Geoinformatica
https://doi.org/10.1007/s10707-019-00378-7

Improving human mobility identification with trajectory augmentation

Fan Zhou[1] · **Ruiyang Yin**[1] · **Goce Trajcevski**[2] · **Kunpeng Zhang**[3] · **Jin Wu**[1] · **Ashfaq Khokhar**[2]

Received: 31 January 2019 / Revised: 3 July 2019 / Accepted: 7 August 2019 /
Published online: 29 August 2019

Abstract

Many location-based social networks (LBSNs) applications such as customized Point-Of-Interest (POI) recommendation, preference-based trip planning, travel time estimation, etc., involve an important task of understanding human trajectory patterns. In particular, identifying and linking trajectories to users who generate them – a problem called Trajectory-User Linking (TUL) – has become a focus of many recent works. TUL is usually studied as a multi-class classification problem and has gained recent attention because: (1) the number of labels/classes (i.e., users) is way larger than the number of motion patterns among various trajectories; and (2) the location-based trajectory data, especially the check-ins – i.e., events of reporting a location at particular Point of Interest (POI) with known semantics – are often extremely sparse. Towards addressing these challenges, we introduce a Trajectory Generative Adversarial Network (TGAN) as an approach to enable learning users motion patterns and location distribution, and to eventually identify human mobility. TGAN consists of two jointly trained neural networks, playing a Minimax game to (iteratively) optimize both components. The first one is the generator, learning trajectory representation by a Recurrent Neural Network (RNN) based model, aiming at fitting the underlying trajectory distribution of a particular individual and generate synthetic trajectories with intrinsic invariance and global coherence. The second one is the discriminator – a Convolutional Neural Network (CNN) based model that discriminates the generated trajectory from the real ones and provides guidance to train the generator model. We demonstrate that the above two models can be well tuned together to improve the TUL performance, while achieving superior accuracy when compared to existing approaches.

Keywords Adversarial learning · Spatio-temporal learning · Synthetic trajectory generation · Motion pattern recognition

✉ Fan Zhou
fan.zhou@uestc.edu.cn

Extended author information available on the last page of the article.

1 Introduction

The advancements in networking technologies and miniaturization of computing and communication devices have enabled plethora of applications that generate large volumes of spatio-temporal data pertaining to both: (a) mobile users locations; and (b) additional contexts associated with the temporal and spatial dimension. Location-based Social Networks (LBSNs) such as Instagram and Twitter generate large scale geo-spatial datasets capturing human behavior at unprecedented volume and level of detail [16]. This, in turn, created opportunities to uncover various patterns created by mobile human users, to better understand different motion plans in various application scenarios [54], such as: inferring latent mobility patterns [1]; personalized recommendation of Point Of Interests (POI) [6] or suggesting next location to visit [12]; geo-aware maximization of influence [32], etc.

A crucial task to enable better understanding of the behavior of mobile users in multiple applications is the *trajectory classification* [18, 37]. Examples of patterns related to a collection of spatio-temporal trajectories include descriptions of values for mobility-related attributes such as *stationary* or moving; *driving* or *walking*; conducting activities (e.g., *dining* while stationary; *listening* to news while driving); etc. – which have recently spurred the topic of trajectory semantic inference [15]. Many of the traditional trajectory classification works leverage techniques aiming at understanding the activity patterns and transportation modes of users given the history of visited locations. Typical tools include Hidden Markov Model (HMM), Dynamic Bayesian Network (DBN), and Conditional Random Fields (CRF). Complementary to this, a body of works addressed the problem of discovering the characteristics of a single user or a group of users, relying on Linear Discriminant Analysis (LDA) and Bayesian probabilistic graph models, targeting applications such as POI recommendation [6] and trip planning [10].

Part of the motivation for this work is based on the observation that the existing bodies of work have not addressed the problem of *linking trajectories* to their corresponding generating *users*. This is an important problem in many LBSN-type of applications like, for example, ride-sharing (bike, car) platforms. On the one hand, the users provide large volumes of trajectories data in terms of (location, time) sequences, however, their identities are usually protected for the sake of privacy. On the other hand, the ability to correlate such trajectories to users may enable more informed decision-making in personalization of, e.g., marketing campaigns. Moreover, from a complementary perspective, such linking could help in detecting potential criminals, based on values from similar types of sparse mobility data such as the transient phone signals as well as other check-in events – i.e., a presence at a location corresponding to a particular Point of Interest (POI), possibly with semantic features available from a POI database.

The *Trajectory-User Linking (TUL)* problem was first studied in [20], where threefold challenges were identified: (1) the number of classes/labels (unique mobile users) is way larger than the number of possible motion patterns in trajectory classification; (2) the sparsity of trajectories, users usually visit a few popular locations among thousands. In addition, the data may involve noise and outliers that can affect the performance of TUL; (3) TUL differs from many traditional mobility pattern-recognition problems in that it requires extracting and analyzing various features from trajectories [48], which entails both the curse of dimensionality, as well as the invasion of user privacy.

The solution proposed in [20] – *TULER (TUL via Embeddings and Recurrent Neural Networks)* – achieves high classification accuracy, however, it fails to address one of the most important issues in mining LBSNs data, i.e., the sparsity of a user-location check-ins. For example, the sparsity of the Gowalla dataset [13] is about 99.98% [59]. Additionally,

mining and characterizing individual motion patterns requires a large number of labeled trajectory data – the lack of which is another setback of most LBSNs studies [54].

Since numerous factors – such as transportation modes; priorities (job, family, friends), etc. (cf. [21]) – may influence human mobility patterns, users trajectories are often approximated with models like random walk [9] or Lévy flight [41]. While human mobility is associated with spatial and temporal constraints that follow reproducible scaling laws, the number of independent parameters characterizing the human movement can be quite large. Moreover, growing evidence suggests that the existing parameter-based scaling law studies on spatio-temporal datasets are limited and affected by the domain of explored datasets, i.e., the results from mobile phone data are not generalizable to LBSNs data, and vice versa.

The above facts are at the heart of the motivation for investigating the *Trajectory Distribution Approximation (TDA)* – a novel trajectory pattern recognition problem in the areas of both human mobility and social networks. TDA has a similar task with traditional statistics-based scaling law discovery problem – that is, uncovering the basic human mobility patterns. However, it learns the individual trajectory probability distribution directly from the data and fits the underlying characteristic of its mobility patterns, rather than quantitatively analyzing various factors and random variables – e.g., displacement distribution [9], rank distribution [4], etc. – that affect users motion. We postulate that an efficient TDA solution should satisfy four desiderata: (1) Ability to capture the natural features determining the individual moving patterns; (2) Enable generating synthetic trajectories of a particular person, once the characteristics of his moving patterns are learned; (3) Ensure theoretical soundness of the method for guiding the training process and testing the results of trajectory generation; and (4) The synthetic trajectories should improve the performance of other supervised trajectory classification problems, such TUL in LBSNs.

One appealing methodology for tackling the TDA problem are the generative models – in particular, Generative Adversarial Nets (GAN) [22]. Essentially, a GAN is a minimax game between a *generator* and a *discriminator* seeking to match the distributions of both the generated and the real data. Unfortunately, the existing GAN based methods – both continuous GANs (e.g., used for image generation [11]) or discrete ones (e.g., used for sequence modeling [50] and text generation [53]) – can not be directly applied to the TDA problem. The main reason is that the true semantics of the generated trajectories can not be identified neither by the GAN discriminators nor by a human labor directly. For example, the generated images or sentences may be easily discriminated from the realistic-looking ones by manual identification of an ordinary person without any supervision. However, when it comes to the generated POIs forming a trajectory from real locations visited by a user, the end result may consist of indistinguishable patterns of a synthetic trajectory. Moreover, as mentioned, the LBSNs datasets are usually too sparse to characterize an individual mobility patterns, not to mention the extremely imbalanced label (user) distribution in LBSNs datasets, both of which require an elastic generator to prevent model collapse – a problem commonly occurring in the existing GANs, largely due to insufficient data.

To tackle this problem, we introduce the TGAN (Trajectory Generative Adversarial) network, which consists of two adversarial training neural networks: (1) a Long Short-Term Memory (LSTM) [24] network employed as a POI sequence generator; and (2) a Convolutional Neural Network (CNN) [28] playing the role of discriminator. The generator learns the trajectory representation and progressively fits the underlying trajectory distribution of a particular person, after which the synthetic trajectories with intrinsic invariance and global coherence are generated, preserving the long-term dependencies of POIs. The discriminator, on the other hand, takes the responsibility of distinguishing the generated trajectory from the real ones and guides the training of the generator. These two jointly trained

neural networks in a two-player game iteratively optimize both sides and learn individual motion patterns and spatio-temporal distribution, until convergence. In addition to improving the TUL, TGAN can also directly evaluate the quality of the trajectory generation, and the synthetic data could be incorporated to improve the supervised trajectory classification accuracy by augmenting the labeled trajectories and balancing the label distribution.

Following are the main contributions of this work:

(1) We introduce a novel formalization of the TUL problem from the perspective of TDA, and we discuss its positioning in the context of TULER – an RNN (Recurrent Neural Network) based solution to the TUL problem.
(2) We present the first comprehensive TDA solution – TGAN (Trajectory Generative Adversarial Network) which: (a) fulfills the aforementioned four properties of TDA; and (b) sidesteps the problems in existing GANs.
(3) We provide omprehensive experimental evaluations conducted on real-world datasets, which illustrate the benefits of TGAN over the existing approaches.

We note that our earlier work [20] introduced and addressed the TUL problem – however, this study presents a significant extension of it: both the novel TDA variant of the problem as well as the TGAN approach for the solution.

The rest of this article is structured as follows: we review the related work in Section 2 and Section 3 formalizes the TUL problem and present the main methodology behind TULER RNN. Section 4 formalizes the problem of TDA and provides the TGAN solution to data sparsity. Experimental evaluations quantifying the benefits of our solutions are presented next (Section 5), followed by concluding remarks and directions for future work.

2 Related work

We now review the related literature and position our work in that context. There are three broad categories of approaches that we overview in the sequel.

Uncovering patterns characterizing human motion has been studied in traffic engineering [3], city planning [51] and many location-based applications [8]. Traditional motion pattern mining studies fall into four broad classes: (1) *individual statistical patterns understanding*: measuring and quantifying the models, e.g., continuous-time random-walk [9] or Lévy flight [21], accounting for characteristics of individual human trajectories [41]; (2) *trajectory similarity mining*: measuring the similarity or distance between two trajectories, such as Dynamic Time Warping or Edit distance [17] – which may exploit the uniqueness and regularity of human mobility, and linking accounts across sites, etc.; (3) *sequential and periodical pattern mining*: finding (sub-)sequences and periodical motion patterns, enabling travel recommendation [10], life pattern understanding [42] and trajectory classification [20] and next location prediction [6]; (4) *trajectory classification*: recognizing trajectories as different types of motion patterns, such as Biking, *Bus*, *Driving*, and *Walking* in transportation classification [55] and *Occupied*, *Nonoccupied* and *Parked* in taxi status inference [58]. The key task involved is to extract representative spatio-temporal features in trajectories. They are different from the complex TUL problem because of the large number of labels (users) and the interleaving sub-trajectories among users.

Recurrent Neural Networks (RNNs) have achieved great successes in many Natural Language Processing (NLP) applications, especially since the introduction of memory units to

networks, such as LSTM [24] and GRU [14]. It has been widely applied to text classifications [30, 33], where the number of labels/classes is relatively small. Most of them are binary (e.g., IMDB[1] dataset), while few have more (but at most 20) classes – e.g., Fudan[2] dataset. This is substantially less than a typical TUL setting. In addition, the sufficient corpus is usually available in the task of text classification, which can alleviate the data sparsity issue that is severe is TUL.

Despite the large body of works in individual trajectory parameterizing and semantic trajectory mining – the TDA problem has not been formally defined and investigated. Our proposed TGAN methodology is different from conventional trajectory models in that it: (1) requires to match individual motion patterns and trajectories distribution; and (2) may generate synthetic trajectories that augment the training data to improve the location-based applications such as clustering and classification. To our best knowledge, TGAN is the first attempt to perform such trajectory distribution approximation.

A recent approach to augment the initial TUL settings and solution was presented in [57]. Essentially, a generative model was proposed to mine human mobility patterns, relying on the Variational Auto-Encoder (VAE) [26] paradigm, and an architecture consisting of three RNNs (encoder, intermediate RNN and decoder) plus a semi-supervised classifier was presented. However, the work in [57] is, in a sense, orthogonal to this work, from two perspectives: (1) it aims at learning the implicit hierarchical structures of trajectories, whereas we focus on the TDA problem, as well as generation of synthetic trajectories datasets; (2) we rely on GAN paradigm and the architecture also employs CNN (in the discriminator).

Generative Adversarial Networks (GANs) have gained a tremendous successes in natural image generation [22]. Recently, GANs has also been used as a tool for modeling sequential data [50] and generating text [53]. However, few efforts have been conducted towards modeling human trajectories with adversarial networks that confront: (1) the sparsity problem of check-ins inherent in trajectory data; (2) extremely longer trajectory sequences than other generated discrete samples, such as sentence in natural language; and (3) the lack of metrics for evaluating the trajectory generation results. TGAN not only provides a complementary approach to traditional scaling law based trajectory models by approaching the latent trajectory distribution of users, but may also inspire potential novel location-based applications – e.g., adversarial trajectory based recommendation and privacy protection.

3 Trajectory-user linking

We now turn the attention to formalizing the TUL problem, and discuss the intricacies of TULER – our proposed method.

3.1 Problem settings

Let $T_{u_i} = \{l_{i1}, l_{i2}, ..., l_{in}\}$ denote a trajectory generated by the user u_i corresponding to a particular time interval, where l_{ij} ($j \in [1, n]$) is the location/POI at time t_j for the user u_i ($\in \mathcal{U}$), in a suitable coordinate system (e.g., longitude + latitude, or a Cartesian coordinate

[1] http://ai.stanford.edu/amaas/data/sentiment/
[2] http://www.datatang.com/data/44139

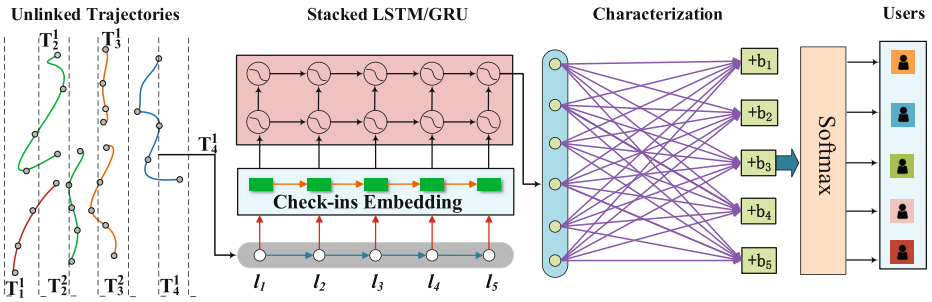


Fig. 1 Architecture of the proposed method TULER. TULER first learns check-in embeddings $\mathbf{T} \in \mathbb{R}^{|C| \times d}$ using all trajectories. Then a RNN model will be trained to characterize latent patterns of linking trajectories to users. Finally, a user of an unlinked trajectory is inferred

system after a suitable projection: $(x_{l_{ij}}, y_{l_{ij}})$). In this paper, each l_{ij} is considered as a *check-in*.

Definition 1 Trajectory-User Linking (TUL): A trajectory $T_k = \{l_1, l_2, ..., l_m\}$ for which it is not known who was the user generating it, is called *unlinked*. Assume that we are given a number of unlinked trajectories $\mathcal{T} = \{T_1, ..., T_m\}$ generated by some of the users in the set $\mathcal{U} = \{u_1, ..., u_n\}$ ($m \gg n$). We call a *solution* to TUL problem the mapping that assigns unlinked trajectories to the users: $\mathcal{T} \mapsto \mathcal{U}$.

Our proposed solution (TULER) works as follows: each unlinked trajectory is first divided (i.e., segmented) into a collection of sub-trajectories based on a fixed time interval [5]. Subsequently, we represent and characterize each trajectory using trajectory embedding via trained RNN models, with a capability to mitigate the curse of dimensionality problem. Finally, we build a multi-class classification model to link these trajectories to users. The overall architecture of TULER is shown in Fig. 1.

3.2 Segmentation and check-in embedding

To capture the rich semantics content of the trajectories and increase the computational efficiency, we segment each trajectory T_{u_i} into k consecutive sub-trajectories $T^1_{u_i}, ..., T^k_{u_i}$. Such a segmentation process can be done by various methods, e.g., based on the semantic meaning and shape of the trajectories [54]. For simplicity, we adopt the method used in [34] in this paper, although other methods are possible, e.g., re-sampling in frequency domain (cf. [5]) – which we defer to our future work.

To alleviate the sparsity issue and dimensionality curse, we represent each check-in with a low-dimensional vector $\mathbf{v}_{l_t} \in \mathbb{R}^d$ rather than a traditional location representation method such as one-hot. We obtain the check-in trajectory representation $\mathbf{T} \in \mathbb{R}^{|C| \times d}$ using the similar technique implemented in word embeddings [35], where $|C|$ is the number of all unique check-ins across all trajectories, d is the dimensionality in the lower dimensional space. Specifically, we maximize the probabilities of check-ins given their neighboring locations in trajectories.

Figure 2 shows the distribution of check-ins in two real-world LBSN datasets from [13], which follow a power-law distribution. Using check-in embedding can somehow address

the sparsity issue as discussed above. In addition, it can mitigate the overfitting problem especially when the amount of training instances is small.

More specifically, the embedding of a check-in l_t is to infer its conditional probability given the context check-ins $\mathcal{C}(l_t) = l_{t-w} : l_{t+w}$, where w is the size of sliding window of each sub-trajectory. To better embed our check-in into a low dimensional representation $\mathbf{v}_{l_t} \in \mathbb{R}^d$, the probability $p(l_t|\mathcal{C}(l_t))$ is defined by the *softmax* function as:

$$p(l_t|\mathcal{C}(l_t)) \prod_{l' \in \mathcal{C}(l_t)} p(l_t|l') \prod_{l' \in \mathcal{C}(l_t)} \frac{\exp(\mathbf{v}_{l_t} \cdot \mathbf{v}_{l'})}{\sum_{l'' \in C} \exp(\mathbf{v}_{l''} \cdot \mathbf{v}_{l'})} \tag{1}$$

3.3 Trajectory characterization

Part of the problem in trajectory characterization stems from the fact that splitting the original trajectories into sub-trajectories even with a shorter duration, may still result in denser check-ins in some of the sub-trajectories. To address this issue (long-term variable-length location sequences), we incorporate several variants of well-known RNN models, i.e., LSTM [24] and GRU [14], as well as the stacked and bidirectional RNNs, into TULER. These are aiming at controlling the input and output of trajectory embeddings. We provide a brief description of the use of LSTM and GRU model used in TULER next.

3.3.1 TULER with LSTM

Given a sub-trajectory $T = \{l_1, l_2, ..., l_k\}$, let h_{t-1}, h_t and $\tilde{h}_t$ denote the respective last, current and candidate embedding state. The LSTM model that we incorporated in TULER is implemented as follows:

$$i_t = \sigma(W_i \mathbf{v}^t(l_i) + U_i h_{t-1} + V_i c_{t-1} + b_i), \tag{2}$$

$$f_t = \sigma(W_f \mathbf{v}^t(l_i) + U_f h_{t-1} + V_f c_{t-1} + b_f), \tag{3}$$

$$o_t = \sigma(W_o \mathbf{v}^t(l_i) + U_o h_{t-1} + V_o c_t + b_o) \tag{4}$$

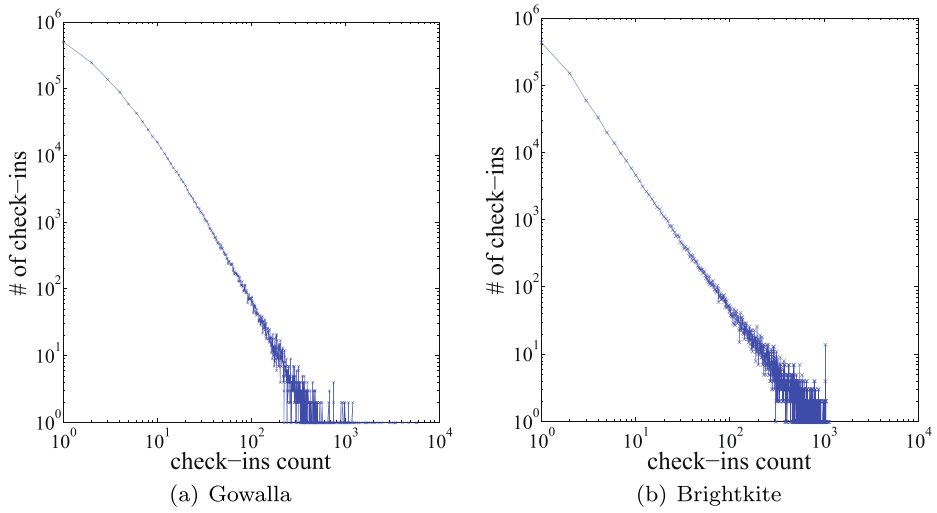


Fig. 2 Frequency count of check-ins from real-world datasets

where i_t, f_t, o_t and b_* respectively denote the input gate, forget gate, output gate and bias vector. Also, σ denotes the logistic sigmoid function; the different gate parameters are specified by the matrices W, U and V ($\in \mathbb{R}^{d \times d}$); and $\mathbf{v}^t(l_i)$ is the embedding of the check-in location l_i. The memory cell c_t is updated by partially replacing the existing memory unit with a new cell $\mathbf{c}_t$ as follows:

$$c_t = f_t c_{t-1} + i_t \tanh(W_c \mathbf{v}(l_i) + U_c h_{t-1} + b_c) \tag{5}$$

Then, the trajectory embedding is updated as:

$$h_t = o_t \odot \tanh(c_t) \tag{6}$$

where $\sigma(\cdot)$ and $tanh(\cdot)$ denote the sigmoid and hyperbolic tangent function, and $\odot$ denotes the entry-wise product.

3.3.2 TULER with GRU

TULER with GRU models the trajectory embedding using extra gating units, in a similar spirit to the LSTM method, however, it does so without separated memory cells. The state of h_t is updated by a linear interpolation between the last state h_{t-1} and the candidate state $\tilde{h_t}$ in the following manner:

$$h_t = (1 - g_t) h_{t-1} + g_t \tilde{h}_t \tag{7}$$

where g_t is the update gate, deciding by how much the unit will updates its activation:

$$g_t = \sigma(W_z \mathbf{v}^t(l_i) + U_z h_{t-1}) \tag{8}$$

The computation of the candidate state $\tilde{h}_t$ is much like a traditional RNN unit:

$$\tilde{h}_t = \tanh(W \mathbf{v}^t(l_i) + U(s_t \odot h_{t-1})) \tag{9}$$

with s_t denoting the set of reset gates; computed, in turn, very similarly to updating the gate, as:

$$s_t = \sigma(W_s \mathbf{v}^t(l_i) + U_s h_{t-1})$$

3.3.3 Variants

We now discuss the implication of having certain variants of TULER (i.e., LSTM/GRU and Bidirectional LSTM [43]) stacked. The peculiarity of a stacked LSTM/GRU, is that the hidden state of a given unit in layer n, is subsequently used as an input to the unit in the layer $n+1$ – except, at a same time step. The main objective of multi-RNN stacking is to grasp the longer check-in dependencies of a trajectory.

There is a drawback to stacking, though – the training time of a stacked TULER increases exponentially with the number of layers. An option to alleviate this issue is to use Bidirectional LSTM by running two LSTMs in parallel: (1) on the sequential check-in embedding vectors, and (2) on the reverse embedding vectors. As it turns out, this may yield a substantial reduction in the time for training the model, in comparison with the general and stacked LSTM/GRU based TULER. We note that the performance of using different types of *deep* TULER is discussed in Section 5.

3.4 Trajectory-user linking

Linking the trajectories to the corresponding users is done by using a softmax output layer to calculate the corresponding estimated user distribution:

$$p(\hat{u}(l_u) = i | l_u; \kappa) = \frac{\exp\{B_i h_u + b^i\}}{\sum_{j=1}^{|u|} \exp\{B_j h_u + b^j\}}, \forall i = 1, \cdots, |u| \tag{10}$$

The interpretaion of the symbols in the equation above is as follows: – $\hat{u}(l_u)$ is the predicted user of l_u and h_u is the final hidden state of RNN module; – the weight matrix $B \in \mathbb{R}^{d \times |u|}$ and bias vector $b \in \mathbb{R}^{1 \times |u|}$ are the corresponding parameters in the softmax layer; B_i indicates the i-th column of B and b^i indicates the i-th element of b; lastly, $\kappa = \{W, U, V, B, b\}$ denotes the set of parameters that need to be learned.

To learn the parameters in the quintuple κ with respect to the given objective function, we proceed as follows. Given a user u and his trajectory sequence $l_u = l_1, l_2, ..., l_m$, we train the TULER to maximize the log-likelihood with respect to κ with:

$$u(l_u) \mapsto \sum_{l_u \in \mathbb{U}} \log p(u | l_u, \kappa) \tag{11}$$

where u and $\mathbb{U}$ denote the ground-truth user of trajectory l_u and the training data, respectively. A *stochastic gradient descent* is used in each step, in order to estimate the parameter set κ:

$$\kappa \leftarrow \kappa + \alpha \frac{\partial \log p(u | l_u, \kappa)}{\partial \kappa} \tag{12}$$

where α is the learning rate.

Lastly, we proceed with the minimization of the following cost function:

$$\Phi(l_{u_i}, \tilde{l}_{u_i}) = -\sum_{i=1}^{|l|} \sum_{j=1}^{|u|} u \log(\tilde{l}_i^j) \tag{13}$$

where $\tilde{l}_i^j$ is the predicted trajectory embedding vector.

After both of the models – the embedding one and the RNN have been trained, we construct the TULER model. Its embedding layer is used to encode the semantics of check-ins, initialized with the corresponding trained weights. A random initialization is used for the stacked or bidirectional layer of RNN and the softmax of the model, while all the parameters are being fine-tuned on the labeled data.

4 Adversarial synthesis trajectory generation

We now present the main results of the TGAN based solution, starting with an overview of GAN, and a formal definition of the TDA problem, and following with the details of our proposed method TGAN. Lastly, we discuss the evaluation and training details.

4.1 Generative adversarial nets

To make the paper self-contained, we now present an overview of the relevant background of GAN and WGAN.

4.1.1 Generative adversarial nets (GAN)

The main objective of **Generative Adversarial Nets (GAN)** is to obtain the equilibrium between a discriminator D and a generator G by optimizing the following minimax objective [22]:

$$\mathcal{L}_{GAN}(p_x, p_g) = \mathop{\mathbb{E}}_{x\sim p_x}[\log D(x)] + \mathop{\mathbb{E}}_{z\sim p_g}[\log(1 - D(G(z)))] \tag{14}$$

where p_x is the data distribution and p_g is the model distribution. The $\mathcal{L}_{GAN}$ is then maximized with respect to $D(x)$ and minimized with respect to $D(G(z))$. The generator G takes a prior noise distribution $z \sim p(z)$ (e.g., uniform or Gaussian) as input and produces a sample $G(z)$ in the data space using a deep neural network, such as Multi-Layer Perceptron (MLP) [22]. The discriminator D – normally another neural network such as CNN or MLP – plays the role of classifier and represents the probability that a certain sample comes from the true data distribution p_x or the generator G. In practice, the parameters of the generator and the discriminator networks are updated in an alternating fashion, based on stochastic gradient descent (SGD) [39]. It has been demonstrated that this game achievesa global equilibrium if and only if $p_g(x) = p_x(x)$, and the optimal discriminator is $D^*(x) = p_x(x)/(p_x(x) + p_g(x))$ [22]. That is, if the optimal discriminator is found in each iteration, minimization of the resulting loss function of the generator implicitly leads to minimization of the lower bound on the Jensen-Shannon divergence (JSD) .

4.1.2 Optimal transport and WGAN

We note that, as demonstrated in [2], the strong probability distances (such as KL divergence and JSD) between p_x and p_g may no longer provide useful gradients forthe generator and correspondingly result in model collapse. This is an implication of the optimal discriminator becoming perfect, and its gradient will be zero almost everywhere. More importantly, this is the case for many real applications where both p_x and p_g have supports that are disjoint or lie on low dimensional manifolds. To address this issue, Wasserstein GAN (WGAN) was proposed [2]. It uses the Earth-Mover distance (EMD), denoted $W(q, p)$, as the measure of the distance between two distributions (rather than JSD in "regular" GAN), which is informally defined as the minimum cost (mass times transport distance) of transporting mass in order to transform one distribution q to another distribution p.

Optimal Transport The theory of Optimal Transport (OT) problem, in addition to Earth Mover's Distance (EMD), has induced a rich class of divergences between probability distributions, among which two main formulations (i.e., Monge's and Kantorovich's) were developed in the past century [27]. In the machine learning field, Kantarovich's formulation is more popular since it is more general and also covers the case of discrete masses (in our case, trajectories).

More formally, let $\mathcal{X}$ be a metric space (e.g., $\mathcal{X} = \mathbb{R}^n$) endowed with a metric $d_{\mathcal{X}}$. A coupling π of p_x and p_g is a probability distribution on $\mathcal{X} \times \mathcal{X}$ such that $\pi(\mathcal{A}, \mathcal{X}) = p_x(\mathcal{A})$ and $\pi(\mathcal{X}, \mathcal{A}) = p_g(\mathcal{A})$ for all Borel probability measures ($\mathcal{A} \subseteq \mathcal{X}$) with finite moments of order k, i.e., $\int_{\mathcal{X}} \rho(X, Y)^k d p_x(x) < \infty$ and $\int_{\mathcal{X}} \rho(X, Y)^k d p_g(x) < \infty, \forall Y \in \mathcal{X}$, where

$\rho(X, Y)$ is a distance function for any two instances in $\mathcal{X}$. Kantorovich's formulation of OT problem is defined as [45]:

$$\begin{aligned} \mathbf{W}^k(p_x, p_g) &= \inf_{\gamma \in \prod(X \sim p_x, Y \sim p_g)} \mathbb{E}_{(X,Y)\sim\gamma}[\text{cost}(X, Y)] \\ &= \left(\inf_{\gamma \in \prod(X \sim p_x, Y \sim p_g)} \iint_{\mathcal{X}\times\mathcal{X}} \rho(X, Y)^k d\pi(X, Y) \right)^{\frac{1}{k}} \end{aligned} \tag{15}$$

where $\text{cost}(X, Y)$ is any measurable cost function and $\prod(X \sim p_x, Y \sim p_g)$ is a set of all joint distributions $\gamma(X, Y)$ of (X, Y) (all coupling of p_x and p_g) whose marginals are respectively p_x and p_g. Intuitively, $\gamma(X, Y)$ indicates how much mass must be transported from X to Y in order to transform the distribution p_x into the distribution p_g. A particularly interesting case is when $(\mathcal{X}, \rho)$ is a metric space and $\text{cost}(X, Y) = \rho(X, Y)^k$ for $k \geq 1$. The k-th root of Wass^k is called the k-Wasserstein distance, which is then the cost of the optimal transport plan. In the case of $k = 1$ (1-Wasserstein distance), it refers to the EMD and the following Kantorovich-Rubinstein duality theorem holds for representing EMD as a form of integral probability metric [46]:

$$\inf_{\gamma \in \prod(X,Y)} \iint \rho(X, Y) d\pi(X, Y) = \sup_{f \in \mathcal{F}_L} \left(\int_{\mathcal{X}} f(x) d p_x(x) - \int_{\mathcal{X}} f(x) d p_g(x) \right) \tag{16}$$

where $\mathcal{F}_L$ is the class of all bounded 1-Lipschitz functions on space $(\mathcal{X}, \rho)$. One of the core merits of this dual representation is that both infimum and supremum exist. That is, the optimal coupling π^* can be obtained by minimizing the value on the LHS of Eq. 16, while any optimal function $f^* \in \mathcal{F}_L$ satisfies $f^*(X) - f^*(Y) = \rho(X, Y)$ for all (X, Y) in the support of π^*, which is attained by maximizing the RHS of Eq. 16.

Wasserstein GAN (WGAN) WGAN has been recently proposed in [2], and the aim is to learn the generator network for any random vector such that the Wasserstein distance is minimized between the resulting distribution p_g of the generated samples and the real distribution g_x underlying the observed data points. Replacing JSD with 1-Wasserstein distance , which is known to induce a much weaker topology than JSD (and other strong distances), yields a more sensible distance function and provides stable gradients. This, in turn, renders WGAN to be better suited for generative modeling.

Formally, the utility function of the minimax game of WGAN is:

$$\mathcal{L}_{WGAN}(p_x, p_g) = \min_G \max_{D \in \mathcal{D}} \mathop{\mathbb{E}}_{x \sim p_x}[D(x)] - \mathop{\mathbb{E}}_{z \sim p_g}[D(G(z))] \tag{17}$$

where $\mathcal{D}$ is any subset of 1-Lipschitz functions on $\mathcal{X}$. Minimizing above objective w.r.t. the generator parameters minimizes the EMD $W(q, p)$, i.e., $\mathcal{L}_{WGAN}(p_x, p_g) \leq \text{Wass}^1(p_x, p_g)$. The loss of the generator in WGAN $\mathcal{L}_g$ is:

$$\mathcal{L}^g_{WGAN} = -\mathbb{E}_{z \sim p_g}[D(G(z))] \tag{18}$$

and the loss of the discriminator (or more precisely, the critic [2] since it is a real-valued function here and is no longer trained to classify) is:

$$\mathcal{L}^d_{WGAN} = \mathop{\mathbb{E}}_{z \sim p_g}[D(G(z))] - \mathop{\mathbb{E}}_{x \sim p_x}[D(x)] \tag{19}$$

This family of functions $\mathcal{D}$ is specified in [2] via neural networks, and then weight clipping is used to enforce Lipschitz continuity.

WGAN-Gradient Penalty (GP) However, as Arjovsky et al. note, the networks' capacities become limited due to the weight clipping and there could be gradient vanishing problems in the training. In the later work [23], they present more concrete examples to illustrate the perils of the weight clipping and propose an alternative way of imposing the Lipschitz continuity by introducing a gradient penalty term into discriminator as:

$$\mathcal{L}^{d}_{WGAN-GP} = \mathop{\mathbb{E}}_{z\sim p_g}[D(G(z))] - \mathop{\mathbb{E}}_{x\sim p_x}[D(x)] + \lambda \mathop{\mathbb{E}}_{\hat{x}\sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \quad (20)$$

where the last term is the constraint with a penalty on the gradient norm of the discriminator output w.r.t. its input, an alternative way to enforce the Lipschitz constraint, and $p_{\hat{x}}$ is defined as the distribution over $\hat{x} = \delta x + (1-\delta)y$ for $\delta \sim U[0,1]$, i.e., a straight line between two points respectively from real data distribution p_x and generator distribution p_g [23].

We note that recent works introduce additional penalty into WGAN-GP for improving the training of WGAN, e.g., [38, 47]. Since how to improve WGAN is not the main concerns of this work, we leave the investigating and comparison of various regularization methods as future work.

4.1.3 Trajectory distribution approximate

In this paper we consider a trajectory-user linking problem from a perspective of TDA, formally defined as:

Definition 2 Trajectory Distribution Approximation (TDA): Given a trajectory dataset $\mathcal{T}$ produced by a set of users $\mathcal{U}$, the task of a TDA is to generate a specific number of trajectories $G(u_i)$ (e.g., using GANs) for each $u_i \in \mathcal{U}$ to approximate the underlying trajectory distribution of each user for ultimately improving the performance of mapping $\mathcal{T} \mapsto \mathcal{U}$. The number of generated trajectories for users is not mandatory to be equal in TDA, which can be allocated to balance the label distribution in the dataset.

Note that the generative model used in TDA is not limited to GANs. One could employ other sophisticated generative models such as Hidden Markov Chians (HMM) [7] Variational Auto-Encoders (VAE) [26] or PixelRNN [44] for synthetic trajectory generation.

4.2 TGAN model – overview and main components

The main aspects of the TGAN model for trajectory distribution approximation are shown in Figure 3. The TGAN model is inspired by the recent advances in WGAN [2] and TULER [20], and consists of three parts. First, the generator samples from a Gaussian distribution (p_g) and generates synthetic trajectories using a LSTM network. Next, the generated and real trajectories are fed into a CNN discriminator that classifies them into real or fake, and feedbacks the generator to update the distribution p_g. The generated trajectory classified as real will be considered as a new training sample to augment the training set.

Specifically, the underlying trajectory distribution of a user u_i can be expressed as $p_x(u_i)$, under which we build the following three models which, when combined, can help addressing the TUL problem:

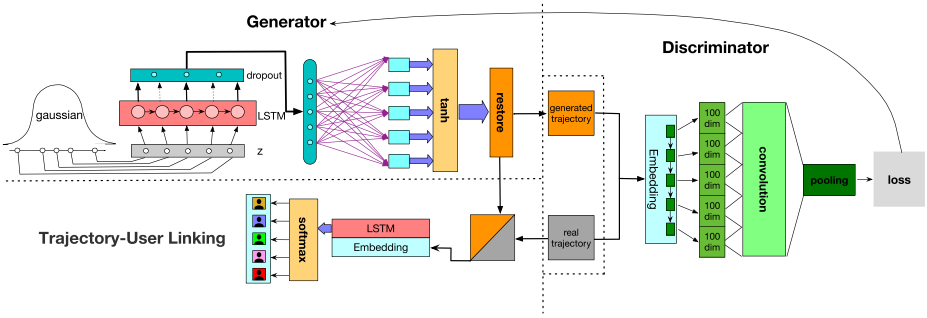


Fig. 3 Overview of TGAN: To augment an existing training set (trajectories with known users generating them), we initially generate trajectories based on some randomly picked distributions (e.g., uniform or Gaussian) using a LSTM-based generator. The likelihood of them being sampled from a true distribution of specific users is determined based on the discriminator (e.g., a CNN network), trained on the existing real trajectories. During the process, the initial distribution is increasingly optimized to approximate the underlying distribution. After convergence (reaching the equilibrium between the generator and the discriminator), all subsequently generated trajectories together with real ones will be used as the final training pool to learn user-trajectory mapping

Generative Trajectory Model *G* it attempts to generate a sequence of check-ins $\widetilde{T} = \{l_1, ..., l_k, ..., l_m\}, l_k \in \mathcal{L}$, where $\mathcal{L}$ is the vocabulary of candidate check-ins (the POIs user u_i has visited). At each time step k, the generator produces a check-in l_k. The goal of this model is to approximate the true distribution of trajectories corresponding to user u_i as much as possible.

Discriminative Trajectory Model *D* aims at discriminating the generated trajectory sample from the real ones and provides the guidance for improving the generator. It is in fact a binary classifier characterized by a CNN and produces a probability indicating how likely a generated sample is produced upon a real distribution $p_x(u_i)$.

Trajectory-User Linking Model[3] L: in essence, this is a multi-class classification model used to link the trajectories, both synthetic and real, to users who generated them. Similar to [20], TUL model is a RNN based neural network learning the intrinsic moving patterns of users.

However, due to the nature of non-continuity and non-differentiability of the classical GAN, the gradient cannot be back-propagated to the generator. Trajectory is inherently constituted of discrete tokens (POIs). Thus, we first embeds POIs in a continuous low-dimensional space (c.f Section 3.2) which makes the step of discrete tokens generating differentiable.

4.3 Trajectory synthesis

We now proceed with discussing each component of TGAN in a greater detail.

[3]Strictly speaking, TUL model is not part of TGAN but a method used for verifying the effectiveness of TGAN.

Trajectory Generator TGAN employs a LSTM as the generator to learn a latent vector z upon which a synthetic trajectory $\widetilde{T}$ is generated. The probability of generating a trajectory with a length of h is

$$p(\widetilde{T}|z) = p(l_1|z) \prod_{t=2}^{h} p(l_t|l_1, \cdots, l_{t-1}, z) \tag{21}$$

where l_t is the t^{th} location in $\widetilde{T}$ and l_1 is sampled from some distribution (e.g., a Gaussian being used here). At each time step ($t \geq 2$) of the generating process, we seek to estimate a probability distribution over all the possible next POIs in the vocabulary given the previous locations using a LSTM [24].

To generate a trajectory, the generator takes l_t from the input sequence and l_{t-1} from the output POI probability distribution at the previous step, combined with previous hidden state $\mathbf{h}_{t-1}$ to update the hidden state $\mathbf{h}_t$, on top of which a softmax layer is used to produce the POI samples. All the other POIs in the trajectory are sequentially generated using the RNN, based on the previously generated locations – until the end-of-trajectory token. Note that in order to obtain a fully-differentiable generator, the input to the LSTM at each time step is the POI embedding vector $\mathbf{v}_t$ for l_t.

Note, however, that if we sample each l_t from its conditional distribution on the previously generated POIs, the performance of the generator would deteriorate as the trajectory becomes longer, due to the generation divergence of the POIs. To alleviate this problem, we leverage a procedure proposed in [31], called Professor Forcing (PF), for matching the generative trajectory with the ground-truth sequence during training. The basic idea of PF is to train the RNN generator to behave (both in its output and hidden states) the same as the input regardless of whether real training sequences or its self-generated ones are fed. Thus, we train two RNN models which share parameters with each other to produce the POIs – however, different from the original PF approach, we do not differentiate between the two RNN models. Thus, instead of directly discriminating the closed-loop (i.e., the generative) and open-loop (i.e., real trajectory training) distribution used in [31], we leverage PF as a joint training process to pretrain the RNN generator, followed by a CNN based discriminator that distinguishes the generated trajectories from the real data.

Trajectory Discriminator The generator deterministically transforms a latent vector into a length h trajectory through a LSTM, where a softmax nonlinearity at the output is directly passed into a discriminator. The discriminator architecture we choose for TGAN is based on a convolutional neural network, similar to the choices of *critic* in WGAN [31] and discriminator in SeqGAN [50]. It involves 2 convolution layers and 2 max-pooling operations over the entire trajectory – represented by a matrix $\mathbf{D} \in \mathbb{R}^{d \times L}$, where the trajectory has length of L and would be padded with 0 if necessary. On top of the convolutional feature vector, a softmax layer is leveraged to produce a probability representing the likelihood of a trajectory being generated from a real user or from the generator.

We also note that RNN has been widely used in modeling discriminator in sequential GAN, e.g., text generation [53]. However, since the generator (as well as the TULER later used for evaluating) are deployed with RNNs, we prefer to introduce variety into TGAN to alleviate the imbalance problem between generator and discriminator inherent in GAN – because training a too strong discriminator may prevent or early-stop updating the generator. In addition, we use trajectory embedding in TGAN to address the non-differentiability of

the discrete variables generation in RNN generator, while an alternative method is Gumbel-softmax [25] which can also make the steps of discrete tokens generating differentiable. Furthermore, unlike text or dialogue generation which can be estimated by language syntax and coherence in a reinforcement manner [53], the semantics of generated trajectories can not be directly evaluated due to the sparsity of the check-ins and the lack of "reward" for the generated ones.

4.4 Generation evaluation and training details

As discussed above, in addition to discriminator used as a constraint and guidance of generator, we should include another method to evaluate the performance of individual trajectory generation. In this work, we employ the TUL task [20], which aims at identifying and linking trajectories to users who generate them, as the generation performance evaluation. That is, we augment the training trajectory data of each user with the generated trajectories and test the TUL accuracy. The basic idea of this choice is that if we can correctly approximate individual trajectory distribution with TGAN, the TUL accuracy should be improved using the augmented training set.

Specifically, we use a Bi-directional RNN [43] for the TUL classification, in which we run two LSTMs in parallel: one is on the sequential check-in embedding vectors, and the other is on the reverse embedding vectors. We note that although Bi-directional RNN performs best for the TUL task in [20], we also evaluate TUL with other TUL classification methods and our experiments show that TGAN can, to varying extent, improve the performance of both neural networks based TUL and the baselines introduced in [20] such as LCSS and SVM.

We train TGAN to generate trajectories for each user, in which the discriminator is a binary classifier to determine how likely a trajectory being generated from a real underlying distribution. This may incur a high computation complexity due to training G and D separately for each user – especially so when the number of users (labels) is large. An alternative method is to incorporate labels in TGAN using a semi-supervised learning [40], where generated samples are added to dataset with a new label "fake" ($y = K + 1$, where K is the original number of labels).

We note that although the label-including method in [40] demonstrates certain efficiency in image classification, we found that this does not improve the performance of TUL; and even decreases the linking accuracy for some methods. Essentially, the method simply adds samples from the generator to the data set and labels them as "fake" which, thereafter, is to be minimized when learning the classifiers. The rationale behind this approach is that we can now learn from unlabeled data, as long as we know that it corresponds to one of the K classes of real data – which is therefore maximized during training. Also, it may benefits the training efficiency by adding the negative ("fake") samples, since the discriminator can directly distinguish them with the label.

However, we can leverage this method as a pre-training process – i.e., to train a model to initialize the G and D for each user trajectory generation.

Due to the sparsity of LBSN datasets, we generate trajectories in an incremental way – the generator starts producing sequences of length 1 and increases to length 2 and 3 and until the maximum length of L. We also leverage a parameter $\alpha (1 \leq \alpha \leq L)$ to control the length of the generated trajectory. In addition, we use another random binary parameter β to indicate the generation conditioned on a ground truth (i.e., whether an actual trajectory or

not) after reaching the equilibrium. For trajectory generation, we train the generator using the RMSProp algorithm with a learning rate at 0.55×10^{-3} and the decay rate at 0.9.

Finally, note that in both TGAN and TUL implementations, we concatenate all check-in locations of each user to form a trajectory which will be further divided into sub-trajectories based on time intervals. To capture richer semantics of individual moving patterns, and for fairness in our experimental evaluation (i.e., comparison with [20]), we set the time interval to 6 hours.

5 Experimental evaluation

We now proceed with describing first the settings and metrics used over several real-world datasets in our experiments, and then present in detail the experimental observations regarding the advantages of our proposed methods, both TULER and TDA.

5.1 Settings and metrics

All models were implemented in tensorflow on Ubuntu 16.04 operating system. The machine is a server with two Intel(R) Xeon(R) CPU E5-2630, 128GB memory, and a single GTX 2080Ti GPU. Following are the specifications of the neural networks and training setup used in TGAN:

- The POI embedding leverages Skip-gram model with window size of 10 and negative samplings with size 10.
- The dimensionality of POI embedding is 100.
- The generator G:
  - LSTM has one hidden layer with 200 neuron units.
  - The dropout rate is 0.5 and the batch size is set to 50.
- The discriminator D:
  - The CNN for discriminator is a 1-D CNN
  - It has 2 convolution layers followed by 2 max-pooling operations.
- The training time ratio of D and G is 5 : 1, that is, we simultaneously train 5 iterations of D as well as 1 iteration of G to obtain a rapid-growth discriminator compared to the generator – when the discriminator is too poor, the gradient propagating into the generator RNN could be detrimental.
- For each user in the dataset, we use TDA to adaptively generate approximately half of their existing training trajectories and incorporate them into our TUL training.

Metrics The performance of TGAN-based trajectory generation is evaluated on the TUL task, for which several methods have already been proposed – e.g., Longest Common Sub-Sequence (LCSS), Linear Discriminant Analysis (LDA), SVM with linear kernel and Bi-directional RNN (Bi-TULER). We use two metrics for evaluating the quality of TUL solutions – ACC@K and macro-F1:

$$\text{ACC@}K = \frac{\#\ correctly\ identified\ trajectories\ @K}{\#\ trajectories} \tag{22}$$

$$\text{macro-F1} = 2 \times \frac{macro\text{-}P \times macro\text{-}R}{macro\text{-}P + macro\text{-}R} \tag{23}$$

both of which have been widely used in earlier works (cf. [20]). The purpose of ACC@K is to evaluate the trajectory-user linking precision, whereas macro-F1 is the typical harmonic mean of the precision@1 (macro-P) and recall@1 (macro-R). We compare the result of TUL with and without TGAN to validate the performance of our trajectory approximation methods.

5.2 Datasets

Three publicly available LBSN datasets were used in our experiments: Gowalla, Brightkite [13] and Geolife [56]. Most of the users have very sparse check-ins in the original datasets which makes the TUL impossible for those users having only several check-ins in a long period, e.g., a month. For Gowalla and Brightkite, we selected top 201 and 92 users who have most check-in data, respectively. For each user, we concatenate all check-in locations to form a trajectory which will be further divided into sub-trajectories based on the time interval we define (i.e., 6 hours). We note that this is a relatively large dataset for testing classification models. For example, the widely used text classification datasets are either binary (e.g., IMDB dataset) or have at most 14 classes (e.g., DBpedia dataset) – which is much less than the number of classes (users) in the TUL settings. We then used 90% of data for training (and generation) and 10% for testing, similarly to [20].

For Geolife dataset, 20% of the data was chose (randomly) for generation, which was fed into TGAN to learn the trajectory distribution, and the other 80% were left for testing. The reason of this partition is that the daily trajectories of the users in Geolife are relatively stable and 20% of training data (combined with generated trajectories) are enough to capture the trajectory distribution of the users. Just as importantly, if more trajectories are used in training (correspondingly the testing data decreases), the generated trajectories may not fit the distribution of data for testing very well even though TGAN correctly matches the training data. This phenomenon happens because of the conflicts of TGAN and TULER and the data characteristics. First, TGAN tries to match the trajectory distribution of training data while the goal of TULER is to accurately classifying the test trajectories by learning the motion patterns of training data. Imagine an extreme case that we use 99% data of a user for training TGAN and use 1% for testing the TGAN via a TUL model. No matter how better we match the training data (and therefore generate more plausible trajectories and learn a better TUL model), it cannot assure that the TULER can be more accurate on the 1% data that randomly selected for testing. Moreover, unlike Gowalla and Brightkite which are sparse and irregular, the data in Geolife is relatively stable and dense (regular daily travel data captured by GPS equipments), a small portion of which can train a good learner to distinguish the mobility patterns. Finally, since the trajectory in the original Geolife dataset contains only the GPS points (longitude and latitude), we cluster all points to obtain 3,646 POIs – that is, we save the two digits after the decimal point of longitude and latitude. The summary of datasets used in our experiments is presented in Table 1.

Table 1 Attributes description of datasets. U: the number of users; S/T: the number of trajectories in the training/testing set; C: the number of unique check-ins; t_r: the range of the number of check-ins across all sub-trajectories

Dataset	U	S/T	C	t_r
Gowalla	201	17,654/2,063	10,956	[1,131]
Brightkite	92	17,934/2,039	2,120	[1,184]
Geolife	20	1,690/6,763	3,646	[1,172]

5.3 Baselines

To demonstrate the superior performance of TULER, we define several baselines in the sequel:

- **LCSS: The Longest Common Sub-Sequence** [49]: It is widely used to measure the trajectory similarity by matching the longest common sub-trajectory between two trajectories via dynamic programming. In this work, LCSS is used as a solution to the TUL problem for the purpose of linking an unlinked testing sub-trajectory to the user of its most similar trajectory among all trajectories.
- **LDA: Linear Discriminant Analysis-based**: LDA has demonstrated a great performance in many text classification tasks. In this paper, we embed the trajectory into one-hot vectors following the method proposed in [29], and then we use Singular Value Decomposition (SVD) to decompose the within-class scatter matrix. We note that other alternative matrix solvers – e.g., eigenvalue decomposition and least squares – have also been tested but are not reported here due to their lower performance in comparison with SVD in our experiments.
- **SVM: Support Vector Machine**: We applied SVM on trajectory embeddings as a multi-class classifier. We observed that the linear kernel shows a superior performance over the other kernels such as RBF and Gaussian.
- **Hidden Markov Model (HMM)**: A classical dynamic Bayesian network in which the system being modeled is assumed to be a Markov process with unobserved hidden states.
- **Random Forest (RF)**: We utilize the multi-class classification algorithm based on random forest to predict the generator for each trajectory.
- **Gradient Boosting Decision Tree (GBDT)**: GBDT is a boosting-based machine learning model that ensembles a set of "weak classifiers" for classifying trajectories, and is widely used in competitions such as Kaggle and KDDCup.
- **Multi Layer Perceptron (MLP)**: An MLP constitutes the simplest and most traditional architecture for classification.

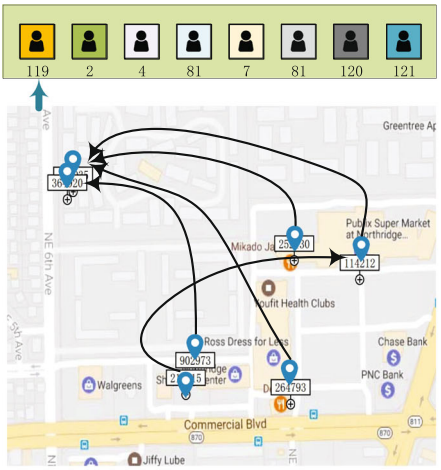
We note that TULER also has five variants based on different types of RNN models, one layer of RNN with LSTM or GRU (TULER-LSTM and TULER-GRU), stacked RNNs (TULER-LSTM-S and TULER-GRU-S) and Bidirectional LSTM (Bi-TULER).

We omit the comparison to another recent work TULVAE [57], which addresses the TUL problem with hierarichical variational autoencoders [26] in a *semi-supervised* manner, due to the unfair comparison reason, i.e., TGAN learns the underlying human mobility in a totally *unsupervised* manner. Furthermore, there is a fundamental difference between TGAN and TULVAE: TGAN is a trajectory generation method aiming at augmenting the sparse LBSN dataset with synthetic trajectories, while TULVAE tries to addressing the data sparsity problem by leveraging the *unlabeled* data.

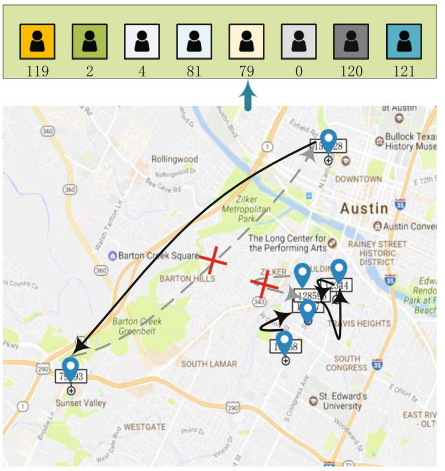
To illustrate the benefits of TGAN, we compare it with two baselines that could be used to generate individual trajectories. The first one is continuous-time Random Walk (RW) [9] which samples randomly from the visited POIs of individuals at each time step. The second model is Markov Chain (MC) which we learn the adjacent POI transition probability to generate trajectories.

5.4 Results on trajectory-user linking

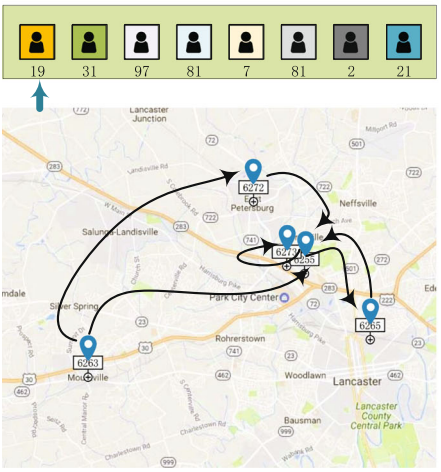
We first report the experimental results on Trajectory-User Linking.



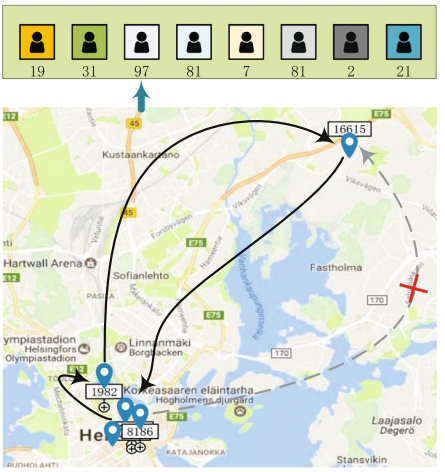
(a) Observation on Gowalla.



(b) Observation on Gowalla.



(c) Observation on Brightkite.



(d) Observation on Brightkite.

Fig. 4 Examples of inferring users of trajectories using TULER

5.4.1 Empirical results

Before delving into the details of the performance comparisons between our proposed approach and baselines, we first visualize the outcomes of randomly selecting several trajectories from two different datasets and their predicted users using TULER (See Fig. 4). Figures 4a and c show that TULER successfully identifies the trajectories produced by user No.119 and No.19, respectively. However, it fails to do so for the trajectories generated by user No. 79 and No. 97 in Fig. 4b and d (marked as red $\times$). This is mainly due to the extreme sparsity of the sequences, with only 1 or 2 check-ins involved. This is a still an open and challenging problem in TUL. That is, how can we understand the sparse check-in trajectories? A natural idea is to incorporate more characteristics of trajectories and the first

Table 2 The list of parameters and values used in this paper

Parameters	Tuned	Commonly suggested
Dimensionality	250	100-300
Hidden size	300	250-1000
Learning rate	0.95×10^{-3}	0.85×10^{-3}-0.1
Dropout rate	0.5	0-1
Stacked TULER	2	≥ 2
LDA Matrix solver	SVD	SVD, LSQR, *etc*
SVM Kernel	Linear	Linear, RBF, *etc*

extensions would be to include the timestamps of check-ins (which might help reduce the complexity).

Table 2 lists the common guidelines of choosing values for parameters as well as the optimal ones, tuned for both TULER and baselines. The results in the rest of this section are reported based on these optimal parameter values (unless otherwise specified).

5.4.2 Performance comparison

The performance comparison between TULER and baselines (the best is shown in **bold**, and the second best is underlined) are summarized in Tables 3 and 4 for Gowalla and Brightkite, respectively.

We observe that on Gowalla dataset, our model TULER with Bidirectional LSTM consistently outperforms the other methods in terms of accuracy, while the TULER with one layer of LSTM achieves the best result with respect to the Macro-F_1 metric. Specifically, Bi-TULER yields 36.9%, 28.1% and 13.8% improvement compared to LCSS, LDA and SVM on ACC@5 metric.

Similar performance by TULER also holds on the Brightkite dataset. As can be seen, TULER-LSTM and Bi-TULER achieve the best and the second best results in terms of accuracy, respectively. LDA obtains the highest Macro-F_1, but TULER based methods still achieve comparable results.

Table 3 Performance of various methods on the dataset of Gowalla

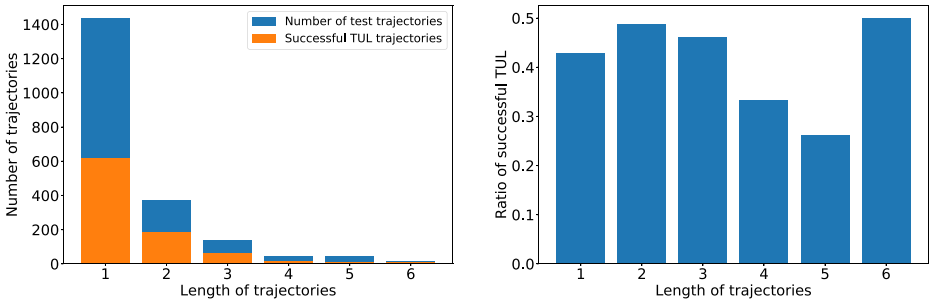
Method \ Metric	ACC@1	ACC@5	Macro-F_1
LCSS	32.65	46.13	27.02
LDA	37.86	49.28	34.08
SVM	41.25	55.50	34.32
HMM	40.54	53.22	33.59
RF	39.86	52.47	32.23
GBDT	42.34	57.55	34.67
MLP	41.45	56.28	33.68
TULER-LSTM	<u>45.03</u>	<u>63.15</u>	**35.77**
TULER-GRU	41.06	60.37	31.46
TULER-LSTM-S	41.68	57.03	32.43
TULER-GRU-S	40.10	59.08	32.37
Bi-TULER	**45.70**	**65.68**	<u>35.56</u>

Table 4 Performance of various methods on the dataset of Brightkite

Method \ Metric	ACC@1	ACC@5	Macro-F_1
LCSS	30.12	39.13	23.02
LDA	40.50	53.38	**39.38**
SVM	42.07	61.46	36.59
HMM	41.49	60.23	35.62
RF	40.82	60.11	35.08
GBDT	43.26	62.25	36.74
MLP	42.58	61.85	36.29
TULER-LSTM	**45.00**	**64.64**	38.18
TULER-GRU	43.29	62.49	34.86
TULER-LSTM-S	43.19	61.56	38.71
TULER-GRU-S	41.38	60.68	38.71
Bi-TULER	44.91	63.91	38.20

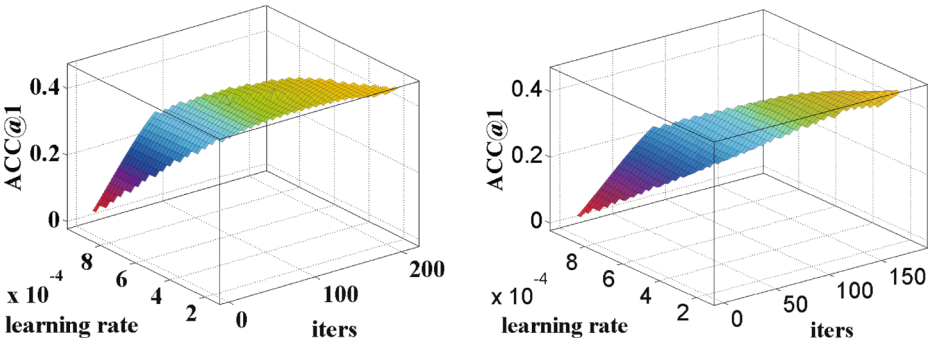
We note that we have observed something counter-intuitive in the results: namely, the stacked TULER (such as stacked LSTM and GRU) – which primarily aims at capturing characteristics of longer trajectory sequences – actually falls behind the one layer TULER, as well as the Bi-TULER (although each of them outperforms the baselines). A possible explanation is that the trajectory segmentation in TULER has truncated the original long trajectories into short sub-trajectories – whereas for longer trajectories, stacked TULER performs the best.

One could be tempted to hypothesize that the length of the trajectory is what affects the performance of TULER – however, upon running additional experiments we observ that the situation is more complicated, as illustrated by Fig. 5. Firstly, after the truncation (i.e., within 6 hours), most of the trajectories have ≤ 6 POIS. This may have some intuitive justification in the sense that most individuals would not check-in to more than 6 POIs within 6 hours. Secondly, one can not readily claim that there is a simple relationship between the length of the trajectories and TUL results. One of the main reasons is that the lenght of the trajectories is not evenly distributed – i.e., the number of trajectories in the dataset significantly decreases with the trajectories' length (cf. Fig. 5a). This, in turn, implies that one cannot simply claim that the performance of TUL will decrease with the trajectory length.



(a) Successful TUL. (b) Ratio of successful TUL.

Fig. 5 Impact of trajectory length (Gowalla). The X-axes in both sub-figures indicate the length in terms of 6-hours segments



(a) Gowalla. (b) Brightkite.

Fig. 6 Sensitivity of parameters under TULER-LSTM

We also note that the impact of the length is twofold: (1) the longer the trajectory, the more context/patterns we can draw from the training data – and, therefore the better performance of TUL; (2) the performance of the TULERs, which are RNN based models, may decrease with the length of the trajectories (c.f Fig. 5b), especially when the testing trajectories are very different than the training data. Thus, in additoin to sophisticated TULER models, improving the TUL performance may require investigating the dependies on dataset characteristics.

5.4.3 Model robustness

Some parameters like the number of iterations and learning rate might have significant impact on the model performance. Figure 6 illustrates that the accuracy of TULER is proportional to the number of iterations. In addition, a small value of learning rate (e.g., 0.95×10^{-3}) can obtain a higher classification accuracy.

5.5 Results on trajectory generation

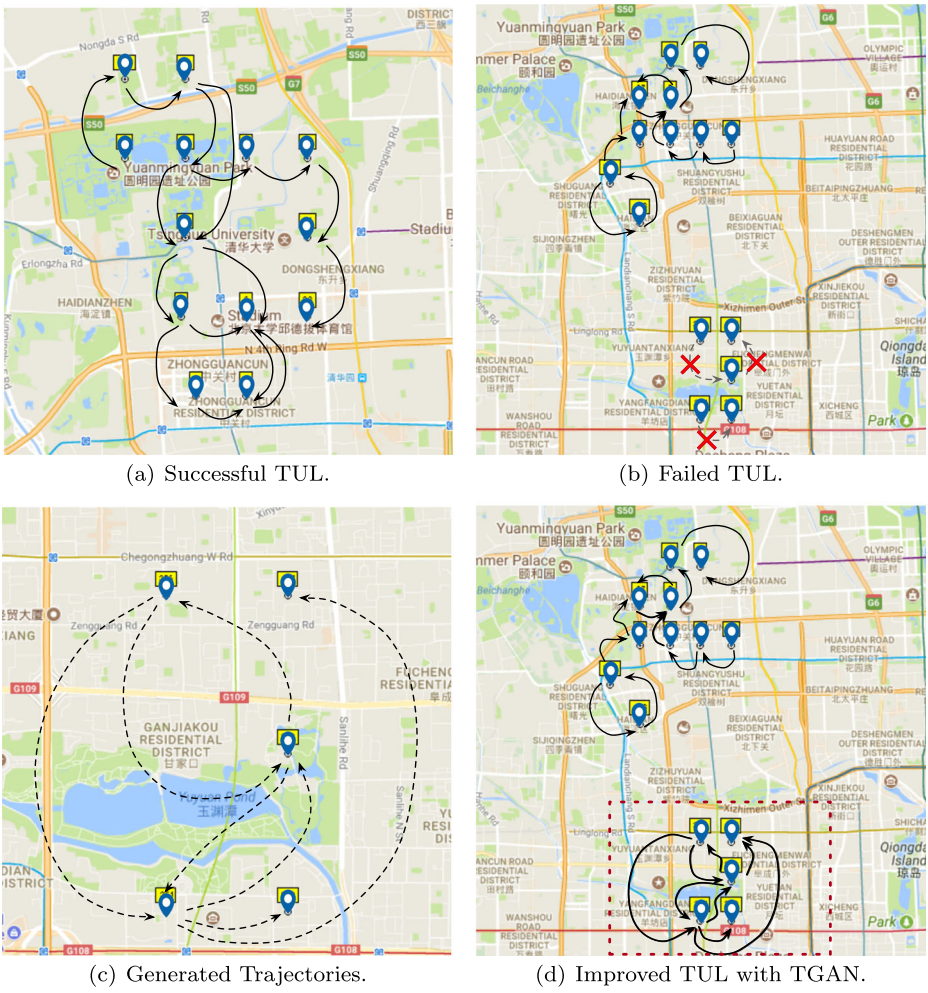
We now present our evaluation of the effects of TDA solution for the TUL problem.

5.5.1 Case study

We first present a visualization study of several trajectories from Geolife dataset and the effect of TGAN. Figure 7 illustrates: (a) A successful TUL case: a dense trajectory by user No.35. (b) A Failed TUL case: some check-ins with red '$\times$' marked are not correctly linked to their user No.62. (c) A generated trajectory using TGAN for user No.62 (dashed arrows). (d) TGAN incorporates the generated trajectory in (c) into training and successfully link all check-ins in (b) to user No.62 (equivalently classify the trajectory into user No.62). The Bi-directional RNN is employed for TUL here.

5.5.2 Quantitative observations

We also quantitatively evaluate the model performance in terms of ACC@K and micro-F1 on three datasets. Bi-TULER is chosen since it is proved to perform well overall from previous results. Table 5 summarizes the following observations: (1) TGAN improves the TUL



(a) Successful TUL. (b) Failed TUL.

(c) Generated Trajectories. (d) Improved TUL with TGAN.

Fig. 7 Visualization examples of using TGAN to improve TUL on Geolife data

accuracy across all TUL methods with generating adversarial trajectories. The improvement originates from learning the underlying individual trajectory distribution and augmenting the size of training data, which supports our initial motivation; (2) the performance of TGAN relies on individual motion patterns and the characteristics of datasets. For example, TGAN greatly improved the TUL results on Geolife dataset especially using Bi-directional RNN – 88.6% for ACC@1 and 155.6% for Macro-F1. The reason is that the trajectories in Geolife are densely and periodically distributed, compared to the sparse check-ins in Gowalla and Brightkite; although, as can be observed, TGAN still achieves satisfactory improvement on the later two datasets.

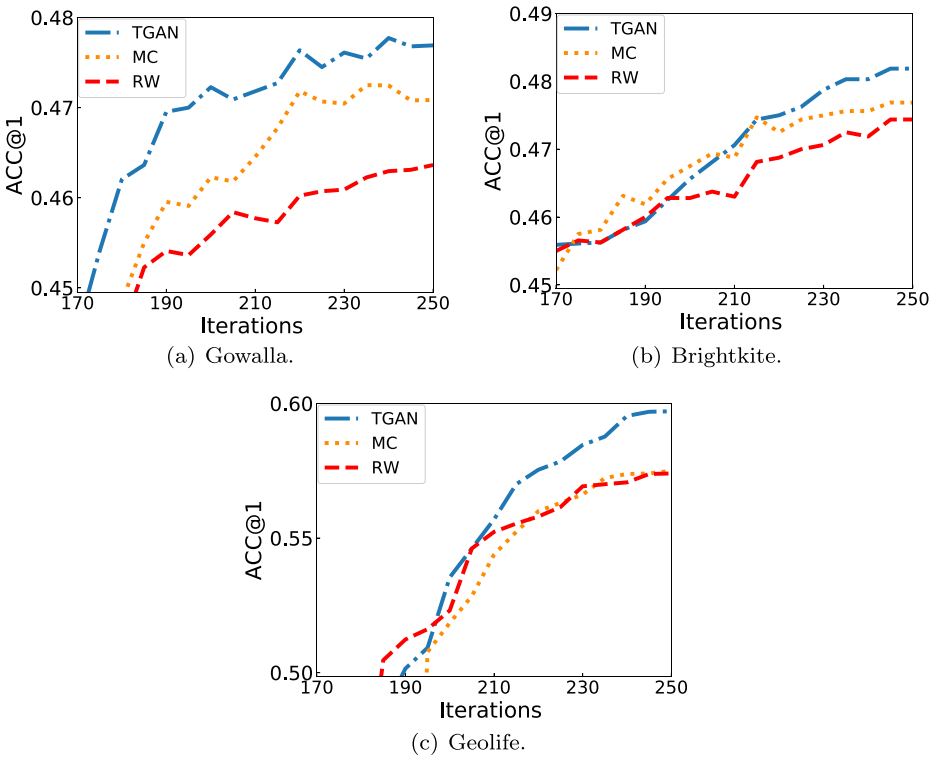
There still remains one question not answered yet: the quality of generated trajectories for TUL comparing to some heuristic methods, such as Random Walk (RM) and Markov Chain (MC) methods. The results, illustrated in Fig. 8, demonstrate the superiority of TGAN on TUL in terms of ACC@1 for all three datasets. Note that it requires time for TGAN

Table 5 Performance comparison of Trajectory-User Linking (TUL) on three datasets

Dataset	Method \ Metric	ACC@1	ACC@3	ACC@5	ACC@10	Macro-F1
		Without TGAN				
Geolife	LCSS	13.35	27.20	32.69	39.15	11.71
	LDA	14.11	31.14	37.96	48.74	13.81
	SVM	15.70	33.21	38.83	56.12	15.77
	Bi-TULER	31.69	59.49	75.22	90.56	19.19
Gowalla	LCSS	32.65	42.00	46.13	51.00	27.02
	LDA	37.86	46.10	49.28	53.30	34.08
	SVM	41.25	51.85	55.50	60.35	34.32
	Bi-TULER	45.70	60.08	65.68	71.94	35.56
Brightkite	LCSS	30.12	36.31	39.13	44.17	23.02
	LDA	40.50	48.46	53.38	58.66	39.38
	SVM	42.07	54.66	61.64	69.55	36.59
	Bi-TULER	44.91	58.08	63.91	72.86	38.20
		With TGAN				
Geolife	LCSS	**13.80**	**28.71**	**34.28**	**40.97**	**11.88**
	LDA	**18.50**	**34.11**	**39.91**	**52.13**	**15.17**
	SVM	**19.45**	**33.74**	**40.98**	**57.94**	**16.83**
	Bi-TULER	**59.75**	**82.21**	**89.85**	**95.33**	**49.05**
Gowalla	LCSS	**35.25**	**45.40**	**51.00**	**55.35**	**29.86**
	LDA	**41.60**	**50.22**	**52.59**	**55.84**	**35.10**
	SVM	**43.80**	**55.80**	**59.35**	**64.40**	**36.13**
	Bi-TULER	**47.96**	**63.34**	**68.45**	**75.48**	**37.69**
Brightkite	LCSS	**32.02**	**39.58**	**43.68**	**49.83**	**25.16**
	LDA	**43.48**	**54.03**	**59.83**	**64.42**	**41.24**
	SVM	**44.80**	**60.23**	**67.16**	**74.72**	**38.54**
	Bi-TULER	**47.71**	**63.61**	**69.66**	**78.10**	**40.22**

to achieve the best performance while learning the underlying mobility patterns – which also means that as the training proceeds, TGAN converges to an equilibrium between the generator and the discriminator.

In fairness, we note that TGAN does have overheads in terms of running time for the training, in comparison with RW and MC. What we observed while running our experiments is that: (1) RW would only require a few seconds for sampling (i.e., generating) the trajectories; (2) MC model is also rather efficient in constructing the transition probability matrices (i.e., approximately 20-25 minutes); (3) TGAN, in contrast, would require up to a couple of hours of training time for matching the trajectories distribution and synthesizing the trajectories. However, in practice, the overhead of the offline training time in TGAN is amortized by the benefits of the accuracy offered when classifying a trajectory, as well as the relevant enrichment of the datasets.



(a) Gowalla. (b) Brightkite.

(c) Geolife.

Fig. 8 Performance comparison of TUL using TGAN with two baselines: RW and MC

6 Concluding remarks

We addressed the Trajectory-User Linking (TUL) problem – an important task for many LBSN applications, targeting the identification of potential users who could have generated location-based trajectories. We presented an RNN based model called TULER which, unlike traditional models mainly based on trajectory similarity measurement and classification, is designed to capture the dependency of check-ins and to infer the latent patterns of user-trajectory interactions. Experiments conducted on three publicly available datasets show that TULER achieves the significant performance improvement, when compared to state-of-the-art baselines.

In addition, to alleviate data sparsity problem during the training phase of TUL, we introduced the Trajectory Distribution Approximation (TDA) problem and proposed the TGAN – a generative adversarial samples-based individual trajectory generation algorithm. TGAN learns the underlying moving patterns of the users, aiming to improve the performance of identifying human mobility. TGAN is a first attempt towards addressing the TDA problem and, as demonstrated, it can be an effective way with augmenting the training trajectories in location-based datasets.

Our ongoing works focuses on two aspects. Firstly, we would like to improve the TGAN performance to target different applications in a more context-aware manner about the datasets – e.g., LBSN in regular vs. abnormal circumstances, such as crowd after concerts

or games [36, 52]. Secondly, we plan to investigate the issue of formally incorporating the uncertainty in the TUL problem [19].

References

1. Alharbi B, Qahtan A, Zhang X (2016) Minimizing user involvement for learning human mobility patterns from location traces. In: Proceedings of the AAAI Conference on Artificial Intelligence
2. Arjovsky M, Chintala S, Bottou L (2017) Wasserstein generative adversarial networks. In: International conference on machine learning (ICML)
3. Bao J, He T, Ruan S, Li Y, Zheng Y (2017) Planning bike lanes based on sharing-bikes' trajectories. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)
4. Barthélemy M (2011) Spatial networks. Physics Reports
5. Bashir FI, Khokhar AA, Schonfeld D (2007) Object trajectory-based activity classification and recognition using hidden markov models. IEEE Trans Image Process 16(7):1912–1919
6. Bhargava P, Phan T, Zhou J, Lee J (2015) Who, what, when, and where: Multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data. In: International world wide web conferences (WWW)
7. Bindschaedler V, Shokri R (2016) Synthesizing plausible privacy-preserving location traces. In: IEEE Symposium on security and privacy (s&p). IEEE, pp 546–563
8. Brockmann D (2013) The hidden geometry of complex, network-driven contagion phenomena. Science
9. Brockmann D, Hufnagel L, Geisel T, Whitaker RM (2006) The scaling laws of human travel. Nature
10. Chen D, Ong CS, Xie L (2016) Learning points and routes to recommend trajectories. In: International conference on information and knowledge management (CIKM)
11. Chen X, Duan Y, Houthooft R, Schulman J, Sutskever I, Abbeel P (2016) Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Neural information processing systems (NIPS)
12. Cheng C, Yang H, Lyu MR, King I (2013) Where you like to go next: successive point-of-interest recommendation. In: International joint conference on artificial intelligence (IJCAI)
13. Cho E, Myers SA, Leskovec JA (2011) Friendship and mobility: User movement in location-based social networks. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)
14. Chung J, Gulcehre C, Cho KH, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv e-prints
15. Damiani ML, Gùting RH (2014) Semantic trajectories and beyond (tutorial). In: International conference on mobile data management (MDM)
16. Deville P, Song C, Eagle N, Blondel VD, Barabási AL, Wang D (2016) Scaling identity connects human mobility and social interactions. Proceedings of the National Academy of Sciences of the United States of America (PNAS)
17. Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh EJ (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. Proceedings of the VLDB Endowment (PVLDB)
18. Dodge S, Weibel R, Ahearn SC, Buchin M, Miller JA (2016) Analysis of movement data. International Journal of Geographical Information Science
19. Gal Y, Ghahramani Z (2016) Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: Proceedings of The 33rd International Conference on Machine Learning, pp 1050–1059
20. Gao Q, Zhou F, Zhang K, Trajcevski G, Luo X, Zhang F (2017) Identifying human mobility via trajectory embeddings. In: International joint conference on artificial intelligence (IJCAI)
21. González M. C., Hidalgo CA, Barabási A. L., Barabási A. L. (2008) Wang: Understanding individual human mobility patterns Nature
22. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Neural information processing systems (NIPS)
23. Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V, Courville A (2017) Improved training of wasserstein gans. In: Neural information processing systems (NIPS)
24. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation
25. Jang E, Gu S, Poole B (2017) Categorical reparameterization with gumbel-softmax. In: International conference on learning representations (ICLR)

26. Kingma DP, Welling M (2014) Auto-encoding variational bayes. In: International conference on learning representations (ICLR)
27. Kolouri S, Park SR, Thorpe M, Slepcev D, Rohde GK (2017) Optimal mass transport - signal processing and machine-learning applications. IEEE Signal Proc Mag 34(4):43–59
28. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Neural information processing systems (NIPS)
29. Lai S, Liu K, He S, Zhao J (2016) How to generate a good word embedding. IEEE Intelligent Systems
30. Lai S, Xu L, Liu K, Zhao J (2015) Recurrent convolutional neural networks for text classification. In: Proceedings of the AAAI Conference on Artificial Intelligence
31. Lamb AM, ALIAS PARTH GOYAL AG, Zhang Y, Zhang S, Courville AC, Bengio Y (2016) Professor forcing: a new algorithm for training recurrent networks. In: Neural information processing systems (NIPS)
32. Li J, Cai Z, Yan M, Li Y (2016) Using crowdsourced data in location-based social networks to explore influence maximization. In: IEEE Conference on computer communications (INFOCOM)
33. Liu P, Qiu X, Huang X (2016) Recurrent neural network for text classification with multi-task learning. In: International joint conference on artificial intelligence (IJCAI)
34. Liu Q, Wu S, Wang L, Tan T (2016) Predicting the next location: a recurrent model with spatial and temporal contexts. In: Proceedings of the AAAI Conference on Artificial Intelligence
35. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. Computer Science
36. Naboulsi D, Fiore M, Ribot S, Stanica R (2016) Large-scale mobile traffic analysis: a survey Communications Surveys and Tutorials. IEEE Communications Society 18(1):124–161
37. Pelekis N, Theodoridis Y (2014) Mobility Data Management and Exploration. Springer, Berlin. https://doi.org/10.1007/978-1-4939-0392-4
38. Petzka H, Fischer A, Lukovnicov D (2018) On the regularization of wasserstein gans. In: International conference on learning representations (ICLR)
39. Ruder S (2016) An overview of gradient descent optimization algorithms. CoRR arXiv:1609.04747
40. Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X, Chen X (2016) Improved techniques for training gans. In: Neural information processing systems (NIPS)
41. Song C, Koren T, Wang P, Barabási A. L. (2010) Modelling the scaling properties of human mobility. Nature Physics
42. Song C, Qu Z, Blumm N (2010) Limits of predictability in human mobility. Science
43. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Neural information processing systems (NIPS)
44. Van Oord A, Kalchbrenner N, Kavukcuoglu K (2016) Pixel recurrent neural networks. In: International conference on machine learning (ICML)
45. Villani C (2003) Topics in optimal transportation. Graduate studies in mathematics
46. Villania C (2008) Optimal transport. Springer Science & Business Media
47. Wei X, Gong B, Liu Z, Lu W, Wang L (2018) Improving the improved training of wasserstein gans: a consistency term and its dual effect. In: International conference on learning representations (ICLR)
48. Yang D, Zhang D, Zheng VW, Yu Z (2015) Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. Transactions on Systems, Man, and Cybernetics: Systems
49. Ying JC, Lee WC, Weng TC, Tseng VS (2011) Semantic trajectory mining for location prediction. In: Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems
50. Yu L, Zhang W, Wang J, Yu Y (2017) Seqgan: sequence generative adversarial nets with policy gradient. In: Proceedings of the AAAI Conference on Artificial Intelligence
51. Zhang J, Zheng Y, Qi D (2017) Deep spatio-temporal residual networks for citywide crowd flows prediction. In: Proceedings of the AAAI Conference on Artificial Intelligence
52. Zhang J, Zheng Y, Qi D, Li R, Yi X, Li T (2018) Predicting citywide crowd flows using deep spatio-temporal residual networks. Artif Intell 259:147–166
53. Zhang Y, Gan Z, Fan K, Chen Z, Henao R, Shen D, Carin L (2017) Adversarial feature matching for text generation. In: International conference on machine learning (ICML)
54. Zheng Y (2015) Trajectory data mining: An overview. ACM Transactions on Intelligent Systems and Technology (TIST)
55. Zheng Y, Li Q, Chen Y, Xie X, Ma WY (2008) Understanding mobility based on gps data. In: Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)
56. Zheng Y, Zhang L, Xie X, Ma WY (2009) Mining interesting locations and travel sequences from gps trajectories. In: International world wide web conferences (WWW)

57. Zhou F, Gao Q, Trajcevski G, Zhang K, Zhong T, Zhang F (2018) Trajectory-user linking via variational autoencoder. In: International joint conference on artificial intelligence (IJCAI)
58. Zhu Y, Zheng Y, Zhang L, Santani D, Xie X, Yang Q (2012) Inferring taxi status using gps trajectories. Computer Science
59. Zhuang C, Jing Yuan N, Song R, Xie X, Ma Q (2017) Understanding people lifestyles: Construction of urban movement knowledge graph from gps trajectory. In: International joint conference on artificial intelligence (IJCAI)

Fan Zhou received his BS degree in Computer Science from Sichuan University, China, in 2003, and his MS and PhD degrees from University of Electronic Science and Technology of China, in 2006 and 2011, respectively. He is currently an Associate Professor at the School of Information and Software Engineering, University of Electronic Science and Technology of China. His research interests include machine learning, spatio-temporal data management and social network knowledge discovery.

Ruiyang Yin received the BS and MS degrees in Software Engineering from University of Electronic Science and Technology of China (UESTC) in 2016 and 2019, respectively. He is currently working in Baidu Inc.. His research interests include recommendation system, spatio-temporal data mining and deep generative learning.

Goce Trajcevski received the B.Sc. degree from the University of Sts. Kiril i Metodij, and the M.S. and Ph.D. degrees from the University of Illinois at Chicago. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Iowa State University. His main research interests are in the areas of spatio-temporal data management, uncertainty and reactive behavior management in different application settings, and incorporating multiple contexts. In addition to a book chapter and three encyclopedia chapters, he has coauthored over 140 publications in refereed conferences and journals. His research has been funded by the NSF, ONR, BEA, and Northrop Grumman Corp. He was the General Co-Chair of the IEEE ICDE 2014, ACM SIGSPATIAL 2019, the PC Co-Chair of the ADBIS 2018 and ACM SIGSPATIAL 2016 and 2017, and has served in various roles in organizing committees in numerous conferences and workshops. He is an Associate Editor of the ACM TSAS and the Geoinformatica Journals.

Kunpeng Zhang received the Ph.D. degree in computer science from Northwestern University. He is a Researcher in the area of large-scale data analysis, with particular focuses on social data mining, image understanding via machine learning, social network analysis, and natural language processing. He is currently an Assistant Professor with the Department of Information Systems, Smith School of Business, University of Maryland, College Park, MA, USA. He has published papers in the area of social media, artificial intelligence, network analysis, and information systems on top conference and journals. He serves as program committees for many conferences and Associate Editors for journals.

Jin Wu received her BS degree in Automatic control from University of Electronic Science and Technology of China (UESTC), in 1993, and her MS and PhD degrees from UESTC, in 1996 and 2004, respectively. She is currently an Associate Professor at the University of Electronic Science and Technology of China. Her research interests include machine learning, Knowledge Mapping, software development techniques and process technology.

Ashfaq Khokhar received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985, the M.S. degree in computer engineering from Syracuse University in 1989, and the Ph.D. degree in computer engineering from the University of Southern California in 1993. He served two years as a Visiting Assistant Professor with the Department of Computer Sciences (CS) and the School of ECE, Purdue University. In 1995, he joined the ECE Department, University of Delaware, where he first served as an Assistant Professor and then as an Associate Professor. In 2000, he joined the CS and ECE Departments at UIC, and served first as Associate Professor and then as Professor and the Director of Graduate Studies of ECE until 2013. From 2013 to 2017, he served as a Professor and the Department Chair of ECE with the Illinois Institute of Technology, Chicago. He is currently a Professor and the Palmer Department Chair with the Department of Electrical and Computer Engineering (ECE), Iowa State University. He has authored over 270 technical papers and book chapters in refereed conferences and journals in the areas of healthcare data mining, wireless networks, multimedia systems, data mining, and high performance computing. His research centers on high performance solutions for diverse application area including, computational biology, health care data mining, and content-based multimedia modeling. He is a fellow of IEEE for his contributions to multimedia computing and databases. He was a recipient of the NSF CAREER award in 1998. He has received numerous outstanding paper awards, and has served as the Program Chair and technical program committee members of leading IEEE/ACM conferences.

Affiliations

Fan Zhou[1] · Ruiyang Yin[1] · Goce Trajcevski[2] · Kunpeng Zhang[3] · Jin Wu[1] · Ashfaq Khokhar[2]

Goce Trajcevski
gocet25@iastate.edu

Kunpeng Zhang
kpzhang@umd.edu

Jin Wu
wj@uestc.edu.cn

Ashfaq Khokhar
ashfaq@iastate.edu

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu, 610054, China

[2] Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA

[3] Department of Decision, Operations & Information Technologies, University of Maryland, College park, MD 20742, USA