# TTPNet: A Neural Network for Travel Time Prediction Based on Tensor Decomposition and Graph Embedding

Yibin Shen, Jiaxun Hua, and Cheqing Jin, *Member, IEEE*

**Abstract**—Travel time prediction of a given trajectory plays an indispensable role in intelligent transportation systems. Although many prior researches have struggled for accurate prediction results, most of them achieve inferior performance due to insufficient feature extraction of travel speed and road network structure from the trajectory data, which confirms the challenges involved in this topic. To overcome those issues, we propose a novel neural *Network* for *Travel Time Prediction* based on tensor decomposition and graph embedding, named *TTPNet*, which can extract travel speed and representation of road network structure effectively from historical trajectories, as well as predict the travel time with better accuracy. Specifically, *TTPNet* consists of three components: the first module (Travel Speed Features Layer) leverages non-negative tensor decomposition to restore travel speed distributions on different roads in the previous hour, and integrates a CNN-RNN model to extract both long-term and short-term travel speed features of the query trajectory; the second module (Road Network Structure Features Layer) utilizes graph embedding to generate the representation of local and global road network structure; the last module (Deep LSTM Prediction Layer) completes the final predicting task. Empirical results over two real-world large-scale datasets show that our proposed *TTPNet* model can achieve significantly better performance and remarkable robustness.

**Index Terms**—Travel time prediction, tensor decomposition, graph embedding, deep learning

## 1 Introduction

Thanks to the popularity of GPS-embedded devices, much more trajectory data has been generated everyday, by analyzing which municipal authorities may identify and optimize the traffic congestion in advance. As to the view of commuters, travel time prediction of a given trajectory based on historical trajectories enables them to plan their trips more reasonably. The ubiquitous applications of travel time prediction push it to be one of the most booming location-based services. However, predicting an accurate travel time is still very demanding as the travel time is affected by many dynamic factors, such as departure time, traffic conditions and driver behavior [1], [2].

Intuitively, travel speed is the most comprehensive reflection of these dynamic factors. Nevertheless, the transportation system is so complex and variable that it is challenging to predict the future moving pattern in an explicit form and infer the future travel speed of the roads [3]. [4] adopts a seasonal ARIMA to predict the future travel speed in the next hour, but it only predicts the average travel speed in a whole city, ignoring the spatial difference in the different parts of the city. [2] partitions a city into disjoint grids and designs a LSTM model to extract the travel speed features of different grids. However, due to the sparsity in trajectory data, a large number of grids lack speed values in a short time interval[1]. To solve the data sparsity, matrix/tensor decomposition is utilized in [5], [6], which estimates the individual vehicle speed or travel time on different roads. Other studies note that seasonal patterns always happen in time series, so they apply seasonal smoothing methods to capture this characteristic in the day/hour-level [7], [8]. These references provide powerful and robust solutions to estimate missing values.

In addition, travel time is also affected by some static factors, such as road network structure. Driving on one-way roads, two-way roads, intersections or T-junctions may result in different travel time. Hence, how to extract the representation of road network structure effectively is also an indispensable part of travel time prediction. [1] proposes a spatio-temporal component to learn the characteristics of going straight and turning based on raw GPS points. Similarly, a trajectory segmentation approach in [9] partitions a trajectory into several segments and obtains different representation of segments by an embedding method. However, depending solely on raw GPS points may lead to a wrong road network structure, such as turning points, as is shown in Fig. 1. Although there exists a classic solution (map-matching algorithm [10]) to map GPS points into the roads on the map, that solution is time-consuming and will also lose some details of GPS points, such as vehicle lane change.


- *Y. Shen is with the School of Data Science and Engineering, East China Normal University, Shanghai 200062, China. E-mail: ybshen@stu.ecnu.edu.cn.*
- *J. Hua is with the School of Software Engineering, East China Normal University, Shanghai 200062, China. E-mail: vichua@stu.ecnu.edu.cn.*
- *C. Jin is with the School of Data Science and Engineering, East China Normal University, Shanghai 200062, China. E-mail: cqjin@dase.ecnu.edu.cn.*


1. Sparsity is ubiquitous in the trajectory data [5]. There are three reasons for sparsity: (1) Trajectory data is a skewed distribution, and more trajectories will be generated in urban areas; (2) Only a sample of vehicles can be recorded in the datasets, such as some GPS equipped taxis; (3) The sample rate is insufficient, and not all roads traversed by vehicles can be sampled.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3038259, IEEE Transactions on Knowledge and Data Engineering



(a) Wrong turning points (b) Correct turning points

Figure 1. Example of wrong turning points based on raw GPS points.

Without the use of GPS points and map-matching algorithm, some studies use grids to obtain road network structure features. [2] directly maps GPS points to the grids and uses a word embedding method to represent each grid with a low-dimensional vector. [11] integrates the trajectory data with morphological layout images and uses a convolutional neural network (CNN) to learn the travel delay during the query trajectory from gridding images. Suitable grids generation can also retain the details of the trajectory data. Unfortunately, these methods don't involve the whole road network structure, so that only the local structure of road network can be preserved.

With the aim of solving aforementioned challenges, we propose a novel neural *Network* for *Travel Time Prediction* based on tensor decomposition and graph embedding, named *TTPNet*, which can extract travel speed and representation of road network structure effectively from historical trajectories, and predict the travel time of a given trajectory with better accuracy. The main contributions of this paper are summarized as follows:

- *Restoring the missing travel speed:* We partition a city into disjoint grids and exploit a fast non-negative tensor decomposition algorithm to restore the travel speed distribution on different grids in the previous hour, which contributes to the extraction of travel speed features.
- *Extracting the travel speed features:* A CNN-RNN model is designed to extract the long-term travel speed features of each grid in the past 7 days, and the short-term travel speed features of each grid and relevant grids in the previous hour. We use time-shifting KL-divergence to obtain the top-$k$ most relevant ones of each grid.
- *Extracting the road network structure features:* We use historical trajectories to construct a directed road network graph, and get low-dimensional representation vectors of road network graph by an autoencoder model (SDNE), which surely preserves the local and global structure of road network.
- *Prediction model:* A deep LSTM prediction model is designed to jointly model travel speed features and road network structure features, then we propose a loss function that works as a multi-task learning to train our model. We evaluate our model on two real-world large-scale datasets which consist of GPS logs generated by taxis in *Beijing* and *Shanghai* (with different sampling rates). The performance of our proposed model on both datasets achieves high accuracy and remarkable robustness.

The rest of this paper is organized as follows. Sect. 2 reviews the related work about tensor decomposition, graph embedding and travel time prediction. Sect. 3 presents several preliminary concepts and the framework of our proposed neural network *TTPNet*. Sect. 4 compares the performance of our proposed model with several state-of-the-art methods based on two real-world large-scale datasets. This paper is summarized in Sect. 5.

## 2 Related Work

### 2.1 Tensor Decomposition

Tensors are multidimensional arrays in mathematics. Specifically, vectors and matrices can be regarded as first-order and second-order tensors respectively, and tensors of order three or higher are called high-order tensors. The aim of tensor decomposition is to restore the missing values in the tensor, which has been widely used in signal processing, computer vision and urban computing [12], [13], [14], [15]. Empirically, the CANDECOMP/PARAFAC (CP) decomposition [16], [17] and Tucker decomposition [18] are the main techniques of tensor decomposition. The CP decomposition expresses a tensor as the sum of a finite number of rank-one tensors, and Tucker decomposition decomposes a tensor into a core tensor multiplied by a matrix along each mode. To obtain better missing values, integrating heterogeneous data sources is proposed in [19], which extends the standard tensor decomposition formulation. Other studies note that seasonal patterns always happen in time series, such as power consumption and traveling behavior, so they apply seasonal smoothing methods to capture this characteristic in the day/hour-level [7], [8]. These references provide powerful and robust solutions to estimate missing values.

### 2.2 Graph Embedding

In the past decade, deep learning has been successfully applied in computer vision and natural language processing, showing superior representation power. However, unlike images and text, graphs lie in irregular domain [20], so that applying deep learning to graphs, such as social networks and road networks, is still challenging. To mine the information in graphs, tremendous studies and literature have been made towards this field. They hope to embed graphs into a low-dimensional space with the goal of preserving the local and global structure of graphs. These methods can be divided into three categories: semi-supervised methods (GCNs) [21], [22], unsupervised methods (GAEs) [23], [24] and other methods [25], [26]. GCNs exploit spectral graph theory to implement the convolution operation on graphs, which enables GCNs to be trained with task-specific loss via back-propagation like standard CNNs. GAEs exploit autoencoders to realize unsupervised learning, which learn node representation for graphs without supervised information. As the road network we build doesn't include any labels, GAEs are more suitable for our work. Other methods employ recurrent neural networks and reinforcement learning to solve the problem of dynamic graphs and graph generation, which are beyond the scope of this paper.

### 2.3 Travel Time Prediction

Studies on the travel time prediction could be grouped into three categories: road-segment-based approaches [27], [28], [29], [30], sub-path-based approaches [4], [31], [32], [33], [34] and deep-learning approaches [1], [2], [9], [11], [35], [36].

#### 2.3.1 Road-Segment-Based Approaches

Early travel time prediction algorithms mainly depend on the loop detectors [27], [28]. The loop detectors record the travel time of each vehicle crossing two adjacent loop detectors. Since all roads aren't necessarily covered by loop detectors in reality, these studies mainly focus on individual road segments. However, it is not accurate because they neglect the relationship between road segments.

In recent years, more studies pay attention to use GPS trajectories [29], [30]. [29] considers the travel time of each road segment as a multivariate Gaussian distribution and predicts the travel time by maximum likelihood estimation. [30] considers the relationship between road segments and uses a spatio-temporal Hidden Markov model to obtain the correlations among adjacent roads. However, they also focus on the accuracy of individual road segments, and get the travel time of a given trajectory by summing time spent on each road segment, which may lead to accumulation errors.

#### 2.3.2 Sub-Path-Based Approaches

In order to eliminate the accumulation errors, sub-path-based approaches consider sub-paths instead of road segments. Some work [31], [32] mines frequent patterns from historical trajectories and predicts the travel time by using the average travel time of a corresponding pattern. [4] uses the travel time of neighboring trajectories, which have similar starting points and destinations as the query trajectory. [33] introduces a dynamic Bayesian network to model traffic congestion state of various road segments and searches for more optimal concatenation of road segments to predict the travel time. [34] splits the query trajectory into $k$ sub-paths, and searches for similar trajectories by considering the influence of drivers, weather and departure time. However, as trajectories are always sparse, these approaches need a balance between the coverage of queries and the accuracy of travel time, which are quite time-consuming.

#### 2.3.3 Deep-Learning Approaches

Recently, some studies try to employ deep learning techniques to predict travel time because of its strong performance in spatio-temporal data mining. [1] proposes a spatio-temporal component to learn the spatial and temporal dependencies from raw GPS points. Specifically, it transforms GPS points into a series of feature maps, and applies convolutional and recurrent neural networks on the feature maps. [2] partitions a city into disjoint grids instead of depending on raw GPS points. It adopts a LSTM model to extract the travel speed features of different grids and designs a new loss function for auxiliary supervision. [9] considers the influence of different road types on travel time. It proposes a trajectory segmentation approach to partition a trajectory into several segments and uses an embedding method to obtain different representation of segments. [11] proposes a CNN to integrate the trajectory data with the information of morphological layout images, which can provide rich and learnable information about built environments. However, most of these methods don't integrate travel speed features and road network structure features simultaneously, resulting in inferior performance. Other studies predict the travel time of origin-destination (without an actual route) [35] or each road (not a trajectory) [36], which are different from our work.

## 3 Methodology

In this section, we first depict several basic concepts for the travel time prediction, and then present the details of our neural network.

### 3.1 Preliminaries

**Definition** 1 (**Grids**). *Instead of presenting road network by using road segments, in our proposal, we partition a city into $N \times N$ grids for further travel speed features and road network structure features extracting. There are two motivation of using grids: (1) Map-matching algorithm is time-consuming; (2) The travel speed on an arterial road is also different at the same time since the road maybe too long.*

**Definition** 2 (**Trajectory**). *A trajectory $Tr$ is a sequence of sampled GPS points, e.g. $Tr : p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_{|Tr|}$, where $|Tr|$ represents the length of the trajectory. Each sample point $p_i = (x_i, y_i, g_i, v_i, t_i)$ contains: latitude $x_i$, longitude $y_i$, gridID $g_i$, speed $v_i$ and timestamp $t_i$. Moreover, we record the additional features for each trajectory, such as departure time (timeID), the day of the week (weekID), the driver (driverID), holidays and the distance of the trajectory.*

**Definition** 3 (**Problem**). *The travel time of a trajectory $Tr : p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_{|Tr|}$ is the value of $(p_{|Tr|}.t_{|Tr|} - p_1.t_1)$. Given a trajectory $Tr$, our goal is to predict the travel time between starting point and destination through $Tr$ with the corresponding features.*

### 3.2 Model Architecture

As is shown in Fig. 2, we introduce the framework of our proposed *TTPNet* model, which is comprised of three major components: Travel Speed Features Layer, Road Network Structure Features Layer and Deep LSTM Prediction Layer.

- **Travel Speed Features Layer**: We first construct a 3D-non-negative tensor $\mathcal{A}$ with the travel speed distribution of different grids in historical hours and the previous hour. After restoring the missing values in $\mathcal{A}$ by using a fast non-negative tensor decomposition algorithm, we obtain a new dense tensor $\mathcal{A}^*$. Then, we propose a CNN-RNN model to extract both long-term and short-term travel speed features of a given trajectory, based on $\mathcal{A}^*$. A fully connected layer at the end of the module combines the two travel speed features. Hence, the output is the whole travel speed features, which is denoted as $V_{speed}$.
- **Road Network Structure Features Layer**: We first use historical trajectories to construct a directed road network graph $G(\mathcal{N}, \mathcal{E})$. Then we introduce SDNE model, which is an autoencoder structure and capable of capturing the local and global structure of road

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3038259, IEEE Transactions on Knowledge and Data Engineering



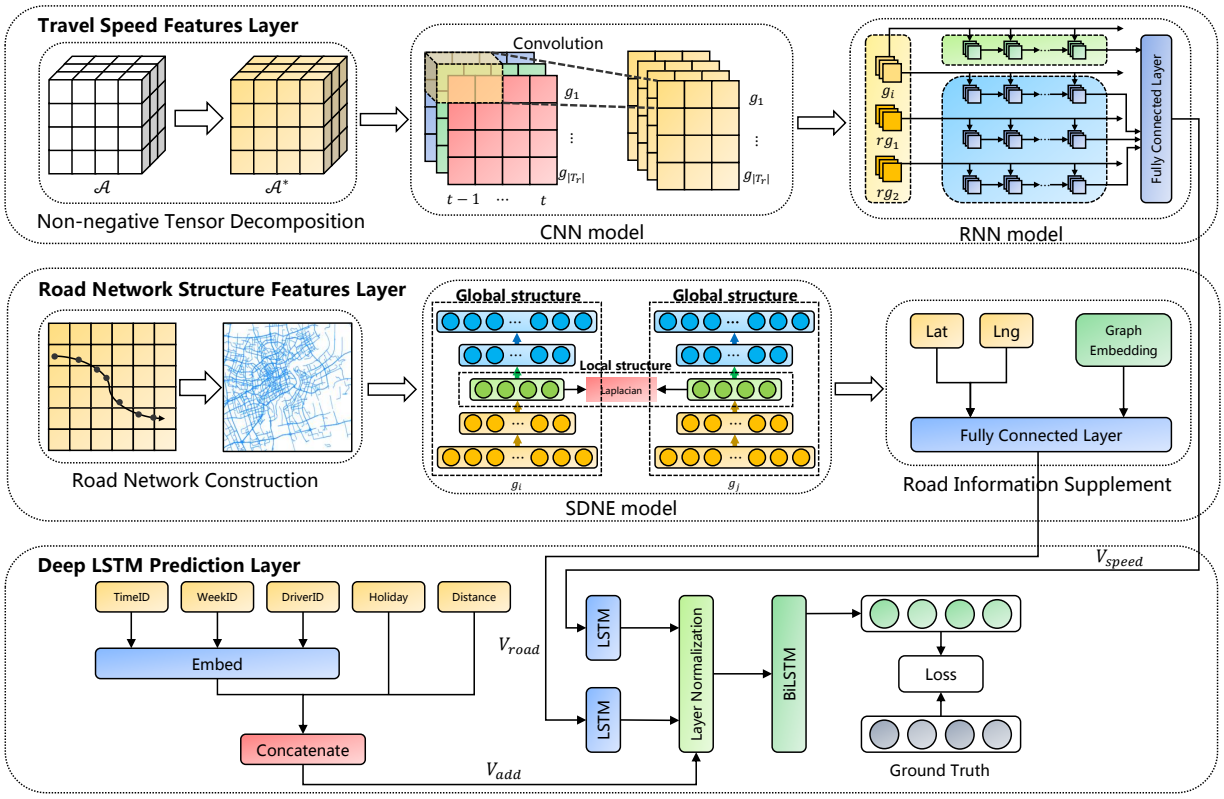



Figure 2. The Framework of our proposed *TTPNet* model.

network. After applying SDNE on $G(\mathcal{N}, \mathcal{E})$, we get the low-dimensional representation vectors (graph embedding vectors) of the road network structure. To supplement more road information, we incorporate the vectors with raw GPS points. Similarly, a fully connected layer completes the whole road network structure features extraction, and the output of the module is denoted as $V_{road}$.

- **Deep LSTM Prediction Layer**: We first encode the travel speed features $V_{speed}$ and road network structure features $V_{road}$ with a single-layer LSTM respectively. Then we exploit a BiLSTM to jointly learn these two important features with the additional features $V_{add}$. The motivation of using the deep LSTM learning architecture is that travel speed and road network structure share information, and are highly correlated with each other. It is difficult to model them by a single LSTM. In addition, we design a new loss function working as a multi-task learning, which can benefit the prediction on the travel time of the entire given trajectory.

### 3.3 Travel Speed Features Layers

#### 3.3.1 Non-negative Tensor Decomposition

As is discussed in Sect. 1, travel time is affected by many dynamic factors and travel speed is the most comprehensive reflection of these dynamic factors. However, many grids lack corresponding travel speed because only a few grids are traversed by drivers in a short time interval. It is not accurate enough to fill the missing travel speed of a grid with an average of historical travel speed in that grid since the traffic conditions are dynamic. Fortunately, we can construct a sparse tensor $\mathcal{A}$ with travel speed and restore the missing values in $\mathcal{A}$ by using tensor decomposition.

If a request for travel time prediction of a given trajectory is submitted at time $t$, we can construct a matrix $A_r \in \mathbb{R}^{N^2 \times M}$ to store travel speed of different grids based on historical trajectories falling within the time window of $[t-1 \text{ hour}, t)$. Specifically, we partition an hour into $M$ time slots, with each time slot corresponding to $60/M$ minutes. Every entry $A_r(i,j) = a$ denotes the $i$-th grid with travel speed $a$ in time slot $j$, but actually the matrix is sparse (the missing values are filled to zero). To supplement more information, another matrix $A_h \in \mathbb{R}^{N^2 \times M}$ is constructed based on historical trajectories over a long period of time (e.g. a week)[2]. Due to their similar structures, we can construct a mixed matrix $A_m$ to combine their respective information, as is shown in Eq. (1).

$$\begin{cases} A_m(i,j) = A_r(i,j), & A_r(i,j) \neq 0 \\ A_m(i,j) = A_h(i,j), & A_r(i,j) = 0 \end{cases} \tag{1}$$

After constructing aforementioned three matrices $A_h$, $A_m$ and $A_r$, we concatenate them together like sandwiches to construct a 3D-non-negative tensor $\mathcal{A} \in \mathbb{R}^{N^2 \times M \times 3}$, as is shown in Fig. 3.

The purpose of constructing the non-negative tensor $\mathcal{A}$ is that we'd like to know the travel speed in any grid within

2. The structure of $A_h$ is the same as that of $A_r$, while $A_h(i,j) = b$ denotes the $i$-th grid with an average of historical travel speed $b$ in time slot $j$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3038259, IEEE Transactions on Knowledge and Data Engineering



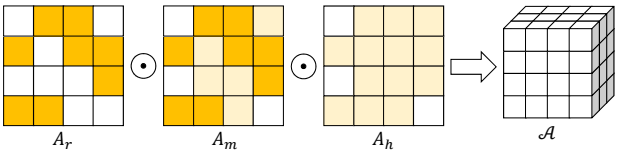



Figure 3. Constructing the non-negative tensor $\mathcal{A}$, where '$\odot$' denotes the concatenate operation.

$[t-1 \text{ hour}, t)$. One of common approaches solving this problem is to use non-negative tensor decomposition. To be more specific, we can establish the following optimization problem to approximate $\mathcal{A}$.

$$\min_{A,B,C} \| \mathcal{A} - \sum_{j=1}^{r} a_j \circ b_j \circ c_j \|_F^2 \quad \text{s.t.} \quad A, B, C \geq 0 \tag{2}$$

where $a_j \in \mathbb{R}^{N^2}$, $b_j \in \mathbb{R}^M$, $c_j \in \mathbb{R}^3$, $A = [a_1, a_2, \cdots, a_r]$, $B = [b_1, b_2, \cdots, b_r]$, $C = [c_1, c_2, \cdots, c_r]$, and '$\circ$' denotes the outer product of vectors. $r$ represents the rank of tensor $\mathcal{A}$ and $\| \cdot \|_F$ denotes the Frobenius norm.

We realize the above non-negative tensor decomposition via a 'tensorly' library [37], which is very efficient and can be accelerated on the GPU platform. For example, it takes less than 1 second to decompose the tensor $\mathcal{A}$ ($N = 128, M = 4$). Afterwards, we can restore the missing values in $\mathcal{A}$ by calculating a new dense tensor $\mathcal{A}^*$.

### 3.3.2 CNN model

Intuitively, traffic conditions of a region have continuity in terms of temporal dimension. For example, a grid where travel speed gradually decreases from 4:30 to 5:30 p.m. may have traffic congestion in the foreseeable future, which will affect the travel time. The fact means that the short-term travel speed distribution in the past period of time is valuable and useful, and motivates us to construct and decompose a non-negative tensor with travel speed falling within the time window of $[t-1 \text{ hour}, t)$.

For a given trajectory $Tr : p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_{|Tr|}$ submitted at time $t$, we design a triple for the grid $p_i.g_i$ to record short-term travel speed features of $g_i$ in time slot $j$. In detail, the triple is in the form of

$$x_i^j = \left(v_i^j, dis_i, dis_i / v_i^j\right) \tag{3}$$

where $v_i^j$ denotes the travel speed of $g_i$ in time slot $j$ based on the dense tensor $\mathcal{A}^*$[3]; $dis_i$ is the distance of the query trajectory through the grid; $dis_i / v_i^j$ roughly estimates the travel time spent in grid $g_i$.

To capture the changing trend of travel speed in grid $g_i$, a convolutional neural network (CNN) is used on the sequence $X_i = \{x_i^1, \cdots x_i^M\}$, as is shown in Fig. 4. CNN is widely used in the computer vision, such as image classification and object detection, which offers an efficient architecture to extract statistical patterns in datasets. In our model, we apply a one-dimensional convolutional filter [38] on the sequence $X_i$, with parameter matrix $W_{conv} \in \mathbb{R}^{k \times M}$ ($k$ denotes the kernel size of the convolution). The sequence $X_i \in \mathbb{R}^{3 \times M}$ can be seen as a 3-channel input, similar to RGB in the image. After the one-dimensional convolution operation, the $j$-th dimension of its output is denoted as

$$c_i^j = \tanh\left(W_{conv} * x_i^{j:j+k-1} + b\right) \tag{4}$$

where '$*$' represents the convolution operation, $b$ is the bias term and '$\tanh$' is the activation function.

By concatenating the outputs of $c$ filters, we get the output sequence $C_i = \{c_i^1, \cdots, c_i^{M-k+1}\} \in \mathbb{R}^{c \times (M-k+1)}$, which records the temporal correlations of travel speed in grid $g_i$.

3. Although the tensor $\mathcal{A}^*$ is much denser than $\mathcal{A}$, it also contains some zero values. To express the missing travel speed of grid $g_i$, we use the average travel speed of the city in time slot $j$.

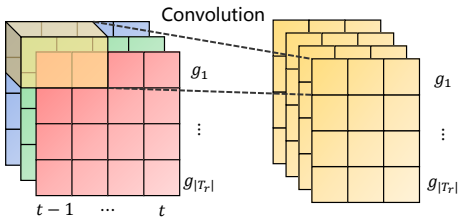



Figure 4. The CNN model in the travel speed features layer.

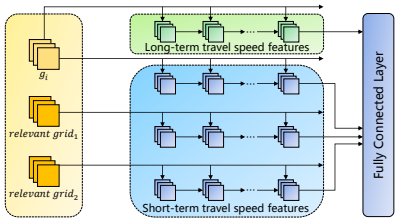



Figure 5. The RNN model in the travel speed features layer.

### 3.3.3 RNN model

To learn the temporal correlations of travel speed in grid $g_i$, we will introduce a recurrent neural network (RNN), as is shown in Fig. 5. RNN is able to memorize the previous historical information in the processed sequence, which is widely used in sequence-to-sequence problems, such as machine translation. The output of our RNN is the hidden state at the last step in the sequence. Specifically, the output can be expressed as

$$h_i^t = \tanh\left(W_c \cdot c_i^t + W_h \cdot h_i^{t-1} + b\right) \tag{5}$$

where $W_c, W_h$ are learnable parameter matrices and $b$ is the bias term. $h_i^t \in \mathbb{R}^{16}$ denotes the hidden state at $t$ step, and the output of our RNN model is $h_i^{M-k+1}$, which reflects the short-term travel speed features of grid $g_i$. For simplicity, we denote $h_i^{M-k+1}$ as $h_i^{short}$.

On the other hand, we note that traffic conditions of a region also have continuity in terms of spatial dimension. For example, the travel speed in grid $g_i$ is affected by its upstream grids. Inspired by that fact, the short-term travel speed features of $g_i$ can be extended. We first define the concept of *upstream grids* and *relevant grids* as follows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3038259, IEEE Transactions on Knowledge and Data Engineering



**Definition** 4 (**Upstream grids**). *Given a grid $g_i$, the upstream grids of $g_i$ are directly reachable from $g_i$ at one step. For example, supposing that there are three trajectories passing thought several grids, $Tr_1 : g_1 \rightarrow g_2 \rightarrow g_3 \rightarrow g_4$, $Tr_2 : g_5 \rightarrow g_3 \rightarrow g_4 \rightarrow g_6$, $Tr_3 : g_7 \rightarrow g_3 \rightarrow g_5 \rightarrow g_8$, the upstream grids of $g_3$ are $\{g_2, g_5, g_7\}$.*

**Definition** 5 (**Relevant grids**). *Given a grid $g_i$, the relevant grids of $g_i$ should satisfy the following two rules: (1) The grids are the upstream grids of $g_i$. (2) The travel speed distributions of grids are similar to that of $g_i$, which is measured by time-shifting KL-divergence.*

Kullback-Leibler divergence (KL-divergence) is utilized for measuring similarity between two distributions, which can calculate the similarity between two grids in the same time slots according to their travel speed distribution. Nevertheless, a grid is often affected by its upstream grids after a period of time. For example, $g_2$ is the upstream grid of $g_3$. If there is a traffic congestion in $g_2$ at 5:30 p.m., the traffic conditions of $g_3$ may be terrible at 5:45 p.m., which is called *time-shifting*. Consequently, we calculate the similarity between $p$ and $q$ by the time-shifting KL-divergence in Eq. (6), where $p$ denotes the travel speed distribution of the target grid, $q$ denotes one of upstream grids' travel speed distribution and $\Delta t$ represents the number of time-shifting.

$$D_{\Delta t}(p||q) = \sum_{t=\Delta t}^{24M} p_t \ln(p_t / q_{t-\Delta t+1}) \tag{6}$$

Eventually, we obtain the top-$k$ relevant grids, which have the smallest time-shifting KL-divergence with the grid $g_i$. We can also extract the short-term travel speed features from these relevant grids by utilizing the CNN-RNN model as we state above. Then, we obtain a sequence $h_i^{rele} = \{h_i^{rele_j}\}_{j=1}^{k}$, which extends the short-term travel speed features of grid $g_i$.

In addition to the short-term travel speed features, the long-term travel speed features also play an important role in travel time prediction. They can reveal the changes of city's road network structure, such as new roads and tunnels. The above extraction process of the short-term travel speed features can be easily adjusted to the long-term travel speed features. We only need to change the time window from $[t-1 \text{ hour}, t)$ to $[t-7 \text{ day}, t)$. Specifically, we design a new triple like Eq. (3) for the grid $g_i$. The element $v_i^j$ denotes the average travel speed in $g_i$ at the same time but in the previous $j$-th day. Similarly, we obtain the long-term travel speed features $h_i^{long}$.

At the end of the travel speed features layer, we concatenate the short-term travel speed features $\{h_i^{short}, h_i^{rele}\}$ and the long-term travel speed features $h_i^{long}$ by a fully connected layer, resulting the whole travel speed features $h_i^{speed}$ of grid $g_i$, as is shown in Eq. (7).

$$h_i^{speed} = \tanh\left(W_{speed} \cdot \left[h_i^{short} \odot h_i^{rele} \odot h_i^{long}\right] + b\right) \tag{7}$$

where $W_{speed}$ is a learnable parameter matrix, $b$ is the bias term and '$\odot$' denotes the concatenate operation. $h_i^{speed} \in \mathbb{R}^{32}$ reflects the travel speed features of grid $g_i$.

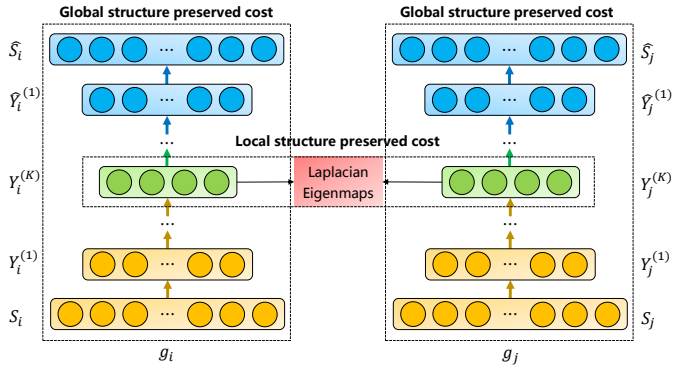



Figure 6. The SDNE model in the road network structure features layer.

After concatenating each sample point of the query trajectory $Tr$, we obtain the sequence $V_{speed} = \{h_i^{speed}\}_{i=1}^{|Tr|}$, which reflects the whole travel speed features of the query trajectory $Tr$.

### 3.4 Road Network Structure Features Layer

As is discussed in Sect. 1, travel time is also affected by some static factors, such as road network structure. Driving on the one-way roads, two-way roads, intersections or T-junctions may lead to different travel time. To extract the road network structure features, we first need to construct a road network graph, which can be defined by the concept of *upstream grids* as we state above.

**Definition** 6 (**Road Network**). *The road network structure could be modeled as a directed graph $G(\mathcal{N}, \mathcal{E})$ according to the upstream grids, where the node set $\mathcal{N}$ represents the grids and the edge set $\mathcal{E}$ represents the connected relationship between two grids. We define the adjacent matrix $S$ of the graph $G(\mathcal{N}, \mathcal{E})$, where $s_{i,j}$ denotes whether the grid $g_i$ and the grid $g_j$ are connected, i.e.*

$$s_{i,j} = \begin{cases} 1, \ g_i \text{ is the upstream grid of } g_j \\ 0, \ \text{otherwise} \end{cases} \tag{8}$$

Fig. 6 introduces SDNE model [24], which is an autoencoder structure for graph embedding. The aim of SDNE is to learn low-dimensional representation vectors of grids in the road network with the goal of preserving the local and global structure of road network in the learned embedding space. We first introduce the concept of first-order proximity and second-order proximity, which corresponds to the local and global structure of road network respectively.

**Definition** 7 (**First-order Proximity**). *The first-order proximity between two grids represents the similarity of the pair, which can be obtained directly from the adjacent matrix $S$. For example, there exists first-order proximity between the grid $g_i$ and $g_j$ if $s_{i,j} = 1$. Otherwise, the first-order proximity between $g_i$ and $g_j$ is 0.*

**Definition** 8 (**Second-order Proximity**). *The Second-order proximity between two grids represents the similarity of the pair's neighborhood structure. For example, we denote $S_i = \{s_{i,1}, \cdots, s_{i,N^2}\}$ as the neighborhood structure of grid $g_i$, and the second-order proximity between $g_i$ and $g_j$ can be obtained by the similarity of $S_i$ and $S_j$.*

Intuitively, the first-order proximity assumes that two grids are similar if the pair is connected in the road network, and the second-order proximity assumes that two grids are similar if the pair's neighborhood structures are similar, even if they are not connected in the road network. The second-order proximity enriches the relationship of grids in the road network and enables SDNE to capture the global structure of road network.

SDNE is an autoencoder structure, which consists of the encoder and decoder. The encoder is responsible for mapping the input data into low-dimensional vectors, while the decoder is responsible for mapping the low-dimensional vectors to the original representation space. Given the neighborhood structure $S_i$ of grid $g_i$, the process of encoder can be represented as

$$\begin{aligned} Y_i^{(1)} &= \sigma\left(W_S^{(1)} \cdot S_i + b^{(1)}\right) \\ Y_i^{(k)} &= \sigma\left(W_S^{(k)} \cdot Y_i^{(k-1)} + b^{(k)}\right),\ k = 2, \cdots, K \end{aligned} \tag{9}$$

where $\{W_S^{(k)}\}_{k=1}^K$ are learnable parameter matrices, $\{b^{(k)}\}_{k=1}^K$ are bias terms and '$\sigma$' is the 'Sigmoid' activation function.

According to the concept of the first-order proximity, we should preserve the similarity of low-dimensional representations between grid $g_i$ and $g_j$ if they are connected in the road network. Therefore, the loss function of first-order proximity can be defined as

$$\mathcal{L}_{1st} = \sum_{i,j=1}^{N^2} s_{i,j} \|Y_i^{(K)} - Y_j^{(K)}\|_2^2 \tag{10}$$

where $\|\cdot\|_2$ denotes the $\mathcal{L}2$-norm.

The loss function of the first-order proximity can also be calculated as $\mathcal{L}_{1st} = tr(Y^T L Y)$, where $Y = \{Y_i^{(K)}\}_{i=1}^{N^2}$ and $L = D - S$. $D$ is the diagonal matrix of road network and $D_{i,i} = \sum_j s_{i,j}$. As $L$ is a Laplacian matrix, the loss function is also called Laplacian Eigenmaps.

The decoder can be seen as the inverse process of the encoder, so we can obtain the output $\hat{S}_i$ of grid $g_i$. The goal of autoencoder structure is to minimize the reconstruction loss of the input data and the output. Since our input is the neighborhood structure of each grid, the reconstruction process will make the grids with similar neighborhood structure trend to similar low-dimensional representation. Therefore, the loss function of second-order proximity can be defined as

$$\begin{aligned} \mathcal{L}_{2nd} &= \sum_{i=1}^{N^2} \|(\hat{S}_i - S_i)\ [*]\ B_i\|_2^2 \\ &= \|(\hat{S} - S)\ [*]\ B\|_F^2 \end{aligned} \tag{11}$$

where '$[*]$' represents the element-wise multiplication. Note that road network is always sparse, there are a number of zero elements in the adjacent matrix $S$. In order to make the model incline to reconstruct the non-zero elements, the vector $B_i = \{b_{i,j}\}_{j=1}^{N^2}$ is cited. Specifically, $b_{i,j} = 10$ if $s_{i,j} = 1$, else $b_{i,j} = 1$.

By concatenating the loss function of first-order proximity and second-order proximity, we obtain the loss function of SDNE, which is defined as

$$\begin{aligned} \mathcal{L}_{\text{SDNE}} =& \alpha\mathcal{L}_{1st} + \mathcal{L}_{2nd} + \\ & \frac{1}{2}\sum_{k=1}^{K}(\|W_S^{(k)}\|_F^2 + \|\hat{W}_S^{(k)}\|_F^2) \end{aligned} \tag{12}$$

where $\alpha$ is a hyperparameter and $\{\hat{W}_S^{(k)}\}_{k=1}^K$ are parameter matrices in the decoder process. The last term in Eq. (12) is a regularization term to prevent over-fitting.

After minimizing the loss function, the low-dimensional representation vectors of each grid are obtained, which preserves the local and global structure of road network. To supplement more road information, we incorporate the vectors with the latitudes and longitudes of raw GPS points via a fully connected layer. Specifically, the road network structure features of grid $g_i$ can be constructed as

$$h_i^{road} = \tanh\left(W_{road} \cdot \left[p_i.x_i \odot p_i.y_i \odot Y_i^{(K)}\right] + b\right) \tag{13}$$

where $\{p_i.x_i, p_i.y_i\}$ denote the latitude and longitude of the sample point $p_i$, $W_{road}$ is a learnable parameter matrix and $b$ is the bias term. $Y_i^{(K)} \in \mathbb{R}^{32}$ is obtained by SDNE model and $h_i^{road} \in \mathbb{R}^{32}$ reflects the road network structure features of grid $g_i$. Similarly, we obtain the whole road network structure features $V_{road} = \{h_i^{road}\}_{i=1}^{|Tr|}$ after concatenating each sample point of the query trajectory $Tr$.

### 3.5 Deep LSTM Prediction Layer

After obtaining the travel speed features $V_{speed}$ and road network structure features $V_{road}$, we first use a single-layer LSTM [39] to extract the spatio-temporal information of the query trajectory $Tr$, as is shown in Eq. (14). LSTM is also a recurrent neural network, which solves the problem of gradient vanishing/exploding on the sequence by integrating three gates and memory cells. Unlike the RNN model in the module of the travel speed features layer, the outputs of LSTM are the hidden states at all steps in the sequence.

$$\begin{aligned} H_i^{speed} &= \text{LSTM}(H_{i-1}^{speed}, h_i^{speed}) \\ H_i^{road} &= \text{LSTM}(H_{i-1}^{road}, h_i^{road}) \end{aligned} \tag{14}$$

Before we jointly learn the travel speed features and road network structure features, we should consider some other factors affecting the travel time, such as timeID, weekID and driverID. We also calculate whether the day is a holiday and the distance of the query trajectory. These factors are concatenated to form a vector $V_{add}$, which reflects the additional features of travel time prediction.

We concatenate the vector $V_{add}$ with each element of the sequence $H_{speed}$ and $H_{road}$, and feed into the BiLSTM [40]. Compared with LSTM, the BiLSTM can capture both the forward and backward information, which is similar to the context in the text. The outputs of BiLSTM model are the forward and backward hidden states at all steps, which can be expressed as

$$\begin{aligned} \overrightarrow{H_i} &= \text{LSTM}(\overrightarrow{H_{i-1}}, [H_i^{speed} \odot H_i^{road} \odot V_{add}]) \\ \overleftarrow{H_i} &= \text{LSTM}(\overleftarrow{H_{i+1}}, [H_i^{speed} \odot H_i^{road} \odot V_{add}]) \\ H_i &= [\overrightarrow{H_i} \odot \overleftarrow{H_i}] \end{aligned} \tag{15}$$

where '$\odot$' represents the concatenate operation.

To accelerate the convergence rate of the model, we add a layer normalization [41] before feeding the sequence into the BiLSTM. Layer normalization calculates the mean and variance of all channels of input data (the sequence can be seen as multi-channel input), and then standardizes them. Compared with batch normalization [42], layer normalization is more suitable for processing the sequential data. Additionally, we use the dropout technology [43] in the BiLSTM to avoid over-fitting.

After obtaining the sequence $H = \{H_1, \cdots, H_{|Tr|}\}$, the problem of travel time prediction is transformed to the regression problem between the sequence $H$ and the travel time $t = \big(p_{|Tr|}.t_{|Tr|} - p_1.t_1\big)$. We can simply use the last hidden state $H_{|Tr|}$ in the sequence or apply an attention mechanism [44] on the sequence with the travel time $t$ to establish the loss function. However, these methods compress the sequence $H$ into a fixed-length vector, which may cause a great loss of useful information, such as the travel time from the starting point to each sample point in the query trajectory. Hence, we construct a vector $T = \big(t_1, \cdots, t_{|Tr|}\big)$, where $t_i = (p_i.t_i - p_1.t_1)$ denotes the travel time from the starting point to sample point $p_i$.

It should be noted that $t_j = t_{j-1} + (t_j - t_{j-1})$, where $(t_j - t_{j-1})$ corresponds to $H_j$ in the sequence $H$. In other words, $\sum_{i=1}^{j} H_i$ corresponds the travel time from the starting point to $p_j$. We exploit a non-linear mapping $f(\cdot)$ to predict the travel time from the starting point to each sample point, i.e.

$$\hat{T} = \left[0, f(H_1 + H_2), \cdots, f(\sum_{i=1}^{|Tr|} H_i)\right] \tag{16}$$

where $\hat{T}$ represents the travel time prediction from the starting point to each sample point in the trajectory, and $f(\cdot)$ denotes a non-linear mapping, which can be a specific non-linear function or a neural network.

Finally, we establish the loss function between the travel time prediction vector $\hat{T}$ and the travel time vector $T$, which can be defined as

$$\mathcal{L} = \frac{\|(\hat{T} - T)\ [/]\ T\|_2}{(|Tr| - 1)^{\frac{1}{2}}} \tag{17}$$

Here, the numerator represents the relative percentage error between the prediction vector $\hat{T}$ and the ground truth vector $T$, and the denominator is designed as $(|Tr| - 1)^{\frac{1}{2}}$ because the first element of $\hat{T}$ and $T$ are zero.

There are two advantages of designing such a loss function. First, it invisibly enlarges the amount and range of data, forcing the model to optimize the travel time of short trajectories and long trajectories simultaneously, which works as a multi-task learning. Second, it helps the model to find key grids in the whole trajectory, such as the grids contain turning points, traffic lights or congested roads, which works as an attention mechanism. Hence, minimizing the loss function can benefit the prediction on the travel time of the entire given trajectory.

# 4 Experiments

In this section, We compare our model with several baseline methods, including traditional models and deep learning models over two real-world large-scale datasets. Our code and the sample data can be downloaded at https://github.com/YibinShen/TTPNet.

## 4.1 Experiments Setup

### 4.1.1 Datasets

We evaluate our model on two real-world large-scale datasets, namely *Beijing* and *Shanghai*. The Beijing dataset contains 3,384,847 trajectories of 10,039 taxis from Oct. $1^{st}$ to Oct. $31^{st}$ in 2013. The Shanghai dataset contains 9,727,798 trajectories of 13,622 taxis from Apr. $1^{st}$ to Apr. $30^{th}$ in 2015. We only keep the trajectories of taxis when they are carrying passengers. More detailed statistical information of two datasets is shown in Table 1.

Table 1
The statistical information of the datasets.

| Dataset | Beijing | Shanghai |
|---|---|---|
| trajectory number | 3,384,847 | 9,727,798 |
| taxi number | 10,039 | 13,622 |
| sampling rate | 60s | 10s |
| area | 29.9km × 29.5km | 34.0km × 31.7km |
| grid number | 128 × 128 | 256 × 256 |
| density ($A_r$) | 34.03% | 26.77% |
| density ($A_h$) | 52.76% | 42.52% |
| graph size ($\|\mathcal{N}\|$) | 16384 | 65536 |
| graph size ($\|\mathcal{E}\|$) | 300505 | 578372 |
| travel time mean | 1137.14s | 642.94s |
| travel time std | 917.30s | 461.21s |
| distance mean | 8.60km | 4.40km |
| distance std | 7.88km | 2.41km |

### 4.1.2 Hyperparameters

The hyperparameters in our experiments are adopted as follows:

- In the travel speed features layer, we fix the number of time slots as $M = 4$ and evaluate the effect of tensor decomposition under different values of tensor rank $r$. For CNN model, we use different parameters to extract short-term and long-term travel speed features. We use the kernel size $k = 2$ in short-term travel speed features and the kernel size $k = 3$ in long-term travel speed features. The number of filters in CNN model is set as $c = 4$. For RNN model, we fix the number of relevant grids as $k = 2$.
- In the road network structure features layer, the number of layers in the encoder and decoder is set as 2, and the hyperparameter is set as $\alpha = 0.1$.[4]
- In the Deep LSTM prediction layer, We embed the timeID as $\mathbb{R}^8$, weekID as $\mathbb{R}^3$, driverID as $\mathbb{R}^8$. The hidden units of LSTM/BiLSTM are set as $32/64$, and the value of dropout in the BiLSTM is set as $0.25$. Finally, the non-linear mapping $f(\cdot)$ in Eq. (16) is a multi-layer perception, where activate function is 'ReLU(x) $= \max(0, x)$'.

4. The choice of these hyperparameters is determined by the accuracy of the reconstruction task in the SDNE model.

The datasets are divided into three parts: the data in the last 7 days is used as the test set, and the rest is divided into the training set and validation set in the ratio of 3:1. We use Adam [45] to train the model with an initial learning rate at 0.001, and the learning rate will be halved every 10 epochs. In addition to layer normalization and dropout, we add weight decay with a value of 1e-5 to avoid over-fitting. The batch size is set as 256/512 in *Beijing*/*Shanghai* respectively. We train the model for 50 epochs and select the model that performs best on the validation set. Our model is implemented in PyTorch 1.4.0, and runs on a server with one NVIDIA Tesla P40 GPU and 16 cores CPU (Sliver 4110).

#### 4.1.3 Metrics

Three popular metrics: *Mean Absolute Error*(MAE), *Mean Absolute Percentage Error*(MAPE) and *Satisfaction Rate*(SR) [11] are employed to evaluate our model. Their definitions are as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{18}$$

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right| \times 100\% \tag{19}$$

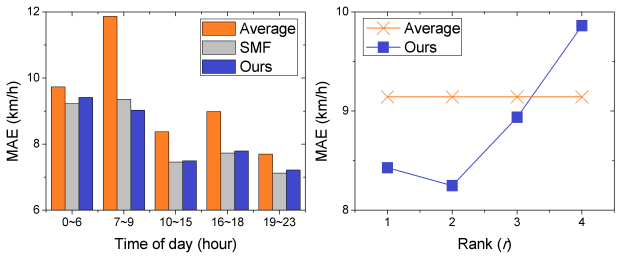
$$\text{SR} = \frac{1}{n}\sum_{i=1}^{n}\left(\left|\frac{\hat{y}_i - y_i}{y_i}\right| \leq 10\%\right) \times 100\% \tag{20}$$

where $\{y_1, \cdots, y_n\}$ denote the ground truth, $\{\hat{y}_1, \cdots, \hat{y}_n\}$ denote the predicted value and $n$ denotes the number of trajectories in the test set.

### 4.2 Experiment Results

#### 4.2.1 Non-negative Tensor Decomposition

To test the performance of non-negative tensor decomposition, we randomly remove $10\%$ of non-zero travel speed on Oct. $8^{th}$ in Beijing dataset and infer these values via different methods. Fig. 7(a) shows the errors of using the historical average travel speed (Average[5]), the seasonal matrix factorization (SMF[6] [8]) and our method.




(a) Time of day (b) Rank

Figure 7. Performance of non-negative tensor decomposition.

Compared with using an average of historical values, the non-negative tensor decomposition and seasonal matrix factorization stand out as better ways to restore the missing values, especially in the daytime. This is because the period from Oct. $1^{st}$ to Oct. $7^{th}$ is a holiday in China, and trajectories of vehicles in that period have a significant difference from weekdays. The long holiday may also account for SMF's inferior performance during the morning peak, though SMF extracts seasonal patterns of traveling behavior. We further compare the effectiveness of non-negative tensor decomposition under different ranks, as is shown in Fig. 7(b). The best effectiveness is reached when the rank is set as $r = 2$ because the tensor with $r = 1$ is too simple, and there is a phenomenon of over-fitting when the rank increases. Note that we should decompose a tensor every time slot, which is a challenge in the efficiency of tensor decomposition. Fortunately, our method runs on a GPU platform, and it takes less than 1 second to decompose a tensor, which is feasible and practical in real applications.

#### 4.2.2 Baseline Methods

We select several state-of-the-art work as baseline methods for comparison with our model, including:

- **TEMP:** *TEMP* [4] retrieves all the neighboring historical trajectories with similar starting points and destinations for the query trajectory, and predicts the travel time by scaling the average travel speed of these neighboring trajectories. *TEMP* works as a KNN query and the number of neighboring trajectories is fixed to 10 in our experiment.
- **XGBoost:** *XGBoost* [46] is a distinguished decision tree ensemble method, which has been widely applied in industry. In our experiment, the input of *XGBoost* is the same as that of our *TTPNet*, including travel speed features and road network structure features. Note that *XGBoost* can only deal with structured data like tables, so we uniformly sample/replace each trajectory to a fixed length[7].
- **DeepTTE:** *DeepTTE* [1] transforms raw GPS points into a series of feature maps, and applies convolutional and recurrent neural networks on the feature maps, which captures the spatio-temporal dependencies from raw GPS points.
- **DeepTravel:** *DeepTravel* [2] partitions a city into disjoint grids instead of depending on raw GPS points. It proposes a LSTM model to extract travel speed features of different grids and represents each grid by an embedding method. It also designs a new loss function for auxiliary supervision.
- **STDR:** *STDR* [9] considers the influence of different road types on travel time and proposes a trajectory segmentation approach based on raw GPS points to partition a trajectory into several segments. The representation of these segments is obtained by an embedding method.
- **TTPNet_S:** *TTPNet_S* is a simplified model of *TTPNet*. It uses the historical average travel speed instead of non-negative tensor decomposition and removes the graph embedding vectors obtained from SDNE.

For each model, the experiment is repeated for 5 times. We calculate the mean and the standard deviation of different runs, as is shown in Table 2.

5. The historical average travel speed is obtained from the matrix $A_h$.

6. We construct a series of sparse matrices in the hour-level with weekly periodicity.

7. The fixed length is equal to the average length of trajectories on the datasets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3038259, IEEE Transactions on Knowledge and Data Engineering

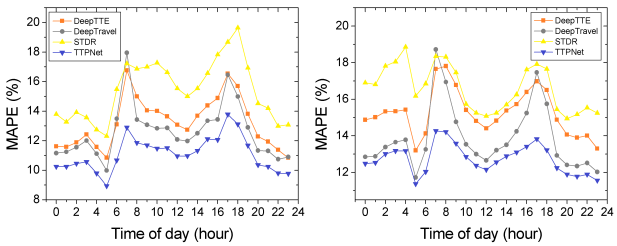


Table 2
Performance comparison of travel time prediction.

| Dataset | Beijing | | | Shanghai | | |
|---|---|---|---|---|---|---|
| Metrics | MAE(sec) | MAPE(%) | SR(%) | MAE(sec) | MAPE(%) | SR(%) |
| TEMP | 247.20 | 23.92 | 28.43 | 142.25 | 23.05 | 30.15 |
| XGBoost | 169.70±0.33 | 15.86±0.02 | 40.54±0.06 | 126.28±0.79 | 18.74±0.12 | 34.24±0.21 |
| DeepTTE | 141.33±1.43 | 13.67±0.16 | 48.94±0.35 | 104.72±1.29 | 15.31±0.10 | 43.67±0.31 |
| DeepTravel | 136.17±1.41 | 12.92±0.03 | 50.42±0.13 | 96.84±1.52 | 13.90±0.08 | 46.46±0.50 |
| STDR | 160.30±0.96 | 16.08±0.20 | 42.97±0.33 | 103.56±0.52 | 16.33±0.04 | 40.77±0.24 |
| TTPNet_S | 127.67±0.63 | 12.34±0.04 | 52.35±0.14 | 93.17±0.73 | 13.67±0.05 | 47.89±0.20 |
| TTPNet | **118.59±0.34** | **11.42±0.03** | **55.96±0.14** | **86.70±0.43** | **12.74±0.04** | **50.75±0.18** |

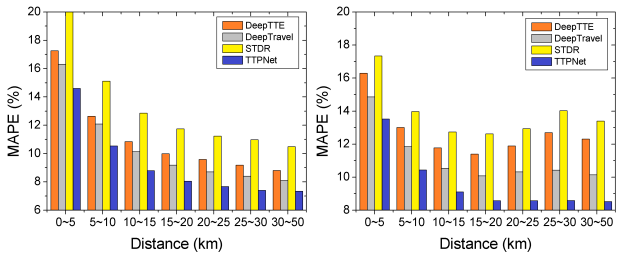
Table 3
Performance comparison during the morning and evening peak.

| Dataset | Beijing | | Shanghai | |
|---|---|---|---|---|
| Metrics | MAPE(%) | SR(%) | MAPE(%) | SR(%) |
| TEMP | 25.93 | 26.32 | 24.62 | 27.71 |
| XGBoost | 17.29 | 36.84 | 20.99 | 30.49 |
| DeepTTE | 15.39 | 43.56 | 16.98 | 38.89 |
| DeepTravel | 14.64 | 44.35 | 16.36 | 39.41 |
| STDR | 17.92 | 38.70 | 17.86 | 37.19 |
| TTPNet | **12.52** | **51.59** | **13.73** | **47.39** |




(a) Beijing (b) Shanghai

Figure 8. Error rates for trajectories with different departure time.




(a) Beijing (b) Shanghai

Figure 9. Error rates for trajectories with different distances.

#### 4.2.3 Performance Comparisons

As we can see, *TEMP* only considers the information of starting points and destinations, resulting in the worst performance on the two datasets. *Xgboost* performs better than *TEMP* because the input features of *Xgboost* are consistent with our model, which proves the importance of travel speed and road network structure features. However, *Xgboost* cannot handle sequence data with variable length, so resampling trajectories to fixed-length sequences will lose spatio-temporal relationship. *DeepTTE* and *STDR* use neural networks to extract the spatio-temporal dependencies between GPS points, which indicates the power of deep learning. However, raw GPS points cannot match the map in reality, which may lead to a wrong road network structure. *DeepTravel* and *TTPNet_S* consider travel speed and road network structure features simultaneously so they outperform aforementioned models. As to our model, *TTPNet* exploits non-negative tensor decomposition to accurately restore the missing values. The CNN-RNN and SDNE model accurately capture travel speed features and road network structure features respectively. Moreover, *TTPNet* is the only model with more than 50% satisfaction rate on both datasets.

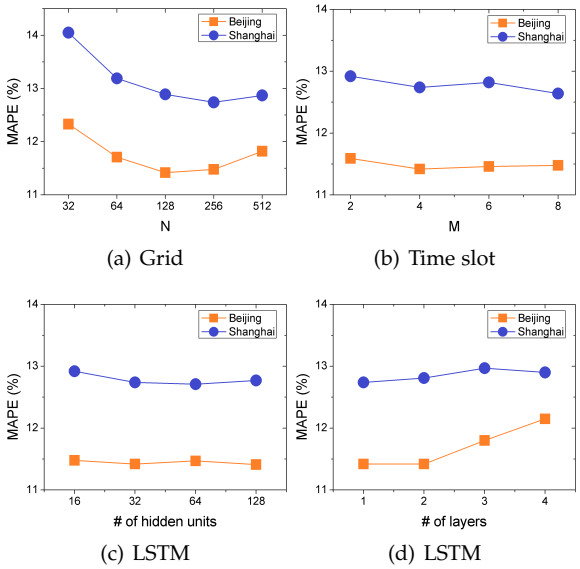
Additionally, we examine the standard deviation of each model. *TEMP* is a kNN query method, whose experiment results are consistent in different runs. Note that our *TTPNet* model has a tiny standard deviation on the three metrics because we use some regularization methods in the model, which brings remarkable robustness to the model.

In Fig. 8, we compare the model performance for trajectories with different departure time. The error rates of *DeepTTE*, *DeepTravel* and *STDR* increase rapidly in two time intervals (around 7-9am and 4-6pm), which correspond to the morning and evening peak in a day. As most people go to work and go home in these two time intervals, traffic conditions become complex and congested, which makes it more difficult to predict the future moving pattern. Fortunately, our *TTPNet* model doesn't fall into this dilemma. The error rates of *TTPNet* increase slightly in the two time intervals because we accurately extract travel speed and road network structure features. Table 3 illustrates the error rates and satisfaction rates of different models during the two time intervals. *TTPNet* still achieves the best performance, especially on Shanghai dataset where the satisfaction rate of *TTPNet* reaches 47.39%, while other models are lower than 40%. Overall, *TTPNet* provides accurate travel time at different departure time.

We compare the model performance for trajectories with different distances in Fig. 9. The error rates of all models decrease when the distance of the query trajectory increases on Beijing dataset, while our *TTPNet* model also outperforms other models. Moreover, when the distance of the query trajectory is larger than 20km on Shanghai dataset, the error rates of *DeepTTE* and *STDR* begin to increase. The reason is that there are only 0.08% trajectories larger than 20km on Shanghai dataset (whereas 7.29% on Beijing dataset), which leads to over-fitting. Fortunately, our *TTPNet* model can handle these long trajectories perfectly because we not only redesign the loss function to expand the amount and range of data but also use regularization methods in the model, which keeps *TTPNet* away from the over-fitting traps.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2020.3038259, IEEE Transactions on Knowledge and Data Engineering






(a) Grid (b) Time slot

(c) LSTM (d) LSTM

Figure 10. Parameter sensitivity of some hyperparameters.

#### 4.2.4 Parameter Sensitivity

We investigate the parameter sensitivity for some hyperparameters in Fig. 10, which can be summarized as follows:

**It is important to choose a suitable grid number.** We can see that the performance of both datasets presents a U-shaped curve. This is because the smaller $N$ simplifies the city structure and affects the extraction accuracy of subsequent road network structure features. The larger $N$ increases the sparsity of trajectory data, which makes it difficult for tensor decomposition to restore the travel speed in each grid. Note that *TTPNet* performs best on Beijing dataset when $N$ is set as 128, while that performs the best on Shanghai dataset when $N = 256$. The reason is that roads in *Shanghai* are more rugged and winding than those in *Beijing*, which requires a higher grid number to capture road network structure features. Overall, the choice of grid number depends on the road conditions of the city.

**The number of time slots is insensitive.** Our model is not sensitive to the number of time slots. Although the increase of $M$ will also aggravate the sparsity of trajectory data, we use tensor decomposition instead of historical average travel speed to accurately restore the travel speed in each grid. Since the efficiency of tensor decomposition will decrease with an increase of $M$, we recommend $M = 4$ as the number of time slots.

**It is unnecessary to pursue a wider and deeper neural network structure.** Our model is insensitive to the number of hidden units (Fig10(c)), but the performance of the model deteriorates as the number of layers increases (Fig10(d)). This is because the travel speed features are highly correlated with the road network structure features, and deeper layers will dilute the correlation. Therefore, there is no need to deliberately pursue a wider and deeper neural network.

#### 4.2.5 Ablation Study

Last but not least, we conduct an ablation study to better understand the effect of each module in our model. We remove travel speed features $V_{speed}$, road network structure features $V_{road}$, additional features $V_{add}$ and modify the loss function to the attention mechanism respectively, and the results are illustrated in Table 4. The performance of *TTPNet* decreases tremendously when $V_{speed}$ or $V_{road}$ is lost, which indicates again that the two features have a significant impact on the travel time prediction. As $V_{add}$ contains some factors affect the travel time, it plays a slight role in the model. When we modify the loss function to an attention mechanism, the performance of *TTPNet* decreases. Although the attention mechanism extracts the key information from the whole sequence, compressing the sequence into a fixed-length vector leads to great loss in information, which fails to handle the travel time prediction of short and long trajectories simultaneously.

Table 4
Effect of different modules in *TTPNet*.

| Dataset | Beijing | | Shanghai | |
|---|---|---|---|---|
| Metrics | MAPE(%) | SR(%) | MAPE(%) | SR(%) |
| TTPNet | **11.42** | **55.96** | **12.74** | **50.75** |
| -$V_{speed}$ | 12.66 | 51.74 | 14.33 | 46.51 |
| -$V_{road}$ | 13.34 | 48.85 | 15.34 | 43.16 |
| -$V_{add}$ | 12.17 | 53.49 | 13.00 | 50.00 |
| Attention | 12.32 | 52.59 | 12.99 | 50.17 |

## 5 Conclusion and Future Work

In this paper, we propose a novel neural network, named *TTPNet*, to predict the travel time of a given trajectory. Specifically, *TTPNet* consists of three major components: travel speed features layer extracts travel speed features effectively from historical trajectories, road network structure layer ensures the preservation of the local and global structure of road network, and deep LSTM prediction layer is capable of predicting travel time with better accuracy. Extensive experiments over two real-world datasets prove that *TTPNet* offers more accurate travel time for trajectories with variant distances and different departure time. In the future, we will focus on the study of road network embedding and expect a better representation of road network structure with labels information (e.g. the number of lanes or the type of roads). The better representation of road network structure will further improve the accuracy of travel time and can be applied to other location-based service topics.

## Acknowledgments

This work is supported by ECNU Academic Innovation Promotion Program for Excellent Doctoral Students (YBNLTS2019-022).

## References

[1] D. Wang, J. Zhang, W. Cao, J. Li, and Y. Zheng, "When will you arrive? estimating travel time based on deep neural networks," in *AAAI*, 2018, pp. 2500–2507.
[2] H. Zhang, H. Wu, W. Sun, and B. Zheng, "Deeptravel: a neural network based travel time estimation model with auxiliary supervision," in *IJCAI*, 2018, pp. 3655–3661.
[3] Z. Wang, K. Fu, and J. Ye, "Learning to estimate the travel time," in *KDD*, 2018, pp. 858–866.
[4] H. Wang, Y. Kuo, D. Kifer, and Z. Li, "A simple baseline for travel time estimation using large-scale trip data," in *SIGSPATIAL/GIS*, 2016, pp. 61:1–61:4.

[5] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *KDD*, 2014, pp. 25–34.

[6] X. Zhou, Q. Luo, D. Zhang, and L. M. Ni, "Detecting taxi speeding from sparse and low-sampled trajectory data," in *APWeb/WAIM (2)*, 2018, pp. 214–222.

[7] P. Chen, S. Liu, C. Shi, B. Hooi, B. Wang, and X. Cheng, "Neucast: Seasonal neural forecast of power grid time series," in *IJCAI*, 2018, pp. 3315–3321.

[8] B. Hooi, K. Shin, S. Liu, and C. Faloutsos, "SMF: drift-aware matrix factorization with seasonal patterns," in *SDM*, 2019, pp. 621–629.

[9] J. Xu, Y. Zhang, L. Chao, and C. Xing, "STDR: A deep learning method for travel time estimation," in *DASFAA (2)*, 2019, pp. 156–172.

[10] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *GIS*, 2009, pp. 336–343.

[11] W. Lan, Y. Xu, and B. Zhao, "Travel time estimation without road networks: An urban morphological layout representation approach," in *IJCAI*, 2019, pp. 1772–1778.

[12] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[13] E. Acar, T. G. Kolda, D. M. Dunlavy, and M. Mørup, "Scalable tensor factorizations for incomplete data," *arXiv preprint math/1005.2197*, 2010.

[14] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 208–220, 2013.

[15] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: Concepts, methodologies, and applications," *ACM TIST*, vol. 5, no. 3, pp. 38:1–38:55, 2014.

[16] C. J. Douglas and C. Jih-Jie, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[17] H. R. A., "Foundations of the parafac procedure: Models and conditions for an explanatory multi-modal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.

[18] T. L. R., "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[19] M. R. de Araujo, P. M. P. Ribeiro, and C. Faloutsos, "Tensorcast: Forecasting with context using coupled tensors (best paper award)," in *ICDM*, 2017, pp. 71–80.

[20] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *CoRR*, vol. abs/1812.04202, 2018.

[21] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *ICLR*, 2014.

[22] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016, pp. 3837–3845.

[23] F. Tian, B. Gao, Q. Cui, E. Chen, and T. Liu, "Learning deep representations for graph clustering," in *AAAI*, 2014, pp. 1293–1299.

[24] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.

[25] Y. Ma, Z. Guo, Z. Ren, Y. E. Zhao, J. Tang, and D. Yin, "Dynamic graph neural networks," *CoRR*, vol. abs/1810.10627, 2018.

[26] J. You, B. Liu, Z. Ying, V. S. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *NeurIPS*, 2018, pp. 6412–6422.

[27] J. Rice and E. van Zwet, "A simple and effective method for predicting travel times on freeways," *IEEE Trans. Intelligent Transportation Systems*, vol. 5, no. 3, pp. 200–207, 2004.

[28] C. Wu, J. Ho, and D. Lee, "Travel-time prediction with support vector regression," *IEEE Trans. Intelligent Transportation Systems*, vol. 5, no. 4, pp. 276–281, 2004.

[29] J. Erik and K. H. N., "Travel time estimation for urban road networks using low frequency probe vehicle data," *Transportation Research Part B Methodological*, vol. 53, no. 4, pp. 64–81, 2013.

[30] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio-temporally correlated time series using markov models," *PVLDB*, vol. 6, no. 9, pp. 769–780, 2013.

[31] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *SIGMOD Conference*, 2000, pp. 1–12.

[32] W. Luo, H. Tan, L. Chen, and L. M. Ni, "Finding time period-based most frequent path in big trajectory data," in *SIGMOD Conference*, 2013, pp. 713–724.

[33] A. Achar, V. Sarangan, R. Regikumar, and A. Sivasubramaniam, "Predicting vehicular travel times by modeling heterogeneous influences between arterial roads," in *AAAI*, 2018, pp. 2063–2070.

[34] R. Waury, C. S. Jensen, and K. Torp, "Adaptive travel-time estimation: A case for custom predicate selection," in *MDM*, 2018, pp. 96–105.

[35] Y. Li, K. Fu, Z. Wang, C. Shahabi, J. Ye, and Y. Liu, "Multi-task representation learning for travel time estimation," in *KDD*, 2018, pp. 1695–1704.

[36] W. Wei, X. Jia, Y. Liu, and X. Yu, "Travel time forecasting with combination of spatial-temporal and time shifting correlation in CNN-LSTM neural network," in *APWeb/WAIM (1)*, 2018, pp. 297–311.

[37] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor learning in python," *J. Mach. Learn. Res.*, vol. 20, pp. 26:1–26:6, 2019.

[38] Y. Kim, "Convolutional neural networks for sentence classification," in *EMNLP*, 2014, pp. 1746–1751.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[40] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[41] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[42] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.

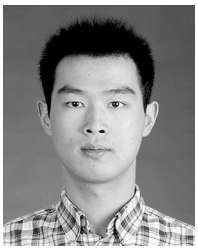
[43] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *NIPS*, 2016, pp. 1019–1027.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR (Poster)*, 2015.

[46] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*, 2016, pp. 785–794.


**Yibin Shen** received the BE degree in Applied Mathematics from East China University of Science and Technology in 2017 and is working toward the PhD degree in the School of Data Science and Engineering, East China Normal University. His research interests include spatio-temporal data mining and natural language processing.



**Jiaxun Hua** received the BE degree in Computer Science and Technology from East China Normal University in 2015 and the Master Degree in the School of Software Engineering from East China Normal University in 2020. His research interests include location-based services and encrypted databases.

**Cheqing Jin** recevied the bachelor's and master's degrees from Zhejiang University, and the PhD degree from Fudan University. He is a professor with East China Normal University. His research interests include streaming data management, location-based services, uncertain data management, data quality, and database benchmarking. He is a member of the IEEE.