Computer Modeling in Engineering & Sciences
DOI:10.32604/cmes.2020.010240
Article

Predicting Human Mobility via Long Short-Term Patterns

Jianwei Chen, Jianbo Li* and Ying Li

College of Computer Science and Technology, Qingdao University, Qingdao, 266071, China
*Corresponding Author: Jianbo Li. Email: lijianbo@188.com
Received: 20 February 2020; Accepted: 02 June 2020

Abstract: Predicting human mobility has great significance in Location based Social Network applications, while it is challenging due to the impact of historical mobility patterns and current trajectories. Among these challenges, historical patterns tend to be crucial in the prediction task. However, it is difficult to capture complex patterns from long historical trajectories. Motivated by recent success of Convolutional Neural Network (CNN)-based methods, we propose a Union ConvGRU (UCG) Net, which can capture long short-term patterns of historical trajectories and sequential impact of current trajectories. Specifically, we first incorporate historical trajectories into hidden states by a shared-weight layer, and then utilize a 1D CNN to capture short-term pattern of hidden states. Next, an average pooling method is involved to generate separated hidden states of historical trajectories, on which we use a Fully Connected (FC) layer to capture long-term pattern subsequently. Finally, we use a Recurrent Neural Net-work (RNN) to predict future trajectories by integrating current trajectories and long short-term patterns. Experiments demonstrate that UCG Net performs best in comparison with neural network-based methods.

Keywords: Human mobility; prediction; CNN; average pooling

1 Introduction

With the rapid development of social networks and location-based services, Location-Based Social Networks (LBSNs) such as Foursquare and Twitter are increasingly popular in our daily life. Many users write some short essays and share their footprints (we refer to such footprints as check-in points) on them, which enables some research efforts [1–4] in collecting users' trajectories and learning their mobility patterns. A critical task in understanding such patterns is the next location prediction, which has been extensively studied [5–7] in recent years. Moreover, this prediction task has great social significance in many applications, including route planning for taxi drives [8] and dispersing the crowd for public traffic [9]. Grapping human movements in advance will help such applications solve the above problems.

Currently, many methods have been extensively proposed for next location prediction, such as Matrix Factorization (MF) [10], Markov Chain (MC) [11] and MC-based methods (e.g., Factorizing Personalized Markov Chains (FPMC) [12] and Hidden Markov Models (HMMs) [13]). These works incorporate human visit preferences and investigate sequential transitions, which cannot obtain the ideal result due to few observations of historical visits. After that, many research efforts [14–16] focused on deep learning

neural network-based models. For example, RNN-based methods are commonly used methods. Based on historical trajectories, these works can capture the sequential regularities of users' mobility preferences. Specifically, they train a RNN module to extract mobility features (such as spatial and temporal features) and predict the next location.

Although the above methods have inspiring results on human mobility prediction, several key problems still exist: (1) Sparsity of data. On the one hand, some users generate only a few check-ins. On the other hand, observable features of users' historical visits are sparse. (2) Complex mobility patterns of historical trajectories. Some users' trajectories collected from LBSNs are usually very long, while existing works tend to be difficult to exploit whole trajectories effectively. (3) Heterogeneity of data. Each user's mobility pattern affects the future check-ins individually, whereas it is difficult to capture multi users' pattern at the same time. In summary, above problems make the prediction difficult.

To address the above problems, we propose a UCG Net to predict the next locations of users. We collect context-aware check-in points of each user and generate the whole trajectory for each one firstly, and then incorporate the whole trajectory into dense representation by the embedding technology, which solve the sparsity of data. Next, we divide the whole trajectory into historical trajectory and current trajectory. Subsequently, we introduce a historical trajectory learning module to capture long short-term pattern of the historical trajectory. Specifically, a CNN architecture is used to capture short-term pattern of trajectories' hidden states; an average pooling method and an FC layer are used to capture long-term pattern of separated hidden states; and then we concatenate two patterns into a vector where reflects long short-term pattern of the historical trajectory. Subsequently, we also introduce a current trajectory learning module where captures the sequential transitions of the current trajectory with the above long short-term pattern. Finally, an FC layer and Softmax layer are involved to predict user's next location. In summary, our main contributions are:

1. We propose a UCG Net to model trajectory and predict human mobility, which consists historical and current trajectory learning modules. Historical trajectory learning module uses CNN-based architecture to proceed historical trajectory sequences and capture long short-term patterns; Current trajectory learning module uses the RNN model to capture sequential transitions of current trajectory.
2. We use a $1 \times D$ CNN to capture the short-term pattern of historical trajectory, and then use a FC layer to capture the long-term pattern of sub-trajectories generated by the Average-Pooling method. Generated long short-term patterns are used as RNN's context, which makes UCG Net consider dual impact of historical and current trajectory.
3. We conduct experiments on real-world datasets consisting of different check-in behavior in two cities, demonstrating that our model improves the prediction performance compared with existing strong deep neural network-based models.

The rest of this paper is organized as follows. We first review the related works in Section 2, then formulate our problem and discuss the motivation of our work in Section 3. Following that, we detail the architecture of UCG Net in Section 4. After that, we introduce the experimental configuration and discuss the experimental results in Sections 5 and 6, respectively. Finally, we conclude our paper in Section 7.

2 Related Work

Human mobility prediction has drawn great attentions for decades, existing research efforts can be categorized into conventional and neural network-based methods. Conventional methods pay more attention on mobility pattern recognition and users' preferences exploration. For example, the popular Matrix Factorization (MF) [10] and MF-based methods [17,18] in POI recommendation learn users' preferences by generating a matrix with user location history. T-pattern [19] and Gapped Spatiotemporal-Periodic (GSTP)

[20] recognize the mobility patterns by mining spatiotemporal trajectory. These methods fail to capture the sequential information of trajectory. Therefore, they are not suitable to predict users' next visits. Afterwards, many research efforts are dedicated to solve this problem with Markov Chain (MC)-based methods [11–13,21]. Actually, MC-based methods can attribute to conventional methods through building different models. They learn sequential transition regularities via building a transition matrix with historical observations. For example, Mathew et al. [21] cluster locations from trajectories and train a Hidden Markov Model to capture unobserved characteristics. These methods are not efficient due to ignoring time dependence. Compared with above conventional methods, our model is able to capture the time dependence and complex transitions.

Recently, neural network-based methods [14–16,22–28] are widely used to predict human mobility. Especially RNNs such as Long Short-term Memory (LSTM) and Gated Recurrent Unit (GRU) are common methods. For example, Liu et al. [14] propose a Spatial Temporal RNN (ST-RNN) where models spatial and temporal contexts to predict next locations. Due to the lack of semantic information, Yao et al. [15] propose a semantic-aware recurrent model (SERM) where considers both sequential and semantic influence to predict future trajectory. However, these two methods cannot deeply capture the sequential information of historical trajectory. Therefore, Feng et al. [16] propose a DeepMove model to capture the multi-level periodicity of historical trajectory with an attentional mechanism based on a seq2seq model. Since it is complicated to train, Gao et al. [25] use a generative probabilistic model together with neural network and propose a VANext to improve this model more recently. They are the first to incorporate CNN to capture long term dependency among human trajectories, and improve the learning efficiency due to parallelization with GPU. Furthermore, they use a variational attention mechanism to learn the attention on the historical trajectories. Undoubtedly, these two attentional mechanism-based methods are efficient by matching current trajectory with long historical trajectory. These two methods mainly compute the contribution of each historical visit, and therefore lose the global sequential impact of historical trajectory. Compared with these neural network-based methods, our model predicts human mobility from global aspect and can learn long short-term mobility pattern of the historical trajectory.

CNNs are not popular for modeling sequence task, but they have received more and more attention on this problem. For example, Kim et al. [29] use a CNN architecture to capture local correlation of sentences and apply it on a text classification problem effectively. Recently, a CNN-based model [30] is proposed to improve machine translation performance. Two of these methods have achieved good performance in the Natural Language Process (NLP) problem. Similarly, some research efforts [31,32] use the CNN architecture to capture mobility patterns of human trajectory. Except for above VANext model, Wang et al. [31] propose a CNN-based Geo-Conv layer and learn feature maps of mobility trajectory. Compared with RNN-based models, CNNs is capable of capturing hierarchical representation of the underlying context with lower time complexity. This paper uses a CNN architecture to capture complex relation information-long-short term pattern of users' historical trajectory.

3 Preliminaries

In this section, we first present several basic definitions, and then formally formulate the mobility prediction problem. Finally, we discuss the motivation and overview our solution.

3.1 Problem Formulation

Context-aware spatiotemporal point p is a tuple of location identification l, timestamp t and activity identification a, i.e., $p = <l, a, t>$. Given a user u, p indicates u visited l for activity a at time t. Then, let $T^u = \{p_1^u, p_2^u, p_3^u, \ldots, p_n^u\}$ denote a time-order trajectory generated by u, where n is the length of T^u. For convenience, we will omit the superscript ^{u}. Next, a trajectory T can be segmented as

$T = (T^1, T^2, \ldots, T^m)$, meaning that there are m sub-trajectories ordered along the temporal dimension. Specifically, we predefine a threshold δ and generate these sub-trajectories, which means that the time span of each T^i is no more than δ.

Let $T^h = (T^1, T^2, \ldots, T^m)$ denote the historical trajectory (concatenating all sub-trajectories) of the user u, and let $T^c = (p_{n+1}, p_{n+2}, \ldots, p_{n+\xi-1})$ denote the current trajectory. With these notations, our problem can then be formulated as: our goal is to predict u's next point $p_{n+\xi}$, given the T^h and the T^c. Note that we also quantify the time interval of each p into a fixed value as in previous work [16], and a is only the supplement of location information. Thus, we actually predict the next l in $p_{n+\xi}$.

3.2 Motivation and Overview

We briefly describe our problem of as a time-series problem. RNN is available to solve our problem since it is a powerful time-series modeling pool of capturing long-range dependencies of sequential information. However, the use of a single RNN is inefficient to predict human mobility using a single RNN. There are two reasons supporting our view.

First, the length of each user's historical trajectory tends to be long. Unfortunately, RNN can't proceed with long sequence directly. Even though using LSTM, it also has poor performance due to the vanishing gradient. As in previous work [16], it has been verified. Secondly, the sparsity of data cannot be ignored, mainly reflected in two aspects. For one thing, collected data tend to have less features (e.g., coordinate and time), which makes the RNN difficult to train; for another, the time interval of each record is usually unfixed, causing wrong sequential information of each user. In other words, it induces the problem of fragmentated trajectory.

Based on above observations, this paper proposes a novel neural network-based method to predict human mobility. Specifically, we first segment entire trajectory of each user and limit the length of sub-trajectory. Next, we use an embedding layer to represent sparse features (e.g., l, a and t) and solve the problem of sparsity. We then concatenate such sub-trajectories to generate the historical trajectory. Subsequently, a CNN architecture is used to capture long short-term mobility pattern of the historical trajectory. The CNN architecture can easily represent the local relationships and long-range relationships with pooling-based methods [33]. Finally, we predict future trajectory with the RNN by generated pattern and the current trajectory.

4 Methodology

In this section, we first introduce our basic framework of prediction model. Subsequently, we detail some important modules of UCG Net. Finally, we interpret our model again and introduce the parameter learning of one. For convenience, the key mathematical notations used in the methodology are shown in Tab. 1.

4.1 Basic Framework

Fig. 1 shows the framework of UCG Net. It mainly consists of four components: trajectory embedding, current trajectory learning module, historical trajectory learning module and classifier.

1. Trajectory Embedding: Before the training process, we first incorporate the features from both T^h and T^c, which can not only present the semantic relationships among different trajectories but also benefit the follow-up computation.
2. Historical Trajectory Learning Module: This module is involved to capture the mobility pattern of T^h. We first use a Multi-Layer Perceptron (MLP) to incorporate the hidden states of T^h, and encode them with a convolutional layer, which can capture the short-term pattern of trajectory. Then, we utilize a FC layer to

capture long-term pattern of separated hidden states. Finally, we generate long short-term patterns with Max Pooling layer and concatenate them as the inputs of current trajectory learning Module.

3. Current Trajectory Learning Module: In this module, we involve a RNN architecture to capture the sequential transitions of T^c. It receives the long short-term patterns and T^c as inputs, and represent the future trajectory with the outputs of it.
4. Classifier: This module is the final component which incorporates the outputs of current trajectory learning module into a feature representation, Softmax layer is used to generate location probability and get predicted future trajectory.

Table 1: Key mathematical notation

Variable	Interpretation
p	Context-aware spatiotemporal trajectory point
l, a, t	The factors of p mean location identification, activity identification and time, respectively.
T^h, T^c	The historical trajectory and current trajectory of user u.
δ, ξ	Predefined threshold of time span and the length of each sub-trajectory.
m, n	The numbers of sub-trajectories and the length of the historical trajectory.
γ, h_τ	The former is the representation of embedded p, the latter is the hidden state of GRU architecture in time τ.
H^{his}, H^{cur}	The size of hidden layer in historical and current trajectory learning modules.
v^{short}, v^{long}	The representation vectors of short-term pattern and long-term pattern.
c, c^{conv}, c^{avg}	The first is the hidden states of the historical trajectory, follow-up two variables are convolutional c and representation result by using the average pooling method, respectively.
W_*, U_*, b_*	The estimated parameter matrices and bias term.

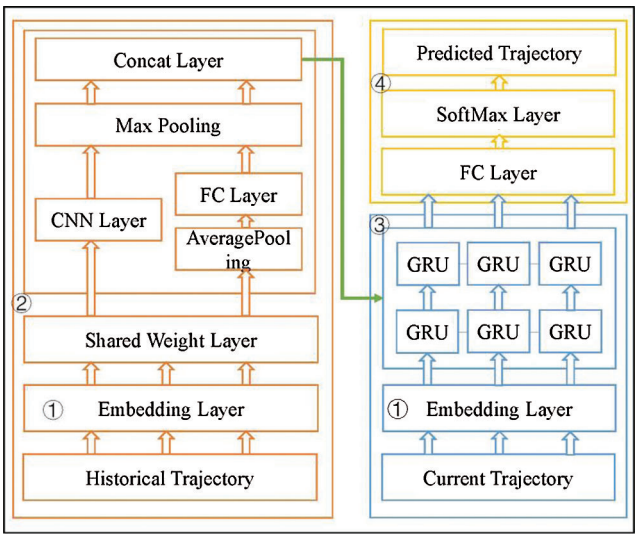

Figure 1: Main Components of UCG Net

4.2 Trajectory Embedding

As we mentioned, the sparsity of data's features is the key problem in prediction tasks. Therefore, we collect some related factors (such as location, activity, timestamp) to denote each record p. As in previous work [15,31], the embedding layer [34] can capture the information of the dynamic factors in each p. Compared with the one-hot encoding, the embedding layer effectively reduces the input dimension and is easily computed. Moreover, the embedding method helps find and share similar patterns among different trajectories. Therefore, we use the embedding layer to incorporate such factors of each p. As shown in Fig. 2, we learn the embeddings for each factor, then we concatenate them into a vector represented by γ. The embedding methods of each factor are as follows:

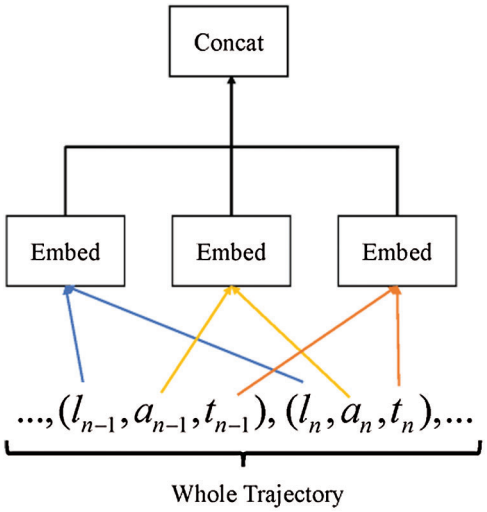

Figure 2: Trajectory embedding module

Timestamp Embedding. The original time information of each p cannot feed to the neural network directly. We reference the previous work and map one week into 48 slots (24 slots for weekdays and 24 slots for weekends). Then, each hour of timestamp is considered as an embedding unit. For any t in each p, the embedding method maps t to a real space (we refer to such space as embedding space) $\mathbb{R}^{d_t}$ by multiplying a parameter matrix $W_t \in \mathbb{R}^{48\times d_t}$.

Location and Activity Embedding. The original location and activity information of each p both cannot feed to the neural network similarly. Their identification l and a are both categorical values, we use the embedding layer to represent them respectively. For any l, each categorical value is mapped into the embedding space $\mathbb{R}^{d_l}$ by multiplying a parameter matrix $W_l \in \mathbb{R}^{L\times d_l}$. Similarly, each a is mapped into the embedding space $\mathbb{R}^{d_a}$ by multiplying a parameter matrix $W_a \in \mathbb{R}^{A\times d_a}$. L and A represent the vocabulary size of the categorical value l and a respectively, d_l and d_a are dimensions of their embedding space.

As we mentioned above, we concatenate each embedding of such factors. Therefore, final output vector of the embedding layer $\gamma(p_i) \in \mathbb{R}^d$ where $d = d_l + d_a + d_t$.

4.3 Historical Trajectory Learning Module

This module presents the historical trajectory learning method. As mentioned above, T^h has great impact on human mobility and helps predict future trajectory.

In this module, the inputs are embedded $T^h \in \mathbb{R}^{n \times d}$. We first use a shared weight layer (e.g., the MLP) to incorporate each point representation vector into a hidden state.

$$c_i = \text{ReLU}(W_H \gamma(p_i) + b_H), i \in [1, 2, \ldots, n] \tag{1}$$

where ReLU is the activation function, the parameter matrix $W_H \in \mathbb{R}^{H^{his} \times d}$. H^{his} is the numbers of neurons in this layer. Then, we obtain all hidden states of T^h. They can be expressed as $c = (c_1, c_2, \ldots., c_n)$, and $c \in \mathbb{R}^{n \times H^{his}}$. Based above operation, we can obtain weighted T^h.

Short-term pattern: As mentioned above, we get the hidden states c of T^h before the convolution operation. Note that c can be seen as a H^{his}-channel input with the length n. Then, we use single layer $1 \times D$ convolution to capture the patterns of c. The convolution filter is parameterized with matrix $W_c \in \mathbb{R}^{k \times H^{his}}$. k is the kernel size of the filter. The filter applies a convolution operation on the c, along with a one-dimension sliding window. Therefore, we can obtain convolutional feature map c_i^{conv} for each c_i. Its calculation formula can be denoted as,

$$c_i^{conv} = W_c * c_{i:i+k-1} + b_c \tag{2}$$

where $*$ denotes the convolution operation, $c_{i:i+k-1}$ is the subsequence in the c from index i to $i + k - 1$. Actually, k is set as 2. We are more interested in first-order transitions of T^h. Therefore, we set k to 2 and obtain short-term convolutional states of c. Furthermore, the out channels of $1 \times D$ convolution are same as input channels, i.e., H^{his}. Thus, we obtain all convolution states $c^{conv} = (c_1^{conv}, c_2^{conv}, \ldots, c_{n-1}^{conv})$ with a shape $[(n-1) \times H^{his}]$.

As the Fig. 3 shows, we use a max-pooling method to capture the short-term pattern of convolutional states c^{conv}. We are interested in using the max value of these convolutional states to represent the short-term pattern, the max value can effectively express the contribution of each c_i^{conv}. Specifically, we represent the short-term pattern with a vector v^{short}, where each dimension is the max value of each out channel over the c^{conv}. Thus, $v^{short} \in \mathbb{R}^{H^{his}}$.

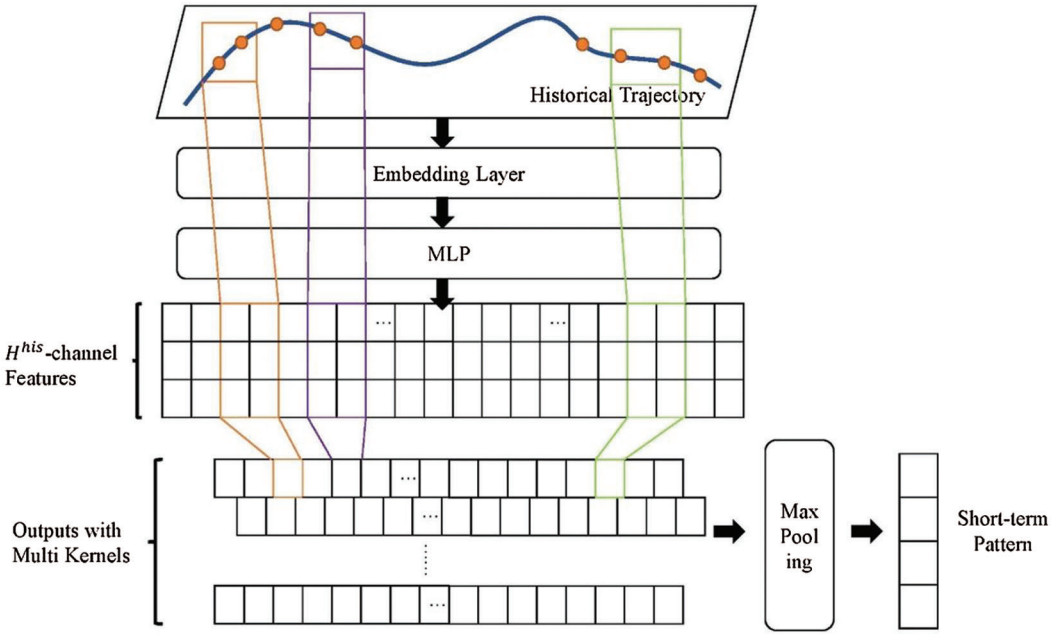

Figure 3: Component of capturing short-term pattern

Long-term pattern: It is incomplete to only capture the short-term pattern, which cannot represent entire mobility pattern of T^h. Not only that, it will influence the effect of prediction model.

Fig. 4 shows the method of capturing the long-term pattern. We first obtain the hidden states c of T^h by the MLP. Actually, T^h is a trajectory sequence of length n with m sub-trajectories. Thus, c contain m hidden states of these sub-trajectories. Subsequently, we use an average pooling method and obtain the representation of the sub-trajectories $c^{avg} \in \mathbb{R}^{m \times H^{his}}$. Note that we fix the length ξ of each sub-trajectory. Moreover, we set the kernel size of the average pooling method to ξ, the stride of sliding window is same as ξ. Hence, $c^{avg} = (c_1^{avg}, c_2^{avg}, \ldots, c_m^{avg})$. Next, a FC layer is used to capture the correlation between these sub-trajectories. This FC layer is parameterized with a matrix $W_F \in \mathbb{R}^{H^{his} \times H^{his}}$. Finally, the shared max pooling method is similarly used to represent the long-term pattern $v^{long} \in \mathbb{R}^{H^{his}}$. To avoid repetition, it is not described here.

Based above two pattern learning methods, we obtain the long-term and short-term pattern. Then, we concatenate two mobility pattern and generate the historical long short-term pattern vector $v \in \mathbb{R}^{2 \times H^{his}}$. Finally, v is used in the current learning module and set as the initial value of h in Eq. (3).

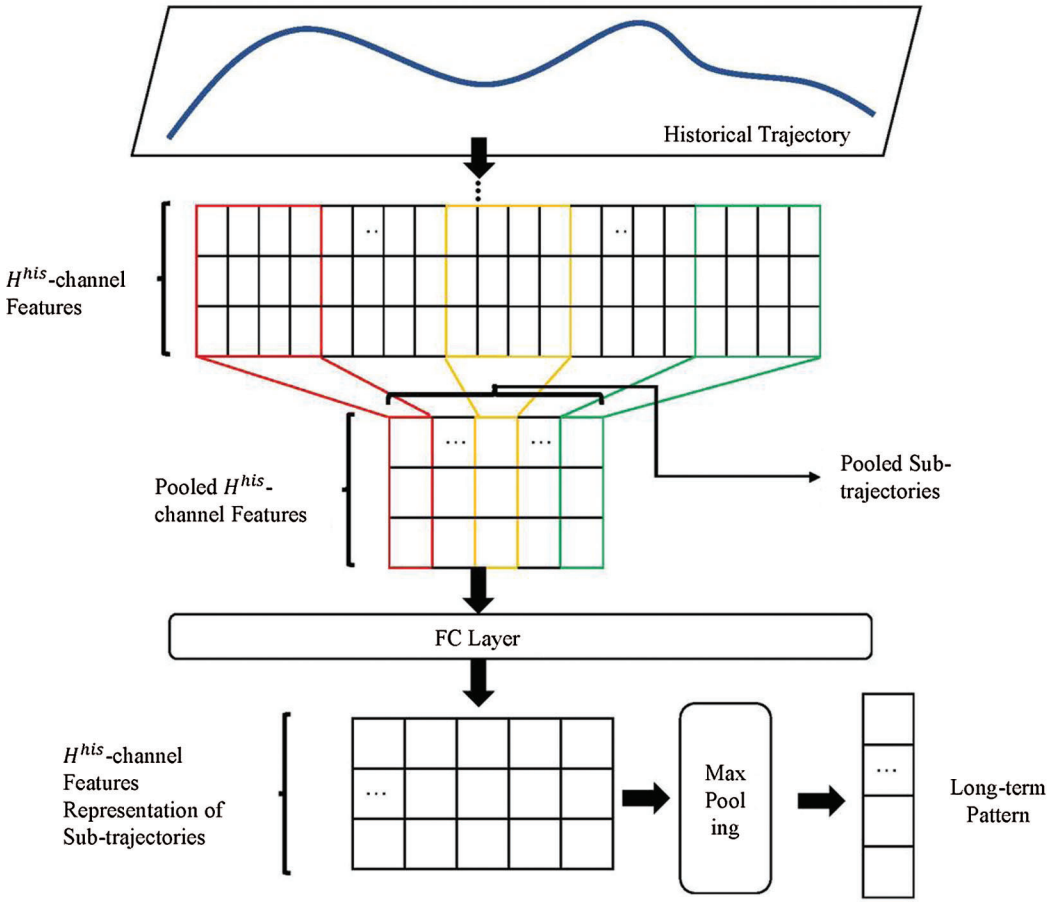

Figure 4: Component of capturing long-term pattern

4.4 Current Trajectory Learning Module and Classifier

This module introduces a GRU architecture and use it to capture sequential information of the T^c. To improve the readability of paper, we detail the classifier at the end of this module.

As mentioned above, the length of T^c is far less than the historical trajectories' length n. Moreover, RNN has powerful capabilities of capturing sequential pattern with short inputs due to its chain structure. Actually, users' mobility tends to have a strong correlation of previous footprint, which is similar to the high-order Markov Process. In other words, it is important to capture the sequential transitions of T^c. Therefore, we involve an RNN architecture to capture the sequential pattern of T^c. Here, we use the variant GRU of RNN proposed by [35]. The calculation formula of this unit is as follows:

$$h_\tau = (1 - z_\tau)h_{\tau-1} + z_\tau \tilde{h}_\tau \tag{3}$$

$$z_\tau = \sigma(W_z\gamma(p_\tau) + U_z h_{\tau-1} + b_z) \tag{4}$$

$$\tilde{h}_\tau = \tanh(W_{\tilde{h}}\gamma(p_\tau) + U_{\tilde{h}}(r_\tau \odot h_{\tau-1}) + b_{\tilde{h}}) \tag{5}$$

$$r_\tau = \sigma(W_r\gamma(p_\tau) + U_r h_{\tau-1} + b_r) \tag{6}$$

where Eq. (3) is the final output of this unit, z_τ is the update gate in time τ and decides how much the unit updates by the sigmoid activation function in Eq. (4). In Eq. (5), the candidate state $\tilde{h}_\tau$ memorizes the current state in time τ selectivity, computed similarly to traditional RNN unit. r_τ denotes the reset gate and decides how much the unit forgets previous information. W and U are both parameter matrices, b is the bias term. Benefited from unique gating mechanism, Eq. (3) can keep the historical information and updates current information at each time step. Thus, the GRU architecture can capture the sequential information for time-series data.

In this module, we set two GRU architecture. We first feed the embedded current trajectory $\gamma(T^c) \in \mathbb{R}^{(\xi-1)\times d}$ and pattern vector $v \in \mathbb{R}^{2\times H^{his}}$ into the GRU architecture. Then, we capture the sequential information by h's iteration over temporal dimension in Eq. (3). The hidden state h contains previous information and is continuously conducted as a new input on the follow iteration. The initial value of h will be introduced in the next section. Finally, the outputs of this architecture are the representation of future trajectory $o_{T^c} \in \mathbb{R}^{(\xi-1)\times H^{cur}}$ and then fed into the classifier to generate predicted trajectory.

Note that $o_{T^c} \in \mathbb{R}^{(\xi-1)\times H^{cur}}$ can be expressed as $o_{T^c} = (o_{n+2}, o_{n+3}, \ldots, o_{n+\xi})$, and H^{cur} is the hidden size of GRU kernel. To predict the next location of T^c, we add a FC layer and incorporate o_{T^c} into the high-level representation matrix ϕ of future trajectory. The ϕ can be denoted as:

$$\phi = W_\phi o_{T^c} + b_\phi \tag{7}$$

where $\phi \in \mathbb{R}^{(\xi-1)\times L}$ is parameterized with a matrix $W_\phi \in \mathbb{R}^{L\times H^{cur}}$. Then, the Softmax layer is constructed to generate the probability scores of future trajectories.

4.5 Model Interpretation and Parameter Learning

To understand our model more clearly, we interpret our model here. Although we cannot capture the sequential information of T^h, we replace the RNN with the $1 \times D$ convolution. The convolutional states c^{conv} contain the transition information of each trajectory point-pair. Actually, each state of c^{conv} is the representation of transition between two points. Then, we obtain the representation vectors of each sub-trajectory by the average pooling layer. These vectors contain the information for each stage of entire T^h. Then the FC layer can help us capture the correlation of these vectors. Next, we generate the long short-term pattern of T^h through using a shared max pooling method. Subsequently, obtained long short-term pattern is the initial input for the current learning module. Finally, we predict users' next location by learning the mobility pattern of T^h and sequential transitions of T^c.

Based on the prediction model described above, we generate a probability distribution of the predicted trajectory over all the location with the GRU kernel. We can easily generate the probability distribution by the Softmax layer added in the final module. To train the model, we use negative log-like as the loss function for our model. In our training set, it contains multi-user trajectories. Therefore, we choose average loss of all users as our objective function. Given a training set, the objective function can be defined as:

$$\ell = -\frac{1}{|u|}\frac{1}{|Z_u|}\sum_{|u|}\sum_{|Z_u|}\sum_{j=1}^{\xi-1}\log y_j[l_{j+1}] + \frac{\lambda}{2}||\Theta||^2 \tag{8}$$

where Z_u denotes the samples of training sets for u, ξ is the length of each sample. y_j and l_{j+1} are the predicted probability vector and the categorical value of target location at the j^{th} step. Θ contains all the parameters to be learned, including W_*,U_*,b_*. To avoid overfitting, we add an L2 regularization term and predefine the value of λ. Moreover, we use Adam [36] and the Back-Propagation Through Time algorithm to optimize the objective function.

5 Experimental Configuration

5.1 Data Information

We conduct our experiment on publicly available datasets of Foursquare collected from different cities, New York (NYC) and Tokyo (TKY). Foursquare data [37] includes long-term (about 10 months) check-in data from 12 April 2012 to 16 February 2013. NYC and TKY are the most prosperous areas in the world, and the most populous cities in USA and Japan respectively. Therefore, it is valuable to study human mobility using the check-ins datasets of two cities. Furthermore, two datasets have different check-in patterns influenced by cultural differences between the East and the West, which can verify the robustness of our model. The overall statistics of two datasets are shown in Tab. 2. Note that we take the categories of POIs (e.g., restaurant, gym and etc.) as activity keywords. For each dataset, we set the time span δ of each sub-trajectory as three days, and then set the time interval of each p is not less than ten minutes. Next, we set the length ξ of each sub-trajectory as 10 and remove users who have sub-trajectories fewer than 5. Finally, we take 80% datasets of each user as the training set, and the 20% datasets as the testing set for the parameters study.

Table 2: Statistics of two datasets

City	#Users	#Check-ins	#Categories of POIs
NYC	1083	227420	376
TKY	2293	573703	338

5.2 Evaluation Metrics

To make fair comparisons, we use the standard evaluation performance metrics, such as Top@k. Recall@k, Precision@k and F1-score@k. Specifically, we present each user with k of locations sorted by the predicted score using the classifier, i.e., S_u^k. Given a top-k predicted location list S_u^k and target location list l_u^* of test set, each evaluation metric can be defined as:

$$Top@k = \frac{1}{|u|}\sum_{|u|}\sum_{j}^{|l_u^*|}\frac{|l_{u,j}^* \cap S_{u,j}^k|}{|l_u^*|} \tag{9}$$

where j denotes each sample in test sets. Then Recall@k and Precision@k are defined as:

$$Recall@k = \frac{1}{|u|}\sum_{|u|}\sum_{j}^{|l_u^*|}\frac{|S_{u,j}^k \cap S_{u,j}^{visited}|}{|S_{u,j}^{visited}|} \tag{10}$$

$$Precision@k = \frac{1}{|u|}\sum_{|u|}\sum_{j}^{|l_u^*|}\frac{|S_{u,j}^k \cap S_{u,j}^{visited}|}{k} \tag{11}$$

where $S_{u,j}^{visited}$ is the locations list of u has visited in the test set. Note that above metrics are computed by averaging of all the values of all the users. Finally, the F1-score is the harmonic mean of Recall and Precision:

$$F1\text{-}score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{12}$$

5.3 Comparative Methods

We compare our model with the following five baseline approaches:

1. Markov Chain (MC): We use the outperforming first-order MC method.
2. RNN: We choose the GRU cell as its basic architecture. Actually, the performance of LSTM is close to the GRU.
3. ST-RNN [14]: ST-RNN is a RNN model taking spatial-temporal context as inputs for location prediction.
4. SERM [15]: SERM is a LSTM model that incorporates spatiotemporal context and activity keywords and predicts user's next visit.
5. DeepMove [16]: DeepMove is a strong neural network-based method based attentional mechanism, which capture historical periodic from historical trajectory and complex sequential transitions from recent trajectory.

5.4 Parameter Settings

In our experiment, the weights of our model are initialized with Xavier initialization. The initial learning rate is set to 1e-3, the embedding spaces d_t and d_a are set to 20 and 50 respectively. As the vocabulary sizes of t and a are much smaller, their embedding spaces are set to a practically small value. The embedding of space d_l is set to the optimal hidden size. Note that we set both hidden size of historical trajectory module H^{his} and hidden size of current trajectory module H^{cur} as a same value for convenient calculation. The setting of hidden size is discussed in the next section. For two datasets, the detail settings of our experiment are shown in Tab. 3.

Table 3: Parameters settings on two datasets

Parameter	Dataset	
	NYC	TKY
H^{his}, H^{cur}	400	900
Learning rate	1e-3	1e-4
L2 regularization λ	1e-6	1e-7
Gradient clip	6	10
Dropout	0.7	0.5

Our model is implemented with Pytorch 0.4.1, an open source Deep Learning Python Library developed by Facebook. We train and evaluate our model on the server with a Nvidia GeForce GTX 1080Ti GPU.

6 Experimental Results and Discussion

In this section, we compare our model with several strong methods. We first discuss the Top@k results of two datasets, and then discuss the results of others evaluation metrics on two datasets. Moreover, we introduce the choice of experimental parameters.

6.1 Performance Comparison

Actually, Top@k is the most important metric among all evaluation metrics. We can directly observe the next location of users by Top@k results, we therefore first discuss this metric. Note that we set k as 1 and 5 because of a greater value is usually be ignored in the reality. For the NYC dataset, we can observe that our model achieves the best results among all the baseline methods from the Tab. 4. Markov method builds the transition matrix according to the last location of each user, so that it is not surprising that Markov gets the worst performance. Then, the RNN model has better performance compared with the Markov model due to its strong sequence modeling ability. ST-RNN and SERM both exhibit higher results than the RNN model due to contextual features (e.g., spatial and temporal context). However, they cannot handle the complex transition regularities well with long trajectory, which leads to that the results are not significant compared with the RNN model. DeepMove has the best result among all the baseline methods due to its ability to exploit historical trajectories with attentional mechanisms. Compared with DeepMove, our model shows an increase of 11% and 20% in the Top@1 and Top@5 results, respectively. For the TKY dataset, we can observe similarly results compared with the NYC dataset. Since we have discussed the results as above description, so that we don't explore them here. Similarly, our model shows an increase of 10% and 19% in the Top@1 and Top@5 results, respectively. Moreover, we can observe that TKY dataset has more users compared with NYC dataset, which makes model train more difficult, so that all the Top@k results on TKY dataset perform worse than the results on NYC dataset.

Table 4: Top@k results on two datasets

Method	NYC		TKY	
	Top@1	Top@5	Top@1	Top@5
Markov	0.132	0.269	0.125	0.244
RNN	0.151	0.318	0.134	0.273
STRNN	0.162	0.345	0.142	0.303
SERM	0.170	0.396	0.144	0.298
DeepMove	0.195	0.378	0.168	0.346
Our model	0.218	0.456	0.186	0.384

As a classifier, we compare the classification performance with all the baseline methods. Therefore, we choose recall, precision and F1-score as others metrics. As shown in Tabs. 5 and 6, we can observe that our model gets the best performance compared with all the baseline methods. For the Recall@k results, our model shows an increase of 12% and 16% in Recall@1 and Recall@5 on NYC dataset, respectively. Besides, our model shows an increase of 11% and 6% in Recall@1 and Recall@5 on TKY dataset, respectively. Recall@k results on two datasets demonstrate that our model has better performance to classify true locations. For the Precision@k results, our model shows an increase of 12% and 31% in

Table 5: Performance of recall, precision and F1-score comparison on NYC

Method	NYC					
	Recall@1	Recall@5	Precision@1	Precision@5	F1@1	F1@5
Markov	0.132	0.233	0.132	0.282	0.132	0.255
RNN	0.151	0.276	0.151	0.322	0.151	0.297
STRNN	0.162	0.297	0.162	0.362	0.162	0.326
SERM	0.170	0.318	0.170	0.425	0.170	0.364
DeepMove	0.195	0.299	0.195	0.348	0.195	0.322
Our model	0.218	0.348	0.218	0.455	0.218	0.395

Table 6: Performance of recall, precision and F1-score comparison on TKY

Method	TKY					
	Recall@1	Recall@5	Precision@1	Precision@5	F1@1	F1@5
Markov	0.125	0.185	0.125	0.270	0.125	0.221
RNN	0.134	0.201	0.134	0.322	0.134	0.248
STRNN	0.142	0.236	0.142	0.349	0.142	0.282
SERM	0.144	0.228	0.144	0.341	0.144	0.273
DeepMove	0.168	0.254	0.168	0.395	0.168	0.309
Our model	0.186	0.274	0.186	0.401	0.186	0.326

Precision@1 and Precison@5 on NYC dataset, respectively. Besides, our model shows an increase of 11% and 2% in Precison@1 and Precision@5 on TKY dataset, respectively. Precision@k results on two datasets demonstrate that our model predicts more true locations. For the harmonic F1-score results, our model shows an increase of 11% and 22% in F1-score@1 and F1-score@5 on NYC dataset, respectively. Similarly, our model shows an increase of 10% and 6% in F1-score@1 and F1-score@5 on the TKY dataset, respectively. Above results demonstrate that our model has satisfactory classification performance. In addition, F1-score@k results on two datasets are similar to Top@k results. Except for more users of TKY dataset, the reason lies in the fact that average check-ins of two datasets are relatively same. Therefore, it is reasonable for the worse performance of our model on TKY dataset than NYC dataset.

In summary, our model obtains the best performance among all the strong neural network-based methods. The improvements on such evaluation metrics can be ascribed to the following reasons. Firstly, we incorporate more contextual features of whole trajectories, which makes our model learn more knowledge. Secondly, we involve CNN-based methods to capture long short-term pattern effectively. Different from DeepMove used attentional mechanism, our model mainly captures mobility pattern of sequential sub-trajectory rather than contribution of separated trajectory points. Thirdly, we use the RNN architecture to capture the sequential transitions of current trajectories together with mobility pattern of historical trajectories. As described above, our model can capture complex mobility pattern of whole trajectories effectively.

6.2 Validation of Capturing Long Short-Term Patterns

Here, we validate the ability of capturing long short-term patterns of our model. To validate the ability of our model, we set three different variants, including RNN, UCG-Short and UCG. UCG-Short is our model without capturing long-term. we choose the same parameters of UCG for UCG-Short. In this experiment, we compare the Top@k results of three models to verify the long short-term ability. Top@k results can show the prediction performance intuitively, the experimental results are shown in Fig. 5.

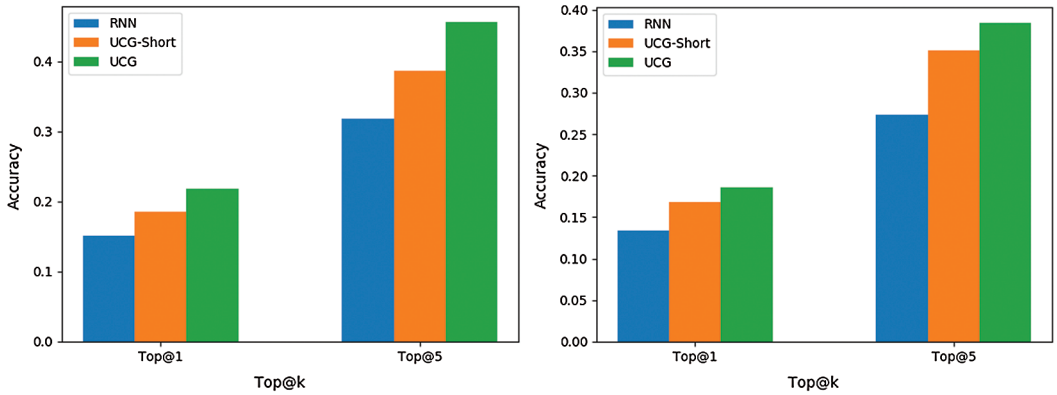

Figure 5: The results of RNN, UCG-short and UCG on two datasets. (a) NYC and (b) TKY

From the results on two datasets, we observe that UCG Net has the best performance. By the comparison of RNN and UCG-Short, the results demonstrate that our model has the ability of capturing short-term patterns. By the comparison of UCG-Short and UCG, the results demonstrate that our model has the ability of capturing long-term patterns. Actually, our model has the best performance due to that our model can capture the long-term patterns.

6.3 Parameter Sensitivity

In this experiment, we investigate the impact of embedding size and the dimension of hidden layer on two datasets. Here, the embedding sizes of activity and time are relatively small, which the impact of their sizes is not significant. Therefore, we mainly discuss the choice of location's embedding size. Fig. 6 shows the influence of embedding size on two datasets, and Tab. 7 depicts the changes of F1-score@k. We can observe that the change of both two metrics is not significant, which denotes the robustness of our model. Here, we can obtain the optimal value when choose 400 and 900 as the embedding size. Therefore, we choose them as the optimal embedding size of two datasets.

Table 7: Impact of embedding size on the performance of F1-score@k

Metric	NYC				TKY			
	100	300	400	600	300	600	900	1100
F1@1	0.214	0.214	0.218	0.216	0.185	0.184	0.186	0.181
F1@5	0.378	0.391	0.395	0.396	0.328	0.319	0.326	0.311

By setting the embedding size as optimal value, we further discuss the impact of dimension of hidden layer on two datasets. Fig. 7 shows the influence of hidden layer' dimension, and Tab. 8 depicts the changes of F1-score@k results. We can observe that the performance of our model is getting better with the increasing

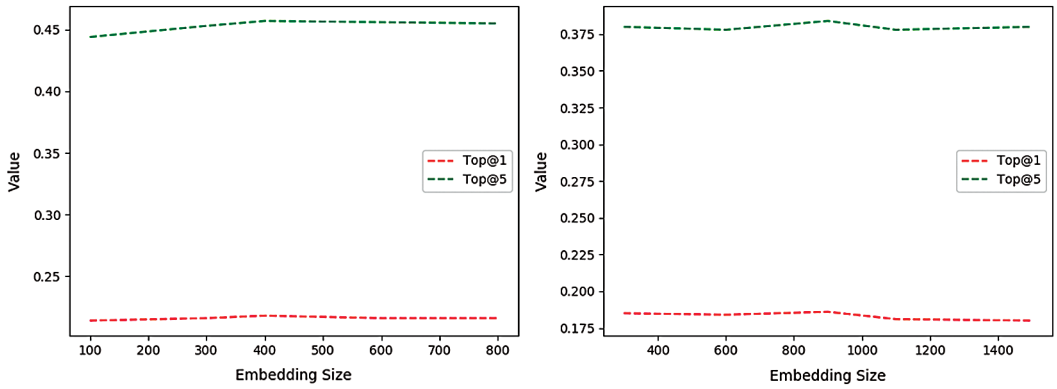

Figure 6: The impact of embedding size on two datasets. (a) NYC and (b) TKY

Table 8: Impact of hidden size on the performance of F1-score@k

Metric	NYC				TKY			
	100	300	400	600	300	600	900	1100
F1@1	0.201	0.210	0.218	0.209	0.172	0.176	0.186	0.179
F1@5	0.356	0.377	0.395	0.371	0.290	0.305	0.326	0.308

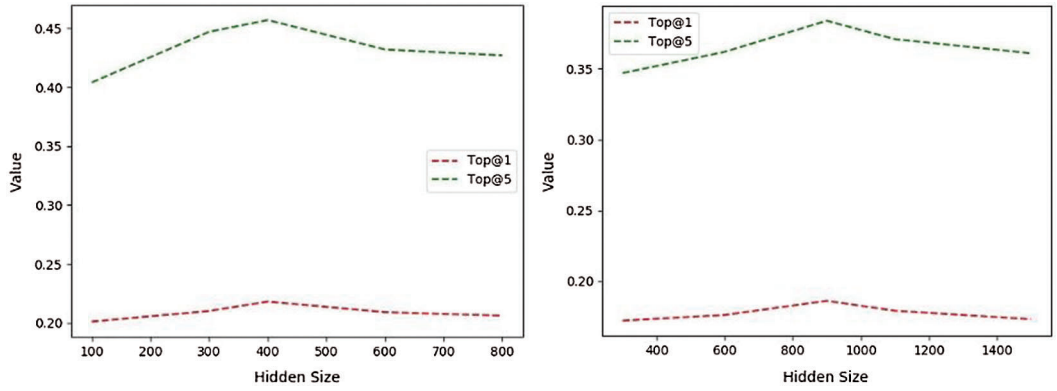

Figure 7: The impact of hidden size on two datasets. (a) NYC and (b) TKY

of hidden size, which denotes that the hidden size influences our model significantly. However, the performances on two datasets are both worse when the hidden size is too large, which means that overfitting problem may occur. Based above observation, we choose 400 and 900 as the hidden size of our model on NYC and TKY datasets, respectively.

7 Conclusions

In this paper, we propose a neural network-based method to capture the mobility pattern of human trajectory. We incorporate user's each trajectory point into a context-aware representation with multiple factors, i.e., location, activity and temporal information. Then, the embedding layer help capture semantic information among these trajectories. Furthermore, our model can capture long short-term pattern using CNN-based method. Specifically, we capture short-term pattern via $1 \times D$ convolution and long-term pattern through the average pooling method. The experiment on two real world datasets prove the

effectiveness of our model. In the future, we will conduct our model deeply. In fact, the architecture of our model is relatively simple, which limits the high-level representation learning process. Therefore, deeply architecture may help capture human mobility pattern more precise.

Funding Statement: This research was supported in part by National Key Research and Development Plan Key Special Projects under Grant No. 2018YFB2100303, Key Research and Development Plan Project of Shandong Province under Grant No. 2016GGX101032, Program for Innovative Postdoctoral Talents in Shandong Province under Grant No. 40618030001, National Natural Science Foundation of China under Grant No. 61802216, and Postdoctoral Science Foundation of China under Grant No. 2018M642613.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Gonzalez, M. C., Hidalgo, C. A., Barabasi, A. L. (2008). Understanding individual human mobility patterns. *Nature, 453(7196),* 779–782. DOI 10.1038/nature06958.
2. Hasan, S., Schneider, C. M., Ukkusuri, S. V., González, M. C. (2013). Spatiotemporal patterns of urban human mobility. *Journal of Statistical Physics, 151(1–2),* 304–318. DOI 10.1007/s10955-012-0645-0.
3. Hasan, S., Zhan, X., Ukkusuri, S. V. (2013). Understanding urban human activity and mobility patterns using large-scale location-based data from online social media. *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing, Chicago, Illinois*, pp. 1–8.
4. Song, C., Qu, Z., Blumm, N., Barabási, A. L. (2010): Limits of predictability in human mobility. *Science, 327 5968,* 1018–1021.
5. Sadilek, A., Krumm, J. (2012). Far out: predicting long-term human mobility. *Twenty-Sixth AAAI Conference on Artificial Intelligence, Ontario,* pp. 814–820.
6. Do, T. M. T., Gatica-Perez, D. (2014). Where and what: using smartphones to predict next locations and applications in daily life. *Pervasive and Mobile Computing, 12,* 79–91. DOI 10.1016/j.pmcj.2013.03.006.
7. Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Li, F. F. et al. (2016). Social lstm: human trajectory prediction in crowded spaces. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI*, pp. 961–971.
8. Yang, C., Sun, M., Zhao, W. X., Liu, Z., Chang, E. Y. (2017). A neural network approach to jointly modeling social networks and mobile trajectories. *ACM Transactions on Information Systems, 35(4),* 1–28. DOI 10.1145/3041658.
9. Sheng, X., Tang, J., Xiao, X., Xue, G. (2014). Leveraging GPS-less sensing scheduling for green mobile crowd sensing. *IEEE Internet of Things Journal, 1(4),* 328–336. DOI 10.1109/JIOT.2014.2334271.
10. Koren, Y., Bell, R., Volinsky, C. (2009): Matrix factorization techniques for recommender systems. *Computer, 42(8),* 30–37.
11. Gambs, S., Killijian, M. O., del Prado Cortez, M. N. (2012). Next place prediction using mobility markov chains. *Proceedings of the First Workshop on Measurement, Privacy, and Mobility, New York, NY,* pp. 1–6.
12. Feng, S., Li, X., Zeng, Y., Cong, G., Chee, Y. M. et al. (2015). Personalized ranking metric embedding for next new POI recommendation. *Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires,* pp. 2069–2075.
13. Lv, Q., Qiao, Y., Ansari, N., Liu, J., Yang, J. (2016). Big data driven hidden Markov model based individual mobility prediction at points of interest. *IEEE Transactions on Vehicular Technology, 66(6),* 5204–5216. DOI 10.1109/TVT.2016.2611654.
14. Liu, Q., Wu, S., Wang, L., Tan, T. (2016). Predicting the next location: a recurrent model with spatial and temporal contexts. *Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, Arizona,* pp. 194–200.
15. Yao, D., Zhang, C., Huang, J., Bi, J. (2017). Serm: a recurrent model for next location prediction in semantic trajectories. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, Singapore,* pp. 2411–2414.

16. Feng, J., Li, Y., Zhang, C., Sun, F., Meng, F. et al. (2018). Deepmove: Predicting human mobility with attentional recurrent networks. *Proceedings of the 2018 World Wide Web Conference. International World Wide Web Conferences Steering Committee, Lyon*, pp. 1459–1468.
17. Jamali, M., Ester, M. (2010). A matrix factorization technique with trust propagation for recommendation in social networks. *Proceedings of the Fourth ACM Conference on Recommender Systems, Barcelona,* pp. 135–142.
18. He, X., Zhang, H., Kan, M. Y., Chua, T. S. (2016). Fast matrix factorization for online recommendation with implicit feedback. *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa*, pp. 549–558.
19. Monreale, A., Pinelli, F., Trasarti, R., Giannotti, F. (2009). Wherenext: a location predictor on trajectory pattern mining. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris,* pp. 637–646.
20. Lee, S., Lim, J., Park, J., Kim, K. (2016). Next place prediction based on spatiotemporal pattern mining of mobile device logs. *Sensors, 16(2),* 145. DOI 10.3390/s16020145.
21. Mathew, W., Raposo, R., Martins, B. (2012). Predicting future locations with hidden Markov models. *Proceedings of the 2012 ACM Conference on Ubiquitous Computing, Pittsburgh,* pp. 911–918.
22. Xue, H., Huynh, D. Q., Reynolds, M. (2018). SS-LSTM: a hierarchical LSTM model for pedestrian trajectory prediction. *IEEE Winter Conference on Applications of Computer Vision,* pp. 1186–1194.
23. Kong, D., Wu, F. (2018). HST-LSTM: A hierarchical spatial-temporal long-short term memory network for location prediction. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, Stockholm,* pp. 2341–2347.
24. Liao, D., Liu, W., Zhong, Y., Li, J., Wang, G. (2018). Predicting activity and location with multi-task context aware recurrent neural network. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, Stockholm,* pp. 3435–3441.
25. Gao, Q., Zhou, F., Trajcevski, G., Zhang, K., Zhong, T. et al. (2019). Predicting human mobility via variational attention. *The World Wide Web Conference, San Francisco,* pp. 2750–2756.
26. Sun, K., Qian, T., Chen, T., Liang, Y., Nguyen, Q. V. H. et al. (2020). Where to go next: modeling long-and short-term user preferences for point-of-interest recommendation. *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20), New York,* pp. 214–221.
27. Zhang, Z., Li, C., Wu, Z., Sun, A., Ye, D. et al. (2020). NEXT: A neural network framework for next poi recommendation. *Frontiers of Computer Science, 14(2),* 314–333. DOI 10.1007/s11704-018-8011-2.
28. Chen, R., Chen, M., Li, W., Guo, N. (2020). Predicting future locations of moving objects by recurrent mixture density network. *ISPRS International Journal of Geo-Information, 9(2),* 116. DOI 10.3390/ijgi9020116.
29. Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint*, arXiv: 1408.5882.
30. Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *Proceedings of the 34th International Conference on Machine Learning, 70,* 1243–1252.
31. Wang, D., Zhang, J., Cao, W., Li, J., Zheng, Y. (2018). When will you arrive? Estimating travel time based on deep neural networks. *Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana,* pp. 2500–2507.
32. Lv, J., Li, Q., Sun, Q., Wang, X. (2018). T-CONV: A convolutional neural network for multi-scale taxi trajectory prediction. *IEEE International Conference on Big Data and Smart Computing (Bigcomp), Shanghai, China,* pp. 82–89.
33. Lin, Z., Feng, J., Lu, Z., Li, Y., Jin, D. (2019). DeepSTN+: Context-aware spatial-temporal neural network for crowd flow prediction in metropolis. *Proceedings of the AAAI Conference on Artificial Intelligence, 33,* 1020–1027. DOI 10.1609/aaai.v33i01.33011020.
34. Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint,* arXiv: 1301.3781.

35. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F. et al. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint,* arXiv: 1406.1078.
36. Kingma, D. P., Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv preprint*, arXiv: 1412.6980.
37. Yang, D., Zhang, D., Zheng, V. W., Yu, Z. (2014). Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45(1),* 129–142. DOI 10.1109/TSMC.2014.2327053.