Self-supervised contrastive representation learning for large-scale trajectories

Shuzhe Li a,1, Wei Chen a,1, Bingqi Yan a,1, Zhen Li a, Shunzhi Zhu b,∗, Yanwei Yu a,∗

a College of Computer Science and Technology, Ocean University of China, 1299 Sansha RD, Qingdao, 266404, Shandong, China
b Xiamen University of Technology, 600 Ligong RD, Xiamen, 361024, Fujian, China

A R T I C L E   I N F O
Article history:
Received 31 January 2023
Received in revised form 21 April 2023
Accepted 13 May 2023
Available online 19 June 2023

Keywords:
Representation learning
Large-scale trajectory
Contrastive learning
Trajectory augmentation

A B S T R A C T
Trajectory representation learning aims to embed trajectory sequences into fixed-length vector representations while preserving their original spatio-temporal feature proximity. Existing works either learn trajectory representations for specific mining tasks or fail to utilize large amounts of unlabeled trajectory data for representation learning. In this work, we propose a self-supervised Trajectory representation learning based on Reconstruction Contrastive Learning called TrajRCL. To be specific, TrajRCL first obtains low-distortion and high-fidelity views of trajectories through trajectory augmentation. Then, TrajRCL leverages a Transformer based encoder–decoder network to reconstruct low-distortion view trajectories to approximate high-fidelity trajectories. Self-supervised contrastive learning is finally used to enhance the consistency of the two view’s trajectory representations. Extensive experiments on two real-world datasets demonstrate the superiority of our model over state-of-the-art baselines and significant efficiency on similarity trajectory search and k-NN query.

1. Introduction

With the rise of mobile devices and the maturity of positioning technology, location-based applications have been widely used in people’s daily life. Various remote sensing satellites, electronic map navigation, and terminal equipment with GPS functions are collecting massive trajectory data [1], which includes not only mobility trajectories such as pedestrian travel and animal migration, but also the driving trajectories of vehicles [2]. The effective mining of such spatial–temporal data is the core foundation for various applications to provide intelligent services [3]. However, in reality, the spatial–temporal application scenarios are complex and changeable, and the collected spatial–temporal data become more complex due to factors such as multi-source, sampling frequency, accuracy, data missing, etc. The trajectory representation learning [4] aims to embed the original trajectory data from a variable-length coordinate-time stamp sequence into a fixed-length vector while maintaining the original spatial–temporal feature proximity, without manually designing various fixed trajectory measurement methods for various specific scenes. This is crucial for various downstream spatial–temporal data mining tasks, ranging from location recommendation (e.g., predicting tourists’ future visit preferences) [5,6], to traffic forecasting (e.g., traffic flow prediction) [7,8], and to public security (e.g., identifying abnormal trajectories) [9].

In recent years, representation learning technology has attracted widespread attention [10,11]. In spatial–temporal data mining, spatial–temporal representation learning has also been partially applied and studied. In literature, Li et al. [12] propose a deep trajectory representation method t2vec, which is similar to the traditional Seq2Seq model. The difference is that its decoder is to maximize the conditional probability of the input trajectory to its high sampling trajectory. Recently, Chen et al. [13] propose a trajectory-enhanced Transformer module, Toast, which uses trajectory data to extract driving semantics on the road network. In addition to obtaining effective road segment representation, this method can also obtain route representation. However, existing methods still have three key challenges to be solved: (1) Complexity: high complexity and large capacity of spatial–temporal data, often accompanied by sampling frequency uncertainty, data sparsity, and noise; (2) Learning paradigm: the trajectory data set with labels for specific tasks is very limited. How to effectively construct learning tasks in a self-supervised or unsupervised way to fully utilize the large-scale trajectory data set is challenging; (3) Applicability: how to design effective spatial–temporal representation learning model to obtain robust trajectory representation, and easily extend the model to various spatial–temporal data mining tasks, has not been extensively explored.

∗ Corresponding authors.
E-mail addresses: zhusz99@qq.com (S. Zhu), yuyanwei@ouc.edu.cn (Y. Yu).
1 Authors contributed equally to this work.
https://doi.org/10.1016/j.future.2023.05.033
0167-739X/© 2023 Elsevier B.V. All rights reserved.
To tackle these challenges, we propose TrajRCL, a self-supervised trajectory representation learning model based on the reconstruction contrastive learning framework, to solve the problem of trajectory representation learning. In TrajRCL, we use a trajectory adaptive transformer-based neural network architecture as the backbone. Specifically, three different trajectory perturbation policies are designed to alter the original trajectory to generate different view trajectories. Then, the Transformer encoder-decoder is used to reconstruct low-distortion view trajectories to high-fidelity view trajectories. For the three tasks of masking, reconstructing, and contrastive learning, three loss functions are designed to jointly train the model. Additionally, we also design a multi-scale spatial-aware embedding layer that uses Hilbert coding to generate the embeddings of spatial–temporal trajectories, which are fed into the backbone for representation learning. We conduct extensive experiments to evaluate the model performance on two real-world trajectory datasets. The experimental results show the superiority of our model over several strong baselines on various trajectory mining tasks.

Our key contributions can be summarized as follows:

- We propose a self-supervised trajectory representation learning framework, TrajRCL, to obtain the effective trajectory representation for various downstream tasks. Our TrajRCL combines data augmentation, trajectory reconstruction, and self-supervised contrastive learning which can effectively capture the spatiotemporal dependencies in trajectory data to obtain high-quality and robust trajectory representations.
- We design three loss functions for enhancing trajectory representation learning in TrajRCL to jointly train the model. Through the masking loss function, the encoder is forced to learn the semantic information of masked tokens. Through the reconstruction loss function, the decoder needs to reconstruct low-distortion trajectories to approximate high-fidelity trajectories. The trajectory representations learned by the encoder and decoder are forced to align via a contrastive loss function.
- Three downstream tasks are performed to evaluate the performance of the proposed model. The results on two real-world datasets demonstrate the effectiveness of the proposed model in learning trajectory representation in comparison to state-of-the-art baselines.

2. Related work

2.1. Representation learning

The goal of representation learning is to automatically embed the original data into the low-dimensional feature vectors, which can be effectively used as input feature information for various machine learning and deep learning models [11,14]. In the past few years, deep learning methods show their unique performance advantages in many fields, so deep representation learning algorithms receive extensive attention in various fields, including text processing [15], network analysis [16,17], recommendation [18,19], trajectory planning [20–23], etc. Le et al. [24] propose an unsupervised text representation learning method, which can learn fixed-length feature representations from variable-length text fragments. Perozzi et al. [25] propose DeepWalk for multi-label network classification, and learn the latent representation of vertices in the network. Song et al. [26] propose a new metric learning scheme based on structured prediction, which aims to optimize clustering quality by grasping the global structure of the embedding space. Guo et al. [27] propose a method for Streaming Session-based Recommendation that leverages matrix factorization-based attention model and reservoir-based streaming model for efficiency. Zhang et al. [18] propose a self-supervised hypergraph learning framework for group recommendation to capture the intra- and inter-group interactions among users and alleviate the data sparsity issue with the raw data itself. Xia et al. [19] develop a self-supervised graph co-training framework for session-based recommendation to enhance data augmentation with genuine self-supervision signals. However, existing representation learning work is difficult to be directly applied to trajectory data with complex spatiotemporal features.

2.2. Self-supervised representation learning

Contrastive learning, as a form of self-supervised learning, plays a critical role in representation learning. With the emphasis on deep learning, comparative learning makes great progress in the field of representation learning, and several important researches are produced [28,29]. Some recent studies show that the success of contrastive learning can be attributed to the maximization of mutual information [30]. More precisely, the widely used InfoNCE loss is a lower bound of the mutual information.

Existing studies [31–34] conduct contrastive learning at the instance level, use data augmentation to generate different views, and learn relevant data representations by comparing positive and negative samples. Chen et al. [31] propose a contrastive learning framework for visual representations and simplify contrastive self-supervised learning algorithms. Lin et al. [32] combine representation learning and data recovery into a unified framework from the perspective of information theory. He et al. [33] build a dynamic dictionary with a queue and a moving average encoder so that the learned representations can be efficiently transferred to downstream tasks. Self-supervised learning has obvious effects in addressing the effects of data sparsity and data noise in recommendation and trajectory analysis tasks. Sun et al. [35] propose a self-supervised hypergraph representation learning approach for sociological analysis to explore richer patterns under various sociological criteria. Jiang et al. [36] propose a self-supervised trajectory representation learning framework with temporal regularities and travel semantics, namely START, to convert raw trajectories into low-dimensional representation vectors by exploiting spatial–temporal characteristics such as temporal regularities and travel semantics.

2.3. Representation learning of trajectories

Trajectory representation learning can be considered as a special kind of representation learning for processing sequence data, which receive a lot of attention recently [11,23,37]. Therefore, it is natural to consider using an RNN-based encoder–decoder to learn representations of sequential data [23,37]. The traditional RNN-based encoder–decoder is designed for text data in natural language processing [38], where text documents have little noise and no time gap between words. To solve this problem, several models based on attention mechanism are proposed [39–42], and the time information is also encoded reasonably [43]. Trajectory data is not only a simple time series, but also contains complex spatial dependencies and road network dependencies. To incorporate road network information, some works extract road network graphs from real roads [44], and others encode spatial gridding trajectory sequences [44,45].

In recent years, trajectory representation learning has been applied to a variety of different trajectory mining tasks, such as trajectory prediction [46], location recommendation [34], traffic forecasting [7,47], outlier detection [48,49], Li et al. [50] develop a
multi-layer LSTM encoder–decoder model in which the temporal attention mechanism is used to enhance the sequence learning ability for human mobility representation. Capobianco et al. [51] leverage attention mechanism to enhance the recurrent network model, which is applied to vessel trajectory representation learning. CNN-based models are also used for mobility sequence representation and trajectory representation learning [52,53]. Recently, the self-attention model is used to replace RNN in trajectory sequence modeling [54]. Lin et al. [55] propose CTLE which is a pre-trained model and applies a Transformer encoder to calculate contextual embeddings for trajectory representation learning. In the follow-up work [56], they further propose a TALE pre-training method based on the CBOW framework, which is able to incorporate temporal information into the learned embedding vectors of locations.

A line of trajectory representation learning studies is proposed for trajectory similarity computation [17,57]. Li et al. [12] propose the first deep learning approach to learning representations of trajectory. Yang et al. [10] propose T3S to embed each trajectory into a vector in a d-dimensional space. Yao et al. [58] develop NEUTRAJ to collaborate with all spatial-based trajectory indexing methods to reduce the search space. Yang et al. [57] design a learning-based model to consider interactions between the trajectory pairs. However, most existing methods learn trajectory representation by approximating some traditional distance metrics as ground truth, while ignoring the exploration of new trajectory distance metrics based on the self-supervised paradigm of spatiotemporal features. Only a few studies have been carried out from this aspect, Liu et al. [59] propose a novel contrastive model to learn trajectory representations by distinguishing the trajectory-level and point-level differences between trajectories. Deng et al. [60] also learn the consistent representations of trajectories by applying trajectory data augmentations under the framework of contrastive learning. Compared with them, we have unified the three joint training objectives of contrast, denoising and reconstruction to achieve more generalized trajectory representation and apply it to downstream tasks.

3. Problem definition

In this section, we first introduce the common definitions and then present problem formulations.

**Definition 1 (Real Path).** The real moving path $\tau$ of a moving object is a continuous spatial curve in the longitude and latitude domain, representing the exact path of the object during its movement.

**Definition 2 (Trajectory).** A trajectory is the sample point $(a, b)$ sequence of the real path, where $a$ is longitude, $b$ is latitude, and $(a, b)$ is the trajectory point in geographic coordinates. Each trajectory can be expressed as $Tr = \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$, $n$ is the length of the trajectory.

Due to the limitation of recording equipment, we cannot obtain the real path of the moving object, but only the raw trajectory. When the sampling rate is high enough, it can be approximated as the real path.

Based on the above definitions, we define the studied problem of this work as follows:

**Problem 1 (Trajectory Representation Learning).** Given the large-scale trajectory dataset, our goal is to learn the low-dimensional vector representation $y \in \mathbb{R}^m$ ($m$ is the embedding dimension) for each trajectory $Tr$ such that the learned representation can reflect the real path of the trajectory, and thus be applied to various downstream trajectory data mining tasks.

4. Methodology

The overall architecture of the proposed TrajRCL is presented in Fig. 1. In particular, TrajRCL consists of three key components: trajectory augmentation module, Transformer-based sequence modeling module, and self-supervised contrastive learning. Trajectory augmentation module uses different data augmentation policies to obtain trajectory sequences in both low-distortion trajectory view and high-fidelity trajectory view. As the backbone of TrajRCL, the goal of Transformer-based sequence modeling module is to reconstruct the potential underlying path through the trajectory sequences, which is mainly divided into two parts: multi-scale spatial-aware embedding and Transformer encoder–decoder. Self-supervised contrastive learning module uses a weight-shared projection layer to obtain deeper trajectory embeddings, and introduces contrastive learning to constrain the trajectory representation learning between two different views to maximize the representation between different views consistency.

4.1. Trajectory augmentation

The real path $\tau$ of a moving object is a continuous spatial curve (e.g., in the latitude–longitude domain) representing the exact path taken by the object. In real-world data, a real path can be represented by different trajectories $Tr$, which is a sequence of sampling points of the underlying path of a moving object. Taking a trajectory $Tr = \{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ as an example, we create two trajectory views for it: low-distortion trajectory $Tr_{low}$ and high-fidelity trajectory $Tr_{high}$, respectively.

**4.1.1. Low-distortion trajectory view**

For the low-distortion trajectory view, we randomly adopt two strategies of downsampling and dynamic distortion to obtain its corresponding trajectory sequence:

- **Downsampling.** For each trajectory sequence with a given length $n$, we dynamically select 20%–60% of trajectory points for random masking. The remaining trajectory sequence is the low-distortion trajectory view for the original trajectory.
- **Dynamic Distortion.** For each trajectory sequence with a given length $n$, we dynamically select 20%–60% of the trajectory points for distorting, and the distorted trajectory sequence is the low-distortion trajectory view. Specifically, point $(a_i, b_i)$ is distorted by adding Gaussian noise with a radius of 50 m, as follows:

\[
\begin{align*}
    a_i &= a_i + 50 \cdot \epsilon, \quad \epsilon \sim \text{Gaussian}(0, 1) \\
    b_i &= b_i + 50 \cdot \epsilon, \quad \epsilon \sim \text{Gaussian}(0, 1)
\end{align*}
\]  

(1)

4.1.2. High-fidelity trajectory view

For the high-fidelity trajectory view, we randomly adopt two strategies of original preservation and linear interpolation to obtain the corresponding trajectory sequence:

- **Original Preservation.** Do nothing to the trajectory, and use its original trajectory sequence as a high-fidelity trajectory view.
- **Linear Interpolation.** For a trajectory sequence of given length $n$, we dynamically randomly select trajectory point pairs of 10%–20%, and use linear interpolation technology to add new points in the middle of point pairs. The new trajectory sequence serves as a high-fidelity trajectory view.
4.2. Transformer-based sequence modeling

Due to the natural sequential nature of trajectories, we choose Transformer encoder–decoder as our backbone network, with the goal of reconstructing the underlying path \( R \) from the trajectory sequence \( T_r \), maximizing the conditional probability \( P(R \mid T_r) \). However, due to the unavailability of the underlying path, we replace maximizing objective \( P(R \mid T_r) \) with maximizing objective \( P(T_r^{(high)} \mid T_r^{(low)}) \), and use an encoder–decoder module with a self-attention network as the backbone. Specifically, the module is mainly composed of two parts: multi-scale spatial-aware embedding module and Transformer encoder–decoder module.

4.2.1. Multi-scale spatial-aware embedding module

We first design a novel multi-scale spatial encoding method, using the Hilbert Curve to encode the latitude and longitude points in the two-dimensional space into binary forms in the corresponding real number domain. This encoding method well preserves the spatial structure of the original data, while increasing the density of multi-scale spatial information. Specifically, different view trajectories \( T_r \) are embedded into the low-dimensional space vector \( H_s \) through a linear layer:

\[
\begin{align*}
X_{low} &= \sigma(W_{low}H_{low} + b_{low}), \\
H_{high} &= \text{Hilbert}(T_{r^{low}}), \\
X_{high} &= \sigma(W_{high}H_{high} + b_{high}), \\
H_{high} &= \text{Hilbert}(T_{r^{high}}),
\end{align*}
\]

where \( H_s \) is the representation of trajectory \( T_r \) obtained through Hilbert curve transformation, \( \sigma \) is the Leaky ReLU activation function with a leaky rate of 0.2, \( W_{low} \) and \( b_{low} \) are the trainable parameters, and \( X_{low} \) and \( X_{high} \) are the low-dimensional embedding of low-distortion view and high-fidelity view trajectories obtained by multi-scale spatial-aware embedding module, respectively.

When encoding trajectory data using the Hilbert curve, each longitude–latitude point \((a, b)\) in a trajectory \( T_r \) is first converted to a two-dimensional index \((x, y)\) in a Cartesian coordinate system. Next, we use the Hilbert curve to map these points to a one-dimensional index space [61], where each point corresponds to a unique index value on the curve. The index values are represented using binary numbers, and concatenated to form a single vector as an alternative to coordinate points. After transformation, a vector sets \( H \) can be obtained. This process can be represented as:

\[
H = \text{Hilbert}(Tr) = [h_1, h_2, \ldots, h_n],
\]

where each \( h_i \) represents the index of the \( i \)-th longitude–latitude point on the Hilbert curve, and \( n \) represents the length of the trajectory. The Hilbert curve transformation enables the preservation of the spatial structure of trajectory data while increasing the density of multi-scale spatial information, thus realizing the coding and compression of trajectory data.

4.2.2. Transformer encoder–decoder module

As our work is to solve the problem of trajectory sequence representation, we employ Transformer network architecture as the backbone network, and its encoder embeds sequence representation \( H_{low} \) of the low-distortion view trajectory \( T_{r^{low}} \) by the bidirectional encoding, so that the embedding of each trajectory point through the encoder can effectively fuse the spatial information on the entire trajectory. Its decoder utilizes an autoregressive language model to reconstruct high-fidelity view trajectories from low-distortion trajectories.

For the Transformer encoder, we stack several transformer layers, each layer consisting of a causal masked multi-head self-attention module and a position-wise feed-forward network (FFN) module. Position-wise FFN will output a bag of embeddings, where the embedding at each position predicts the corresponding next point of the sequence. Residual connections and normalization have been applied to both modules.

For the \( i \)-th layer, the input \( X^{(i)} \in \mathbb{R}^{m \times d} \) is firstly transformed by the multi-head self-attention module. The output of the first attention head is:

\[
\text{head}^{d_1} = \text{softmax} \left( \frac{X^{(i)}W^{0}(X^{(i)}W^{0})^T}{\sqrt{d}} \right) X^{(i)}W^{r},
\]

where \( W^{0}, W^{r}, W^{h} \in \mathbb{R}^{d \times h} \) are the weight matrices corresponding to “Query”, “Key” and “Value”, respectively, \( X^{(i)} \) is the learned embedding matrix in the input low-distortion trajectory. Then, we stack multiple attention heads and merge the outputs from different attention heads by performing a linear transformation operation:

\[
\text{MultiHead} \left( X^{(i)} \right) = \left[ \text{head}^{d_1}; \text{head}^{d_2}; \cdots; \text{head}^{d_h} \right] \times W_s,
\]

where \( W_s \in \mathbb{R}^{d \times d} \) is the learnable parameter matrix.

Next, layer normalization and residual connection on attention module are performed to obtain the final output of the attention module:

\[
X^{(i)}_{\text{ATT}} = \text{LayerNorm} \left( X^{(i)} + \text{MultiHead} \left( X^{(i)} \right) \right),
\]

After the attention module, it will pass through a position-wise FFN:

\[
X_{\text{FFN}}^{(i)} = \max \left( 0, W_{\text{FFN}}X_{\text{ATT}} + b_1 \right) W_2 + b_2,
\]

where \( W_{\text{FFN}}, W_2 \) are trainable weight matrices, and \( b_1, b_2 \) are biases. Lastly, the output of the \( l \)-th encoder layer can be obtained:

\[
X^{(l+1)}_{\text{ATT}} = \text{LayerNorm} \left( X_{\text{ATT}}^{(l)} + X_{\text{FFN}}^{(l)} \right).
\]

The decoder uses an autoregressive language model to reconstruct high-fidelity view trajectories through low-distortion...
trajectories. At the reconstruction step of each trajectory, the new decoder query $W^{(2)}_{\text{Dec}}$ is compared with the encoded keys $W^k$ and values $W^v$ according to Eq. (3) to complete the reconstruction of high-fidelity view trajectories.

In this part, we design two loss functions, which are the masking distortion trajectory reconstruction loss of the bidirectional encoder $L_1$ and the overall decoding loss of the autoregressive decoder $L_2$. $L_1$ is used to capture the dynamic spatial proximity information of trajectory points, and $L_2$ is used to reconstruct the high-fidelity view trajectory. The two loss functions are formalized as follows:

$$L_1 = -\sum_{x \in M(x)} \log P(\hat{x} | \tilde{M}(x)), \quad (9)$$

$$L_2 = -\log \prod_{i} P(z_i | z_{1:i-1}, \tilde{M}(x)), \quad (10)$$

where $\tilde{M}(x)$ denotes the embedding of unmasked or undistorted trajectory points in the low-distortion view trajectory, $\hat{x} \in M(x)$ denotes the embedding of masked or distorted trajectory points in the low-distortion view trajectory, and $z_i$ is the embedding of reconstructed $i$th trajectory point in the corresponding high-fidelity view trajectory.

Next, we send the embeddings of the low-distortion view trajectory through the encoder and the embeddings of the high-fidelity view trajectory decoded by the decoder to the maximum pooling layer to obtain the corresponding overall trajectory representations $y^{(i)}$ and $z^{(i)}$ as follows:

$$y^{(i)} = \text{MaxPooling}(y_1, y_2, \ldots, y_n), \quad (11)$$

$$z^{(i)} = \text{MaxPooling}(z_1, z_2, \ldots, z_n). \quad (12)$$

4.3. Self-supervised contrastive learning

After obtaining trajectory representations of different views, we further feed them into a weight-shared projection layer to obtain their deep representations. Contrastive learning has demonstrated its superiority in various representation learning applications. Inspired by that, we introduce a self-supervised contrastive learning framework to enhance the model’s representation learning ability, maximizing the consistency of representations learned from different trajectory views.

We implement the operation of trajectory projection using two dense layers to obtain a deep representation:

$$y^{(i)} = W^{(2)} \sigma \left( W^{(1)} y^{(i)} + b^{(1)} \right) + b^{(2)},$$

$$z^{(i)} = W^{(2)} \sigma \left( W^{(1)} z^{(i)} + b^{(1)} \right) + b^{(2)}, \quad (13)$$

where $W^{(1)}$ is the learnable weight vector and $b^{(*)}$ is the bias. $\sigma$ is a leaky ReLU activation function with a leaky rate of 0.2.

In self-supervised contrastive learning, the deep representation pairs under different views of the same trajectory are regarded as positive sample pairs, while other samples under the same batch are regarded as negative samples. The goal of optimization is to make the different views (i.e., low-distortion trajectory view and high-fidelity trajectory view) of the same trajectory sample as consistent as possible in the representation space and at the same time be as far away as possible from other negative samples in the same batch. The loss function $L_3$ is formalized as follows:

$$L_3 = -\log \frac{e^{\text{sim}(y^{(i)}, z^{(i)}) / \tau}}{\sum_{j=1}^{N} e^{\text{sim}(y^{(i)}, z^{(j)}) / \tau}}, \quad (14)$$

where $\tau$ is the temperature parameter, $\text{sim}(\cdot, \cdot)$ is the cosine similarity function, and $(y^{(i)}, z^{(j)})$ represents a pair of negative samples.

4.4. Model learning

The overall loss $L_{\text{model}}$ of the model is obtained by combining the trajectory reconstruction loss $L_1$, the overall decoding loss $L_2$, with the contrastive learning loss $L_3$. More specifically, we optimize our model by maximizing the following objective function:

$$L_{\text{model}} = \beta (L_1 + \alpha L_2) + L_3. \quad (15)$$

where $\alpha$ and $\beta$ are used to balance the importance of reconstruction loss, decoding loss, and contrastive learning loss.

5. Experiment

To verify the effectiveness of our proposed model, we conduct extensive experiments on two public real-world trajectory datasets.

5.1. Datasets

We evaluate TrajRCL on two public real-world trajectory datasets in two cities, i.e., Porto taxi trajectory dataset and T-Drive trajectory dataset. The Porto dataset comes from the ECML-PKDD competition and contains more than 1.7 million complete trajectories collected from 442 taxis operating in the city of Porto from July 1, 2013 to June 30, 2014. Its sampling frequency is once every 15 s. The T-Drive dataset is a sample of the T-Drive trajectory dataset, which contains the weekly trajectories of 10,357 taxis from February 2 to February 8, 2008, and the total number of trajectory points is about 17 million.

5.2. Baseline methods

To evaluate the performance of our TrajRCL, we study the most similarity search problem and k-NN query problem on the Porto dataset, and compare TrajRCL with six classical trajectory distance measures: the Hausdorff distance [62], the Fréchet distance [63], Dynamic Time Warping (DTW) [64], Longest Common Subsequence (LCSS) [65], Edit distance with Real Plenty (ERP) [66], Edit Distance on Real sequence (EDR) [67] and the latest Contrastive Learning based Trajectory Similarity Computation (CL-TSim) [60] method.

We also evaluate the effectiveness of our TrajRCL for the trajectory prediction task on both Porto and T-Drive datasets, and compare our model with the following baselines:

- ST-LSTM [68]: This method is a long- and short-term trajectory prediction model considering spatial trajectory relationship.
- STAN [54]: This method explicitly exploits relative spatiotemporal information of all the points with self-attention layers along the trajectory.
- CTLE [55]: This is a Context and Time aware Location Embedding (CTLE) model, which calculates a location’s representation vector with consideration of its specific contextual neighbors in trajectories.
- TALE [56]: This is a Time-Aware Location Embedding (TALE) pre-training method based on the CBOW framework, which is able to incorporate temporal information into the learned embedding vectors of locations.
- Graph-Flashback (G-Fback) [34]: This is a state-of-the-art graph-based model with strong location representation ability.
- PreCLN [46]: This is a pretrained-based contrastive learning network for vehicle trajectory prediction.
5.3. Experimental settings

One of the most important tasks in trajectory analysis is similar trajectory search. Following [12], we design two experiments (i.e., most similar trajectory search and k-nearest-neighbors (k-NN) query) to evaluate the accuracy of methods for computing trajectory similarity using self-similarity and cross-similarity comparisons. Specifically, we randomly select 1,000 trajectories from the test dataset, denoted as Q. Then, we select another n trajectories, denoted as P. For each trajectory Tr \in Q, we generate two sub-trajectories from it by the odd-even sampling of trajectory points, denoted as Trs and Trd, and use them to construct two datasets D0 = {Trs} and Dd = {Trd}. We perform the same operation for the trajectories in P to get Dp and Dp.

For each trajectory Tr \in D0, we compute its top-k most similar trajectories from database D0 \cup Dp and calculate the rank of Tr in both D0 and Dp. Ideally, Trk is ranked at the top since it is produced from the same original trajectory as Trk.

Moreover, we further validate TrajRCL using the trajectory prediction task. Following [46], for each trajectory Tr = ((t1, lat1, lon1), (t2, lat2, lon2), ..., (tn, latn, lonn)), we encode the latitude lati and longitude loni, and then generate the trajectory location coding sequence to predict the future trajectory location sequence. Given all trajectories of all vehicles TR and the road network G, our goal is to predict the future vehicle trajectory of the next \Delta time steps for any given vehicle.

5.4. Performance evaluation for most similar trajectory search

We first evaluate the performance of our TrajRCL for the most similar trajectory search task compared to six trajectory distance measurement methods by increasing the number of trajectories from 5k to 60k. We report the mean rank of 1k queries in D0 and search time on the 60k dataset on Porto dataset in Table 1.

<table>
<thead>
<tr>
<th>Method</th>
<th>5k</th>
<th>10k</th>
<th>20k</th>
<th>30k</th>
<th>40k</th>
<th>50k</th>
<th>60k</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hausdorff</td>
<td>2.14</td>
<td>3.1</td>
<td>5.26</td>
<td>7.47</td>
<td>9.74</td>
<td>11.94</td>
<td>14.18</td>
<td>17886s</td>
</tr>
<tr>
<td>Frechet</td>
<td>2.11</td>
<td>2.99</td>
<td>4.91</td>
<td>6.96</td>
<td>9.06</td>
<td>10.88</td>
<td>12.79</td>
<td>34280s</td>
</tr>
<tr>
<td>DTW</td>
<td>3.32</td>
<td>1.78</td>
<td>2.12</td>
<td>2.63</td>
<td>3.21</td>
<td>3.78</td>
<td>4.32</td>
<td>18420s</td>
</tr>
<tr>
<td>LCSS</td>
<td>25.31</td>
<td>28.45</td>
<td>40.77</td>
<td>39.43</td>
<td>42.09</td>
<td>50.85</td>
<td>48.81</td>
<td>21060s</td>
</tr>
<tr>
<td>ERP</td>
<td>41.41</td>
<td>78.59</td>
<td>152.62</td>
<td>229.07</td>
<td>307.07</td>
<td>383.06</td>
<td>460.09</td>
<td>22800s</td>
</tr>
<tr>
<td>EdR</td>
<td>16.00</td>
<td>29.92</td>
<td>58.29</td>
<td>83.31</td>
<td>105.14</td>
<td>139.23</td>
<td>169.53</td>
<td>20340s</td>
</tr>
<tr>
<td>CL-TSim</td>
<td>1.21</td>
<td>1.97</td>
<td>2.74</td>
<td>2.86</td>
<td>3.05</td>
<td>3.47</td>
<td>4.89</td>
<td>1764s</td>
</tr>
<tr>
<td>TrajRCL</td>
<td>1.42</td>
<td>1.61</td>
<td>2.47</td>
<td>2.58</td>
<td>4.20</td>
<td>6.85</td>
<td>7.90</td>
<td>68s</td>
</tr>
</tbody>
</table>

5.5. Performance evaluation for k-NN query

We next evaluate the performance of different methods on Porto dataset for the kNN query task. In particular, we randomly select 10,000 trajectories from the test set as the target database, and 1,000 trajectories as the query database. We query the k-nearest neighbors for each trajectory from the target database as its ground truth. Then we transform the trajectories in both the query and target databases by randomly dropping or distorting points at a certain rate. Next, for each transformed query, we use each method to find its k-NNs from the target database, and then compare the result with the corresponding ground truth. Table 2, 3, and 4 show the Precision results of all methods w.r.t. different dropping rates and distorting rates. The corresponding average query time for k-NN queries is also shown in the last column in each table.

From the results in the three tables, we have the following three findings: (1) As the dropping/distorting rate increases, the performance of all methods decreases. More specifically, dropping has a greater impact on model performance than distorting on the same scale. It is obvious that discarding trajectory points is more likely to change the spatial characteristics of the trajectory. (2) As k increases, the performance of our TrajRCL gets better and better. For example, when k = 20, our model achieves the best performance in most cases, but when k = 40 our model achieves the best performance in all cases. This also demonstrates that our model can learn the representations of similar trajectories more closely. (3) The query efficiency of our TrajRCL is far superior to other measurement methods. Specifically, our model is up to 78 times more efficient than the Fréchet measurement. Compared with the state-of-the-art CL-TSim for 40-NN query, the efficiency is improved by 8.2 times.

5.6. Performance evaluation for trajectory prediction

To further evaluate the trajectory representation ability of the model, we introduce a trajectory prediction task. Specifically, we use TrajRCL to learn trajectory representations, i.e., y and z, and then concatenate them to obtain the final trajectory representation. Then we import the trajectory representation into an MLP trajectory decoder to predict future trajectory sequences. We adopt the MSE, RMSE, and their standard deviation as evaluation metrics to verify the performance of the proposed model.

Table 5 shows the comparison results of TrajRCL with the baseline models, where the best values are shown in bold. From all the results, we can see that TrajRCL achieves the best trajectory prediction performance, which demonstrates that our model has a strong learning ability to represent trajectories. In particular, the relative performance improvement of our TrajRCL over the best-performed baseline PreCLN is 6.1% and 8.3% in terms of MAE and RMSE on T-Drive dataset. TrajRCL also significantly outperforms the graph-based baseline Graph-Flashback by an average of 14.91% and 52.36% improvements in terms of MAE and RMSE on two datasets, respectively. Although PreCLN model considers the contrastive learning framework and three pre-training tasks,
Ablation study

To validate the effectiveness of each component in **TrajRCL**, we further conduct the ablation study. We compare our **TrajRCL** with four carefully designed variants.

---

**Table 2**

Overall performance comparisons for 20-NN queries on Porto dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>k = 20, dropping rate</th>
<th>k = 20, distorting rate</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2 0.3 0.4 0.5 0.6</td>
<td>0.2 0.3 0.4 0.5 0.6</td>
<td></td>
</tr>
<tr>
<td>Hausdorff</td>
<td>0.5043 0.4021 0.2230 0.1542 0.1593</td>
<td>0.3712 0.2215 0.1342 0.1215 0.0852</td>
<td>12.18 s</td>
</tr>
<tr>
<td>Fréchet</td>
<td>0.5523 0.3673 0.2815 0.1725 0.1730</td>
<td>0.4125 0.2921 0.1354 0.1230 0.0850</td>
<td>68.19 s</td>
</tr>
<tr>
<td>DTW</td>
<td>0.4079 0.0300 0.0250 0.0279 0.0270</td>
<td>0.2629 0.0216 0.0178 0.0147 0.0143</td>
<td>19.76 s</td>
</tr>
<tr>
<td>LCS</td>
<td>0.3720 0.3521 0.2315 0.1763 0.1763</td>
<td>0.3140 0.4521 0.4012 0.3536 0.3573</td>
<td>20.96 s</td>
</tr>
<tr>
<td>ERP</td>
<td>0.0325 0.0230 0.0190 0.0145 0.0145</td>
<td>0.8170 0.7700 0.7590 0.7035 0.6985</td>
<td>40.64 s</td>
</tr>
<tr>
<td>ED</td>
<td>0.0130 0.0060 0.0040 0.0075 0.0075</td>
<td>0.0995 0.0790 0.0640 0.0565 0.0570</td>
<td>20.32 s</td>
</tr>
<tr>
<td>CL-TSim</td>
<td>0.6524 0.6243 0.5874 0.5079 0.5441</td>
<td>0.7164 0.6948 0.6793 0.6185 0.6523</td>
<td>7.23 s</td>
</tr>
<tr>
<td><strong>TrajRCL</strong></td>
<td><strong>0.7820</strong> 0.7048 0.6518 0.6045 0.6027</td>
<td><strong>0.8002</strong> 0.7763 0.7428 0.7137 0.7125</td>
<td><strong>8.73 s</strong></td>
</tr>
</tbody>
</table>

**Table 3**

Overall performance comparisons for 30-NN queries on Porto dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>k = 30, dropping rate</th>
<th>k = 30, distorting rate</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2 0.3 0.4 0.5 0.6</td>
<td>0.2 0.3 0.4 0.5 0.6</td>
<td></td>
</tr>
<tr>
<td>Hausdorff</td>
<td>0.4815 0.3819 0.2155 0.1342 0.1334</td>
<td>0.3568 0.2158 0.1275 0.1024 0.0880</td>
<td>12.46 s</td>
</tr>
<tr>
<td>Fréchet</td>
<td>0.5368 0.3564 0.2743 0.1656 0.1638</td>
<td>0.4052 0.2825 0.1346 0.1145 0.0879</td>
<td>69.75 s</td>
</tr>
<tr>
<td>DTW</td>
<td>0.0287 0.0233 0.0183 0.0197 0.0197</td>
<td>0.2467 0.1930 0.1533 0.1247 0.1217</td>
<td>20.21 s</td>
</tr>
<tr>
<td>LCS</td>
<td>0.3964 0.3820 0.2640 0.2015 0.1995</td>
<td>0.5246 0.4785 0.4314 0.3840 0.3821</td>
<td>21.44 s</td>
</tr>
<tr>
<td>ERP</td>
<td>0.0233 0.0167 0.0147 0.0127 0.0127</td>
<td><strong>0.7883</strong> 0.7577 0.7373 0.6830 0.6907</td>
<td>41.57 s</td>
</tr>
<tr>
<td>ED</td>
<td>0.0127 0.0067 0.0047 0.0057 0.0057</td>
<td>0.0873 0.0677 0.0523 0.0453 0.0450</td>
<td>20.76 s</td>
</tr>
<tr>
<td>CL-TSim</td>
<td>0.6520 0.6345 0.5846 0.5077 0.5312</td>
<td>0.7125 0.6912 0.6702 0.6117 0.6433</td>
<td>7.64 s</td>
</tr>
<tr>
<td><strong>TrajRCL</strong></td>
<td><strong>0.7762</strong> 0.7021 0.6558 0.6017 0.6005</td>
<td><strong>0.7968</strong> 0.7740 0.7385 0.7129 0.7014</td>
<td><strong>8.89 s</strong></td>
</tr>
</tbody>
</table>

**Table 4**

Overall performance comparisons for 40-NN queries on Porto dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>k = 40, dropping rate</th>
<th>k = 40, distorting rate</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2 0.3 0.4 0.5 0.6</td>
<td>0.2 0.3 0.4 0.5 0.6</td>
<td></td>
</tr>
<tr>
<td>Hausdorff</td>
<td>0.4835 0.3840 0.2214 0.1387 0.1380</td>
<td>0.3611 0.2248 0.1243 0.1079 0.1045</td>
<td>9.87 s</td>
</tr>
<tr>
<td>Fréchet</td>
<td>0.5413 0.3613 0.2790 0.1842 0.1830</td>
<td>0.4005 0.3029 0.1394 0.1144 0.1131</td>
<td>55.30 s</td>
</tr>
<tr>
<td>DTW</td>
<td>0.0237 0.0207 0.0183 0.0175 0.0175</td>
<td>0.2430 0.1983 0.1535 0.1233 0.1188</td>
<td>16.02 s</td>
</tr>
<tr>
<td>LCS</td>
<td>0.4453 0.4230 0.3045 0.2482 0.2475</td>
<td>0.5573 0.5140 0.4720 0.4280 0.4275</td>
<td>17.00 s</td>
</tr>
<tr>
<td>ERP</td>
<td>0.0212 0.0148 0.0143 0.0115 0.0115</td>
<td>0.8022 0.7593 0.7388 0.6805 0.6882</td>
<td>32.96 s</td>
</tr>
<tr>
<td>ED</td>
<td>0.0133 0.0070 0.0060 0.0063 0.0063</td>
<td>0.0825 0.0623 0.0513 0.0423 0.0417</td>
<td>16.46 s</td>
</tr>
<tr>
<td>CL-TSim</td>
<td>0.7142 0.6578 0.5050 0.5480 0.5744</td>
<td>0.7245 0.7102 0.6980 0.6443 0.6832</td>
<td>6.89 s</td>
</tr>
<tr>
<td><strong>TrajRCL</strong></td>
<td><strong>0.7963</strong> 0.7228 0.6745 0.6023 0.6220</td>
<td><strong>0.8034</strong> 0.7903 0.7542 0.7200 0.7122</td>
<td><strong>8.84 s</strong></td>
</tr>
</tbody>
</table>

**Table 5**

Overall performance comparisons for trajectory prediction.

<table>
<thead>
<tr>
<th>Method</th>
<th>Porto</th>
<th>T-Driver</th>
<th>MAE Std</th>
<th>RMSE Std</th>
<th>MAE Std</th>
<th>RMSE Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST-LSTM</td>
<td>876.59</td>
<td>11.39</td>
<td>1395.26</td>
<td>238.19</td>
<td>99.31</td>
<td>189</td>
</tr>
<tr>
<td>STAN</td>
<td>613.24</td>
<td>8.36</td>
<td>2578.12</td>
<td>121.83</td>
<td>84.39</td>
<td>3.42</td>
</tr>
<tr>
<td>TALE</td>
<td>598.95</td>
<td>9.31</td>
<td>2385.60</td>
<td>85.61</td>
<td>71.23</td>
<td>2.91</td>
</tr>
<tr>
<td>G-Back</td>
<td>211.44</td>
<td>5.79</td>
<td>1636.35</td>
<td>32.43</td>
<td>68.38</td>
<td>2.73</td>
</tr>
<tr>
<td>CL-TSim</td>
<td>198.77</td>
<td>7.11</td>
<td>464.29</td>
<td>22.10</td>
<td>59.33</td>
<td>2.69</td>
</tr>
<tr>
<td><strong>TrajRCL</strong></td>
<td><strong>184.71</strong></td>
<td>3.17</td>
<td><strong>412.79</strong></td>
<td>21.04</td>
<td><strong>62.67</strong></td>
<td><strong>1.82</strong></td>
</tr>
</tbody>
</table>
w/o CL variant shows the worst performance, which also shows that the contrastive learning module designed in this paper plays a major role in improving the model performance. Followed by the w/o MS variant, which also verifies the rationality of our proposed multi-scale spatial-aware embedded module. In addition, this experiment also shows that the joint use of multiple different trajectory enhancement strategies further improves the model performance.

5.8. Sensitivity study

We finally investigate the sensitivity of our TrajRCL with respect to the important parameters, including hyper-parameters $\alpha$ and $\beta$ to balance the importance of reconstruction loss, decoding loss, and contrastive learning loss. We conduct sensitivity experiments on the KNN query. We select $k = 20$ and $k = 40$ and the dropping rate is 0.4 to evaluate the performance of our model. Results on Porto dataset are shown in Fig. 3. We can observe that: (1) When $\beta = 0$ (i.e., ignoring $\mathcal{L}_2$ and $\mathcal{L}_3$), the performance is the worst, this emphasizes the importance of the reconstruction loss and decoding loss. (2) When $\alpha$ increases from 0.01, the performance of the model is significantly improved, which verifies the effectiveness of the decoding loss. When $\alpha = 0.3$, the performance of the model is the best, indicating that the reconstruction loss and decoding loss have reached a better balance.

6. Conclusion

In this paper, we present a self-supervised framework, TrajRCL, for learning effective trajectory representation. It combines data augmentation, reconstruction, and contrastive learning to capture dependencies in trajectories to obtain high-quality and robust trajectory representations. TrajRCL is then jointly trained with three designed loss functions to enhance trajectory representation learning. The performance of our proposed model is evaluated through three trajectory analytical tasks on two real-world datasets and results show the superiority of our model compared to state-of-the-art baselines. In future work, we plan to investigate efficient pre-trained representation learning techniques for large-scale multi-source multi-scale trajectory data.

CRediT authorship contribution statement

Shuzhe Li: Methodology, Software, Data curation, Validation, Visualization, Writing – original draft. Wei Chen: Methodology, Software, Validation, Formal analysis, Writing – review & editing. Bingqi Yan: Methodology, Software, Visualization, Writing – review & editing. Zhen Li: Validation, Writing – original draft. Shunzhi Zhu: Writing – review & editing. Yanwei Yu: Conceptualization, Supervision, Writing – review & editing. Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The used data is publicly available.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under grant Nos. 62176243 and 61773331.

References


Shuzhe Li graduated from Shandong University of Science and Technology in 2021 with a bachelor degree in network engineering. He is currently a master student at the College of Computer Science and Technology, Ocean University of China. His research interests include large-scale spatio-temporal data mining and deep learning.

Wei Chen received the bachelor degree in computer science from the Jinggangshan University, China, in 2020. He is currently a master student with the College of Computer Science and Technology, Ocean University of China. His research interests include data mining, information retrieval and causal learning, especially in spatio-temporal field.

Bingqi Yan received his bachelor degrees from the Yantai University, China, in 2020. He is currently pursuing the master degree with the College of Computer Science and Technology, Ocean University of China. His research interests include multivariate sequence data mining and spatio-temporal trajectory prediction.

Zhen Li received the bachelor degree in computer science from the Ocean University of China, China, in 2020. He is currently a master student with the College of Computer Science and Technology, Ocean University of China. His research interests include spatio-temporal data mining and machine learning.

Shunzhi Zhu received the Ph.D. degree from Xiamen University, in 2007. He was a visiting professor with Florida International University, USA. He is currently a full professor with the School of Computer and Information Engineering, Xiamen University of Technology. His research interests include big data management and analytics, data mining, and information retrieval.

Yanwei Yu received the Ph.D. degree in computer science from the University of Science and Technology Beijing, China, in 2014. From 2016 to 2018, he was a postdoctoral researcher with the College of Information Sciences and Technology, Pennsylvania State University. He is currently a professor with the College of Computer Science and Technology, Ocean University of China. His research interests include data mining, machine learning, and database systems.