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

Abstract

The travel time estimation is an important yet challenging problem. It is a fundamental ingredient of many location-based services such as navigation, route planing systems etc. In this paper, given a path and the corresponding start time, we study the problem of estimating the time for traveling the path. Prior work usually focuses on estimating the travel times of individual road segments or sub-paths, and then summing up these estimated travel times. However, such approach leads to an inaccurate estimation, since the travel time is also affected by the number of road intersections or traffic lights in the path, and the estimation errors for individual road may accumulate. We propose an end-to-end framework for Travel Time Estimation called *DeepTTE*. Our model estimates the travel time of the whole path directly, based on deep recurrent neural networks. In our model, we consider the spatial and temporal dependency in the path as well as various factors which may affect the travel time such as the driver's habit, the day of the week etc. We conduct extensive experiment result on a large scale dataset. The experiment result shows that our model significantly outperforms the other existing methods.

1 Introduction

Estimating the travel time for a given path is a fundamental problem in route planning, navigation, and traffic dispatching. An accurate estimation of travel time helps people better planning their routes. Almost all the electronic maps and online car-hailing services provide the travel time estimation in their apps, such as Google Map, Uber, Didi, etc. When a user searches the routes to the destination, the map app provides several candidate routes with estimated travel times (and possibly other measures such as gas consumptions, tolls) for the user to decide. Although the problem has been widely studied in the past years, providing an accurate travel time is still a challenging problem. Prior work usually focuses on estimating the travel speed/time of an individual road segment [Yang *et al.*, 2013; Wang *et al.*, 2016b; Pan *et al.*, 2012]. However, 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 etc. Simply summing up the travel time of the road segments in the path does not lead to an accurate estimation, as the errors may accumulate [Jenelius and Koutsopoulos, 2013]. Alternatively, some work decomposes the path into several longer sub-trajectories instead of the road segments and estimates the travel time based on the sub-trajectories [Wang *et al.*, 2014]. Although such method enhances the estimation accuracy, it suffers from the data sparsity problem since there are many sub-trajectories that were visited by very few drivers.

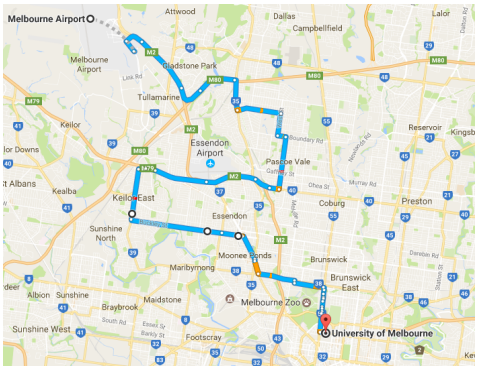

Figure 1: An illustration of the query path.

In this paper, we view a path as a sequence of location points (see Fig. 1 for an illustration), and we learn to estimate the travel time from historical trajectories based on the deep learning approach. To make our exposition more concrete, we first present some challenges in our problem.

- To estimate the travel time, we have to consider the spatial and temporal dependencies in the given trajectory at the same time. Prior work usually formalizes such dependency by discretizing the trajectory into several grids [Zhang *et al.*, 2016a] or road segments [Yang *et al.*, 2013]. However, on the one hand discretizing the GPS points into grids causes the information loss due to the coarse granularities. On the other hand, inferring the travel time based on the individual road segments misses the effects of road intersections and traffic lights. Few work studies capturing the spatio-temporal dependency of GPS points directly.
- The travel time of a specific path can be very different at different time intervals. For example, in the peak hours, it usually takes much longer time than that in non-peak hours. Even for a fixed time interval, different days of the week reveal very different distributions of travel times. Prior

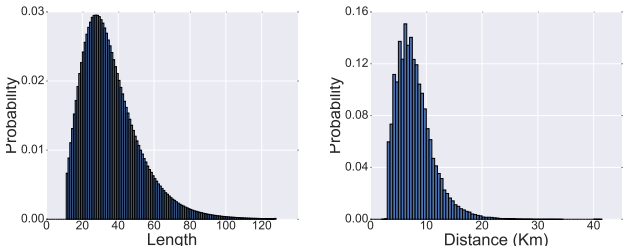

Figure 2: Length and Distance Distribution

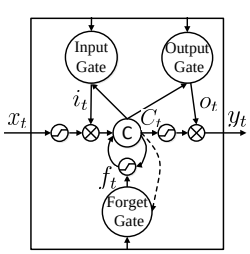

Figure 3: LSTM

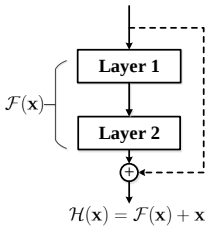

Figure 4: Residual Network

work usually builds several sub-models for different days of the week [Hu *et al.*, 2016; Yuan *et al.*, 2013]. Such implementations, on the one hand, make the model tedious, on the other hand, each sub-model only utilizes a small part of data which may suffer from the lack of training data.

- Different drivers have different driving habits. The experienced drivers are usually very familiar with the traffic conditions in the city and drive very fast. On the contrary, the new drivers usually drive relatively slow which leads to a longer travel time under the same condition. Most of the prior work does not consider the effects of drivers (the driver information is available in our dataset) when estimating the travel time.
- Different historical trajectories have very different values of length (i.e., the number of points) or distance. Fig. 2 shows the distribution of the values of trajectory length and distance in our dataset. It is not easy to process the variable-length trajectories directly with the traditional machine learning models such as Random Forest, Gradient Boosting, etc.

To address the above challenges, we propose an end-to-end framework, based on deep recurrent neural networks. The primary contribution of this paper can be summarized as follows:

- We design an end-to-end framework for Travel Time Estimation, called *DeepTTE*, based on deep recurrent neural networks. We incorporate various factors which may affect the travel time (e.g., the driver, the day of the week and the time interval, etc.) in a unified model, instead of building several sub-models manually.
- We devise a novel neural network architecture which can easily process variable-length GPS trajectories. Furthermore, by carefully designing the input sequence, our model effectively captures the spatial and temporal dependency in the trajectory simultaneously without much information loss.
- We further extend our model to a *multi-task learning* model by introducing an auxiliary component. The auxiliary component estimates the travel time between each pair of adjacent GPS points which we take as the auxiliary output. We show that the auxiliary component effectively improves the model performance.
- We conduct extensive experiments on a large scale data set which consists of GPS points generated by over $14,684$ taxis collected in one month in Chengdu, China. Our model achieves a high-quality prediction result with the error rate of 12.74% which significantly outperforms several other off-the-shell machine learning algorithms, as shown in Section 4.

This paper is organized as follows. In Section 2, we formally define our problem and present several preliminaries of our model. In Section 3, we describe our model architecture in detail. We conduct extensive experiments to show the strength of our model in Section 4. Finally, we present some related work and conclude our paper in Section 5 and Section 6.

2 Preliminary

2.1 Problem Definition

Definition 1 (Historical Trajectory) *We define a* historical trajectory T *as a sequence of consecutive historical GPS points, i.e.,* $T = \{p_1, \ldots, p_{|T|}\}$. *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 corresponding driver ID which we denote as* $\mathrm{driverID}$.

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, based on the historical trajectories as we defined in Definition. 1. During the test phase, given the path* S, *the corresponding driver ID and the start time, our goal it to estimate the travel time from the source to the destination through* S. *We assume that the travel path* S *is specified by the user or generated by the route planing apps.*

To make the testing data consistent with the training data, we convert the path S *to a sequence of location points by sampling. Each location is represented as a pair of longitude and latitude.*

Remark: In our experiment, we remove the timestamp in the historical trajectories and use such trajectories as the test data. In this paper, we do not consider how to optimize the path S.

2.2 Recurrent Neural Network

Recurrent neural network (RNN) is an artificial neural network which contains an internal state and a directed cycle between units. It is suitable for capturing the temporal dependency and has been used successfully in sequential learning such as the natural language processing, speech recognition, etc [Krizhevsky *et al.*, 2012; Mesnil *et al.*, 2013; Graves *et al.*, 2013]. Especially, an internal state can be viewed as the "memory" of previous time steps and it captures the temporal dependency with all the previous time steps of input sequence when calculating a new internal state. However, vanilla RNN failed in processing the long sequence due to vanishing gradient and exploding gradient problems [Hochreiter and Schmidhuber, 1997]. To overcome such issue, Long

Short-Term Memory was developed [Hochreiter and Schmidhuber, 1997]. An LSTM contains several LSTM units. Each LSTM unit contains a memory cell and three gates which are used to control the flow of information in/out of their memory. Mathematically, given the input vector $x = \{x_0, x_1, \ldots, x_n\}$, and denoting the output as $y = \{y_0, y_1, \ldots, y_n\}$, the expected output (the internal states of LSTM) are updated as follows:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{yi}y_{t-1} + W_{ci}c_{t-1} + b_i),\\
f_t &= \sigma(W_{xf}x_t + W_{yf}y_{t-1} + W_{cf}c_{t-1} + b_f),\\
c_t &= f_t \otimes c_{t-1} + i_t \otimes tanh(W_{xc}x_t + W_{yc}y_{t-1} + b_c)\\
o_t &= \sigma(W_{xo}x_t + W_{yo}y_{t-1} + W_{co}c_{t-1} + b_o),\\
y_t &= o_t \otimes tanh(c_t)
\end{aligned}
$$

where σ denotes the logistic sigmoid function and $\otimes$ denotes elementwise multiplication (See Fig. 3 for an illustration). In this paper, we utilize two stacked LSTM layers to process the trajectory and obtain the corresponding feature vector.

2.3 Residual Network

Many non-trivial tasks have significantly benefited from very deep neural networks. However, an obstacle to training a very deep model is the gradient *vanishing/exploding* problem. To overcome such issue, He et al. [He *et al.*, 2015] proposed a new network architecture called the residual network (ResNet), which allows one to train very deep Convolutional neural networks successfully.

The residual learning adds the *shortcut connections* (dashed line in Fig. 4) between different layers. Thus, the input vector can be directly passed to the following layers through the shortcut connections. For example, in Fig. 4, we use $\mathbf{x}$ to denote the input vector and $\mathcal{H}(\mathbf{x})$ to denote the desired mapping after two stacked layers. Instead of learning the mapping function $\mathcal{H}(\mathbf{x})$ directly, we learn the residual mapping $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$ and broadcast $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ to the following layers. It has been shown that optimizing the residual mapping is much easier than optimizing the original mapping [He *et al.*, 2015], which is the key to the success of deep residual network.

3 Model Architecture

We first describe the architecture of our model as shown in Fig. 5. Our model consists of four parts: the *attribute* component, the *sequence learning* component, the *residual* component, and the *auxiliary* component.

The *attribute* component processes the attributes of the driver ID, the current day of the week and the timeslot of the start time. The *sequence learning* component processes the GPS location sequence. The *residual* component utilizes the outputs of the first two components to estimate of the given path. Finally, we introduce an *auxiliary* component which estimates the travel times between consecutive GPS points and helps improve the model performance.

3.1 Attribute Component

In the attribute component, we first process the attributes of the driver ID, the day of the week and the timeslot of travel start[1]. We use $\mathrm{driverID}$, weekID and timeID to denote these three attributes respectively. Since these attributes are categorical,

[1]We divide one day into 1440 timeslots.

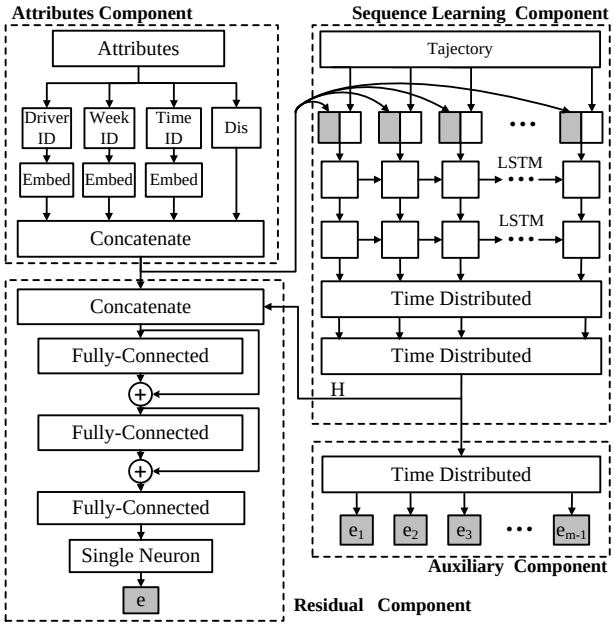

Figure 5: Overview of DeepETT.

to feed the attributes into the neural network, we need to transform them into real values. We use the embedding method [Mesnil *et al.*, 2013] to process each categorical attribute. The embedding method maps each categorical value $v \in [V]$ to a low-dimensional space $\mathbb{R}^{E\times 1}$ by a matrix $W \in \mathbb{R}^{V\times E}$ (we refer to such space as the embedding space). An important property of embedding method is that the categorical values with similar semantic meaning are usually very close in the embedding space. Thus, the embedding method helps us discover the similarities in the data [Mesnil *et al.*, 2013]. In our case, for example, for some specific paths, driving at 7:00 A.M. usually takes similar time with that at 18:00 P.M. since they are both peak hours. Thus, we use the embedding method to reduce the computational cost and help us discover such similarities in the data automatically.

We further consider the attribute of *travel distance*. We denote $\Delta d_{p_a \to p_b}$ as the travel distance from GPS point p_a to p_b, i.e., $\Delta d_{p_a \to p_b} = \sum_{i=a}^{b-1} \mathsf{Dis}(p_i, p_{i+1})$ where Dis is the geographic distance between two GPS points. Then, we concatenate the embedded vectors of $\mathrm{driverID}$, weekID and timeID together with the travel distance $\Delta d_{p_1 \to p_n}$ to form the output of the attribute component, which we denote as $attr$.

3.2 Sequence Learning Component

In this part, we demonstrate how the sequence learning component extracts the compressed information from the trajectory using a novel architecture.

Recall that each trajectory T is represented by a sequence of GPS points $\{p_1, \ldots, p_{|T|}\}$. Since different trajectories have different lengths, to handle the sequences with variable lengths, we propose two candidate processing methods.

Sampling Trick

We randomly sample each trajectory to a fixed length m. We denote the indices of the sampled points as an ordered list L

and the sampled trajectory as T'. Furthermore, to make sure that the source and the destination are included in T', we have that $L_1 = 1$ and $L_m = |T|$. Thus, the sampled trajectory T' can be represented as $\{p_{L_1}, \ldots, p_{L_m}\}$.

Recall that the trajectory contains both the temporal and spatial dependencies. To capture the spatio-temporal dependency, we use two stacked LSTM layers to process the GPS point sequence. A direct way is to take the longitude and the latitude of point p_{L_i} to the LSTM layers at each time step i. However, discovering the relative position and the distance between two GPS points is not easy for neural networks. Instead, for each time step i, we concatenate the coordinates of two consecutive points $p_{L_i}, p_{L_{i+1}}$, the travel distance between p_{L_i} and $p_{L_{i+1}}$ (which we denote as $\Delta d_{p_{L_i} \to p_{L_{i+1}}}$) and the output feature vector $attr$ obtained from the attribute component. We use such concatenation as the input of LSTM layers.

We pass the input sequence to two stacked LSTM layers and obtain the output sequence $\{y_1, y_2, \ldots, y_{m-1}\}$. Then, we use two stacked *time-distributed (fully-connected)* layers to map each y_i to a hidden state vector h_i. A time-distributed fully-connected layer takes a sequence of vectors and maps each vector to an output vector using the same mapping function. We finally concatenate the hidden state sequence $\{h_1, h_2, \ldots, h_{m-1}\}$ into a vector H, where H indicates the representation vector of the trajectory.

Pooling Trick

Alternatively, since the recurrent neural network can process the variable length sequences, we can also use the original trajectory T directly without sampling. In such implementation, instead of using the concatenate layer, we use a *mean pooling* layer as our merge layer, i.e., $H = \frac{1}{|T|-1} \sum_{i=1}^{|T|-1} h_i$ and obtain the representation vector in the same way. We compare these two different sequence processing implementations in the experiment part.

3.3 Residual Component

The residual component simply concatenates all the obtained feature vectors from sequence learning component and the attribute component. It thus forms a high-dimensional vector which indicates the representation of the trajectory and the related information (driver ID, the day of the week, etc.). Then, we use three residual layers to extract the feature vectors from this high-dimensional vector. Finally, we use a single neuron with linear activation to obtain the estimated travel time of the given path.

3.4 Auxiliary Component

In fact, the residual component is already available to estimate the travel time of the given path. However, it only considers the "global information"of the whole trajectory but ignores the travel time of sub-trajectories which we refer to as the "local information". Note that during the training phase, since the time stamps of all the GPS points $p.ts$ are available, we can easily infer the local information of the trajectory.

To utilize the local information, a feasible way is to estimate the travel time between each pair of consecutive GPS points and sum up the travel time of all pairs. However, on one hand, the travel time between the consecutive points is usually very small but has a large variance. Estimating such small values is difficult and leads to a very inaccurate result. On the other hand, since we only care the estimation accuracy of the whole trajectory, it is not necessary to accurately estimate the travel time of each individual pair.

Instead, we introduce an auxiliary component to make use of the local information, and we take the estimated travel time between the point pairs as the "auxiliary output". Formally, we denote the travel time of a sub-trajectory $p_a \to p_b$ as $\Delta t_{p_a \to p_b}$. For convenient, suppose we adopt the sampling trick in the sequence learning component. Based on the first three components, the auxiliary component receives the hidden state sequence $\{h_1, \ldots, h_{m-1}\}$ from the sequence learning component. Then it maps the hidden state sequence into a real value sequence $\{e_1, e_2, \ldots, e_{m-1}\}$ using a time-distributed fully-connected layer. Each real value e_i corresponds to the estimated travel time $\Delta t_{p_{L_i} \to p_{L_{i+1}}}$ where L is the indices of sampled points as we defined in Section 3.2. We use the estimated travel time sequence $\{e_1, e_2, \ldots, e_{m-1}\}$ as the auxiliary output of our neural network. The auxiliary output is trained together with the estimated travel time of the whole trajectory. See Section 3.5 for more details.

3.5 Model Training

Our model is trained end-to-end. We use the *mean absolute percentage error (MAPE)* as our objective function. Formally, suppose we adopt the sampling trick in the sequence learning component. Then, the loss function of sequence learning component is defined as

$$\text{loss}_{seq} = |e - \Delta t_{p_1 \to p_{L_m}}| / \Delta t_{p_1 \to p_{L_m}}.$$

For the auxiliary component, the loss function is the average MAPE loss of each time step, i.e.,

$$\text{loss}_{aux} = \frac{1}{m-1} \sum_{i=1}^{m-1} \frac{|e_i - \Delta t_{p_{L_i} \to p_{L_{i+1}}}|}{\Delta t_{p_{L_i} \to p_{L_{i+1}}} + \epsilon}. \quad (1)$$

Note that in Equ. (1) we use a small factor ϵ to prevent the exploded loss values when $\Delta t_{p_{L_i} \to p_{L_{i+1}}} \to 0$.

The loss function we used during the training phase is defined as the weighted sum of loss_{seq} and loss_{aux}, i.e.,

$$\text{loss} = \text{loss}_{seq} + \alpha \cdot \text{loss}_{aux} \quad (2)$$

where α is the weight factor which is specified in the experiment section. Note that the auxiliary loss is just used to improve the accuracy, we still use loss_{seq} to evaluate our model performance.

Thus, we can train our model by the standard backpropagation and gradient descent method.

Discussion

Learning to estimate the travel time of the whole trajectory and the travel time sequence of consecutive location pairs at the same time is one of the advantages of using deep learning model. Even though our original problem is to estimate the travel time of the whole path, we convert it to a *multi-task learning* problem, and optimize multiple objectives with a shared neural network architecture. This technique allows us to fully utilize the information contained in the training data. Multi-task learning has been widely studied in the computer visions and the natural language processing problems [Li *et al.*, 2015; Zhang *et al.*, 2014; Liu *et al.*, 2016]. It is unclear how to implement such multi-task learning using traditional machine learning techniques such as the random forest, or gradient boosting, in our setting.

4 Experiment

In this section, we report our experimental results on the real world dataset. We first describe the experimental setting and the details of dataset in Section 4.1. We then compare our model with several baseline methods in Section 4.2. In Section 4.3 to Section 4.5 we present the effects of different components and parameters.

4.1 Experiment Setting

Data Description

Our dataset consists of 1.4 billion GPS records of 14864 taxis from 2014/08/03 to 2014/08/30 in Chengdu, China[2]. Each record contains three attributes: the longitude, the latitude and the corresponding time stamp. We eliminate the trajectories during the mid-night (00:00 - 05:59) . The total number of trajectories after the elimination is $9,653,822$. The shortest trajectory contains only 11 GPS location points (2km) and the longest trajectory contains 128 GPS location points (41km).

We use the last 7 days (from 24th to 30th) as the test set and the remaining ones as the training set.

Parameter Setting

We present the parameter settings of different components.
Attribute Component: We embed $\mathrm{driverID}$ to $\mathbb{R}^{1\times 16}$, weekID to $\mathbb{R}^{1\times 3}$ and timeID to $\mathbb{R}^{1\times 4}$.
Sequence Learning Component: We set the dimension of each LSTM internal state as 128 and the dimensions of two time-distributed fully-connected layers as 128 and 64 respectively. We use leaky relu function [Glorot *et al.*, 2011] as the activation of the time-distributed fully-connected layers. Moreover, we test our model under different sampling rates (i.e., the number of sampled points) when we adopt the sampling trick. See Section 4.3 for more details.
Residual Component: We set the dimension of all three residual layers as 128. The activation functions of the residual layers are all leaky relu function.

Furthermore, in the auxiliary loss in Equ.(1), we set $\epsilon = 10$. In Equ. (2), we set the weight factor α as 3.0. We fix the batch size of our model as 512 and we adopt *Adam* [Kingma and Ba, 2014] optimizer with learning rate 0.001 to train our model. Our model is trained by 40 epochs.

To evaluate our model, we use 5-fold cross-validation in the training set. For each fold, we select the best model based on the validation. We thus obtain 5 best models. To estimate the travel time on the test set, we use each selected model to obtain an estimation respectively and average the estimations as our final result. The final estimation is evaluated by MAPE as we mentioned in Section 3.5.

Experiment Environment

Platform: Our model is trained on the server with one GeForce 1080 GPU. We implement our model with Keras 0.8.2 (Theano backend), a widely used Deep Learning Python library.

4.2 Performance Comparison

To demonstrate the strength of our model on estimating the travel time. We compare our model with several popular machine learning methods. Since part of the baseline methods cannot process the sequences with variable lengths, to make a fair comparison, we first sample each trajectory to a fixed length of 30. The methods are shown as follow:

Table 1: Performance Comparison of Baseline Methods

Model	MAPE
Gradient Boosting	20.32%
MLP-3 layers	16.17%
MLP-5 layers	15.75%
Vanilla RNN	18.85%
DeepTTE	$\mathbf{13.14}\%$

- **Gradient Boosting Decision Tree (GBDT)**: Gradient Boosting Decision Tree is a powerful and widely used ensemble method [Friedman *et al.*, 2001]. To estimate the travel time using GBDT, we concatenate the all the attributes contained in our attribute component and the input sequence in our sequence learning component. We use the concatenated vector as the input of GBDT. In our experiment, we use XGBoost, a widely used GBDT library [Chen and Guestrin, 2016]. The optimal parameters are achieved by the grid search.
- **Multi-Layer Perceptron (MLP)**: A multi-layer perceptron is a fully-connected neural network with multiple hidden layers. We test our data with a 3-layer MLP and a 5-layer MLP respectively. The input vector of MLP is the same as the input of GBDT. The dimension of each hidden layer is fixed as 128 and the activation is leaky relu.
- **Vanilla RNN**: We further compare our model with a vanilla RNN architecture. Each time step, the vanilla RNN receives the coordinates of p_{L_i} and $p_{L_{i+1}}$ as well as the corresponding travel distance between them. Similarly, we build an attribute component. We concatenate the output the attribute component and RNN, and we pass the concatenation to a 3-layer perceptron to obtain the estimated travel time.

We show the experiment result in Table 1. As we can see, the GBDT results in a large error of 20.32%. We stress that although GBDT is a powerful and widely used method, it can not capture the temporal dependency in the data. Furthermore, GBDT relies on carefully hand-crafted features. However, extracting useful features from the GPS point sequence is not easy. For vanilla RNN, it considers the temporal dependency between GPS location points but it failed to process the long sequence due to the gradient vanishing problem, as we mentioned in Section 1.

4.3 Effects on the Sampling Rate

We compare the performances of DeepTTE when we use different sampling rates (i.e., the length m of sampled trajectories as we defined in Section 3.2). We use x to denote the lengths of sampled trajectories and DeepTTE-x to denote the corresponding model. We further compare the performance of DeepTTE if we use the pooling trick, which we denote as DeepTTE-Var. The experiment result is shown in Table 2.

The result shows that enlarging the sampling rate increases the estimation accuracy of our model. When the sampling rate is 100, our model achieves the best performance of 12.74%. However, using large sampling rate also increases the training time. For DeepTTE-100, it takes about 1.5 hours to train a single epoch. Choosing a proper sampling rate is a trade-off between the speed and accuracy.

[2]The dataset and the corresponding code can be downloaded at `https://github.com/DeepTTE/DeepTTE`

Table 2: Performance of Different Sampling Capacity

Sampling Capacity	MAPE	Time (per epoch)
DeepTTE-10	15.45%	674s
DeepTTE-30	13.14%	1729s
DeepTTE-70	13.02%	3879s
DeepTTE-100	**12.74%**	5484s
DeepTTE-Var	12.87%	5841s

Table 3: Effects of Attribute Component

Experiment Setting	MAPE
DeepTTE-30	**13.14%**
Eliminate driverID	13.37%
Eliminate weekID	13.58%
Eliminate both	13.59%

DeepTTE-Var achieves the estimation accuracy of 12.85% which is slightly worse than DeepTTE-100. Although it utilizes all the information of the original trajectory T (recall that the max length of T is 128), it causes the information loss when we simply averaging the hidden state by the mean pooling layer.

4.4 Effects of the Auxiliary Component

Recall that in our model, we introduce an auxiliary component to estimate the travel time between consecutive GPS points. To verify the effectiveness of the auxiliary component, we eliminate the auxiliary component in DeepTTE-30 and train our model under the same condition. The experiment result shows that the estimation error increases from 13.14% to 13.95% dramatically.

Furthermore, if we directly predict the travel time between the consecutive GPS points and use the summation as our estimation, the MAPE is as high as 28.44%.

4.5 Effects of the Attribute Component

To show the effects of the attribute component (driver ID, day of the week), we compare the performance of DeepTTE-30 under three different settings. In the first setting, we eliminate the day of the week in the attribute component. In the second setting, we eliminate the driver ID. In the last setting, we eliminate both two attributes and only keep the start time and the travel distance in the attribute component. The experiment result is shown in Table 3.

The result shows that eliminating any attribute in our model leads to a reduction of estimation accuracy. As we can see, dropping out the day of the week increases the estimation loss dramatically. Moreover, if we eliminate the driver ID, the MAPE loss increases from 13.14% to 13.37% which verifies that the driving habits affects the travel time estimation, as we mentioned in Section 1.

5 Related work

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

5.1 Road Segment-Based Travel Time Estimation

Estimating travel time has been studied extensively [Zhong and Ghosh, 2001; Sevlian and Rajagopal, 2010; Pan *et al.*, 2012]. However, these works estimated the travel time of individual road segment without considering the correlations between the roads. [Yang *et al.*, 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.*, 2016b] 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 1, 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].

5.2 Path-Based Travel Time Estimation

[Rahmani *et al.*, 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 travelled records can not be estimated accurately. Furthermore, [Wang *et al.*, 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.

5.3 Deep Learning in Spatial Temporal Data

Recently, the deep learning techniques demonstrate the strength on spatio-temporal data mining problems. [Song *et al.*, 2016] built an intelligent system called DeepTransport, for simulating the human mobility and transportation mode at a citywide level. [Zhang *et al.*, 2016b] 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.

6 Conclusion

In this paper, we study estimating the travel time of a given path. We propose an end-to-end framework based on deep recurrent neural networks. Our model effectively captures the spatial and temporal dependencies in the given path at the same time. Furthermore, our model considers various factors which may affect the travel time such as the drivers, the day of the week etc. We conduct extensive experiments on a very large scale real-world dataset. The experimental result shows that our model achieve a high estimation accuracy and outperforms the other off-the-shell methods significantly.

References

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *arXiv preprint arXiv:1603.02754*, 2016.

[Dong *et al.*, 2016] Weishan Dong, Jian Li, Renjie Yao, Changsheng Li, Ting Yuan, and Lanjun Wang. Characterizing driving styles with deep learning. *CoRR*, abs/1607.03611, 2016.

[Friedman *et al.*, 2001] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[Glorot *et al.*, 2011] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. *Journal of Machine Learning Research*, 15, 2011.

[Graves *et al.*, 2013] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Computer Science*, 2015.

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

[Hu *et al.*, 2016] Huiqi Hu, Guoliang Li, Zhifeng Bao, Yan Cui, and Jianhua Feng. Crowdsourcing-based real-time urban traffic speed estimation: From trends to speeds. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 883–894. IEEE, 2016.

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

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

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Li *et al.*, 2015] Ya Li, Xinmei Tian, Tongliang Liu, and Dacheng Tao. Multi-task model and feature joint learning. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3643–3649. AAAI Press, 2015.

[Liu *et al.*, 2016] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2873–2879, 2016.

[Mesnil *et al.*, 2013] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*, pages 3771–3775, 2013.

[Pan *et al.*, 2012] Bei Pan, Ugur Demiryurek, and Cyrus Shahabi. Utilizing real-world transportation data for accurate traffic prediction. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 595–604. IEEE, 2012.

[Rahmani *et al.*, 2013] Mahmood Rahmani, Erik Jenelius, and Haris N Koutsopoulos. Route travel time estimation using low-frequency floating car data. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 2292–2297. IEEE, 2013.

[Sevlian and Rajagopal, 2010] Raffi Sevlian and Ram Rajagopal. Travel time estimation using floating car data. *arXiv preprint arXiv:1012.4249*, 2010.

[Song *et al.*, 2016] Xuan Song, Hiroshi Kanasugi, and Ryosuke Shibasaki. Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level. IJCAI, 2016.

[Wang *et al.*, 2014] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 25–34. ACM, 2014.

[Wang *et al.*, 2016a] Dong Wang, Wei Cao, Mengwen Xu, and Jian Li. Etcps: An effective and scalable traffic condition prediction system. In *International Conference on Database Systems for Advanced Applications*, pages 419–436. Springer, 2016.

[Wang *et al.*, 2016b] Jingyuan Wang, Qian Gu, Junjie Wu, Guannan Liu, and Zhang Xiong. Traffic speed prediction and congestion source exploration: A deep learning method. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 499–508. IEEE, 2016.

[Yang *et al.*, 2013] Bin Yang, Chenjuan Guo, and Christian S Jensen. Travel cost inference from sparse, spatio temporally correlated time series using markov models. *Proceedings of the VLDB Endowment*, 6(9):769–780, 2013.

[Yuan *et al.*, 2013] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. T-drive: enhancing driving directions with taxi drivers' intelligence. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):220–232, 2013.

[Zhang *et al.*, 2014] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision*, pages 94–108. Springer, 2014.

[Zhang *et al.*, 2016a] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. AAAI 2017, November 2016.

[Zhang *et al.*, 2016b] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. *arXiv preprint arXiv:1610.00081*, 2016.

[Zhong and Ghosh, 2001] Shi Zhong and Joydeep Ghosh. A new formulation of coupled hidden markov models. *Dept. Elect. Comput. Eng., Univ. Austin, Austin, TX, USA*, 2001.