GRLSTM: Trajectory Similarity Computation with Graph-Based Residual LSTM

Silin Zhou[1*], Jing Li[2*], Hao Wang[3], Shuo Shang[1,4†], Peng Han[1†]

[1] University of Electronic Science and Technology of China
[2] Harbin Institute of Technology, Shenzhen, China
[3] School of Computer Science, Wuhan University, China
[4] Sichuan Artificial Intelligence Research Institute, Yibin, 644000, China
{zhousilinxy, wanghao.hku, jedi.shang}@gmail.com, {jingli.phd, penghan_study}@hotmail.com

Abstract

The computation of trajectory similarity is a crucial task in many spatial data analysis applications. However, existing methods have been designed primarily for trajectories in Euclidean space, which overlooks the fact that real-world trajectories are often generated on road networks. This paper addresses this gap by proposing a novel framework, called GRLSTM (Graph-based Residual LSTM). To jointly capture the properties of trajectories and road networks, the proposed framework incorporates knowledge graph embedding (KGE), graph neural network (GNN), and the residual network into the multi-layer LSTM (Residual-LSTM). Specifically, the framework constructs a point knowledge graph to study the multi-relation of points, as points may belong to both the trajectory and the road network. KGE is introduced to learn point embeddings and relation embeddings to build the point fusion graph, while GNN is used to capture the topology structure information of the point fusion graph. Finally, Residual-LSTM is used to learn the trajectory embeddings.To further enhance the accuracy and robustness of the final trajectory embeddings, we introduce two new neighbor-based point loss functions, namely, graph-based point loss function and trajectory-based point loss function. The GRLSTM is evaluated using two real-world trajectory datasets, and the experimental results demonstrate that GRLSTM outperforms all the state-of-the-art methods significantly.

Introduction

With the ubiquitousness of GPS-enabled devices, massive trajectory data is being collected at an unprecedented rate. A trajectory portrays the spatial-temporal motion of an object over a while. Trajectory similarity computation is an essential function in many real-world applications, such as trajectory clustering (Lee, Han, and Whang 2007), anomaly trajectory detection (Meng et al. 2019), route planning(Shang et al. 2012; Chen et al. 2019; Chen, Shang, and Guo 2020; Chen et al. 2021), transportation optimizations (Zheng et al. 2013; Yang et al. 2021; Zheng et al. 2021; Li et al. 2021), and trajectory matching (Shang et al. 2014, 2017a).

Many existing metrics (Yi, Jagadish, and Faloutsos 1998; Vlachos, Gunopulos, and Kollios 2002; Chen and Ng 2004; Chen, Özsu, and Oria 2005) adopt pairwise matching and rely on dynamic programming to compute the optimal alignments, which results in quadratic time complexity $O(n^2)$, where n is the average length of trajectories. Thus, these methods are not desirable for large-scale trajectory data. To address this issue, some recent studies (Li et al. 2018; Yao et al. 2019; Zhang et al. 2020) propose to leverage embedding-based similarity computation methods, where each trajectory is encoded as a latent vector with deep learning models. This means that the trajectory similarity can be computed in linear time with vector similarity computation.

Although these methods mentioned above are effective for measuring trajectory similarity in Euclidean space, they are not as effective as expected for computing trajectory similarity on the road network. Road network (Wang et al. 2021) refers to a road system consisting of various roads that are interconnected and interwoven into a network. Previously, a handful of studies (Chen et al. 2010; Shang et al. 2017b; Li, Cong, and Cheng 2020) have attempted to solve the problem of trajectory similarity computation on road networks. Especially, these studies adopt hand-crafted heuristic approaches to align trajectories to the road network and define some effective similarity functions (Shang et al. 2017b) to measure trajectory similarity over the road network. However, these

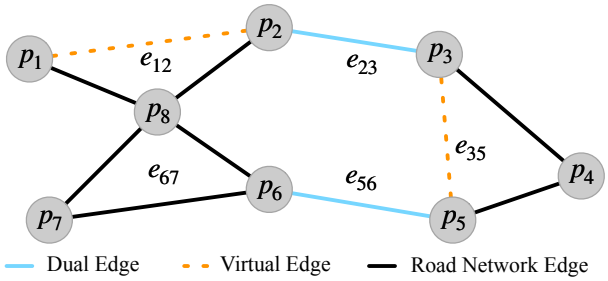

Figure 1: An example of multi-relation of points on the road network. $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ is a point set of the road network and $\langle p_1, p_2, p_3, p_5, p_6, p_7\rangle$ is a trajectory sequence. There are three types of edges: (i) Virtual edges (e.g., $\{e_{12}, e_{35}\}$) exist solely in trajectories and do not exist in the road network, (ii) Dual edges (e.g., $\{e_{23}, e_{56}\}$) exist in both the road network and trajectories. (iii) Road network edges (e.g., e_{67}) exist solely in the road network.

*These authors contributed equally.
†Shuo Shang and Peng Han are corresponding authors.

methods still suffer from the issue of high computation complexity.

To solve the above challenges, the latest framework GTS (Han et al. 2021) utilizes deep learning method and Graph Neural Network (GNN) (Kipf and Welling 2017) on the road network and achieves decent performance. However, GTS leaves out two issues. First, GTS only considers the road network information and ignores the trajectory information during the graph processing. A key point is that adjacent points in a trajectory are not necessarily adjacent on the road network due to data loss or inconsistent sampling rates. As shown in Figure 1, *virtual edges* $\{e_{12}, e_{35}\}$ are in trajectories while do not exist on the road network. On the other hand, GTS uses the optimization strategy on trajectories without considering point optimization.

To leverage the point information of trajectories and road networks, we propose a novel framework, namely GRLSTM. In particular, we take into account that each point has multiple relations in the road network and trajectories by building a knowledge graph. Then, we apply the KGE method to the knowledge graph and further build a fusion graph by k-nearest selection. Next, the GAT (Velickovic et al. 2018) is introduced to capture the topology structure information of the fusion graph. The final trajectory representation is learned by the multi-layer LSTM. To solve the vanishing gradient problem between layers in the deep layer LSTM, we design a novel module, namely Residual-LSTM, by incorporating the residual network (He et al. 2016) into the multi-layer LSTM. In addition, we design two new neighbor-based point-aware loss functions to optimize GRLSTM: *(i) Graph-based point loss*. To optimize the graph-level point embedding, we consider that points connected in the graph should be similar. *(ii) Trajectory-based point loss*. We consider that points in a trajectory as a similar set. In other words, point embeddings in the same trajectory should be similar. This is based on the observation that a trajectory is usually sampled from the same vehicle.

In short, the main contributions of this paper are summarized as follows:

- We propose **GRLSTM**, a novel trajectory similarity computation framework on road networks. It models multi-relation of points with knowledge graph using KGE. We introduce the residual network into multi-layer LSTM to learn trajectory embeddings, which can solve gradient vanishing problem.
- We design two new neighbor-based point-aware loss functions (i.e., graph-based point loss and trajectory-based point loss) to effectively train GRLSTM.
- We conduct extensive experiments on two real-world datasets. The experiments show significant improvements of our method over state-of-the-art methods.

Related Work

Trajectory Similarity without Deep Learning

In traditional methods, there are two main streams of trajectory similarity computation methods. One is based on the Euclidean space where dynamic programming is used to analyze trajectory segments, such as Dynamic Time Warping (DTW) (Yi, Jagadish, and Faloutsos 1998), longest common subsequence (LCSS) (Vlachos, Gunopulos, and Kollios 2002), edit distance with real penalty (ERP) (Chen and Ng 2004), and edit distance on real sequences (EDR) (Chen, Özsu, and Oria 2005). These methods commonly utilize some optimization (Rakthanmanon et al. 2012) techniques to reduce computation time but will be disturbed by the noise points, resulting in a decrease in the final accuracy. The other trajectory similarity computation is based on road networks. In these methods, all coordinate points of each trajectory need to be mapped into the road network to find the corresponding node. Then the similarity function is used for similarity computation between trajectories. The early methods simply apply the shortest path algorithm and set similarity to calculate the trajectory similarity. Later, Shang et al. (Shang et al. 2017b, 2018, 2019) proposed a joint similarity function after considering the similarity of spatial and temporal, and accelerated the calculation through pruning and indexing techniques. There was also appearing to design new functions (Wang et al. 2018) to compute the similarity between trajectories. (Zheng et al. 2021) propose a distributed in-memory management framework to accelerate trajectory similarity queries. However, the traditional methods based on road constraints are either too simple or affected by high computational complexity, which is difficult to be applied in practice because of the issue of handling large-scale data.

Trajectory Similarity with Deep Learning

In recent years, deep learning models have shown excellent performance on trajectory similarity computation. There are some studies on applying deep learning to spatial data analysis (Han et al. 2019, 2020; Zhao et al. 2020). Due to the characteristics of trajectory sequence, some existing studies apply the deep learning model of natural language processing to generate trajectory representation. The similarity between trajectories can be obtained by measuring the relationship between trajectory representations. The encoder-decoder model is adopted by Li et al. (Li et al. 2018) to obtain trajectory embedding. Yao et al. (Yao et al. 2019, 2020) introduces the attention mechanism into the spatial network and uses the pair-wise distance to assist the learning processing, making the performance more effective. To improve the learning quality of trajectory embeddings, Zhang et al. (Zhang et al. 2020) devise several new loss functions. Han et al. (Han et al. 2021) first introduces the GNN into trajectory similarity computation on the road network.

Preliminaries

In this section, we define the basic data structure of the road network and trajectory. Then we give a brief description of our target problem.

Trajectory with Road Network

A road network is a graph $G = (V, E)$, where V and E are the sets of nodes and edges, respectively. Each node $v \in V$ is a point featured with a geographic coordinate, representing an endpoint of a road segment or a road intersection. Each edge $e = \langle u, v \rangle \in E$ represents a road segment connecting

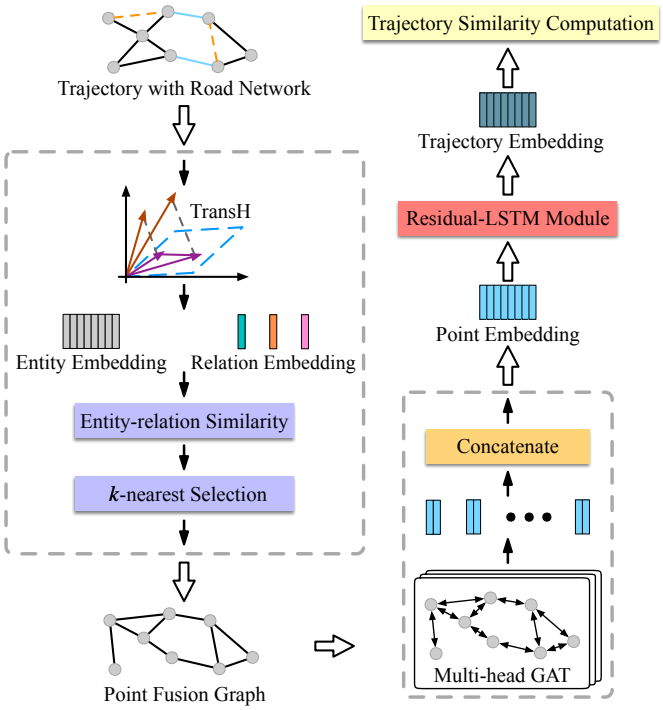

Figure 2: The overall of GRLSTM.

points u and v. A length-n trajectory $\tau = \langle v_1, v_2, ..., v_n \rangle$ in the road network graph G consists of an ordered sequence of n points (i.e., their geographical coordinates).

Problem Definition

Given a graph $G = (V, E)$ of a road network and a trajectory set $T = \{\tau_1, \tau_2, ..., \tau_m\}$, for $\forall \tau_a \in T$, the trajectory similarity computation is to find $\tau_b \in T$ such that τ_b is most similar to τ_a, where $a \neq b$.

Graph-based Residual LSTM

In this section, we introduce a novel framework GRLSTM for trajectory representation learning, which includes three parts: point knowledge graph embedding, fusion graph embedding, and Residual-LSTM. Figure 2 shows the overall framework of our model.

Knowledge Graph Embedding on Point

Each point belongs to both the road network and the trajectory. Due to different sampling rates of devices or data loss, the trajectory is usually not a continuous sequence on the road network, which means that adjacent points within the trajectory are not necessarily adjacent to each other on the road network. In other words, there are some *virtual edges* existing in the trajectory while not in the road networks, as edges $\{e_{12}, e_{35}\}$ shown in Figure 1.

To solve the above problem, we combine road networks and trajectories to construct a knowledge graph. The knowledge graph is a semantic network that can reveal the relations between entities. Specifically, given a trajectory dataset and road network graph, we construct a road network-trajectory knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, where $\mathcal{R}$ has three relation types: road network edge r_n, trajectory virtual edge r_t, dual edge r_{nt}. Therefore, each $e = \langle u, v \rangle \in \mathcal{E}$ with a relation $r \in \mathcal{R}$ can be built a triplet (e.g., $\langle u, r_n, v \rangle$).

To effective reveal the relations between entities (i.e, points), we use TransH (Wang et al. 2014) to learn entity and relation embeddings. The basic idea of TransH is to use different hyperplanes to represent different relation spaces and regard relations as translation operation on hyperplanes. Formally, given a triplet $\langle u, r, v \rangle$, where $u, v \in \mathcal{V}$ and $r \in \mathcal{R}$, the corresponding entity embeddings e_u and e_v are first projected to the hyperplane $\boldsymbol{w}_r$ with constraint $||\boldsymbol{w}_r||_2 = 1$, which is detailed as follows:

$$e_{u_\perp} = e_u - \boldsymbol{w}_r^T e_u \boldsymbol{w}_r, \tag{1}$$

$$e_{v_\perp} = e_v - \boldsymbol{w}_r^T e_v \boldsymbol{w}_r, \tag{2}$$

where $e_{u_\perp}$ and $e_{v_\perp}$ denote projection embeddings of e_u and e_r, respectively. Then, we use a score function $f(\cdot)$ to compute the difference of the triplet, which can be denoted as:

$$f(e_u, e_v) = ||e_{u_\perp} + h_r - e_{v_\perp}||_2^2, \tag{3}$$

where h_r is the translation embedding on the hyperplane. The desired result of the score function $f(\cdot)$ is a lower value if the relation of the triplet is correct, and a higher value if the opposite is true.

So far, each entity (i.e., point) and each relation have been represented by embeddings. To capture the correlation of points within different relations, we design a *entity-relation similarity* function $s(\cdot)$ to compute embedding similarity between point u and point v within specific relation r. Inspired by the KGE algorithms, the distance between u and v can be written as:

$$d_r(e_u, e_v) = ||e_u + e_r - e_v||, \tag{4}$$

where e_u, e_v denote the learned embeddings of point u and point v, respectively, e_r is the learned relation embedding between u and v and $||\cdot||$ represents L_2-norm. The lower result of the distance function $d_r(\cdot)$ means the closer distance. To show higher similarity for the closer distance, we compute similarity between u and v as follows:

$$s(e_u, e_v) = \boldsymbol{e}^{-d_r(e_u, e_v)}, \tag{5}$$

To integrate trajectory and road network information, we construct the point fusion graph $G_f \in \mathbb{R}^{|V| \times |V|}$ by similarity. Here, we use k-nearest selection to obtain point v_i neighbor set $\mathcal{N}_s(v_i)$ based on similarity to keep sparsity on graph and decrease noisy. Then, we have the following graph:

$$G_f(i, j) = \begin{cases} 1 & \text{if point } v_j \in \mathcal{N}_s(v_i) \text{ or } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

G_f is a new graph that has new edges constructed by similarity. Note that the adjacent points in the trajectory are not necessarily adjacent in the fusion graph either. However, the point of the fusion graph can contain both road network and trajectory characteristics by modeling the point with KGE as well as similarity. Thus, G_f can better represent the properties of each point in fusion graph, which is not available in the road network graph.

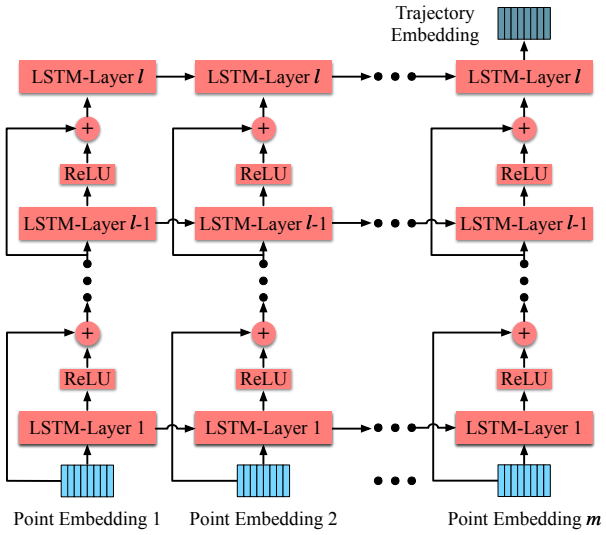

Figure 3: The residual-LSTM module.

Graph Embedding on Point

To capture the topology in the fusion graph G_f, we adopt the graph attention network (GAT) (Velickovic et al. 2018) to learn graph embedding. GAT uses message passing graph neural network to implement the *aggregate* and *update* processing. Formally, given the points embedding $P = \{p_1, p_2, ..., p_{|V|}\}$, we updates node embedding as follows:

$$\alpha_{ij} = \frac{\exp\Big(\text{LeakyReLU}\Big(\mathbf{a}^T\Big[\mathbf{W}p_i || \mathbf{W}p_j\Big]\Big)\Big)}{\sum_{k\in\mathcal{N}_i}\exp\Big(\text{LeakyReLU}\Big(\mathbf{a}^T\Big[\mathbf{W}p_i || \mathbf{W}p_k\Big]\Big)\Big)}, \quad (7)$$

$$\hat{p}_i = \sigma\Big(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}p_j\Big), \quad (8)$$

where $\mathbf{W}$ and $\mathbf{a}$ are learnable parameters, σ is the activation function, $||$ represents concatenation, and $\mathcal{N}_i$ is the neighbor set which can be obtained from G_f. To improve the feature expressiveness of GAT, we use multi-head GAT as suggested by (Velickovic et al. 2018). We concatenate the output embeddings of multi-head GAT, which is presented as follows:

$$\hat{p}_i = ||_{h=1}^{H}\sigma\Big(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^h\mathbf{W}^h p_j^h\Big), \quad (9)$$

where H is the number of the heads, $||$ is the concatenation operation, and σ is the activation function.

So far, new point embeddings are learned by graph embedding layers, which contain the rich graph structure information from G_f. Then we take these point embeddings as the input of the Residual-LSTM module to learn trajectory embeddings.

Multi-Layer LSTM with Residual Network

The trajectory data is a typical sequential structure, thus RNN-based method is an appropriate method. In our implementation, we choose LSTM to learn trajectory embeddings. Specifically, we first transform each point in a trajectory to graph-based point embeddings. Then we use LSTM to fuse sequence embeddings. Here, we take the last time step output of LSTM as the final trajectory embeddings. To achieve better performance, we further apply the multi-layer LSTM in our model. However, simply stacking the layers of the LSTM causes serious gradient vanishing problem, making a sharp drop in convergence and getting worse learning results. In addition, the multi-layer LSTM also makes the training time longer. Therefore, we need to find a way to solve the gradient vanishing problem of multi-layer LSTM without increasing the training time.

Inspired by ResNet (He et al. 2016), we introduce the residual network into the multi-layer LSTM. ResNet has made a big splash in computer vision and effectively solve the gradient vanishing problem of deep model learning. Due to its excellent performance, we integrate ResNet into the multi-layer LSTM in our model. For every timestep i in the multi-layer LSTM, we denote the residual block in Residual-LSTM by follows:

$$\hat{p}_i^{(l)} = \mathcal{H}\big(\hat{p}_i^{(l-1)}\big) + \text{ReLU}\left(\mathcal{F}\big(\hat{p}_i^{(l-1)}; \mathbf{W}_i^{(l)}\big)\right). \quad (10)$$

where $\hat{p}_i^{(l-1)}$ and $\hat{p}_i^{(l)}$ are the input and output of the layer l Residual-LSTM at timestep i, respectively. $\mathbf{W}_i^{(l)}$ denotes all learnable parameters of the time step i in the layer l. $\mathcal{F}(\cdot)$ is the residual function, which stands for the learned residuals. $\mathcal{H}(\cdot)$ reprensents the identity connection, which can be obtained by $\mathcal{H}(\hat{p}_i^{(l-1)}) = \hat{p}_i^{(l-1)}$. We take the last timestep hidden output of the last layer in the Residual-LSTM as the final trajectory embedding.

As shown in the Figure 3, we build the identity connection operation between different LSTM layers. We combine the input of the previous layer l with the output of the previous layer l as the input of the next layer $l+1$. This structure brings two benefits. On the one hand, the introduction of the residual network can solve the gradient vanishing problem in the multi-layer LSTM by identity connection. One the other hand, the residual block does not add any additional training parameters except for the element-wise addition and activation operation. Consequently, compared with the multi-layer LSTM, the increase of Residual-LSTM in training time is almost negligible.

Objective Functions

Embedding Similarity

After training, all trajectories in the training set are converted to trajectory embeddings. In this study, we use the dot-product to measure similarity between trajectory embeddings. Suppose embeddings of trajectories τ_i and τ_j are t_i and t_j, respectively. The trajectory embedding similarity score can be computed as follows:

$$s_t(\tau_i, \tau_j) = t_i^T t_j, \quad (11)$$

Likewise, we can define the point embedding similarity score using the dot-product. Suppose embeddings of points v_i and v_j are p_i and p_j respectively. The point embedding similarity score can be defined as:

$$s_p(v_i, v_j) = p_i^T p_j, \quad (12)$$

Then, we can define objective function to optimize our model by these similarity functions.

Point-Aware Loss Function

The Figure 1 shows that points have different properties on the road network, and the point fusion graph G_f is constructed based on these properties. Inspired by these different relations, we define two neighbor-based point-aware loss functions (i.e., graph-based point loss and trajectory-based point loss) to optimize point embeddings. Here, we first define two types of point neighbor relations to better detail our neighbor-based point-aware loss functions.

Graph-Based Point Neighbors. We define the graph-based point neighbors as the first-order neighbors on the fusion graph. Intuitively, each node on the graph has high similarity to its first-order neighbors. We denote the first-order neighbors set of a point as $\mathcal{N}_{gp}$, which $|\mathcal{N}_{gp}| = k$.

Note that the number of k-nearest selection has a significant effect on the number of the first-order neighbors on the fusion graph. To reduce these influence, we apply pair-wise method within sampling which is widely used in other ranking-based applications, to design the graph-based point loss function. Formally, given a trajectory set $T = \{\tau_1, \tau_2, ..., \tau_m\}$ and $\tau_i = \left\langle v_1^i, v_2^i, ..., v_{|\tau_i|}^i \right\rangle \in T$, we define graph-based point loss function as follows:

$$L_{gp} = -\sum_{i=1}^{m}\sum_{j=1}^{|\tau_i|} \log \sigma(s_p(v_j^i, v_{pos}^i) - s_p(v_j^i, v_{neg}^i)), \quad (13)$$

where σ is sigmoid function, v_j^i is a point of trajectory τ_i, v_{pos}^i is a positive point sampled from $\mathcal{N}_{gp}(v_j^i)$, and $v_{neg}^i \notin \mathcal{N}_{gp}(v_j^i)$ is a negative sample from fusion graph.

Trajectory-Based Point Neighbors. According to the previous analysis, points that are adjacent in the trajectory are not necessarily adjacent on the road network, which also is true for the fusion graph G_f. However, there is a high similarity between adjacent points in the trajectory, such as the movement trend of a vehicle. Specifically, given a trajectory $\tau = \langle v_1, v_2, .., v_{|\tau|} \rangle$ and a point $v_i \in \tau$, we define the previous point and the next point of v_i as the trajectory-based point neighbors, where v_1 only has the next point and $v_{|\tau|}$ only has the previous point. We denote trajectory-based point neighbors as $\mathcal{N}_{tp}$ and $|\mathcal{N}_{tp}| \in \{1, 2\}$.

Similarly, we use the pair-wise to design trajectory-based point loss function. Formally, given a trajectory set $T = \{\tau_1, \tau_2, ..., \tau_m\}$ and $\tau_i = \left\langle v_1^i, v_2^i, ..., v_{|\tau_i|}^i \right\rangle \in T$, we define the trajectory-based point loss function for a trajectory as follows:

$$L_{tp} = -\sum_{i=1}^{m}\sum_{j=1}^{|\tau_i|} \log \sigma(s_p(v_j^i, v_{pos}^i) - s_p(v_j^i, v_{neg}^i)), \quad (14)$$

where σ is sigmoid function, v_j^i is a point of trajectory τ_i, v_{pos}^i is a positive point sampled from $\mathcal{N}_{tp}(v_j^i)$, and $v_{neg}^i \notin \tau_i$ is a negative sample from fusion graph.

	Beijing	New York
#Nodes	28,342	95,581
#Edges	27,690	260,855
#Trajectories	5,621,428	10,541,288
Average Length	25	38

Table 1: Statistics of datasets.

Trajectory-Aware Loss Function

According to Equation 11, we can compute the similarity between two trajectories in the linear complexity. However, the goal of our work is to find the most similar trajectories, not just compute similarity scores. Therefore, we need define the trajectory-aware loss function to optimize our model.

Following (Han et al. 2021), we also use the pair-wise method to design the trajectory loss function. Formally, given a trajectory set $T = \{\tau_1, \tau_2, ..., \tau_m\}$, we define trajectory loss function for a trajectory as follows:

$$L_t = -\sum_{i=1}^{m} \log \sigma(s_t(\tau_i, \tau_{pos}) - s_t(\tau_i, \tau_{neg})), \quad (15)$$

where σ is sigmoid function, τ_{pos} is the most similar trajectory to τ_i and τ_{neg} is negative sample from $T / \{\tau_i, \tau_{pos}\}$.

Finally, the final objective function can be written as:

$$\mathcal{L} = L_{gp} + L_{tp} + L_t. \quad (16)$$

Experiments

In this section, we conduct a series of experiments. The main presentation includes the dataset, hyperparameter settings, and evaluation metrics. Then, we present a detailed analysis of the experimental results.

Experiment Settings

Datasets. We use two real-world road networks from different cities, Beijing and New York. Table 1 shows the details of the two datasets. In the Beijing road network, there are 28,342 nodes and 27,690 edges. For the trajectories in Beijing, we use taxi trajectories from the T-drive project [1]. These taxi trajectories are collected by taxi id, GPS coordinates, and timestamp from 10,357 taxis during several days. We split these trajectories by hours and we drop the short-length trajectories. Then we get 5,621,428 trajectories and the average length of Beijing trajectories is 25. We use the spatial similar function (Shang et al. 2017b) to create ground truths on the Beijing road network by GPS coordinates. In the New York road network, there are 95,581 nodes and 260,855 edges. For the trajectories in New York, we collect taxi driving data from the website [2]. We use the same preprocessing method to process these trajectories and get ground truths. Finally, we get 10,541,288 trajectories and the average length of these trajectories is 38. For both two datasets, we randomly split these data into training set, validation set, and test set in the ratio of [0.2, 0.1, 0.7].

[1] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample

[2] https://opendata.cityofnewyork.us

	Beijing					New York				
	HR@1	HR@5	HR@10	HR@20	HR@50	HR@1	HR@5	HR@10	HR@20	HR@50
Traj2vec	5.82%	10.57%	18.64%	28.83%	40.07%	4.95%	9.33%	16.13%	24.57%	37.24%
Siamese	6.33%	13.25%	20.17%	32.61%	45.58%	5.23%	11.12%	18.75%	27.74%	42.16%
NeuTraj	7.72%	19.78%	27.54%	39.63%	53.57%	6.15%	15.57%	23.28%	30.18%	48.43%
Traj2SimVec	7.81%	20.42%	29.17%	40.14%	56.75%	6.31%	17.03%	26.46%	32.52%	50.55%
GTS	*9.21%	*25.00%	*35.48%	*48.07%	*66.12%	*8.43%	*21.64%	*32.53%	*41.69%	*58.17%
GRLSTM	**12.96%**	**32.71%**	**44.38%**	**57.34%**	**74.02%**	**11.33%**	**27.85%**	**39.36%**	**48.12%**	**62.47%**
Improvement	**40.71%**	**30.84%**	**25.08%**	**19.28%**	**11.95%**	**34.40%**	**28.65%**	**20.99%**	**15.42%**	**7.39%**

Table 2: Experimental results on two datasets. The mark * indicates the compared baseline for improvements.

Hyperparameters. In our model, the dimension of embedding is 128 and the same settings are applied to the baselines. We use Adam (Kingma and Ba 2015) to optimize our model, and the learning rates are set as 5e-4 in the Beijing dataset and 1e-3 in the Newyork dataset. The training batch sizes are 256 in the Beijing dataset and 512 in the Newyork dataset. We set the layer of Residual-LSTM as 4. The number head of GAT is set as 8, and the number of GAT layer is 1. The k-nearest selection is set as 10 in Beijing dataset and 30 in Newyork dataset. Our model is implemented in Pytorch and trained on an Nvidia RTX3090 GPU. The detail of implementation can be referred in this website. [3]

Evaluation Metrics. Following the previous studies, we adopt HR@K as the major performance metric. The top-k hitting ratio (HR@k) examines the overlap between the returned top-k results and the ground truth. In our experiments, we adopt HR@1, HR@5, HR@10, HR@20 and HR@50 as the major performance index.

Baselines. In our experiment, we evaluate GRLSTM against the following competitors:

- Traj2vec (Yao et al. 2018): A sequence-to-sequence model to learn trajectory embeddings. The mean squared error is used as the loss function to optimize the model.
- Siamese (Pei, Tax, and van der Maaten 2016): A time series learning method over the Siamese network. It uses the cross-entropy loss function to train the model.
- NeuTraj (Yao et al. 2019): This framework modifies the structure of the LSTM to learn trajectory embeddings based on the grid.
- Traj2SimVec (Zhang et al. 2020): This method employs a new loss for the trajectory similarity by point matching.
- GTS (Han et al. 2021): The framework is the first model that performs the trajectory similarity computation with graph learning on the road network. It uses GCN and LSTM to learn trajectory embeddings.

Overall Performance

Table 2 reports experimental results against baselines. From the results, our analysis are summarized as follows:

Overall, our GRLSTM achieves the best performance on the two datasets, significantly outperforming all the state-of-the-art baseline methods in terms of top-k hitting ratio index. The average improvement of our model to the best baseline GTS is **25.57%** on Beijing dataset and **21.37%** on New York dataset. It is worth mentioning that GRLSTM outperforms GTS by relative HR@1 improvements of **40.71%** and **34.40%** on Beijing and New York datasets, respectively. These huge improvements can be attributed to four reasons: 1) We consider the multiple relations of point on the road network and apply KGE to learn entity embeddings and relation embeddings. Based on these embeddings, we further construct the point fusion graph where each point integrates the trajectory and road network information. 2) We use the multi-head GAT to learn graph-based point embeddings, which can capture properties of fusion graph from different hidden channels. 3). We introduce the residual network into multi-layer LSTM, which allows our model to train deeper LSTM and substantially improve performance. 4) We design two new neighbor-based point-aware loss functions, which can optimize our model from different point aspects.

Specifically, the loss function of Traj2vec is designed based on the fitting approach and and the accuracy is significantly reduced for more complex models. GRLSTM learns graph-based point embedding and uses these learned point embeddings to represent trajectory embeddings. Three pair-wise loss functions are adopted to optimize GRLSTM, which is more effective than the mean squared error loss in Trajvec, the cross-entropy in Siamese and the regression loss in NeuTraj. Moreover, the pair-wise loss in our model can learn the partial order relation which is more suitable for top-k recommendation. Traj2SimVec uses L_2-norm to compute the difference between embedding vectors, while our model uses the dot-product to compute the similarity, resulting in the linear complexity and better results. Compared to GTS which uses the simple GNN to aggregate the neighbors information, our model uses GAT that has a stronger ability to learn graph representations than the simple GNN. More importantly, our model considers multiple relations of points in road networks and trajectories, while GTS learns trajectory embeddings solely on road networks.

Ablation Experiment

We set up a variety of different experiments to verify the validity of our model components:

- **w/o FG**: We remove the fusion graph and only use the road network.

[3] https://github.com/slzhou-xy/GRLSTM

	HR@1	HR@5	HR@10	HR@20	HR@50
w/o FG	10.47%	27.92%	40.94%	52.16%	70.30%
w/o GAT	11.53%	30.63%	42.43%	55.51%	72.66%
w/o P	12.54%	31.43%	43.28%	56.23%	72.84%
w/o TP	12.06%	31.16%	42.54%	55.36%	71.99%
w/o ResNet	9.52%	25.57%	36.11%	48.52%	66.04%
GRLSTM	**12.96%**	**32.71%**	**44.38%**	**57.34%**	**74.02%**

Table 3: The ablation experiment.

- **w/o GAT**: In this method, we use GCN to replace the multi-head GAT.
- **w/o P**: We remove the graph-based point loss function and keep the other two loss functions.
- **w/o TP**: We remove the trajectory-based point loss function and keep the other two loss functions.
- **w/o ResNet**: We remove ResNet from GRLSTM and keep the multi-layer LSTM.

We use the Beijing dataset to conduct the ablation experiments and the results are shown in Table 3, and we have the following analysis and conclusions.

First, we find that the point fusion graph can significantly improve the performance. In fact, points in road networks do not contain any trajectory sequence information. However, each point in the fusion graph fully incorporates road network and trajectory information, which can better represent the multi-relation of points, because point fusion graph is constructed by the entity-relation similarity which is computed by entity (i.e., point) embeddings and relation embeddings from KGE.

Second, GAT achieves better performance than GCN, especially multi-head GAT. GCN uses degree information of neighbors to update the central node embedding, while GAT adopts the attention mechanism, which allows to aggregate neighbor node embeddings and update central node embeddings with attention factors and benefits from assigning different weights to the neighbors.

Third, our two novel neighbor-based point-aware loss functions have a great contribution to our model. For the graph-based point loss function, it is based on the graph and can optimize our model by using the neighbor information of the graph. During the training, it uses the first-order neighbor similarity in the graph as well as the pair-wise function to optimize point embeddings. For the trajectory-based point loss function, it uses the pair-wise function with positive and negative sampling to measure the similarity of points on trajectories.

Moreover, residual network improves performance of our model significantly, which shows the importance of residual network in the multi-layer LSTM. The multi-layer LSTM cause severe gradient vanishing when back-propagating, which leads to serious performance degradation problems. The training time of multi-layer LSTM keeps increasing with the number of layers. Through introducing the residual network into multi-layer LSTM, it can solve gradient vanishing problem while not increase the training time.

	HR@1	HR@5	HR@10	HR@20	HR@50
GLSTM-1	11.69%	30.30%	41.74%	54.41%	71.29%
GLSTM-2	11.03%	28.89%	39.89%	52.35%	69.31%
GRLSTM-2	11.66%	31.46%	43.30%	56.24%	72.80%
GLSTM-3	10.52%	27.22%	38.21%	51.96%	68.28%
GRLSTM-3	12.62%	32.49%	43.49%	57.25%	73.72%
GLSTM-4	9.52%	25.57%	36.11%	48.52%	66.04%
GRLSTM-4	**12.96%**	**32.71%**	**44.38%**	**57.34%**	**74.02%**
GLSTM-5	7.45%	21.35%	30.95%	42.69%	60.12%
GRLSTM-5	12.58%	32.20%	43.93%	56.77%	73.19%

Table 4: Residual-LSTM layers experiment.

Residual-LSTM Layers Experiment

To reveal the role of the residual network in multi-layer LSTM, we design the Residual-LSTM layers experiment based on the Beijing dataset. The results are shown in Table 4, where GRLSTM-l represents the l-layer LSTM with residual network, and GLSTM-l denotes the l-layer LSTM without residual network.

First, the residual network can significantly improve the performance of multi-layer LSTM. As the number of LSTM layers increases, the performance of multi-layer LSTM begins to decrease. In particular, compared to GLSTM-1, the HR@1 performance of GLSTM-5 is even reduced by 36.27%. On the contrary, with the help of the residual network, multi-layer LSTM has a substantial improvement. It is clear that the residual network can solve the gradient vanishing problem in the multi-layer LSTM well. Moreover, when the number of LSTM layers exceeds 4, the final results of GRLSTM have leveled off, which denotes that our model reaches its optimum at layer-4. There is a slight decrease in the performance of GRLSTM-5 compared to GRLSTM-4, indicating that our model appears to be slightly overfitted.

Conclusion

In this paper, we proposed a novel trajectory similarity computation framework named GRLSTM on road networks. In our model, we studied multi-relation of points on road networks and construct a point knowledge graph. We utilize the knowledge graph embedding method to learn entity (i.e, point) and relation embeddings. We further constructed the point fusion graph to integrate trajectory and road network information by learning embedding and k-nearest selection. To capture topology properties on the point fusion graph, we use multi-head GAT to learn graph-based point embeddings. Then, we designed a novel module named Residual-LSTM to learn the final trajectory embedding, where residual network in multi-layer LSTM can solve the gradient vanishing problem. Moreover, we design two new neighbor-based point-aware loss functions to optimize GRLSTM, including graph-based point loss function and trajectory-based point loss function. Extensive experiments show that our model GRLSTM significantly outperforms the state-of-the-art methods on two real-world datasets.

Acknowledgments

This work was supported by the NSFC (U2001212, 62032001, and 61932004) and Sichuan Science and Technology Program (2021YFS0007).

References

Chen, L.; and Ng, R. T. 2004. On The Marriage of Lp-norms and Edit Distance. In *VLDB*, 792–803.

Chen, L.; Özsu, M. T.; and Oria, V. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*, 491–502.

Chen, L.; Shang, S.; Feng, S.; and Kalnis, P. 2021. Parallel Subtrajectory Alignment over Massive-Scale Trajectory Data. In *IJCAI*, 3613–3619.

Chen, L.; Shang, S.; and Guo, T. 2020. Real-Time Route Search by Locations. In *AAAI*, 574–581.

Chen, L.; Shang, S.; Jensen, C. S.; Yao, B.; Zhang, Z.; and Shao, L. 2019. Effective and Efficient Reuse of Past Travel Behavior for Route Recommendation. In *KDD*, 488–498.

Chen, Z.; Shen, H. T.; Zhou, X.; Zheng, Y.; and Xie, X. 2010. Searching trajectories by locations: an efficiency study. In *SIGMOD*, 255–266.

Han, P.; Li, Z.; Liu, Y.; Zhao, P.; Li, J.; Wang, H.; and Shang, S. 2020. Contextualized Point-of-Interest Recommendation. In *IJCAI*, 2484–2490.

Han, P.; Shang, S.; Sun, A.; Zhao, P.; Zheng, K.; and Kalnis, P. 2019. AUC-MF: Point of Interest Recommendation with AUC Maximization. In *ICDE*, 1558–1561.

Han, P.; Wang, J.; Yao, D.; Shang, S.; and Zhang, X. 2021. A Graph-based Approach for Trajectory Similarity Computation in Spatial Networks. In *The 27th Conference on Knowledge Discovery and Data Mining, SIGKDD*, 556–564.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, 770–778.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Lee, J.; Han, J.; and Whang, K. 2007. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, 593–604.

Li, K.; Chen, L.; Shang, S.; Kalnis, P.; and Yao, B. 2021. Traffic Congestion Alleviation over Dynamic Road Networks: Continuous Optimal Route Combination for Trip Query Streams. In *IJCAI*, 3656–3662.

Li, X.; Cong, G.; and Cheng, Y. 2020. Spatial Transition Learning on Road Networks with Deep Probabilistic Models. In *ICDE*, 349–360.

Li, X.; Zhao, K.; Cong, G.; Jensen, C. S.; and Wei, W. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *ICDE*, 617–628.

Meng, F.; Yuan, G.; Lv, S.; Wang, Z.; and Xia, S. 2019. An overview on trajectory outlier detection. *Artif. Intell. Rev.*, 2437–2456.

Pei, W.; Tax, D. M. J.; and van der Maaten, L. 2016. Modeling Time Series Similarity with Siamese Recurrent Networks. *CoRR*.

Rakthanmanon, T.; Campana, B. J. L.; Mueen, A.; Batista, G. E. A. P. A.; Westover, M. B.; Zhu, Q.; Zakaria, J.; and Keogh, E. J. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*, 262–270.

Shang, S.; Chen, L.; Jensen, C. S.; Wen, J.; and Kalnis, P. 2017a. Searching Trajectories by Regions of Interest. *IEEE Trans. Knowl. Data Eng.*, 29(7): 1549–1562.

Shang, S.; Chen, L.; Wei, Z.; Jensen, C. S.; Zheng, K.; and Kalnis, P. 2017b. Trajectory Similarity Join in Spatial Networks. *Proc. VLDB Endow.*, 10(11): 1178–1189.

Shang, S.; Chen, L.; Wei, Z.; Jensen, C. S.; Zheng, K.; and Kalnis, P. 2018. Parallel trajectory similarity joins in spatial networks. *VLDB J.*, 27(3): 395–420.

Shang, S.; Chen, L.; Zheng, K.; Jensen, C. S.; Wei, Z.; and Kalnis, P. 2019. Parallel Trajectory-to-Location Join. *IEEE Trans. Knowl. Data Eng.*, 31: 1194–1207.

Shang, S.; Ding, R.; Yuan, B.; Xie, K.; Zheng, K.; and Kalnis, P. 2012. User oriented trajectory search for trip recommendation. In *EDBT*, 156–167.

Shang, S.; Ding, R.; Zheng, K.; Jensen, C. S.; Kalnis, P.; and Zhou, X. 2014. Personalized trajectory matching in spatial networks. *VLDB J.*, 23(3): 449–468.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *ICLR*.

Vlachos, M.; Gunopulos, D.; and Kollios, G. 2002. Discovering Similar Multidimensional Trajectories. In *ICDE*, 673–684.

Wang, S.; Bao, Z.; Culpepper, J. S.; and Cong, G. 2021. A Survey on Trajectory Data Management, Analytics, and Learning. *ACM Comput. Surv.*

Wang, S.; Bao, Z.; Culpepper, J. S.; Xie, Z.; Liu, Q.; and Qin, X. 2018. Torch: A Search Engine for Trajectory Data. In *SIGIR*, 535–544.

Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *AAAI*, 1112–1119.

Yang, C.; Chen, L.; Wang, H.; and Shang, S. 2021. Towards Efficient Selection of Activity Trajectories based on Diversity and Coverage. In *AAAI*, 689–696.

Yao, D.; Cong, G.; Zhang, C.; and Bi, J. 2019. Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach. In *ICDE*, 1358–1369.

Yao, D.; Cong, G.; Zhang, C.; Meng, X.; Duan, R.; and Bi, J. 2020. A Linear Time Approach to Computing Time Series Similarity based on Deep Metric Learning. *IEEE Transactions on Knowledge and Data Engineering*, 1–1.

Yao, D.; Zhang, C.; Zhu, Z.; Hu, Q.; Wang, Z.; Huang, J.; and Bi, J. 2018. Learning deep representation for trajectory clustering. *Expert Syst. J. Knowl. Eng.*

Yi, B.; Jagadish, H. V.; and Faloutsos, C. 1998. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *ICDE*, 201–208.

Zhang, H.; Zhang, X.; Jiang, Q.; Zheng, B.; Sun, Z.; Sun, W.; and Wang, C. 2020. Trajectory Similarity Learning with Auxiliary Supervision and Optimal Matching. In *IJCAI*, 3209–3215.

Zhao, K.; Zhang, Y.; Yin, H.; Wang, J.; Zheng, K.; Zhou, X.; and Xing, C. 2020. Discovering Subsequence Patterns for Next POI Recommendation. In *IJCAI*, 3216–3222.

Zheng, B.; Weng, L.; Zhao, X.; Zeng, K.; Zhou, X.; and Jensen, C. S. 2021. REPOSE: Distributed Top-k Trajectory Similarity Search with Local Reference Point Tries. In *ICDE*, 708–719.

Zheng, K.; Shang, S.; Yuan, N. J.; and Yang, Y. 2013. Towards efficient search for activity trajectories. In *ICDE*, 230–241.