Extracting Typical Users’ Moving Patterns Using Deep Learning

Nam Tuan Nguyen†, Yichuan Wang*, Husheng Li†, Xin Liu*, and Zhu Han †
†EECE Department, University of Tennessee, Knoxville, TN

Abstract—When GPS devices are widely integrated into smart phones, researchers stand a big chance of collecting massive location information, that is necessary in studying users’ moving behavior and predicting the next location of the users. Once the next location of a user can be determined, it can serve as input for many applications, such as location based service, scheduling users access in a mobile network or even home automation. One important task in predicting the next location is to identify typical users’ moving patterns. In this paper, we propose a novel method to extract the patterns using deep learning. Experiment results show significant performance improvement of the proposed method compared to the classical principal component analysis method.

I. INTRODUCTION

Predicting user movement has been always an interesting topic for researchers in the past decades. However, due to the lack of smart hand held devices, it has never surged until smart phone with integrated GPS was invented recently. There are many works which focus on location prediction, however, most of them are based on the Markov model [1] [2] [3]. The Markov model is powerful, but it poses some problems such as expressing different patterns over different times or exploiting embedded frequencies in human behavioral patterns [4]. For example, at 6am during the weekdays, an officer normally goes to his office, but on the weekend, at the same time, he more likely stays at home. Markov model treats these two situations in the same way, the probability of going to office when the user is at home at 6am during the weekends is as high as that probability during the weekdays. Hence, if typical users’ moving patterns can be extracted, they will be very useful by combining with the Markov model with different moving patterns for different days. The proposed approach can also serve as a stand-alone solution if we want to predict the next possible visiting locations in the rest of the day given location information from midnight to noon [5].

Existing efforts in identifying users’ moving patterns can be divided into two major approaches. The first approach is from frequentist statistics [6] [7], in which the most frequent moving sequences of visiting places are extracted. Nevertheless, this approach lacks flexibility. For example, when a user stops by a handy mart on the way going from home to office, the user breaks the stored moving sequence, e.g, [Home → Office], and consequently, the system fails to predict the next place. In addition to that, locations are only considered to be in the same sequence if and only if they are in the same time slot [7]. This condition does not fit the real scenarios, many factors can affect users’ schedules.

The second approach is based on principle component analysis (PCA) to extract eigenbehaviors [5] or eigenplaces [8] [9]. An eigenbehavior is a principle component of our daily moving patterns. In [5], three principle patterns were extracted, including weekday, weekend, and no-signal patterns using PCA. The authors then tried to predict future locations given the observed data and obtained promising experiment results. However, compared to deep learning, which has multiple layers, PCA has a huge disadvantage of shallow structure since it consists of only two layers. With multiple layers, deep learning is more powerful and flexible since it is able to combine many layers to generate observations. In addition to that, only three eigenbehaviors might not be sufficient enough to construct different users’ movement patterns. We believe that the results can be further improved by extracting more typical patterns and applying a “deeper” structure.

From the human point of view, given the users daily trajectories, we can tell approximately how many typical moving patterns there are. Our brain can extract multilayer representations of the trajectories, and at the highest abstract level it concludes a number of typical moving patterns. The question is how to reverse engineer the human brain which has multiple layers of expressions. Theoretical results suggest that we should use a deep structure to model this complicated extraction process [10] [11]. Nevertheless, there was not any closed form solution for inference in the deep learning due to the complicated structure and the lack of an efficient algorithm to set up initial weights until recently. Hinton et. al. [12] proposed a method to pretrain the model layer by layer and set the prior to cancel out the explain away effect. The DBN proves to outperform shallow architectures such as PCA in many applications [12]. We distinguish our method from the others in the following aspects:

First, to the best of our knowledge, it is the first attempt to apply deep learning in extracting users’ typical moving patterns. Since deep learning outperforms PCA by far, we expect to achieve a better result compared to the result in [5]. Second, in our approach, to extract the patterns, users do not have to manually label their logical places, which was a requirement in [5]. And finally, we apply deep learning to some collected users trajectories and prove that the deep learning can reconstruct the trajectories with much less error compared to the classical PCA.

The rest of the paper is organized as follows. In Section II, the process of collecting and representing the trajectories to
Fig. 1. 6-week trajectory.

(a) Raw GPS trace  (b) Equivalent binary image

Fig. 2. Converting raw trace to binary image.

Feed to the input of the deep learning is described. In Section III, we discuss about the deep learning and its basic components, the Restricted Boltzmann Machine (RBM). Experiment results are shown in Section IV, and finally, the conclusion is drawn in Section V.

II. DATA COLLECTION

An Android application was written and installed on the HTC Wildfire S smart phones to collect GPS signal every 30 seconds. The data consists of GPS coordinates and recorded timestamps. A group of volunteers live in different cities were chosen to bring the phones on their normal daily routines. GPS is always turned on the phone and locations are collected periodically when the phone is not connected to the wall charger. Data is then automatically uploaded to a dropbox account at the end of the day. A trajectory of a user collected in 38 days is shown on Figure 1.

Based on the timestamps, GPS data in the same day is grouped together to form a data set. The area that includes all the GPS measurements shown on Figure 1 is divided into 784 small cells, corresponding to a (28x28) pixels image. One example of a trajectory for a day is illustrated on Figure 2 where the raw trace on the left is converted to a binary image on the right. For now, we assume the image pixel values are binary. Within a day, if the user shows up at a specific cell, then the corresponding cell on the image is assigned a value of 1. Otherwise, the cell value remains at 0. Furthermore, the image is reshaped to create a 1x784 vector. This vector will serve as the input for the deep learning described in the next section.

III. DEEP LEARNING BASICS

Deep learning is a generative model that consists of multiple layers of hidden stochastic latent variables or features. In this paper, we use a variant of deep learning called Deep Autoencoder (DAE) [12]. DAE stacks multiple layers of RBM, thus, in a few subsections below, RBM is first described and then the generative model of DAE is introduced.

A. Restricted Boltzmann Machine

The Boltzmann machine is a parallel computational model supports “weak” constrains [13]. Units in the Boltzmann machine are stochastic and have energy assigned to them. The RBM is a restricted version of the Boltzmann machine. It does not have any intralayer connection between the hidden nodes, which leads to possible inference algorithms. RBM is an undirected graph that consists of two layer: a hidden layer and a visible layer as well as one bias unit that is always on. Hidden nodes are not connected and are conditionally independent given the observations. This important property makes inference tractable in the RBM. The RBM is illustrated on Figure 3, where v is the visible unit or observation, h represents hidden units and b is the bias unit.

For illustration purpose, we assume that both the observations and the hidden nodes are binary. The activation energy of unit i is:

\[
E_i = \sum_j w_{ij} x_j, \tag{1}
\]

where the index of the sum runs over all units connected to hidden unit i, \(x_j\) represents the \(j^{th}\) observation and \(w_{ij}\) is the weight of the connection between the hidden unit and the observation unit. With activation energy \(E_i\), the probability to turn unit \(i\) on is a logistic sigmoid function:

\[
\rho_i = 1/[1 + \exp(-E_i)]. \tag{2}
\]

The larger the value of the activation energy, the more likely unit \(i\) will be turned on. A weight of a connection represents a mutual connection between the two units. For example, if both units are very likely to be on or off at the same time, then the weight between them should be positive and large. On the other hand, if they are very unlikely to have the same value then the weight should be small and negative. For example, in our application, we have 7 typical patterns, or equivalently, 7 hidden units on the top layer. The hidden units are corresponding to 7 days in a week and are the output of the algorithm. It is essential to mention that each hidden unit
is not a trajectory. It is instead only one hidden node with an activation energy, which is proportional to the probability that the unit is turned on, and a set of related weights. Nevertheless, if the hidden unit is turned on, it can generate a full trajectory by following the weights and activation energies of the lower level. If we apply a trajectory collected on Monday to the visible unit, it is very likely that the Monday hidden unit at the top layer will be turned on and the weight of the connection between them is high.

If we name the vector of visible nodes \( \mathbf{v} \) and the vector of hidden nodes \( \mathbf{h} \), then the energy of any joint configuration \((\mathbf{v}, \mathbf{h})\) is [14]:

\[
E(\mathbf{v}, \mathbf{h}) = - \sum_{i \in \mathbf{v}} a_i v_i - \sum_{j \in \mathbf{h}} b_j h_j - \sum_{i,j} v_i h_j w_{ij},
\]

where \( a_i, b_j \) are the bias associated with visible units and hidden units, respectively, and \( w_{ij} \) is the weight between a hidden unit \( j \) and a visible unit \( i \). The probability of any configuration \((\mathbf{v}, \mathbf{h})\) is:

\[
p(\mathbf{v}, \mathbf{h}) = \frac{\exp[-E(\mathbf{v}, \mathbf{h})]}{Z},
\]

where \( Z \) is a normalizing factor to limit the probability in the range of \([0, 1]\). Given the join probability of the configuration, it is easy to find the marginal probability of a set of visible units:

\[
p(\mathbf{v}) = \sum_{\mathbf{h}} \frac{\exp[-E(\mathbf{v}, \mathbf{h})]}{Z}.
\]

This probability represents the chance that the model with the determined values of weights will generate the observations set \( \mathbf{v} \).

In the training phase, it is expected that the probability is maximized by varying the weights. Thus, the optimum weights can be obtained by taking the derivative of the probability with respect to the weights:

\[
\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = < v_i h_j >_{data} - < v_i h_j >_{\text{model}},
\]

where the brackets \(< \cdot \>_\text{data} \) stands for the expectation of the product of the weights and the observed data and \(< \cdot \>_\text{model} \) is the same expectation for the model observations that is generated according to the model. Obviously, when the derivative is 0, or in other words, when the training data and generated data are similar, we obtain the optimum performance. Hence, the learning algorithm is simply to update the weights with the value in (6).

\[
\Delta w_{ij} = \epsilon (< v_i h_j >_{data} - < v_i h_j >_{\text{model}}),
\]

where \( \epsilon \) is a learning rate. The bigger the \( \epsilon \) is, the faster the algorithm converges, but the coarser grain of the optimal values the algorithm can achieve. For this reason, \( \epsilon \) should be chosen conservatively to achieve the best performance. Experiment results suggest that the value of \( \epsilon \) should be set at 0.1.

**B. A Deep Auto-Encoder for Typical Users’ Moving Patterns Extraction**

A DAE is an artificial neural network used to learn multi-layer representations of an input data. Figure 4 describes the DAE for our application. It consists of 4 hidden layers. The lowest hidden layer has 500 hidden units, the next one has 400, the 3\(^{rd}\) layer has 300 and the top layer has 7 hidden units. Except for the top layer, all the other hidden layers are binary. 0 means the hidden unit is turned off while 1 means the hidden unit is turned on. The top layer has only 7 hidden units since human has 7 typical moving patterns corresponding to 7 days in a week.

Our purpose is to find the weights that minimize the error. However, optimization is over sensitive to initial chosen weights values. The performance is very poor if a large initial weights are chosen, while with small initial weights, optimization is intractable. Recently, Hinton et. al. [12] have shown that by pretraining the network layer by layer, we can obtain a good initial weights values to minimize network errors. After that, a fine-tuning phase is applied to find the local maxima. In the next two subsections, we will describe shortly these two phases.

**C. Pretraining Phase**

Figure 5 shows the pretraining phase that include two steps. At the first step shown on Figure 5(a), for each layer, the hidden units are separated from their upper level hidden units and connected to the lower level hidden units to form an RBM. Hidden units in every RBM are binary except for the top layer, where the hidden units are drawn from a Gaussian distribution with the mean equals to the activation probability. The binary input of the trajectories is fed to input of the first RBM. As described in the previous section, the activation probability can be calculated according to (2). In turn, the hidden units of the first RBM now become visible units in the second RBM. The same process is repeated until the activation probabilities of the top layer hidden units are determined.

The step described above is a bottom to up process to achieve activation probabilities for all the hidden units. Now, given the probabilities and the weights, we implement the second step, shown on Figure 5(b). In this step, a process from top to bottom is followed to generate the model “visible” units at the bottom layer. The difference between the model “visible” data and the actual data gives us a hint to update the weights according to (7). On Figure 5(b), the top trajectory is created by the model while the bottom trajectory is the training data.

After this step, we obtain coarse grain optimum values for the weights. To further improve the result, a fine-tuning
process is implemented using backpropagation in the next subsection.

D. Fine-tuning Phase

Backpropagation is a supervised method to train deep learning. Since the top hidden units have been learned from the above pretraining phase, we first forward the training data to the input of the network and calculate the difference between the inferred top-layer hidden unit and the learned top-layer hidden. The difference at the top layer is then fed backward to go through the network from the top to the bottom layer using the learned weights, producing differences at every layer. The differences at each layer are later used to update the weights of the corresponding hidden layers. The above process is repeated until the differences are under some threshold or the algorithm reaches its maximum number of iterations.

In this paper, we use a backpropagation method called conjugate gradients. Basic backpropagation methods update the weights in the steepest descent direction. While it is sufficient enough to find minima, it is proved that the convergence time is not optimum since the direction of the deepest descent direction may circle around toward a global minima. The conjugate gradients method searches for different directions that are pedicular or conjugate to each other so that optimization in one direction does not affect the optimization in other directions. By searching in different directions, the algorithm can reach to the minima much faster. We based on the code provided by [12] to perform the algorithm.

E. Typical Pattern Extraction Algorithm

Algorithm 1 summarizes our algorithm to extract typical patterns. Weights are initialized by randomly choosing values from the range of $[-0.1, 0.1]$. Then, pretraining phase is implemented to find initial weights for the next phase, the fine-tuning phase. After learning all the weights and the typical patterns, a test trace is fed to the network and a corresponding model trace is then regenerated. The difference between the test trace and the regenerated trace is calculated to measure the error, which is numerically described in the next section.

IV. Experiment Results

In this section, we will evaluate the performance of deep learning in reconstructing user trajectory. A trajectory of 38 days from a graduate student was selected, from which 30 days are used for training and the other 8 days are used for testing. After the learning process, weights are learned and then, a test trajectory is applied to the input of the network and used to reconstruct a model trajectory. We will prove that with the learned weights of the hidden units, we can successfully reconstruct the tested trajectories and compare the result with PCA based on the mean square error. The mean square errors of the two algorithms are calculated as the difference between the pixels of the reconstructed trajectory and the input trajectory.

Figure 6 shows an input trajectory and its corresponding reconstructed one by deep learning. Notice that the number of dimensions is reduced from 28 x 28 to 7. Define the compression ratio as the ratio of the number of the dimensions originally needed to describe the trajectory to the number of dimensions extracted at the top level. We can see that even at a large compression ratio, deep learning performance is still noticeable, especially when compared with PCA. The reconstructed trajectory is not deviated far away from the original one. The mean square error between the two trajectories is 5.3127.

To compare the result with PCA, the same set of training and testing data are used. 7 principle components are extracted to reconstruct the trajectories. As illustrated on Figure 7, PCA has much worse performance compared to deep learning. The mean square error of the PCA method is $6.282 \times 10^5$, compared to 5.3127 if reconstructed using the deep learning method. Obviously, deep learning beats PCA by far. The noticeable improvement in performance that the deep learning can achieve is due to its deep structure while PCA only has a shallow structure of two layers.
We further test the algorithm with a various hidden units on the top layer and show the result on Figure 8. As we can observe, when the number of hidden units in each layer increases, the performance also increases. Obviously, when the number of hidden units increases, the deep network is more flexible and powerful since it has more branches to generate different set of trajectories. However, the complexity increases at the same time as a trade off. Hence, depending on applications, the deep network can be changed to meet both the performance requirements and complexity requirements.

A. Possible Future Applications

In our preliminary work, we only extract typical users’ moving patterns. However, many applications can be developed based on our result. For example, at the beginning of the day, we can use the learned extracted moving patterns to generate a set of the most possible locations that a user may visit during the day. In combination with signal strength profile collected at the locations, we can effectively schedule users’ access to the network. Delay tolerant packet transferring can be delayed until the time the user has a strong signal strength instead of transferring the packets immediately.

Another application is to predict next possible locations given the locations observed from beginning of the day until the current time. As soon as new data are collected, they are used to feed to the deep network to learn the weights of the typical patterns. The weights are then matched with the stored weights of different days and the most similar moving pattern will be selected to generate possible future locations for the rest of the day. This application is specifically tested by Eagle et. al. in [5]. Since deep learning significantly outperforms PCA, which is used to extract typical patterns in [5], we would believe that our proposed algorithm can significantly improve their results.

V. CONCLUSION

In this paper, we have implemented a 4-layer deep network to learn typical users’ moving patterns. At the first step, we pretrain the network by learning it layer by layer using RBM learning algorithm. Then, a backpropagation is deployed to fine-tune the network. Real traces were collected and transferred into binary images. The performance evaluated with the traces is shown to be significantly improved compared to the traditional methods. Our preliminary result also suggests some promising applications and can be extended to many other fields.

ACKNOWLEDGMENTS

The authors acknowledge the valuable help provided by Sophia Tsang in collecting data used in testing our algorithm.

REFERENCES


