Graph Convolutional Networks with Kalman Filtering for Traffic Prediction

Fanglan Chen
Virginia Tech CS Dept.
fanglanc@vt.edu

Zhiqian Chen
Mississippi State University CSE Dept.
zchen@cse.msstate.edu

Subhodip Biswas
Virginia Tech CS Dept.
subhodip@cs.vt.edu

Shuo Lei
Virginia Tech CS Dept.
slei@vt.edu

Naren Ramakrishnan
Virginia Tech CS Dept.
naren@cs.vt.edu

Chang-Tien Lu
Virginia Tech CS Dept.
ctlu@vt.edu

ABSTRACT

Traffic prediction is a challenging task due to the time-varying nature of traffic patterns and the complex spatial dependency of road networks. Adding to the challenge, there are a number of errors introduced in traffic sensor reporting, including bias and noise. However, most of the previous works treat the sensor observations as exact measures ignoring the effect of unknown noise. To model the spatial and temporal dependencies, existing studies combine graph neural networks (GNNs) with other deep learning techniques but their equal weighting of different dependencies limits the models' ability to capture the real dynamics in the traffic network. To deal with the above issues, we propose a novel deep learning framework called Deep Kalman Filtering Network (DKFN) to forecast the network-wide traffic state by modeling the self and neighbor dependencies as two streams, and their predictions are fused under the statistical theory and optimized through the Kalman filtering network. First, the reliability of each stream is evaluated using variances. Then, the Kalman filter is leveraged to properly fuse noisy observations in terms of their reliability. Experimental results reflect the superiority of the proposed method over baseline models on two real-world traffic datasets in the speed prediction task.

CCS CONCEPTS

• **Information systems** → *Data mining*; • **Applied computing** → **Transportation**.

KEYWORDS

Graph Neural Networks, Kalman Filtering, Traffic Forecasting

ACM Reference Format:
Fanglan Chen, Zhiqian Chen, Subhodip Biswas, Shuo Lei, Naren Ramakrishnan, and Chang-Tien Lu. 2020. Graph Convolutional Networks with Kalman Filtering for Traffic Prediction . In *28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '20), November 3–6, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3397536.3422257

SIGSPATIAL '20, November 3–6, 2020, Seattle, WA, USA

ACM ISBN 978-1-4503-8019-5/20/11.
https://doi.org/10.1145/3397536.3422257

1 INTRODUCTION

Accurate traffic prediction is crucial to the development of Intelligent Traffic System (ITS), which nowadays plays an essential role in smart city planning, road safety management, and traffic control. Traffic prediction aims to forecast future states, including flow, speed, density, and trends, in the traffic network with historical observations of traffic states in a road network. Reliable traffic prediction not only provides scientific support to traffic operators for early detection of traffic congestion and manage detour in advance but also provides road users with recommendations of optimal travel routes to improve their travel experience [7].

Traffic prediction is considered a challenging task for a variety of factors. (1) Traffic patterns vary greatly with time, and the topology of road networks is complex. Temporally, the traffic state changes dynamically with time and presents periodical patterns; spatially, the topological structure of the urban road network has a large impact on the traffic state. The intertwining of spatial and temporal dependencies adds to the difficulty of accurate traffic state modeling. (2) Bias and noise widely exist in real-world traffic reporting. A large number of existing traffic data collection techniques, such as Bluetooth sensor and remote traffic microwave sensor, rely on data collected from sensors. The real-time traffic state on highway loops is usually recorded by loop detectors, but sometimes their reporting is not reliable due to noisy observations or reporting failures caused by the malfunction of the sensor, accidental manual system closure, signal transmission errors, etc.

The modeling of spatial and temporal dependencies is a fundamental issue in the transportation domain. Many traditional traffic prediction methods mainly focus on temporal dependency, taking full advantage of their strengths in dealing with time series. Examples include the Autoregressive Integrated Moving Average (ARIMA) [3], Support Vector Regression (SVR) [11], and deep learning models like Recurrent Neural Networks (RNNs) [2, 5]. However, these models do not consider spatial dependency, thereby limiting their ability to capture the traffic state governed by the road network structure.

Recent research works [3, 12] have extended the convolution operator to graph-structured data, and the concept of convolution can be naturally applied to model the spatial dependencies within a traffic network. These models successfully apply convolution to graph-structured data, but they do not fully capture the unique properties of graphs, such as road networks. Specifically, the challenges include (1) Traffic sensor reporting inevitably has bias and noise, but previous works ignore the unknown bias and noise by

treating the sensor observations as exact measures [3, 12]. Due to this assumption, errors are introduced into the model from the beginning and later gets further amplified. (2) The equal weighting of different dependencies is not guaranteed in the real world, limiting the model's ability to capture the real dynamics in the traffic network and their different levels of reliability are not considered.

To deal with these issues, we propose a novel deep learning framework called DKFN to forecast the network-wide traffic state via two streams, one for self dependency modeling, and the other for neighbor dependency modeling. The main idea is to capture the reliability of predictions from the historical observations of the sensor itself and those of the neighboring sensors separately over time. The predictions based on self and neighbor dependencies are fused and optimized through the proposed framework. The contributions of this paper include:

- Develop a framework to model the complex self and neighbor dependencies in the traffic network.
- Propose a novel Deep Kalman Filtering Network to capture the bias and noise in sensor reporting on traffic data.
- Conduct extensive experiments on existing benchmarks.

2 DKFN MODEL

In this section, we introduce our proposed model and describe how it works for the traffic speed prediction task.

2.1 Overview of Model Architecture

The state data collected in many traffic-related tasks are time-series. Thus, RNNs are commonly used in the traffic literature to capture the temporal dependency. In this work, we utilize the same strategy to model temporal dependency but with a different cutting point. In modeling the traffic network as a graph, we treat the influence from one node based on the historical observations from itself and its neighbors as two different dependencies.

As shown in Figure 1, our proposed model consists of three modules: self dependency modeling network, neighbor dependency modeling network, and Kalman filtering network.

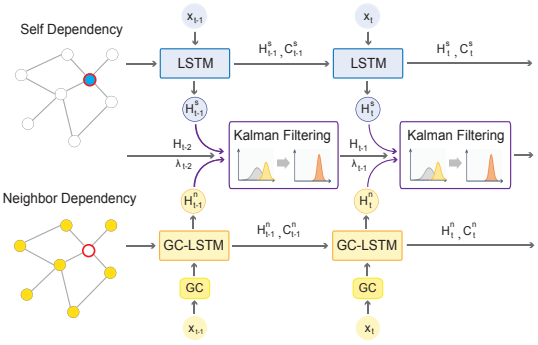

Figure 1: DKFN Model Architecture

2.2 Self Dependency Modeling Network

The self dependency modeling network targets at learning a function to estimate the current state based on the historical observations of each node itself. For modeling self dependency, or the influence of historical observations to the future state of each sensor in our case, we use the LSTM with three gates and memory cells similar to the vanilla LSTM. The LSTM estimates the traffic state at time t by iteratively taking the hidden states at previous time steps and the current traffic features as inputs. For simplicity, the self dependency modeling process is defined as follows:

$$\mathbf{H}_t^s, C_t^s = \text{LSTM}\left(X_{t-p}, \ldots, X_{t-2}, X_{t-1}\right), \tag{1}$$

where p is the length of historical time steps of the reported traffic state. At the final time step t, the hidden state $\mathbf{H}_t^s$ is the output of the self dependency modeling network, i.e., the predicted value $\hat{y}_s$.

2.3 Neighbor Dependency Modeling Network

The neighbor dependency modeling network aims to learn a function to estimate the current state based on historical observations from spatially proximal nodes. Graph is used to acquire the spatial dependency from the nodes adjacent to each other, with the recurrent neural network to obtain the neighbor dependency at each time step in the traffic network.

Since we intend to model neighbor dependency apart from self dependency, we adopt the graph constructed without self-loop, which disentangles the self and neighbor dependencies and provide our proposed model with flexibility to determine the weight of the dependencies towards more accurate predictive results. As discussed by Wang *et al.* [9], no single normalization strategy is optimal for all graphs under different scenarios. Since we consider the extension of the neighbors of every node in the traffic network, we chose *row normalization* as our strategy and the normalized adjacency matrix is computed as

$$\tilde{A} = D^{-1}A. \tag{2}$$

The main idea of a convolution layer is to extract localized features from inputs in a 2D or 3D matrices structure [6]. The localized area of the input space which has an impact on the convolution operation results, can be seen as the receptive field. Similarly, the operation of a graph convolution layer is to extract localized features from graph structured inputs. Thus, the product of the input and the normalized adjacency matrix and a trainable weight matrix is considered as a graph convolution operation that extracts features from each node's localized region [6]. The graph convolution generated at the time step t is defined as follows:

$$\mathbf{GC}_t = \left(W_{gc} \odot \tilde{A}\right) x_t, \tag{3}$$

where $\odot$ is the Hadamard product, and $x_t \in \mathbb{R}^N$ denotes the vector of traffic state (speed in our case) of all nodes (traffic sensors) at time t. $W_{gc} \in \mathbb{R}^{N \times N}$ is a trainable weight matrix for the graph convolution and the $\mathbf{GC}_t \in \mathbb{R}^N$ represents the extracted traffic features. Moreover, the trainable weight W_{gc} has the ability to capture the interactive impact between traffic sensors in the network.

Along with the graph convolution layer, neighbor dependency modeling network also utlizes LSTM, in which the gate structure and the hidden states are unchanged from its vanilla version [5], but inputs are substituted by the extracted graph convolution features, which are reshaped into a vector $\mathbf{GC}_t \in \mathbb{R}^N$. The input gate i_t, output gate o_t, forget gate f_t, and the memory cell state C_t at the

time step t are calculated as follows:

$$i_t = \sigma\left(W_i \cdot [\mathbf{H}_{t-1}, \mathbf{GC}_t] + b_i\right), \tag{4}$$

$$o_t = \sigma\left(W_o \cdot [\mathbf{H}_{t-1}, \mathbf{GC}_t] + b_o\right), \tag{5}$$

$$f_t = \sigma\left(W_f \cdot [\mathbf{H}_{t-1}, \mathbf{GC}_t] + b_f\right), \tag{6}$$

$$\tilde{\mathbf{C}}_t = \tanh\left(W_c \cdot [\mathbf{H}_{t-1}, \mathbf{GC}_t] + b_c\right), \tag{7}$$

where $\cdot$ denotes the matrix multiplication operator, W_i, W_o, W_f, and W_c are the weight matrices. b_i, b_o, b_f, and b_c are bias vectors. σ is the sigmoid function, and $tanh$ is the hyperbolic tangent function.

Since the traffic state at each sensor location would be influenced by the previously reported traffic states at the same location and neighboring stations, the LSTM cell state of each node should also be affected by neighboring cell states in the graph. Hence, a cell state gate is defined and added to the original LSTM cell. The cell state at the last time step $t-1$ is defined as follows:

$$\mathbf{C}^*_{t-1} = \left(W_N \odot \tilde{A}\right) \cdot C_{t-1}, \tag{8}$$

where W_N is a weight matrix to measure the influence each node received from its neighboring nodes. Then, the final cell state and the hidden state are calculated as follows:

$$\mathbf{C}^n_t = f_t \odot \mathbf{C}^*_{t-1} + i_t \odot \tilde{\mathbf{C}}_t, \tag{9}$$

$$\mathbf{H}^n_t = o_t \odot \tanh\left(\mathbf{C}^n_t\right). \tag{10}$$

At the final time step t, the hidden state $\mathbf{H}^n_t$ is the output of the neighbor dependency modeling network, which is equivalent to the predicted value $\hat{y}_n$. The predicted values, $\hat{y}_s$ and $\hat{y}_n$, of the above two networks, are the inputs to our Kalman Filtering network.

2.4 Kalman Filtering Network

Different from most of the existing works, our work treats the self dependency and neighbor dependency observations as noisy measurements rather than exact ground truth. Therefore, each observation is not completely reliable to predict the future traffic state. To alleviate the impact of noise and robust the modeling of traffic state, the Kalman filter is leveraged to properly fuse the dependencies.

Assume that these self dependency and neighbor dependency observations are subject to Gaussian distributions:

$$y_s\left(x; \mu_s, \sigma_s\right) \triangleq \frac{e^{-\frac{(x-\mu_s)^2}{2\sigma_s^2}}}{\sqrt{2\pi\sigma_s^2}}, \quad y_n\left(x; \mu_n, \sigma_n\right) \triangleq \frac{e^{-\frac{(x-\mu_n)^2}{2\sigma_n^2}}}{\sqrt{2\pi\sigma_n^2}}, \tag{11}$$

where subscript s and n denote *self dependency* and *neighbor dependency*, respectively.

To handle the noise, we employ the Kalman filter to derive accurate information from multiple observations. Specifically, their probability distribution function can be fused by multiplication:

$$y_{\text{fused}}\left(x; \mu_s, \sigma_s, \mu_n, \sigma_n\right) = \frac{e^{-\frac{(x-\mu_s)^2}{2\sigma_s^2}}}{\sqrt{2\pi\sigma_s^2}} \times \frac{e^{-\frac{(x-\mu_n)^2}{2\sigma_n^2}}}{\sqrt{2\pi\sigma_n^2}} \tag{12}$$

$$= \frac{1}{2\pi\sqrt{\sigma_s^2\sigma_n^2}} e^{-\left(\frac{(x-\mu_s)^2}{2\sigma_s^2} + \frac{(x-\mu_n)^2}{2\sigma_n^2}\right)}. \tag{13}$$

By reorganizing and transforming, it can be rewritten as:

$$y_{\text{fused}}\left(x; \mu_{\text{fused}}, \sigma_{\text{fused}}\right) = \frac{1}{\sqrt{2\pi\sigma^2_{\text{fused}}}} e^{-\frac{(x-\mu_{\text{fused}})^2}{2\sigma^2_{\text{fused}}}}, \tag{14}$$

where,

$$\mu_{\text{fused}} = \frac{\mu_s\sigma_n^2 + \mu_n\sigma_s^2}{\sigma_s^2 + \sigma_n^2} \quad \text{and} \quad \sigma^2_{\text{fused}} = \frac{\sigma_s^2\sigma_n^2}{\sigma_s^2 + \sigma_n^2}. \tag{15}$$

This means that different observations can be fused using weighted sum, and the weight is the variance of the opposite one. The physical meaning of variance is the degree of reliability (lower is the better). However, calculating the variance of one observation requires the complete data, which is computationally expensive to obtain. Therefore, we estimate the variance by computing the variance distribution of each training samples:

$$\mathbb{E}[\sigma^2_{\{s,n\}}] = \frac{1}{N_T}\sum_i \frac{(T_i - \bar{T})^2}{N}, \tag{16}$$

where N is the length of each sample sequence, and N_T is the number of samples. Further to improve the fusion, we add a weight variable to re-balance different observations:

$$\hat{y}_{\text{fused}} = \frac{\hat{y}_s(\gamma \cdot \sigma_n^2) + \hat{y}_n\sigma_s^2}{\sigma_s^2 + \gamma \cdot \sigma_n^2}, \tag{17}$$

where γ is the weight variable. To feed it into neural networks, we can remove the constant denominator since the self and neighbor dependency modeling networks are aware of scalability:

$$\hat{y} = \gamma \cdot \sigma_n^2\hat{y}_s + \sigma_s^2\hat{y}_n, \tag{18}$$

where $\hat{y}$ is the prediction. Note that our method can reduce computational overhead by assuming two Gaussian distributions and only calculating the estimated variances. The original Kalman filter is an iterative algorithm, but it involves too many matrix multiplications [4], which makes it not suitable for practical implementation.

The pseudo-code of the proposed framework DKFN is provided in Algorithm 1.

Algorithm 1: DKFN model

Input: $X_T = [x_1, \ldots, x_T], A$
Parameters: W_{gc}, W, U, b, W_N, and γ
Initialize: $\tilde{A} = D^{-1}A, \mathbf{H}^s_0, \mathbf{H}^n_0 = \mathbf{0} \in \mathbb{R}^N, C^s_0, C^n_0 = \mathbf{0} \in \mathbb{R}^N$
for $t = 1$ ***to*** T **do**
  $\mathbf{H}^s_t, C^s_t = \text{LSTM}\left(X_{t-p}, \ldots, X_{t-2}, X_{t-1}\right)$ ▷ Eq. 1
  $\mathbf{GC}_t = \left(W_{gc} \odot \tilde{A}\right) x_t$ ▷ Eq. 3
  $\mathbf{H}^n_t, C^n_t = \text{GC-LSTM}\left(x_t, \boldsymbol{GC}_t, h_{t-1}, C^n_{t-1}\right)$ ▷ Eq. 9 and 10
  $\sigma_s = \text{var}\left(X_{t-p}, \ldots, X_{t-2}, X_{t-1}\right)$ ▷ self dependency
  $\sigma_n = \text{var}\left(\mathbf{GC}_{t-p}, \ldots, \mathbf{GC}_{t-2}, \mathbf{GC}_{t-1}\right)$ ▷ neighbor dependency
  $\mathbf{H}_t = \text{DKFN}\left(\mathbf{H}^s_t, \mathbf{H}^n_t, \sigma_s, \sigma_n\right)$ ▷ Eq. 18
return $\mathbf{H}_t$

3 EXPERIMENTATION

In this section, we evaluate the predictive performance of our model on two real-world traffic datasets. Our DKFN implementation is available at https://github.com/Fanglanc/DKFN.

3.1 Experimental Setting

Dataset Description. **Seattle-Loop** is collected by 323 inductive loop detectors installed on the roadway surface [10]. We used the traffic speed data from June 1 to August 31 in 2015 with time step interval as 5 minutes. **METR-LA** is collected by 207 traffic sensors on the highways of Los Angeles County. We aggregated the traffic speed data during May 1 and May 31 in 2012 by 5 minutes. The missing rate is 2.6% and linear interpolation is applied.

Model Implementation. The dimensions of the hidden states of the proposed model are set as the number of traffic sensors in each of the datasets. We trained our model using Adam optimizer with an initial learning rate of 10^{-4}. 70% of the data are used for training, 10% for validation, and the remaining 20% for testing. We evaluated the model performance via four metrics: mean absolute error (MAE), mean absolute percentage error (MAPE), root mean squared error (RMSE), and coefficient of determination (R^2).

Baseline Models. We compared our framework with existing classical and deep learning models including HA [7], SVR [8], ARIMA [1], RNN [2], LSTM [5], TGC-LSTM [3], and T-GCN [12].

3.2 Results

Table 1: Predictive results of the proposed DKFN and other baseline methods on Seattle-Loop and METR-LA datasets

	Seattle-Loop				METR-LA			
	MAE	MAPE	RMSE	R^2	MAE	MAPE	RMSE	R^2
HA	3.456	6.935%	6.427	0.793	3.207	8.527%	5.639	0.784
SVR	3.140	9.388%	5.839	0.856	3. 039	8.169%	4.698	0.821
ARIMA	9.659	31.529%	12.064	*	7.167	19.874%	9.410	*
RNN	2.914	6.917%	4.152	0.885	2.794	6.217%	4.196	0.873
LSTM	2.723	6.566%	4.033	0.892	2.592	5.974%	4.239	0.872
TGC-LSTM	2.797	6.870%	4.529	0.864	2.857	6.451%	4.623	0.791
T-GCN	3.043	7.649%	4.449	0.868	2.930	6.680%	4.563	0.852
GC-LSTM	2.699	6.433%	3.999	0.894	2.601	6.026%	4.304	0.867
DKFN	**2.635**	**6.267%**	**3.960**	**0.896**	**2.510**	**5.870%**	**4.069**	**0.881**

Table 1 presents the performance of the proposed framework and other baseline methods on Seattle-loop and Metr-LA datasets. By looking at the results, we can conclude: (1) Generally, deep learning models obtain better predictive precision than non-deep learning baselines. HA achieves moderate results in regards to MAE, but it cannot capture the trend which can explain its poor performance on R^2. As a mature time series forecasting method, ARIMA's predictive precision is relatively lower than that of the HA, which highlights ARIMA's disadvantage in dealing with long-term and non-stationary data. Compared to other deep learning baselines, our proposed model achieves the best performance across all the metrics. (2) Overall, RNNs achieved good results, which emphasized the importance of the temporal features in traffic prediction task. On both of the datasets, LSTM achieves better predictive results compared to vanilla RNN. Especially on the METR-LA dataset, LSTM-based models, including LSTM, TGC-LSTM, and our neighbor dependency modeling network GC-LSTM, outperform other baseline models on both of the datasets. Compared with the performance of vanilla RNN model on the Seattle-Loop dataset, the RMSEs of the GC-LSTM and our proposed model are reduced by 7.4% and 9.6%. (3) Performance enhancement cannot be guaranteed by introducing the spatial dependency via graph convolution layers. We can observe that on the METR-LA dataset, the gain of introducing additional spatial information to temporal dependency is not that significant, and the RNN models perform better than the baseline TGC-LSTM and T-GCN models. It could be due to the distance from node to node is long in the METR-LA traffic network, thus the information received by neighbors is not that reliable due to the sparsity of the network. Our designed GC-LSTM obtains good initialization through using the normalized adjacency matrix to generate graph convolution, and achieves good performance. The proposed DKFN takes advantage of adjusting the ratio of predictions from the self historical observations and those of its neighbors. The reliability of both stream modelings as self dependency and neighbor dependency can be automatically learned by the model and obtained more accurate results compared to the single source.

4 CONCLUSION

In this work, we propose a novel deep learning framework DKFN , which takes advantage of modeling self and neighbor dependencies separately for traffic speed prediction and fuse the two streams under the statistical theory and optimized through the Kalman filtering network. When evaluated on two real-world traffic datasets, our proposed model achieves the best predictive results.

ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation via grant #DGE-1545362, UrbComp: Data Science for Modeling, Understanding, and Advancing Urban Populations.

REFERENCES

[1] Mohammed S Ahmed and Allen R Cook. 1979. *Analysis of freeway traffic time-series data by using Box-Jenkins techniques.* Number 722.
[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.
[3] Z. Cui, K. Henrickson, R. Ke, and Y. Wang. 2019. Traffic Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting. *IEEE Transactions on Intelligent Transportation Systems* (2019), 1–12.
[4] Ramsey Faragher. 2012. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal processing magazine* 29, 5 (2012), 128–132.
[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[6] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR).*
[7] Jing Liu and Wei Guan. 2004. A summary of traffic flow forecasting methods [J]. *J. of Highway and Transportation Research and Development* 3 (2004), 82–85.
[8] Alex J Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Statistics and computing* 14, 3 (2004), 199–222.
[9] Yewen Wang, Ziniu Hu, Yusong Ye, and Yizhou Sun. 2020. Demystifying Graph Neural Network Via Graph Filter Assessment. https://openreview.net/forum?id=r1erNxBtwr
[10] Yinhai Wang, Weibin Zhang, Kristian Henrickson, Ruimin Ke, Zhiyong Cui, et al. 2016. *Digital roadway interactive visualization and evaluation network applications to WSDOT operational data usage.* Technical Report. Washington (State). Dept. of Transportation.
[11] Zhi-sheng Yao, Chun-fu Shao, and Yong-liang Gao. 2006. Research on methods of short-term traffic forecasting based on support vector regression [J]. *Journal of Beijing Jiaotong University* 30, 3 (2006), 19–22.
[12] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li. 2020. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems* 21, 9 (2020), 3848–3858.