When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks

Dong Wang[1], **Junbo Zhang**[2], **Wei Cao**[3], **Jian Li**[3], **Yu Zheng**[2,4,5]
[1]Department of Electrical and Computer Engineering, Duke University, NC, USA
[2]Urban Computing Group, Microsoft Research, Beijing, China
[3]Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
[4]School of Computer Science and Technology, Xidian University, China
[5]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
dong.wang363@duke.edu, junbo.zhang@microsoft.com, {cao-w13@mails, lijian83@mail}.tsinghua.edu.cn, msyuzheng@outlook.com

Abstract

Estimating the travel time of any path (denoted by a sequence of connected road segments) in a city is of great importance to traffic monitoring, route planning, ridesharing, taxi/Uber dispatching, etc. However, it is a very challenging problem, affected by diverse complex factors, including spatial correlations, temporal dependencies, external conditions (e.g. weather, traffic lights). Prior work usually focuses on estimating the travel times of individual road segments or sub-paths and then summing up these times, which leads to an inaccurate estimation because such approaches do not consider road intersections/traffic lights, and local errors may accumulate. To address these issues, we propose an end-to-end *Deep* learning framework for *Travel Time Estimation* (called *DeepTTE*) that estimates the travel time of the whole path directly. More specifically, we present a geo-convolution operation by integrating the geographic information into the classical convolution, capable of capturing spatial correlations. By stacking recurrent unit on the geo-convoluton layer, our DeepTTE can capture the temporal dependencies as well. A multi-task learning component is given on the top of DeepTTE, that learns to estimate the travel time of both the entire path and each local path simultaneously during the training phase. Extensive experiments on two trajectory datasets show our DeepTTE significantly outperforms the state-of-the-art methods.

Introduction

Estimating the travel time for a given path, which is denoted by a sequence of connected sub-paths, is a fundamental problem in route planning, navigation, and traffic dispatching. When users are searching for candidate routes, accurate travel time estimations help them better planning routes and avoiding congested roads, which in turn helps to alleviate traffic congestion. Almost all electronic maps and online car-hailing services provide the *travel time estimation* (TTE) in their apps, such as Google Map, Uber and Didi. The quality of the estimation is critical to the user experience of these apps.

Although the problem has been widely studied in the past, providing an accurate travel time is still very challenging, affected by the following aspects:
1) *Individual vs. Collective*: There mainly exist two approaches to estimate the travel time of a path: a) *Individual TTE* that firstly splits a path into several road segments (or local paths), and then estimates the travel time for each local path, finally sums over them to get the total travel time. b) *Collective TTE* that directly estimates the travel time of the entire path. Although individual TTE methods (Yang, Guo, and Jensen 2013; Wang et al. 2016c; Pan, Demiryurek, and Shahabi 2012; Wang, Zheng, and Xue 2014) can estimate accurate travel time for each road segment, it cannot model complex traffic conditions within the entire path, including road intersections, traffic lights, and direction turns. Besides, local errors may accumulate if there are many road segments in the given path. Collective TTE methods (e.g. (Jenelius and Koutsopoulos 2013)) are able to capture the aforementioned traffic conditions implicitly. However, as the length of a path increases, the number of trajectories traveling on the path decreases, which reduces the confidence of the travel time (derived from few drivers), pointing out that longer path is harder to estimate. Moreover, in many cases, there is no trajectory passing the entire path.
2) *Diverse complex factors*: the traffic is affected by spatial correlations, temporal dependencies, and external factors. Spatial correlations are various, even complex, as shown in Fig. 1. With three consecutive GPS points, it depicts different driving situations, showing the driver may go straight, turn right, turn around, drive into the main road or ramp road. Explicitly extracting these features are time-consuming, even infeasible because the driving situations are more complex in the real-world. We need to consider them implicitly in the method. In addition, these spatial correlations are time-varying. Taking Fig. 1 (b) as an example, in the evening rush hour, there are many vehicles the main road, drivers have to drive into the main road from the ramp road one after another slowly, but driving out of the main road may be very quick. But in the non-rush hour, driving into the main road is fast. Furthermore, traffic is affected by many external factors, like weather, driver habit, day of the week.

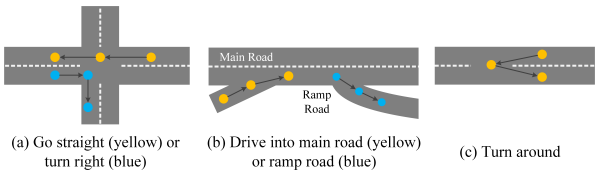

Figure 1: Various driving situations

To address the above challenges, in this paper, we propose an end-to-end framework for *Travel Time Estimation*, called

DeepTTE. The primary contributions of this paper can be summarized as follows:

- We propose a spatio-temporal component to learn the spatial and temporal dependencies from the raw GPS sequence. In detail, the spatio-temporal component consists of two parts: a) *a geo-based convolutional layer* that transforms the raw GPS sequence to a series of feature maps, capable of capturing the local spatial correlations (like various driving situations in Fig. 1) from consecutive GPS points implicitly; b) *recurrent neural nets* (LSTMs) that learn the temporal dependencies of the obtained feature maps and embeddings from external factors.
- We propose a multi-task component that learns to estimate the travel time for each local path and the entire path simultaneously by a multi-task loss function, capable of balancing the tradeoff between the individual and collective estimations. For estimating the entire path accurately, we design a multi-factor attention mechanism to learn the weights for different local paths based on their hidden representations and the external factors.
- We present an attribute component that integrates external factors, including the weather condition, day of the week, distance of the path, and the driver habit. The learned latent representations are fed into several parts of the model to enhance the importance of these external factors.
- We conduct extensive experiments on two real-world large scale data sets which consists of GPS points generated by taxis in Chengdu and Beijing. The percentage error on these two datasets are 11.89% and 10.92% respectively, which significantly outperforms the existing methods.

Preliminary

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

Definition 1 (Historical Trajectory) *We define a* historical trajectory T *as a sequence of consecutive historical GPS points, i.e.,* $T = \{p_1, \ldots, p_{|T|}\}$[1]. *Each GPS point* p_i *contains: the latitude* ($p_i.lat$), *longitude* ($p_i.lng$) *and the timestamp* ($p_i.ts$). *Furthermore, for each trajectory we record its external factors such as the starting time (*timeID*), the day of the week (*weekID*), the weather condition (*weatherID*) and corresponding driver (*driverID*).*

We then illustrate the our objective.

Definition 2 (Objective) *During the training phase, we learn how to estimate the travel time of the given path and the corresponding external factors, based on the spatio-temporal patterns extracted from the historical trajectories as we defined in Definition. 1. During the test phase, given the path* P, *our goal is to estimate the travel time from the starting to the destination through* P, *with the corresponding external factors. We assume that the travel path* P *is specified by the user or generated by the route planing apps.*

[1]The GPS devices usually generate one record for every fixed length time gap. This can cause our model to learn a trival pattern (e.g., simply counts the number of GPS records). To avoid such case, we resample each historical trajectory such that the distance gap between two consecutive points are around 200 to 400 meters.

During the test phase, to make the testing data consistent with the training data, we convert a path P *to a sequence of location points with equal distance gaps. Each location is represented as a pair of longitude and latitude.*

Remark: In our experiment, to generate the test data, we remove the timestamps in the historical trajectories and resample each trajectory into a GPS location sequence with equal distance gaps. In this paper, we do not consider how to optimize the path P.

Model Architecture

In this section, we describe the architecture of our proposed DeepTTE, as shown in Fig. 2. DeepTTE is comprised of three components: attribute component, spatio-temporal learning component, and multi-task learning component. The attribute component is used to processes the external factors (e.g. weather) and the basic information of the given path (e.g. start time). Its output is fed to the other components as a part of their inputs. The spatio-temporal learning component is the main building component that learns the spatial correlations and temporal dependencies from the raw GPS location sequences. Finally, the multi-task learning component estimates the travel time of the given path based on the previous two components, capable of balancing the tradeoff between individual estimation and collective estimation.

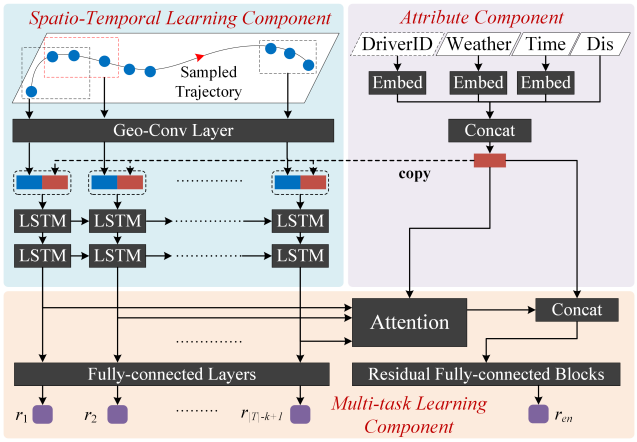

Figure 2: DeepTTE Architecture. Dis: distance; concat: concatenate.

Attribute Component

As we mentioned, the travel time of a path is affected by many complex factors, such as the start time, the day of week, the weather condition and also the driving habits. We design a simple yet effective component to incorporate such factors into our model, in which we call it the attribute component.

As an example in Fig. 2, we incorporate the attributes of the weather condition (rainy/sunny/windy etc.) , the driver ID, the time information (day of the week and timeslot of travel start[2]). We use weatherID, driverID, weekID and timeID to

[2]We divide one day into 1440 timeslots. Each timeslot corresponds to one minute.

denote these attributes respectively. Note that these factors are categorical values which can not feed to the neural network directly. In our model, we use the embedding method (Gal and Ghahramani 2016) to transform each categorical attribute into a low-dimensional real vector. Specifically, the embedding method maps each categorical value $v \in [V]$ to a real space $\mathbf{R}^{E\times 1}$ (we refer to such space as the embedding space) by multiplying a parameter matrix $W \in \mathbf{R}^{V\times E}$. Here V represents the vocabulary size of the original categorical value and E represents the dimension of embedding space. Usually, we have that $E \ll V$. Comparing with the one-hot encoding (Gal and Ghahramani 2016), the embedding method mainly has two advantages. First, since the vocabulary size of the categorical values can be very large (e.g., there are 20442 drivers in our dataset), the embedding method effectively reduces the input dimension and thus it is more computationally efficient. Furthermore, it has been shown that the categorical values with similar semantic meaning are usually embedded to the close locations (Gal and Ghahramani 2016). Thus, the embedding method helps find and share similar patterns among different trajectories.

Besides the embedded attributes, we further incorporate another important attribute, the *travel distance*. Formally, we use $\Delta d_{p_a\to p_b}$ to denote the total distance of traveling from GPS point p_a to p_b along the path, i.e., $\Delta d_{p_a\to p_b} = \sum_{i=a}^{b-1} \mathrm{Dis}(p_i, p_{i+1})$ where Dis is the geographic distance between two GPS points. Then, we concatenate the obtained embedded vectors together with the travel distance $\Delta d_{p_1\to p_{|T|}}$. The concatenation is used as the output of the attribute component. We denote such output vector as $attr$.

Spatio-Temporal Component

The spatio-temporal component consists of two parts. The first part is a geo-convolutional neural network which transforms the raw GPS sequence to a series of feature maps. Such component captures the local spatial correlation between consecutive GPS points. The second part is the recurrent neural network which learns the temporal correlations of the obtained feature maps.

Geo-Conv Layer We first present the Geo-Conv layer. Recall that a historical trajectory T is a sequence of GPS location points $\{p_1, \ldots, p_{|T|}\}$ where each p_i contains the corresponding longitude/latitude(See Definition 1). As we mentioned in the introduction part, capturing the spatial dependencies in the GPS sequence is critical to travel time estimation. A standard technique to capture the spatial dependencies is the *convolutional neural network* (CNN), which is widely used in the image classification, object tracking and video processing etc (Simonyan and Zisserman 2014; Krizhevsky, Sutskever, and Hinton 2012). A typical convolutional layer consists of several convolutional filters. For a multi-channel input image, a filter learns the spatial dependencies in the input by applying the convolution operation on each of the two dimensional local patches. We refer to such convolutional layer as 2D-CNN. (Zhang et al. 2016) used the 2D-CNN to predict the citywide crowd flow. In their work, they first partitioned the city into a $I \times J$ grid and then mapped each GPS coordinate into a grid cell. However, in our case, directly mapping the GPS coordinates into grid cells is not accurate enough to represent the original spatial information in the data. For example, we can not distinguish the turnings if the related locations are mapped into the same cell. Thus, our task requires a much finer granularity. Motived by this, we proposed a *Geo-Conv layer* which is able to capture the spatial dependency in the geo-location sequence while retains the information in a fine granularity.

The architecture of Geo-Conv layer is shown in Fig. 3. For each GPS point p_i in the sequence, we first use a non-linear mapping

$$loc_i = \tanh(W_{loc} \cdot [p_i.lat \circ p_i.lng]) \quad (1)$$

to map the i-th GPS coordinates into vector $loc_i \in \mathbf{R}^{16}$, where $\circ$ indicates the concatenate operation and W_{loc} is a learnable weight matrix. Thus, the output sequence $loc \in \mathbf{R}^{16\times|T|}$ represents the non-linearly mapped locations. Note that such sequence can be seen as a 16-channel input. Each channel describes the geographical features of the original GPS sequence. We introduce a convolutional filter, with parameter matrix $W_{conv} \in \mathbf{R}^{k\times 16}$ (we refer to k as the *kernel size* of the filter). It applies the convolution operation on the sequence loc, along with a one-dimensional sliding window. We use $*$ to denote the convolutional operation. The i-th dimension of its output is denoted as,

$$loc_i^{conv} = \sigma_{cnn}(W_{conv} * loc_{i:i+k-1} + b) \quad (2)$$

where b is the bias term, $loc_{i:i+k-1}$ is the subsequence in loc from index i to index $i+k-1$ and σ_{cnn} is the corresponding activation function.

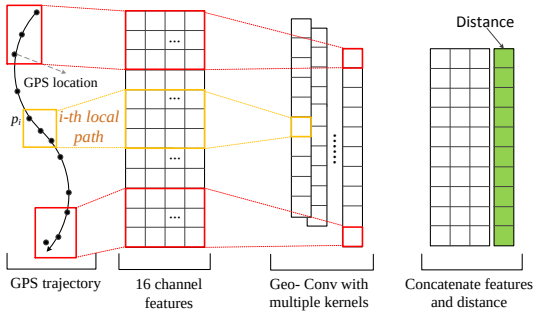

Figure 3: Geo-Conv Layer.

Definition 3 (Local Path) *We refer to the sub-sequence from point p_i to point p_{i+k-1} as the i-th* local path.

Thus, loc_i^{conv} essentially captures the spatial feature of the i-th local path. By concatenating the outputs of c filters, we obtain a feature map of local paths with shape $\mathbf{R}^{c\times(|T|-k+1)}$. In the literature of natural language processing, such architecture is so called 1D-CNN (Kim 2014).

Nevertheless, in our task, the travel time is highly related to the total distance of the path. It is hard for 1D-CNN to extract the geometric distance directly from the raw longitudes/latitudes. Therefore, in Geo-Conv layer, we further append a column to the previously obtained feature map. The i-th element of the new appended column (green part in Fig. 3) is $\sum_{j=i+1}^{i+k-1} \mathrm{Dis}(p_{j-1}, p_j)$, i.e., the distance of the i-th local path. Thus, we obtain the final feature map of shape $\mathbf{R}^{(c+1)\times(|T|-k+1)}$ by our Geo-Conv layer. We denote this feature map as loc^f.

Recurrent Layer The feature map loc^f captures the spatial dependencies of all the local paths. To further capture the temporal dependencies among these local paths, we introduce the recurrent layers in our model. The recurrent neural network (RNN) is an artificial neural network which is widely used for capturing the temporal dependency in sequential learning, such as the natural language processing and speech recognition (Krizhevsky, Sutskever, and Hinton 2012; Graves, Mohamed, and Hinton 2013). The recurrent neural network is able to "memorize" the history in the processed sequence. When processing the current time step in the sequence, it updates its memory (also called hidden state) according to the current input and the previous hidden state. The output of the recurrent neural network is the hidden state sequence at all the time steps in the sequence.

In our model, the input sequence of the recurrent neural network is the feature map loc^f outputted by Geo-Conv layer. The feature map loc^f can be regarded as a sequence of spatial features with length $|T| - k + 1$. Moreover, we find that incorporating the attributes information is helpful to further enhance the estimating ability of the recurrent layers (recall that we have obtained the attributes representation vector $attr$ in the attribute component). Thus, in simple terms, the updating rule of our recurrent layer can be expressed as

$$h_i = \sigma_{rnn}(W_x \cdot loc_i^f + W_h \cdot h_{i-1} + W_a \cdot attr) \qquad (3)$$

where h_i is the hidden state after we processed the i-th local path and σ_{rnn} is the non-linear activation function which we specify in the experiment part. W_x, W_h and W_a are learnable parameter matrices used in the recurrent layer. In practice, Eq.(3) usually fails in processing the long sequence due to vanishing gradient and exploding gradient problems (Hochreiter and Schmidhuber 1997). To overcome such issue, we use two stacked Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) layers instead. LSTM introduces an input gate and a forget gate to control the in/out information flow. Such gates enables LSTM to forget some unimportant information and effectively alleviate the gradient vanishing/exploding problem. Furthermore, it has been shown that a stacked LSTM is more efficient to increase the model capacity compared with a single layer LSTM (Yao et al. 2017).

Now, by utilizing the Geo-Conv layer and the recurrent layer, we obtain the sequence $\{h_1, h_2, \ldots, h_{|T|-k+1}\}$ which represents the spatio-temporal features of the raw GPS sequence.

Multi-task Learning Component

We finally introduce a multi-task learning component which combines the previous components and estimates the travel time of input path. As we mentioned in the introduction, prior work in estimating the travel time can be divided into two types, the *individual estimation* and the *collective estimation*. If we adopt the individual estimation, the local errors may accumulate since such method does not consider the spatio-temporal dependencies among the local paths. In the mean time, if we use the collective estimation, we usually face the data sparsity problem since only a few trajectories traveled through the entire path or the longer sub-paths.

In our model, we combine these two methods with our multi-task learning component. During the training phase, we enforce the multi-task learning component to accurately estimate the travel time of both entire path and each local path simultaneously. During the test phase, we eliminate the local path estimation part and report the estimated travel time of the entire path.

Estimate the local paths Recall that we use the spatio-temporal component to obtain a sequence $\{h_1, h_2, \ldots, h_{|T|-k+1}\}$. Here each h_i corresponds to the spatio-temporal feature of local path $p_i \to p_{i+1} \to \ldots \to p_{i+k-1}$. We simply use two stacked fully-connected layers with size 64 and 1 to map each h_i to a scalar r_i. Here r_i represents the travel time of the i-th local path, as shown in Fig. 2.

Estimate the entire path Note that the length of spatio-temporal feature sequence $\{h_i\}$ is also variable. To estimate the travel time of the entire path directly, we first need to transform the feature sequence into a fixed length vector. A simple method to achieve this is to use *mean pooling*, i.e., $h_{mean} = \frac{1}{|T|-k+1} \sum_{i=1}^{|T|-k+1} h_i$. Such mechanism is simple yet effective. However, the mean pooling method treats all the spatio-temporal features h_i equally. In fact, the uncertainty of accurately estimating the travel time is usually caused by several critical local paths. For example, if the path contains multiple road intersections, traffic lights, or road segments which can be extremely congested, we should pay more attention to such parts since they are more difficult to estimate. Motived by this, in our model, we adopt the *attention mechanism* instead of the mean pooling. The attention mechanism is essentially the weighted sum of sequence $\{h_i\}$, where the weights are parameters learned by the model. Formally, we have that

$$h_{att} = \sum_{i=1}^{|T|-k+1} \alpha_i \cdot h_i \qquad (4)$$

where α_i is the weight for the i-th local path, and the summation of all α_i equals 1. To learn the weight parameter α, we consider the spatial information of the local paths, as well as the external factors such as the start time, the day of week and the weather condition.

In our model, the vector $attr$ in the attribute component captures the effect of external factors, and the feature sequence $\{h_i\}$ captures the spatio-temporal features of local paths. Thus, we devise our attention mechanism based on $attr$ and $\{h_i\}$:

$$\begin{aligned} z_i &= \langle \sigma_{att}(attr), h_i \rangle \\ \alpha_i &= \frac{e^{z_i}}{\sum_j e^{z_j}} \end{aligned} \qquad (5)$$

where $\langle . \rangle$ is the inner product operator and σ_{att} is a non-linear mapping which maps $attr$ to a vector with the same length as h_i. Substituting Eq. (5) into Eq. (4), we obtain the attention based vector h_{att}.

Finally, we pass h_{att} to several fully-connect layers with equal size (we use the size of 128 in our experiment). The fully-connected layers are connected with residual connections (Residual fully-connected Blocks of Fig. 2) which is a technique to train a very deep neural network (He et al. 2015). The residual connection adds "shortcuts" between different layers. Thus, previous information flow can skip one or more non-linear layers through the shortcut and the skipped layers

just need to learn the "residual" of the non-linear mapping. In our model, we use σ_{f_i} to denote the i-th residual fully-connected layer. For the first layer, the output of this layer is $\sigma_{f_1}(h_{att})$. For the rest of the residual fully-connected layers, suppose the output of the i-th layer is x. Then, the output of the $(i+1)$-th layer can be represented as $x \oplus \sigma_{f_{i+1}}(x)$ where $\oplus$ is the element-wise add operation. It has been shown that training the neural networks with residual connections is easier and more robust (He et al. 2015). At last, we use a single neuron to obtain the estimation of the entire path, which we denote as r_{en}.

Model Training

We finally present the training procedure of our model. Our model is trained end to end. During the training phase, we use the *mean absolute percentage error (MAPE)* as our objective function. Since MAPE is a relative error, we can enforce our model to provide accurate results for both the short paths and the long paths. However, we use multiple criterions to evaluate our model, including the rooted mean squared error (RMSE) and the mean absolute error (MAE).

Recall that during the training phase, we estimate the travel time of all the local paths and the entire path simultaneously. For the local path estimation, we define the corresponding loss as the average loss of all local paths, i.e.,

$$L_{local} = \frac{1}{|T|-k+1} \sum_{i=1}^{|T|-k+1} \left| \frac{r_i - (p_{i+k-1}.ts - p_i.ts)}{p_{i+k-1}.ts - p_i.ts + \epsilon} \right| \tag{6}$$

where ϵ is a small constant to prevent the exploded loss value when the denominator is close to 0 (in our experiment, we set ϵ as 10 seconds). For the entire path, we define the corresponding loss as

$$L_{en} = |r_{en} - (p_{|T|}.ts - p_1.ts)|/(p_{|T|}.ts - p_1.ts). \tag{7}$$

Our model is trained to minimize the weighted combination of two loss terms

$$\beta \cdot L_{local} + (1-\beta) \cdot L_{en} \tag{8}$$

where β is the combination coefficient that linearly balances the tradeoff between L_{local} and L_{en}. Be default, during the test phase, we use the travel time estimation of the entire path r_{en} as our final estimation.

Experiment

In this section, we report our experimental results on two large scale real-world datasets. We first compare our model with several baseline methods, including the state-of-art collective estimation method TEMP (Wang et al. 2016b). We then present the effectiveness of our model by a set of controlled experiments[3].

Experiment Setting

Data Description We evaluate our model on two large scale datasets:

- **Chengdu Dataset**: Chengdu Dataset consists of $9,737,557$ trajectories (1.4 billion GPS records) of 14864 taxis in August 2014 in Chengdu, China. The shortest trajectory contains only 11 GPS records (2km) and the longest trajectory contains 128 GPS records (41km).
- **Beijing Dataset**: Beijing Dataset consists of $3,149,023$ trajectories (0.45 billion GPS records) of 20442 taxis in April 2015 in Beijing, China. The shortest trajectory contains 15 GPS records (3.5km) and the longest trajectory contains 128 GPS records (50km).

The trajectories in both datasets are associated with the corresponding weekID, timeID and $\mathrm{driverID}$. For Beijing Dataset, we further collected the corresponding weather conditions (16 types including sunny, rainy, cloudy etc) as well as the road ID of each GPS point.

Parameter Setting The parameters we used in our experiment are described as follows:

- In the attribute component, we embed weekID to $\mathbf{R}^3$, timeID to $\mathbf{R}^{16}$, $\mathrm{driverID}$ to $\mathbf{R}^8$ and the corresponding weather type to $\mathbf{R}^3$.
- In the geo-conv layer, we fix the number of filters $c = 32$ and we use ELU function (Clevert, Unterthiner, and Hochreiter 2015) as the activation σ_{cnn} in Eq.(2). The ELU is defined as $\mathsf{ELU}(x) = e^x - 1$ for $x \leq 0$ and $\mathsf{ELU}(x) = x$ for $x > 0$. For the kernel size k, we evaluate our model under different values of k.
- In the recurrent layer, we use $\tanh$ as the activation σ_{rnn} in Equ. 3. We fix the size of the hidden vector h_i as 128.
- In the multi-task learning component, we first use a fully-connected layer with $\tanh$ activation as σ_{att} in Eq. (5). we use $\mathsf{ReLU}(x) = \max(0, x)$ (Glorot, Bordes, and Bengio 2011) as the activation of residual fully-connected layers. We fix the number of the residual fully-connected layers as 4 and the size of each layer as 128. Furthermore, we evaluate our model for different combination coefficient β in Eq. (8) from 0.0 to 0.99.

For each dataset, we use the trajectories generated in the last 7 days as the test set and the rest of trajectories as the training set. We adopt Adam optimization algorithm (Kingma and Ba 2014) to train the parameters. The learning rate of Adam is 0.001 and the batch size during training is 400. We train the model for 100 epochs and select the best models by 5-fold cross-validation.

Our model is implemented with PyTorch 2.0, a widely used Deep Learning Python library. We train/evaluate our model on the server with one NVIDIA GTX1080 GPU and 24 CPU (2960v3) cores.

Performance Comparison

To demonstrate the strength of our model. We first compare our model with several baseline methods, including:

- **AVG:** We simply calculate the average speed in the city during a specific time interval (e.g. 13:00-14:00 PM on Monday). We estimate the travel time of given trajectory based on its starting time and the historical average speed.
- **TEMP:** TEMP (Wang et al. 2016b) is the state-of-art collective estimation method. It estimates the travel time of

[3]The code and the sample data can be downloaded at https://github.com/UrbComp/DeepTTE

Table 1: Performance Comparison

	Chengdu			Beijing		
	MAPE (%)	RMSE (sec)	MAE (sec)	MAPE (%)	RMSE (sec)	MAE (sec)
AVG	28.1	533.57	403.71	24.78	703.17	501.23
D-TEMP	22.82	441.50	323.37	19.63	606.76	402.50
GBDT	19.32 ± 0.04	357.09 ± 2.44	266.15 ± 2.24	19.98 ± 0.02	512.96 ± 3.96	393.98 ± 2.99
MlpTTE	16.90 ± 0.06	379.39 ± 1.94	265.47 ± 1.53	23.73 ± 0.14	701.61 ± 1.82	489.54 ± 1.61
RnnTTE	15.65 ± 0.06	358.74 ± 2.02	246.52 ± 1.65	13.73 ± 0.05	408.33 ± 1.83	275.07 ± 1.48
DeepTTE	$\mathbf{11.89 \pm 0.04}$	$\mathbf{282.55 \pm 1.32}$	$\mathbf{186.93 \pm 1.01}$	$\mathbf{10.92 \pm 0.06}$	$\mathbf{329.65 \pm 2.17}$	$\mathbf{218.29 \pm 1.63}$

the given path based on the "neighbor" trajectories, i.e., the trajectories which have the closed starting and destination as the query path. This work outperforms the most outstanding individual TTE (Wang, Zheng, and Xue 2014) as well as Bing/Baidu Map API in their experiment. However, there are about 10% paths that the original TEMP method can not estimate due to the lack of neighbor trajectories. For those paths, we enlarge the neighborhood dynamically until we can find enough neighbor trajectories (10 trajectories in our experiment). We refer to such implementation as *dynamic TEMP* (D-TEMP).

- **GBDT:** Gradient Boosting Decision Tree (GBDT) is a powerful ensemble method (Friedman, Hastie, and Tibshirani 2001) and widely used in practice. In our problem, the input of GBDT is as same as the input of DeepTTE, including all the inputs in the attribute component and the raw GPS sequence. Note that since the length of GPS sequence is variable, GBDT can not handle such sequence directly. Here, we uniformly sample (with replacement) each GPS sequence to a fixed length of 128.
- **MlpTTE:** We use a 5-layer perceptron with ReLU activation to estimate the travel time. The input of MlpTTE is almost the same as GBDT, except that the categorical values are properly embedded to real vectors. The size of hidden layers in MlpTTE is fixed as 128.
- **RnnTTE:** RnnTTE is also a simplified model of DeepTTE. We use a vanilla RNN and mean pooling to process the raw GPS sequence into a 128-dimensional feature vector. We concatenate such feature vector and the output of the attribute component. The concatenation then is passed to the residual fully-connected layers to obtain the estimation.

For our model DeepTTE, we fix the kernel size as 3 and combination coefficient β as 0.3. We repeat each experiment for 5 times. We report the mean and the standard variation of the different runs. The experiment result is shown in Table 1.

As we can see, simply using the average speed leads to a very inaccurate result. In contrast, the collective method D-TEMP and the ensemble method GBDT are much better than AVG. MlpTTE shows a good performance on Chengdu Dataset. However, it does not consider the spatio-temporal dependencies in the data. For the dataset which contains more complex traffic conditions (e.g., Beijing Dataset), we find that MlpTTE tends to overfit the training data and thus leads to a bad result. RnnTTE use the recurrent layers to capture the temporal dependencies in the data. It achieves 15.65% and 13.73% on two datasets which are much better than above mentioned methods. Our model DeepTTE further

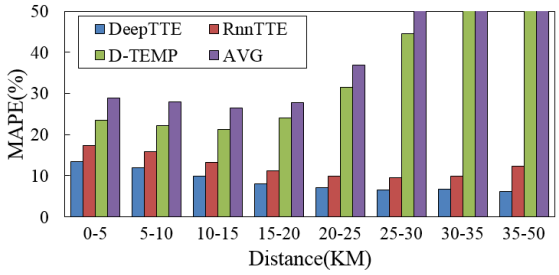

Figure 4: Error rates for trajectories with different lengths.

significantly outperforms RnnTTE. The error rates in two datasets are only 11.89% and 10.92%. We use Paired T-Test (Friedman, Hastie, and Tibshirani 2001) to further test the significance of our model. The p-value for all the test pairs are less than 10^{-8} which demonstrates that our model is significantly better than baselines. In Fig. 4, we compare the model performances for the trajectories with different lengths. When the path length goes larger, AVG and TEMP methods face a serious individual-collective trade-off problem. As a consequence, the error rates increase sharply. RnnTTE shows a better result for the trajectories less than 35 km. However, it fails to handle the long paths (with length greater than 35 km). In contrast, our model demonstrates remarkable results for longer paths. Finally, recall that there are about 10% paths that the original TEMP can not estimate. For the rest of paths in two datasets, the average error rate is 19.96% for the original TEMP but only 11.21% for our model.

Effect of Attribute Component

We show the effectiveness of different attributes, including the weatherID, driverID and weekID. We devise a set of controlled experiments on Beijing Dataset. For each experiment, we eliminate exactly one attribute.We find that weatherID and weekID affect the estimation significantly. Eliminating such two attributes causes an error growth of 1.09% and 0.77% respectively. This also conforms to our intuitive sense, i.e., we usually spend much more time for traveling the same path under bad weather conditions. Eliminating the driver information causes an error increment of 0.30% which seems not significant. However, we stress the trajectories in our dataset are generated by taxi drivers. Most of the taxi drivers are very experienced and have similar driving habits. The driver information might be more useful for estimating the travel time of normal people. We leave it as an intriguing direction

for future work.

Effect of Geo-Conv

We first evaluate our model under the absence of Geo-Conv layer. To achieve this, we eliminate the geo-conv layer and directly pass the sequence of $\{p_i.lat \circ p_i.lng\}$ to the following LSTM layers. The MAPE in Chengdu/Beijing Dataset under this setting is 13.14% and 12.68% (comparing with 11.89% and 10.92%).

We further test our model if we increase the kernel size k in Equ. (2) from 3 to 4, 5 and 6. We find that the performance of our model in Beijing Dataset decreases when the kernel size goes larger. The percentage errors for $k = 4, 5, 6$ are 11.18%, 11.31% and 11.38%. The reason might be that when the kernel size $k = 3$, a local path consists of two consecutive road segments which can exactly represent the turnings or road intersections (see Fig. 1). Thus, it is more easily for our model to capture such spatial features in the local paths.

Effect of Multi-task Learning

To show the effectiveness of the multi-task learning component, we first evaluate our model under different combination coefficient β from 0.0 to 0.99. The result is shown in Fig 5. We find that our model is pretty robust for a wide range of β. The high error rates are only found in two ends, i.e., when $\beta = 0$ or when β goes to 1. We then replace the attention mechanism in the multi-task learning component with the meaning pooling method. The error rates in Chengdu/Beijing Dataset increase to 12.09% and 11.11% respectively.

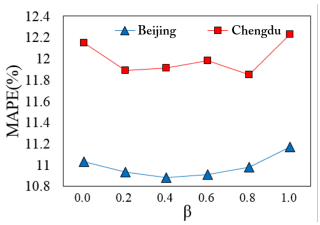

Figure 5: Error rates for different β.

Incorporate Road Information

Note that our model does not rely on any map-matching algorithm but directly handle the raw GPS sequence. Nevertheless, it is very easy to incorporate the road information into our model when it is available. We collect the corresponding road ID of each GPS point in Beijing Dataset and embed each road ID to a 32-dimensional vector. For each GPS point, we use $p_i.rid$ to denote the embedded road ID of p_i. To utilize such road information, we can simply modify Eq. (1) into

$$loc_i = \tanh(W_{loc} \cdot [p_i.lat \circ p_i.lng \circ p_i.rid])$$

in Geo-Conv layer. The MAPE of DeepTTE after incorporating the road information decreases from 10.92% to 10.59%.

Predicting Time

Despite the training time of DeepTTE is longer, the predicting is very efficient. During the test phase, estimating every 1000 paths takes DeepTTE 0.037s, RnnTTE 0.031s, MlpTTE 0.029s on GPU (including I/O time), GBDT 0.017s, and TEMP 0.47s. The original TEMP is slower for searching neighbors. Despite the searching can be accelerated with more advanced data structures, it is out of scope of this paper.

Related work

There is a large body of literature on the estimation of travel time; we only mention a few closely related ones.

Road Segment-Based Travel Time Estimation

Estimating travel time has been studied extensively (Zhong and Ghosh 2001; Sevlian and Rajagopal 2010; Pan, Demiryurek, and Shahabi 2012). However, these works estimated the travel time of individual road segment without considering the correlations between the roads. (Yang, Guo, and Jensen 2013) used a spatial-temporal Hidden Markov Model to formalize the relationships among the adjacent roads. (Wang et al. 2016a) improved this work through an ensemble model based on two observed useful correlations in the traffic condition time series. (Wang et al. 2016c) proposed an error-feedback recurrent Convolutional neural network called eRCNN for estimating the traffic speed on each individual road. These studies considered the correlation between different roads. However, they focused on accurately estimating the travel time or speed of individual road segment. As we mentioned in the Section , the travel time of a path is affected by various factors, such as the number of road intersections and the traffic lights in the path. Simply summing up the travel time of the road segments in the path does not lead to an accurate result (Jenelius and Koutsopoulos 2013).

Path-Based Travel Time Estimation

(Rahmani, Jenelius, and Koutsopoulos 2013) estimated the travel time of a path based on the historical data of the path. However, the historical average based model may lead to a poor accuracy. Moreover, as new queried path may be not included in the historical data, it suffers from the data sparse problem. (Yuan et al. 2013) built a landmark graph based on the historical trajectories of taxis, where each landmark represents a single road. They estimate the travel time distribution of a path based on the landmark graph. However, as the landmarks are selected from the top-k frequently traversed road, the roads with few traveled records can not be estimated accurately. Furthermore, (Wang, Zheng, and Xue 2014) estimated the travel time of the path, based on the sub-trajectories in the historical data. They used the tensor decomposition to complete the unseen sub-trajectory and such method enhance the accuracy effectively. Nevertheless, it still suffers from the data sparsity problem since there are many sub-trajectories which were visited by very few drivers.

Deep Learning in Spatial Temporal Data

Recently, the deep learning techniques demonstrate the strength on spatio-temporal data mining problems. (Song, Kanasugi, and Shibasaki 2016) built an intelligent system called DeepTransport, for simulating the human mobility and transportation mode at a citywide level. (Zhang, Zheng, and Qi 2017) proposed a deep spatio-temporal residual network for predicting the crowd flows. (Dong et al. 2016) studied characterizing the driving style of different drivers by a stacked recurrent neural network. To best of knowledge, no

prior work studies estimating the travel time of the whole path based on the deep learning approach.

Conclusion

In this paper, we study estimating the travel time for any given path. We propose an end-to-end framework based on deep neural networks. Our model can effectively capture the spatial and temporal dependencies in the given path at the same time. Our model also considers various factors which may affect the travel time such as the driver habit, the day of the week. We conduct extensive experiments on two very large scale real-world datasets. The results show that our model achieve a high estimation accuracy and outperforms the other off-the-shell methods significantly.

Acknowledgment

- Yu Zheng were supported by NSFC (Nos. 61672399, U1401258), and the 973 Program (No. 2015CB352400).
- Wei Cao and Jian Li were supported by NSFC (Nos. 61772297, 61632016), and the 973 program (No. 2015CB358700).

References

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Dong, W.; Li, J.; Yao, R.; Li, C.; Yuan, T.; and Wang, L. 2016. Characterizing driving styles with deep learning. *CoRR* abs/1607.03611.

Friedman, J.; Hastie, T.; and Tibshirani, R. 2001. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin.

Gal, Y., and Ghahramani, Z. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, 1019–1027.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 315–323.

Graves, A.; Mohamed, A.-r.; and Hinton, G. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, 6645–6649. IEEE.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. *Computer Science*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735.

Jenelius, E., and Koutsopoulos, H. N. 2013. Travel time estimation for urban road networks using low frequency probe vehicle data. *Transportation Research Part B: Methodological* 53:64–81.

Kim, Y. 2014. Convolutional neural networks for sentence classification. In *In EMNLP*. Citeseer.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Pan, B.; Demiryurek, U.; and Shahabi, C. 2012. Utilizing real-world transportation data for accurate traffic prediction. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, 595–604. IEEE.

Rahmani, M.; Jenelius, E.; and Koutsopoulos, H. N. 2013. Route travel time estimation using low-frequency floating car data. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, 2292–2297. IEEE.

Sevlian, R., and Rajagopal, R. 2010. Travel time estimation using floating car data. *arXiv preprint arXiv:1012.4249*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, X.; Kanasugi, H.; and Shibasaki, R. 2016. Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level. In *IJCAI*, 2618–2624.

Wang, D.; Cao, W.; Xu, M.; and Li, J. 2016a. Etcps: An effective and scalable traffic condition prediction system. In *International Conference on Database Systems for Advanced Applications*, 419–436. Springer.

Wang, H.; Kuo, Y.-H.; Kifer, D.; and Li, Z. 2016b. A simple baseline for travel time estimation using large-scale trip data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 61. ACM.

Wang, J.; Gu, Q.; Wu, J.; Liu, G.; and Xiong, Z. 2016c. Traffic speed prediction and congestion source exploration: A deep learning method. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, 499–508. IEEE.

Wang, Y.; Zheng, Y.; and Xue, Y. 2014. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 25–34. ACM.

Yang, B.; Guo, C.; and Jensen, C. S. 2013. Travel cost inference from sparse, spatio temporally correlated time series using markov models. *Proceedings of the VLDB Endowment* 6(9):769–780.

Yao, S.; Hu, S.; Zhao, Y.; Zhang, A.; and Abdelzaher, T. 2017. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, 351–360.

Yuan, J.; Zheng, Y.; Xie, X.; and Sun, G. 2013. T-drive: enhancing driving directions with taxi drivers' intelligence. *Knowledge and Data Engineering, IEEE Transactions on* 25(1):220–232.

Zhang, J.; Zheng, Y.; Qi, D.; Li, R.; and Yi, X. 2016. Dnn-based prediction model for spatio-temporal data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 92. ACM.

Zhang, J.; Zheng, Y.; and Qi, D. 2017. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, 1655–1661.

Zhong, S., and Ghosh, J. 2001. A new formulation of coupled hidden markov models. *Dept. Elect. Comput. Eng., Univ. Austin, Austin, TX, USA*.