An Attentional Recurrent Neural Network for Personalized Next Location Recommendation

Qing Guo,[1] **Zhu Sun,**[2] **Jie Zhang,**[3] **Yin-Leng Theng**[4]
[1,2,3,4]Nanyang Technological University, Singapore
[1,2,3,4]{qguo006, zhu.sun, zhangj, tyltheng}@ntu.edu.sg

Abstract

Most existing studies on next location recommendation propose to model the *sequential regularity* of check-in sequences, but suffer from the severe data sparsity issue where most locations have fewer than five *following locations*. To this end, we propose an *Attentional Recurrent Neural Network* (ARNN) to jointly model both the sequential regularity and transition regularities of similar locations (neighbors). In particular, we first design a meta-path based random walk over a novel knowledge graph to discover location neighbors based on heterogeneous factors. A recurrent neural network is then adopted to model the sequential regularity by capturing various contexts that govern user mobility. Meanwhile, the transition regularities of the discovered neighbors are integrated via the attention mechanism, which seamlessly cooperates with the sequential regularity as a unified recurrent framework. Experimental results on multiple real-world datasets demonstrate that ARNN outperforms state-of-the-art methods.

Introduction

With the advancement of mobile technologies, location-based social network (LBSN) services (e.g., Foursquare, Facebook place and Yelp) have become increasingly more popular in recent years. Next location recommendation in LBSNs is an important function as it enables users to explore more interesting locations and better plan their trips (Cheng et al. 2013; Liu et al. 2016; Feng et al. 2018). Existing approaches mostly focus on capturing the *sequential regularity* for modeling human mobility (Liu et al. 2016; Feng et al. 2018; Tang and Wang 2018), i.e., users' next movement highly relates to previously visited locations. Yet next location recommendation in LBSNs is still very difficult due to the extreme sparsity issue (Yao et al. 2017; Feng et al. 2018; Sun et al. 2019) – our data analysis shows that most locations are only followed by fewer than 5 locations consecutively. Such data sparsity issue makes it hard to learn effective sequential patterns among locations.

To solve the sparsity problem, we leverage the *transition regularities* of similar locations (neighbors) by assuming that they share similar transitional patterns. Fig. 1 presents a running example to illustrate our intuition. l_1 is an Italian restaurant followed by only a few locations, so it is challenging to predict the next location that a user might visit after l_1 by only capturing the sequential regularity. In fact, the transitional information of l_1's neighbors can help ease such sparsity issue. For instance, if users often watch a film at a cinema l_3 after having dinner at l_2 (l_1's neighbor which is a nearby Sushi restaurant), then l_3 can also be recommended as the next location for l_1. In other words, the transition regularities of neighbors serve as complementary information for locations with insufficient following check-ins. Finally, the transition regularities of neighbors at each time step can cooperate with the sequential regularity of previous check-in sequence to jointly predict users' next movement.

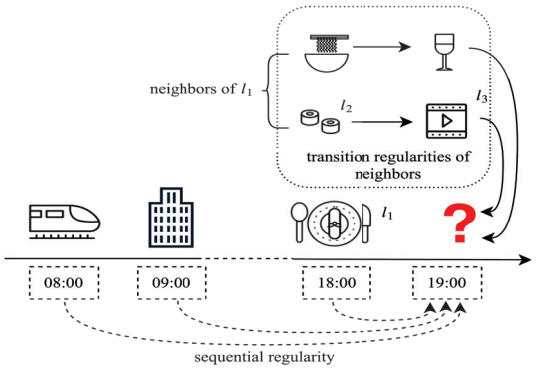


Figure 1: A running example

Due to the heterogeneous nature of LBSNs, multiple factors make connections between two locations including geographical distance, semantics (POI categories and tags) and user preference (Xie et al. 2016; Guo et al. 2017; Yin et al. 2017). We expand similar locations by assuming two locations are similar if they are: 1) geographically close; 2) described by the same semantics; 3) preferred by the same user. Our idea can be considered as a neighborhood-based strategy, i.e., aggregating useful features of neighbors for recommendation. To our best knowledge, we are the first to solve such data sparsity issue by coherently modeling the se-

quential regularity with the transition regularities of neighbors for next location recommendation.

To jointly model both regularities, we propose a novel *Attentional Recurrent Neural Network* framework (ARNN), which seamlessly integrates RNN (Hochreiter and Schmidhuber 1997) and attention mechanism (Bahdanau, Cho, and Bengio 2014) in a unified framework. A multi-modal embedding layer is first adopted to transform the sparse features of each check-in in the sequence into dense representations that are further fed into the recurrent layer. To extract various types of neighbors, we construct a novel knowledge graph (KG) to accommodate the heterogeneous information including users, locations and semantics (categories and tags of locations) into a unified space. A *meta-path* based random walk process over KG is then designed to efficiently discover the neighbors based on geographical, semantic and user preference factors. Next, we propose a novel method of using attention mechanism to capture the transition regularities of neighbors. Specifically, the attention layer is capable of selecting highly salient neighbors that are positively correlated to the current location at each time step. Finally, the attention layer is tailored to effectively cooperate with the recurrent layer in a unified manner. The experimental results on multiple LBSN datasets show that ARNN outperforms state-of-the-art algorithms, with significant improvements of accuracy by 9.21% on average.

Related Work

There are two types of methods for user next movement prediction – pattern-based and model-based methods (Yao et al. 2017). Pattern-based methods can only mine explicitly pre-defined patterns on dense trajectories without capturing all mobility regularities about user movement (Monreale et al. 2009; Li et al. 2010). Thus, they are not suitable for LBSN scenario with extremely sparse trajectory data. In contrast, model-based methods are favored in LBSNs, due to their ability to model complex movement regularities and fuse heterogeneous data (He et al. 2016; Tang and Wang 2018). Hence, we mainly discuss state-of-the-art model-based methods that can be further classified into Markov-based and NN-based methods.

Markov-based Models

Generally, Markov-based models (MMs) predict the probability of future visit by constructing a location transition matrix. Due to the sparse check-in data, latent factor model (Koren, Bell, and Volinsky 2009) is always adopted to help learn dense representations. By combining MMs with matrix factorization (MF) (Mnih and Salakhutdinov 2008), Rendle et al. (2010) propose FPMC to model user sequential behavior at personalized level. Cheng et al. (2013) extend FPMC to learn transition regularities with localized spatial constraint. He et al. (2016) incorporate temporal and categorical information via a weighting scheme based on first-order Markov chain property.

However, MMs aim to learn transition probability between successive locations, thus failing to capture high-order sequential regularity. Besides, existing MMs fuse different contexts in a linear fashion such that impacts of those contexts cannot be well captured. Instead, we adopt the recurrent neural network, which can model the long-term sequences and flexibly fuse the diverse contextual information at each time step via the multi-modal embedding layer.

NN-based Models

Recent studies apply Neural Network (NN) in next location recommendation, thanks to its strong capability of capturing sequential information (Hochreiter and Schmidhuber 1997; Mikolov et al. 2010; Zhang et al. 2014; Huang et al. 2019). By leveraging the similar users, Massimo and Ricci (2018) propose an approach based on Inverse Reinforcement Learning (IRL) that can allow users to maximise their estimated reward (utility) by visiting the suggested POIs. By using the attention mechanism, Huang et al. (2019) propose an attention-based LSTM model (ATST-LSTM) that can focus on the relevant historical check-in records in a check-in sequence by resembling the sequence to sequence model in machine translation (Sutskever, Vinyals, and Le 2014). To better capture the multi-level periodicity, Feng et al. (2018) propose DeepMove by fusing an attention module into Recurrent Neural Network (RNN), to automatically select highly correlated historical records for current status. Caser adopts Convolutional Neural Network (CNN) to capture the joint effects of previous check-ins on the current check-in (Tang and Wang 2018). This enables Caser to advance DeepMove, as DeepMove models the impact of each check-in independently.

Another research line explores new methods of using contextual information. Some studies model human mobility by using temporal and spatial contexts via time and distance transition matrices (Liu et al. 2016; Zhang et al. 2017). However, the model training is complicated by the heavy parameters. SERM jointly learns the embeddings of multiple contexts (e.g., temporal and semantic information) with user preference (Yao et al. 2017). It is more flexible to incorporate various contexts with fewer parameters involved. However, only limited improvements are achieved by modeling both sequential regularity and multiple contexts, as these methods still cannot generate decent recommendations for the locations with only few following records. Therefore, we propose a novel recurrent framework (ARNN) to exploit the transition regularities of neighbors based on the heterogeneous contextual factors to overcome the sparsity challenge.

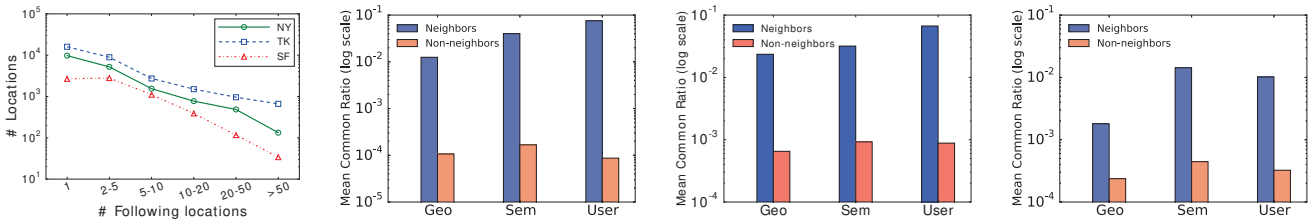
Problem Analysis and Formulation

Problem Analysis

We adopt several real-world datasets from Foursquare (Yang et al. 2015) and Gowalla[1]. As most successive check-ins happen within $10km$ (Cheng et al. 2013; Feng et al. 2015), we choose three major cities for the data analysis and experiments: New York (NY) and Tokyo (TK) from Foursquare and San Francisco (SF) from Gowalla. Foursquare APIs[2] are applied to collect categories and tags of locations in SF.

[1] https://snap.stanford.edu/data/loc-gowalla.html

[2] https://developer.foursquare.com/places-api



(a) Three cities (b) New York (c) Tokyo (d) San Fransisco

Figure 2: Fig. 2(a) reports the distribution of locations w.r.t. # following locations; Fig. 2(b-d) presents the mean common ratio of neighbors vs. non-neighbors based on the three factors, including geographical, semantic and user preference factors.

Sparsity Issue. To assess the sparsity, for each location l_k, we calculate the number of its following locations. A following location of l_k is defined as the location visited after l_k successively. Fig. 2(a) shows that most locations have a very limited number of following locations. In particular, over 50% locations have only one following location in NY and TK while more than 80% have fewer than 5 following locations in all three cities. Such insufficiency of following locations create extreme hardness to learn the transitional patterns between locations to predict the next movement.

Transition Regularities of Neighbors. To quantitatively validate our assumption on transition regularities of neighbors, we calculate the *common ratio* about the following locations for a given location pair – l_k and l_j. Let $L_f(l_k)$ and $L_f(l_j)$ denote the sets of following locations of l_k and l_j, respectively. The common ratio measures the overlap degree of $L_f(l_k)$ and $L_f(l_j)$ defined as: $\alpha(l_k, l_j) = \frac{|L_f(l_k) \cap L_f(l_j)|}{|L_f(l_k) \cup L_f(l_j)|}$. Based on our intuition, if l_k is similar to l_j, $L_f(l_k)$ and $L_f(l_j)$ should have more overlap. We calculate the mean common ratios of three types of neighbors w.r.t. geographical (Geo), semantic (Sem) and user preference (User) factors compared with non-neighbors. Fig. 2(b-d) show that the mean common ratios of neighbors are consistently much higher than non-neighbors. This verifies that similar locations share more similar transitional patterns. Therefore, aggregating the transition regularities of neighbors can help ease the sparsity issue for high-quality recommendation performance.

Problem Formulation

To formulate the problem, let $U = \{u_1, u_2, ..., u_{|U|}\}$, $L = \{l_1, l_2, ..., l_{|L|}\}$ and $V = \{w_1, w_2, ..., w_{|V|}\}$ denote a set of users, locations and words (POI categories and tags), respectively.

Definition 1. Historical Sequence. The historical sequence of user u_i is a temporally ordered sequential check-in records, i.e., $His(u_i) = \{r_1, r_2, ..., r_H\}$. And each check-in $r_k \in His(u_i)$ is a tuple (u_i, l_k, t_k) where l_k is the location and t_k is the timestamp. $S(l_k) \subset V$ is the semantics of l_k including a bag of words (the categories and tags of l_k).

Definition 2. Trajectory. A trajectory of a user u_i is a subsequence of $His(u_i)$ where the time interval between two successive check-ins is smaller than the pre-defined time threshold, Δt, i.e., $Tra(u_i) = \{r_1, r_2, ..., r_M\}$ is a segment of $His(u_i)$, if $0 < t_{m+1} - t_m \leqslant \Delta t, \forall 1 \leqslant m < M$.

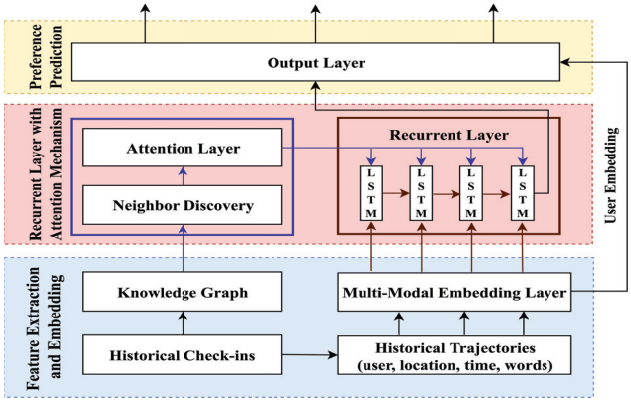


Figure 3: The architecture of ARNN

The next location recommendation is formulated as: given a trajectory of user u_i, i.e., $Tra(u_i) = \{r_1, r_2, ..., r_{K-1}\}$, a list of locations that u_i would visit at time step t_K are generated. As the following location of l_k is highly sparse, i.e., $|L_f(l_k)|$ is always fewer than 5, we thus leverage transition regularities of neighbors at each time step to ease such sparsity issue for better recommendation.

The Proposed Framework

Our proposed *Attentional Recurrent Neural Network* framework (ARNN) consists of four layers: embedding layer, attention layer, recurrent layer and output layer, depicted in Fig. 3. We first design a multi-modal embedding layer to learn the dense representations of locations and various contexts. To find relevant neighbors, a novel knowledge graph (KG) is built by fusing the heterogeneous data into a unified space. Then, a meta-path based random walk over the KG is designed to efficiently discover the neighbors based on multiple factors. To capture the transition regularities of relevant neighbors, an attention layer is developed to generate a weighted embedding by distinguishing each neighbor of the current location. By integrating the weighted embedding from attention layer and current status, the recurrent layer

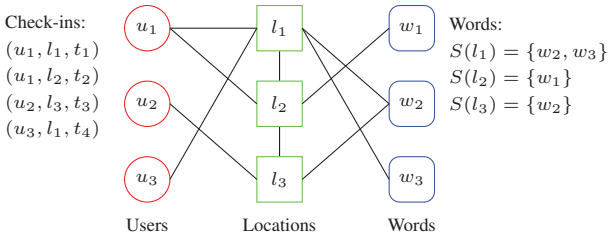


Figure 4: An illustrative example of knowledge graph

can generate the hidden state that encodes the information observed until the current time step. Finally, the output layer jointly considers both the information of the trajectory and user general interest to recommend the next location.

Neighbor Discovery

To capture the heterogeneous factors that enable the connections among locations, we leverage the knowledge graph and meta-path for neighbor discovery. First, a novel knowledge graph is constructed, i.e., $G = \{U \cup L \cup V, E_{UL} \cup E_{LL} \cup E_{LV}\}$, where E_{LL} represents location-location relations, that is, two locations are linked if they are geographically close to each other, i.e., $dist(l_k, l_j) \leqslant \Delta d$ where Δd is a distance threshold; E_{LV} denotes location-word affiliations; E_{UL} denotes user historical visits. Each input tuple (u_i, l_k, t_k) denotes a check-in, r_k, while each location is described by a set of words, $S(l_k)$. By aggregating the historical check-ins, the words of locations, and geographical distance between locations, G is capable of accommodating the heterogeneous data of historical records into a unified representation space as illustrated in Fig. 4.

Next, we utilize *meta-path*, which is a critical technique to capture diverse semantic relations in a heterogeneous network (Sun et al. 2011), to extract various kinds of neighbors from G. Specifically, a meta-path is represented by a sequence of linked entities at the scheme level, e.g., meta-path LUL can build bridges for two locations visited by the same users. Based on our analysis in Fig. 2 (b-d), three meta-paths, LL, LVL and LUL, are selected, representing geographical, semantic and user preference factors, respectively. For each location, a random walk process is conducted to generate paths based on the above meta-paths. Then, the neighbors are extracted from those paths. The random walk process is illustrated as follows. A meta-path is defined as $p = T_1 T_2 \cdots T_m \cdots$, where T_m is the type of m-th entity. The transition probability between two linked entities is determined by the neighborhood size with constraint based by p, i.e., $\text{Prob}(v_m | v_{m-1}, p) = \frac{1}{|N_{T_m}(v_{m-1})|}$, if $T(v_{m-1}) = T_{m-1}$ and $T(v_m) = T_m$, where $T(v)$ is the type of entity v and $N_{T_m}(v)$ is the first-order neighbor set of v in type T_m. By following p with the transition probability, the random walker could generate a path until it reaches the walk length. The process terminates if enough paths are created. Finally, we extract locations from those paths to create the neighbor set of a given location. For example, we have $l_1 \to w_2 \to l_3$ in Fig. 4, so $l_3 \in N_p(l_1)$ where $p = LVL$ and $N_p(l_1)$ is the neighbor set of l_1 w.r.t. meta-path p. Algorithm 1 describes the details about the random walk process.

Algorithm 1: Meta-Path based Random Walk

Input: G, location l_k, meta-path p, walk length $walk\_len$, walk number $walk\_num$
Output: A set of neighbors, $N_p(l_k)$

```
Paths = [];
for i ← 1 to walk_num do
    path = [];
    m ← 1;
    while m ≤ walk_len do
        walk to entity v_m by Prob(v_m | v_{m-1}, p);
        if T(v_m) == T_m then
            Add v_m to path ;
            m ← m + 1 ; //move one step forward
    Add path to Paths ;
Extract neighbors from Paths as N_p(l_k);
return N_p(l_k) ;
```

Embedding Layer

A multi-modal embedding layer is devised to jointly learn the embeddings of the location with its temporal context, semantic context, and user general interest. Specifically, different information of each check-in is initially represented as one-hot vectors. As the timestamp is a continuous value, we then map one day into 24 hours so that t_k can be transformed into 24-dimensional one-hot vector. Each location is represented by a $|L|$-dimensional one-hot vector. For the semantics of each location, we first transfer each word into a $|V|$-dimensional one-hot vector, and sum them up as a new vector. Besides, we assume each user has her general preference, represented as a $|U|$-dimensional one-hot vector. These one-hot vectors are fed into the embedding layer to learn the low-dimensional dense representations of the timestamp, location, semantics and user general interest, denoted as $\boldsymbol{e}_t \in \mathbb{R}^{D_t}$, $\boldsymbol{e}_l \in \mathbb{R}^{D_l}$, $\boldsymbol{e}_s \in \mathbb{R}^{D_s}$ and $\boldsymbol{e}_u \in \mathbb{R}^{D_u}$, respectively. Note, unlike other embeddings, user embedding, $\boldsymbol{e}_u$, is not taken as an input into the recurrent layer, but only used by the output layer at the last time step of the trajectory, as it is considered to be stable through time (Yao et al. 2017; Feng et al. 2018; Tang and Wang 2018). These dense representations can model the semantic and spatio-temporal features of each check-in more precisely as well as reduce the computation.

Attention Layer

To model the transition regularities of neighbors, we exploit the attention mechanism (Mnih et al. 2014) to choose salient neighbors based on current location automatically at each time step along the sequence. Specifically, the attention layer is designed to calculate the similarity (i.e., attention weights) between current location and each neighbor. If a neighbor is more similar with current location w.r.t. transition patterns, it would be assigned with a larger attention weight. Besides, the attention layer is parametrized as a feed-forward neural

network, which is jointly trained with other layers. Fig. 3 also illustrates the structure of the attention layer. The following describes the attention computation at time step t_k:

$$\boldsymbol{c}_k = \sum\nolimits_n \boldsymbol{\alpha}_k(n)\boldsymbol{e}_{l_n}, \tag{1}$$

$$\boldsymbol{\alpha}_k(n) = \mathrm{softmax}(f_a(\boldsymbol{e}_{l_k}, \boldsymbol{e}_{l_n})), \tag{2}$$

$$f_a(\boldsymbol{e}_{l_k}, \boldsymbol{e}_{l_n}) = \tanh(\boldsymbol{e}_{l_k}^{\intercal} \cdot \boldsymbol{W}_a \cdot \boldsymbol{e}_{l_n}) \tag{3}$$

where $\boldsymbol{e}_{l_k}$ is the embedding of current location; $\boldsymbol{e}_{l_n}$ is the embedding of location l_n, a neighbor of l_k; $\boldsymbol{c}_k \in \mathbb{R}^{D_l}$ is computed as the weighted average embedding over all the neighbors; $\boldsymbol{W}_a$ is the weight matrix; $\boldsymbol{\alpha}_k(n) \in [0, 1]$ is the attention weight of l_n and $\sum_n \boldsymbol{\alpha}_k(n) = 1$; f_a is the score function, which measures the relevance between the neighbor and current location.

Recurrent Layer

To capture the high-order sequential regularity, we adopt *Long Short-Term Memory* (LSTM) as the recurrent unit, due to its strong capability of memorizing long-range sequential information (Hochreiter and Schmidhuber 1997). At time step t_k, we have the representations generated by different components of ARNN, including: 1) $\boldsymbol{e}_{l_k}$, $\boldsymbol{e}_{t_k}$, $\boldsymbol{e}_{s_k}$ from the multi-modal embedding layer, which embed current location, its temporal and semantic contexts; 2) $\boldsymbol{c}_k$ from the attention layer, which represents the weighted embedding based on relevant neighbors; 3) $\boldsymbol{h}_{k-1}$ from previous recurrent unit, which encodes the information of the trajectory until t_{k-1}. The goal of the recurrent unit is to jointly integrate the three groups of representations to update the hidden state, which preserves the information observed until the current time step t_k. First, they are concatenated to generate a new representation, i.e., $\boldsymbol{e}_k = [\boldsymbol{e}_{l_k}; \boldsymbol{e}_{t_k}; \boldsymbol{e}_{s_k}; \boldsymbol{c}_k]$, where $\boldsymbol{e}_k \in \mathbb{R}^{D_e}$, and $D_e = 2D_l + D_t + D_s$. Then, the k-th recurrent unit takes $\boldsymbol{e}_k$ and $\boldsymbol{h}_{k-1}$ to update the hidden state as follows:

$$\boldsymbol{h}_k = f(\boldsymbol{W}_{r_1} \cdot \boldsymbol{h}_{k-1} + \boldsymbol{W}_{r_2} \cdot \boldsymbol{e}_k + \boldsymbol{b}_r) \tag{4}$$

where $\boldsymbol{h}_k \in \mathbb{R}^{D_h}$ is a D_h-dimensional vector which represents the hidden state at t_k; $\boldsymbol{W}_{r_1}$ and $\boldsymbol{W}_{r_2}$ are the weight matrices, and $\boldsymbol{b}_r$ is the bias term; $f(\cdot)$ denotes the forward function of LSTM[3].

Output Layer

By performing Eq. 4 through time, we can obtain the hidden state, $\boldsymbol{h}_{K-1}$, which coherently inherits the information of the trajectory until time step t_{K-1}. We combine it with user general preference to predict where user u_i would visit at t_K. For this purpose, we first decode $\boldsymbol{h}_{K-1}$ into a D_u-dimensional vector, i.e., $\boldsymbol{o}_{K-1} \in \mathbb{R}^{D_u}$. Then, we concatenate the user embedding, $\boldsymbol{e}_{u_i}$, with $\boldsymbol{o}_{K-1}$ to calculate the distribution over $|L|$ locations as follows:

$$\boldsymbol{o}_{K-1} = \boldsymbol{W}_{o_1} \cdot \boldsymbol{h}_{K-1} + \boldsymbol{b}_{o_1} \tag{5}$$

$$\boldsymbol{o}'_{K-1} = \boldsymbol{W}_{o_2} \begin{bmatrix} \boldsymbol{o}_{K-1} \\ \boldsymbol{e}_u \end{bmatrix} + \boldsymbol{b}_{o_2} \tag{6}$$

$$\hat{\boldsymbol{y}}_K^{u_i} = \mathrm{softmax}(\boldsymbol{o}'_{K-1}) \tag{7}$$

[3]The activation functions of input, forget and output gates are all *sigmoid* function while it is the *tanh* function for cell gate.

Table 1: Statistics of datasets ($\Delta t = 12h$)

City	Users	Locations	Check-ins	Trajectories	Words
NY	490	19253	77853	9082	242
TK	1499	33530	247794	31180	235
SF	170	7340	32058	2953	306

where $\boldsymbol{W}_{o_1} \in \mathbb{R}^{D_u \times D_h}$, $\boldsymbol{W}_{o_2} \in \mathbb{R}^{|L| \times 2D_u}$ are transformation matrices and $\boldsymbol{b}_{o_1}, \boldsymbol{b}_{o_2}$ are the bias terms; $\hat{\boldsymbol{y}}_K^{u_i}$ represents the probability distribution over L via softmax function at time step t_K. Note that user embedding, $\boldsymbol{e}_{u_i}$, is only utilized in the output layer as it represents the general interest of u_i, which does not change with time.

Model Optimization

Given the training samples, the parameters are optimized by minimizing the following loss function:

$$J = \sum\nolimits_{i=1}^{|U|} \sum\nolimits_{k=1}^{|L|} \boldsymbol{y}_K^{u_i}(l_k) \cdot log\hat{\boldsymbol{y}}_K^{u_i}(l_k) + \lambda||\boldsymbol{\Theta}||_2 \tag{8}$$

where J is the Cross Entropy Loss between the predictions and ground truth; $\boldsymbol{y}_K^{u_i}$ is a one-hot vector to describe the ground truth location at t_K, i.e., $\boldsymbol{y}_K^{u_i}(l_k) = 1$ if u_i visited l_k at t_K; $||\boldsymbol{\Theta}||_2$ is the regularization term to avoid over-fitting; λ controls the strength of the regularization.

Experiments

We carry out experiments to investigate the following questions: (1) How will different meta-paths affect our model performance? (2) How will the time threshold and the embedding dimensionality affect our model accuracy? (3) How will our approach compare with state-of-the-art methods? (4) What is the convergence property of our model?

Experimental Setup

Data Prepossessing. Following previous work (Yao et al. 2017), users with fewer than 5 trajectories and trajectories with fewer than 3 check-ins are removed. Previous works (Yao et al. 2017; Zhang et al. 2017; Feng et al. 2018) also usually filter out inactive locations to ease the sparsity, but this filtering operation may induce the model to learn wrong transition information. Thus, no filtering is conducted to locations in our experiments. We adopt the time threshold $\Delta t = 12h$ to create high-quality trajectories, because our experiments show that it is the optimal setting (see Table 2). The statistics of datasets are summarized in Table 1.

Comparison Methods. We compare ARNN with state-of-the-art methods: 1) **UCF** (Sarwar et al. 2001): is a user-based collaborative filtering by using the user-location matrix; 2) **FPMC** (Rendle, Freudenthaler, and Schmidt-Thieme 2010): extends Markov Chain for sequential prediction; 3) **FPMC-LR** (Cheng et al. 2013): extends FPMC by considering geographical constraints; 4) **LSTM** (Hochreiter and Schmidhuber 1997): is a popular variant model of RNN for sequential prediction; 5) **ST-RNN** (Liu et al. 2016): is a recent RNN-based model that incorporates temporal and geographical information; 6) **SERM** (Yao et al. 2017): is a state-of-the-art method combining multiple contexts in a recurrent

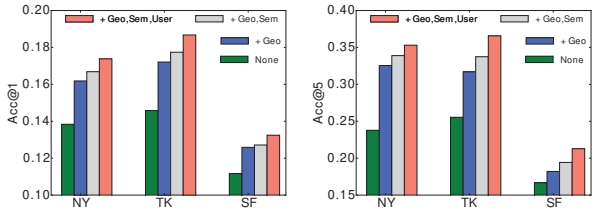


Figure 5: Performance of variants of ARNN on the three datasets when various meta-path are cumulatively incorporated.

model; 7) **DeepMove** (Feng et al. 2018): is a state-of-the-art recurrent model capturing multi-level influence of previous check-ins; 8) **Caser** (Tang and Wang 2018): is a state-of-the-art method that models the joint effects of previous visits based on CNN.

Evaluation Metrics. Following the existing works (Zhang et al. 2017; Yao et al. 2017; Feng et al. 2018), we employ prediction accuracy ($Acc@N$, $N = 1, 5, 10, 20$) to check whether the ground-truth location appears in the top-N recommendation list.

Parameter Settings. We use the earliest 70% check-ins of each user as training set (70%), and the latest 20% as test set and remaining 10% as validation set. The parameters are turned to achieve the best results, or set as suggested by the original papers for all comparison methods. For the neighbor discovery procedure in ARNN, the distance threshold $\Delta d = 2km$, walk number $walk\_num = 50$ and walk length $walk\_len = 10$ for the meta-path based random walk. We apply a grid search to find the optimal settings for the size of user embedding (D_u), location embedding (D_l), timestamp embedding (D_t) and semantics embedding (D_s) (see Fig. 6). The 4 embedding sizes are set as the same ($D_u = D_l = D_t = D_s$) and are 120/160/120 for NY, TK and SF, respectively. The number of hidden units, D_h, is set as 64 for all cities. The number of recurrent layers is 1. The epoch number is set as 20. The learning rate is 0.01. The regularization parameter λ is chosen as 0.01. The hidden state and cell state are initialized as zero. We use Stochastic Gradient Descent (Bottou 1991) and Back Propagation Through Time (Rumelhart, Hinton, and Williams 1986) to learn the parameters with a batch size of 64.

Analysis of Meta-paths

To investigate the effect of various meta-paths, we cumulatively incorporate the selected meta-paths into neighbor discovery (see Fig. 5). The results w.r.t. Acc@1 and Acc@5, where *Geo*, *Sem* and *User* represent LL, LVL and LUL, respectively. *None* means no meta-path is used, i.e., ARNN is degraded to LSTM. Note that similar trends can also be observed for Acc@10 and Acc@20. It can be observed that the performance becomes better as more meta-paths are incorporated. Moreover, all the ARNN variants outperform LSTM, which confirms the effectiveness of utilizing transition regularities of neighbors. In addition, meta-path LUL delivers more significant enhancement than others, which verifies that the similar properties of two locations can be well captured by user preference.

With the incorporation of all meta-paths, the performance is enhanced significantly by 14.69% and 18.05% averagely w.r.t. Acc@1 and Acc@5 across the three datasets. Nevertheless, the improvements are not always significant with LVL and LL. With LVL, the random walker could reach neighbors that are far away from the location. The distant locations may have no common transition patterns, as the distance of two successive check-ins is often less than $10km$ (Feng et al. 2015). By following LL, although the random walker can find nearby neighbors, these neighbors (e.g., a museum) may not share similar characteristics with the location (e.g., a gym club). Despite some unsatisfactory neighbors could be retrieved, their saliences can be distinguished automatically by the attention layer.

Table 2: Performance variation of ARNN with different time threshold (Δt) on three datasets evaluated by Acc@N. The best performance is highlighted in bold.

City	Metric	3h	6h	12h	1d	2d
NY	Acc@1	0.1816	**0.2058**	0.1738	0.1526	0.1220
	Acc@5	0.3437	0.348	**0.3530**	0.2943	0.2498
	Acc@10	0.4076	0.4087	**0.4162**	0.3317	0.2977
	Acc@20	0.4474	**0.4571**	0.4393	0.3544	0.3097
	Pct($\leqslant \Delta t$)	46.75%	54.45%	63.83%	80.04%	90.45%
TK	Acc@1	0.1811	0.1801	**0.1867**	0.1651	0.1321
	Acc@5	0.3448	0.3605	**0.3657**	0.3044	0.2283
	Acc@10	0.4121	0.4180	**0.4285**	0.3569	0.2637
	Acc@20	0.4625	**0.4889**	0.4864	0.4125	0.3392
	Pct($\leqslant \Delta t$)	58.96%	63.89%	71.74%	84.33%	92.18%
SF	Acc@1	0.0699	0.1297	**0.1324**	0.0816	0.0829
	Acc@5	0.1441	0.2010	**0.2128**	0.1780	0.1685
	Acc@10	0.1594	0.2318	**0.2336**	0.2126	0.2072
	Acc@20	0.1703	0.2431	**0.2530**	0.2336	0.2251
	Pct($\leqslant \Delta t$)	50.26%	56.68%	63.48%	80.34%	90.52%

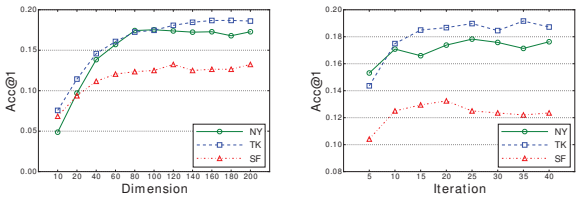
Analysis of Time Threshold

As the time interval of two successive check-ins has critical influence on users' next location (Yao et al. 2017; Feng et al. 2015), it is necessary to investigate the impact of various settings of the time threshold (Δt) on the accuracy of ARNN (Table 2). To better understand the results, we calculate the percentage of the successive check-in pairs that happen within various Δt, i.e., $\text{Pct}(\leqslant \Delta t) = \frac{\sum_{i=1}^{|U|}|\{(r_{k-1},r_k)\ |r_k \in His(u_i), t_k - t_{k-1} \leqslant \Delta t\}|}{\sum_{i=1}^{|U|}|\{(r_{k-1},r_k)\ |r_k \in His(u_i)\}|}$, where $His(u_i)$ represents the historical sequences of u_i, r_{k-1} indicates a check-in and its following check-in is r_k.

For about 50% and 60% of successive check-ins, their time interval is smaller than $3h$ and $6h$, respectively, reinforcing the previous finding that the majority of successive check-ins happen within a few hours (Cheng et al. 2013). The best results are achieved with $\Delta t = 12h$, implying successive check-ins with the time interval in $6h$ – $12h$ still carry effective transition information. However, the performance deteriorates when Δt increases to $24h$, because the time interval exceeds to a point such that wrong transition information is introduced. Overall, $\Delta t = 12h$ is a robust setting to

Table 3: Performance of all the comparison methods on the three real-world datasets measured Acc@N. The best performance is highlighted in bold while the second best performance is marked by '†'.

City	Metric	UCF	FPMC	FPMC-LR	LSTM	ST-RNN	SERM	Caser	ARNN	Improve
NY	Acc@1	0.0010	0.0214	0.0248	0.1384	0.1292	0.1571	0.1608†	**0.1738**	8.07%
	Acc@5	0.0055	0.0386	0.0403	0.2378	0.2474	0.2880	0.3210†	**0.3530**	9.97%
	Acc@10	0.0096	0.0735	0.0783	0.2752	0.2803	0.3230	0.3767†	**0.4162**	10.50%
	Acc@20	0.0131	0.0872	0.0929	0.3058	0.3299	0.3437	0.4016†	**0.4393**	9.37%
TK	Acc@1	0.0092	0.0196	0.0227	0.1458	0.1572	0.1608	0.1709†	**0.1867**	10.34%
	Acc@5	0.0197	0.0312	0.0358	0.2554	0.2443	0.2899	0.3302 †	**0.3657**	8.74%
	Acc@10	0.0385	0.0597	0.0626	0.2989	0.2794	0.3407	0.3852 †	**0.4285**	10.51%
	Acc@20	0.0199	0.0686	0.0753	0.3399	0.3285	0.3842	0.4387 †	**0.4864**	13.09%
SF	Acc@1	0.0005	0.0122	0.0167	0.1116	0.1035	0.1205	0.1220†	**0.1324**	8.54%
	Acc@5	0.0025	0.0245	0.0314	0.1667	0.1630	0.1949†	0.1894	**0.2128**	9.43%
	Acc@10	0.0045	0.0426	0.0512	0.1845	0.1860	0.2098	0.2177†	**0.2336**	7.33%
	Acc@20	0.0076	0.0541	0.0625	0.2009	0.2098	0.2321	0.2345†	**0.2530**	7.90%



(a) Dimentionality (b) Convergence

Figure 6: Analysis of Dimensionality and Convergence.

generate high-quality trajectories for model training.

Analysis of Dimensionality and Convergence

Fig. 6(a) describes the recommendation performance w.r.t. Acc@1 for various dimensionality (i.e., embedding size) with other optimal hyperparameters fixed on all the three datasets. Note similar trends can be observed for other metrics. The performance becomes stable when the dimension increases to a certain level. Besides, the optimal dimension size of Tokyo is higher than others, since there are much more locations in Tokyo. Moreover, ARNN consistently outperforms other state-of-the-art methods when its results become stable, indicating that it can well deal with the variation of dimensionality. Furthermore, to understand the convergence of ARNN, we report the performance change with respect to the number of iterations. As shown in Fig. 6(b), we can observe that ARNN can converge within 20 iterations. In summary, with strong ability to dealing with the dimensionality and fast convergence speed, ARNN is a robust approach that can be applied in practical scenarios.

Comparative Results

With optimal settings of involved parameters, we conduct experiments to evaluate our ARNN with state-of-the-art methods. Table 3 presents the performance of all comparison methods evaluated by Acc@N. UCF performs worst since it only models user general interest without considering sequential regularity. By modeling sequential regularity, FPMC performs better than UCF. FPMC-LR outperforms FPMC by further fusing geographical information. However, the results of both Markov-based methods are still poor, as they fail to model high-order sequential regularity.

In general, RNN-based methods outperform Markov-based methods because of their capability of memorizing long-term dependencies (Hochreiter and Schmidhuber 1997). ST-RNN delivers decent results by exploiting spatial and temporal contexts. However, ST-RNN is outperformed by LSTM in some cases due to the stronger ability of LSTM to model long sequence than basic RNN. By considering semantic context, SERM performs better than ST-RNN. DeepMove adopts a similar attention-based GRU architecture applied in language translation (Luong, Pham, and Manning 2015), to capture the multi-level influence of distant check-ins in the sequence. The way of integrating attention layer in ARNN is divergent from DeepMove. To explain, the attention layer in ARNN is designed to capture the transition regularities of each neighbor at each time step while DeepMove captures the impacts of previous check-ins along the entire sequence in DeepMove. In general, Caser outperforms other RNN-based methods by modeling the effects of past check-ins jointly while others model previous check-ins independently. However, either the methods of learning embeddings of different contexts (ST-RNN and SERM) or modeling the complex sequential regularity (DeepMove and Caser) are deficient in solving the sparsity issue. By jointly modeling the transition regularities of various neighbors and sequential regularity, our ARNN performs the best – the average improvements are 9.48%, 10.26%, 7.89% on the three datasets, respectively. The significant improvement margins demonstrate the strong ability of ARNN in resolving the sparsity for more accurate recommendation.

Conclusion and Future Work

In this paper, we explore the transition regularities of neighbors to resolve the sparsity issue for next location recommendation. We first design a meta-path based random walk on a novel knowledge graph to discover relevant neighbors based on heterogeneous factors. Next, we jointly model both sequential regularity and transition regularities of neighbors by proposing a novel attentional recurrent neural network framework, ARNN. Experimental results on multiple real-world datasets demonstrate the superiority of ARNN to state-of-the-art approaches. In the future, we plan to explore more useful data sources such as weather and traffic information, to provide more satisfying recommendations.

Acknowledgments

This work is partially supported by the MOE AcRF Tier 1 funding (M4011894.020) awarded to Dr. Jie Zhang.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bottou, L. 1991. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes* 91(8):12.

Cheng, C.; Yang, H.; Lyu, M. R.; and King, I. 2013. Where you like to go next: Successive point-of-interest recommendation. In *IJCAI*, volume 13, 2605–2611.

Feng, S.; Li, X.; Zeng, Y.; Cong, G.; Chee, Y. M.; and Yuan, Q. 2015. Personalized ranking metric embedding for next new poi recommendation. In *IJCAI*, 2069–2075.

Feng, J.; Li, Y.; Zhang, C.; Sun, F.; Meng, F.; Guo, A.; and Jin, D. 2018. Deepmove: Predicting human mobility with attentional recurrent networks. In *WWW*, 1459–1468.

Guo, Q.; Sun, Z.; Zhang, J.; Chen, Q.; and Theng, Y.-L. 2017. Aspect-aware point-of-interest recommendation with geo-social influence. In *UMAP*, 17–22. ACM.

He, J.; Li, X.; Liao, L.; Song, D.; and Cheung, W. K. 2016. Inferring a personalized next point-of-interest recommendation model with latent behavior patterns. In *AAAI*, 137–143.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Huang, L.; Ma, Y.; Wang, S.; and Liu, Y. 2019. An attention-based spatiotemporal lstm network for next poi recommendation. *IEEE Transactions on Services Computing*.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer* (8):30–37.

Li, Z.; Ding, B.; Han, J.; Kays, R.; and Nye, P. 2010. Mining periodic behaviors for moving objects. In *SIGKDD*, 1099–1108. ACM.

Liu, Q.; Wu, S.; Wang, L.; and Tan, T. 2016. Predicting the next location: A recurrent model with spatial and temporal contexts. In *AAAI*, 194–200.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Massimo, D., and Ricci, F. 2018. Harnessing a generalised user behaviour model for next-poi recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 402–406. ACM.

Mikolov, T.; Karafiát, M.; Burget, L.; Cernockỳ, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *Interspeech*, volume 2, 3.

Mnih, A., and Salakhutdinov, R. R. 2008. Probabilistic matrix factorization. In *NIPS*, 1257–1264.

Mnih, V.; Heess, N.; Graves, A.; et al. 2014. Recurrent models of visual attention. In *NIPS*, 2204–2212.

Monreale, A.; Pinelli, F.; Trasarti, R.; and Giannotti, F. 2009. Wherenext: a location predictor on trajectory pattern mining. In *SIGKDD*, 637–646. ACM.

Rendle, S.; Freudenthaler, C.; and Schmidt-Thieme, L. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, 811–820. ACM.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *nature* 323(6088):533.

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW*, 285–295. ACM.

Sun, Y.; Han, J.; Yan, X.; Yu, P. S.; and Wu, T. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB* 4(11):992–1003.

Sun, Z.; Guo, Q.; Yang, J.; Fang, H.; Guo, G.; Zhang, J.; and Burke, R. 2019. Research commentary on recommendations with side information: A survey and research directions. *Electronic Commerce Research and Applications* 37:100879.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*, 3104–3112.

Tang, J., and Wang, K. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WSDM*, 565–573. ACM.

Xie, M.; Yin, H.; Xu, F.; Wang, H.; and Zhou, X. 2016. Graph-based metric embedding for next poi recommendation. In *WISE*, 207–222. Springer.

Yang, D.; Zhang, D.; Zheng, V. W.; and Yu, Z. 2015. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45(1):129–142.

Yao, D.; Zhang, C.; Huang, J.; and Bi, J. 2017. Serm: A recurrent model for next location prediction in semantic trajectories. In *CIKM*, 2411–2414. ACM.

Yin, H.; Wang, W.; Wang, H.; Chen, L.; and Zhou, X. 2017. Spatial-aware hierarchical collaborative deep learning for poi recommendation. *TKDE* 29(11):2537–2551.

Zhang, Y.; Dai, H.; Xu, C.; Feng, J.; Wang, T.; Bian, J.; Wang, B.; and Liu, T.-Y. 2014. Sequential click prediction for sponsored search with recurrent neural networks. In *AAAI*, 1369–1375.

Zhang, Z.; Li, C.; Wu, Z.; Sun, A.; Ye, D.; and Luo, X. 2017. Next: a neural network framework for next poi recommendation. *arXiv preprint arXiv:1704.04576*.