Matrix Factorization for Spatio-Temporal Neural Networks with Applications to Urban Flow Prediction*

Zheyi Pan[1], Zhaoyuan Wang[4], Weifeng Wang[1], Yong Yu[1], Junbo Zhang[2,3,4], Yu Zheng[2,3,5]
[1]Department of Computer Science and Engineering, Shanghai Jiaotong University, China
[2]JD Intelligent Cities Research, China; [3]JD Intelligent Cities Business Unit, China
[4]Institute of Artificial Intelligence, Southwest Jiaotong University, China
[5]School of Computer Science and Technology, Xidian University, China
{zhpan,wfwang,yyu}@apex.sjtu.edu.cn;wang_zhaoyuan@foxmail.com;{msjunbozhang,msyuzheng}@outlook.com

ABSTRACT

Predicting urban flow is essential for city risk assessment and traffic management, which profoundly impacts people's lives and property. Recently, some deep learning models, focusing on capturing spatio-temporal (ST) correlations between urban regions, have been proposed to predict urban flows. However, these models overlook latent region functions that impact ST correlations greatly. Thus, it is necessary to have a framework to assist these deep models in tackling the region function issue. However, it is very challenging because of two problems: 1) how to make deep models predict flows taking into consideration latent region functions; 2) how to make the framework generalize to a variety of deep models. To tackle these challenges, we propose a novel framework that employs matrix factorization for spatio-temporal neural networks (MF-STN), capable of enhancing the state-of-the-art deep ST models. MF-STN consists of two components: 1) a ST feature learner, which obtains features of ST correlations from all regions by the corresponding sub-networks in the existing deep models; and 2) a region-specific predictor, which leverages the learned ST features to make region-specific predictions. In particular, matrix factorization is employed on the neural networks, namely, decomposing the region-specific parameters of the predictor into learnable matrices, *i.e.,* region embedding matrices and parameter embedding matrices, to model latent region functions and correlations among regions. Extensive experiments were conducted on two real-world datasets, illustrating that MF-STN can significantly improve the performance of some representative ST models while preserving model complexity.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**; **Data mining**; • **Computing methodologies → Neural networks**.

KEYWORDS

Urban flow; neural networks; matrix factorization

*Yu Zheng and Junbo Zhang are the corresponding authors.

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357832

ACM Reference Format:
Zheyi Pan[1], Zhaoyuan Wang[4], Weifeng Wang[1], Yong Yu[1], Junbo Zhang[2,3,4], Yu Zheng[2,3,5]. 2019. Matrix Factorization for Spatio-Temporal Neural Networks with Applications to Urban Flow Prediction. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3357384.3357832

1 INTRODUCTION

Accurately predicting citywide flows, such as the total crowd flows entering and leaving a region during a given time interval [29], is an essential task for the development of an intelligent city, as it can provide insights to city administrators for risk assessment, traffic management, and urban planning. Particularly, in risk assessment, by knowing that overwhelming crowds will stream into a region ahead of time, government can implement traffic control, send out warnings, or even evacuate people, to prevent tremendous risks to public safety (*e.g.,* the catastrophic stampede caused by social riots, which endangers huge life and economic losses for people).

Recent advances of mobile technologies generate a large collection of citywide flow data, enabling researchers to solve this challenging problem from a data-driven perspective. In the beginning, some studies adopted traditional machine learning methods, such as probabilistic graphical models, to predict urban flows [5, 10]. However, these models cannot effectively learn high-level ST representation from raw input data. Thereafter, the success of deep learning boosted the research of ST data mining and in particular the flow prediction [23, 24, 29]. These models are attributed to the powerful representation learning of deep network components, such as convolution neural network (CNN) and recurrent neural network (RNN), capable of learning high-level ST features to make better predictions. In general, these existing deep ST models consist of two main components: a ST feature learner (*e.g.,* a network consisting of CNNs, RNNs, or both) and a predictor (*e.g.* a fully connected network), as illustrated in Figure 1 (a). Concretely, the

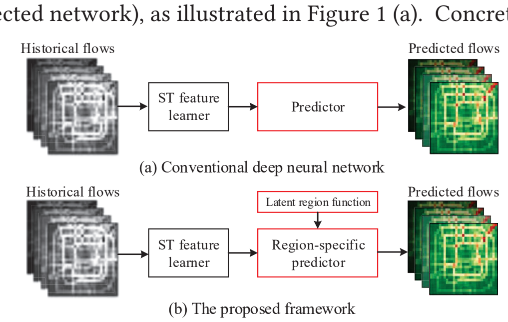
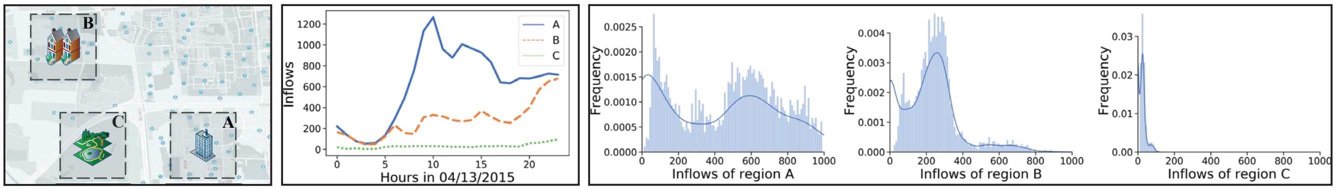


Figure 1: Conventional model vs. our MF-STN.



(a) Three real-world regions (b) A day's inflow of regions (c) The inflow distribution of three regions, respectively.

Figure 2: The example of region discrepancy in flow data.

ST feature learner can fully leverage training samples of all regions to capture complex ST correlations by sharing parameters. On this basis, the predictor makes the final prediction for each region. However, the predictor also uses shared parameters for all regions' flow prediction, which in fact makes it difficult to capture diverse flow trends caused by the latent function of a region.

More specifically, as shown in Figure 2 (a), A, B, and C are three real-world regions[1] with different latent functions, corresponding to a business, a residential and a park zone, respectively. On workdays, as citizens go to work in the morning and return home at night, Region A witnesses upward inflows in the morning while Region B experiences high traffic during the evening, as shown in Figure 2 (b). In addition, comparing with Region A and B, Region C has much less inflows, because fewer people go to park on workdays. As a result, Figure 2 (c) shows the totally different flow distributions of these three regions. Thus, a predictor using shared parameters to predict all regions' flows can hardly capture such diverse flow distributions, leading to the limited predictive ability.

To make more accurate predictions, it is necessary to have a general deep learning framework to assist these off-the-shelf deep ST models in tackling the region function issue. However, it is very challenging because of the following two problems:

- How to make models *collaboratively* predict urban flows with consideration of latent region functions?
  First, learning different latent region functions is necessary, because they have diverse impacts on regions' flow trends. Moreover, regions essentially have inherent correlations among them, *e.g.,* if two business districts exhibit similar flow patterns, the learned predictors for the two regions should be close to each other. However, it is difficult to train a predictor for a certain region taking into consideration the specific latent region function, as well as leveraging the information (*i.e.,* training samples) of other regions in addition to its own data.
- How to make the framework *portable* and *lightweight*?
  As previously discussed, there is a variety of deep ST models. How our framework generalizes to them is non-trivial. Meanwhile the framework needs to be as simple as possible to preserve model complexity, so as to prevent over-fitting and hard optimization.

To tackle these two challenges, we propose a novel deep learning framework that leverages a matrix factorization approach for modelling spatio-temporal neural networks, entitled MF-STN, to enhance the existing deep ST models. As shown in Figure 1 (b), the framework is comprised of two components: 1) a ST feature learner that is employed to capture features of ST correlations for all regions, which can be a sub-network in the existing deep models for capturing ST correlations; and 2) a region-specific predictor, which leverages the learned ST features to make a region-specific flow prediction. Our contributions are four-fold:

- We are the first to analyze different impacts of the latent region functions on urban flow trends. In light of this insight, we propose a novel deep learning framework, consisting of a ST feature learner and a region-specific predictor, capable of enhancing the state-of-the-art deep models on ST forecasting tasks.
- We propose a region-specific predictor, which is a matrix factorization based neural networks, to decompose the region-specific parameters of the predictor into learnable matrices, *i.e.,* region embedding matrices and parameter embedding matrices. As a result, the latent region functions along with the correlations among regions can be modeled.
- We illustrate that MF-STN is a portable and lightweight framework, which can be applied to a variety of existing deep ST models, including ST-ResNet [29], DMVST-Net [24], STDN [23], etc., while preserving the model complexity.
- We conduct extensive experiments on two real-world datasets. The experiment results demonstrate that MF-STN can effectively and efficiently enhance the performance of a wide range of deep ST models. Moreover, the framework has been deployed in the real-world applications.

2 PRELIMINARIES

In this section, we provide the definition and the problem statement for urban flow prediction. For brevity, the frequently used notations in this paper are presented in Table 1.

Table 1: Notations.

Notations	Description
$n_r \in \mathbb{R}$	Number of regions.
$n_t \in \mathbb{R}$	Number of timestamps.
$n_v \in \mathbb{R}$	Number of measured flow values.
$n_f \in \mathbb{R}$	Number of collected ST features.
$\tau_{\text{hist}}, \tau_{\text{pred}} \in \mathbb{R}$	Number of historical/predicted timestamps.
$\{X_i \in \mathbb{R}^{n_r \times n_v}\}$	The flow data at timestamp i.

Definition 1. ***Urban flow dataset***. *The urban flow dataset discussed in this paper is denoted as a tensor* $\mathbf{X}_{data} = [X_1, ..., X_{n_t}] \in \mathbb{R}^{n_t \times n_r \times n_v}$*, where* n_t *is the number of timestamps,* n_r *is the number of regions, and* n_v *is the number of measured values (e.g., inflow and outflow). Given an index* (i, j, k)*, where* $1 \le i \le n_t$, $1 \le j \le n_r$*, and* $1 \le k \le n_v$*, the corresponding value of tensor* $\mathbf{X}_{data}$ *at this index denotes the* k*-th flow value of the region* j *at timestamp* i*.*

Problem 1. ***Urban Flow Prediction***. *Given historical flow readings* $\mathbf{X} = [X_1, ..., X_{\tau_{hist}}] \in R^{\tau_{hist} \times n_r \times n_v}$ *of all regions at previous* τ_{hist} *timestamps, predict the future readings for the next* τ_{pred} *timestamps, denoted as* $\hat{\mathbf{Y}} = [\hat{Y}_1, ..., \hat{Y}_{\tau_{pred}}] \in R^{\tau_{pred} \times n_r \times n_v}$.

[1]Located around Peking University in Haidian District, Beijing

3 METHODOLOGY

3.1 Framework Overview

MF-STN consists of two components, a spatio-temporal feature learner and a region-specific predictor, as shown in Figure 3. We briefly overview the framework as follows.

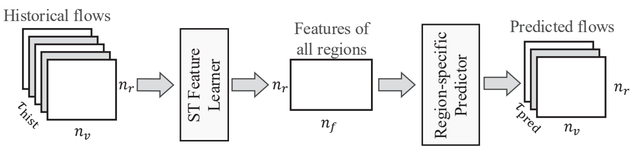


Figure 3: Framework overview

1) ***Spatio-Temporal feature learner***. This component takes all regions' flows as inputs, aiming to capture features with flows' ST correlations for each region. The outputs of the ST feature learner, *i.e.,* the ST features of all regions, will be used as inputs of the latter component, *i.e.,* the region-specific predictor.
2) ***Region-specific predictor***. This component takes the ST features produced by the ST feature learner as the inputs, and use them to make predictions. This predictor has region-specific parameters, which can be regarded as n_r neural networks, each of which makes a prediction for a single region, respectively.

In the following subsections, we will illustrate the detail structures of these two components.

3.2 Spatio-Temporal Feature Learner

As shown in Figure 3, the ST feature learner, denoted as $\mathcal{G}$, aims to collect ST features from the original data for all regions. Formally, the input of the ST feature learner is a tensor $\mathbf{X} = [X_1, ..., X_{\tau_{\text{hist}}}] \in \mathbb{R}^{\tau_{\text{hist}} \times n_r \times n_v}$, denoting all regions' flow readings in the previous τ_{hist} timestamps. Then, the input $\mathbf{X}$ is mapped by the ST feature learner $\mathcal{G}$ into a matrix $F = \mathcal{G}(\mathbf{X}) = [f_1, ..., f_{n_r}] \in \mathbb{R}^{n_r \times n_f}$, where each f_i is a vector denoting the ST feature values of i-th region.

As many deep ST models, *e.g.,* ST-ResNet [29], DMVST-Net [24], and STDN [23], were proposed to capture ST correlations for all regions simultaneously, we can directly adopt the sub-networks of these models, *i.e.,* the components capturing ST correlations, as the ST feature learner. One straightforward way to extract the ST feature learner from a conventional deep model is using the prefix network, *i.e.,* the original network removing one or several suffix layers. For example, we can use the whole residual neural network in ST-ResNet [29] by removing its last layer as the ST feature learner. Note that we do not make any other constraints on the choice of existing deep ST models, except that the model can be trained end-to-end by back-propagation. The portability of MF-STN is further discussed in Section 3.5.

3.3 Region-Specific Predictor

Conventionally, existing deep ST networks employ a predictor with shared parameters for all regions' flow prediction. However, due to different regions' latent functions, it is essential to have a predictor with an individual set of parameters for each region, respectively. Meanwhile, as there are inherent correlations among regions, we need to collaboratively learn the region-specific parameters. Previously in non-deep models, the problem was solved by regularizing the region-specific parameters according to region similarity [14]. However, it needs prior knowledge to make assumptions on the similarity function, such as defining similarity scores based on the difference between the distributions of POIs in regions or pairwise distance between regions, which are often unavailable (*e.g.,* lack of external data) or unreliable (assumptions are set by experience, not always accurate or do not even hold.). Therefore, a question arises: *can we directly learn region-specific parameters in a collaborative manner while considering the inherent region correlations?*

Looking at this issue from another view, the learning process of region-specific parameters can be regarded as a collaborative filtering task, with the region and the parameter corresponding to the user and the item respectively, and the parameter values at a region being the score that a user rates an item. Thus, we propose to build a region-specific predictor, which is a deep neural network, consisting of some matrix factorization based dense (MFDense) layers and non-linear activating functions, to learn high-level region-specific features and make predictions, as shown in Figure 4 (a). Formally, suppose $F = [f_1, ..., f_{n_r}] \in \mathbb{R}^{n_r \times n_f}$ denotes the output of the ST feature learner where f_i is the learned features for the i-th region, and $\{\mathcal{H}^{(1)}, ..., \mathcal{H}^{(m)}\}$ denotes the m MFDense layers in the region-specific predictor. Then the predicted values can be calculated by chaining all layers, formulated as:

$$\hat{\mathrm{Y}} = \mathcal{H}^{(m)}(\sigma(...\mathcal{H}^{(1)}(\sigma(F))...)),$$

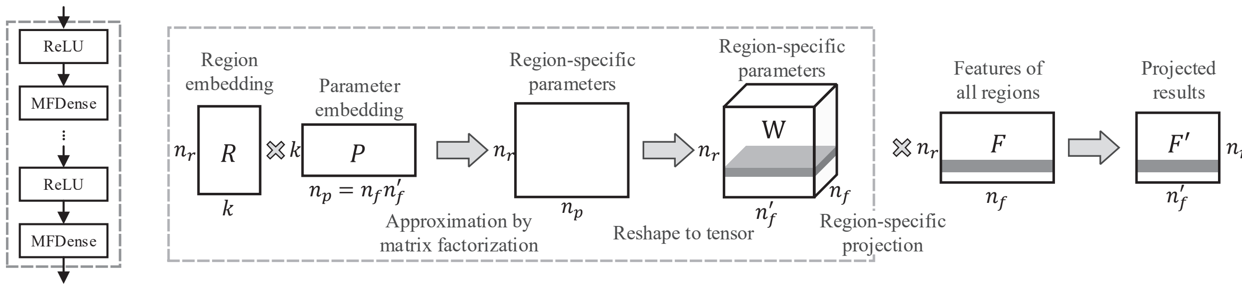
where σ is the activation function, such as ReLU and sigmoid function. More specifically, each MFDense layer contains n_r sets of parameter values, *i.e.,* n_r weight matrices, to model feature weights in each region, respectively. To capture correlations among regions, we employ the insights from the matrix factorization technique to calculate n_r weight matrices from two small learnable matrices, *i.e,* a region embedding matrix and a parameter embedding matrix. Next, we will detail the MFDense layer, and discuss its interpretability.

Matrix Factorization Based Dense Layer

MFDense layer aims to learn high-level region-specific features. As shown in Figure 4 (b), suppose the input of MFDense is a matrix $F = [f_1, ..., f_{n_r}] \in \mathbb{R}^{n_r \times n_f}$ which denotes features for each region, and its output is a matrix $F' = [f'_1, ..., f'_{n_r}] \in \mathbb{R}^{n_r \times n'_f}$, representing the region-specific high-level features after projection. Being different from the standard dense layer, that shares a single weight matrix for all regions, we employ a weight tensor $\mathbf{W} = [W_1, ..., W_{n_r}] \in \mathbb{R}^{n_r \times n'_f \times n_f}$, where $W_i \in \mathbb{R}^{n'_f \times n_f}$ is the weight matrix to project the i-th region's features. However, directly using $\mathbf{W}$ in networks has two severe problems:

1) $\mathbf{W}$ has $n_r \times n'_f \times n_f$ parameters to be optimized, which is much more than the standard dense layer with $n'_f \times n_f$ parameters, especially when n_r is large in real-world applications. As deep models with excessive parameters could be hardly optimized and easily over-fitting, such a method does not work in practice.
2) As previously mentioned, there are inherent correlations among regions. Directly using this big weight tensor makes the flow prediction separately for different regions. As a result, the inherent correlations among regions are ignored.

To tackle the aforementioned problems, we can leverage the inherent correlations among regions, where some regions are similar in certain ways, indicating that weight tensor $\mathbf{W}$ has redundant



(a) Region-specific predictor (b) Details of MFDense layer

Figure 4: Detail structure of region-specific predictor.

information. So we adopt the insights from collaborative filtering (region≈user, parameter≈item) and matrix factorization, making the assumption that $\mathbf{W}$ can be approximated by the dot product of two matrices, *i.e.,* a region embedding matrix $R = [r_1, ..., r_{n_r}] \in \mathbb{R}^{n_r \times k}$ and a parameter embedding matrix $P = [p_1, ..., p_{n_p}] \in \mathbb{R}^{n_p \times k}$, where $n_p = n_f n'_f$ is the number of parameters of a dense layer for each region and $k \ll n_r, n_p$ indicates the embedding dimension. As shown in Figure 4 (b), $\mathbf{W}$ can be formulated as:

$$\mathbf{W} = \text{reshape}(R \cdot P^\top),$$

where $\cdot$ is the matrix dot product, and the *reshape* operator reforms the weight matrix to the target three dimensional weight tensor $\mathbf{W}$. In this way, the learnable targets become R and P, instead of $\mathbf{W}$. After getting $\mathbf{W}$, the output features of a MFDense layer can be calculated by:

$$f'_i = W_i \cdot f_i + b_i, \quad i = 1 \to n_r,$$

where i is the region index, $f_i \in \mathbb{R}^{n_f}$ is the input features, $f'_i \in \mathbb{R}^{n'_f}$ is the output features, $W_i \in \mathbb{R}^{n'_f \times n_f}$ is the weight matrix, and $b_i \in \mathbb{R}$ is the region-specific bias that can be calculated with the same strategy as W_i.

Discussion on Matrix Factorization Based Dense Layer
MFDense layer can effectively learn region-specific parameter values for each region because of the following two reasons. First, as the model learns two small matrices R and P instead of the weight tensor $\mathbf{W}$, the number of training parameters is significantly reduced, *i.e.,* from $n_r n_p$ to $(n_r + n_p)k$, where $k \ll n_r, n_p$. In this way, it makes the model easier to be optimized in practice. Second, region embedding matrix R can help retain the information about $\mathbf{W}$ in a compact manner, and it can describe inherent correlations among regions by showing the region similarity. We conduct empirical experiments to show the interpretability of region embedding matrix R in Section 5.

3.4 Algorithm & Optimization

Suppose that MF-STN is optimized by a differentiable loss function $\mathcal{L}_{\text{train}}$ (*e.g.,* mean square error), which denotes the difference between the ground truth and the prediction values. Then, MF-STN can be trained end-to-end by back-propagation.

More specifically, suppose that the region-specific predictor has m MFDense layer $\{\mathcal{H}^{(1)}, ..., \mathcal{H}^{(m)}\}$, where each $\mathcal{H}^{(i)}$ has a weight tensor $\mathbf{W}^{(i)} \in \mathbb{R}^{n_r \times n_f n'_f}$, a region embedding matrix $R^{(i)}$, and a parameter embedding matrix $P^{(i)}$. Then the gradient of weight tensor $\nabla_{\mathbf{W}^{(i)}} \mathcal{L}_{\text{train}}$ can be calculated by chain rule, which is the same as a standard dense layer. After that, the gradient of $R^{(i)}$ is:

$$\nabla_{R^{(i)}} \mathcal{L}_{\text{train}} = \nabla_{\mathbf{W}^{(i)}} \mathcal{L}_{\text{train}} \cdot P^{(i)},$$

while the gradient of $P^{(i)}$ is:

$$\nabla_{P^{(i)}} \mathcal{L}_{\text{train}} = (\nabla_{\mathbf{W}^{(i)}} \mathcal{L}_{\text{train}})^\top \cdot R^{(i)}.$$

As for the ST feature learner $\mathcal{G}$, the gradient of any parameter $\theta \in \mathcal{G}$ can be expressed as:

$$\nabla_\theta \mathcal{L}_{\text{train}} = \nabla_F \mathcal{L}_{\text{train}} \nabla_\theta \mathcal{G},$$

where F is the output features of $\mathcal{G}$, and $\nabla_F \mathcal{L}_{\text{train}}$ can be calculated by applying chain rule on the region-specific predictor.

Algorithm 1 outlines the training process of MF-STN. We first construct training data (Lines 1-5). Then we iteratively optimize MF-STN by gradient descent (Lines 7-15) until the stopping criteria is met. In this loop, we first apply the forward-backward operation on network with a random batch data (Lines 8-9) to get gradients of parameters, and then update the parameters within $\{\mathcal{H}^{(i)}\}$ (Line 10-12) and $\mathcal{G}$ (Line 13-14) by gradient descent respectively.

Algorithm 1: Training algorithm of MF-STN

```
Input    :Flow data 𝒳 = (X_1, ..., X_{n_t}).
1  𝒟_train ← ∅
2  for available t ∈ {1, ..., n_t} do
3      X ← (X_{t−τ_hist+1}, ..., X_t) // input data
4      Y ← (X_{t+1}, ..., X_{t+τ_pred}) // label
5      put {X, Y} into 𝒟_train
6  initialize all trainable parameters
7  do
8      randomly select a batch 𝒟_batch from 𝒟_train
9      forward-backward on ℒ_train by 𝒟_batch
10     for i ∈ {1, ..., m} do
11         R^(i) = R^(i) − α∇_{R^(i)} ℒ_train // α is learning rate
12         P^(i) = P^(i) − α∇_{P^(i)} ℒ_train
13     for θ ∈ 𝒢 do
14         θ = θ − α∇_θ ℒ_train
15 until stopping criteria is met
16 Output: learned MF-STN model
```

3.5 Discussion on Portability and Complexity

Framework Portability
MF-STN is an extension of the conventional deep ST models, in

which the region-specific predictor can be regarded as a plugin to capture latent region function taking into consideration region correlations. Note that MF-STN should be trained end-to-end by back-propagation and the region-specific predictor needs all regions' ST features as the input. Thus, from the view of model availability, a common neural network that can output all regions' features is sufficient to be integrated with MF-STN. To illustrate the portability, we classify the conventional deep models into three categories:

1) Basic deep models, such as FNN and GRU [2].
2) Basic ST models, such as CNN [8] and ConvGRU [1].
3) Flow prediction models, including ST-ResNet [29], DMVST-Net [24], and STDN [23].

All these models can be applied on flow prediction task. In the experiments (Section 5.2), we show that these models can have significant improvement when they are integrated with MF-STN.

Framework Complexity

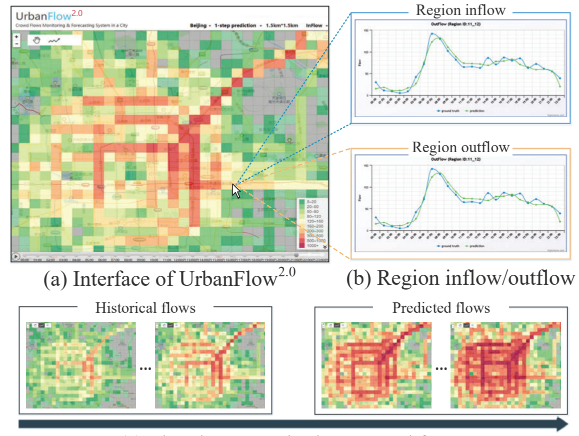
We discuss the framework complexity from the following two aspects to show that MF-STN is lightweight:

1) **Number of parameters**. MF-STN only applies stacked MF-Dense layers after the ST feature learner, each of which contains $(n_r + n_p)k$ parameters. Taking the flow prediction task in Beijing as an example, $n_r = 1024$, k is a small constant denoting the dimension of region embedding (*e.g.,* $k = 4$), and n_p is the number of parameters in a standard dense layer (*e.g.,* a dense layer mapping 64 hidden units to another 64 hidden units has $n_p = 4096$), it only introduces 20k additional parameters, much less than the number of parameters in the ST feature learner (*e.g.,* ST-ResNet has 1130k parameters). In Section 5.2, we conduct experiments to show that with very limited additional parameters, MF-STN can still significantly improve the performance.
2) **Training/inference time**. First, due to the complexity of ST correlations, the bottleneck of deep ST models is learning ST features. Second, as k is a very small number, the computational complexity of MFDense layer, *i.e.,* $O(kn_r n_p)$, does not introduce too much additional computational consumption, compared with the complexity of a standard dense layer $O(n_r n_p)$. Thus MF-STN does not visibly degrade the efficiency of base ST models. In Section 5.2, we also show that MF-STN can achieve better performance without degrading training/inference speed in the real-world applications.

4 FRAMEWORK DEPLOYMENT

UrbanFlow [29, 31], our previously deployed cloud-based system, is capable of monitoring the real-time crowd flows and providing the forecasting crowd flows in the near future. Now it is upgraded to version 2.0 by using our MF-STN as the bedrock model for the prediction. Here, we overview the functions of UrbanFlow briefly. More details about the system deployment please refer to [31].

Figure 5 presents the interface of UrbanFlow where each grid on the map stands for a region. The color of each grid is determined in accordance with its crowd flows, *e.g.,* "red" means dense crowd flows and "green" means sparse crowd flows. A user can select any grid on the interface and click it to see the region's detailed flows. The bottom of the interface shows a few sequential timestamps. The heatmap at a certain timestamp will be shown in the interface when a user clicks the associated timestamp. The user can watch the



(a) Interface of UrbanFlow$^{2.0}$ (b) Region inflow/outflow

(c) Flow heatmaps in the past and future

Figure 5: Interface of UrbanFlow$^{2.0}$.

movie-style heatmaps by clicking "playbutton" at the bottom-left of Figure 5 (a).

5 EVALUATION

In this section, we conduct experiments based on two real-world taxi flow prediction tasks to evaluate MF-STN. Particularly, we answer the following questions:

Q1. Can MF-STN be applied to a wide range of conventional deep ST models?

Q2. Does MF-STN effectively improve the prediction beyond the deep models it builds on and achieve state-of-the-art result?

Q3. Is MF-STN lightweight? More specifically, how does MF-STN impact the number of network parameters, the training speed, and the inference speed?

Q4. How do the settings of MF-STN, *i.e.,* the number of MFDense layers and the region embedding dimension, impact the prediction result?

Q5. Can the region embedding learned by the region-specific predictor reflect the relationship among regions?

5.1 Experimental Settings

Datasets

We conducted extensive experiments based on two real-world datasets, *i.e.,* TaxiBJ and TaxiNYC, as shown in Table 2. The detail descriptions are as follows:

Table 2: Datasets.

Dataset	TaxiBJ	TaxiNYC
Data type	Taxi trajectory	Taxi trip
Prediction target	# inflow/outflow	# pick-up/drop-off
City	Beijing	New York
Time span	2/1/2015 - 6/2/2015	1/1/2011 - 12/30/2014
Time interval	1 hour	1 hour
# Region	32×32 grids	16×16 grids
# Timestamps	3600	35064

1) **TaxiBJ**. This dataset is built from the trajectory dataset T-Drive [27, 28], which contains a large number of taxicab trajectories

from 2/1/2015 to 6/2/2015 in Beijing. We first partition Beijing city into 32×32 grids. Then for each grid, we calculate the hourly inflows and outflows from these trajectories by counting the number of taxis entering or exiting each grid. Finally, we aim to predict the future inflows and outflows for each grid based on the historical data.

2) **TaxiNYC**. This dataset records taxi trips in NYC from 2011 to 2014. Each trip contains information about pick-up/drop-off time, pick-up/drop-off locations. We partition NYC into 16×16 grids, and then count the hourly number of pick-ups and drop-offs for each grid. We aim to predict the number of pick-ups/drop-offs for each grid in the next hour(s) based on the historical data.

In both tasks, we use previous 12-hour records to predict values in the next 3 hours. Each dataset is partitioned along time axis into three non-overlapping parts, including training dataset, validation dataset, and test dataset, with a ratio of 8:1:1.

Evaluation Metrics

We adopt two widely used metrics: mean absolute error (MAE) and mean absolute percentage error (MAPE), to evaluate the accuracy of the prediction results. The metrics can be expressed as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \qquad \mathrm{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i}$$

where n is the number of values, y_i is the ground truth, and $\hat{y}_i$ is the prediction value. Note that when y_i is small, it gives a very large penalty to MAPE loss, so the loss does not easily reveal the effectiveness. Thus when calculating MAPE loss, we adopt the same strategy as [23], filtering out the samples with $y_i < 10$.

Baselines

We first compare MF-STN with some non-deep models, including:

- **HA**. Historical Average. We model the ST data as a seasonal process, with a period of one day. The prediction result for a certain timestamp is the average values of all historical data in this timestamp.
- **ARIMA**. Autoregressive Integrated Moving Average is a widely used model for time series prediction, which combines moving average and autoregression.
- **GBRT**. Gradient Boosting Regression Tree. It produces prediction results by an ensemble of some tree models.

Second, we implement many deep models, including state-of-the-art methods in flow prediction, to compare with our model. They are listed as follows:

- **FNN**. Feed Forward Neural Network. It is a network with stacked fully connected layers, activated by some sigmoid functions.
- **GRU** [2]. Gated Recurrent Unit is a simple but effective RNN structure for time series modeling. We implement a network with stacked GRUs for prediction.
- **CNN** [8]. Convolutional Neural Network can capture ST correlations on grid-based ST data, by applying the convolution operator on spatial and temporal domains simultaneously. We implement CNN as several stacked convolution layers, which are activated by the ReLU function.
- **ConvGRU** [1]. It uses convolution and GRUs to model spatial and temporal correlations respectively, and combines them to learn ST features.
- **ST-ResNet** [29]. It adopts ResNet to model ST correlations in grid-based ST prediction like CNN.
- **DMVST-Net** [24]. This model combines learned ST features from three views: a spatial view (modeled by local CNNs), a temporal view (modeled by LSTMs), and a semantic view (graph embedding, which are retrieved from similarity between regions), to make ST predictions.
- **STDN** [23]. It employs CNNs for spatial correlations, LSTMs for temporal correlations, and a periodically shifted attention mechanism to handle long-term periodic temporal shifting.

Note that all these deep models use standard dense layers as predictors. They can also be integrated with MF-STN. Next, we give the settings of MF-STN in detail.

Settings of MF-STN (Q1.)

We directly adopt the prefix networks of above deep models, *i.e.,* the network with only the suffix dense layers removed and no any other modification, as the base ST feature learner in MF-STN. In addition, we apply ReLU function and m MFDesne layers after the ST feature learner with embedding dimension k as the region-specific predictor. For simplicity, each MFDense layer has 64 hidden units, and we conduct grid search on m, k over {1,2,3}, {4,8,16} respectively. We select the best model according to the prediction accuracy on validation dataset.

All deep models are trained end-to-end by Adam optimizer [6] with gradient descent. The initial learning rate is set as 0.01, and we apply learning rate decay every 10 epochs with a ratio of 0.1. All deep models are implemented based on MXNet 1.5.1 [2], and trained/tested on Ubuntu 16.04 with a single GTX 1080GPU.

5.2 Performance Results

Effectiveness Comparison (Q2.)

The performance of non-deep models, basic deep ST models and the enhanced deep ST models are shown in Table 3. For simplicity, the deep ST models enhanced by MF-STN are named as "base model+". For all deep models, we train and test each of them at least five times, and show the results as the format "mean ± standard deviation". Moreover, the percentage values in parenthesis denote the improvement of the enhanced deep models compared with the corresponding base models.

First, conventional non-deep models, including ARIMA and GBRT, are not good enough to predict flows, because of the limitation of the model expressiveness that is unable to capture very complex ST correlations. Second, the base deep ST models have much better accuracy, compared with the non-deep models, because of their good ability in learning meaningful features. Finally, we compare the enhanced deep models by our MF-STN with their basic versions. On average, the enhanced models have 13.7% MAE improvement and 17.9% MAPE improvement in the TaxiBJ dataset, as well as 8.0% MAE improvement and 10.4% MAPE improvement in the TaxiNYC dataset. Notably when applying MF-STN on ST-ResNet, a very complex model with a large amount of parameters to capture ST correlations, it can still significantly improve the MAE (6% in TaxiBJ, 7% in TaxiNYC) and MAPE (7.9% in TaxiBJ,

[2] https://github.com/apache/incubator-mxnet

Table 3: Performance results on TaxiBJ and TaxiNYC.

Model [# parameters]		TaxiBJ		TaxiNYC	
		MAE	MAPE	MAE	MAPE
HA		26.2	22.9%	32.3	43.5%
ARIMA		40	38.8%	N/A (Timeout)	N/A (Timeout)
GBRT		37.7	37.8%	25.7	44.6%
FNN	[7k]	24.5 ± 0.1	24.0% ± 0.2%	17.7 ± 0.2	43.6% ± 0.5%
FNN+	[16k, +128%]	19.4 ± 0.1 (-20.8%)	19.4% ± 1.3% (-19.2%)	14.9 ± 0.1 (-15.8%)	36.2% ± 0.2% (-17.0%)
GRU	[39k]	23.2 ± 0.1	23.0% ± 0.4%	15.2 ± 0.1	40.0% ± 0.2%
GRU+	[42k, +7.1%]	18.6 ± 0.1 (-19.8%)	17.8% ± 0.4% (-22.6%)	13.8 ± 0.0 (-9.2%)	34.7% ± 0.1% (-13.3%)
CNN	[300k]	23.0 ± 0.6	25.1% ± 0.3%	15.0 ± 0.4	43.0% ± 2,6%
CNN+	[304k, +1.3%]	19.0 ± 0.2 (-17.4%)	19.7% ± 0.4% (-21.5%)	13.1 ± 0.3 (-12.7%)	33.0% ± 0.8% (-30.2%)
ConvGRU	[338k]	17.4 ± 0.2	18.1% ± 0.2%	11.8 ± 0.1	31.4% ± 0.3%
ConvGRU+	[341k, +0.8%]	16.9 ± 0.1 (-2.8%)	17.1% ± 0.1% (-5.5%)	11.4 ± 0.1 (-3.4%)	30.5% ± 0.3% (-2.9%)
ST-ResNet	[1130k]	16.6 ± 0.6	17.7% ± 0.8%	11.4 ± 0.1	30.3% ± 0.1%
ST-ResNet+	[1133k, +0.2%]	15.6 ± 0.1 (-6.0%)	16.3% ± 0.1% (-7.9%)	10.6 ± 0.0 (-7.0%)	29.1% ± 0.1% (-4.0%)
DMVST-Net	[57k]	18.7 ± 0.1	20.6% ± 0.1%	13.7 ± 0.3	34.2% ± 0.4%
DMVST-Net+	[60k, +5%]	17.0 ± 0.0 (-9.1%)	17.8% ± 0.5% (-13.6%)	12.4 ± 0.2 (-9.5%)	32.8% ± 0.3% (-4.1%)
STDN	[198k]	23.4 ± 2.5	30.1% ± 5.8%	11.5 ± 0.1	31.5% ± 0.2%
STDN+	[203k, +2.5%]	18.7 ± 0.5 (-20.1%)	19.6% ± 0.4% (-34.9%)	11.7 ± 0.1 (+1.7%)	31.0% ± 0.3% (-1.5%)

4% in TaxiNYC) by only introducing three thousand additional parameters, and achieve the state-of-the-art result.

The reason of such significant improvement is that TaxiBJ and TaxiNYC have too many regions, *i.e.,* 1024 and 256 respectively, across the whole city, and they are very different from each other. Thus a model with sharing parameters across all regions cannot effectively learn such diverse flow trends. Instead, MF-STN learns a predictor with region-specific parameters by matrix factorization, enabling the model to tackle the region function issue while considering correlations among regions. As a result, the deep models enhanced by MF-STN can have much better performance than the basic versions.

Efficiency Comparison (Q3.)
We show the efficiency of MF-STN from two aspects:

- **Model complexity**. The number of parameters for the deep models are shown in Table 3 where the percentage values refer to the additional parameters compared with the basic model. Note that integrating with MF-STN introduces very few additional parameters, except for FNN (this model is too simple). All deep models can have very significant improvement with such a small number of additional parameters.
- **Training/inference speed**. As shown in Figure 6 and Figure 7, we compare the efficiency of the base deep models and their enhanced versions, by plotting the average speed (samples/seconds) in the training and inference processes. Note that except for the very simple model FNN, the bottleneck of most deep models are the ST feature learners. We find that MFDense layers only bring a small amount of additional computation consumption. Therefore, MF-STN does not visibly degrade the performance of base models, illustrating that it is a lightweight framework.

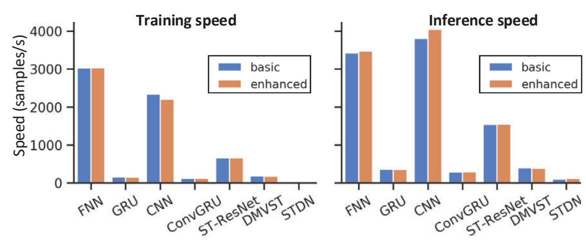


Figure 6: Processing time for TaxiBJ dataset.

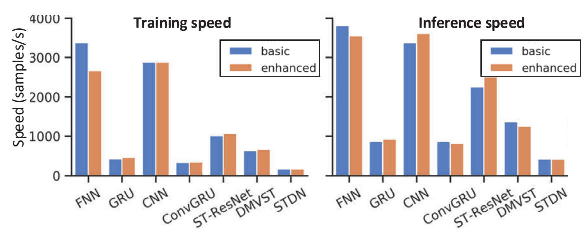


Figure 7: Processing time for TaxiNYC dataset.

5.3 Evaluation on Framework Settings (Q4.)

To show the robustness of MF-STN, we conduct experiments on how different choices of two important parameters, *i.e.,* the number of MFDense layer m in the region-specific predictor, and the region embedding dimension k, can affect the performance of MF-STN.

- Evaluation on m. We fix $k = 8$ by default and obtain the prediction accuracy of the deep models enhanced by MF-STN with $m = \{1, 2, 3\}$ MFDense layers. As shown in Figure 8, adding more MFDense layers for simple models, including FNN, GRU, and CNN, can improve the prediction. The reason is that these models are not well designed for ST data, so they are not strong enough to learn ST features. As simply adding layers can make models have more capacity for feature learning, the prediction accuracy can be improved. As for complex deep ST models, including ConvGRU, ST-ResNet, DMVST-Net and STDN, that are capable of learning abundant ST features, adding more MFDense layers on them has similar prediction accuracy compared with only a single MFDense layer, showing the robustness of MF-STN.

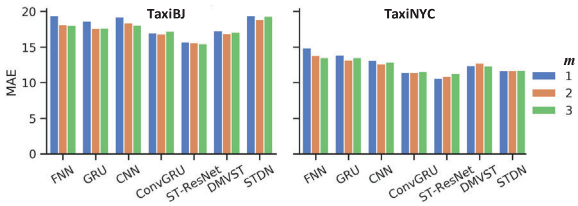


Figure 8: Evaluation on the number of MFDense layers m.

- Evaluation on k. We fix $m = 1$ and obtain the prediction accuracy of the deep models enhanced by MF-STN with embedding dimension $k = \{4, 8, 16\}$. Larger k can roughly correspond to the larger capacity of the embedding space, indicating more different the regions could be. As shown in Figure 9, the prediction accuracy is stable with different k settings for complex ST models, showing the robustness of MF-STN. In addition, it demonstrates that though there are many regions in the cities, the function of a region can be described by a low-dimensional vector. This fact supports the feasibility of our insights, *i.e.,* using matrix factorization to model the inherent correlations among regions.

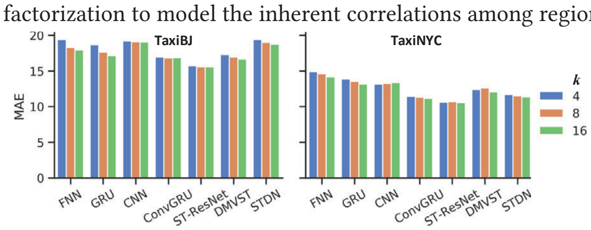


Figure 9: Evaluation on the region embedding dimension k.

5.4 Case Study on Region Embedding (Q5.)

To further explain why MF-STN works, we present a case study on the region embeddings learned by the enhanced ST-ResNet. To support the insights of MF-STN, two properties of region embedding should be demonstrated: 1) embeddings show the region functions; and 2) a good region embedding space should reveal the similarity of regions, with nearby regions owning similar flows.

First, as shown in Figure 10 (a), we plot the region embeddings on a two-dimensional plane. We select three representative regions, *i.e.,* Zhongguancun (a business zone), Yongtaiyuan (a residential zone), and Beijing Olympic Park (a park zone), as the symbol "×" marked in Figure 10 (a). We also mark 3 nearest neighbors of each selected region as symbol "+". Clearly, regions with different functions are far away from each other as shown in Figure 10 (a), indicating that the model learns very different region embeddings. Second, to further verify that the region embeddings can reveal the region similarity, we plot the flows of selected regions and their neighborhoods in Figure 10 (b). Notice that these three regions have analogous flow trends compared with their neighbors in the embedding space, illustrating that region embeddings can effectively represent the similarity among regions. In summary, this case can show meaningful embedding space learned by MF-STN, explaining why our proposed framework is effective in urban flow prediction.

6 RELATED WORK

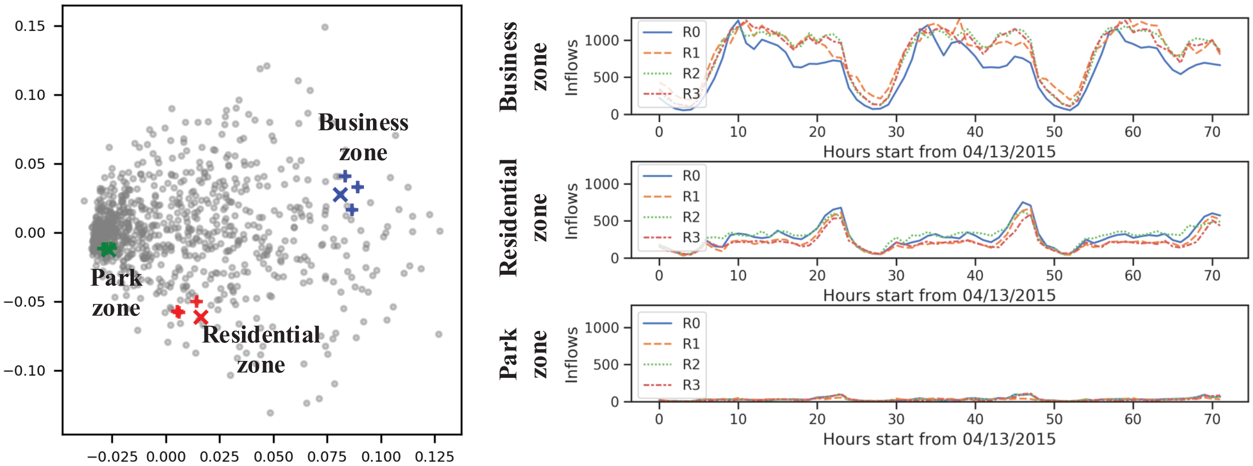
We study several categories of related works, positioning our work in the research community.

- **Urban Flow Prediction.** Urban flow prediction is an important topic in the field of urban computing. Initially, [5, 10, 11] proposed non-deep models for this task. However, these models highly depend on the hand-crafted features, that are hardly to comprehensively depict the complex ST correlations among data. Thereafter, the powerful representation learning of deep network boosted the research of flow prediction. In the beginning, [30] presented a deep neural network based flow prediction model. Next, [29, 31] improved the prediction by separately modeling flow trends, periods, and closeness, and using ResNet [4] to better capture ST correlations among regions. [24] developed a multi-view prediction model with a temporal view, a spatial view, and a semantic view of flows. [23] studied the dynamic temporal shifting problem of flows. [3] adopted multi-graph convolution to model spatial dependencies in advance. In addition, [32] employed a multi-task learning framework to simultaneously predict regions' flows and flow transitions between regions. Recently, [17] proposed to model diverse traffic flow from other auxiliary geographical information by deep meta learning method.
- **Deep Learning for Spatio-Temporal Prediction.** There are many deep learning models proposed to solve related prediction task on ST data. Specifically, CNN [8] is widely used as a basic structure in capturing spatial correlations of grid-based data [7, 18]. In addition, due to the success of RNN [2] in modeling sequence data, many studies employed RNN to capture the temporal correlations of ST data [15, 16, 20]. Recently, some research payed attention to modeling more complex ST correlations, such as employing graph convolution neural networks to extract spatial correlations of non-grid data [9, 21, 25], and attention mechanisms to capture the dynamic ST correlations [12].
- **Matrix Factorization on Neural Networks' Weights.** In the field of network compression, [13, 19, 26] adopted matrix factorization for low-rank approximation to remove redundant information. In the field of multi-task learning, [22] applied tensor factorization to realize automatic learning of end-to-end knowledge sharing in deep neural networks.

In summary, being different from all above works, this paper focuses on modeling the diverse flow trends, as well as inherent correlations among regions w.r.t. the regions' flows. Moreover, the proposed framework is the first to tackle such model-level discrepancy in spatio-temporal prediction by matrix factorization on neural network.

7 CONCLUSION

In this paper, we propose a novel deep learning framework, named MF-STN, for incorporating the latent region function into the state-of-the-art deep ST models to further improve the model ability on citywide flow prediction. Our MF-STN provides a portable ST feature learner, accommodating sub-networks of an existing deep model to capture ST correlations, and a region-specific predictor, leveraging the learned ST features to make region-specific predictions by using matrix factorization on neural networks. We evaluate MF-STN on two real-world datasets and the results show that these deep models integrating with our framework can achieve significantly better performance. In particular, experiments have shown that our framework advances baselines by average 13.7%/17.9% on the TaxiBJ dataset, and 8.0%/10.4% on the TaxiNYC dataset, in terms of MAE/MAPE metrics, respectively. In addition, empirical studies



(a) Region embedding space

(b) Comparison of a region's flow with its neighbors

Figure 10: Visualization of region embedding. (a) "×" denotes the selected regions, while "+" denotes the neighborhoods of the selected regions in the region embedding space. (b) "Rk"($k > 0$) stands for the kth-nearest neighbor of selected region R0 in the region embedding space.

and visualization have confirmed the advantages of MF-STN on the effectiveness, efficiency, and robustness. In the future, we will focus on generalizing our framework to more ST tasks.

ACKNOWLEDGMENTS

We thank Chentian Jin for his feedback on the draft of this paper, and Yifang Zhou for her help on the system deployment. This work was supported by the National Natural Science Foundation of China Grant (61672399, U1609217, 61773324, 61702327, 61772333, and 61632017).

REFERENCES

[1] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. 2015. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432* (2015).

[2] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[3] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. 2019. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *AAAI*.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

[5] Minh X Hoang, Yu Zheng, and Ambuj K Singh. 2016. FCCF: forecasting citywide crowd flows based on big data. In *SIGSPATIAL*. ACM.

[6] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[7] Benjamin Klein, Lior Wolf, and Yehuda Afek. 2015. A dynamic convolutional layer for short range weather prediction. In *CVPR*.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

[9] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR*.

[10] Yexin Li and Yu Zheng. 2019. Citywide Bike Usage Prediction in a Bike-Sharing System. *TKDE* (2019).

[11] Yexin Li, Yu Zheng, Huichu Zhang, and Lei Chen. 2015. Traffic prediction in a bike-sharing system. In *SIGSPATIAL*. ACM.

[12] Yuxuan Liang, Songyu Ke, Junbo Zhang, Xiuwen Yi, and Yu Zheng. 2018. GeoMAN: Multi-level Attention Networks for Geo-sensory Time Series Prediction.. In *IJCAI*.

[13] Shaohui Lin, Rongrong Ji, Xiaowei Guo, Xuelong Li, et al. 2016. Towards Convolutional Neural Networks Compression via Global Error Reconstruction. In *IJCAI*.

[14] Ye Liu, Yu Zheng, Yuxuan Liang, Shuming Liu, and David S Rosenblum. 2016. Urban water quality prediction based on multi-task multi-view learning. In *IJCAI*.

[15] 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.

[16] 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.

[17] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban Traffic Prediction from Spatio-Temporal Data Using Deep Meta Learning. In *SIGKDD*. ACM.

[18] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*.

[19] Matthew Sotoudeh and Sara S Baghsorkhi. 2018. DeepThin: A self-compressing library for deep neural networks. *arXiv preprint arXiv:1802.06944* (2018).

[20] Yongxue Tian and Li Pan. 2015. Predicting short-term traffic flow by long short-term memory recurrent neural network. In *2015 IEEE international conference on smart city/SocialCom/SustainCom (SmartCity)*. IEEE, 153–158.

[21] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. *IJCAI* (2019).

[22] Yongxin Yang and Timothy Hospedales. 2017. Deep multi-task representation learning: A tensor factorisation approach. *ICLR* (2017).

[23] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. 2019. Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction. *AAAI*.

[24] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, and Jieping Ye. 2018. Deep multi-view spatial-temporal network for taxi demand prediction. *AAAI*.

[25] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *IJCAI* (2018).

[26] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. In *CVPR*.

[27] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2011. Driving with knowledge from the physical world. In *SIGKDD*. ACM.

[28] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *SIGSPATIAL*. ACM.

[29] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In *AAAI*.

[30] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, and Xiuwen Yi. 2016. DNN-based prediction model for spatio-temporal data. In *SIGSPATIAL*. ACM.

[31] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, Xiuwen Yi, and Tianrui Li. 2018. Predicting citywide crowd flows using deep spatio-temporal residual networks. *AI* 259 (2018), 147–166.

[32] Junbo Zhang, Yu Zheng, Junkai Sun, and Dekang Qi. 2019. Flow Prediction in Spatio-Temporal Networks Based on Multitask Deep Learning. *TKDE* (2019).