Predicting Traffic Flow Based on Encoder-Decoder Framework

Xiaosen Zheng, Zikun Yang, Liwen Liu, and Li Kuang(✉)

School of Computer Science and Engineering,
Central South University, Changsha 410075, China
kuangli@csu.edu.cn

Abstract. Predicting traffic flow is of great importance to traffic management and public safety, and it has high requirements on accuracy and efficiency. However, the problem is very challenging because of high-dimensional features, spatial levels, and sequence dependencies. On the one hand, we propose an effective end-to-end model, called FedNet, to predict traffic flow of each region in a city. First, for the temporal *trend, period, closeness* properties, we obtain low-dimensional features by downsampling high-dimensional input features. Then we perform temporal fusion to get temporal aggregations of different spatial levels. Next, we generate traffic flow by upsampling the fused features which are obtained by combining the corresponding temporal aggregation and the output of the previous upsample block. Finally, the traffic flow is adjusted by external factors like weather and date. On the other hand, we transfer the original task into a sequence task and then use teacher forcing to train our model, which make it learn the sequence dependencies. We conduct extensive experiments on two types of traffic flow (new-flow/end-flow and inflow/outflow) in New York City and Beijing to demonstrate that the FedNet outperforms five well-known methods.

Keywords: Traffic flow prediction · Encoder-Decoder framework · Skip connection · Teacher forcing

1 Introduction

Predicting traffic flow in a city is of great importance to traffic management and public safety. For example, massive crowds of people streamed into a strip region by different vehicles like the bike, taxi, bus, subway etc. at the 2015 New Year's Eve celebrations in Shanghai, resulting in a catastrophic stampede. If we can predict the traffic flow with high accuracy and efficiency, then adopt emergency measures, such tragedies can be mitigated even prevented.

New-flow is the transportation flow originating from a region at a given time interval. End-flow is the transportation flow terminated in region. Intuitively, new-flow and end-flow track the origins and final destinations of the transportation. These thus summarize the movements of transportations and are enough for traffic management and risk assessment, as shown in Fig. 1.

X. Wang et al. (Eds.): CollaborateCom 2019, LNICST 292, pp. 518–533, 2019.
https://doi.org/10.1007/978-3-030-30146-0_36

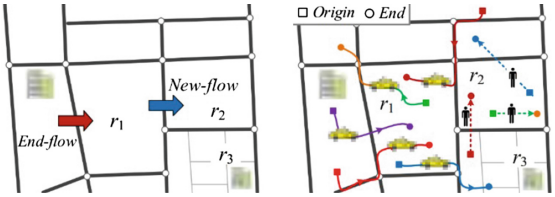

Fig. 1. Traffic flow in a region [1].

Although recently published works consider spatial dependencies, temporal dependencies and external influence, model them properly and get the state-of-the-art accuracy, simultaneously predicting the traffic flow in each region of a city is still challenging, affected by the following aspects:

1. **High-Dimensional Features.** A city usually has a very large size, containing many regions, so the dimension of the input matrix will be high. And for citywide traffic prediction, the output matrix will have the same size with the input matrix. So, previous mentioned state-of-the-art models, which are based on deep neural networks, usually obtain high-dimensional features in hidden layers. First, from the perspective of auto-encoder, if the dimension of feature is too high, it will be difficult for the model to learn useful information from the input and get more general representations [2]. Second, using these high-dimensional features usually requires more parameters and computation.
2. **Spatial Levels.** To capture the city-wide spatial dependencies, we stack some convolution layers because one convolution layer only accounts for near spatial dependencies. However, if we only consider the output of the final convolution layer which contains coarse semantic information of city-wide spatial dependencies, we will lose too much detail information from low-level spatial dependencies like distinct-wide spatial dependencies, especially when we use stride-2 convolution.
3. **Sequence Dependencies.** Some external factors like events may tremendously change the traffic flow in continuous time steps. Although we employ external component, we cannot collect all the information due to many realistic limitations. If we train the model by one-time-step predicting, then the model will focus on the next time step and is not robust enough to tackle some unexpected continuous volume change. For this issue, we consider the relationship among multiple future time steps as sequence dependencies.

To tackle these challenges, we propose an effective model, called FedNet, to collectively predict traffic flow in every region more accurately and more efficiently. The primary contributes of this paper can be summarized as follows:

1. FedNet adopts the encoder-decoder framework to obtain the low-dimensional features by downsampling the high-dimensional features for each temporal property, which leads it to learn more general representations and reach higher accuracy while needs fewer parameters and computation.
2. Not only consider temporal properties, but we also take the influence of spatial properties into account. Inspired by Skip Architecture [3], we add some skip

connections between corresponding temporal fusion blocks and upsample blocks to model different level spatial properties respectively. Especially, we make use of the detail information from lower spatial level.

3. We model the sequence dependencies, which means the relationship among multiple future time steps, by forcing the model to predict multi-time-step at the training stage. Instead of using recursive multi-step forecast which is hard to train because of slow convergence, model instability and poor skill, we adopt teacher forcing [4], which *works by using the actual output from the training dataset at the current time step y(t) as input in the next time step X(t + 1) rather than the output generated by the network.*

2 Related Work

Traffic Flow Prediction. For individual-scale traffic flow prediction, some previous work mainly predicts massive individuals' traces based on people's location history [5, 6], which requires massive computation. For road-scale, some researchers focus on predicting travel speed and traffic volume on the road [7–9]. For region-scale, there are previously published works like FCCF [1], which naturally focus on the individual region instead of the city and need a complex method to find irregular regions. However, such tasks that focus on part of the city are not always necessary for applications like traffic management which needs the information of the overall situation. Recently, researchers have started to focus on city-scale traffic flow prediction, tried to adopt deep learning methods and proposed some effective models like DeepST and ST-ResNet [10, 11]. These DNN-based models firstly partition the city using a grid-based method. However, all these methods are different from ours where they did not tackle the challenges of high-dimensional features, spatial levels, and sequence dependencies.

Deep Learning. To capture spatial dependencies, the convolution neural network has been successfully applied to various problems like image classification [12]. For capturing temporal dependencies, recurrent neural networks based on the long short-team memory unit has been successfully used to various sequence learning task [13]. To capture spatial-temporal dependencies, researchers recently proposed a convolutional LSTM network [14]. However, this network exists gradient vanishing problem, so it cannot model very long-range temporal dependencies. Also, training becomes more difficult as depth increases. Instead, ST-ResNet employs residual learning that enables networks to have a deep structure and a parametric-matrix-based fusion mechanism to model the spatial-temporal dependencies of traffic flow [15]. Though it shows state-of-the-art results, it has massive parameters and requires huge computation, which limits its application. Also, it did not consider modeling different spatial level properties and explore the sequence dependencies of traffic flow predicting task.

Encoder-Decoder Framework. The idea of the encoder-decoder framework is simple: An encoder processes the input and emits a fixed-dimension context. Then a decoder generates the output based on the context. For machine translation, researchers

proposed the encoder-decoder or sequence to sequence architecture to map a variable-length sequence to another variable-length sequence for machine translation which obtained state-of-the-art translation [13]. In computer vision field, the convolution encoder-decoder framework is widely applied, especially for some problems that must generate an image output but not a label such image segmentation, style transfer and super-resolution [3, 16, 17]. To the best of our knowledge, no prior work studies predicting traffic flow based on the encoder-decoder framework.

3 Preliminary

In this section, we first present several preliminaries and define our problem formally.

Definition 1 (Region [11]). *There are many definitions of a location in terms of different granularities and semantic meanings. In this study, we partition a city into an $H * W$ grid map based on the longitude and latitude where a grid denotes a region.*

Definition 2 (New-Flow/End-Flow [1]). *The movement of a transportation can be recorded as a trajectory $\mathcal{T}$, which is a sequence of time-ordered points, $\mathcal{T} : p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_{|\mathcal{T}|}$, where each point $p_i = (a_i, b_i, t_i)$ has a geospatial coordinate position (a_i, b_i) and a timestamp t_i, and $|\mathcal{T}|$ is the number of point in $\mathcal{T}$. Likewise, the movement of crowds can be represented by a collection of trajectories $\boldsymbol{P}$. Specifically, for a region $g(i,j)$, the two types of flow at timestamp t, namely new-flow and end-flow, are defined respectively as*

$$x_t^{new,i,j} = \left|\{\mathcal{T} \in \boldsymbol{P} : (a_1, b_1) \in g(i,j), t_1 = t\}\right|$$
$$x_t^{end,i,j} = \left|\left\{\mathcal{T} \in \boldsymbol{P} : \left(a_{|\mathcal{T}|}, b_{|\mathcal{T}|}\right) \in g(i,j), t_{|\mathcal{T}|} = t\right\}\right|$$

where $(a_i, b_i) \in g(i,j)$ means that point p_i lies within region $g(i,j)$. A simple example of new-flow and end-flow in every region is as shown in Fig. 2. At the t^{th} time interval, new-flow and end-flow in all $I * J$ regions can be denoted as tensor $\boldsymbol{X}_t \in \mathbb{R}^{2*I*J}$ where $(\boldsymbol{X}_t)_{0,i,j} = x_t^{new,i,j}$, $(\boldsymbol{X}_t)_{1,i,j} = x_t^{end,i,j}$.

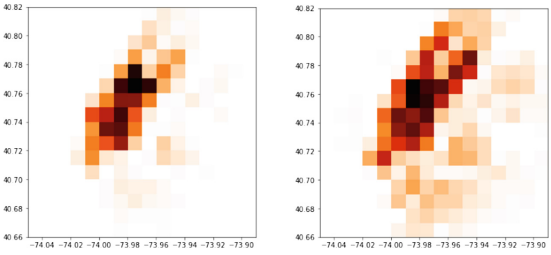

Fig. 2. The taxi new-flow and end-flow example of New York City. (The deeper red means higher traffic volume) (Color figure online)

Problem 1. Given the historical observation $\{\boldsymbol{X}_t | t = 0, 1, \ldots, n-1\}$ predict $\boldsymbol{X}_n$.

4 Model Architecture

In this section, we describe the architecture of our proposed FedNet, as shown in Fig. 3. FedNet is comprised of four components: an encoder component, a fusion component, a decoder component, and an external component. Referring to previous work, we also turn the traffic flow like new-flow and end-flow into 2-channel image-like matrices according to Definitions 1 and 2, and then divide the input sequence into three fragments: *trend, period, closeness* according to the analyzation [10, 11]. The encoder component captures the spatial dependencies of the input sequence *trend, period, closeness* respectively. The fusion component contains two sub-components: temporal fusion sub-component and the spatial fusion one. The temporal fusion sub-component outputs temporal aggregations of different spatial levels. The spatial one is used to combine corresponding temporal aggregation with the output of previous the upsample block. The decoder component generates the traffic flow output by upsampling the fused features. The traffic flow is further adjusted by the external component that processes the external factors (e.g. holidays) and combines traffic flow with external features.

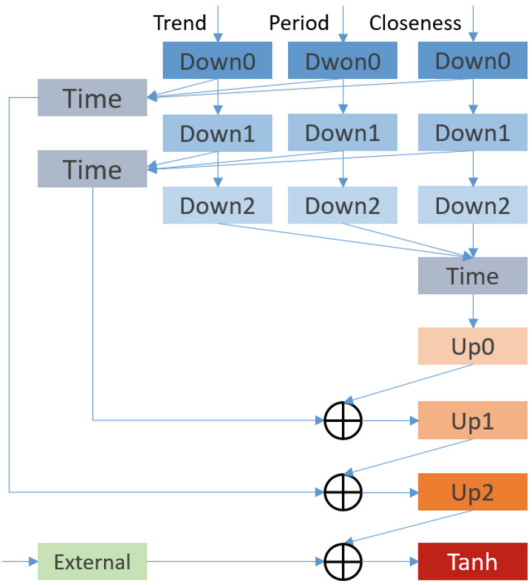

Fig. 3. FedNet (Flow Encoder-Decoder Network) architecture. Down0: No. 0 downsample block; Time: Temporal fusion block; Up0: No. 0 upsample block; External: External block. Same color means sharing the same weight. (Color figure online)

4.1 Encoder Component

The *trend, period, closeness* fragments share the same encoder component that comprises some downsample blocks, which are composed of stride-2 convolution layer, batch normalization layer and leaky relu layer, as shown in Fig. 4.

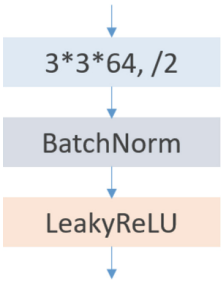

Fig. 4. Downsample block structure. 3*3: kernel size; 64: channel number; /2: Stride-2; BatchNorm: Batch Normalization.

Downsample Blocks Sharing. We share one group of downsample blocks among *trend, period, closeness* instead of using three groups because it is obvious that the spatial pattern of a city is relatively stable among *trend, period, closeness* fragments as shown in Fig. 5. This structural change cuts down the amount of parameter, leads the model to learn more general representations and reduces the risk of overfitting.

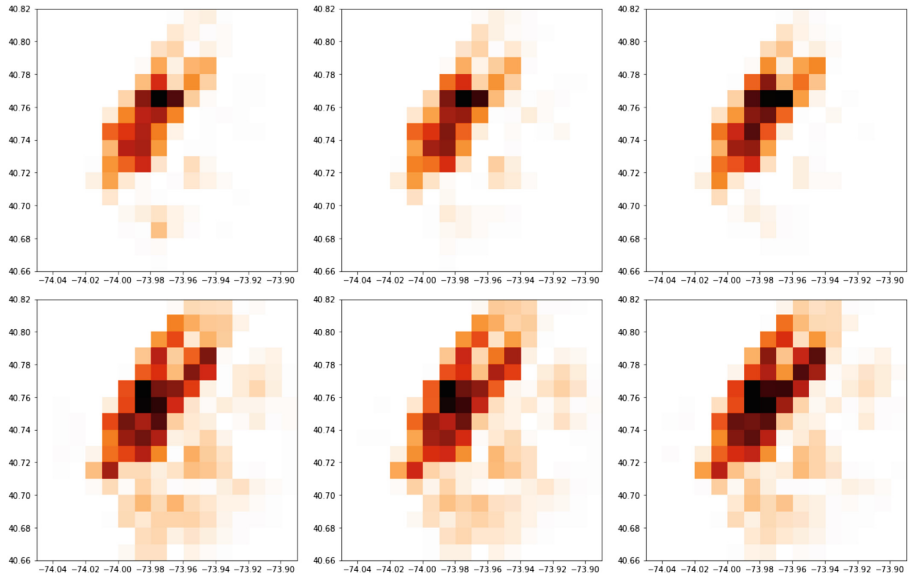

Fig. 5. Spatial dependency example of *trend, period and closeness*.

Stried-2 Convolution. A stack of convolution layers can capture the spatial dependencies among regions in the city. For serval benefits, we use the stride-2 convolution to downsample the input.

The first benefit is about feature extraction. An autoencoder whose output dimension is less than the input dimension is called under complete. We can obtain most salient features of the training data from the under complete autoencoder because it is forced to learn an under complete representation. Motivated by under complete autoencoder, we consider stride-2 convolution can help us get more general representations.

The second benefit is about computation. The task of traffic flow prediction needs a real-time reaction, so we want to reduce the computation as much as possible while ensuring effectiveness. We use flops of convolution to measure the computation of the model as follows

$$FLOPs = 2\left(C_{in}K^2 + 1\right)H'W'C_{out} \tag{1}$$

where $H'W'$, the output size of stride-n convolution, is the $1/n^2$ of normal convolution, so the FLOPs will also be the $1/n^2$ [18]. C_{in} and C_{out} are input channels and output channels respectively. And K means kernel size.

The third benefit is about effective receptive field size. Citywide traffic flow prediction requires each pixel in the output to have a large effective receptive field in the input. For example, each 3×3 convolution layer increases the effective receptive field size by only 2. But using stride-2 convolution layer, each layer increases the effective receptive field more efficiently as follows

$$R_l = K_l, l = 0 \tag{2}$$

$$S_l = S_0 * S_1 * \ldots * S_{l-1}, l = 1, 2, ..m \tag{3}$$

$$R_l = (R_{l-1} - 1) * S_l + K_l, l = 1, 2, ..m \tag{4}$$

where R is the receptive field size, K is the convolution kernel size, S is the convolution stride and layers are numbered by l.

In FedNet, we stack $(m+1)$ downsample blocks upon the input sequence $[\boldsymbol{X}^t, \boldsymbol{X}^p, \boldsymbol{X}^c]$. Take the *closeness* property as an example, which is shown as follows

$$\boldsymbol{X}_l^{dc} = down_l(\boldsymbol{X}^c), l = 0 \tag{5}$$

$$\boldsymbol{X}_l^{dc} = down_l\left(\boldsymbol{X}_{l-1}^{dc}\right), l = 1, 2, \ldots m \tag{6}$$

where $\boldsymbol{X}_l^{dc}$ is the *closeness* property output of No. l downsample block.

4.2 Fusion Component

The fusion component comprises two sub-components: Temporal fusion sub-component and the spatial fusion one. The Temporal fusion sub-component comprises some temporal fusion blocks, which are composed of the concatenate layer, one

by one convolution layer and leaky relu layer, as shown in Fig. 6. The spatial fusion one is composed of some skip connections as shown in Fig. 3. Like downsample blocks sharing above, we share only one temporal fusion block among different spatial level, because we want the model to capture a general temporal pattern of *trend, period, closeness*, which is independent to specific spatial level. This structural change also cuts down the amount of parameter, leads the model to learn more general representations and reduces the risk of overfitting.

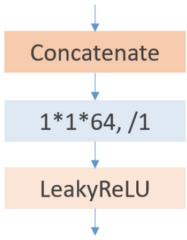

Fig. 6. Temporal fusion block structure. 1*1: kernel size; 64: channel number; /1: Stride-1.

1*1 Convolution. Instead of using parametric-matrix-based fusion [11], which requires a different size for different input feature and is unstable with weight initialization methods, we adopt one by one convolution [19] as our temporal fusion method. One by one convolution was first introduced by Lin et al. [20] to generate a deeper network without simply stacking more layers. However, in our paper, we majorly consider it as a feature transformation method. *Although 1*1 convolution is a 'feature pooling' technique, there is more to it than just sum pooling of features across various channels/features maps of a given layer*. Because this transformation is learned through the (stochastic) gradient descent, so we can use it to learn the different influence of *trend, period, closeness* according to the training data instead of manual setting specify weights.

In this sub-component, for every spatial level, we first concatenate three downsample output and then pass the intermediate output to 1*1 convolution layer that is followed by a non-linear activation layer like leaky relu, as shown as follows

$$\boldsymbol{X}_{m-l}^{time} = time\left(\boldsymbol{X}_{l}^{dt}, \boldsymbol{X}_{l}^{dp}, \boldsymbol{X}_{l}^{dc}\right), l = 0, 1, \ldots \mathrm{m} \tag{7}$$

where $\boldsymbol{X}_{m-l}^{time}$ is the temporal aggregation of No. $(m-l)$ spatial level.

Skip Connection. To capture the city-wide spatial dependencies, we stack some convolution layers in the encoder because one convolution layer only accounts for near spatial dependencies. However, if we only consider the output of the final convolution layer which contains coarse semantic information of city-wide spatial dependencies, we will lose too much detail information from low-level spatial dependencies like distinct-wide spatial dependencies, especially when we use stride-2 convolution. Inspired by Skip Architecture, we add some skip connections between corresponding temporal

fusion block and upsample blocks to model these spatial properties respectively. Especially, we make use of the detail information from lower spatial level. This sub-component is also learned end-to-end to refine the traffic flow prediction, as shown as follows

$$\boldsymbol{X}_l^s = \boldsymbol{X}_{m-l}^{time} + \boldsymbol{X}_{l-1}^u, l = 1, \ldots \mathrm{m} \tag{8}$$

where $\boldsymbol{X}_l^s$ is the spatial fused feature of No. l skip connection.

4.3 Decoder Component

The decoder component comprises some upsampling blocks, which is composed of the interpolate layer, convolution layer, batch normalization layer, and leaky relu layer, as shown in Fig. 7.

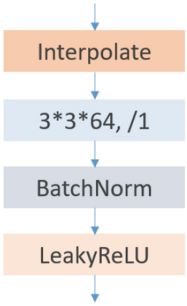

Fig. 7. Upsampling component block structure.

Instead of using transposed convolution that causes Checkboard Artifacts [21], we use factor-2 nearest-neighbor interpolation [22] to upsample the input feature until getting the expected size output, which is simple yet effective as follows

$$\boldsymbol{X}_l^u = up_l\left(\boldsymbol{X}_l^s\right), l = 0, 1, \ldots m \tag{9}$$

$$\boldsymbol{X}_{dec} = \boldsymbol{X}_m^u \tag{10}$$

where $\boldsymbol{X}_l^u$ is the output of No. l upsample block and $\boldsymbol{X}_{dec}$ is the output of the final upsample block.

4.4 External Component

External encoder component is composed of the linear layer, dropout layer, and leaky relu layer, as shown in Fig. 8.

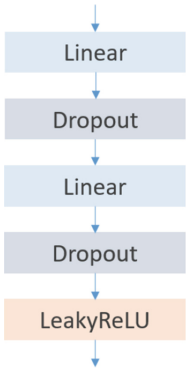

Fig. 8. External component structure.

Referencing previous work [10, 11], we know that traffic flow can be affected by many external factors like the date. In our experiments, we mainly consider date features, which can be obtained directly, like is_month_start. Also, we use weather features that can be approximated by the forecasting weather. Then, we stack two linear layers to process external input which is the feature vector that represents the external factors. Finally, we use the output of external component $\boldsymbol{X}_{ext}$ to adjust $\boldsymbol{X}_{dec}$ by summing them up as follows

$$\widehat{\boldsymbol{X}_t} = \tanh(\boldsymbol{X}_{dec} + \boldsymbol{X}_{ext}) \tag{11}$$

4.5 Model Training

We finally present the training method of our model. FedNet is trained end to end. Especially, to tackle some hard task, at training stage, we transform the one-time-step prediction problem into a multi-time-step sequence prediction problem, and then adopt Teacher Forcing and design a new loss function as shown in Fig. 9. At inferencing stage, for the sake of comparing, we only perform one-time-step prediction and use *f_RMSE* (factored Root Mean Square Error) as the metric.

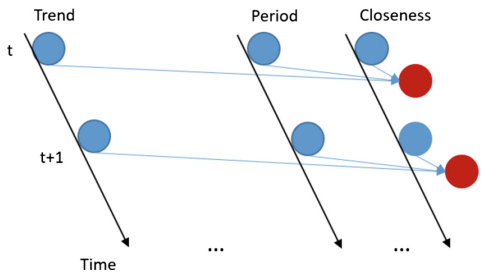

Fig. 9. Sequence predicting and teacher forcing training.

Teacher Forcing. Some external factors like events may tremendously change the traffic flow in continuous time steps. Although we employ external component, we cannot collect all the information due to many realistic limitations. If we train the model by one-time-step predicting, then the model will focus on the next time step and is not robust enough to tackle some unexpected continuous volume change. For this issue, we consider the relationship among multiple future time steps as sequence dependencies. To address this issue, we model the sequence dependencies by force the model to predict multi-time-step at the training stage. Instead of using recursive multi-step forecast which is hard to train because of slow convergence, model instability and poor skill, we adopt teacher forcing, which *works by using the actual output from the training dataset at the current time step y(t) as input in the next time step X(t + 1) rather than the output generated by the network*.

For one time-step training, our model is trained to minimize the MSE (mean square error) between the predicted flow matrix and the ground truth:

$$L_t^s = \frac{1}{z}\sum\nolimits_i (x_i - \widehat{x_i})^2 \tag{12}$$

where $\widehat{x_i}$ and x_i are the predicted value and the ground truth, respectively; z is the number of all predicted values.

For multi-time-step training, we define the corresponding loss as the average loss of $(j+1)$ time-step.

$$L_t^m = \frac{1}{j+1}(L_t^s + L_{t+1}^s + \ldots + L_{t+j}^s) \tag{13}$$

To evaluate our model, we design the factored Root Mean Square Error as

$$factor = \frac{H * W}{available} \tag{14}$$

$$f\_RMSE = \sqrt{factor * \frac{1}{z}\sum\nolimits_i (x_i - \widehat{x_i})^2} \tag{15}$$

where *available* is the amount of available regions and $H * W$ is the amount of all regions.

5 Experiments

5.1 Experiment Settings

Datasets. We use 4 different sets of data as shown in Table 1, as detailed as follows.

- TaxiBJ [11]: *Trajectory data is the taxicab GPS data and meteorology data in Beijing from four intervals: 1st Jul. 2013 - 30th Oct. 2013, 1st Mar. 2014 - 30th Jun. 2014, 1st Mar. 2015 - 30th Jun. 2015, 1st Nov. 2015 - 10th Apr. 2016. Using*

Definition 2, we obtain two types of crowd flows. We choose data from the last four weeks as the testing data, and all data before that as training data.

- BikeNYC [11]: *Trajectory data is taken from the NYC Bike system in 2014, from Apr. 1st to Sept. 30th. Trip data includes: trip duration, starting and ending station IDs, and start and end times. Among the data, the last 10 days are chosen as testing data, and the others as training data.*
- CitiBikeNYC: Trajectory data is taken from the NYC Citi Bike System in 2014, from Apr. 1st to Sept. 30th. Among the data, the last 10 days are chosen as testing data, and others as training data. Among the data, the last 10 days are chosen as testing data, and others as training data.
- TaxiNYC: Trajectory data is taken from the Taxi & Limousine Commission System in 2014, from Apr. 1st to Sept. 30th. Among the data, the last 10 days are chosen as testing data, and the other as training data. Among the data, the last 10 days are chosen as testing data, and the others as training data.

Table 1. Datasets

Attribute	Dataset			
	BikeNYC	TaxiBJ	CitiBikeNYC	TaxiNYC
Data type	Bike GPS	Taxi GPS	Bike GPS	Taxi GPS
Location	New York	Beijing	New York	New York
Gird map size	(16, 8)	(32, 32)	(16, 16)	(16, 16)
Time span	4/1/2014 - 9/30/2014	7/1/2013–10/30/2013 3/1/2014–6/30/2014 3/1/2015–6/30/2015 11/1/2015–4/10/2016	4/1/2014–9/30/2014	4/1/2014–9/30/2014
Time interval	1 h	30 min	1 h	1 h
Available time Interval	4392	22459	4392	4392
Holidays	20	41	20	20
Weather	\	16 types (e.g. Sunny)	\	\
Temperature/°C	\	[− 24.6, 41.0]	\	\
Wind/mph	\	[0, 48.6]	\	\

Baselines. We compare our model with six different baselines as detailed as follows.

- ARIMA: *Auto-Regressive Integrated Moving Average (ARIMA) is a well-known model for understanding and predicting future values in a time series.*
- SARIMA: *Seasonal ARIMA.*

- VAR: *Vector Auto-Regressive (VAR) is a more advanced spatial-temporal model, which can capture the pairwise relationships among all flows, and has heavy computational costs due to the large number of parameters.*
- DeepST: *a deep neural network (DNN)-based prediction model for spatial-temporal data.*
- ST-ResNet: a deep spatial-temporal residual network to collectively predict traffic flow of every region, which shows state-of-the-art results on crowd flow prediction.

Hyper Parameters. For input, we fix trend, period, closeness to one week ago, one day ago and one hour ago, which have the same fragment length 1. For teacher forcing, the amount of predicting time step is 4. Our model is implemented with PyTorch 0.4.1, a popular Deep Learning Python library [23]. The stride-2 convolutions of all downsample blocks use 64 kernels of size 3 * 3, the convolution of time block use 64 kernels of size 1 * 1 and those of upsample blocks use 64 kernels of size 3 * 3. We select part of the data as training data and then use it to train the model, the rest of the data like the final 10 days' data is the test set. The batch size is 32. We use Adam [24] with the default learning rate 0.001 with a fixed number of epochs (e.g. 100 epochs) and the batch size is 32, and decide whether to update the parameters based on the validation score.

5.2 Experiment Results

Table 2 shows the results of our models and other baselines on BikeNYC and TaxiBJ. Being different from BikeNYC, TaxiBJ is another type of traffic flow, including inflow and outflow [11]. Comparing with the previous models, FedNet_ETS_T, which adopts the encoder-decoder framework, temporal fusion, spatial fusion, and teacher forcing, has 9.00% and 7.01% lower RMSE respectively. Table 3 shows the results of our model on CitiBikeNYC and TaxiNYC. Comparing with the previous best model, FedNet_ETS_T has 9.92% and 3.00% lower RMSE respectively. These results demonstrating the effectiveness of our model.

Table 2. Comparisons with baselines on BikeNYC and TaxiBJ. The results of ARIMA, SARIMA, VAR, ST-ANN, and DeepST are taken from (Zhang et al. 2017).

Model	RMSE	
	BikeNYC	TaxiBJ
ARIMA	*10.07*	*22.78*
SARIMA	*10.56*	*26.88*
VAR	*9.92*	*22.88*
DeepST	*7.43*	*18.18*
ST-ResNet	6.33	16.69
FedNet_ET	5.90	16.22
FedNet_ETS	5.83	15.73
FedNet_ETS_T	**5.76**	**15.52**

Table 3. Comparisons with baselines on CitiBikeNYC and TaxiNYC.

Model	RMSE	
	CitiBikeNYC	TaxiNYC
ST-ResNet	8.57	22.52
FedNet_ET	7.84	24.55
FedNet_ETS	7.84	22.47
FedNet_ETS_T	**7.72**	**21.85**

5.3 Ablation Studies

Considering previous works [10, 11] have approved the effectiveness of temporal fusion and external features, so we majorly discuss the effectiveness of the encoder-decoder framework, spatial fusion, and teacher forcing.

- Encoder-Decoder Framework: For BikeNYC and CitiBikeNYC, comparing to ST-ResNet, FedNet_ET brings 6.79% and 8.52% lower RMSE respectively, which shows the effectiveness of the encoder-decoder framework. However, for TaxiBJ, we get less improvement, and the result of TaxiNYC even becomes worse, which shows it is hard for a simple encoder-decoder framework to tackle hard dataset which has a higher RMSE on baseline models.
- Spatial Fusion: For TaxiBJ and TaxiNYC, comparing to FedNet_ET, FedNet_ETS brings 3.02% and 8.47% lower RMSE respectively, which shows the effectiveness of spatial fusion. However, for BikeNYC and CitiBikeNYC, we get relatively the same result, which shows spatial fusion is unnecessary for an easy dataset which has a lower RMSE on baseline models.
- Teacher Forcing: For TaxiNYC, comparing to FedNet_ETS, FedNet_ETS_T brings 2.76% lower RMSE, which is nearly twice as much improvement comparing to the improvement on the other datasets which is 1.20%, 1.34%, 1.53% respectively. The overall result demonstrates the effectiveness of teacher forcing, and the result on TaxiNYC shows that modeling sequence dependencies is more effective on the dataset that has a higher RMSE on baseline models.

6 Conclusion and Future Work

In this paper, combining the benefits of the end to end encoder-decoder framework, spatial fusion, and teacher forcing, we propose an effective model, called FedNet, to predict traffic flow in each region of a city. We conduct extensive experiments on two types of traffic flows (new-flow/end-flow and inflow/outflow) in New York City and Beijing to demonstrate that the FedNet outperforms five well-known methods. These results confirm that our model is better and more applicable to the traffic flow prediction. The code and datasets will be released at GitHub.

In the future, we will explore appropriate fusion mechanisms for multiple vehicle data (e.g. taxi, bike, bus, subway). Also, we will consider the multi-time-step predicting task at both training stage and inferencing stage, which is much harder.

Acknowledgement. The research is supported by National Natural Science Foundation of China (No. 61772560), and Natural Science Foundation of Hunan Province (No. 2019JJ40388).

References

1. Hoang, M.X., Zheng, Y., Singh, A.K.: FCCF: forecasting citywide crowd flows based on big data. In: Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, p. 6. ACM (2016)
2. Deep Learning Book Homepage. http://www.deeplearningbook.org/. Accessed 06 Mar 2019
3. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3431–3440 (2015)
4. What is Teacher Forcing for Recurrent Neural Networks? https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/. Accessed 06 Mar 2019
5. Fan, Z., Song, X., Shibasaki, R., Adachi, R.: Citymomentum: an online approach for crowd behavior prediction at a citywide level. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 559–569. ACM (2015)
6. Song, X., Zhang, Q., Sekimoto, Y., Shibasaki, R.: Prediction of human emergency behavior and their mobility following large-scale disaster. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 5–14. ACM (2014)
7. Abadi, A., Rajabioun, T., Ioannou, P.A.: Traffic flow prediction for road transportation networks with limited traffic data. IEEE Trans. Intell. Transp. Syst. **16**(2), 653–662 (2015)
8. Silva, R., Kang, S.M., Airoldi, E.M.: Predicting traffic volumes and estimating the effects of shocks in massive transportation systems. Proc. Natl. Acad. Sci. **112**(18), 5643–5648 (2015)
9. Xu, Y., Kong, Q.J., Klette, R., Liu, Y.: Accurate and interpretable bayesian mars for traffic flow prediction. IEEE Trans. Intell. Transp. Syst. **15**(6), 2457–2469 (2014)
10. Zhang, J., Zheng, Y., Qi, D., Li, R., Yi, X.: DNN-based prediction model for spatial-temporal data. In: Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 92. ACM (2016)
11. Zhang, J.B., Zheng, Y., Qi, D.K.: Deep spatio-temporal residual networks for citywide crowd flows prediction. In: Thirty-First AAAI Conference on Artificial Intelligence, pp. 1655–1661 (2017)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
13. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)
14. Shi, X.J., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K., Woo, W.C.: Convolutional LSTM network: a machine learning approach for precipitation nowcasting. In: Advances in Neural Information Processing Systems, pp. 802–810 (2015)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

16. Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: a deep convolutional encoder-decoder architecture for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **39**(12), 2481–2495 (2017)
17. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9906, pp. 694–711. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46475-6_43
18. Molchanov, P., Tyree, S., Karras, T, Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv preprint arXiv:1611.06440 (2016)
19. One by One [1 x 1] Convolution – counter-intuitively useful https://iamaaditya.github.io/2016/03/one-by-one-convolution/. Accessed 06 Mar 2019
20. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
21. Deconvolution and checkerboard artifacts. https://distill.pub/2016/deconv-checkerboard/. Accessed 06 Mar 2019
22. Nearest-neighbor_interpolation. https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation. Accessed 06 Mar 2019
23. Pytorch. https://pytorch.org/. Accessed 06 Mar 2019
24. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)