Predicting Taxi Destination by Regularized RNN with SDZ

Lei ZHANG†a, Member, Guoxing ZHANG†b, Zhizheng LIANG†, Qingfu FAN†, and Yadong LI†, Nonmembers

SUMMARY  The traditional Markov prediction methods of the taxi destination rely only on the previous 2 to 3 GPS points. They neglect long-term dependencies within a taxi trajectory. We adopt a Recurrent Neural Network (RNN) to explore the long-term dependencies to predict the taxi destination as the multiple hidden layers of RNN can store these dependencies. However, the hidden layers of RNN are very sensitive to small perturbations to reduce the prediction accuracy when the amount of taxi trajectories is increasing. In order to improve the prediction accuracy of taxi destination and reduce the training time, we embed surprisal-driven zoneout (SDZ) to RNN, hence a taxi destination prediction method by regularized RNN with SDZ (TDPRS). SDZ can not only improve the robustness of TDPRS, but also reduce the training time by adopting partial update of parameters instead of a full update. Experiments with a Porto taxi trajectory data show that TDPRS improves the prediction accuracy by 12% compared to RNN prediction method in literature[4]. At the same time, the prediction time is reduced by 7%.

key words: taxi destination prediction, recurrent neural network, surprisal-driven zoneout

1. Introduction

With the increasing amount of data, the number of taxis in big cities increases rapidly. It is important to predict the taxi destination for urban transportation planning. It is possible to predict the destination as most taxis with global positioning system (GPS) produce trajectory data. Predicting the taxi destination accurately can help the transport dispatch system in its operational planning [1]. To our knowledge, Markov models and neural networks are two methods used to predict taxi destinations. A.Y. Xue et al. [2] used the lower order Markov method to predict the taxi destination. M. Zhang et al. [3] considered the influence of time and proposed a time feature based on the Markov model for trajectory destination prediction. But, the Markov model relies only on the first 2 or 3 GPS points and can not solve such trajectories very well when the number of interdependent points in a trajectory is large. However, long-term dependencies between the destination and long trajectory is critical to predict the destination accurately. In order to solve long-term dependencies, A.D. Brébisson et al. [4] and Y. Endo et al. [5] used the recurrent neural network (RNN) to predict the taxi destination as the hidden layer of RNN can store the long-term dependencies between points. However, with the increasing amount of data, the hidden layers of RNN are very sensitive to small perturbations. Perturbations make the error components of RNN enlarged to reduce the prediction accuracy. Aiming to this problem, G.E. Hinton et al. [6] adopted dropout to reserve or abandon the middle state directly, and W. Di et al. [7] proposed a parametric dropout algorithm for RNN to capture dependencies. Dropout prevents the error components partly and prevents many correct components in the middle state. Lately, D. Krueger et al. [8] proposed zoneout to preserve hidden state randomly, S. Alemany et al. [9] imitated zoneout to regularize RNN and A. Mujika et al. [10] used zoneout to regularize the proposed method FS-LSTM. However, the preserved state also includes the error components. A recent proposed surprisal-driven zoneout (SDZ) [11] is flexible to control the activation of a given cell. SDZ determines the transmission of the middle state according to the middle state itself and prevents the transmission of error components.

Aiming at exploring long-term dependencies between trajectory points, we propose a taxi destination prediction method by regularized RNN with SDZ (TDPRS). TDPRS exploits RNN to store the long-term dependencies in the taxi destination prediction. In order to deal with large amount of trajectory data, we apply SDZ to improve prediction accuracy and reduce prediction time. TDPRS passes the correct components and prevents the error components in middle state through SDZ. In this way, the influence of perturbations can be reduced in the prediction process. So, the predicted state is closer to the real state and the prediction accuracy is improved. At the same time, SDZ changes the approach of parameters update. Every time, TDPRS only updates some parameters rather than all parameters as RNN. It reduces the training time.

2. Taxi Destination Prediction by Regularized RNN with SDZ

Firstly, we analyze how SDZ deals with the middle state and the way of parameters update. Secondly, we apply the SDZ to regularize RNN to predict the taxi destination.

2.1 SDZ

Zoneout can maintain the previous values of the middle states. SDZ adds a surprisal feedback to change the rate of zoneout maintenance. The structure of SDZ is shown in
Fig. 1 The structure of SDZ.

Firstly, SDZ set a variable $S_t$ as shown in Eq. (1).

$$S_t = p_{t-1} - x_t$$  \hspace{1cm} (1)

Where $p_{t-1}$ is a probability distribution at time $t - 1$, $x_t$ is the input data at time $t$. Then to generate the flag $z_t$ as shown in Eq. (2).

$$z_t = \min (\tau + |S_{t-1} \cdot W_{hh}|, 1)$$  \hspace{1cm} (2)

Where $\tau$ is a threshold parameter for numerical stability, $W_{hh}$ is a connection matrix between hidden layers. Sample a binary mask $Z_t$ as shown in Eq. (3).

$$Z_t \sim z_t$$  \hspace{1cm} (3)

The middle state $c_t$ is shown in Eq. (4).

$$c_t = (1 - f_t \odot Z_t) \odot c_{t-1} + Z_t \odot i_t \odot u_t$$  \hspace{1cm} (4)

Where $c_{t-1}$ is the middle state at time $t - 1$. $f_t$ is the value after forget gate, $i_t$ is the value after input gate and $u_t$ is the value after output gate. $h_t$ is the hidden state. $u_t$ is the value after activations.

2.2 Taxi Destination Prediction by RNN with SDZ

Taxi Destination Prediction by regularized RNN with SDZ (TDPRS) is shown in Fig. 2.

Input data includes some embedded data and GPS points. Embedded data are weeks of the year, taxi id and so on. Taxi destination is predicted by RNN. RNN can establish the corresponding relationship between the trajectory data and destination on the basis of input layers, hidden layers and output layers, and then predict the trajectory destination. Process 1 represents the operation between the input and hidden layers as shown in Eq. (5).

$$h_t = f(w_{ih} \cdot (x + c_t) + b_{ih})$$  \hspace{1cm} (5)

Where $w_{ih}$ is the connection weight between input and hidden layers, $b_{ih}$ is the bias, $x$ is the input data, and $c_t$ is the output of SDZ. $f$ is ReLU $[12]$. $h_t$ is the output of the hidden layer.

Process 2 represents the operation between the hidden and original output layers as shown in Eq. (6).

$$r_t = f(w_{ho} \cdot h_t + b_{ho})$$  \hspace{1cm} (6)

Where $w_{ho}$ is the connection weight between hidden and original output layers, $b_{ho}$ is the bias, $h_t$ is the output of process 1 and $f$ is ReLU. $r_t$ is the output of the original output layer.

Process 3 uses Softmax function to generate a probability distribution of trajectory destination clusters as shown in Eq. (7).

$$p_t = \frac{r_t}{\sum_{j=1}^{R} f_j}$$  \hspace{1cm} (7)

Where $R$ is the number of trajectory destination clusters by Meanshift, $r_t$ is output of process 2.

Process 4 represents the sum of multiplies the corresponding elements between trajectory destination clusters and probability distribution as shown in Eq. (8).

$$y = \sum_{t=1}^{R} O_t \odot p_t$$  \hspace{1cm} (8)

Where $\odot$ represents multiplying the corresponding elements, $O_t$ is each cluster point, $p_t$ is output of process 3, $y$ is the final output and represents the predicted point.

How to apply SDZ in TDPRS is shown in dashed line in Fig. 2. The last iteration output of Softmax $p_{t-1}$ acts as input of SDZ to generate the flag $Z_t$ and SDZ filters the last iteration of the $c_{t-1}$ through $Z_t$. In this way, the correct components in $c_{t-1}$ are passed and the error components are prevented. It makes the predicted state closer to the real state and increase the prediction accuracy.

Then, we reserve $Z_t$ into a tuple $TF$ which indicates the updated parameters. At last, we determine the loss function according to the gap between $c_t$ and $c_{t-1}$ and use the loss function to update parameters as shown in Eq. (9).

$$W_t = (c_t - c_{t-1}) \cdot (TF \odot W_{t-1})$$  \hspace{1cm} (9)

Where $W_t$ is the updated parameters at time $t$, $W_{t-1}$ is the parameters at time $t - 1$. After a batch, we update parameters once. The advantage of updating parameters this way is to reduce the number of parameters update and the training time.

The TDPRS can be described as follows:

Algorithm 1. TDPRS

Input: train dataset $train\_data$, test dataset $test\_data$, embedded data $test\_embed\_data$ and $train\_embed\_data$, embedded data includes $taxi\_id$, weeks of year, days of week and so on

Output: the predicted destination $y$ of each trajectory in the $test\_data$

1. $R \leftarrow$ Meanshift($train\_data$);//cluster the training data
to confirm the value of $R$
2. Initialize $M$, $\eta$, $\alpha$, $W$, $b$;
3. Set iterations = 0; //The initial value of the number of iterations
4. Set end_iterations; //the condition of iteration stop;
5. Set batch_size; //the number of samples from train_data in each iteration;
6. //start train
7. for($i = 0; i <$ batch_size; $i$++)
8. $x = sample + train\_embed\_data$ of this sample; //take one sample from train_data randomly
9. $c_t, Z_t \leftarrow SDZ(p_{t-1});$ //use Eqs. (1), (2), (3), (4) to calculate
10. $h_t = f(w_{ih} \cdot (x + c_t) + b_{ih});$ //use Eq. (5) to calculate
11. $r_t = f(w_{hr} \cdot h_t + b_{hr});$ //use Eq. (6) to calculate
12. $p_t = \frac{r_t}{\sum_{t=1}^{T} r_t};$ //use Eq. (7) to calculate
13. $TF \leftarrow Z_t;$ //reserve $Z_t$ into $TF$
14. $y = \sum_{t=1}^{K} O_i \otimes p_t;.$ //use Eq. (8) to calculate
15. $W_t = (c_i - c_{i-1}) \cdot (TF \otimes W_{t-1});$ //update parameters use Eq. (9)
16. iterations += batch_size;
17. if(iterations $\geq$ end_iterations or error $<\$ predefined threshold)$; //the condition of iteration stop
18. end train
19. else;
20. skip step 7 and continue training;
21. //start test
22. for sample in test_data:
23. using the trained model to predict destination of the sample;

In this algorithm, lines 1–5 are the initialization of the TDPRS parameters. Lines 6–20 are the training processes of TDPRS, lines 7–8 are the input of trajectory data and lines 17–18 are the condition of iteration stop. Lines 21–23 are the testing processes.

3. Experiments and Analysis

3.1 Dataset and Environment

The experiments take the Porto taxi trajectory data [4]. It contains the trajectory data for 442 taxis collected in Porto (Portugal) from July 1, 2013 to June 30, 2014. We randomly extract 150 thousand trajectories in the dataset as a training dataset and the test dataset includes 320 trajectories [4].

The experimental program is written in Python 2.7 and uses the third party library of Theano, Fuel and Blocks. The operating system uses Ubuntu16.04. Experimental hardware environment: CPU quad-core, Core i5 processor 2.3GHz, memory is 8GB.

3.2 Initialization of Parameters

Firstly, we use the MeanShift clustering algorithm to get 1379 clusters. We set $R = 1379$ and $M = 500$. In order to maintain the stability in the training process, we set $\eta = 0.01$ and $\alpha = 0.9$ mainly for changing the magnitude of the gradient increasing. The initial value of weight is 0.1 and initial value of bias is 0.01. In order to accelerate the convergence speed, we set batch_size is 200. After determining the initial parameters, TDPRS starts training. After completing of training, the test dataset is used to test TDPRS.

3.3 Evaluation of the Method

We compare TDPRS with RNN method in literature [4]. The performance is measured by Average Distance Error (ADE) and Prediction Accuracy (PA). ADE is calculated by averaging the distance between the point predicted for each track and the true destination. The distance between predicted destination and true destination within 1.0 kilometer is considered accurate. We use the two properties to demonstrate the different performance. The comparison result between the TDPRS method and RNN method is shown in Figs. 3 and 4.

In Fig. 3, the average distance error decreases as the number of iterations increases. Average distance error of TDPRS is lower than RNN method. The minimum average distance error of TDPRS is 2.24 km while RNN method is 3.14 km. The minimum average distance error is reduced by 28% approximately then compared with RNN method. In Fig. 4, Prediction Accuracy of TDPRS is higher than RNN method. The maximum prediction accuracy of TDPRS is 0.731 while RNN method is 0.611. Prediction accuracy of TDPRS is improved by 12% approximately. The accuracy of TDPRS is better than RNN method because SDZ ensures the transmission of the correct components in the hidden layer state, and prevent the transmission of the error components.

In TDPRS, we take different values of $Z_t$ to check the sensibility of $Z_t$ as shown in Fig. 5. In Fig. 5, the lower value of $Z_t$ makes the average distance error lower. When $Z_t = 0.3$, the minimum average distance error is 2.55 km. When $Z_t$ is low means the gap
between adjacent states is very small, the change between predicted state and real state is tiny, and then reduce the average distance error.

In order to verify the advantages of TDPRS in solving long-term dependencies, we select trajectories with more than 200 GPS points from the dataset. The comparison result between TDPRS and RNN method is shown in Figs. 6 and 7.

In Figs. 6 and 7, TDPRS is better than RNN method in dealing with long-term dependencies. This is because of using SDZ. With the increasing of the data, it can still solve long-term dependence very well.

Finally, we record the training time of RNN method and TDPRS when using the same dataset. And the time comparison result is shown in Fig. 8.

In Fig. 8, TDPRS is less than RNN method in training time. The best result in training time of TDPRS is 17837s while RNN method is 19205s. The time of TDPRS is improved by 7% approximately. This is because TDPRS updates some parameters every time rather than updating all parameters like RNN method.

4. Conclusion

We propose TDPRS to predict the taxi destination. TDPRS confirms the correct components of the middle state to pass, making the predicted state closer to the real state, and improving the prediction accuracy. At the same time, TDPRS updates some parameters rather than all parameters of the training process, hence reduces the training completed time. And TDPRS also has more advantages in long-term dependencies.

Acknowledgements

This work was supported by the Fundamental Research Funds for the Central Universities (2017XKQY078).

References

[1] A.K. Laha and S. Putatunda, “Real time location prediction with taxi-GPS data streams,” Iima Working Papers, 2017.
[2] 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,” Proc. 2013 IEEE International Conference on Data Engineering, Brisbane, Australia, pp.254–265, April 2013.
[3] M. Zhang, Y. Yang, L. Huang, and X. Zhang, “Research on destination prediction for urban taxi based on GPS trajectory,” International Journal of Performability Engineering, vol.13, no.4, pp.530–539, 2017.
[4] A.D. Brébisson, S. Étiennet, A. Auvolat, P. Vincent, and Y. Bengio, “Artificial neural networks applied to taxi destination prediction,” J. Computer Science, 2015.
[5] Y. Endo, K. Nishida, H. Toda, and H. Sawada, “Predicting destinations from partial trajectories using recurrent neural network,” Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp.160–172, 2017.
[6] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” J. Computer Science, vol.3, no.4, pp.212–223, 2012.
[7] W. Di, Y.-P. Nie, and J.-M. Huang, “Parametric dropout in RNN,” DEStech Trans. Computer Science and Engineering, 2017.
[8] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. Rosemary Ke, A. Goyal, Y. Bengio, A. Courville, and C. Pal, “Zoneout: Regularizing RNNs by randomly preserving hidden activations,” J. arXiv preprint arXiv:1606.01305, 2016.
[9] S. Alemany, J. Beltran, A. Perez, and S. Ganzfried, “Predicting hurricane trajectories using a recurrent neural network,” “arXiv preprint arXiv:1802.02554, 2018.
[10] A. Mujika, F. Meier, and A. Steger, “Fast-slow recurrent neural networks,” Advances in Neural Information Processing Systems, pp.5917–5926, 2017.
[11] K. Rocki, T. Kornuta, and T. Maharaj, “Surprisal-driven zoneout,” J. arXiv preprint arXiv:1610.07657, 2016.
[12] S. Talathi and A. Vartak, “Improving performance of recurrent neural network with relu nonlinearity,” J. Computer Science, 2015.