Predicting Destinations from Partial Trajectories Using Recurrent Neural Network

Yuki Endo, Kyosuke Nishida, Hiroyuki Toda, Hiroshi Sawada

NTT Service Evolution Laboratories, NTT Corporation,
1-1 Hikarinooka, Yokosuka-shi, Kanagawa-ken, 239-0847, Japan
endo-wop@hotmail.co.jp,
{nishida.kyosuke, toda.hiroyuki, sawada.hiroshi}@lab.ntt.co.jp

Abstract. Predicting a user's destinations from his or her partial movement trajectories is still a challenging problem. To this end, we employ recurrent neural networks (RNNs), which can consider long-term dependencies and avoid a data sparsity problem. This is because the RNNs store statistical weights for long-term transitions in location sequences unlike conventional Markov process-based methods that count the number of short-term transitions. However, how to apply the RNNs to the destination prediction is not straight-forward, and thus we propose an efficient and accurate method for this problem. Specifically, our method represents trajectories as discretized features in a grid space and feeds sequences of them to the RNN model, which estimates the transition probabilities in the next timestep. Using these one-step transition probabilities, the visiting probabilities for the destination candidates are efficiently estimated by simulating the movements of objects based on stochastic sampling with an RNN encoder-decoder framework. We evaluate the proposed method on two different real datasets, i.e., taxi and personal trajectories. The results demonstrate that our method can predict destinations more accurately than state-of-the-art methods.

1 Introduction

Mobile devices equipped with a GPS sensor enable us to easily collect location information on moving objects, known as *trajectories*. Predicting future destinations from their current trajectories is crucial for various location-based services such as personal navigation systems and ride sharing services. For example, it is more effective to deliver advertisements on sightseeing places around the destinations rather than advertisements on the current places.

To predict destinations from a partial trajectory, existing methods model the movement tendencies by using data-driven approaches and have achieved satisfactory results in some applications [7, 19, 14, 13]. In particular, they are based on relatively low-order Markov processes, which count the number of transitions of short sequences in historical trajectories to alleviate a data sparsity problem. However, the ability to learn *long-term dependencies* between a destination and long sequences towards the destination is important for accurate

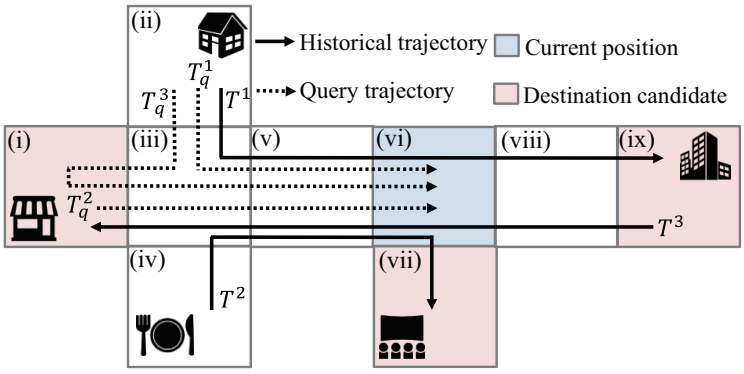

Fig. 1. Our destination prediction problem. Given a query trajectory, our method predicts top-k destinations on the basis of only historical trajectories. E.g., given T_q^1, it iterates one-step ahead predictions, starting from the cell (vi), to derive multi-step ahead predictions such as the cell (ix).

prediction because individuals move through the same area with different contexts (**challenge 1**). Figure 1 shows an example. We use a grid space because such discretized information is useful for modeling changes in trajectories. Given query trajectories T_q^1, T_q^2, and T_q^3, the task is to predict likely destinations, on the basis of historical trajectories T^1, T^2, and T^3. For example, given T_q^1, cell (ix) is the location that the user is most likely to visit. To compute this, Markov processes first calculate the transition probabilities from the current location (vi) to the subsequent locations (v), (vii), and (viii) for each query trajectory. If we assume a first-order Markov process for T_q^1, we can compute the transition probabilities $P(\mathrm{v}|T_q^1) = P(\mathrm{vii}|T_q^1) = P(\mathrm{viii}|T_q^1) = 0.33$ by counting the same transitions in the historical trajectories. Meanwhile, if we assume a second-order one, $P(\mathrm{vii}|T_q^1) = P(\mathrm{viii}|T_q^1) = 0.5$, and a fourth-order one can narrow down the candidates, that is, $P(\mathrm{viii}|T_q^1) = 1$. However, high-order Markov processes cause *the data sparsity problem* (**challenge 2**). For example, when a user departs another place, like in T_q^2, or takes a detour, like in T_q^3, $P(\mathrm{vii}|T_q^2)$ and $P(\mathrm{vii}|T_q^3)$ are not calculable in the fourth-order one. These two challenges lie in modeling the movements of specific individuals and also unknown individuals such as taxi users.

To overcome these two challenges, we exploit a recurrent neural network (RNN) [12], which can store sequential information in hidden layers. In order to model the transition information consisting of location points by using an RNN model, we represent the sequence of locations in a discretized grid space and let the model learn transitions from one cell to the next in each timestep. Different from the other count-based models, the RNN embeds sequences of sparse representations of cell locations into dense vectors consisting of statistical weights for the sequence of locations. The RNN can also handle variable lengths of trajectories without a strict built-in limit. These characteristics are useful for avoiding the data sparsity problem.

Specifically, the contributions in this paper are summarized as follows:

- Location sequence modeling for movement trajectories using an RNN architecture, which can learn long-term dependencies of trajectories as well as alleviate the data sparsity problem.
- Efficient and effective destination prediction algorithm with an RNN encoder-decoder framework using voting-based sampling simulation.
- Extensive evaluation using taxi and personal trajectories, in which our method produced overall improvements on the previous work in terms of predictive accuracy and distance error.

2 Related Work

Several studies have focused on the problem of destination prediction using external information such as trip time distribution [7, 8, 6] and road conditions [19]. Although external information is often useful to improve predictive accuracy, it is costly to obtain.

As a method that takes only trajectory data, Xue et al. [14, 13] proposed Sub-Trajectory Synthesis (SubSyn) algorithm. This algorithm first estimates every transition probability in a grid space by using sub-trajectories based on low-order Markov process. Given a query trajectory, the algorithm then estimates visiting probabilities for destination candidates by using the transition probabilities from the starting and current locations to the candidates. Although their method efficiently alleviates the data sparsity problem, modeling transitions based on low-order Markov processes is not sufficient in terms of predictive accuracy for trajectories with diverse and long movements.

Brébisson et al. [1] formulated the destination prediction as a regression problem and solved it by using a multilayer perceptron (MLP). Their method assumes that the location of a destination can be represented as a linearly weighted combination of popular destination clusters, and the weights are computed using the MLP. Because the number of dimensions of the input of the MLP must be fixed, the oldest five points and newest five points in each trajectory are fed to the MLP. While their sequence-to-point prediction can minimize the overall distance error in dense areas of the training data, accurately predicting destinations is difficult especially in areas where trajectories are sparse. Furthermore, unlike the SubSyn algorithm [14, 13] and ours, theirs is designed for predicting a single destination; it does not output multiple top k results. Although they also tried to use RNNs based on this method but it did not outperformed their MLP-based method.

Recently, another RNN-based model [9] was proposed and achieved satisfactory results in the task of the next location prediction based on check-in history. However, their model is not suitable for the multi-step prediction based on trajectory data. That is, while a next destination can be directly computed by considering a single transition for check-in data, transitions between multiple location points on the routes from a current location to a destination must be considered for trajectory data. Our focus in this paper is to efficiently predict destinations from trajectory data, which are collected at shorter intervals and enable earlier prediction than check-in data.

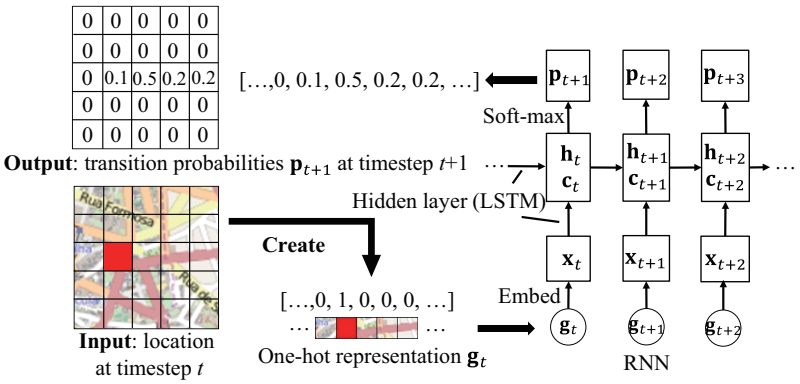

Fig. 2. Modeling transitions in trajectories using the RNN.

3 Method

Given a query trajectory $T_q = \{g_1, g_2, ..., g_c\}$ from starting time $t = 1$ to current time $t = c$, our goal is to accurately predict the probabilities $\mathbf{P} \in \mathbb{R}^{|C|}$ of visiting destination candidates $d \in C$. The trajectories are represented in a grid space, where an index g is assigned to each cell. The data available for solving this problem are only training data D consisting of historical trajectories $T = \{g_1, g_2, ..., g_{e_T}\}$ from starting time $t = 1$ to arrival time $t = e_T$, which are obtained from either unknown or specific individuals.

Note that our focus is to predict not a single destination but the visiting probabilities of multiple destinations. This is because the probabilities are useful in diverse applications. For example, location-based services can deliver multiple advertisements to users according to probability values. Additionally, car navigation systems allow users to efficiently select their destinations from candidates.

The procedure of our method is split into two phases: learning and prediction. In the learning phase, it uses historical trajectories to generate destination candidates $d \in C$ and estimates the model parameter θ of an RNN for the destination prediction. In the prediction phase, our method computes the visiting probabilities $\mathbf{P}$ from a query trajectory T_q by using the learned RNN model with θ. Section 3.1 describes our RNN model, and Section 3.2 describes our prediction algorithm based on the learned model.

3.1 Model

Architecture Figure 2 illustrates our RNN architecture. The RNN takes as input a vector $\mathbf{g}_t$, which is a one-hot representation computed from a location index g_t at timestep t. The one-hot vector $\mathbf{g}_t$ is then embedded into a relatively low-dimensional space to obtain semantic representations of a location. Next, the embedded features are fed to the hidden layers consisting of long short-term memory (LSTM) units [5, 3], which can memorize long-term sequences with variable sequence lengths. The LSTM units also take two previous state

vectors, hidden state $\mathbf{h}_{t-1}$ and cell state $\mathbf{c}_{t-1}$. Finally, the soft-max layer outputs a transition probability $\mathbf{p}_{t+1} = P(g_{t+1}|g_t, g_{t-1}, ..., g_1)$ for each grid cell while the LSTM units compute the next two state vectors $\mathbf{h}_t$ and $\mathbf{c}_t$. Although there is an alternative way that directly uses as input and output the sequences of two scalar values of raw spatial coordinates (longitude and latitude), it is difficult to train such a simple model in the RNN architecture because any prior information on the distribution of the data is not taken into account [1].

Learning We first generate destination candidates C. Given historical trajectories $T \in D$, we consider the last location point of each trajectory T as a past destination (e.g., taxi drop-off locations and stay points). The algorithm extracts the index d on the basis of the past destination for each trajectory and uses the set of the indices as the destination candidates C.

Next, we learn the model parameters of the RNN by maximizing the conditional likelihood over the set of all historical trajectories as:

$$\hat{\theta} = \arg\max_{\theta} \sum_{T \in D} \prod_{t=1}^{e_T - 1} P(g_{t+1}|g_t, g_{t-1}, ..., g_1; \theta). \tag{1}$$

This is equivalent to minimizing the cross-entropy loss between the output probability distributions $\mathbf{p}_t$ and the one-hot representations $\mathbf{g}_t$ at $t = 2, 3, ..., t_{e_T}$. To optimize θ based on Equation (1), we use truncated back propagation through time (BPTT) [20] based on mini-batch AdaDelta [15], which is more efficient than vanilla stochastic gradient decent (SGD) or full-batch optimization. In our experiments, we set the length of truncated BPTT and size of a mini-batch to 20 and 100, respectively. We also clip the norm of the gradients (normalized by mini-batch size) at 5 to deal with exploding gradients [11].

3.2 Destination Prediction

Using the RNN model with θ, our method predicts visiting probabilities $\mathbf{P}$ for C from an input query trajectory T_q. Specifically, we exploit the RNN encoder-decoder framework. First, we compute hidden states $\mathbf{h}_c$ and $\mathbf{c}_c$ at the current timestep $t = c$ by using the RNN encoder with the LSTM units, which takes as input the sequence of one-hot representations $\mathbf{g}_1$,..., $\mathbf{g}_c$ of T_q and recurrently updates the state vectors. The visiting probabilities $\mathbf{P}$ are then computed from the current hidden states $\mathbf{h}_c$ by using the RNN decoder with the soft-max layer. However, the soft-max layer only estimates transition probabilities $P(g_{c+1}|T_q)$ on the next timestep because the RNN decoder is based on sequence-to-sequence modeling. In most cases, more than one timestep is taken to get to destinations from current locations, and thus we need additional operations to obtain $\mathbf{P}$.

A naïve solution is to directly calculate transition probabilities for all transition patterns and integrate them. Given the maximum step size M for detours, which is computed from training data, the visiting probability for a destination

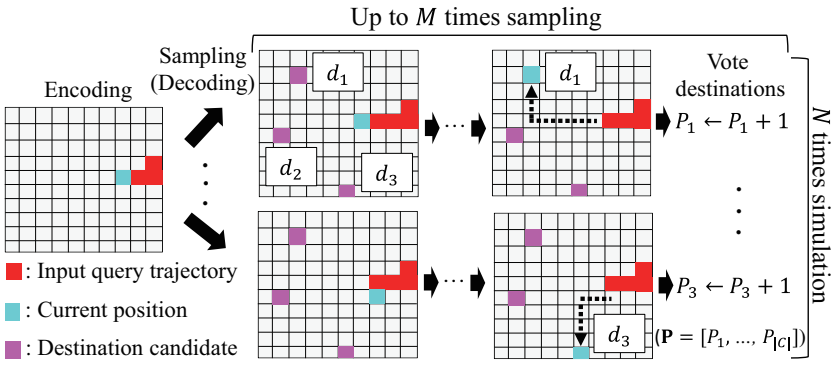

Fig. 3. Sampling simulation for destination prediction using the RNN encoder-decoder.

d of $\mathbf{P}$ is obtained as $P(g_{c+1}{=}d|T_q) + P(g_{c+2} = d|T_q) + ... + P(g_{c+M} = d|T_q)$, where $P(g_{c+m}|T_q)$ for variable m and any d is defined as:

$$\sum_{\forall g_{c+m-1}} \cdots \sum_{\forall g_{c+1}} \prod_{t=c}^{c+m-1} P(g_{t+1}|g_t, g_{t-1}, ..., g_{c+1}, T_q; \theta). \tag{2}$$

Compared with low-order Markov processes, the RNN decoder takes a large amount of time to compute $P(g_{c+m}|T_q)$ because RNN depends on longer sequences. For example, to compute $P(g_{c+M}|T_q)$ as exact as possible, we need to iterate RNN decoding $|G|^{M-1}$ times; this is impossible because $|G|$ and M are usually a few thousand and a few dozen, respectively. If we assume that users move to adjacent cells in a single transition, the computational complexity reduces to $O(8^{M-1})$, but the prediction still takes a long time.

Instead, analogous to what is done in Monte Carlo methods and word sequence generation [4], our algorithm efficiently estimates the visiting probabilities for each destination candidate. As shown in Figure 3, the algorithm simulates the movement of objects by stochastically sampling a position at the next timestep according to the transition probabilities obtained by the RNN decoder. Specifically, the algorithm first samples a position according to $P(g_{c+1}|T_q)$ and then samples a position according to $P(g_{c+2}|g_{c+1}, T_q)$. This process is repeated up to M times or until the sampling reaches one of the destination candidates. If one of the destination candidates is reached, a vote is cast on the element of $\mathbf{P}$ corresponding to the destination. After $\mathbf{P}$ is initialized to a zero vector, this sampling simulation is iterated N times. Finally, by normalizing $\mathbf{P}$, we can estimate the visiting probability distribution for each destination candidate. This procedure can be easily parallelized for each simulation step.

Considering spatial proximity In our sampling simulation, there is no constraint on the distance of a single transition in a discretized grid space. This results in the algorithm predicting wrong destinations far away from a true destination because a simulation sample may suddenly jump to a distant cell. To

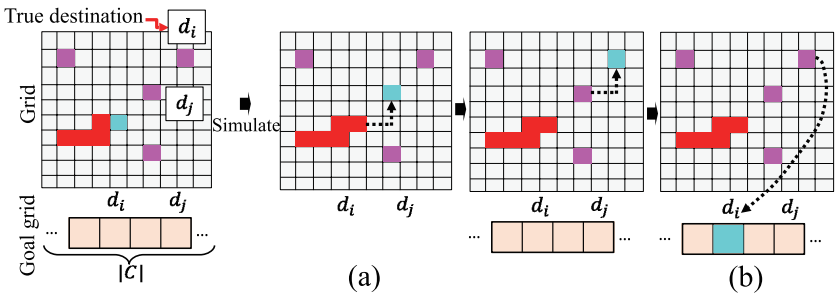

Fig. 4. Virtual goal grid cells to distinguish arrival states from goal states. If indices of destination candidates are defined on the common grid space, the sample stops at a false destination candidate on the routes to a true destination (a). If indices of goal grid cells are defined, the sample does not stop at the false destination candidates and jumps to the goal grid cell of a true destination from a nearby cell (b).

consider distance information, we control the transition probabilities on the basis of the spatial proximity of the cells. In practice, we update the transition probabilities $\mathbf{p}_t$ estimated using the RNN decoder and obtain new transition probabilities $\mathbf{p}'_t$ so that a sample does not easily jump to a distant cell:

$$\mathbf{p}'_t = \frac{\mathbf{p}_t \circ \mathbf{s}_{g_{t-1}}}{|\mathbf{p}_t \circ \mathbf{s}_{g_{t-1}}|}, \tag{3}$$

$$\mathbf{s}_g = [\exp(-\frac{\mathrm{Dist}(g,1)}{\sigma^2}), ..., \exp(-\frac{\mathrm{Dist}(g,|G|)}{\sigma^2})], \tag{4}$$

where $\circ$, $\mathrm{Dist}(\cdot,\cdot)$, and $|G|$ denote element-wise multiplication, distance (in meters) between two cells, and the number of all grid cells, respectively. σ^2 denotes the variance of the distribution. The smaller σ^2 is, the harder it is for the sampling to jump to a distant cell. We set σ^2 to $200m$ based on the size of a cell (i.e., $150m \times 150m$ as explained in the Section 4.1). σ^2 can also be automatically determined from historical trajectories.

Distinguishing arrival states from moving states If indices of destination candidates are defined on the common grid space, the sampling procedure stops when a sample reaches any of the other false destinations on the route to the true destination (Figure 4(a)). This fact may prevent the sample from getting to a true destination far away from its current place. As shown in Figure 4(b), we solve this problem by assigning destination candidates with indices of virtual goal grid cells to distinguish arrival states from moving states. The goal grid cells are connected with other common grid cells and indicate whether the sample arrives at a destination or moves through it.

4 Experiments

To validate the effectiveness of our method, we compared our method with the SubSyn algorithm [14], which is a state-of-the-art parameter-free algorithm that uses low-order Markov process modeling, and the MLP-based algorithm [1].

Table 1. Geographical ranges and statistics of the datasets.

Dataset	TST	GL
Latitude	[41.131571, 41.162477]	[39.68, 40.19]
Longitude	[-8.613876, -8.565437]	[116.05, 116.72]
# of trajectories	154,616	8,390 (in 46 users)
Avg. distance (m)	2127.4 ± 1130.0	5258.1 ± 8172.7
Avg. cell length	14.8 ± 7.0	13.3 ± 13.8

4.1 Datasets

We used a taxi service trajectory (TST) dataset [10], which contains trajectories for hundreds of taxis. Each trajectory includes sequences of latitude and longitude from a boarding location to a drop-off location. Additionally, we used a GeoLife dataset [18, 16, 17], which is publicly available dataset of personal trajectories in Beijing. We assumed that a change point of a transportation mode was a stay point and considered a segment with the same transportation mode as a single trajectory from an origin to a destination. We omitted users that had less than ten trajectories from the dataset. For our method and the SubSyn algorithm, cell size and a grid space need to be defined. We used $150m \times 150m$ cells and a grid space consisting of a part of the full dataset to reduce the high computational costs of conducting various experimental conditions. Table 1 summarizes statistics of these datasets.

4.2 Experimental settings

Evaluation measures. We used *Accuracy*@k and *Distance*@k as the evaluation measures. *Accuracy*@k indicates the ratio of destinations that are accurately predicted in a cell to all query trajectories T_q. On the other hand, *Distance*@k indicates the average distance error between the true destinations and the predicted destinations for all query trajectories T_q. We computed these measures for the top k destinations based on destination visiting probabilities $\mathbf{P}$ and used the best values in top k.

Evaluation methods. We sorted the trajectories in each dataset in ascending order of time and used the first 70% for training and the remaining 30% for test. For the TST dataset, we generated a single model for all of the trajectories to evaluate our method for unknown individuals. Meanwhile, for the GL dataset, we generated multiple models for multiple users to evaluate our method for specific individuals. In this case, we computed *Accuracy*@k and *Distance*@k for each user and averaged them. As a query trajectory T_q, we used the older α% location points in each test trajectory and took the last location point to be the ground-truth destination.

4.3 Results and analysis

In this section, we answer the research questions that correspond to the two challenges, explained in Section 1, by analyzing our experimental results.

RQ1 Can the proposed method make improvements to the destination prediction by learning long-term dependencies?

Figure 5 shows *Accuracy*@k and *Distance*@k for each dataset. As can be seen in the results for the TST dataset, our method outperformed SubSyn in terms of both *Accuracy* and *Distance* when using the same input length α. In particular, the degree of *Accuracy* improvement was remarkable when larger values of α were used (i.e., query trajectories have longer sequences). Moreover, Ours50% outperformed SubSyn70% that used a longer input. MLP, which formulates the destination prediction as a regression problem, minimizes the overall *Distance* in the training dataset; on the other hand, our method optimizes the probabilities of visiting destination cells, i.e., the *Accuracy* measure. Therefore, our method performed significantly better than MLP in terms of *Accuracy*@1. In contrast, MLP significantly outperformed our method in *Distance*@1 when $\alpha = 30\%$, but ours yielded comparable or slightly poor results to MLP when $\alpha = 50\%$ and performed slightly better than MLP when $\alpha = 70\%$. These results indicate the capability of our method for handling long-term dependencies.

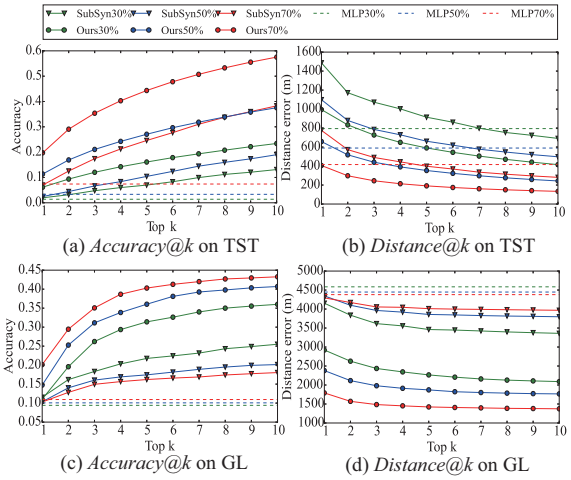

Fig. 5. Overall performance of destination prediction.

Our method worked much better than the other methods on the GL dataset, especially for larger values of α. One reason for the improvements is that our method can capture personal routines based on long sequences of trajectories. Meanwhile, larger values of α worsened the predictive accuracy and distance error of SubSyn in contrast to the results of Ours and MLP. In the GL dataset, there are several frequent origins (e.g., home), and the current location of a query trajectory will be distant (near) from such origins when α is large (small). That is, larger values of α seemed to decrease the number of training transitions between the current and destination locations although SubSyn needs the transition probabilities between current and destination locations.

RQ2 Can the proposed method alleviate the data sparsity problem?

Figure 6 shows performance for each method with different training data sizes on the TST dataset. Despite modeling longer sequences, our method often worked well even when the number of historical trajectories for learning was small. This is because our RNN stored the statistical weights of variable-length trajectories instead of counting fixed-length transitions.

Additionally, as shown in Table 1 and Figure 5, although the GL dataset had a smaller number of trajectories than the TST dataset, the improvements of our method over the other methods on it were larger than on the TST dataset. In this case, MLP performed worst because it faced the data sparsity problem when determining the density-based destination clusters and when learning the sequence-to-point relation. Although SubSyn performed slightly better than MLP thanks to their sub-trajectory approach, Ours performed best because our sequence-to-sequence RNN model can cope with a small dataset more effectively.

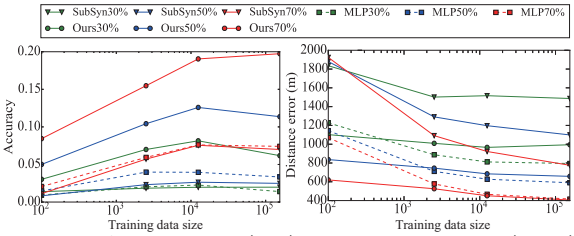

Fig. 6. *Accuracy*@1 (left) and *Distance*@1 (right) with different training data size in the TST dataset.

Computational time Figure 7(a) shows the time needed to make a prediction for a query trajectory. We parallelized the sampling simulation using 10 processes on the CPU. In this figure, computational times linearly increase depending on the number of sampling simulations N, except when $N = 10$ because most time was taken for overhead costs of the parallelization. In particular, the prediction takes only a few seconds when N is a few hundred. Figure 7(b) shows *Accuracy* depending on N in 2,500 trajectories sampled from the TST dataset. As can be seen, while a large N tends to give better performance, the gain in accuracy begins to level off after $N = 100$. As for the learning time, we took a few days worth of data from the TST dataset, and took about a half hours worth for one user of the GL dataset. Although the learning time exceeded the prediction time, this does not matter much in real application services because the model can be learned in advance.

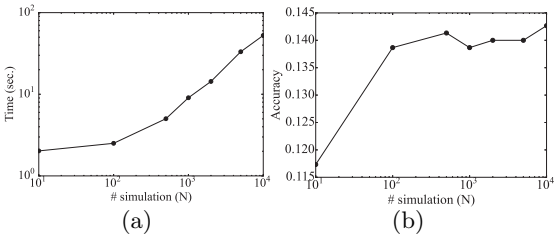

Fig. 7. (a) Prediction time and (b) *Accuracy*@1 depending on the number of simulation N.

Parameter sensitivity We evaluated the sensitivity of our method to variations of the cell size using the GL dataset as shown in Tables 2. For the grid-based methods (Ours and SubSyn), the cell size is important to determine the granularity of predicted destinations. The larger cell, the more difficult it is to identify the true location of a destination in a cell. That is, the large cell increases *Accuracy* while it decreases the resolution of predicted locations, which affects *Distance*. Nevertheless, ours outperformed the existing methods for every cell size in terms of both metrics. For MLP, *Distance* did not depend on the cell size because it directly predicts a location of a specific destination. When the cell size was small (the number of cells was large), SubSyn could not predict

Table 2. Performance of each method with different cell sizes when $\alpha = 50\%$.

Cell width	5000m	1200m	150m	40m
MLP (Accuracy@1)	0.466	0.268	0.101	0.012
SubSyn (Accuracy@1)	0.506	0.310	0.103	N/A
Ours (Accuracy@1)	**0.618**	**0.408**	**0.147**	**0.029**
MLP (Distance@1)	4448	4448	4448	4448
SubSyn (Distance@1)	4376	4247	4346	N/A
Ours (Distance@1)	**3427**	**2401**	**2380**	**2647**

destinations because its orders of computational complexity and memory space were $O(|G|^{3.5})$ and $O(|G|^{2.5})$, where $|G|$ is the number of cells.

For the network structure, we evaluated our method with other parameters (128-1024 LSTM units and 1-3 RNN layers). In the results, there were no significant differences in performance between them.

Comparison with other possible approaches We also validated our individual approaches (i.e., sampling simulations (SS), spatial proximity (SP), and goal cells (GC) described in Section 3.2). Table 3 compares the performance of the methods with and without these approaches using 2,500 trajectories sampled from the TST dataset. The results demonstrated that each of these approaches improved the accuracy and distance metrics.

Table 3. Performance of Ours and other possible approaches when $\alpha = 50\%$.

Method	Ours	Ours w/o SS	Ours w/o SP	Ours w/o GC
Accuracy@1	**0.141**	0.051	0.115	0.050
Distance@1	**701**	1200	939	770

5 Conclusion

We proposed a method of predicting destinations from partial trajectories. Our method represents trajectories as sequences of one-hot representations on a grid space and explicitly learns transitions of objects by using an RNN model that stores statistical weights of sequential information in its hidden layers. This enables us (i) to model long-term dependencies while (ii) avoiding the data sparsity problem. Additionally, our method efficiently predicts destinations using a stochastically sampling simulation based on the RNN encoder-decoder framework. We conducted evaluation experiments using two different datasets and demonstrated that our method was effective for both unknown individuals and specific individuals.

Limitations and future work. The computational time for learning the RNN model linearly increases with the number of grid cells $|G|$, and thus it seems to be difficult to target a huge area with fine grids. Although we did not use finer grids with $150m \times 150m$ cells in our experiments, our method performed in large areas such as downtown in Beijing. A simple solution to expand a target area would be to divide a single RNN model that covers a huge area into multiple RNN models that cover a small area.

Another challenge is to incorporate time information into our method. Time information is helpful for predicting user activities; e.g., a user goes to an office

in the morning and a bar in the evening. Multi-view models [2] for handling such information in trajectories would thus be an interesting topic of future work.

References

1. A. de Brébisson, É. Simon, A. Auvolat, P. Vincent, and Y. Bengio. Artificial neural networks applied to taxi destination prediction. *CoRR*, abs/1508.00021, 2015.
2. A. M. Elkahky, Y. Song, and X. He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *WWW*, 278–288, 2015.
3. F. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *ICANN*(2), 850–855, 1999.
4. A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.
5. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.
6. E. Horvitz and J. Krumm. Some help on the way: Opportunistic routing under uncertainty. In *UbiComp*, 371–380, 2012.
7. J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *UbiComp*, 243–260, 2006.
8. J. Krumm and E. Horvitz. Predestination: Where do you want to go today? *Computer*, 40(4):105–107, April 2007.
9. Q. Liu, S. Wu, L. Wang, and T. Tan. Predicting the Next Location: A Recurrent Model with Spatial and Temporal Contexts *AAAI*, 194–200, 2016.
10. L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas. Predicting taxi-passenger demand using streaming data. *Intelligent Transportation Systems, IEEE Transactions on*, 14(3):1393–1402, Sept 2013.
11. R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 1310–1318, 2013.
12. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, 696–699. 1988.
13. A. Y. Xue, J. Qi, X. Xie, R. Zhang, J. Huang, and Y. Li. Solving the data sparsity problem in destination prediction. *The VLDB Journal*, 24(2):219–243, Apr. 2015.
14. A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang, and Z. Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *ICDE*, 254–265, 2013.
15. M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.
16. Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma. Understanding mobility based on GPS data. In *UbiComp 2008*, 312–321, 2008.
17. Y. Zheng, X. Xie, and W. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.
18. Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *WWW*, 791–800, 2009.
19. B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *UbiComp*, 322–331, 2008.
20. D. Zipser. Advances in neural information processing systems 2. chapter Subgrouping Reduces Complexity and Speeds Up Learning in Recurrent Networks, 638–641. Morgan Kaufmann Publishers Inc., 1990.