Department of Informatics, University of Zürich

MSc Basic Module

Implementation of Attentional Recurrent Neural Network for Human Mobility Prediction

Shaoyan Li

Matrikelnummer: 20-741-278

Email: shaoyan.li@uzh.ch

June 27, 2021

supervised by Jamal Mohammed

University of Zurich[UZH]

Department of Informatics

1 Introduction

Human mobility prediction is of great importance for a lot of location based applications. Human mobility is defined by human's trajectory. Trajectory is a sequence with points containing information about time and location. To be precise, it refers to the path followed by user's move with fixed temporal resolution. Fixed temporal resolution is defined by the time consumed when a person moves from current location to next adjacent location (he/she could still stay in the same location). So in a dedicated trajectory, time interval implied in either two adjacent locations is the same since temporal resolution is fixed. Assume there is a trajectory array [1, 2, 3, 4, 6, 2] defined in Figure 1.1, which contains six symbolized locations, indicating the user once visited these six places in time order. Regarding symbolized location, it depends on how high spatial resolution is applied. There are nine black bordered boxes in Figure 1.1 and each one represents a symbolized location. However, symbolized location could change if the size of box changes. For instance, after restricting location 1 to the blue box, some parts of former location 1 is no longer location 1. Spatial resolution is defined as the size of one box representing how large a symbolized location is. The larger box is defined, the higher spatial resolution is. When applying different temporal resolution and spatial resolution, it's possible to get a different trajectory. Assume the trajectory in Figure 1.1 is obtained with a temporal resolution A. If applying a lower temporal resolution B, it's possible to get a trajectory array like this: [1, 1, 1, 1, 1, 1]. In this case, starting from location 1, the user's position is computed every fixed time interval corresponding to temporal resolution B. Since with a rather lower temporal resolution, the user may not be able to move to another location within a very short time frame so he/she still stays in the same location.

Human mobility prediction problem is proposed to predict future human trajectory based on current and history information under given fixed temporal resolution and spatial resolution. The metric to evaluate prediction is the accuracy, which is defined as the degree of predicted and real trajectory matching.

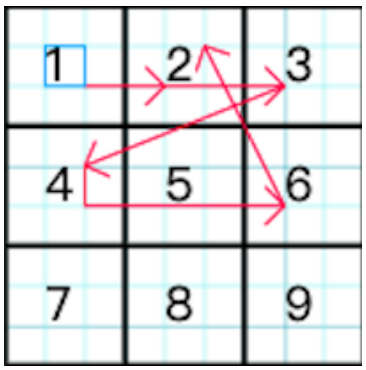


Figure 1.1: trajectory array [1, 2, 3, 4, 6, 2]

Regarding this topic, two different algorithms, DeepMove and MobilityUpperBoundPrediciton are discussed in the report. DeepMove is an algorithm that utilizes deep learning knowledge to predict human mobility, while MobilityUpperBoundPrediciton uses a refined method to gives the upper bound of predicted accuracy based on one's history trajectory. Reimplementation of repository [1] using DeepMove [2] and repository [3] using MobilityUpperBoundPrediciton [4] by pure Python is also included in this report. The goal this report intends to achieve is to experimentally evaluate the implementation in this work by comparing the results to the ones published in the original papers. Additionally, it's also possible to verify whether DeepMove is a feasible algorithm to perform human mobility prediction by applying cross-dataset experiment on these two datasets that are included in corresponding papers.

2 Approaches

2.1 DeepMove

2.1.1 The architecture of DeepMove algorithm

The architecture of DeeMove algorithm is shown in Figure 2.1. The intuition of DeepMove algorithm is based on Multi-level periodicity of human mobility: daily routines, weekend leisure , yearly festivals, and even other personal periodic activities, which indicates a user's mobility not only relies on sequential effect, but also relies on historical effect.

According to inferences above, DeepMove algorithm uses symbolized current and history trajectory as inputs. These trajectories contains sparse features of symbolized location, time and user ID. Then, the embedding layer is applied to extract features of the input and embed them densely in embedding layer. These features are combined into one input by linear and concatenate layer.

Afterwards, the algorithm uses an essential module-recurrent layer. Recurrent Neural Network is a class of neural networks with cycle and internal memory units to capture sequential information. Long short-term memory (LSTM) and gated recurrent unit (GRU) are widely used recurrent units. LSTM consists of one cell state and several controlled gates to keep and update the cell state. Based on the input and last cell state, LSTM first updates the cell state with parts to keep and parts to drop. Then, LSTM generates the output from the cell state with learnable weight. GRU is a popular variant of LSTM which replaces the forget gate and the input gate with only one update gate. The recurrent neural network can capture long-range dependencies of sequential information.

However, when the sequence is too long, the worse performance the prediction achieves. Thus, with the recurrent neural network, it's only effective to process limited length trajectory with a short duration of one day or even shorter. Based on the above theories, an attentional recurrent neural network is proposed for predicting human mobility from the lengthy, periodical and incomplete trajectories.

The attentional layer is designed to capture the multi-level periodical nature of human mobility by jointly selecting the most related historical trajectories under the current mobility status. The historical attention module first extracts spatiotemporal features from the historical trajectories by an extractor. These features contains information of location and history (time). Then, these features are selected by the current mobility status based on the spatiotemporal relations to generate the most related context.

Finally, the algorithm proposes prediction module, which is the final component that combines the context from different modules to complete the prediction task. It consists of a concatenate layer and some fully-connected layers. The concatenate layer combines all the

features from the historical attention module, recurrent module, and embedding module into a new vector. Following the concatenate layer, fully connected layers further process the feature vector into a more expressing vector. The final output is processed with a soft-max layer with negative sampling.

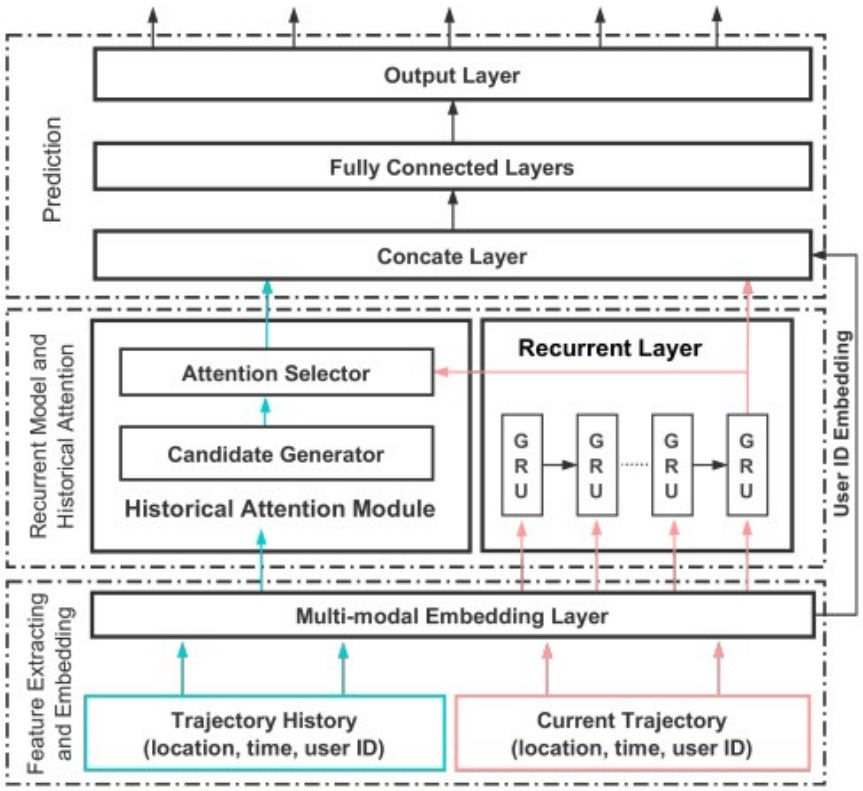


Figure 2.1: the architecture of DeeMove

2.1.2 The datasets and data preparation process

The data contains multiple trajectories from multiple users, which is stored in foursquare.pk file. To extract data, the method pickle.load() is used, from which a nested Python Dictionary in Figure 2.2 is obtained. The essential keys of the dictionary are vid_list, uid_list and data_neural. Both values, data[vid_list] and data[uid_list] corresponding to vid_list and uid_list, are Python Dictionarys as well. The length of key list of data[vid_list] represents the number of distinct symbolized locations, while the length of key list of data[uid_list] represents the number of users.

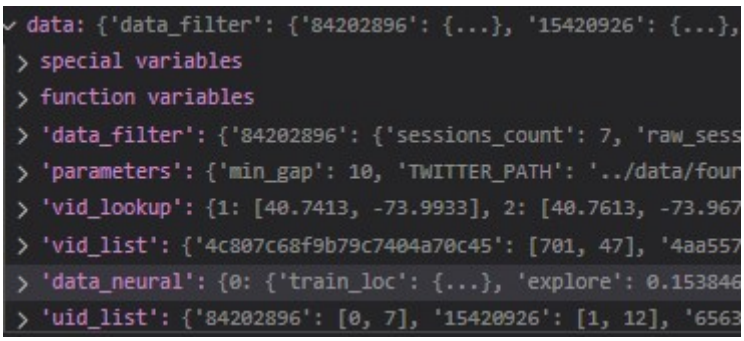


Figure 2.2: the structure of data

Then the structure of data_neural is shown in Figure 2.3. The keys of data_neural represent indexes of users. And the value of each key stores all information of trajectories with regard to that user, which would be used in the implementation as raw data.

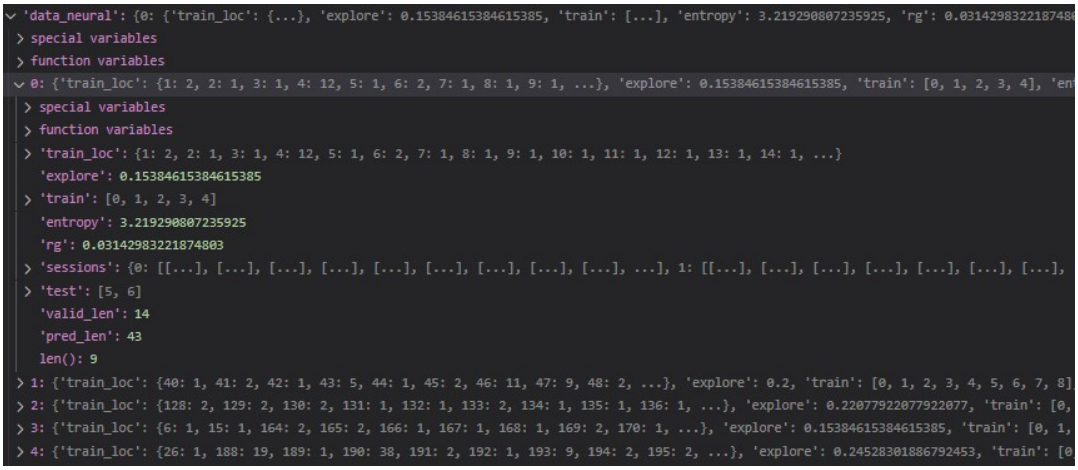


Figure 2.3: the structure of data_neural

From Figure 2.3, it's easy to know data_neural[user_index] is also a Python Dictionary. For instance, assume user 0 is selected and it's shown in Figure 2.3 that data_neural[0] has several keys. However, only train, test and sessions are used. In Figure 2.4, the (key: value) pairs in data_neural[0][sessions] can be interpreted as (trajectory index: trajectory content), so user 0 has 7 different recorded trajectories. Take the first trajectory of user 0 for example, the trajectory is a nested list which consists of many two-element lists. The first element is the index of symbolized location and the second element is the time slot. In this dataset, time slots are divided into 48 equal portions in one day. If user 0 is at location 1 in time slot 30, a two-element list [1, 30] is generated and added to a trajectory. So that's the way the trajectory is recorded. The value of key train is a list, which represents index number of the session in sessions. These sessions would be applied to training set. In the same way, the value of key train could be referred as index number of the session in sessions which are applied to test set.

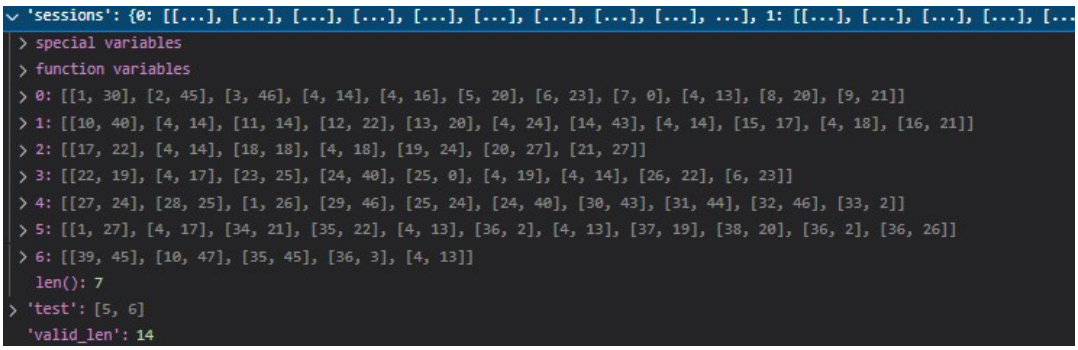


Figure 2.4: the structure of data_neural[0][sessions]

The dataset has been pre-processed in this paper, so it's easy to apply corresponding data to different models, which will be introduced precisely in Implementation section. For instance, as for simple model, these data is enough for computing. However, with respect to attn_local_long model, it's also necessary to generate some history information using existed data. Since inputs are different under different models, the algorithm should preform different predictions.

2.2 MobilityUpperBoundPrediction

2.2.1 The process of MobilityUpperBoundPrediction algorithm

In some research, it is claimed that human movement is predicted based on history information and the ideal prediction accuracy is given. The paper is based on these research and considers real-world constraints that are able to achieve a tighter upper bound, representing a more refined limit to the predictability of human movement.

The process of this algorithm consists of data pre-process, calculating entropy and calculating upperbound prediction probability.

Step 1: It's essential to preprocess the data from Geolife. This step aims to get useful trajectories based on raw Geolife data and specified spatial and temporal resolution. In the paper, spatial resolution 1000, 5000, 10000, 50000, 100000, 1000000, 3000000, 10000000, 50000000 m^2 and temporal resolution '0:05:00', '0:10:00', '0:15:00', '0:30:00', '0:45:00', '1:00:00' (hour:min:sec) are used. This part would be elaborated in the following report.

Step 2: Then, after obtaining trajectory data from preprocessing, the paper calculates the entropy of each trajectory. Here, the entropy rate is calculated by:

$$entropy\_rate = \lim_{t\to\infty} \frac{1}{t} H(X_t|X_{t-1}, X_{t-1}, ..., X_1) \tag{2.1}$$

In this equation, t represents time, random variable X_t represents all possible locations which the user may stay in at time t, and $H(\chi)$ denotes the entropy rate of a stochastic process. However, this equation is not feasible in calculation since time is limited in a dedicated trajectory. So the implementation uses an alternative equation below for estimation, which

converges to Equation 2.1 quickly.

$$est\_entropy\_rate = \frac{N}{\sum_{i=1}^{N} \frac{s_i}{log(i+1)}} \tag{2.2}$$

In Equation 2.2, s is maximum matched sequence which has the same length as the trajectory and s_i represents the i-th element of s. N is the length of sequence s. The computation of maximum matched sequence s is as follows. Assume there is a user trajectory array t [1, 2, 3, 1, 2, 3]. Sequence s actually computes the maximum length of sub-trajectory which is observed for the first time from current point compared with history sub-trajectory (all points before current one). For instance, to compute s_1, it can be observed that $t_1 = 1$ and $s_1 = 1$ since there is no point before t_1 so sub-trajectory [1] is observed for the first time compared with empty. For the same reason, $s_2 = 2$ and $s_3 = 3$ can be computed. However, when computing s_4, since [1], [1, 2], [1, 2, 3] has shown before, then $s_4 = 0$. So on and so forth, s is computed as [1, 2, 3, 0, 0, 0]. Afterwards, it's simple to get entropy rate applying Equation 2.2.

Setp3: The paper calculates the prediction probability by:

$$H_F(\pi) = -\pi \log_2 \pi - (1 - \pi) \log_2 \frac{1 - \pi}{N - 1} \tag{2.3}$$

In Equation 2.3, $H_F(\pi)$ represents the entropy of a trajectory, and π denotes upper bound of prediction probability range from 0 to 1, which is the algorithm for. N represents the number of distinct locations or reachable locations in this trajectory. In former work, standard method defines N as the number of all possible locations, while in this work, refined method defines N as reachable positions. Reachable means the user is possible to reach some other locations from one dedicated location in the trajectory.

After calculating the possibility from each trajectory, final prediction possibility can be computed by averaging all individual possibility.

2.2.2 The datasets and data preparation process

Raw data is Geolife dataset, which is a open dataset downloadable via Microsoft. Every single folder of this dataset stores a user's GPS log files, which were converted to PLT format. Each PLT file contains a single trajectory and is named by its starting time. PLT format is as follows.

PLT format: Line 1 to 6 are useless in this dataset, and can be ignored. Points are described in following lines, one for each line.

Field 1: Latitude in decimal degrees.
Field 2: Longitude in decimal degrees.
Field 3: All set to 0 for this dataset.
Field 4: Altitude in feet (-777 if not valid).
Field 5: Date - number of days (with fractional part) that have passed since 12/30/1899.
Field 6: Date as a string.
Field 7: Time as a string.

Take two data item from one trajectory for example:

39.906631,116.385564,0,492,40097.5864583333,2009-10-11,14:04:30
39.906554,116.385625,0,492,40097.5865162037,2009-10-11,14:04:35

In this paper, only latitude, longitude and time are concerned. Given spatial resolution, latitude and longitude, it's simple to get numbers of healpix pixels through function healpy.pixelfunc .ang2pix(), which indicates that a spatial resolution defines an area which composes numbers of pixels. The process of generating specified symbolized location is the same as what is discussed in Introduction section. Different numbers of pixels correspond to different locations. Then, with comparison of temporal resolution and time consumed between two data item, it's easy to decide where the user stays after a temporal resolution. For instance, if temporal resolution is greater than time interval between current data item and next adjacent data item, it's necessary to iterate data item until there is a data item whose time interval with current one is greater than time resolution, and next location in trajectory is computed by healpy.pixelfunc.ang2pix() using information of this data item. That is how a trajectory is generated.

3 Implementation

3.1 DeepMove

The original implementation is based on Pytorch, while Tensorflow, another popular deep learning framework, is used in this work. To verify a algorithm, the framework of implementation is not that important compared with the process of the algorithm. Following the architecture of the algorithm, it should be able to implement it using difficult approaches.

But things become a little different when tackling with details. For example, Pytorch is good at processing data one by one using iteration, while Tensorflow experts in dealing with pipeline inputs. Thus, it's significant to use suitable data flow. Therefore, it's necessary to convert inputs to other formats when using Tensorflow. Here DeepMove dataset can be encapsulated in a python generator , which is a kind of input pipelines for Tensorflow.

Regarding predicting models, it's possible to build four distinct models based on this paper, namely, simple, simple_long, attn_local_long and attn_avg_long_user. The discrepancy of these models lies in different inputs. For simple model, inputs are only current trajectories. Simple_long model, has similar inputs compared with simple model, however, the trajectories in simple_long are much longer than these in simple model. With regard to attn_local_long and attn_avg_long_user models, history trajectories are generated and added to trajectories using current information, so the attentional layer is implemented in these two models. The difference between these two models is that in attn_local_long model, more history information is used while in attn_avg_long_user model, more user information is applied. There are two main ways to build a Tensorflow model, high-level Functional API and subclass method respectively. The reason why the functional API is not chosen is because the input is a numpy array with variable size, then the first dimension of the input in Tensorflow is "None", which refers that the framework doesn't know the exact number of data in each input. And it's rather difficult to create an attentional layer with built-in functions if the exact shape of input varies each time. With another more complicated method, models based on human mobility prediction are created successfully via subclass. When some proper data flow is fed to a specific model, call() function is called and predicted trajectory is given.

Another important point that need to stress out is self-built attentional layer, which is borrowed from Practical Pytorch in original code. In this work, based on the same principle, it's not hard to build own self-attentional layer by the conversion of numpy and tf variable. First, it's simple to build tf tensors with history trajectories. Then it's necessary to convert tf tensors to numpy arrays, and perform inner or outer product on np arrays, finally result arrays are assigned to tf variables. Thus with tensor input and tensor output, the self-attentional layer is built, which aims to extract most relative location in history trajectories and be used for prediction. Besides, to plot a designated user's predicted and real trajectory, it's convenient

to get predicted positions and real positions from models and original dataset and plot them using matplotlib.

3.2 MobilityUpperBoundPrediciton

The original implementation is based on Python, C++ and Matlab. And here's a way to translate it to pure Python. First, EntropyCalc function which is written by C++ could be translated to Python. This function aims to find the longest matched sequences in a full history trajectory, which is easy to reproduce using nested for loop, except for GPU acceleration that's not important in this project. Within this step, s in Equation 2.2 is obtained. Afterwards, vpasolve which is used for calculating the possibility is substituted with scipy.optimize.fsolve, which performs similar functionality as above. Since possibility lies between 0 and 1, estimated root takes values from 0.05 to 0.95 with the step of 0.05 in an iteration. After some time, the estimated solution to the equation could be found. The root of the equation should be the upper bound probability for the trajectory.

3.3 Cross-dataset implementation

Applying DeepMove dataset to UpperBoundPrediciton algorithm, since the dataset has already been symbolized, it's simple to apply it directly to the algorithm. However, things become a little different when applying UpperBoundPrediciton dataset to DeepMove dataset. As Geolife dataset is not symbolized, it's inappropriate to use this dataset directly on DeepMove algorithm. But data could be symbolized when preprocessing using GeolifeSymbolisation.py, it's easy to get symbolized data from pre-built database.

However, when feeding symbolized data to DeepMove algorithm, the shape of fullyconnected layer is so huge that it needs to much memory size to run, which is impractical. After debugging, it seems the real value of symbolized position is much larger than the real number of distinct positions. Thus, re-index these symbolized positions from 0 to reduce memory usage works.

4 Results

For DeepMove [2] algorithm, there are four different models implemented. Regarding simple and simple_long model, the accuracy obtained from the trained model is a little lower than the original ones. However, higher accuracy with attn_local_long model and attn_avg_long_user model compared with original ones is achieved.

For MobilityUpperBoundPrediciton [4] algorithm, similar results with original ones are obtained after translating Matlab and C++ code to Python. The only difference lies in the number of users in failed mask. Failed mask refer to the list containing users whose data is not supposed to be included in calculation.

Also, applying cross-datasets to these two algorithm is performed. The answer is that DeepMove is a algorithm which could predict human mobility, especially with attn_local_long model. However, the algorithm could improve further or there exists some other algorithms which work better.

4.1 The accuracy of four models using DeepMove

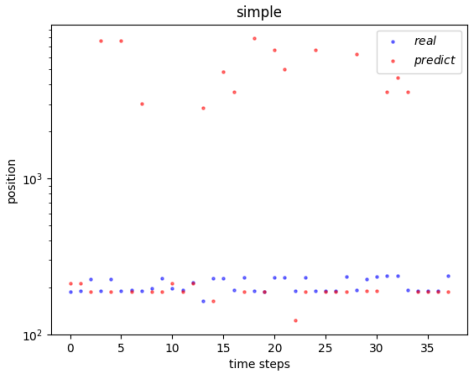
Below there is a table listing predicting accuracy of different models from DeepMove [2] algorithm.

Table 4.1: Accuracy table with some specified parameters

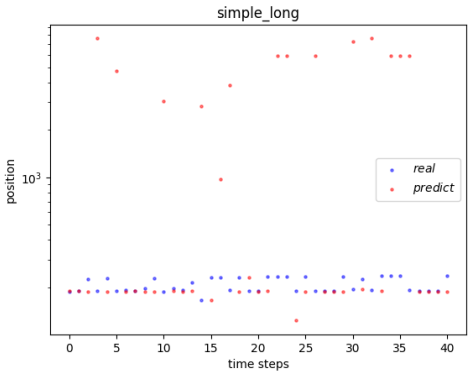
model_mode	L2	attn_type	clip	dropout	hidden_size	learning_rate	loc_size	rnn_type	tim_size	uid_size	original_acc	my_acc
markov											0.082	0.082
simple	1.00E-06	dot	5	0.3	500	0.0001	500	LSTM	10	40	0.09587167	0.082337454
simple_long	1.00E-05	dot	5	0.5	200	0.0007	500	LSTM	10	40	0.117923069	0.082788173
attn_avg_long_user	1.00E-05	dot	5	0.2	300	0.0007	100	LSTM	10	40	0.133689175	0.135096371
attn_local_long	1.00E-06	dot	2	0.6	300	0.0001	300	LSTM	20	40	0.145342384	0.150585050

Regarding simple and simple_long model, the accuracy of trained models is a little lower than the original ones. However, higher accuracy with attn_local_long model and attn_avg_long_user model compared with original ones are obtained. The difference may result from the different process for initialization since Pytorch and Tensorflow use different parameters and methods to initialize layers.

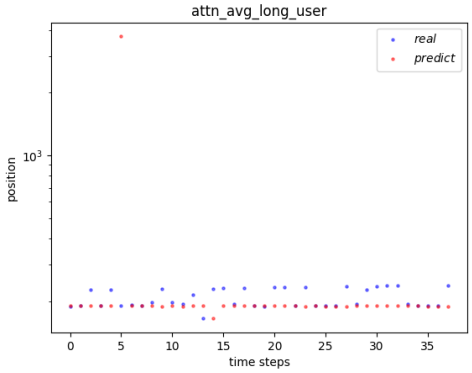
4.2 Plotting user's trajectory using DeepMove



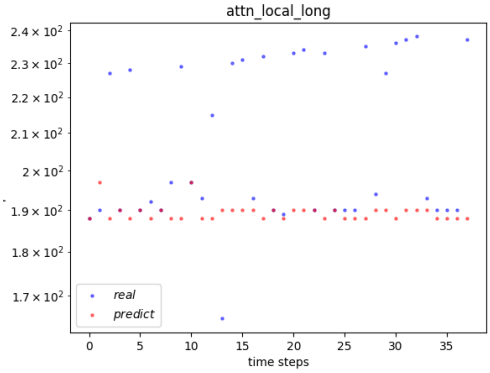
(a) the trajectory using simple model



(b) the trajectory using simple_long model



(c) the trajectory using attn_avg_long_user model



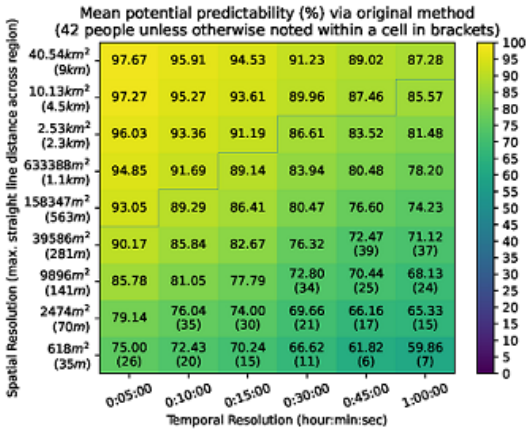
(d) the trajectory using attn_local_long model

Figure 4.1: the trajectory of user 4

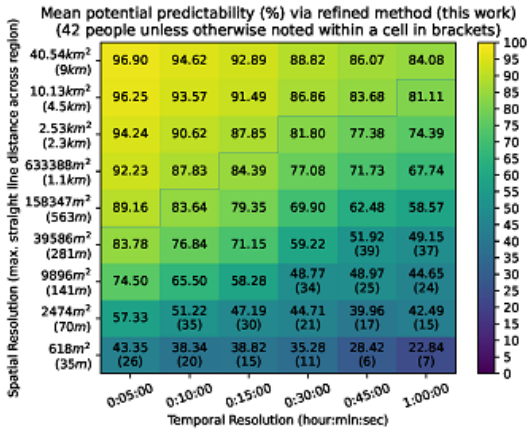
It's simple to plot the trajectory of any user (valid from 0 to 885) to observe discrepancy between predicted and real trajectory. For instance, user four's trajectory is plotted with different models as follow. So how to explain the results? For instance, in simple model, red and blue points matching refers that the algorithm predicts correctly. However, if red and blue points don't match, the algorithm has a wrong prediction. Besides, the distance between blue and red pairs at each time step doesn't refer to the distance between the pairs. For example, the distance between (blue: 1, red: 10000) doesn't need to be farther than that of (blue:1, red: 2) since the locations have been symbolized. If a user stays in one place for a long time, that place should be a regular place he/she always visits, such as home and school. However, if he/she visited one place only once, then that place may be a location in another city where he/she doesn't live.

4.3 The accuracy of standard and refined methods using MobilityUpperBoundPrediction

Below there are several figures to show the accuracy of original results in this work. In each sub-graph, x-axis represents temporal resolution, y-axis represents spatial resolution and the number in each cell is prediction accuracy possible to get. The information in these graphs shows the higher spatial resolution and lower temporal resolution are, the higher accuracy could be achieved, which is intuitive, since one can't walk out of a large area within a very short time. So he/she could only travel among just a few different locations, which is predictable. However, if a trajectory is defined with a low spatial resolution and a high temporal resolution, it's impossible to get a very precise prediction.

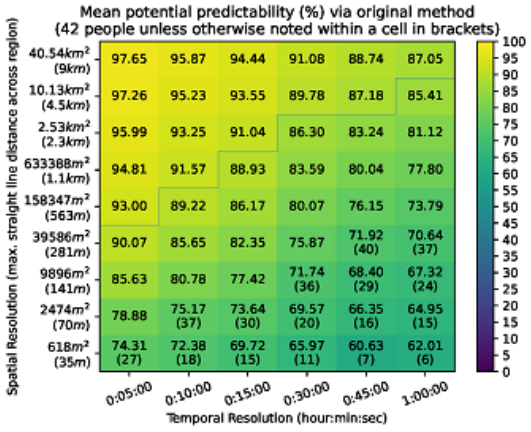


(a) the accuracy using standard method

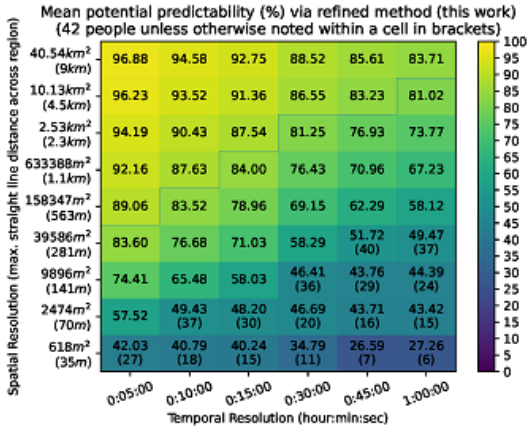


(b) the accuracy using refined method

Figure 4.2: the accuracy using author's code



(a) the accuracy using standard method



(b) the accuracy using refined method

Figure 4.3: the accuracy using modified code

It can be observed that the results in this work and original one are similar. The only difference lies in the number of user in failed mask, which means there exists some bias when calculating the entropy, which may cause by the precision of empirical entropy rate.

4.4 Applying DeepMove dataset to the refined method on upper bound predictability

	training set
number of failed personIDs	355
possibility using DL method	0.4857
possibility using RL method	0.3307

Table 4.2: applying DeepMove dataset to DL and RL methods

With DeepMove dataset, the highest accuracy an algorithm could get ideally is 33.07% when using the refined method. However, compared with the result obtained from subsection 4.1, it could be found that DeepMove's accuracy is less than half of upper bound accuracy, which doesn't work out perfectly.

4.5 Applying Geolife dataset to DeepMove algorithm

	dataset1	dataset2
possibility using DL method	0.9765	0.6201
possibility using RL method	0.9688	0.2726
possibility using DeepMove	0.8750446	0.04101216

Table 4.3: applying Geolife dataset to DeepMove algorithm

dataset1: spatial resolution 40.54km^2 with temporal resolution 0:05:00
dataset2: spatial resolution 618m^2 with temporal resolution 1:00:00

DeepMove algorithm performs very precise with dataset1 since the user always stays in the same symbolized position with a high spatial resolution and a low temporal resolution. For the same reason, DeepMove algorithm performs bad with dataset2 because the area of one symbolized location is small, after a long time, the user could travel through several locations, thus the trajectory is less predictable in this case.

5 Conclusion

When implementing code in practice, some difficulties are encountered. First, to implement DeepMove code, after building RNN cell manually, gradients were very difficult to be computed manually since the size and depth of this neural network is huge with great complexity. Besides, gradients are even harder to compute after adding embedding layers. In this case, Tensorflow Framework was chosen to reduce manual work and make implementation feasible , which makes life easier. Another challenge was that EntropyCalculation function in repository [3] was incapable to be installed from the author's instruction as the project is legacy and some features in Python2.7 are deprecated in Python3. Fortunately, source code of that function written in C++ is found [5]. Thus, it's feasible to translate it to Python3.

Human mobility prediction is rather difficult at least in this work since the highest accuracy obtained from DeepMove is lower than 20%. It could be concluded from section 4 that the accuracy of this algorithm depends heavily on the temporal and spatial resolution. Nevertheless, DeepMove algorithm sheds light on how to cope with the problem. Besides, the refined method restricts the upper bound an algorithm could achieve. Therefore, it's possible to invent an algorithm whose accuracy is close to the upper bound.

Bibliography

[1] Deepmove. [Online]. Available: https://github.com/vonfeng/DeepMove

[2] J. Feng, Y. Li, C. Zhang, F. Sun, F. Meng, A. Guo, and D. Jin, "Deepmove: Predicting human mobility with attentional recurrent networks," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, pp. 1459–1468. [Online]. Available: https://doi.org/10.1145/3178876.3186058

[3] Mobilitypredictabilityupperbounds. [Online]. Available: https://github.com/gavin-s-smith/MobilityPredictabilityUpperBounds

[4] G. Smith, R. Wieser, J. Goulding, and D. Barrack, "A refined limit on the predictability of human mobility," in *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2014, pp. 88–94.

[5] Entropyrateest. [Online]. Available: https://github.com/gavin-s-smith/EntropyRateEst