, Haifeng, Zhang, Changshi, Mu, Yuanjie, Zheng, Yayu, and Gong, wei (2016). "Dynamic Sorting Method for Road Network Primary Intersections Based on PageRank Algorithm". In: *Journal of Southwest Jiaotong University* 51.5, pp. 925–930.

Hochreiter, Sepp and Schmidhuber, Jürgen (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Klicpera, Johannes, Bojchevski, Aleksandar, and Günnemann, Stephan (2018). "Predict then propagate: Graph neural networks meet personalized pagerank". In: *arXiv preprint arXiv:1810.05997*.

Page, Lawrence, Brin, Sergey, Motwani, Rajeev, and Winograd, Terry (1999). *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab.

Park, Dongjoo and Rilett, Laurence R (1999). "Forecasting freeway link travel times with a multilayer feedforward neural network". In: *Computer-Aided Civil and Infrastructure Engineering* 14.5, pp. 357–367.

Polson, Nicholas G and Sokolov, Vadim O (2017). "Deep learning for short-term traffic flow prediction". In: *Transportation Research Part C: Emerging Technologies* 79, pp. 1–17.

Pop, Florin and Dobre, Ciprian (2012). "An efficient pagerank approach for urban traffic optimization". In: *Mathematical Problems in Engineering* 2012.

Wang, Xiangxue, Xu, Lunhui, and Chen, Kaixun (2019). "Data-driven short-term forecasting for urban road network traffic based on data processing and LSTM-RNN". In: *Arabian Journal for Science and Engineering* 44.4, pp. 3043–3060.

Zhao, Zheng, Chen, Weihai, Wu, Xingming, Chen, Peter CY, and Liu, Jingmeng (2017). "LSTM network: a deep learning approach for short-term traffic forecast". In: *IET Intelligent Transport Systems* 11.2, pp. 68–75.

Zhu, Jiawei, Wang, Qiongjie, Tao, Chao, Deng, Hanhan, Zhao, Ling, and Li, Haifeng (2021). "AST-GCN: Attribute-Augmented Spatiotemporal Graph Convolutional Network for Traffic Forecasting". In: *IEEE Access* 9, pp. 35973–35983.

Appendix A

Appendix A

A.1 Transition Matrix and Graph (365 days)

After generating the experimental data, we count the $delta_{ij}^{t}$ through time to get the frequency of hopping between nodes. Then we can form a transition frequency matrix $\tilde{\boldsymbol{A}}$. The result is shown in FigureA.1. The entry in the first row and second column is $\tilde{a}_{12} = 3105$, which means there are 3105 hops from Node 1 to Node 2 during the whole process.

	0	1	2	3	4	5	6	7	8
0	5070.000...	3105.000...	0.00000	3170.000...	0.00000	0.00000	0.00000	0.00000	0.00000
1	3114.000...	7699.000...	3094.000...	0.00000	3046.000...	0.00000	0.00000	0.00000	0.00000
2	0.00000	3085.000...	5197.000...	0.00000	0.00000	3054.000...	0.00000	0.00000	0.00000
3	3061.000...	0.00000	0.00000	7507.000...	3137.000...	0.00000	3169.000...	0.00000	0.00000
4	0.00000	3137.000...	0.00000	3094.000...	10313.00...	3044.000...	0.00000	3085.000...	0.00000
5	0.00000	0.00000	3090.000...	0.00000	3126.000...	7788.000...	0.00000	0.00000	3084.000...
6	0.00000	0.00000	0.00000	3142.000...	0.00000	0.00000	5061.000...	3053.000...	0.00000
7	0.00000	0.00000	0.00000	0.00000	3058.000...	0.00000	3095.000...	7739.000...	3104.000...
8	0.00000	0.00000	0.00000	0.00000	0.00000	3094.000...	0.00000	3146.000...	5092.000...

Figure A.1: Transition frequency matrix $\tilde{\boldsymbol{A}}$

Then we normalize the matrix to make the sum of each row is 1. The normalization process can be formulated to

$$\hat{\tilde{a}}_{ij} = \frac{\tilde{a}_{ij}}{\sum_{j=1}^{9} \tilde{a}_{ij}} \tag{A.1}$$

Through this process, we can get the final transition matrix, which is shown in FigureA.2.

Therefore, each entry represents the probability of hops happening between nodes, for example $\hat{\tilde{a}}_{12} = 0.27369$ means the probability of hopping from Node 1 to Node 2 is 0.27369.

	0	1	2	3	4	5	6	7	8
0	0.44689	0.27369	0.00000	0.27942	0.00000	0.00000	0.00000	0.00000	0.00000
1	0.18368	0.45414	0.18250	0.00000	0.17967	0.00000	0.00000	0.00000	0.00000
2	0.00000	0.27214	0.45845	0.00000	0.00000	0.26941	0.00000	0.00000	0.00000
3	0.18140	0.00000	0.00000	0.44489	0.18591	0.00000	0.18780	0.00000	0.00000
4	0.00000	0.13836	0.00000	0.13646	0.45486	0.13426	0.00000	0.13606	0.00000
5	0.00000	0.00000	0.18083	0.00000	0.18294	0.45576	0.00000	0.00000	0.18048
6	0.00000	0.00000	0.00000	0.27914	0.00000	0.00000	0.44963	0.27123	0.00000
7	0.00000	0.00000	0.00000	0.00000	0.17992	0.00000	0.18210	0.45534	0.18263
8	0.00000	0.00000	0.00000	0.00000	0.00000	0.27303	0.00000	0.27762	0.44935

Figure A.2: Transition matrix $\hat{\tilde{\boldsymbol{A}}}$

According to the transition matrix, the graph can be modeled as Figure A.3.

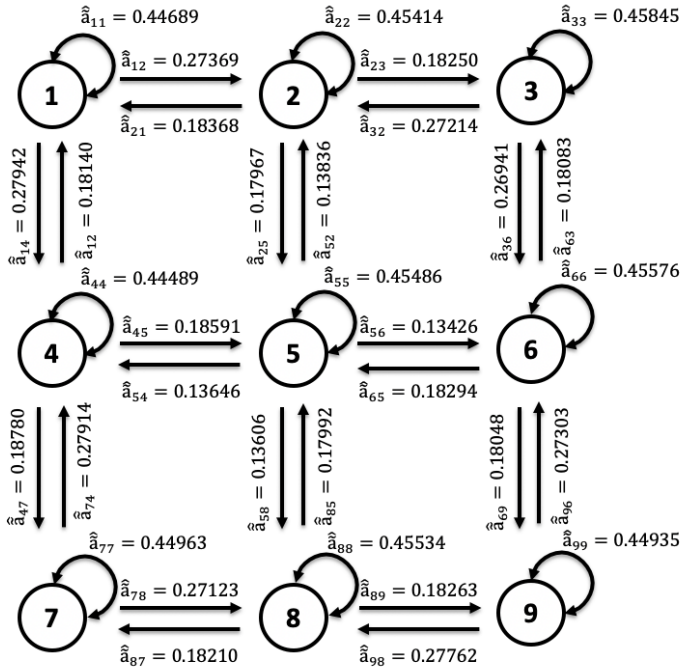

Figure A.3: Graph

A.2 Plot Results

Below are some plot results of our experiments. Blue lines are the ground truth and orange lines are the prediction.

A.2.1 LSTM-PageRank (365 days)

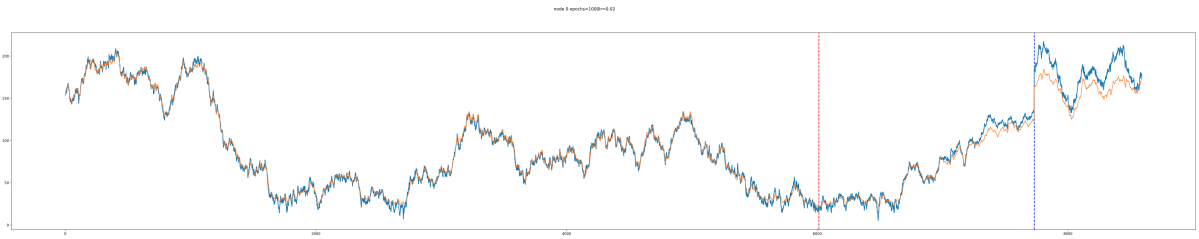

Figure A.4: Result of Node 1 (LSTM-PageRank)

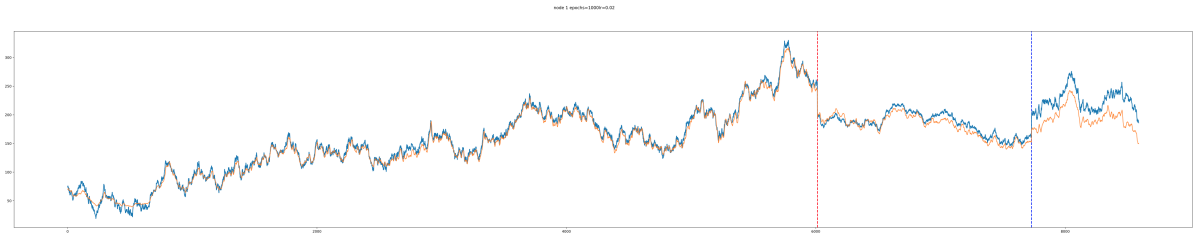

Figure A.5: Result of Node 2 (LSTM-PageRank)

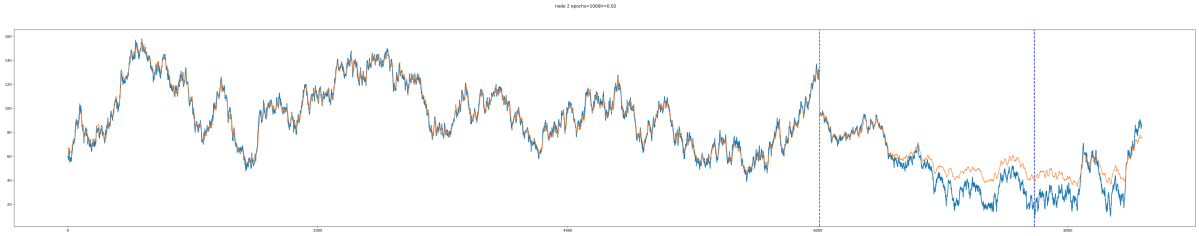

Figure A.6: Result of Node 3 (LSTM-PageRank)

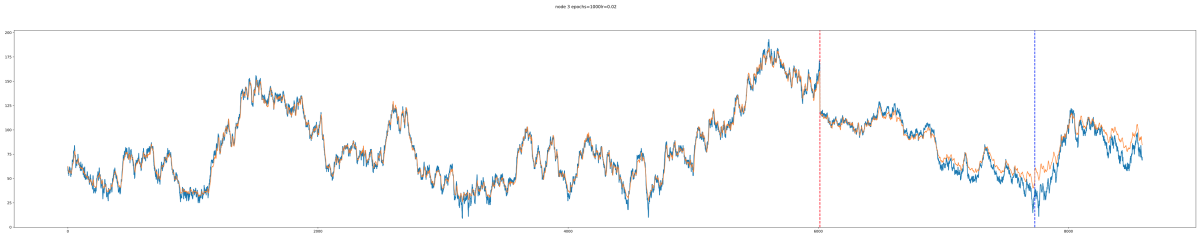

Figure A.7: Result of Node 4 (LSTM-PageRank)

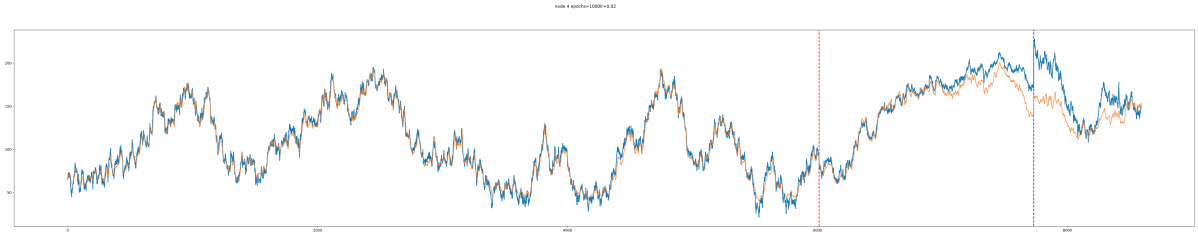

Figure A.8: Result of Node 5 (LSTM-PageRank)

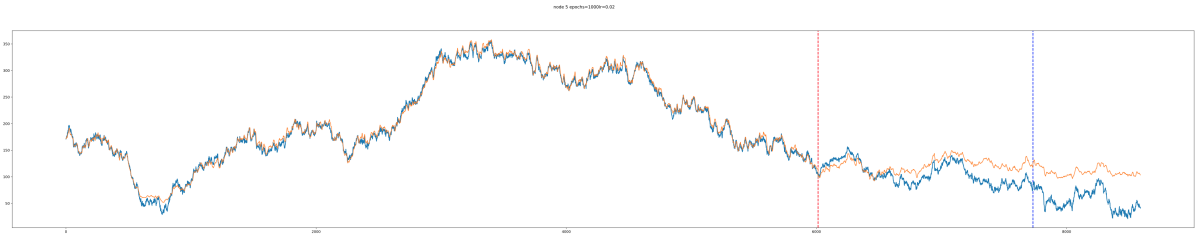

Figure A.9: Result of Node 6 (LSTM-PageRank)

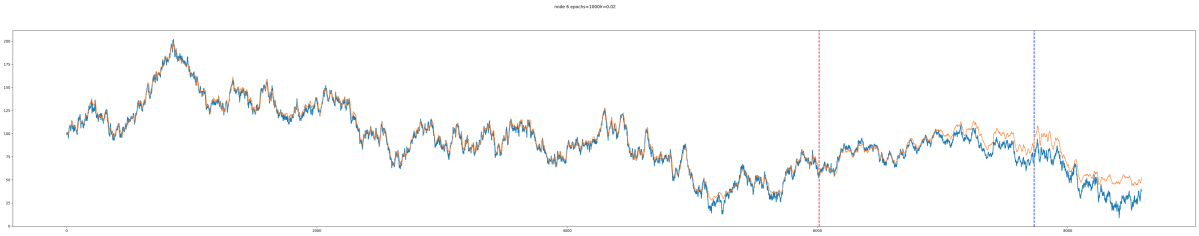

Figure A.10: Result of Node 7 (LSTM-PageRank)

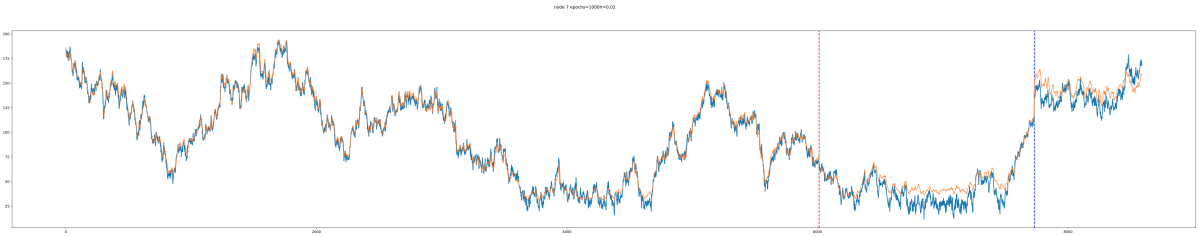

Figure A.11: Result of Node 8 (LSTM-PageRank)

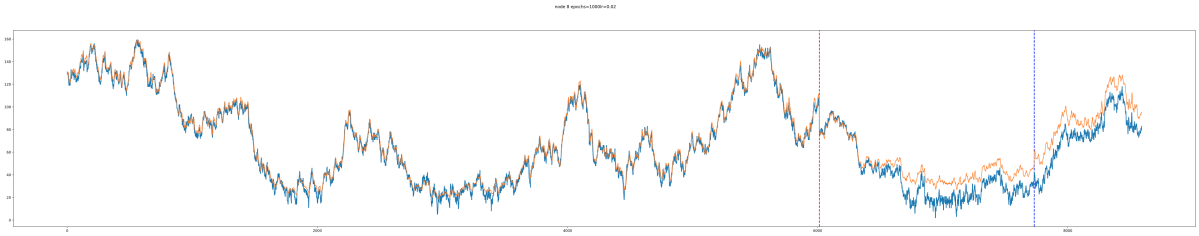

Figure A.12: Result of Node 9 (LSTM-PageRank)

A.2.2 LSTM (365 days)

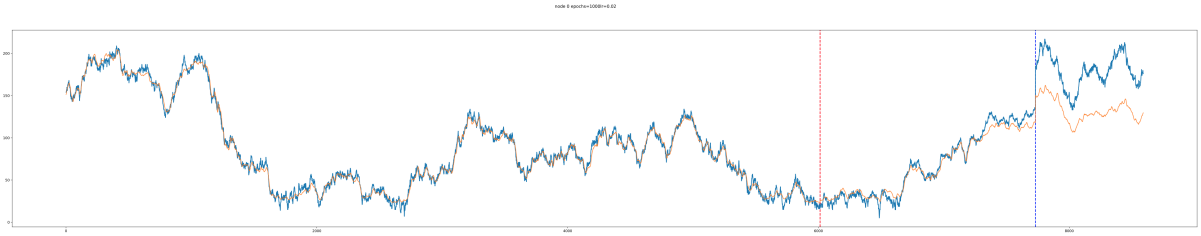

Figure A.13: Result of Node 1 (LSTM)

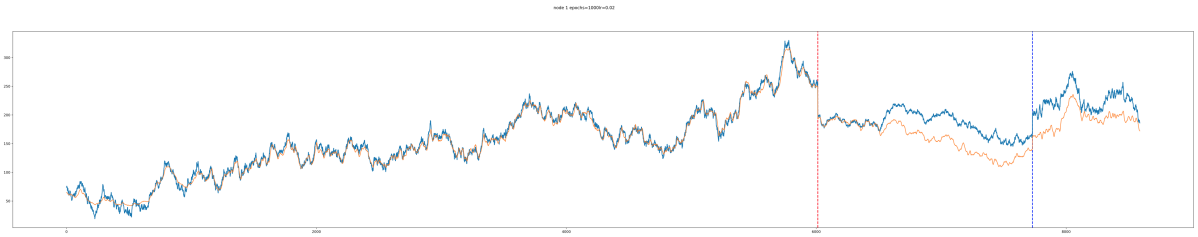

Figure A.14: Result of Node 2 (LSTM)

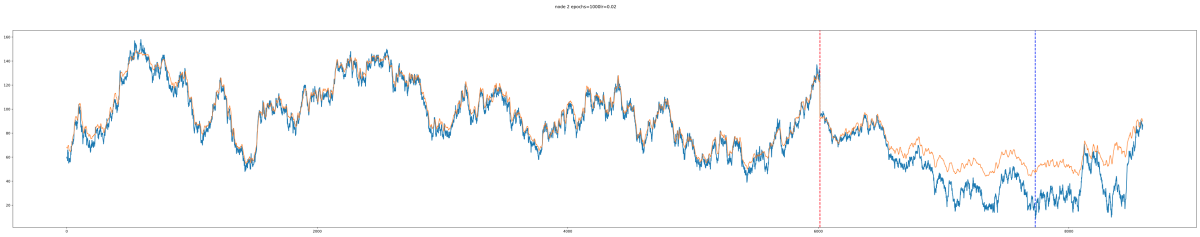

Figure A.15: Result of Node 3 (LSTM)

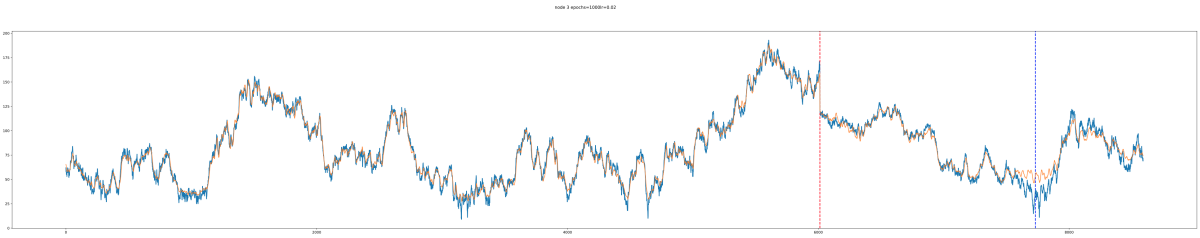

Figure A.16: Result of Node 4 (LSTM)

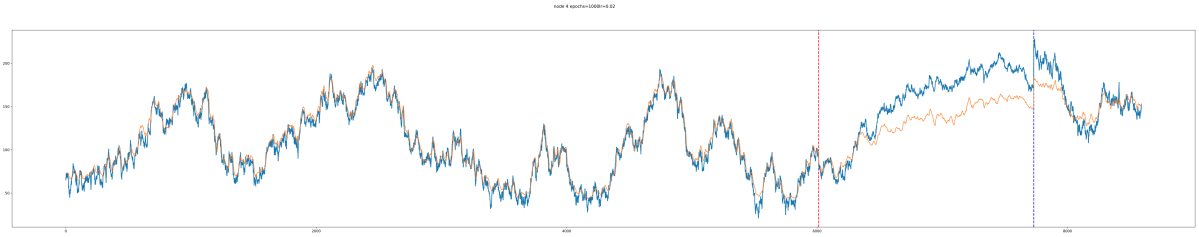

Figure A.17: Result of Node 5 (LSTM)

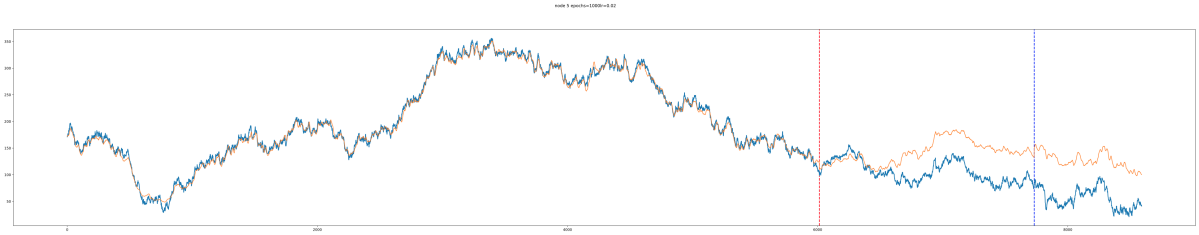

Figure A.18: Result of Node 6 (LSTM)

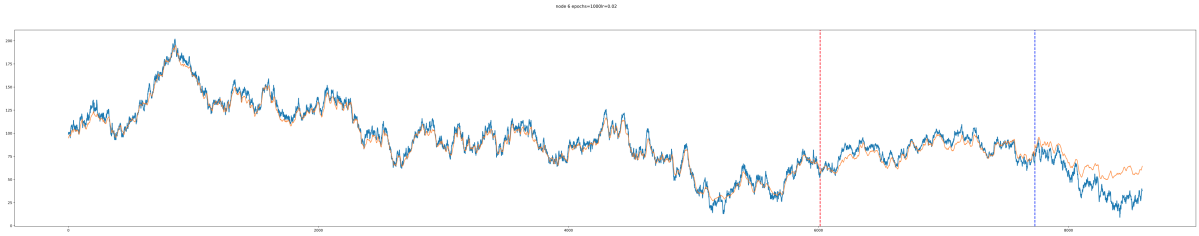

Figure A.19: Result of Node 7 (LSTM)

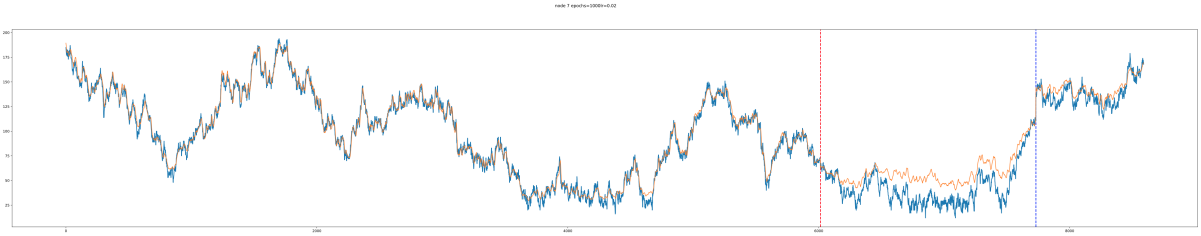

Figure A.20: Result of Node 8 (LSTM)

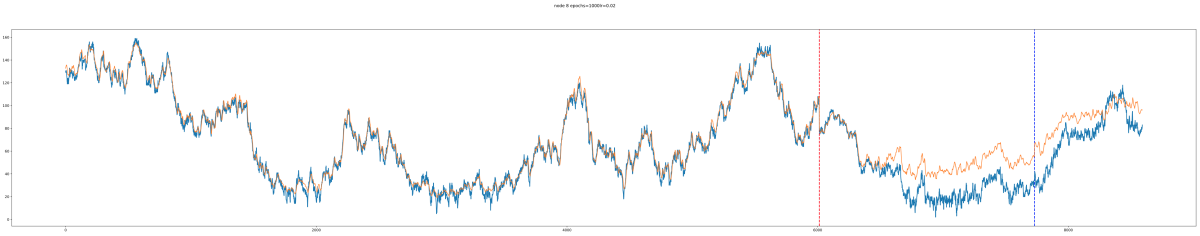

Figure A.21: Result of Node 9 (LSTM)

Appendix B

Appendix B

B.1 Code for Data Generation

Below is python code for data generation.

```
import random
import csv

total_of_first = 1000
region = 9
day = 365
hour = 24

random.seed(9)


def init_array(total, n):
    total = int(total)
    random_ser = random.sample(range(1, total), k=n - 1)
    random_ser.append(0)
    random_ser.append(total)
    random_ser = sorted(random_ser)
    array = [random_ser[i] - random_ser[i - 1] for i in range(1, len(random_ser))]
    return array


def rand_array(total, num, h):
    array = []
    for i in range(num):
```

```
        if total == 0:
            array.append(0)
            continue
        elif total >= 10:
            if h % hour < hour/4:
                array.append(random.randint(0, 2))
            elif (h % hour >= hour/4) & (h % hour < 2*hour/4):
                array.append(random.randint(2, 5))
            elif (h % hour >= 2*hour/4) & (h % hour < 3*hour/4):
                array.append(random.randint(5, 8))
            elif (h % hour >= 3*hour/4) & (h % hour < hour):
                array.append(random.randint(8, 10))
        else:
            array.append(random.randint(0, total))
        total -= array[i]
    return total, array


def rand_move(data_last, h):
    delta_now = [0]*12
    Delta_now = [0]*9
    for id in range(9):
        if id+1 == 5:  # from 5 to four side regions(2, 4, 6, 8)
            total = data_last[id]
            total, array = rand_array(total, 4, h)
            delta_now[3] += array[0]
            delta_now[5] += array[1]
            delta_now[6] -= array[2]
            delta_now[8] -= array[3]

        elif id+1 in [1, 3, 7, 9]:
            total = data_last[id]
            total, array = rand_array(total, 2, h)
            if id+1 in [1, 3]:
                if id+1 == 1:
                    delta_now[id] -= array[0]
                else:
                    delta_now[id-1] += array[0]
                delta_now[id+2] -= array[1]
            elif id+1 in [7, 9]:
                delta_now[id+1] += array[0]
                if id+1 == 7:
                    delta_now[id+4] -= array[1]
                else:
                    delta_now[id+3] += array[1]

        else:  # in region [2, 4, 6, 8]
            total = data_last[id]
```

```
            total, array = rand_array(total, 3, h)
            if id+1 in [2, 4, 6]:
                delta_now[id-1] += array[0]
                if id+1 == 2:
                    delta_now[id] -= array[1]
                    delta_now[id+2] -= array[2]
                elif id+1 == 4:
                    delta_now[id+2] -= array[1]
                    delta_now[id+4] -= array[2]
                elif id+1 == 6:
                    delta_now[id+1] += array[1]
                    delta_now[id+4] -= array[2]
            else:  # region in 8
                delta_now[id+1] += array[0]
                delta_now[id+3] += array[1]
                delta_now[id+4] -= array[2]

    return delta_now, Delta_now


def hourly_move(data_last, n, h):
    side = 2*n*(n-1)
    delta_now, Delta_now = rand_move(data_last, h)

    data_now = [a + b for a, b in zip(data_last, Delta_now)]

    for i in range(3):  # for vertical moving e.g. 1->2, 5->6
        for j in range(2):
            data_now[3*i+j] += delta_now[5*i+j]
            data_now[3*i+j+1] -= delta_now[5*i+j]

    for i in range(2):  # for horizontal moving e.g. 1->4, 5->8
        for j in range(3):
            data_now[3*i+j] += delta_now[5*i+j+2]
            data_now[3*i+j+3] -= delta_now[5*i+j+2]
    # print(data_now)
    return data_now, delta_now, Delta_now


def main():
    data_n = dict()  # length = 9
    data_delta = dict()  # length = 12
    data_Delta = dict()  # length = 9, data_Delta[4]=0

    for i in range(day):
        for j in range(hour):
            if i == 0 and j == 0:
                data_n[0] = init_array(total_of_first, region)
```

```
                data_delta[0] = [0]*12
                data_Delta[0] = [0]*9
            else:
                data_n[hour * i + j], data_delta[hour * i + j], data_Delta[hour * i +
                ↪  j] = hourly_move(data_n[hour * i + j - 1], region, j)

    export_csv(data_n, data_delta, data_Delta)


def export_csv(data_n, data_delta, data_Delta):
    with open("./data/%d_data_n.csv" % day, "w") as csv_file:
        writer = csv.writer(csv_file)
        for key, value in data_n.items():
            writer.writerow([key, value[0], value[1], value[2], value[3],
                             value[4], value[5], value[6], value[7], value[8]])

    with open("./data/%d_data_delta2.csv" % day, "w") as csv_file:
        writer = csv.writer(csv_file)
        for key, value in data_delta.items():
            writer.writerow([key, value[0], value[1], value[2], value[3],
                             value[4], value[5], value[6], value[7], value[8],
                             ↪  value[9], value[10], value[11]])

    with open("./data/%d_data_Delta1.csv" % day, "w") as csv_file:
        writer = csv.writer(csv_file)
        for key, value in data_Delta.items():
            writer.writerow([key, value[0], value[1], value[2], value[3],
                             value[4], value[5], value[6], value[7], value[8]])


main()
```

B.2 Code for Transition Matrix Generation

Below is python code for transition matrix generation.

```
import numpy as np

day = 365

delta = np.loadtxt("./data/%d_data_delta2.csv" % day, usecols=list(range(1, 13)),
↪  delimiter=",")
```

```
6
7  transition = np.zeros([9, 9])
8
9  transition[0, 0] = np.sum(delta[:, 0] == 0) + np.sum(delta[:, 2] == 0)
10 transition[1, 1] = np.sum(delta[:, 0] == 0) + np.sum(delta[:, 1] == 0) +
   ↪  np.sum(delta[:, 3] == 0)
11 transition[2, 2] = np.sum(delta[:, 1] == 0) + np.sum(delta[:, 4] == 0)
12 transition[3, 3] = np.sum(delta[:, 2] == 0) + np.sum(delta[:, 5] == 0) +
   ↪  np.sum(delta[:, 7] == 0)
13 transition[4, 4] = np.sum(delta[:, 3] == 0) + np.sum(delta[:, 5] == 0) +
   ↪  np.sum(delta[:, 6] == 0) + np.sum(delta[:, 8] == 0)
14 transition[5, 5] = np.sum(delta[:, 4] == 0) + np.sum(delta[:, 6] == 0) +
   ↪  np.sum(delta[:, 9] == 0)
15 transition[6, 6] = np.sum(delta[:, 7] == 0) + np.sum(delta[:, 10] == 0)
16 transition[7, 7] = np.sum(delta[:, 8] == 0) + np.sum(delta[:, 10] == 0) +
   ↪  np.sum(delta[:, 11] == 0)
17 transition[8, 8] = np.sum(delta[:, 9] == 0) + np.sum(delta[:, 11] == 0)
18
19 n1 = []
20 for i in range(12):
21     n1.append(np.sum(delta[:, i] > 0))
22
23 n2 = []
24 for i in range(12):
25     n2.append(np.sum(delta[:, i] < 0))
26
27 transition[0, 1] = n2[0]
28 transition[1, 2] = n2[1]
29 transition[0, 3] = n2[2]
30 transition[1, 4] = n2[3]
31 transition[2, 5] = n2[4]
32 transition[3, 4] = n2[5]
33 transition[4, 5] = n2[6]
34 transition[3, 6] = n2[7]
35 transition[4, 7] = n2[8]
36 transition[5, 8] = n2[9]
37 transition[6, 7] = n2[10]
38 transition[7, 8] = n2[11]
39
40 transition[1, 0] = n1[0]
41 transition[2, 1] = n1[1]
42 transition[3, 0] = n1[2]
43 transition[4, 1] = n1[3]
44 transition[5, 2] = n1[4]
45 transition[4, 3] = n1[5]
46 transition[5, 4] = n1[6]
47 transition[6, 3] = n1[7]
48 transition[7, 4] = n1[8]
```

```
transition[8, 5] = n1[9]
transition[7, 6] = n1[10]
transition[8, 7] = n1[11]


def normalize(trans):
    """Normalize transition matrix"""
    sum_row = np.sum(trans, axis=1)
    for i in range(trans.shape[0]):
        for j in range(trans.shape[1]):
            trans[i][j] = trans[i][j] / sum_row[i]
    return trans


tran_matrix = normalize(transition)
np.savetxt("%d_Adj_nor.csv" % day, tran_matrix, delimiter=',')
```

B.3 Code for Models

Below is python code for models and experiments.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import time
import torch
torch.set_default_dtype(torch.float64)
from torch.autograd import Variable
from torch.nn.parameter import Parameter
import torch.nn as nn
import torch.utils.data as utils
from sklearn.preprocessing import MinMaxScaler


################################################


# LOG = True
LOG = False
day = 365
# ------------------------------------------------
```

```
CUDA_id = 0
os.environ["CUDA_VISIBLE_DEVICES"] = "%d" % CUDA_id
CUDA = torch.cuda.is_available()
print("device is cuda:", CUDA)
scaler = MinMaxScaler()


################################################################


def PrepareDataset(num_matrix, BATCH_SIZE=40, seq_len=72, pred_len=1,
↪  train_propotion=0.8, valid_propotion=0.1):
    time_len = num_matrix.shape[0]

    max_num = num_matrix.max().max()
    # num_matrix = num_matrix / max_num

    num_matrix = scaler.fit_transform(num_matrix)

    num_sequences, num_labels = [], []
    for i in range(time_len - seq_len - pred_len):
        num_sequences.append(num_matrix[i:i + seq_len])
        num_labels.append(num_matrix[i + seq_len:i + seq_len + pred_len])
    num_sequences, num_labels = np.asarray(num_sequences), np.asarray(num_labels)
    sample_size = num_sequences.shape[0]

    # shuffle and split the dataset to training and testing datasets
    # index = np.arange(sample_size, dtype=int)
    # np.random.shuffle(index)

    train_index = int(np.floor(sample_size * train_propotion))
    valid_index = int(np.floor(sample_size * (train_propotion + valid_propotion)))

    train_data, train_label = num_sequences[:train_index], num_labels[:train_index]
    valid_data, valid_label = num_sequences[train_index:valid_index],
    ↪  num_labels[train_index:valid_index]
    test_data, test_label = num_sequences[valid_index:], num_labels[valid_index:]

    train_data, train_label = torch.Tensor(train_data), torch.Tensor(train_label)
    valid_data, valid_label = torch.Tensor(valid_data), torch.Tensor(valid_label)
    test_data, test_label = torch.Tensor(test_data), torch.Tensor(test_label)

    train_dataset = utils.TensorDataset(train_data, train_label)
    valid_dataset = utils.TensorDataset(valid_data, valid_label)
    test_dataset = utils.TensorDataset(test_data, test_label)

    train_dataloader = utils.DataLoader(train_dataset, batch_size=train_index,
    ↪  shuffle=False, drop_last=True)
```

```
66      valid_dataloader = utils.DataLoader(valid_dataset, batch_size=valid_index -
        ↪  train_index, shuffle=False, drop_last=True)
67      test_dataloader = utils.DataLoader(test_dataset, batch_size=sample_size -
        ↪  valid_index, shuffle=False, drop_last=True)
68
69      return train_dataloader, valid_dataloader, test_dataloader, max_num
70
71
72  def TrainModel(model, train_dataloader, valid_dataloader, learning_rate=0.02,
    ↪  num_epochs=2000, patience=500, l2_lamb=1e-5, min_delta=1e-5):
73
74      inputs, labels = next(iter(train_dataloader))
75      [batch_size, step_size, fea_size] = inputs.size()
76
77      use_gpu = torch.cuda.is_available()
78      if use_gpu:
79          model.cuda()
80
81      loss_MSE = torch.nn.MSELoss()
82
83      # learning_rate = 1e-5
84      optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate,
        ↪  weight_decay=l2_lamb)
85
86      interval = 100
87      losses_train = []
88      losses_valid = []
89      losses_epochs_train = []
90      losses_epochs_valid = []
91
92      cur_time = time.time()
93      pre_time = time.time()
94
95      # Variables for Early Stopping
96      is_best_model = 0
97      patient_epoch = 0
98
99      for epoch in range(num_epochs):
100
101         trained_number = 0
102
103         valid_dataloader_iter = iter(valid_dataloader)
104
105         losses_epoch_train = []
106         losses_epoch_valid = []
107
108         for data in train_dataloader:
109             inputs, labels = data
```

```
110
111             if inputs.shape[0] != batch_size:
112                 continue
113
114             if use_gpu:
115                 inputs, labels = Variable(inputs.cuda()), Variable(labels.cuda())
116             else:
117                 inputs, labels = Variable(inputs), Variable(labels)
118
119             # log transform
120             if LOG:
121                 inputs, labels = torch.log(1 + inputs), torch.log(1 + labels)
122
123             model.train()
124
125             outputs = model(inputs)
126
127             loss_train = loss_MSE(outputs, torch.squeeze(labels, 1))
128
129             losses_train.append(loss_train.data)
130             losses_epoch_train.append(loss_train.data)
131
132             optimizer.zero_grad()
133             loss_train.backward()
134             optimizer.step()
135
136             # validation
137             try:
138                 inputs_val, labels_val = next(valid_dataloader_iter)
139             except StopIteration:
140                 valid_dataloader_iter = iter(valid_dataloader)
141                 inputs_val, labels_val = next(valid_dataloader_iter)
142
143             if use_gpu:
144                 inputs_val, labels_val = Variable(inputs_val.cuda()),
                    ↪ Variable(labels_val.cuda())
145             else:
146                 inputs_val, labels_val = Variable(inputs_val), Variable(labels_val)
147
148             # log transform
149             if LOG:
150                 inputs_val, labels_val = torch.log(1+inputs_val),
                    ↪ torch.log(1+labels_val)
151
152             model.eval()
153             outputs_val = model(inputs_val)
154
155
```

```
            loss_L1 = torch.nn.L1Loss(reduction='none')
            loss_l1 = loss_L1(outputs_val, torch.squeeze(labels_val, 1))
            loss_l1_np = loss_l1.cpu().data.numpy()

            losses_valid.append(np.mean(loss_l1_np))
            losses_epoch_valid.append(np.mean(loss_l1_np))

            # output
            trained_number += 1

            # inverse train log transform
            if LOG:
                inputs, labels, outputs = torch.exp(inputs) - 1, torch.exp(labels) -
                ↪  1, torch.exp(outputs) - 1

        avg_losses_epoch_train = sum(losses_epoch_train) /
        ↪  float(len(losses_epoch_train))
        avg_losses_epoch_valid = sum(losses_epoch_valid) /
        ↪  float(len(losses_epoch_valid))
        losses_epochs_train.append(avg_losses_epoch_train)
        losses_epochs_valid.append(avg_losses_epoch_valid)

        # Early Stopping
        if epoch == 0:
            best_epoch = epoch
            is_best_model = 1
            best_model = model
            best_train_outputs = outputs
            best_valid_outputs = outputs_val
            min_loss_epoch_valid = 10000.0
            if avg_losses_epoch_valid < min_loss_epoch_valid:
                min_loss_epoch_valid = avg_losses_epoch_valid
        else:

            if min_loss_epoch_valid - avg_losses_epoch_valid > min_delta:
                is_best_model = 1
                best_model = model
                best_train_outputs = outputs
                best_valid_outputs = outputs_val
                min_loss_epoch_valid = avg_losses_epoch_valid
                best_epoch = epoch
                patient_epoch = 0
            else:
                is_best_model = 0
                patient_epoch += 1
                if patient_epoch >= patience:
                    print('Early Stopped at Epoch: {}, min_loss_epoch_valid: {}, at
                    ↪  epoch: {}'.format(epoch, min_loss_epoch_valid, best_epoch))
```

```
                    break

        # Print training parameters
        if (epoch % 100) == 0:
            cur_time = time.time()
            print('Epoch: {}, train_loss: {}, valid_loss: {}, time: {}, best model:
            ↪  {}'.format(
                epoch,
                np.around(avg_losses_epoch_train.cpu().data.numpy(), decimals=8),
                np.around(avg_losses_epoch_valid, decimals=8),
                np.around([cur_time - pre_time], decimals=2),
                is_best_model))
            pre_time = cur_time

    loss_data = {"losses_train": losses_train,
                 "losses_valid": losses_valid,
                 "losses_epochs_train": losses_epochs_train,
                 "losses_epochs_valid": losses_epochs_valid,
                 "min_loss_epoch_valid": min_loss_epoch_valid}

    train_data = {"inputs": inputs, "labels": labels, "outputs": best_train_outputs}
    valid_data = {"inputs": inputs_val, "labels": labels_val, "outputs":
    ↪  best_valid_outputs}

    return best_model, loss_data, train_data, valid_data


def TestModel(model, test_dataloader):
    inputs, labels = next(iter(test_dataloader))
    [batch_size, step_size, fea_size] = inputs.size()

    cur_time = time.time()
    pre_time = time.time()

    use_gpu = torch.cuda.is_available()

    for data in test_dataloader:
        inputs, labels = data

        if inputs.shape[0] != batch_size:
            continue

        if use_gpu:
            inputs, labels = Variable(inputs.cuda()), Variable(labels.cuda())
        else:
            inputs, labels = Variable(inputs), Variable(labels)

        # log transform
```

```
246         if LOG:
247             inputs, labels = torch.log(1 + inputs), torch.log(1 + labels)
248 
249         outputs = model(inputs)
250 
251         # inverse train log transform
252         if LOG:
253             inputs, labels, outputs = torch.exp(inputs) - 1, torch.exp(labels) - 1,
                ↪  torch.exp(outputs) - 1
254 
255         # inverse minmiax scaler
256         outputs_np = outputs.cpu().data.numpy()
257         outputs_normal = scaler.inverse_transform(outputs_np)
258         labels_np = torch.squeeze(labels, 1).cpu().data.numpy()
259         labels_normal = scaler.inverse_transform(labels_np)
260 
261         APE = abs(outputs_normal-labels_normal)/labels_normal
262         APE = np.ma.masked_invalid(APE)
263         MAPE = np.mean(APE, axis=0).reshape(-1, 1)
264         MedAPE = np.median(APE, axis=0).reshape(-1, 1)
265 
266     MedAPE_mean = np.mean(MedAPE)
267 
268     MAPE_mean = np.mean(MAPE)
269 
270     print('Tested: MedAPE: {}'.format(MedAPE_mean))
271     print('Tested: MAPE: {}'.format(MAPE_mean))
272     test_loss = {"MedAPE": MedAPE,
273                  "MAPE": MAPE
274                  }
275     test_data = {"inputs": inputs, "labels": labels, "outputs": outputs}
276     return test_loss, test_data
277 
278 
279 def plot_result(d_train, d_val, d_test, special_note=""):
280     global fea_size, num_epoch, lr, hidden_dim
281 
282     input_data_train, label_data_train, out_data_train = d_train["inputs"],
        ↪  d_train["labels"], d_train["outputs"]
283     input_data_val, label_data_val, out_data_val = d_val["inputs"], d_val["labels"],
        ↪  d_val["outputs"]
284     input_data_test, label_data_test, out_data_test = d_test["inputs"],
        ↪  d_test["labels"], d_test["outputs"]
285 
286     train_len = label_data_train.size(0)
287     val_len = label_data_val.size(0)
288 
289     real_data = torch.squeeze(torch.cat((label_data_train, label_data_val,
        ↪  label_data_test), 0)).cpu().numpy()
```

```
290     predict_data = torch.cat((out_data_train, out_data_val, out_data_test),
        ↪  0).cpu().detach().numpy()
291
292     real_data_normal = scaler.inverse_transform(real_data)
293     predict_data_normal = scaler.inverse_transform(predict_data)
294
295     dic = './plot_result/LOG_%d_day_%d' % (LOG, day) + time.strftime("%Y-%m-%d
        ↪  %H:%M", time.localtime()) + special_note
296     if not os.path.exists(dic):
297         print("---  new folder(%s)...  ---" % dic)
298         os.makedirs(dic)
299         print("---  OK  ---")
300
301     for i in range(fea_size):
302         plt.figure(figsize=(60, 10))
303         plt.axvline(x=train_len, linestyle='--', c="r", linewidth=2.0)
304         plt.axvline(x=train_len + val_len, linestyle='--', c="b", linewidth=2.0)
305         plt.plot(np.arange(0, 0 + len(real_data_normal[:, i])), real_data_normal[:,
            ↪  i], linewidth=2.0)
306         plt.plot(np.arange(0, 0 + len(predict_data_normal[:, i])),
            ↪  predict_data_normal[:, i], linewidth=1.5)
307         plt.suptitle('node %d' % i + ' epochs=' + str(num_epoch) + 'lr=' + str(lr))
308
309         plt.savefig(dic + '/' + 'node%d' % i + '_lr' + str(lr) + '_epoch' +
            ↪  str(num_epoch) + '_hidden' + str(
310             hidden_dim) + '.eps', format='eps',
311                     dpi=1000)
312         plt.close()
313
314
315 ###################################################################
316
317
318 '''LSTM'''
319 class LSTM(nn.Module):
320     def __init__(self, input_size, hidden_size, num_classes, num_layers):
321         super(LSTM, self).__init__()
322
323         self.num_classes = num_classes
324         self.input_size = input_size
325         self.hidden_size = hidden_size
326         self.num_layers = num_layers
327
328         self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
            ↪  num_layers=num_layers, batch_first=True)
329
330         self.fc = nn.Linear(self.hidden_size, self.num_classes)
331         if torch.cuda.is_available():
```

```
            self.lstm, self.fc = self.lstm.cuda(), self.fc.cuda()

    def forward(self, x):
        # initialize
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
        if CUDA:
            h_0, c_0 = h_0.cuda(), c_0.cuda()

        # Propagate input through LSTM
        ula, (h_out, _) = self.lstm(x, (h_0, c_0))
        hidden = h_out.view(-1, self.hidden_size)
        out = self.fc(hidden)
        out = torch.sigmoid(out)

        return out


'''LSTM_PageRank'''
class LSTM_PageRank(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes, num_layers, adj_matrix,
    ↪  restart):
        super(LSTM_PageRank, self).__init__()

        self.num_classes = num_classes
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
        ↪  num_layers=num_layers, batch_first=True)

        self.fc = nn.Linear(self.hidden_size, self.num_classes)

        self.restart = restart
        self.A_tilde_hat = torch.from_numpy(adj_matrix)

        use_gpu = torch.cuda.is_available()
        if use_gpu:
            self.lstm, self.fc, self.A_tilde_hat = self.lstm.cuda(), self.fc.cuda(),
            ↪  self.A_tilde_hat.cuda()

    def forward(self, x):
        # initialize
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size))
        if CUDA:
            h_0, c_0 = h_0.cuda(), c_0.cuda()
```

```
377
378         # Propagate input through LSTM
379         ula, (h_out, _) = self.lstm(x, (h_0, c_0))
380         hidden = h_out.view(-1, self.hidden_size)
381
382         use_gpu = torch.cuda.is_available()
383         if use_gpu:
384             Pppr = self.restart * torch.inverse(torch.eye(x.size(-1)).cuda() - (1 -
                ↪ self.restart) * self.A_tilde_hat)
385         else:
386             Pppr = self.restart * torch.inverse(torch.eye(x.size(-1)) - (1 -
                ↪ self.restart) * self.A_tilde_hat)
387
388         out = torch.sigmoid(torch.matmul(self.fc(hidden), Pppr))
389
390         return out
391
392
393 ##################################################################
394
395
396 """HYPER PARAMETERS"""
397 K = 1  # predict signal K steps ahead
398 seqLen = 7 * 24  # sequence length
399 num_epoch = 1000
400 lr = 0.02
401 hidden_size = 3
402 num_layers = 1
403 alpha = 0.3
404 print("alpha=", alpha)
405
406 """LOAD DATA"""
407 data = np.loadtxt("./data/%d_data_n.csv" % day, delimiter=",", usecols=list(range(1,
    ↪ 10)))
408
409 train_dataloader, valid_dataloader, test_dataloader, max_num = PrepareDataset(data,
    ↪ seq_len=seqLen, pred_len=K, train_propotion=0.7, valid_propotion=0.2)
410 inputs, labels = next(iter(train_dataloader))
411 [batch_size, step_size, fea_size] = inputs.size()
412 input_dim = fea_size
413 hidden_dim = fea_size * hidden_size
414 output_dim = fea_size
415
416 """ADJACENT MATRIX"""
417 adj_nor = np.loadtxt("./%d_Adj_nor.csv" % day, delimiter=",")
418 nNodes = adj_nor.shape[0]
419
420 print("device is cuda:", torch.cuda.is_available())
```

```
421
422
423 ##################################################################
424
425
426 """Experiment: LSTM"""
427 start_time = time.time()
428 lstm = LSTM(input_dim, hidden_dim, output_dim, num_layers)
429 print('Start training LSTM...')
430 lstm, lstm_loss, data_train, data_val = TrainModel(lstm, train_dataloader,
    ↪  valid_dataloader, learning_rate=lr, num_epochs=num_epoch)
431 lstm_test, data_test = TestModel(lstm, test_dataloader)
432 print("Time:%.2f" % (time.time()-start_time))
433 plot_result(data_train, data_val, data_test, "LSTM")
434 MAPE = lstm_test["MAPE"].data
435 MedAPE = lstm_test["MedAPE"].data
436
437
438 """Experiment: LSTM_PageRank"""
439 start_time = time.time()
440 lstm_pr = LSTM_PageRank(input_dim, hidden_dim, output_dim, num_layers, adj_nor,
    ↪  alpha)
441 print('Start training LSTM_PageRank...')
442 lstm_pr, lstm_pr_loss, data_pr_train, data_pr_val = TrainModel(lstm_pr,
    ↪  train_dataloader, valid_dataloader, learning_rate=lr, num_epochs=num_epoch)
443 lstm_pr_test, data_pr_test = TestModel(lstm_pr, test_dataloader)
444 print("Time:%.2f" % (time.time()-start_time))
445 plot_result(data_pr_train, data_pr_val, data_pr_test, "LSTM_PageRank")
446 MAPE_pr = lstm_pr_test["MAPE"].data
447 MedAPE_pr = lstm_pr_test["MedAPE"].data
448
449
450 ##################################################################
451 # """Pre-experiment"""
452 # def experiment(model):
453 #     global data,  train_dataloader, valid_dataloader, test_dataloader, max_num
454 #     """HYPER PARAMETERS"""
455 #     K = 1  # predict signal K steps ahead
456 #     seqLen = 24 * 7  # sequence length
457 #     num_epoch = 2000
458 #     lr = 0.02
459 #
460 #     threshold = 1e-5
461 #
462 #     """Experiment: LSTM_PageRank"""
463 #     lstm_pr, train_loss, data_pr_train, data_pr_val = TrainModel(model,
    ↪  train_dataloader, valid_dataloader,
464 #
    ↪  learning_rate=lr, num_epochs=num_epoch)
```

```
#     test_loss, data_pr_test = TestModel(lstm_pr, test_dataloader)
#
#
# for i in range(1, 10):
#     alpha = i/10
#
#     """Experiment: LSTM_PageRank"""
#     model = LSTM_PageRank(input_dim, hidden_dim, output_dim, num_layers, adj_nor,
↪  alpha)
#     print('Start training LSTM_PageRank...')
#     print("alpha=", alpha)
#
#     experiment(model)
#     print("End! alpha=", alpha)
```