High-Order Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting

Zhiyong Cui[1], Kristian Henrickson[1], Ruimin Ke[1], Yinhai Wang[12]*
[1]Department of Civil and Environmental Engineering, University of Washington, Seattle
[2]Department of Electrical Engineering, University of Washington, Seattle
{zhiyongc, henr2237, ker27, yinhai}@uw.edu, * corresponding author

ABSTRACT

Traffic forecasting is a challenging task, due to the complicated spatial dependencies on roadway networks and the time-varying traffic patterns. To address this challenge, we learn the traffic network as a graph and propose a novel deep learning framework, High-Order Graph Convolutional Long Short-Term Memory Neural Network (HGC-LSTM), to learn the interactions between links in the traffic network and forecast the network-wide traffic state. We define the high-order traffic graph convolution based on the physical network topology. The proposed framework employs L1-norms on the graph convolution weights and L2-norms on the graph convolution features to identify the most influential links in the traffic network. We propose a novel Real-Time Branching Learning (RTBL) algorithm for the HGC-LSTM framework to accelerate the training process for spatio-temporal data. Experiments show that our HGC-LSTM network is able to capture the complex spatio-temporal dependencies efficiently present in the traffic network and consistently outperforms state-of-the-art baseline methods on two heterogeneous real-world traffic datasets. The visualization of graph convolution weights shows that the proposed framework can accurately recognize the most influential roadway segments in real-world traffic networks.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; neural networks • **Applied computing → Forecasting**; Transportation • **Mathematics of computing → Time series analysis**; **graph algorithms**

KEYWORDS

Traffic forecasting, Spatio-temporal, High-order Graph Convolution LSTM, Real-Time Branching Learning

1 INTRODUCTION

Traffic forecasting is one of the most challenging components of Intelligent Transportation Systems (ITS). The goal of traffic forecasting is to predict future traffic states in the traffic network given a sequence of historical traffic states and the physical roadway network. Due to the increasing volume and variety of traffic data that has become available in recent years, data-driven traffic forecasting methods have shown considerable promise in their ability to outperform conventional and simulation-based methods [23].

Previous work on this topic [10, 15] roughly categorizes existing models into two categories, i.e. classical statistical methods and machine learning models. Much of the work in statistical methods for traffic forecasting was developed years ago, when traffic systems were less complex and transportation datasets relatively small in size. The capability of such classical methods to deal with high dimensional, complex, and dynamic time series data is quite limited. With the more recent rapid development in traffic sensing technologies and computational power, as well as growth in traffic data volume, much of the more recent work on this topic focuses on machine learning methods for traffic forecasting.

With the ability to address high dimensional data and the capability of capturing complex non-linear relationships, machine learning methods, like support vector regression (SVR), tend to outperform the statistical methods, such as autoregressive integrated moving average (ARIMA) and its many variants [12], with respect to handling complex traffic forecasting problems [15]. However, the full potential of artificial intelligence approaches to traffic forecasting was not exploited until the rise of deep neural network (NN) models (also referred to as deep learning models). Following early work applying NNs to the traffic prediction problem, many NN-based methods, like feed forward NN [16] and recurrent NN (RNN) [20] have been adopted for traffic forecasting.

Deep learning models for traffic forecasting, like deep belief networks (DBN) [8] and stacked autoencoders [13], can effectively learn high dimensional features and achieve good forecasting performance. Due to their efficient use of temporal dependencies, RNN and its variants (long short-term memory (LSTM) recurrent neural network and gated recurrent unit (GRU)) show excellent potential for traffic forecasting [4, 15, 24]. Although previous RNN-based methods can learn the spatial dependencies, they tend to be over-complex and inevitably capture a certain amount of noise and spurious relationships which likely do not represent the true causal structure in a physical traffic network. Moreover, interpreting the network parameters in terms of real-world spatial dependencies is most often impossible. To address this, other works, like [14, 26], attempt to model spatial dependencies with convolutional neural network (CNN) and convolution-based residual networks. However, conventional CNNs are most appropriate for spatial relationships in Euclidean space as represented by 2D matrices or images. Thus, spatial features learned in a CNN are not optimal for representing the traffic network structure.

Recently, substantial research has focused on extending the convolution operator to more general, graph-structured data, which can be applied to capture the spatial relationships present in a traffic network. There are two primary ways to apply convolution to graph-structured data. The first method proposed makes use of spectral graph theory, by designing spectral filter/convolutions [2, 6, 9] and proposing a diffusion convolutional neural network (DCNN) [1]. These models successfully apply CNN to graphs, but they do not fully capture the unique properties of graph-structured data [25].

Several newly-published studies conduct graph convolution on graph data dynamically, for example, dynamic edge-conditioned filters in graph convolution [17] and the high-order adaptive graph convolutional network (HA-GCN) [25]. Spectral-based graph convolution has been adopted and combined with RNN [10] and CNN [23] to forecast traffic states. A trajectory-based convolution operator has also been proposed to capture the spatial dependency in the geo-location sequences [27]. Still, these methods are still not capable of fully accommodating the physical specialties of traffic networks, and thus, their learned spatial dependencies are hard to interpret.

In this work, we learn the traffic network as a graph and conduct high-order convolution on the traffic network graph based on the physical roadway characteristics. Combining this high-order traffic graph convolution with LSTM RNNs, we propose a high-order graph convolutional LSTM (HGC-LSTM) to model the dynamics of the traffic flow and capture the spatial dependencies. L1-norms on graph convolution weights and L2-norms on graph convolution features are added to the loss function of the proposed model as two regularization terms to make the graph convolution weight more stable and interpretable. We also design a novel training algorithm, named real-time branching learning (RTBL), to accelerate the convergence. Evaluation results show that the proposed HGC-LSTM outperforms multiple state-of-the-art traffic forecasting baselines. More importantly, the framework combining traffic graph convolution, HGC-LSTM and the RTBL algorithm as a whole turns out to be capable of identifying the most influential roadway segments in the real-world traffic networks.

The rest of the paper is organized as follows. In section 2, we introduce each building block of the proposed framework. Section 3 presents the experimental results. Finally, we conclude the paper in section 4.

2 Methodology

2.1 Traffic Forecasting Problem

Traffic forecasting refers to predicting future traffic states, in terms of speed or volume, given previously observed traffic states from a roadway network consisting of N sensor locations and road segments connecting the sensors. The traffic network and the relationship between sensor locations can be represented by an undirected graph $\mathcal{G}$ where $\mathcal{G} = (\mathcal{V}, \mathcal{E}, A)$ with N nodes (vertices) $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$. The connectedness of nodes is represented by an adjacency matrix $A \in \mathbb{R}^{N\times N}$, in which each element $A_{i,j} = 1$ if there is a link connecting node i and node j and $A_{i,j} = 0$ otherwise ($A_{i,i} = 0$). Based on the adjacency matrix, a link counting function $d(v_i, v_j)$ can be defined as counting the minimum number of links traversed from node i to node j. The edges (links) in a graph representing a traffic network are distinct from social network graphs, document citation graphs, or molecule graphs, in several respects: 1) there are no isolated nodes/edges; 2) the traffic status of each link in a traffic network varies with time; and 3) links in a traffic graph have meaningful physical characteristics, such as length of the road segment represented by a link, type of roadway, speed limit, and number of traffic lanes. Thus, we define a distance adjacency matrix, $D \in \mathbb{R}^{N\times N}$, where each element $D_{i,j}$ represents the real roadway distance from node i to j ($D_{i,i} = 0$). Let $X_t \in \mathbb{R}^{N\times P}$ be the graph signals, namely traffic state, for each node at time t, where P is the number of features associated with each node. The traffic forecasting problem aims to learn a function $F(\cdot)$ to map T' time steps of historical graph signals to the next subsequent T time step of graph signals:

$$F\big([X_{t-T'+1}, \dots, X_t]; \mathcal{G}(\mathcal{V}, \mathcal{E}, A, D)\big) = [X_{t+1}, \dots, X_{t+T}]$$

During the forecasting process, we also want to learn the traffic impact transmission between adjacent and neighboring nodes in a traffic network graph.

2.2 High-Order Traffic Graph Convolution

Firstly, we define the k-hop (k-th order) neighborhood $\mathcal{NB}_i = \{v_i \in \mathcal{V} | d(v_i, v_j) \le k\}$ for each node i. The one-hop neighborhood matrix for a graph $\mathcal{G}$ is exactly the adjacency matrix. Then, the k-hop neighborhood matrix can be acquired by calculating the k-th product of A. Here, we define that k-th order adjacency matrix as follows

$$\tilde{A}^k = \mathrm{Bi}(\textstyle\prod_{i=1}^{k} A + I) = \mathrm{Bi}(A^k + I) \tag{1}$$

where $\mathrm{Bi}(\cdot)$ is a function to clip the values of all elements of $A^k + I$ to 1, and thus, $A^k + I \in \{0,1\}^{N\times N}$. The identity matrix I added to A^k makes the nodes self-accessible in the graph. An example $\tilde{A}^k$ with respect to a node (a red star) is shown by blue points in the right part of Figure 1.

Then, the k-hop graph convolution can be defined as follows

$$GC^k = (W_{gc\_k} \odot \tilde{A}^k) X_t \tag{2}$$

where $\odot$ is the element-wise matrix multiplication operator, W_{gc_k} is the k-hop weight matrix for the k-hop adjacency matrix, and $X_t \in \mathbb{R}^{N\times 1}$ is the traffic feature at time t. However, when taking the underlying physics of vehicle traffic on a road network into consideration, we need to understand that the impact of a roadway segment on adjacent segments is transmitted in two primary ways: 1) slowdowns and/or blockages propagating upstream; and 2) driver behavior and vehicle characteristics associated with a particular group of vehicles traveling downstream. Thus, for a

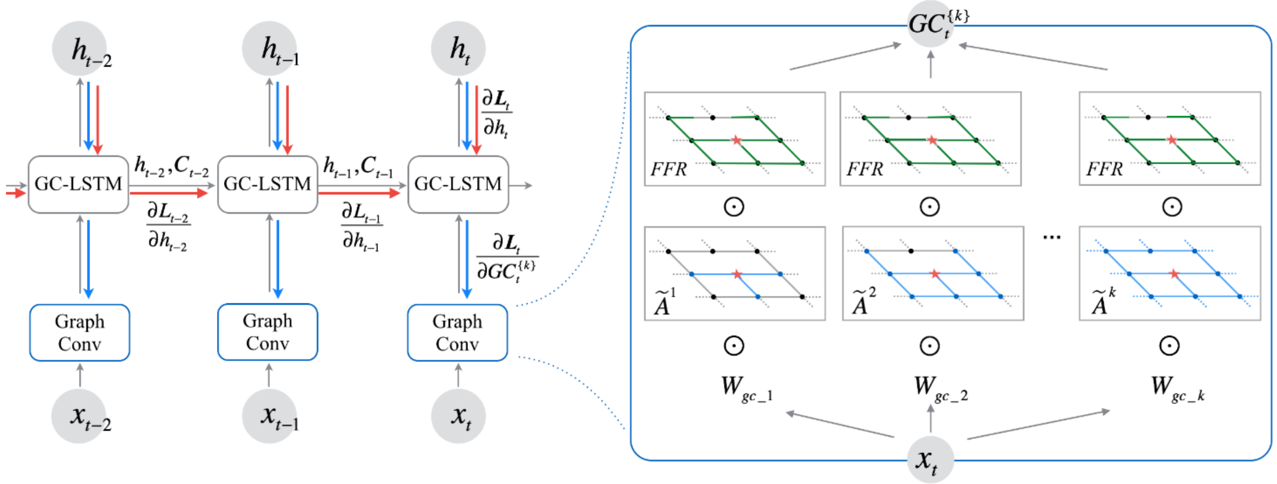

Figure 1: The architecture of the proposed High-order Graph Convolution LSTM RNNs is shown in the left side. The gray arrows represent the directions of data flows. The red arrows and the blue arrows demonstrate the directions of the gradient flows of the HGC-LSTM and the graph convolution component, respectively. The detailed Graph Convolution process is shown in the right part of the figure by unfolding the high-order traffic graph convolution component at time t, in which $\tilde{A}^k$s and $\mathcal{FFR}$ with respect to a star node are demonstrated.

traffic network-based graph or other similar graphs, the impact transmission between non-adjacent nodes cannot bypass the intermediate node/nodes, and thus, we need to consider the reachability of the impact between adjacent and nearby node pairs in the graph convolution. We define a free-flow reachable matrix, $\mathcal{FFR} \in \mathbb{R}^{N\times N}$, that

$$\mathcal{FFR}_{i,j} = \begin{cases} 1, & S_{i,j}^{\mathcal{FF}} * (m * \Delta t) - D_{i,j} \geq 0 \\ 0, & \text{otherwise} \end{cases}, \forall\, v_i, v_j \in \mathcal{V} \quad (3)$$

where $S_{i,j}^{\mathcal{FF}}$ is the free-flow speed between node i and j, and free-flow speed refers to the average speed that a motorist would travel if there were no congestion or other adverse conditions (such as severe weather). Δt is the duration of the minimum time interval in the sequence data and m is a number counting how many time intervals are considered to calculate the distance travelled under free-flow speed. Thus, m determines the temporal influence of formulating the $\mathcal{FFR}$, which is associated with the training algorithm introduced in section 2.5. Each element $\mathcal{FFR}_{i,j}$ equals one if vehicles can traverse from node i to j in m time-step, $m \cdot \Delta t$, with free-flow speed, and $\mathcal{FFR}_{i,j} = 0$ otherwise. All diagonal values of $\mathcal{FFR}$ are set as one. An example $\mathcal{FFR}$ with respect to a node (a red star) is shown by green points in the right part of Figure 1. Thus, the road network graph convolution is defined as

$$GC^k = (W_{gc\_k} \odot \tilde{A}^k \odot \mathcal{FFR}) X_t \quad (4)$$

where the k-hop adjacency matrix is multiplied element-wise with $\mathcal{FFR}$ to ensure the traffic impact transmission between k-hop adjacent nodes follow established traffic flow theory [5]. For a specific graph, when we increase k, the $\tilde{A}^k \odot \mathcal{FFR}$ will eventually converge such that $k = K$ and $\tilde{A}^K \odot \mathcal{FFR} = \mathcal{FFR}$.
Thus, at most, only K hops of graph convolution features need to be extracted from the data X_t. Features extracted by the graph convolution within the K hops adjacent nodes with respect to time t are concatenated together as follows

$$\boldsymbol{GC}_t^{\{K\}} = [GC_t^1, GC_t^2, \ldots, GC_t^K] \quad (5)$$

The $\boldsymbol{GC}^{\{K\}} \in \mathbb{R}^{N\times K}$ is set of high-order (K-th order) graph convolutional features, shown in the right part of Figure 1, that can be fed to the network models described in the following subsection.

2.3 Graph Convolutional LSTM

We propose a High-order Graph Convolutional LSTM (HGC-LSTM) recurrent neural network which learns both the complex spatial dependencies and the dynamic temporal dependencies present in traffic data. In this model, the gates structure in the traditional LSTM [7] and the hidden state are unchanged, but the input is replaced by the graph convolution features, which are reshaped into a vector $\boldsymbol{GC}^{\{K\}} \in \mathbb{R}^{NK}$.The forget gate, f_t, the input gate, i_t, the output gate, o_t, and the input cell state, $\tilde{C}_t$, at time t are defined as follows

$$\mathrm{f}_t = \sigma_g\left(W_f \boldsymbol{GC}_t^{\{k\}} + U_f h_{t-1} + b_f\right) \quad (6)$$

$$\mathrm{i}_t = \sigma_g\left(W_i \boldsymbol{GC}_t^{\{k\}} + U_i h_{t-1} + b_i\right) \quad (7)$$

$$\mathrm{o}_t = \sigma_g\left(W_o \boldsymbol{GC}_t^{\{k\}} + U_o h_{t-1} + b_o\right) \quad (8)$$

$$\tilde{C}_t = tanh\left(W_C \boldsymbol{GC}_t^{\{k\}} + U_C h_{t-1} + b_C\right) \quad (9)$$

where W_f, W_i, W_o, and $W_C \in \mathbb{R}^{KN\times N}$ are the weight matrices, mapping the hidden layer input to the three gates and the input cell state, while U_f, U_i, U_o, and $U_C \in \mathbb{R}^{N\times N}$ are the weight matrices connecting the previous cell output state to the three gates and the input cell state. b_f, b_i, b_o, and $b_C \in \mathbb{R}^N$ are four bias vectors. The

σ_g is the gate activation function, which typically is the sigmoid function, and tanh is the hyperbolic tangent function.

Since each node in a traffic network graph is influenced by its previous states and the states of neighboring graph nodes, the hidden state of each node should also only be effected by the hidden states of neighboring nodes. Thus, we define the graph hidden state as follows

$$C_t = \mathrm{f}_t \cdot \left(C_{t-1} \cdot \left(W_{\mathcal{N}} \odot \tilde{A}^K\right)\right) + \mathrm{i}_t \cdot \tilde{C}_t \tag{10}$$

where $\cdot$ is the matrix multiplication operator. The previous hidden state, C_{t-1}, multiplies a weighted K-hop adjacency matrix, $W_{\mathcal{N}} \odot \tilde{A}^K$, to calculate the hidden state, C_t. To characterize different contributions of neighboring nodes to the hidden state of each node, a neighboring weight matrix, $W_{\mathcal{N}}$, is multiplied element-wise with $\tilde{A}^k$. Then, the output, h_t, can be calculated as follows

$$h_t = \mathrm{o}_t \cdot \tanh(C_t) \tag{11}$$

2.4 Loss and Graph Convolution Regularization

As mentioned in the previous section, the HGC-LSTM output at time t is h_t, namely the predicted value $\hat{Y}_t = h_t$. Let Y_t denote the label at time t, and thus, the loss is defined as

$$L_t = \mathrm{Loss}\left(\hat{Y}_t - Y_t\right) \tag{12}$$

where $\mathrm{Loss}(\cdot)$ is a function to calculate the error between the predicted value $\hat{Y}_t$ and the label/true value Y_t. Normally, the $\mathrm{Loss}(\cdot)$ function is a Mean Squared Error (MSE) function for predicting continuous values or a Cross Entropy function for classification problems.

For a time series data, the label of time step t is the input of the next step, $t+1$, such that $Y_t = X_{t+1}$, and thus, the loss can be defined as

$$L_t = \mathrm{Loss}\left(\hat{Y}_t - Y_t\right) = \mathrm{Loss}(h_t - X_{t+1}) \tag{13}$$

To make the graph convolution feature more stable and interpretable, we add two regulation terms on the loss function.

2.4.1 Regularization on Graph Convolution weights

Because the graph convolution weights are not confined to be positive and each node's extracted features are influenced by multiple neighboring nodes, the graph convolution weights can vary a lot while training. Ideally, the convolution weights would be themselves informative, so that the relationships between different nodes in the network could be visualized by plotting the convolution weights. This is not likely to be possible without regularization, because very high or low weights tend to appear somewhat randomly, with the result that high/low weights tend to cancel each other out. In combination, such weights can still represent informative features for the network, but they cannot reflect the true relationship between nodes in the graph. Thus, we add L1-norm of the graph convolution weight matrices to the loss function as a regularization term to make these weight matrices as sparse as possible. The L1 regularization term is defined as follows

$$R^{\{1\}} = \left\|\boldsymbol{W}_{gc}\right\|_1 = \sum_{i=1}^{K} \left|W_{gc\_i}\right| \tag{14}$$

In this way, the trained graph convolution weight can be sparse and stable, and thus, it will be more intuitive to distinguish which neighboring node or group of nodes contribute most.

2.4.2 Regularization on Graph Convolution features

Considering that the impact of neighboring nodes w.r.t. a specific node must be transmitted through all nodes between the node of interest and the influencing node, features extracted from different hops in the graph convolution should not vary dramatically. Thus, to restrict the difference between features extracted from adjacent hops of graph convolution, we add an L2-norm regularization term on loss function at each time step. The L2 regularization term is defined as follows

$$R_t^{\{2\}} = \left\|\boldsymbol{GC}_t^{\{K\}}\right\|_2 = \sum_{i=1}^{K-1} \left(GC_t^i - GC_t^{i+1}\right)^2 \tag{15}$$

In this way, the features extracted from adjacent hops of graph convolution should not differ dramatically, and thus, the graph convolution operator should be more in keeping with the physical realities of the relationships present in a traffic network.

Then, the total loss function at time t can be defined as follows

$$L_t = \mathrm{Loss}(h_t - X_{t+1}) + \lambda_1 R^{\{1\}} + \lambda_2 R_t^{\{2\}} \tag{16}$$

where λ_1 and λ_2 are penalty terms to control the weight magnitude of the regularization terms on graph convolution weights and features.

2.5 Real-Time Branching Learning Algorithm

The training process of RNN-based neural networks typically uses the backpropagation through time (BPTT) algorithm [21], in which gradients of a weight at a sequence's all time steps are summed up and the weight is updated accordingly. Taking the weight of the forget gate, W_f, in the LSTM for an example, let $\frac{\partial L_t}{\partial W_f}$ as the gradient with regard to W_f at time step t. $[X_{t-T'+1}, \dots, X_t]$ is Then the gradient with regard to W_f over the whole input sequence, $[X_{t-T'+1}, \dots, X_t]$, in one training iteration is

$$\frac{\partial \mathrm{Loss}}{\partial W_f} = \sum_{i=t-T'+1}^{t} \frac{\partial L_t}{\partial W_f} \tag{17}$$

Then, the W_f can be updated using the gradient descent method or other similar updating methods. All the other weights in the LSTM are trained and updated in a similar way. The detailed BPTT algorithm can be found in [3].

However, network-wide traffic data has some peculiarities that warrant a slight modification of this training approach. For one, the traffic state at one node at time t is primarily determined by its own state and that of neighboring nodes at time $t-1$. That is because the traffic state at a specific roadway segment is determined and transmitted by the traffic flows in nearby segments [16]. In addition, as introduced in the aforementioned subsection, when applying RNN-based structures, the label of a time step in a traffic state sequence is the input of the next step. Thus, it is available to iteratively learn the spatial dependencies of neighbors in a traffic network graph solely based on one step historical data. If the weights of an RNN-based model are trained and updated in each

Algorithm 1 RTBL algorithm in HGC-LSTM's one iteration

```
1: Let $\boldsymbol{X_{T'}} = [X_{t-T'+1}, \dots, X_t]$ denotes input, $Y_t = X_{t+1}$ as label.
2: Let $\boldsymbol{W_{gc}} = \{W_{gc_1}, \dots, W_{gc_K}\}$.
3: Let $\boldsymbol{W_H} = \{\text{weights in HGC-LSTM}\}$.
4: $\eta_1, \eta_2$ are learning rates, $\lambda_1$ and $\lambda_2$ are penalties.
5: $m$ is the number of steps need to be applied BPTT
6: $Q_H \leftarrow$ a queue list with length $m$ to store gradients of $\boldsymbol{W_H}$.
7: for $i \leftarrow t - T' + 1$ to t do
8:     Input $X_i$ to HGC-LSTM
9:     $L_i \leftarrow \text{Loss}(h_i - X_{i+1}) + \lambda_1 R^{\{1\}} + \lambda_2 R_i^{\{2\}}$
10:    $\boldsymbol{W_{gc}} \leftarrow \boldsymbol{W_{gc}} + \frac{\partial L_i}{\partial \boldsymbol{W_{gc}}} * \eta_1$
11:    $Q_H$.pop($\frac{\partial L_{i-m-1}}{\partial \boldsymbol{W_H}}$)
12:    $Q_H$.append($\frac{\partial L_i}{\partial \boldsymbol{W_H}}$)
13:    $\boldsymbol{W_H} \leftarrow \boldsymbol{W_H} + \text{sum}(Q_H) * \eta_2$
14: end for
```

time step, the convergence rate will be accelerated and the training time will be increased. This kind of methods can be described as the real-time recurrent learning method [22].

From an overall view of the training process of the proposed HGC-LSTM model, the temporal dependencies need to be measured from historical sequences and the spatial dependencies can be captured from only one previous time step data. Thus, we proposed a real-time branching learning (RTBL) algorithm by combining the BPTT algorithm and the real-time recurrent learning algorithm together to achieve faster convergence and more informative feature learning. In detail, the gradient will only be back-propagated inside each time step (in real-time) and all the weights in the HGC-LSTM will be updated in each step. However, the calculated gradients in each step will be transmitted via two directions, like a branch, shown in the left part of Figure 1. The gradients with respect to the weights of the graph convolution component are back-propagated from the loss in each time step and the weights will be updated accordingly. The gradients with regard to the weights of the HGC-LSTM, which are also back-propagated from the loss, will be fed forward to the next subsequent step, just like the BPTT process. Thus, the gradients are transmitted into two branch in each time step. Taking time step t as an example and the RTBL process is shown in the left part of Figure 1, in which the gradients of the weights of the graph convolution and the HGC-LSTM are represented by blue arrows and red arrows, respectively.

The weights in the HGC-LSTM is updated using a truncated BPTT algorithm. In each time step, only gradients from a few previous steps, m ($m \ll T'$), will be summed and used to update weights. One reason is that using fewer time steps can accelerate the training process. Another reason is that the scale of the traffic graph's neighboring nodes is determined by the $\mathcal{FFR}$, and $\mathcal{FFR}$ is determined by m, described in Equation (4). Thus, the graph's neighboring nodes is inherently determined by m. Taking the previous example, the gradient of the forget gate weight in the LSTM at time t can be represented as follows

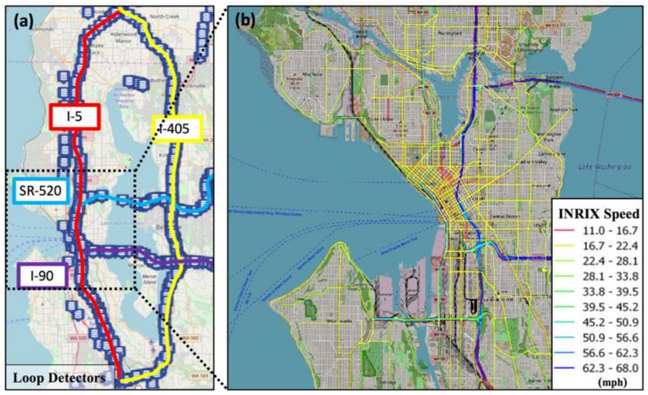

Figure 2: (a) Loop detector dataset covering the freeway network in Seattle area; (b) INRIX dataset covering the downtown Seattle area, where traffic segments are plotted with colors.

$$\frac{\partial \text{Loss}}{\partial W_f} = \sum_{i=t-m}^{t} \frac{\partial \text{Loss}(h_i - X_{i+1}) + \lambda_1 R^{\{1\}} + \lambda_2 R_i^{\{2\}}}{\partial W_f} \quad (18)$$

where m is the number of steps that can be back-propagated, defined in Equation (4), and T' is the total length of the input's time steps.

The gradients of the graph convolution component, $[W_{gc\_1}, \dots, W_{gc\_k}]$, are updated based on a single time step data. The gradient of the i-hop graph convolution weight at time t is calculated as follows

$$\frac{\partial \text{Loss}}{\partial W_{gc\_i}} = \frac{\partial L_t}{\partial W_{gc\_i}} = \frac{\partial \text{Loss}(h_t - X_{t+1}) + \lambda_1 R^{\{1\}} + \lambda_2 R_t^{\{2\}}}{\partial W_{gc\_i}} \quad (19)$$

which is shown by the blue arrows in Figure 1.

Algorithm 1 shows the details of RTBL updating algorithm in one training iteration of HGC-LSTM. By using this updating strategy, the spatial and temporal dependencies are properly used in the training process. In this case, for data with T' time steps, each training iteration will perform backpropagation T' times rather than once. There is an obvious trade-off between weight update frequency and the accuracy of the gradients. Moreover, there may be some concerns that this backpropagation process will take more time to train compared the BPTT method. However, for this particular traffic forecasting problem using RTBL strategy, it turns out that the training loss decreases faster and converges is achieved in fewer training iterations compared to the conventional BPTT. In addition, the proposed RTBL algorithm should be applicable in many other spatio-temporal deep learning frameworks, like the convolution recurrent neural network and its variants.

3 Experiments

3.1 Dataset Description

In this study, two real-world network-scale traffic speed dataset are utilized. The first contains data from fixed location mechanical sensors, which are collected by loop detectors on 4 connected freeways (I-5, I-405, I-90, and SR-520) in the Greater Seattle Area,

Table 1 Performance comparison of different approaches for traffic speed forecasting. The proposed TCG-LSTM with the RTBL strategy achieves the best performance with all the three metrics. (K=3 and m=3 in the proposed model)

Model	Loop Data			INRIX Data		
	MAE (mph)±STD	MAPE	RMSE	MAE (mph)±STD	MAPE	RMSE
ARIMA	6.10± 1.09	13.85%	10.65	4.80 ± 0.32	13.51%	10.85
SVR	6.85± 1.17	14.39%	11.12	4.78 ± 0.37	13.37%	10.44
FNN	4.45± 0.81	10.19%	7.83	2.31 ± 0.17	8.35%	5.92
LSTM	3.87± 0.63	9.51%	7.18	1.59 ± 0.12	5.78%	4.11
G-LSTM	3.34± 0.43	7.97%	5.98	1.54 ± 0.11	5.77%	3.28
HCG-LSTM	2.85± 0.23	6.54%	5.03	1.32 ± 0.09	4.04%	2.67
HGC-LSTM-RTBL	2.62± 0.20	6.21%	4.81	1.07 ± 0.07	3.74%	2.28

shown in Figure 2 (a). This dataset contains traffic state data from 323 sensor stations over the entirety of 2015 at 5-minute intervals. The second contains road link-level traffic speeds aggregated from GPS probe data collected by commercial vehicle fleets and mobile apps provided by the company INRIX. The INRIX traffic network covers the Seattle downtown area, shown in Figure 2 (b). This dataset describes the traffic state at 5-minute intervals for 1014 road segments and covers the entire year of 2012. We use LOOP data and INRIX data to denote the two datasets, respectively, in this study.

We adopt the speed limit as the free-flow speed, which for the segments in the LOOP traffic network is 60mph in all cases. The INRIX traffic network contains freeways, ramps, arterials and urban corridors, and so the free-flow speeds of INRIX traffic network range from 20mph to 60mph. The distance adjacency matrices D and free-flow reachable matrices $\mathcal{FFR}$ for both datasets are calculated based on the roadway characteristics and topology.

3.2 Experimental Settings

Baselines

We compare HGC-LSTM with the following baseline models, (1) ARIMA: Auto-Regressive Integrated Moving Average model with Kalman filter; (2) SVR: Support Vector Regression [18]; (3) FNN: Feed forward neural network with two hidden layers. (4) LSTM: Long Short-Term Memory recurrent neural network [7]; (5) G-LSTM: Graph LSTM [11]; (6) HGC-LSTM: proposed by this work without the RTBL training strategy. All the neural networks are implemented based on PyTorch and they are trained and evaluated on a single NVIDIA GeForece GTX 1080 Ti with 11GB memory.

HGC-LSTM Model

For both datasets, the dimensions of the hidden states of the HGC-LSTM are set as the amount of the nodes in the traffic network graphs. The size of hops in the graph convolution can vary, but we set it as 3, $K = 3$, for the model evaluation and comparison in this experiment. In the model comparison section (section 3.3), the learning rates for the two regularization terms are all set as 0.01. The $\mathcal{FFR}$ is calculated based on three time steps, and thus, m is set as three in the RTBL training algorithm. We train our model by minimizing the mean square error using RMSProp [19] with the batch size of 10. The initial learning rate is 10^{-5} and early stop mechanism is used to avoid overfitting.

Evaluation

In this study, the samples of the input are traffic time series data with 10 time steps. The output/label is the next one subsequent data of the input sequence. The performance of the proposed and the compared models are evaluated by three commonly used metrics in traffic forecasting, including 1) Mean Absolute Error (MAE), 2) Mean Absolute Percentage Error (MAPE), and 3) Root Mean Squared Error (RMSE).

$$MAE = \frac{1}{n}\sum\nolimits_{i=1}^{n}\left|Y_t - \hat{Y}_t\right| \tag{20}$$

$$MAPE = \frac{1}{n}\sum\nolimits_{i=1}^{n}\left|\frac{Y_t - \hat{Y}_t}{Y_t}\right| * 100\% \tag{21}$$

$$RMSE = \sqrt{\frac{1}{n}\sum\nolimits_{i=1}^{n}\left(Y_t - \hat{Y}_t\right)^2} \tag{22}$$

3.3 Experimental Results

Table 1 demonstrates the results of the HGC-LSTM with the RTBL training strategy and other baseline models on the two datasets. The proposed method outperforms other models with all the three metrics on the two datasets. These results suggest that non-neural-

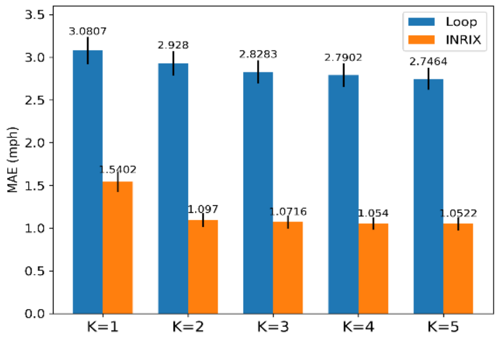

Figure 3: Histogram of performance comparison for the influence of orders (hops) of graph convolution in the HGC-LSTM on INRIX and LOOP datasets

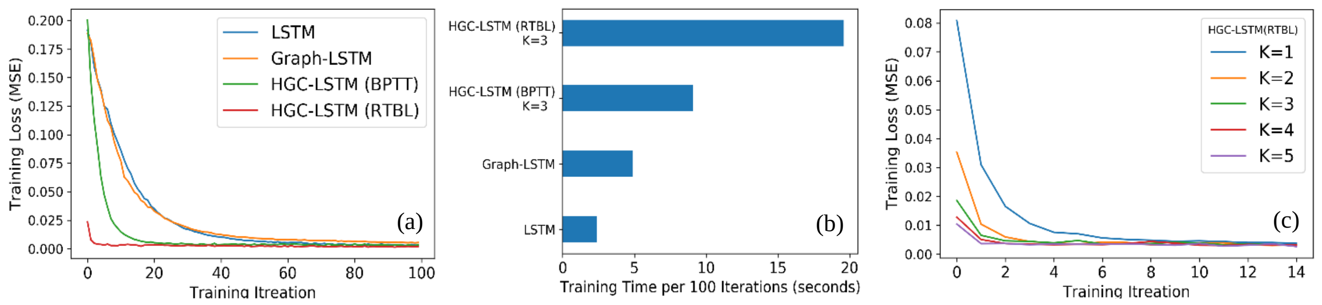

Figure 4: (a) Compare training efficiency with other models: training MSE versus training iteration (batch size = 40). (b) Compare training efficiency with different hops of graph convolutions: training MSE versus training iteration (batch size = 40). (c) Histogram of model's training time per 100 iterations. (The results in the three figures are based on the INRIX data)

network approaches are less appropriate for this network-wide prediction task, due to the complex spatio-temporal dependencies and the high dimension features in the datasets. Regardless of the training strategy, the proposed HGC-LSTM performs better than all other approaches. Moreover, when using the RTBL training strategy, the performance of the HGC-LSTM is improved over the conventional BPTT training strategy. It should be noted that the INRIX data is missing a number of observations during nighttime and off-peak hours, these are filled with free-flow speed. Thus, there are few variations at the non-peak hours in the INRIX data. Further, the values of INRIX data are all integers, which can be reflected by Figure 4(c). Therefore, the calculated errors of the INRIX data is less than that of the LOOP data and the evaluated performance on INRIX data is inflated somewhat.

Figure 3 shows a histogram of performance comparison on the effects of orders (hops) of the graph convolution in the HGC-LSTM with RTBL. The model performance is improved when the value of K increases. For the LOOP data, the performance improves slightly when K is gradually increased. But for the INRIX data, there is a big improvement in when K increases to two from one. The complex structure and the various road types in the INRIX traffic network should be the main reason for this performing difference. Further, when K is larger than two, the improvement of the prediction is quite limited. This is also the reason why we choose K=3 in the model comparison, shown by Table 1.

3.4 Training Efficiency

In this subsection, we compare the training efficiency of the proposed model and other models. Figure 4 (a) shows the training loss of different models with respect to the training iteration. The loss of the Graph-LSTM network decreases faster than LSTM at the beginning but slows down as training progresses. The HGC-LSTM with the BPTT training strategy, shown by the green curve, converges much faster than LSTM and Graph-LSTM. However, it is obvious that the training loss of the HGC-LSTM with the RTBL strategy decreases much faster and more evenly than that of other models.

Since there is a trade-off between weight update frequency and the accuracy of the gradients [22], it will cost more time to adopt the RTBL training strategy. Figure 4 (b) compares the training time of different models per 100 iterations. Graph-LSTM cost twice as much as LSTM does. The training time of HGC-LSTM with the BPTT training strategy is doubled comparing with Graph-LSTM. Further, the time required for HGC-LSTM with RTBL is around twice that of HGC-LSTM with BPTT. Although the proposed RTBL training method need more time, in our experiments, the HGC-LSTM with RTBL only needs at most 1/6 training iterations to reach the same training performance of other models. That means the RTBL strategy significantly accelerates the training process.

Figure 4 (c) shows the training losses of HGC-LSTM with different hops of graph convolution components. The rate of convergence become faster obviously when increasing the number of hops, k. In our experiments, when k is larger than 3, the training and the validation results improve slightly by increasing k for both INRIX and LOOP datasets.

3.5 Effect of Regularization

The model's loss function contains two regularization terms, the L1-norm on the graph convolution weights and the L2-norm on the graph convolutional features. These regularization terms help the graph convolution weights in the trained model to be more sparse, more clustered, and thus, more interpretable. Figure 4 (a) and (b) show portions of the averaged graph convolution weight matrices for the INRIX data and the LOOP data, respectively, where $K = 3$ and the average weight is calculated by $\frac{1}{K}\sum_{i=1}^{K} W_i \odot \tilde{A}^i \odot \mathcal{FFR}$. The road segment names, which are not displayed, are aligned on the vertical and horizontal axes with the same order in each figure. The colored dots in the matrices in Figure 4 (a) and (b) illustrate the weight of the contribution of a single node to its neighboring nodes. Since we align the traffic states of roadway segments based on their interconnectedness in the training data, most of the weights are distributed around the diagonal line of the weight matrix.

Due to the INRIX network is more complex and the average degree of nodes in the INRIX graph is higher than that in the LOOP graph, the dots in the average weight matrix of the INRIX graph convolution are more scattered. Compare to other approaches, using pre-defined graph weights [23] or training without regularization [10], our proposed graph convolution weights are still more clustered, especially in the weight matrix of the LOOP data. The clusters formed by the colored dots in Figure 4 (a) and (b) demonstrate groups of neighboring nodes. It is reasonable that

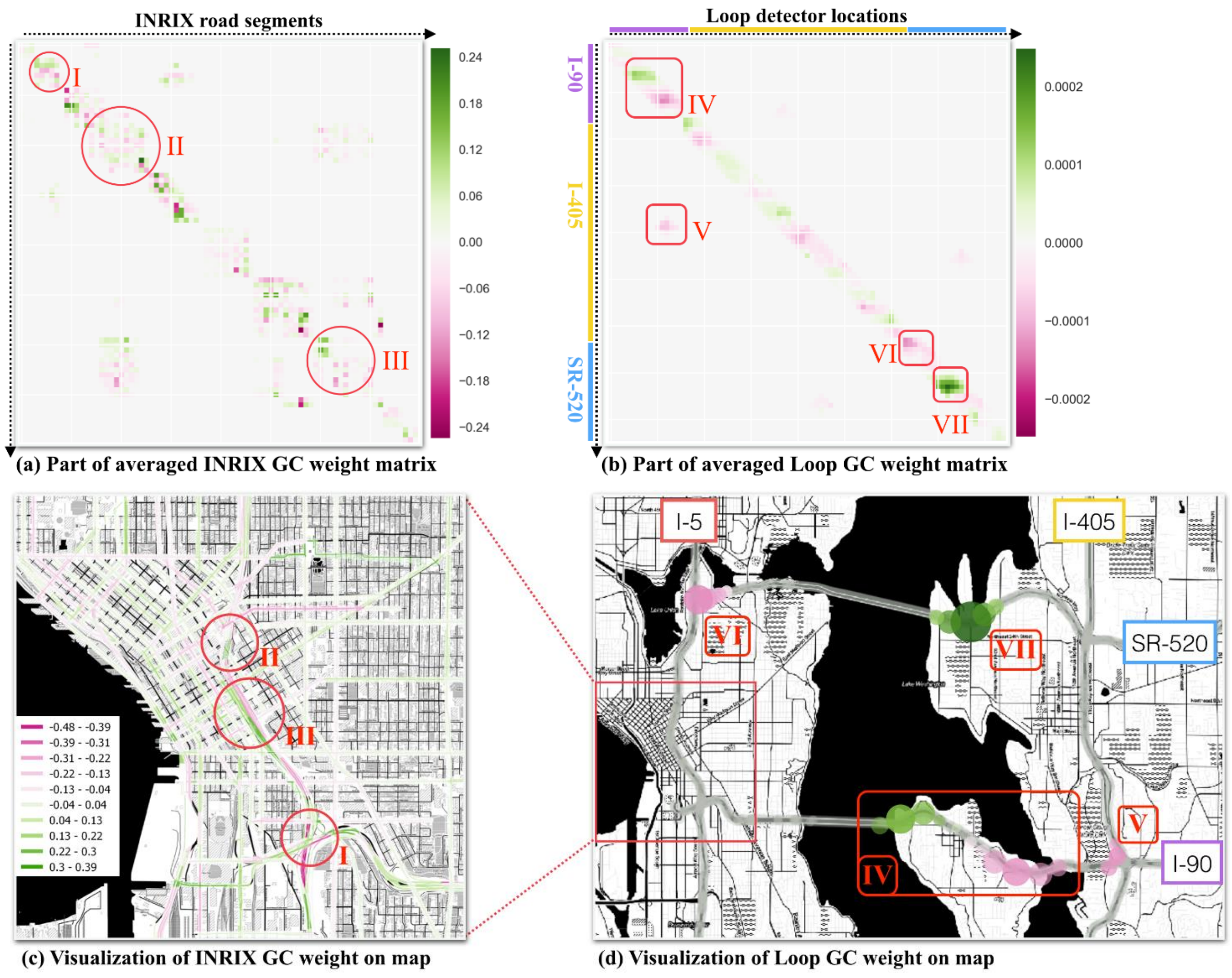

Figure 5: (a) Visualization of a proportion of the INRIX GC weight matrix, in which three representative weight areas are tagged. (b) Visualization of a proportion of the LOOP GC weight matrix, in which four representative weight areas are tagged. (c) Visualization of the INRIX graph convolution weight on the Seattle urban traffic network using colored lines. (d) Visualization of the four tagged weight areas in the LOOP graph convolution weight on the Seattle freeway network using colorful circles.

roadway segments should be influenced by their neighboring or nearby connected segments. The node with the largest absolute weight in a cluster is very likely to be a key road segment in the local traffic network. In this way, we can infer the bottlenecks of the traffic network from the graph convolution weight matrices.

3.6 Model Interpretation and Visualization

To better understand the contribution of the graph convolution weight, we mark seven groups of representative weights in Figure 4 (a) and (b) to show their physical locations on the real map in Figure 4 (c) and (d), by highlighting them with Roman numerals and red boxes. The influence of these marked weights of INRIX data and LOOP data on neighboring nodes are visualized by the lines and circles, respectively, due to the INRIX traffic network is too dense to use circles. The darkness of the green and pink colors and the size of circles represent the magnitude of influence. The distribution of the visualized dark lines and large circles makes intuitive sense, since the clusters in the weight matrix almost perfectly located at road segments connected with ramps or other arterials.

From Figure 4 (c), we can find the marked areas in the INRIX graph convolution weight matrix, (I), (II) and (III), are all located at very busy and congested freeway entrance and exit ramps, with dark colors, near the Seattle downtown area. In Figure 4 (d), the area tagged with (IV) is quite representative since the two groups of circles are located at the intersections between freeways and two main corridors that represent the entrances to the island (Mercer Island). Areas (V) and (VI) are the intersections between I-90 and I-405 and between I-5 and SR-520, respectively. Area (VII) located on SR-520 contains an often congested ramp connecting to the city of Bellevue, the location of which is highlighted by the biggest green circle. Additionally, there are many other representative areas in the graph convolution weight matrix, but we cannot show all of them due to the space limitations. Thus, by comparing the weight matrix with the physical realities of the traffic network, it can be shown that the proposed method effectively captures spatial

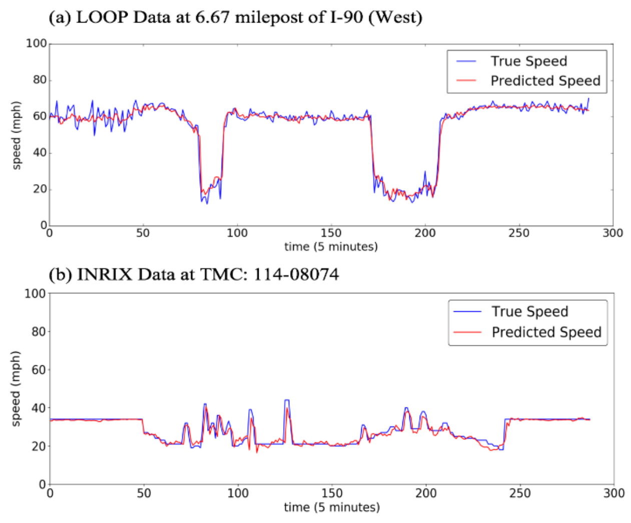

Figure 6: Traffic time series forecasting visualization for LOOP and INRIX datasets on two randomly selected days. The x-axis is time and the unit is 5-minutes.

dependencies and helps to identify the most influential points/segments in the traffic network.

Figure 6 visualizes the predicted traffic time series and the ground truth for two locations on two randomly selected days from the LOOP dataset and the INRIX dataset, respectively. Though the traffic networks of the two datasets are very different, the curves demonstrate that the trends of the traffic speed are predicted well at both peak traffic and off-peak hours. Thus, the proposed HGC-LSTM method is capable of capturing spatial and temporal dependencies from representations of the traffic network.

4 Conclusion

In this paper, we learn the traffic network as a graph and define a traffic graph convolution to capture spatial features from traffic network. We propose the high-order graph convolutional LSTM recurrently neural network as a framework to forecasting traffic spatio-temporal data. The regularization terms on the graph convolution help the proposed model be more stable and interpretable. We further proposed a real-time branching learning algorithm to accelerate the training convergence. By evaluating on two large-scale real-world traffic datasets, our approach outperforms the baselines. For the future work, we will move forward to conduct the high-order convolution on both spatial and temporal dimensions by incorporating the real-world traffic network.

REFERENCES

[1] James Atwood and Don Towsley. 2016. Di↵usion-convolutional neural networks. In Advances in Neural Information Processing Systems. 1993–2001.
[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203 (2013).
[3] Gang Chen. 2016. A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation. arXiv preprint arXiv:1610.02583 (2016).
[4] Zhiyong Cui, Ruimin Ke, and Yinhai Wang. 2018. Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction. arXiv preprint arXiv:1801.02143 (2018).
[5] Carlos F Daganzo. 1994. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. Transportation Research Part B: Methodological 28, 4 (1994), 269–287.
[6] Micha¨el De↵errard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems. 3844–3852.
[7] Sepp Hochreiter and J¨urgen Schmidhuber. 1997. Long short-term memory. Neural computation 9, 8 (1997), 1735–1780.
[8] Wenhao Huang, Guojie Song, Haikun Hong, and Kunqing Xie. 2014. Deep architecture for traffic flow prediction: deep belief networks with multitask learning. IEEE Transactions on IntelligentTransportation Systems 15, 5 (2014), 2191–2201.
[9] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016).
[10] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2017. Graph Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. arXiv preprint arXiv:1707.01926 (2017).
[11] Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. 2016. Semantic object parsing with graph lstm. In European Conference on Computer Vision. Springer, 125–143.
[12] Marco Lippi, Matteo Bertini, and Paolo Frasconi. 2013. Shortterm traffic flow forecasting: An experimental comparison of timeseries analysis and supervised learning. IEEE Transactions on Intelligent Transportation Systems 14, 2 (2013), 871–882.
[13] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. 2015. Traffic flow prediction with big data: a deep learning approach. IEEE Transactions on Intelligent Transportation Systems 16, 2 (2015), 865–873.
[14] Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. 2017. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. Sensors 17, 4 (2017), 818.
[15] Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. 2015. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. Transportation Research Part C: Emerging Technologies 54 (2015), 187–197.
[16] Dongjoo Park and Laurence R Rilett. 1999. Forecasting freeway link travel times with a multilayer feedforward neural network. Computer-Aided Civil and Infrastructure Engineering 14, 5 (1999), 357–367.
[17] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In Proc. CVPR.
[18] Alex J Smola and Bernhard Sch¨olkopf. 2004. A tutorial on support vector regression. Statistics and computing 14, 3 (2004), 199–222.
[19] Tijmen Tieleman and Geo↵rey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4, 2 (2012), 26–31.
[20] J Van Lint, S Hoogendoorn, and H Van Zuylen. 2002. Freeway travel time prediction with state-space neural networks: modeling state-space dynamics with recurrent neural networks. Transportation Research Record: Journal of the Transportation Research Board 1811 (2002), 30–39.
[21] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. Proc. IEEE 78, 10 (1990), 1550–1560.
[22] Ronald J Williams and David Zipser. 1989. Experimental analysis of the real-time recurrent learning algorithm. Connection Science 1, 1 (1989), 87–111.
[23] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal Graph Convolutional Neural Network: A Deep Learning Framework for Traffic Forecasting. arXiv preprint arXiv:1709.04875 (2017).
[24] Rose Yu, Yaguang Li, Cyrus Shahabi, Ugur Demiryurek, and Yan Liu. 2017. Deep learning: A generic approach for extreme condition traffic forecasting. In Proceedings of the 2017 SIAM International Conference on Data Mining. SIAM, 777–785.
[25] Zhenpeng Zhou and Xiaocheng Li. 2018. Convolution on Graph: A High-Order and Adaptive Approach. (2018).
[26] Junbo Zhang, Yu Zheng, Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In AAAI, pp. 1655-1661.
[27] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, Yu Zheng. 2018. When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks. In the proceeding of AAAI.