TMN: Trajectory Matching Networks for Predicting Similarity

Peilun Yang[1,2], Hanchen Wang[3,*], Defu Lian[4], Ying Zhang[1,2], Lu Qin[2], and Wenjie Zhang[3]
[1]*Zhejiang Gongshang University, China,* [2]*University of Technology Sydney, Australia,*
[3]*University of New South Wales, Australia,*
[4]*University of Science and Technology of China, China*
[2]peilun.yang@student.uts.edu.au; {ying.zhang, lu.qin}@uts.edu.au;
[3]{hanchen.wang,wenjie.zhang}@unsw.edu.au;
[4]liandefu@ustc.edu.cn;

2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00173

***Abstract*—Trajectory similarity computation is the cornerstone of many applications in the field of trajectory data analysis. To cope with the high time complexity of calculating exact similarity between trajectories, learning-based models have been developed for a good trade-off between the similarity computing time and the accuracy of the learned similarity. As each trajectory can be represented by a fixed-length vector regardless of the size of the trajectory, the similarity computation among the trajectories is highly time-efficient. Nevertheless, we observe that these learning-based models are designed based on recurrent neural networks (RNN), which cannot properly capture the correlations among the trajectories. Moreover, these learning-based models simply use the similarity scores of the pairs of trajectories in the training for a specific similarity metric, while a vital piece of information is neglected: the mappings of the points between two trajectories are readily available when the similarity score is calculated.**

These motivate us to design a new learning-based model, named TMN, based on attention networks, aiming to significantly improve the accuracy such that a better trade-off between the similarity computing time and the accuracy can be achieved. The proposed matching mechanism associates points across trajectories by computing attention weights of point pairs so that TMN learns to simulate similarity computation between the trajectory pair. Apart from taking interactions between trajectories into consideration, the sequential information of each individual trajectory is also considered, thereby making full use of spatial features of a pair of trajectories. We evaluate various approaches on real-life datasets under extensive trajectory distance metrics. Experimental results demonstrate that TMN outperforms state-of-the-art methods in terms of accuracy. Besides, ablation studies prove the effectiveness of our novel matching mechanism.

I. Introduction

As location acquisition technologies have gradually matured, a large amount of trajectory data have appeared that records the traces of various kinds of objects. Further, a broad range of applications have emerged that are driven by trajectory data mining [1], [2], such as tracking the migration patterns of animals [3], predicting traffic conditions [4], and so on. Basically, measuring the similarity between trajectories is a fundamental procedure in application scenarios like detecting anomaly trajectory [5], [6] and trajectory clustering [7], [8]. For the purpose of evaluating the trajectory similarity, many distance metrics have been devised in the literature, such as the Hausdorff distance [9], the Féchet distance [10], Dynamic Time Warping (DTW) [11], Edit Distance on Real sequence (EDR) [12], Edit distance with Real Penalty (ERP) [13], Longest Common Subsequence (LCSS) [14] and etc.

Approximate trajectory similarity computation. These distance metrics have generally demonstrated good performance in a variety of trajectory data mining tasks, and it is important to note that they each reflect and capture a different perspective on the similarity of two trajectories. Given the fact that: (1) trajectory similarity computation is fundamental and frequently used in many applications; (2) the exact computation of these distance metrics is quite time consuming; and (3) a reasonable approximation is acceptable in many applications, many research efforts have been devoted to approximate the trajectory similarity computation, which can be roughly classified into the following two categories:

(1) Non-learning-based methods (e.g., [15]–[17]) usually involve indexing and pruning strategy [18], [19]. Alternatively, they rely on an approximation-based algorithm [15], [17] that speeds up the distance metric computation. However, these methods are usually designed for one or two specific trajectory distance metrics, thus cannot be used with other metrics. In addition, experiments of previous work [20] have shown that most of these methods do not have very good approximation accuracy when being used to calculate trajectory similarity.

(2) Learning-based methods have been proposed which leverage the machine learning techniques to learn proper representations of the trajectories for any one of distance metrics. Conventional distance metrics often cost quadratic time to compute exact distance between trajectories (e.g., DTW, ERP takes $O(n^2)$ time where n is the length of trajectory), so many applications need approximation to accelerate computation. The most attractive advantage of the learning-based methods is the significant speed up of the trajectory similarity computation time. Generally speaking, after the preprocessing phase, i.e., the training of these learning-based models and the embedding of the trajectories, each trajectory in the database is converted into a vector (i.e., point) in a d-dimensional space. Then the similarity between two trajectories can be approximated by the distance (e.g., Euclidean distance) between two corresponding vectors, which only takes $O(d)$ time. By doing

* Hanchen Wang is the corresponding author

2375-026X/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDE53745.2022.00173

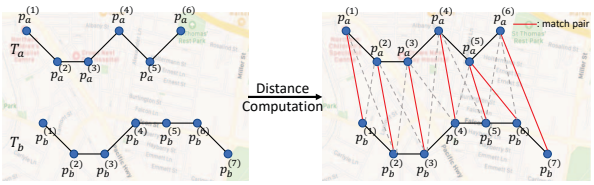

Fig. 1. Point match pairs between two trajectories that are generated during DTW distance computation.

this, computing the similarities between trajectories becomes highly time-efficient. For example, as reported in Section V, this can achieve around 6 orders of magnitude speed up for DTW distance computation. Moreover, for similarity-based operations, existing multi-dimensional indexing techniques can be immediately used to further speed up the computation. For instance, the state-of-the-art indexing techniques (e.g., HNSW [21]) can be immediately applied to the vectors of the embedded trajectories for nearest neighbor search. In addition to the speed up, the learning-based methods are generic to different distance metrics since their models can be immediately applied to a new distance metric without modifying the architectures. Plus, the embedded trajectories can be used for other downstream tasks or applications. For example, an e-commerce company might use the similarities between user-generated shopping trajectories to develop a better advertising strategy. These advantages coupled with the requirements of today's applications motivate us to further study the trajectory similarity learning and prediction.

Bottleneck of the existing learning-based methods. Because learning-based models embed the trajectories as multi-dimensional points, there is no difference in the efficiency with which these models compute distances. Thus, the key to developing a good learning-based model is the approximation accuracy of the similarity computation. We observe that all existing learning-based methods are designed based on recurrent neural networks (RNNs), which is a popular approach to learning the representations for sequences. Under this framework, a representation of each trajectory is learned almost independently from the other trajectories during training. Thus, existing learning-based methods primarily focus on the intra-information of each individual trajectory but neglect the inter-information, i.e., the interactions/correlations between trajectories. These interactions/correlations refer to the cross-trajectory information, specifically, the point matching information. For instance, in Figure 1, both the red and gray lines represent the point matching between trajectories. This reflects the vertex correspondences between trajectories during similarity computation. The lack of correlations between trajectories leads to the limited performance of these learning-based models in terms of approximation accuracy.

Motivation. We observe that popular trajectory distance metrics first find a match of the points between two trajectories, and then accumulate the information of these matched pairs to obtain a trajectory similarity score. For instance, Figure 1 shows the point match pairs when calculating the DTW distance between two trajectories. The solid red lines represent the matched point pairs during the DTW distance computation between T_a and T_b, while the dashed gray lines are the possible point pairs that could be considered in the calculation. The final distance is derived by summing up the distances between the matched pairs. This procedure indicates that the trajectory distance computation is heavily reliant on the process of point match between a pair of trajectories. However, previous learning-based methods all overlook this important information, i.e., the mapping of the points between trajectories. Thus, these models cannot adaptively capture the correlation between two trajectories for similarity computation during the training, bounding the approximation accuracy from the above. In the field of machine learning, a model can naturally approximate the match relationship by means of attention techniques. Attention mechanisms mimic cognitive attention by looking for the important parts of the input data and fading out the rest. Thus, an attention mechanism is likely to capture a matched point for every point during the calculation of trajectory similarity. *This motivates us to use the attention network as the backbone of our new model for better approximation accuracy*. Notably, we are not the only trajectory similarity computation framework to incorporate an attention mechanism. There are some existing learning-based models for this purpose that also rely on attention mechanism (e.g., [20], [22]). However, their models mainly depends on RNNs, and the attention mechanism is simply used for points *within* one trajectory or historically processed trajectories which cannot properly capture the correlations among trajectories.

In this paper, we propose a novel matching-based model called TMN, which stands for Trajectory Matching Networks for learning to approximate trajectory similarity. TMN takes advantage of a matching mechanism, whose objective is to match the points in one trajectory to the points in the other. The matching mechanism is essentially dependent on attention mechanism, which is able to compute the similarities between points such that the points can be matched across trajectories. The matching information is then combined with the spatial information from the trajectories and sent to an RNN. RNNs are good at handling sequence data, and they output low-dimensional vectors at each time step. Hence, we exploit RNN in TMN to learn the trajectory representations. The matching information and the spatial information are fed into the RNN model so that the learned representations contain both pieces of information. In addition, inspired by deep metric learning and regression-oriented loss functions, we propose the framework to further enhance the adaptability of TMN on learning trajectory similarity under popular distance metrics.

Our principle contributions can be summarized as follows:

- We propose an effective, learning-based model TMN for trajectory similarity learning task. As to our best knowledge, this is the first attempt to build the trajectory similarity learning model based on the attention networks,

which significantly improves the accuracy compared to the existing learning-based models built on recurrent neural networks.
- We propose a matching mechanism which computes similarity scores through a cross-trajectory attention mechanism. This mechanism associates points across trajectory pairs, which exploits the trajectory matching information for representation learning.
- We conduct extensive experiments on real-life datasets to evaluate TMN and previous state-of-the-art methods. Experimental results demonstrate the superiority of TMN and the effectiveness of our matching mechanism.

II. Related Work

In this section, we begin by reviewing the distance metric learning for similarity computation. Then we introduce relevant work on RNNs and attention-based networks. The section concludes with a summary and analysis of the current learning-based methods for computing trajectory similarities.

A. Metric Learning for Similarity Computation

Similarity learning for complicated objects, such as sets [23], strings [24], trajectories [25], [26], graphs [27], has been extensively studied. These works generally rely on metric learning, which aims to learn the (dis)similarity of target objects by learning a distance metric between the data points. Usually, the goal of metric learning is to make the distance between similar objects small while the distance between dissimilar objects as large as possible. This property can be beneficial to classification and clustering tasks, which have the same objective. However, there exists another kind of metric learning, where the aim is to learn the exact similarity value of complicated objects, such as the edit distance in graphs and strings. Thus, the goal of methods in this line of research differs from the previous line. These methods often predict a real-valued number and compare it with a ground truth similarity. Then the models are optimized in order to make the differences between the predicted similarity and the ground truth as small as possible. More recently, researchers have begun to combine representation learning with distance metric learning in many applications. These methods employ deep learning-based models to learn representations for complicated objects. Then the representations are used to compute predicted similarity. In this paper, our focus is on learning representation of sequence data and utilizing similarity metric learning to learn the representations for trajectories which are able to benefit trajectory similarity learning task.

B. Recurrent neural networks

Learning representations of many types of data has also been studied extensively. This includes studies on the representaitons of images, text, time series and graphs. Among the literature on representation learning, there is a stream that deals with preserving sequential information such that the learned representations can be used to solve downstream tasks. RNNs are developed to model the sequential information. These networks are able to use internal cells to process the elements in sequence. For example, long short-term memory network (LSTM) [28] and gated recurrent unit (GRU) [29] employ gated mechanism to process the input sequences. They use different gates to reserve or forget information of elements in sequences. Take LSTM as an example. It employs three gates, i.e., input gate, output gate and forget gate along with a memory cell to process the input data at every computation step. These gates can be thought as neurons and each of them has impact on the final output of LSTM. The input gate and the forget gate learn to update the cell state of LSTM. Then the output gate and the cell state output a vector at every time step. The output at a time step is the representation of this time step, so the output at the final time step is often used as the representation of the whole input sequence. Trajectories are simply sequences that contain location information, so such techniques can be naturally used for trajectory representation and its downstream tasks (See [1]) such as trajectory similarity learning (e.g., [20]), trajectory classification (e.g., [30]–[32]) and trajectory prediction (e.g., [33], [34]).

C. Attention-based networks

Attention mechanisms have been widely used in various modeling tasks for different types of sequences. For example, attention mechanisms are often used to improve the effectiveness of sequential models on natural language processing (NLP) tasks. Attention is useful for computing the correlations between two elements in a sequence or the correlations between an element in the sequence and another one that belongs to another sequence. This property helps deep learning-based models look for the important part of the sequence to ensure that this part has more impact on the output of models. Self-attention mechanism, a more recent variant of attention, aims to study the correlations among all the elements in a sequence by calculating and preserving the importance of all elements during the representation learning process. Important elements are assigned larger attention scores. As such, the entire representation is affected by every element differently according to its importance. Transformer [35] is a representative neural network based on self-attention mechanism. It has attracted a lot of interests in the natural language processing community because it has strong learning ability to learn the sequence representations. It is based on an encoder-decoder framework and completely relies on a self-attention mechanism to learn representations for sequences. Previous work [22] incorporate a self-attention-based network to process the trajectories so that their model can capture the structural information of trajectories.

D. Learning-based Trajectory Similarity Computation

Significant to many trajectory applications, measuring the similarity of trajectories has been widely studied. Further, a range of distance metrics have been proposed to compute distance between trajectories. Since existing distance metrics assess trajectory similarity from different perspectives, it is necessary to design a model that is generic to kinds of

trajectory metrics. Previous studies have been devoted to approximating different distance metrics with the assistance of deep learning models. Among these works, NeuTraj [20], T3S [22] and Traj2SimVec [36] are representative learning-based models. We introduce these algorithms briefly and summarize the differences between their models and ours.

[20] firstly proposes a deep learning-based model named NeuTraj for trajectory similarity computation. NeuTraj learns the representations of trajectories mainly depending on a RNN model, i.e., LSTM network. LSTM takes the spatial information of trajectories as input. More specifically, it takes a point in the trajectory at every time step and retains a part of information from that point. The output is a vector at each time step, and the vector contains the spatial information from the start point to the current one. The representation of the final point is used to stand for the whole trajectory. In addition, NeuTraj employs grid cells to represent trajectories and introduces a grid-based memory module, namely SAM. SAM relies on a memory unit to memorize and retrieve information from the trajectories that NeuTraj has processed since the start of training. With the grid representation and the SAM module, the augmented LSTM can learn the trajectory representations with the historical information about processed trajectories. Generally speaking, NeuTraj learns the trajectory representations with the assistance of SAM module. And the representations are then used to approximate the similarity between the given pair of trajectories. T3S [22] employs a self-attention-based network to learn the representations of trajectories. [22] proposes that LSTM is suitable to ordered sequence data, however, it does not consider the importance of each individual element, and hence cannot properly capture the structure information of the single trajectory. Therefore, they employ a self-attention network to learn the intra-interactions between points within a trajectory so that T3S can learn the structural information of a trajectory. Moreover, T3S also uses LSTM to learn the spatial information of trajectories. By combining the results of the self-attention network and LSTM, they claim that the learned representations are able to preserve the structural and spatial information of trajectories at the same time and the output embedding vectors are used to approximate trajectory similarity. Traj2SimVec [36] is the state-of-the-art method on trajectory similarity learning task. It simplifies trajectories in order to shorten the sampling time and designs corresponding sampling method based on the simplified trajectories. Besides, Traj2SimVec incorporates a sub-trajectory loss function to leverage the distances between sub-trajectories and improve the performance of the whole model.

In conclusion, all the aforementioned methods employ LSTM as their backbone network and they all feed the coordinate tuples into LSTM directly. However, the downside that all these methods share is that they overlook the interactions between the points in trajectory pairs. That is, when the models process the points in a trajectory, they do not consider the interactions between it and other trajectories, which is vital in trajectory similarity computation. In addition, current methods only use attention mechanism to focus on a single trajectory. As described, NeuTraj and T3S simply exploit attention to calculate the importance of the points in one trajectory, which means the significance of the points in the other trajectory is ignored. Traj2SimVec exploits the sub-trajectory loss as supervision information to enhance the learning ability of the model, but it still does not consider the points in the other trajectory. To this end, we propose a matching-based approach in this paper such that TMN learns to simulate similarity computation between pairs of trajectories. Our approach calculates the importance of the points in both trajectories. This process can be regarded as matching points during the similarity computation, which simulates the calculation procedure of distance metrics. So the matching mechanism is able to benefit the whole framework when learning the representations for trajectories.

Remark 1: Recently, a framework called GTS [37] has been proposed, which aims to solve trajectory similarity learning over spatial network. There are several differences between GTS and the aforementioned approaches. First, GTS involves a trajectory similarity measurement that considers structure in the same way as a road network and calculates similarity as between nodes in a graph, whereas the aforementioned methods are designed for the metrics that do not involve network structures. Second, the format of input data is different between GTS and previous methods. The input data of previous methods are coordinate tuples of trajectories, while the input of GTS is a series of nodes in the network. Third, GTS maximizes the similarity between the most similar trajectory and the query trajectory, which is widely used in ranking based applications, while the other models employ regression loss function to learn the ground truth similarity. The main aim of GTS is different with our task and it studies the correlations between trajectories in the network, so we mainly compare the methods which are designed for approximating the conventional distance metrics in this paper.

III. Preliminaries

In this section, we provide definitions for trajectory and the target of trajectory similarity learning task. Then we introduce several well-known trajectory similarity metrics. The notations used in this paper are briefly summarized in Table I and fully explained throughout the paper.

1) Definition 1 (Trajectory): A trajectory is a sequence of sample points ordered in terms of time stamps t. Each sample point p is a location in 2-dimensional space. A trajectory T is represented by a series of sample points, i.e., $T = (p^{(1)}, p^{(2)}, \cdots, p^{(n)})$. Usually, a point $p^{(i)}$ is denoted as a longitude $lon^{(i)}$ and latitude $lat^{(i)}$.

2) Definition 2 (Trajectory similarity learning): Given a specific trajectory similarity metric $f(\cdot, \cdot)$ and a pair of trajectories T_i, T_j, trajectory similarity learning aims to learn an approximate similarity function $g(T_i, T_j)$ to minimize $||f(T_i, T_j) - g(T_i, T_j)||$.

3) Distance metrics: Quite a few trajectory distance metrics have been proposed to measure the (dis)similarity between

TABLE I
THE SUMMARY OF NOTATIONS

Notation	Definition
T	A trajectory, which is made up of a sequence of locations.
p	A point in a trajectory that represents a geographical location.
lon, lat	The longitude and latitude of a point in the trajectory.
$f(\cdot,\cdot)$	One specific distance metric that is used to measure trajectory similarity.
$f_P(\cdot,\cdot)$, $f_R(\cdot,\cdot)$, $f_L(\cdot,\cdot)$	Three additional distance measures used in our paper (ERP distance, EDR distance and LCSS distance).
$g(\cdot,\cdot)$	The approximation function for trajectory similarity computation learned by learning-based models.
W, b	The weights and biases of linear layers that are used in TMN.
X, $x^{(i)}$	The point embedding matrix and the point embedding for i-th point in the trajectory.
$m_{a\leftarrow b}^{(i,j)}$	The match score between the i-th point of T_a and the j-th point of T_b.
$P_{a\leftarrow b}$	The match pattern for T_a considering the points in T_b.
$\mu_{a\leftarrow b}$, $S_{a\leftarrow b}$	The matching information of points in T_b with a single point in T_a.
$M_{a\leftarrow b}$	The discrepancies between points in T_a and all the points in T_b.
Z_a	The outputs of LSTM containing all the time step's hidden vectors.
O_a	The final representations of points in T_a and can be used to compute trajectory similarity.
S_{near}, S_{far}	The near trajectories and far trajectories used as training samples.
D	The distance matrix which stores the ground truth distances between trajectories.
S	The similarity matrix that is transformed from the distance matrix.
L, L_{sub}, L_{entire}	The loss functions employed in TMN.

trajectories (see [38] for a comprehensive survey). Different distance metrics emphasize on different information contained in trajectories. To date, researchers have mostly experimented with the Hausdorff, Féchet and DTW distance metrics. In this work, we evaluate different methods under extensive distance metrics which contain not only the aforementioned metrics, but also some other algorithms including ERP, EDR and LCSS. Here we list the equations of computing these three distance metrics and we use f_P, f_R, f_L to represent ERP, EDR and LCSS, respectively.

$$\begin{aligned} f_P(T_i,T_j) = \min\{&f_P(T_i,R(T_j)) + d(g,H(T_j)),\\ &f_P(R(T_i),T_j) + d(H(T_i),g),\\ &f_P(R(T_i),R(T_j)) + d(H(T_i),H(T_j))\} \end{aligned} \tag{1}$$

$$\begin{aligned} f_R(T_i,T_j) = \min\{&f_R(T_i,R(T_j)) + 1), f_R(R(T_i),T_j) + 1,\\ &f_R(R(T_i),R(T_j)) + d(H(T_i),H(T_j))\} \end{aligned} \tag{2}$$

$$f_L(T_i,T_j) = \begin{cases} f_L(R(T_i),R(T_j)) + 1, \\ \qquad\qquad if\ d(H(T_i),H(T_j)) \le \epsilon \\ \max\{f_L(T_i,R(T_j)), f_L(R(T_i),T_j)\}, \\ \qquad\qquad otherwise \end{cases} \tag{3}$$

where $H(T_i)$ is the first (head) point in T_i and $R(T_i)$ stands for the sub-trajectories which contains the remaining points in T_i excluding the first point. From above functions, we can see that different distance measures emphasize different information, i.e., these methods compute similarity from different perspectives. This is why the particular distance metric used often depends on the requirements of the application.

IV. METHODOLOGY

This section begins with an introduction to our proposed TMN, which is used to approximate the similarity between trajectories. We then move on to detail our novel trajectory matching mechanism, followed by the sampling method and TMN 's training objective.

A. Overview

Existing distance metrics measure the trajectory similarity by considering trajectory features in various aspects. In addition to the geographical distance between two trajectories, the order of sequence elements and the match information among elements of different trajectories are of great significance when calculating trajectory similarity. For instance, the Fréchet distance takes the order of points along trajectories into consideration. EDR and LCSS both look for point match pairs between two trajectories and employ the length of the longest common subsequence or edit distance to compute the trajectory similarity, respectively. However, because different metrics evaluate different aspects of trajectory similarity, a unified model is needed that can support all these measures. Further, to date, most studies have focused on learning independent representations for each individual trajectory. Yet ignoring the interactions between points in a pair of trajectories may result in a low accuracy similarity approximation. As is well-known, most trajectory similarity metrics need to find matched pairs of points between trajectories. Therefore, we propose a matching mechanism that matches points such that their representations are learned dependently. The result is a more effective approximation of the similarity between trajectories.

The framework of our proposed TMN is shown in Figure 2. Given a pair of trajectories T_a and T_b, TMN takes their coordinate tuples as input features. By processing the coordinate tuples, TMN can not only capture the spatial information of each trajectory, but also learn a representation which performs well on trajectory similarity computation task. As each trajectory is essentially a sequence composed of coordinate tuples, an LSTM is used to learn the dependencies among elements of each sequence. In addition, inspired by the match procedures

of distance metric computations, we propose the matching mechanism to learn the interactions between trajectories when calculating their similarity. Intuitively, the former network, i.e., LSTM, mainly focuses on the intra-interactions within a trajectory, which helps TMN extract features of trajectories. On the other hand, the latter mechanism emphasizes on the inter-interactions between trajectories. The inter-interactions allow every point in the sequence to be matched to points in the other sequence. The effectiveness of deriving the point matching information in this novel way is outlined in the experiments in Section V.

B. Trajectory Matching Networks

As introduced in Section III, TMN model aims to learn a function g which encodes an input pair of trajectories into vectors, i.e., $g(T_a, T_b) \rightarrow (o_a, o_b)$. By this way, we can transform the distance between trajectories into the Euclidean distance between two vectors in the multi-dimensional space, i.e., $||T_a - T_b|| \rightarrow ||o_a - o_b||$. The goal is to minimize the gap between the ground truth similarity and predicted similarity.

Given a pair of trajectories T_a and T_b of lengths m and n respectively, i.e., $T_a = (p_a^{(1)}, p_a^{(2)}, ..., p_a^{(m)})$ and $T_b = (p_b^{(1)}, p_b^{(2)}, ..., p_b^{(n)})$, TMN computes the similarity between them as follows. Firstly, TMN makes the length of the trajectories equal by adding padding to the ending of the shorter one. For simplicity, suppose $m>n$ in our case and we pad these two trajectories' length to m. We pad trailing zero points to the end of the shorter one to make its length equal to m. Then TMN employs a linear layer to map the coordinates of points in trajectories into multi-dimensional vectors,

$$x^{(i)} = \sigma(W_0 \cdot p^{(i)} + b_0) \tag{4}$$

where $x^{(i)}$ is a $\hat{d}$-dimensional vector in our network and $\hat{d} = d/2$, thereafter both T_a and T_b can be represented using vectors $(x^{(1)}, x^{(2)}, ..., x^{(m)})$. All the point embeddings $x^{(i)}$ in each trajectory (T_a and T_b) are then represented as individual matrices (X_a and X_b respectively), where $X \in \mathbb{R}^{m \times \hat{d}}$. $\sigma(x)$ is a LeakyReLU function which is defined as follows:

$$\sigma(x) = \begin{cases} x, & if\ x \geq 0 \\ 0.1 * x, & otherwise \end{cases} \tag{5}$$

Next, TMN exploit the novel matching mechanism to capture the inter-information between a pair of trajectories. To measure how well two points in a pair are matched, the points in one trajectory need to be compared with the points in the other. For brevity, this process is only described for T_a as the process for T_b is exactly the same. For every point $p_a^{(i)}$ in T_a, TMN computes the dot product of its corresponding embedding $x_a^{(i)}$ with embeddings of points in T_b, i.e., $x_b^{(j)}$. The dot product result of $x_a^{(i)}$ and $x_b^{(j)}$ is regarded as the match score between $p_a^{(i)}$ and $p_b^{(j)}$. Then we apply a softmax layer to all the dot product results of $p_a^{(i)}$ with points $p_b^{(k)}$ where $p_b^{(k)} \in T_b$, thereby getting the match pattern $P_{a \leftarrow b}^{(i,\cdot)}$ for $p_a^{(i)}$. This match pattern measures the importance of points in T_b from the view of every point $p_a^{(i)}$ in T_a. The matching mechanism is essentially based on attention computation which calculates cross-trajectory's correlations across points in different trajectories. In this way, the match pattern measures how well a point in one trajectory is matched to points in the other. TMN computes the match patterns for all the points in T_a by multiplying the embedding matrix X_a and $X_b^{\mathcal{T}}$, which are also the attention scores for every point in T_a. Note, however, that the trajectories may contain padded points because the lengths of trajectories were made equal. To eliminate the effect of these padded points, masks are used to cover the results in the corresponding positions. In other words, the results of the padded points are covered by zeros. Further, masks are also employed to alleviate the influence of padded points after applying the softmax layer. The above processes can be described as follows:

$$m_{a \leftarrow b}^{(i,j)} = x_a^{(i)} \odot x_b^{(j)}, x_a^{(i)} \in T_a\ and\ x_b^{(j)} \in T_b \tag{6}$$

$$P_{a \leftarrow b}^{(i,j)} = \frac{exp(m_{a \leftarrow b}^{(i,j)})}{\sum_{k=1}^{n} exp(m_{a \leftarrow b}^{(i,k)})} \tag{7}$$

$$P_{a \leftarrow b} = softmax(X_a \cdot X_b^{\mathcal{T}}) \tag{8}$$

where $m_{a \leftarrow b}^{(i,j)}$ is the match score of $p_a^{(i)}$ and $p_b^{(j)}$. $P_{a \leftarrow b} \in \mathbb{R}^{m \times m}$ is the match pattern and $P_{a \leftarrow b}^{(i,j)}$ is the element at i-th row and j-th column. $X_b^{\mathcal{T}}$ represents the transpose of X_b. Equation 8 is the matrix format of Equation 7. Each row vector of $P_{a \leftarrow b}$ can be regarded as the importance of all the points in T_b to this row's corresponding point in T_a. Then TMN uses $P_{a \leftarrow b}$ computing the dot product with X_b to obtain the weighted representations of all points in T_b for each individual point in T_a. Note that these two matrices cannot be multiplied directly, since their shapes are not the same. Hence, both of them are expanded to the same shape, i.e., $P_{a \leftarrow b}^{e}, X_b^{e} \in \mathbb{R}^{m \times m \times \hat{d}}$. After applying dot product to $P_{a \leftarrow b}^{e}$ and X_b^{e}, TMN obtains the results $\mu_{a \leftarrow b}^{(i,\cdot)}$ for every point of T_a. For each point $p_a^{(i)}$, TMN adds up the vectors $\mu_{a \leftarrow b}^{(i,j)}$ where $j \in [1, n]$, to summarize the inter-trajectory information from points in T_b. Moreover, the embedding matrix of T_a is used to subtract the above dot product results. The subtracted results $M_{a \leftarrow b}$ reflect the discrepancies between $p_a^{(i)}$ and all the points in T_b. These cross-trajectory differences can be used by the succeeding network, which is expert in learning sequential information. We show all the operations described above with following equations:

$$\mu_{a \leftarrow b}^{(i,j)} = P_{a \leftarrow b}^{e(i,j)} \odot X_b^{e(i,j)} \tag{9}$$

$$S_{a \leftarrow b}^{(i)} = \sum_{j=1}^{n} \mu_{a \leftarrow b}^{(i,j)} \tag{10}$$

$$M_{a \leftarrow b} = X_a - S_{a \leftarrow b} \tag{11}$$

where $P_{a \leftarrow b}^{e}, X_b^{e}, \mu_{a \leftarrow b} \in \mathbb{R}^{m \times m \times \hat{d}}$, $S_{a \leftarrow b} \in \mathbb{R}^{m \times \hat{d}}$, $M_{a \leftarrow b} \in \mathbb{R}^{m \times \hat{d}}$. This matching mechanism is essentially based on an attention mechanism, which helps TMN to calculate the importance between two points from different trajectories. In TMN, all the points $p_a \in T_a$ are compared with points

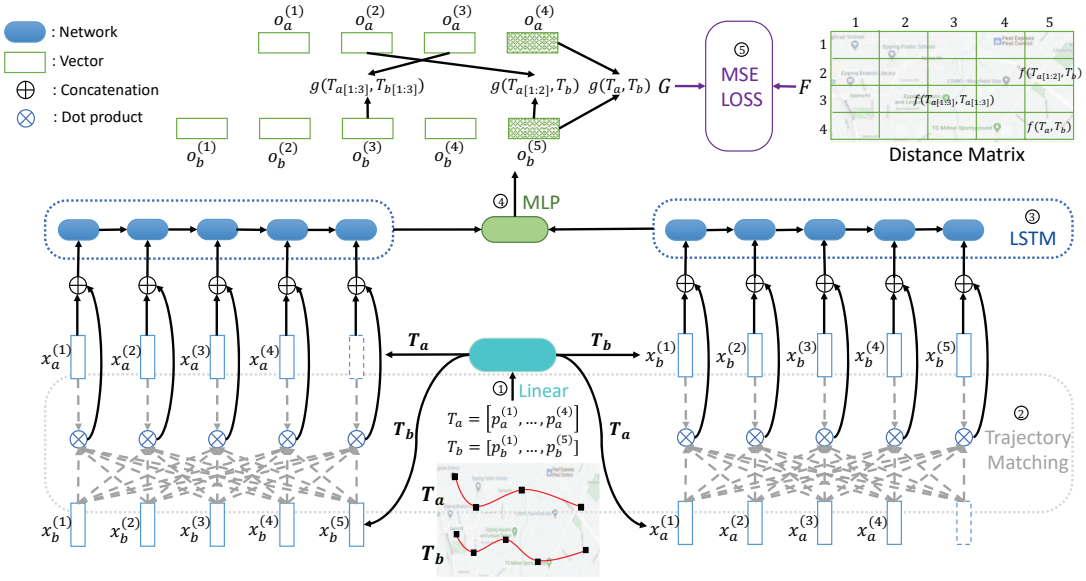

Fig. 2. The framework of TMN. Given a pair of trajectories, T_a and T_b, TMN takes them as inputs at the same time. The learning for the representations of these two trajectories is interdependent. The output of final time step is used to compute the trajectory similarity.

in T_b, so the match pattern $P_{a\leftarrow b}$ can be regarded as the attention matrix for T_a to measure the importance of T_b. As we mentioned in Section II, previous works [20], [22] exploit attention mechanism in their models. In the NeuTraj model, an attention mechanism is used to read more historical information. It employs a memory tensor to memorize the grid information of processed trajectories. And when NeuTraj reads a grid cell, it reads the information from its surrounding grid cells as well and NeuTraj uses attention mechanism on all the read grid cells to obtain a vector to represent this grid. As for T3S, it employs a self-attention network, which is similar to the attention mechanism in Transformer [35], to learn the structural information of a trajectory itself. Obviously, neither of them focuses on finding the match information between two trajectories and it is apparent that this is quit different with our matching mechanism. Again, we will show the effectiveness of our model in Section V.

Subsequently, the matching matrix $M_{a\leftarrow b}$ with the embedding matrix X_a are fed into a recurrent neural network, which performs well on handling the sequence learning. In the TMN, the LSTM network is utilized to learn the representations for the points in trajectories as follows:

$$Z_a = LSTM(X_a \oplus M_{a\leftarrow b}) \tag{12}$$

where $\oplus$ stands for the concatenation of two vectors and $Z_a \in \mathbb{R}^{m\times d}$ are the representations of points in T_a. In essence, Z_a is a matrix containing the outputs of all the time steps of the LSTM. There are two immediate advantages by employing LSTM, on one hand, LSTM learns the intra-trajectory information for every individual trajectory via its gated mechanism which is able to learn the spatial information of every point in trajectories at each time step. On the other hand, during the process of LSTM handling the input data, the discrepancies between points in T_a and T_b are accumulated and fed into it, making LSTM sensitive to the point match discrepancies. These differences between points assist TMN to generate point representations with cross-trajectory match information. Our work is to optimize a regression problem, which requires our model to output a real-valued number for the trajectory similarity of a given trajectory pair. Inspired by other regression tasks, we therefore turned to a Multi-Layer Perceptron (MLP) to process the row vectors of Z_a.

$$O_a = MLP(Z_a) \tag{13}$$

$O_a \in \mathbb{R}^{m\times d}$ denotes the final representations of all the points in T_a and can be used to predict the similarity between trajectories. We describe how TMN learns the representations O_a for T_a above and the process of learning the representations O_b for T_b is the same as O_a. Since the lengths of T_a and T_b are m and n respectively, the representations of T_a and T_b are represented by the m-th and n-th row vector of O_a, O_b respectively, i.e., $O_a^{(m)}, O_b^{(n)} \in \mathbb{R}^{1\times d}$. With trajectory representations $O_a^{(m)}$ and $O_b^{(n)}$, the similarity between them is calculated by the Euclidean distance between them: $||O_a^{(m)} - O_b^{(n)}||$.

C. Sampling method

TMN takes pairs of trajectories as inputs, therefore, pairs of trajectories need to be provided to TMN during training. Basically, trajectory similarity learning is based on metric learning [39] which aims to establish similarity or dissimilarity between trajectories. Specifically, two types of trajectories are sampled for each trajectory in the training set: those that are close to the anchor trajectory and those that are far away from the anchor trajectory. Both the near trajectories and the far trajectories can help TMN learn to approximate the trajectory similarity. These trajectory pairs are fed into TMN to

approximate their ground truth distances during training. Similarly, NeuTraj [20] also samples anchor-near pairs and anchor-far pairs as training data for their model. Traj2SimVec [36] compresses trajectories evenly into segments of the same number and then builds a k-d tree to store the simplified trajectories. With the assistance of k-d tree, Traj2SimVec chooses near samples from the k nearest neighbours of the anchor trajectory in the k-d tree. However, this method may suffer from the following drawbacks: The sampling method of Traj2SimVec keeps the same under all distance metrics, so it may not be able to well capture the similarity information for every distance metric. Besides, since the near samples are always chosen from the k nearest neighbors where k is set to 5 in Traj2SimVec, the model is only able to choose these data as near samples and ignore all the other points in the k-d tree.

In the TMN framework, we propose a different sampling method. For an anchor trajectory T_{anc}, we randomly select $2k$ samples from the training set. Then we sort all these samples according to their distances away from T_{anc} and the sorted samples are recorded in a list, i.e., $[T_1, T_2, \ldots, T_{2k}]$ with the closer samples at the front of the list. Next, we use the first k trajectories to form near training pairs and the last k trajectories as far training samples, i.e., $[T_1, T_2, \ldots, T_k] \in S_{near}$ and $[T_{k+1}, T_{k+2}, \ldots, T_{2k}] \in S_{far}$. This strategy ensures that the near samples are always more similar to the anchor trajectory than the far samples in every mini-batch. In addition, more trajectories can be sampled as near training samples with this method compared to Traj2SimVec and the training samples vary under different trajectory similarity metrics. We conduct experiments to show that our model has better performance than previous works through this sampling method.

D. Training Objective

In the previous subsection, we introduced our sampling method for trajectories in the training set. Next, we introduce the objective function for training TMN on the trajectory similarity learning task. Given a specific distance metric, we pre-calculate the distance matrix D for training and test use. During training, $D_{t_n \times t_n} \in \mathbb{R}^{t_n \times t_n}$ is exploited to provide the distance between training pairs, where t_n represents the number of trajectories in the training set. Since trajectory distance metrics measure the distance between trajectories instead of similarity, the distance matrix D is transformed into the similarity matrix $S \in \mathbb{R}^{n \times n}$. Similar to previous works, we employ exponential function to obtain normalized similarity matrix S, i.e., $S = exp(-\alpha \cdot D)$ where $S_{i,j} \in (0, 1)$ is used as the similarity between T_i and T_j.

Approximating the trajectory similarity is essentially a regression problem [40], therefore we employ mean square error (MSE) as our loss function. Our loss function is composed of two parts, considering various information contained in the trajectories. The first part of our loss function adds supervision information from the global perspective. The difference between the ground truth similarity of the trajectory pair and the predicted similarity is used to compute gradient and backpropagated to train the model. As we have introduced in previous subsection, the final output o_a of LSTM is used as the representation of the entire trajectory T_a. The Euclidean distance between two vectors of trajectories then reflects the predicted similarity between trajectories. And it is compared with the ground truth trajectory similarity as a loss function, which can be represented as follows:

$$L_{entire} = w_{as} * ||g(o_a, o_s) - f(T_a, T_s)|| \tag{14}$$

where T_a is the anchor trajectory, T_s is a near or far sample and o_a, o_s are their corresponding representations. w_{as} is a hyperparameter related to the similarity between T_a and T_s. Larger weights are assigned to trajectories that are more similar with T_a. To be specific, suppose that we sample n trajectories as near samples for an anchor trajectory, then we rank these samples and obtain a sample list $[T_{s1}, T_{s2}, ..., T_{sn}]$ according to their similarity with the anchor trajectory in descending order. Further, these ranked samples are assigned weights using the following list, $[\frac{2n}{n^2+n}, \frac{2(n-1)}{n^2+n}, ..., \frac{2}{n^2+n}]$. As shown, the nearest trajectory to the anchor trajectory is assigned the largest weight so that TMN is mostly affected by the nearer trajectories. Inspired by previous work [36], the second part of our loss function utilizes the rich sub-trajectory information contained in trajectories. Each trajectory contains many sub-trajectories, and the distances between all these sub-trajectories can be pre-calculated for use during training. These extra training data help to improve TMN 's learning ability. The procedure is as follows. The trajectories in the training set are divided into several sub-trajectories and the distances between every sub-trajectory pair are calculated. Then the sub-trajectory loss is calculated as follows:

$$L_{sub} = \frac{1}{r} * \sum_{i=1}^{r} w_{as} * ||g(o_a^{(i)}, o_s^{(i)}) - f(T_a^{(:i)}, T_s^{(:i)})|| \tag{15}$$

where $T_a^{(:i)}$ is the sub-trajectory which includes the first i points in T_a and $o_a^{(i)}$ is the corresponding vector. r is the number of sub-trajectory pairs used during training. In TMN, we sample sub-trajectories by taking the first point as start point and every 10th points as a new end point. For example, given a trajectory T_a of length 53, we sample five sub-trajectories $T_{a[1:10]}, ..., T_{a[1:50]}$ as training data. Finally, the overall loss function is determined by the above two terms, i.e.,

$$L = L_{entire} + L_{sub} \tag{16}$$

Besides the MSE loss function, we also experimented with the Q-error loss [41] which is also an effective loss function to cope with regression problem. However, the experiments on several trajectory similarity metrics show that it does not work as well as the MSE loss function in our framework. The experimental results are detailed in the next section.

V. Experiments

In this section, we compare TMN with current state-of-the-art methods using several popular distance metrics on the similarity search tasks. Besides, we study the impacts of our proposed matching mechanism and some other designs

on TMN. In addition, we analyze the influences of different parameters.

A. Experimental Settings

1) Datasets: We exploit two real-life datasets in our experiments, named Geolife and Porto:

- Geolife [42] was collected by 182 users in Beijing, China and it contains a broad range of human outdoor movements which are GPS locations of the users. In total, there are 17,612 trajectories in Geolife.
- Porto [43] contains more than 1.7 millions vehicle route trajectories, which were mainly collected by 442 taxis in Porto, Portugal.

Following previous works [20], [22], we filter out the trajectories that locate in the sparse area and remain the ones in the center area of the city for training and test. We also remove the trajectories less than 10 records. This is because it is more difficult and time-consuming to compute similarities with longer sequences. As such, these longer computation times more distinctly show the differences between models. Further, trajectory datasets are often characterized by many GPS errors and other problems, and, if affected, short trajectories suffer heavily from these errors. After preprocessing, there are around 8,000 trajectories in the Geolife dataset and 600,000 trajectories in the Porto dataset.

2) Compared models: To evaluate the effectiveness of TMN, we compare our model with following baselines:

- SRN [44] is based on siamese architecture and employs recurrent neural network to process time series. Trajectory is a kind of time series data, therefore [44] can be employed to approximate trajectory similarity. Following previous works [20], [22], we implement SRN with LSTM and apply it to trajectory similarity learning.
- NeuTraj [20] employs metric learning method for trajectory similarity learning task. It adds a newly-proposed memory unit called SAM to LSTM which stores and retrieves historical information of processed trajectories.
- T3S [22] proposes a neural network which combines LSTM with a self-attention-based network to learn the spatial and structural information of trajectories at the same time.
- Traj2SimVec [36] is the state-of-the-art method for trajectory similarity learning. It exploits the sub-trajectory information contained in trajectories so that their model benefits from auxiliary supervision.
- TMN is based on our novel matching-based mechanism which aims to match the points of different trajectories. By learning the inter-information and intra-information from trajectory pairs simultaneously, TMN learns trajectory similarity computation effectively.

3) Evaluation metrics: In order to evaluate the effectiveness of various baseline models and TMN, we employ Hausdorff, Féchet, DTW, ERP, EDR, LCSS distance metrics in our experiments. Following prior works [20], [22], we conduct trajectory similarity search to evaluate the performance of different models on the two datasets. We employ HR-10, HR-50 and R10@50 as major evaluation metrics [20], [22], [36]. HR-k is the top-k hitting ratio which examines the overlap percentage of ground truth trajectories that are recovered by the learned top-k results, i.e., the overlap percentage of the top-k results and the ground truth. Rk@t is the top-t recall for the top-k ground truth which evaluates the top-k ground truth recovered by the top-t produced by different methods. A higher recall value indicates better performance.

Moreover, we also evaluate the efficiency of compared models. Learning-based models often contain three phases for trajectory similarity computation, i.e., the training time of models, the generation time of trajectory embeddings and the similarity computation time. We evaluate and compare the time cost of all the models taken by these 3 phases.

4) Parameter settings: In the experiments, we implement TMN in the Python Pytorch framework along with each of the other models that are not open source, including T3S and Traj2SimVec. The training ratio of tr is 0.2 in our experiments, that is, 20% of trajectories in the dataset will be used as training set. We employ Adam [45] as the optimizer in TMN. The learning rate lr of TMN and the dimension of hidden vectors d are set to 5×10^{-3} and 128, respectively. The encoded data is stored in the GPU memory for all the methods. α, which is used to normalize the ground truth distance is set to 16 under DTW and ERP distances and 8 under Hausdorff, Féchet, EDR and LCSS distances. All the experiments are conducted on a machine with Intel Xeon E-2288G 8 cores CPU and NVIDIA Quadro RTX 6000 GPU.

B. Performance Evaluation

1) Effectiveness study: Following the literature [20], [36], we evaluate all the compared models in top-k similarity search task under 6 aforementioned distance metrics to show their effectiveness on trajectory similarity learning. For the Geolife dataset, we choose 9,000 trajectories for this task and 7,200 trajectories are used as test data. For the Porto dataset, 10,000 trajectories are randomly selected and 8,000 of them are used for testing. The experimental results are reported in Table II. As shown, TMN performs better than other models with both datasets in terms of 3 evaluation metrics. In particular, TMN significantly outperforms other methods on the Porto dataset. TMN is also obviously better than other models on DTW, ERP, EDR and LCSS. Previous study shows these four distance metrics are matching-based. These metrics find lots of point match pairs among two compared trajectories and then accumulate the distance for matched pairs or count the number of matched pairs. Our proposed matching mechanism aims to find match point for every point in trajectory, therefore TMN is able to better capture the match pairs of these metrics. The results on these distance metrics show the power of TMN and its matching mechanism. TMN improves the hitting ratio with the top-10 search and achieves much better gain with the top-50 search. In particular, our method achieves great improvement on the EDR distance on Porto dataset. Here, the hitting ratio of TMN on the top-10 and top-50 tasks is almost

TABLE II
PERFORMANCE EVALUATION FOR DIFFERENT METHODS UNDER POPULAR DISTANCE MEASURES.

Dataset	Method	DTW distance			Fréchet distance			ERP distance		
		HR-10	HR-50	R10@50	HR-10	HR-50	R10@50	HR-10	HR-50	R10@50
Geolife	SRN	0.2778	0.4009	0.6255	0.4670	0.6094	0.8152	0.7456	0.7974	0.9686
	NeuTraj	0.3002	0.4262	0.6619	0.5079	0.6617	0.8433	0.7896	0.8485	0.9884
	T3S	0.3208	0.4316	0.6601	0.5231	0.6732	0.8667	0.7931	0.8489	0.9914
	Traj2SimVec	0.3742	0.4864	0.6981	0.5058	0.6534	0.8530	0.7331	0.7707	0.9772
	TMN-NM	0.2585	0.4241	0.6148	0.4854	0.6490	0.8453	0.7577	0.8210	0.9658
	TMN	**0.4034**	**0.5961**	**0.7890**	**0.5295**	**0.6869**	**0.8669**	**0.8095**	**0.8915**	**0.9943**
Porto	SRN	0.3810	0.4952	0.7727	0.4720	0.5828	0.7749	0.5415	0.6929	0.9267
	NeuTraj	0.4341	0.5625	0.8390	0.5466	0.6524	0.8453	0.5586	0.7095	0.9310
	T3S	0.4345	0.5809	0.8350	0.5518	0.6560	0.8550	0.5756	0.7197	0.9443
	Traj2SimVec	0.5568	0.6098	0.8576	0.5475	0.6564	0.8596	0.4553	0.7358	0.8601
	TMN-NM	0.4613	0.6048	0.8763	0.5559	0.6649	0.8614	0.5471	0.7042	0.9254
	TMN	**0.6072**	**0.7511**	**0.9563**	**0.6012**	**0.7181**	**0.9006**	**0.6194**	**0.7537**	**0.9613**

Dataset	Method	EDR distance			Hausdorff distance			LCSS distance		
		HR-10	HR-50	R10@50	HR-10	HR-50	R10@50	HR-10	HR-50	R10@50
Geolife	SRN	0.1019	0.2093	0.2994	0.3075	0.4324	0.6662	0.3651	0.4135	0.5901
	NeuTraj	0.0617	0.1760	0.2419	0.3416	0.4761	0.6802	0.3515	0.3814	0.5629
	T3S	0.1089	0.2246	0.3338	0.3807	0.5463	**0.7690**	0.4027	0.4457	0.6426
	Traj2SimVec	0.1353	0.2746	0.3725	**0.3839**	0.5320	0.7488	0.4485	0.5051	0.7075
	TMN-NM	0.0745	0.2970	0.3707	0.3635	0.5311	0.7297	0.3805	0.4545	0.6199
	TMN	**0.1998**	**0.4147**	**0.5249**	0.3692	**0.5497**	0.7435	**0.4599**	**0.5421**	**0.7387**
Porto	SRN	0.2652	0.3395	0.5503	0.3800	0.4998	0.7421	0.4799	0.5492	0.7848
	NeuTraj	0.2814	0.3528	0.5580	0.4645	0.6009	0.8288	0.4910	0.5588	0.7847
	T3S	0.3027	0.3728	0.5874	0.4672	0.5977	0.8344	0.4907	0.5596	0.7817
	Traj2SimVec	0.3162	0.3680	0.5622	0.4970	0.6203	0.8475	0.5752	0.6506	0.8626
	TMN-NM	0.3808	0.5261	0.7887	0.4854	0.6214	0.8423	0.5769	0.6493	0.8733
	TMN	**0.5678**	**0.7400**	**0.9431**	**0.5128**	**0.6680**	**0.8856**	**0.6454**	**0.7299**	**0.8824**

TABLE III
EFFICIENCY STUDY OF EXACT DISTANCE METRICS AND LEARNING-BASED METHODS ON PORTO DATASET.

Dataset	Method	Time Cost		
		Training(s)	Inference(s)	Computation(s)
Porto	Fréchet	/	/	2125
	DTW	/	/	100
	ERP	/	/	260
	SRN	15.62	0.00004	0.0002
	NeuTraj	104.52	0.00059	0.0002
	T3S	55.31	0.00023	0.0002
	TMN	111.37	0.072	0.0002

[a] Training is the time cost of a learning-based model for each epoch.
[b] Inference is the average time for encoding a trajectory.
[c] Computation is the time of calculating similarity for trajectories.

2 times higher than it of current state-of-the-art methods. As to the other distance metrics, i.e., Fréchet and Hausdorff, TMN still shows small improvements with three evaluation metrics. Besides, as mentioned by [46], in some applications, it is sufficient to retrieve a part of the top-k results (e.g., report 10 results among top 50). In our experiments, R10@50 evaluates the top-10 ground truth recovered by the top-50 produced by models, and results show that our proposed model performs well under this important metric. Such evaluation metrics and popular distance metrics are able to prove that TMN has a good command of trajectory similarity learning task.

2) Efficiency study: In addition to the effectiveness of the models, we also evaluate the time efficiency of the exact distance metrics and learning-based methods. To comprehensively compare the efficiency of various methods, we divide the whole time into three parts, i.e., the training time, the inference time and the computation time. The training time and the inference time are concepts associated with learning-based methods. Specifically, the training time refers to the time cost a learning-based model spends in training for each epoch and the inference time is the average time for a model to encode a trajectory during test. The computation time is the cost of calculating similarity between two vectors for the learning-based models. In terms of the distance metrics, the computation time is the time cost of calculating distance according to the definition of metrics. The time costs of various methods are listed in the Table III. We randomly sample 1,000 trajectories from the Porto dataset as test data and it takes 2125, 100 and 260 seconds to compute the exact similarity scores of all pairwise similarities under Fréchet distance, DTW and ERP respectively. While it only takes 0.0001 second for learning-based models because they only need to compute the Euclidean distance between 128-dimensional vectors. Note that, the main focus of the study is to find good representations of the trajectories to preserve the pairwise trajectory similarity. Previous studies have used a straightforward implementation

TABLE IV
THE EFFECTIVENESS OF DIFFERENT SAMPLING METHODS UNDER VARIOUS DISTANCE MEASURES.

Dataset	Evaluation	Fréchet		DTW		ERP		Hausdorff		EDR		LCSS	
		TMN	TMN-kd	TMN	TMN-kd	TMN	TMN-kd	TMN	TMN-kd	TMN	TMN-kd	TMN	TMN-kd
Porto	HR-10	**0.6012**	0.5939	0.6072	**0.6117**	**0.6194**	0.6119	**0.5128**	0.4944	**0.5678**	0.3235	**0.6454**	0.5056
	HR-50	**0.7181**	0.6740	**0.7511**	0.7380	**0.7537**	0.7527	**0.6680**	0.6121	**0.7400**	0.3596	**0.7299**	0.5105
	R10@50	**0.9006**	0.8799	**0.9563**	0.9356	**0.9613**	0.9604	**0.8856**	0.8642	**0.9431**	0.6111	**0.8824**	0.7609

for the top-k similarity search in that they first compute the pairwise similarity between trajectories, and then sort the trajectories based on their similarity values. The top-k results are then retrieved. In our experiments, we use the implementation of the above top-k similarity search from [20]. So there is no difference among the learning-based models in terms of trajectory similarity computation (as shown in Table III) and top-k similarity search after the trajectories are encoded. Regarding the training time, there is no significant difference among the models. Specifically, these models all process 2,000 trajectories in each epoch and it takes 111.37, 104.52, 55.31 seconds for TMN, NeuTraj, T3S, respectively. Turning to the encoding time, TMN takes around 0.072 second to encode a trajectory, which is efficient in practice since we only need to encode each trajectory once. It takes around 0.00059 second for other learning models since they mainly rely on LSTM model for the encoding. And T3S is more efficient than other methods. In conclusion, the learning-based methods are highly time-efficient in terms of the similarity computation time, compared to calculating similarity directly according to the definitions of distance metrics.

C. Ablation Study

1) The effectiveness of matching mechanism: To further prove the significance of our proposed matching mechanism, we conduct ablation study on this mechanism. We remove the matching mechanism in TMN and name this variant as TMN-NM which stands for TMN with No Matching. To be specific, we remove the matching mechanism used in TMN, thus the input of LSTM becomes the point embedding matrix X and the rest structure of network remains the same as TMN. Then we conduct similarity search tasks using TMN-NM model on two datasets and results are also listed in Table II. Even with the matching mechanism removed, the performance of TMN-NM is still competitive with baseline models, such as T3S and NeuTraj, on the three evaluation metrics. The previous state-of-the-art method Traj2SimVec performs better than TMN-NM on most of distance metrics. But TMN-NM is able to outperform Traj2SimVec in several cases, for example, TMN-NM has better hitting ratio under ERP and EDR distance on the Porto dataset. TMN is clearly better than TMN-NM on all the distance metrics. In particular, TMN shows great improvement on matching-based distance metrics, e.g. DTW, ERP, EDR and LCSS distances. In conclusion, our novel matching mechanism has strong learning ability on trajectory similarity learning.

2) The effectiveness of different sampling methods: In Section IV, we introduce the sampling methods of TMN and Traj2SimVec. To further compare the impacts of the two methods, we replace the method in TMN with the sampling strategy of Traj2SimVec. We use TMN-kd to stand for the TMN with new sampling method and the structures of TMN keep unchanged. Experimental results are shown in Table IV. In terms of the evaluation metrics including HR-50 and R10@50, TMN always outperforms TMN-kd. While TMN-kd slightly outperforms TMN on the top-10 search task under Fréchet and DTW distance. The main difference is the sampling method for the near trajectories. The sampling method based on k-d tree always chooses near samples from the k nearest neighbors for the anchor trajectory. This may limit the contribution of near samples for optimizing the models. By contrast, our sampling method takes more trajectories into consideration when sampling near trajectories, and having more supervision information benefits TMN when learning to approximate trajectory similarity. Overall, our proposed sampling method has better effectiveness on the trajectory similarity learning task.

3) The effectiveness of different loss functions: In Section IV, we introduce the loss function used in TMN, i.e. the MSE loss function. Besides the MSE loss function, we compare Q-error [41] loss function with MSE loss function. Similar to MSE loss function, Q-error loss function is also used in the regression problems. MSE loss function computes the squared difference between the predicted values and the ground truth values, while Q-error loss function computes the fraction between the larger one of the predicted values, the ground truth values and the smaller value. Figure 3 shows the performance of TMN with different loss functions under four distance metrics, i.e., Fréchet distance, DTW distance, Hausdorff distance and LCSS distance. As shown in the figures, MSE loss function has better hitting ratios and recalls except for the R10@50 under LCSS distance. This may be caused by the fact that Q-error loss function may reduce the percentage of error when the error is close to 1. In addition, if the smaller value is too small, then the loss may be too large and the model will be significantly affected. These phenomena may lead to the inaccuracy of TMN in some situations, thus making TMN less effective.

D. Parameter Analysis

To evaluate the impacts of different settings of dimension d, learning rate lr and sampling number sn, we conduct parameter sensitivity experiments on Porto dataset under DTW distance. For each parameter, other parameters are fixed according to our default parameter settings. In addition, we also evaluate the influence of additional supervision from the sub-trajectories. Results are shown in Figure 4 and Figure 5.

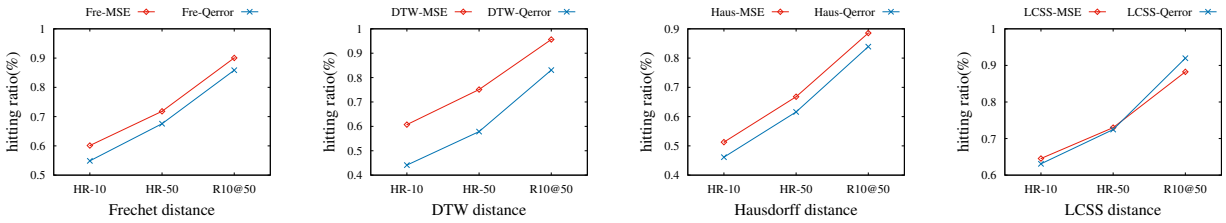

Fig. 3. The performance of TMN with different loss functions MSE and Q-error under four distance metrics on Porto dataset.

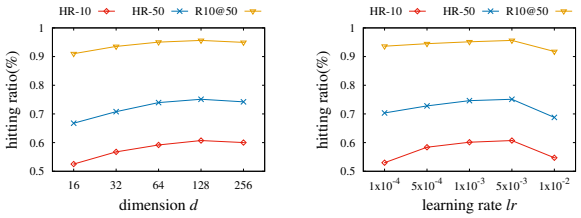

Fig. 4. The performance of TMN with different dimension d and learning rate lr.

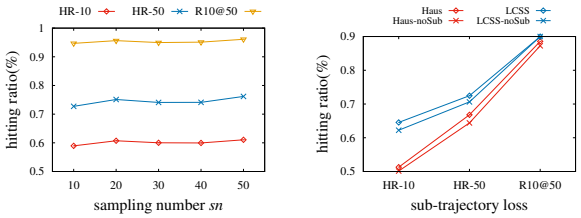

Fig. 5. The performance of TMN with different sampling number sn and the impact of sub-trajectory supervision information.

1) The influence of dimension: We vary the dimension d of hidden vectors in TMN from 16 to 256. The dimension of the hidden vectors influences the complexity of the model. When d is too large, the model becomes less efficient due to the increased time and space cost. If d is too small, it makes the model lack expressive to a certain extent. From Figure 4, it can be concluded that the best performance is reached when the embedding dimension is set to 128. Additionally, we conduct experiments with learning-based models to measure the space cost with different dimension settings. During training phase, it takes a total of 1.0 MB to store the embeddings of 2,000 training trajectories. Since the space required is linear to the number of trajectories, the trajectory embeddings required for most tasks should easily fit into memory.

2) The influence of learning rate: We vary the learning rate lr of our model from 1×10^{-4} to 1×10^{-2}. We can observe that large learning rate, e.g. 1×10^{-2}, leads to extremely poor performance of TMN. This may be because the loss function fluctuates too much, thus making it difficult for the model to converge. When the learning rate is too small, TMN requires much Our test indicates that 5×10^{-3} is a good choice of learning rate for TMN under DTW on the Porto dataset.

3) The influence of sampling number: We vary the sampling number sn of our model from 10 to 50. As we discussed in Section IV, a half of samples are exploited as near samples and the other half are far samples. We can observe that large sampling number is not able to make TMN work better. In fact, all it does is increasing the amount of memory required. However, 10 is too small a sampling number for the model to perform well. As shown, 20 is a good choice of sampling number for TMN under DTW distance.

4) The influence of sub-trajectory loss: In addition to aforementioned parameters, we also show the effectiveness of our additional supervision from sub-trajectories. As we mentioned in Section IV, the second part of our loss function exploits the rich spatial information contained in sub-trajectories. In this experiment, we remove the second part of our loss function to evaluate TMN on the Porto dataset. Results are shown in Figure 5, the blue lines are results tested under LCSS metric and the red lines are evaluated under Hausdorff distance. noSub refers to TMN trained without the sub-trajectory loss. As we can see, TMN is more effective when it is equipped with sub-trajectory loss function. This indicates that the sub-trajectory supervision is beneficial to TMN.

VI. Conclusion

Trajectory similarity computation under a variety of distance metrics is one of key building blocks in many trajectory-related computing and mining tasks. In this paper, inspired by the calculation process of trajectory distance metrics, we propose a novel trajectory matching network named TMN for trajectory similarity learning. Both the spatial information and the matching information play vital roles in trajectory representation learning in TMN and the learned representations are used to approximate trajectory similarity. Benefiting from the novel matching mechanism, the carefully designed sampling method and the appropriate loss criteria, TMN approximates trajectory similarity accurately under popular distance metrics. Experimental results on real-life datasets demonstrate the superiority of TMN over all the other existing approaches in terms of effectiveness. In addition, ablation studies show the significance of our matching mechanism and other designs.

Acknowledgment. We thank the reviewers for their thoughtful suggestions. Defu Lian is supported by the National Natural Science Foundation of China (No. 61976198 and 62022077). Ying Zhang is supported by ARC DP210101393 and ZJNSF LY21F020012. Lu Qin is supported by ARC FT200100787 and DP210101347. Wenjie Zhang is supported by ARC DP200101116 and FT210100303.

REFERENCES

[1] S. Wang, Z. Bao, J. S. Culpepper, and G. Cong, "A survey on trajectory data management, analytics, and learning," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 39:1–39:36, 2021.

[2] Z. Feng and Y. Zhu, "A survey on trajectory data mining: Techniques and applications," *IEEE Access*, vol. 4, pp. 2056–2067, 2016.

[3] Z. Li, J. Han, M. Ji, L.-A. Tang, Y. Yu, B. Ding, J.-G. Lee, and R. Kays, "Movemine: Mining moving object data for discovery of animal movement patterns," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 4, 2011.

[4] E. D'Andrea and F. Marcelloni, "Detection of traffic congestion and incidents from gps trace analysis," *Expert Systems with Applications*, vol. 73, pp. 43–56, 2017.

[5] H. Wu, W. Sun, and B. Zheng, "A fast trajectory outlier detection approach via driving behavior modeling," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*. ACM, 2017, pp. 837–846.

[6] Y. Wang, K. Qin, Y. Chen, and P. Zhao, "Detecting anomalous trajectories and behavior patterns using hierarchical clustering from taxi gps data," *ISPRS International Journal of Geo-Information*, vol. 7, 2018.

[7] K. Buchin, M. Buchin, D. Duran, B. T. Fasy, R. Jacobs, V. Sacristan, R. I. Silveira, F. Staals, and C. Wenk, "Clustering trajectories for map construction," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2017, pp. 1–10.

[8] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin, "Fast large-scale trajectory clustering," *Proc. VLDB Endow.*, vol. 13, no. 1, pp. 29–42, 2019.

[9] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of vehicle trajectories," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 647–657, 2010.

[10] T. Eiter and H. Mannila, "Computing discrete fréchet distance," Citeseer, Tech. Rep., 1994.

[11] C. Myers, L. Rabiner, and A. Rosenberg, "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1980.

[12] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," ser. SIGMOD '05. Association for Computing Machinery, 2005, p. 491–502.

[13] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04. VLDB Endowment, 2004, p. 792–803.

[14] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Proceedings 18th International Conference on Data Engineering*, 2002, pp. 673–684.

[15] P. K. Agarwal, K. Fox, J. Pan, and R. Ying, "Approximating dynamic time warping and edit distance for a pair of point sequences," in *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016*, vol. 51, 2016, pp. 6:1–6:16.

[16] A. Backurs and A. Sidiropoulos, "Constant-distortion embeddings of hausdorff metrics into constant-dimensional l_p spaces," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, France*, vol. 60, 2016, pp. 1:1–1:15.

[17] A. Driemel and F. Silvestri, "Locality-sensitive hashing of curves," in *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017*, vol. 77, 2017.

[18] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching trajectories by locations: an efficiency study," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*. ACM, 2010, pp. 255–266.

[19] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis, "Trajectory similarity join in spatial networks," *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1178–1189, 2017.

[20] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE, 2019, pp. 1358–1369.

[21] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[22] P. Yang, H. Wang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "T3s: Effective representation learning for trajectory similarity computation," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 2183–2188.

[23] Y. Li, X. Yu, and N. Koudas, "Les3: Learning-based exact set similarity search," 2021.

[24] X. Dai, X. Yan, K. Zhou, Y. Wang, H. Yang, and J. Cheng, "Convolutional embedding for edit distance," in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*. ACM, 2020, pp. 599–608.

[25] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 2018, pp. 617–628.

[26] X. Liu, X. Tan, Y. Guo, Y. Chen, and Z. Zhang, "Cstrm: Contrastive self-supervised trajectory representation model for trajectory similarity computation," *Computer Communications*, 2022.

[27] Z. Zhang, J. Bu, M. Ester, Z. Li, C. Yao, Z. Yu, and C. Wang, "H2MN: graph similarity learning with hierarchical hypergraph matching networks," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*. ACM, 2021, pp. 2274–2284.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[30] Q. Gao, F. Zhou, K. Zhang, G. Trajcevski, X. Luo, and F. Zhang, "Identifying human mobility via trajectory embeddings," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, ser. IJCAI'17, 2017, p. 1689–1695.

[31] Y. Endo, H. Toda, K. Nishida, and J. Ikedo, "Classifying spatial trajectories using representation learning," *International Journal of Data Science and Analytics*, vol. 2, 07 2016.

[32] J. Bian, D. Tian, Y. Tang, and D. Tao, "Trajectory data classification: A review," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, pp. 33:1–33:34, 2019.

[33] Q. Liu, S. Wu, L. Wang, and T. Tan, "Predicting the next location: A recurrent model with spatial and temporal contexts," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16, 2016, p. 194–200.

[34] F. Zhou, X. Yue, G. Trajcevski, T. Zhong, and K. Zhang, "Context-aware variational trajectory encoding and human mobility inference," in *The World Wide Web Conference*, ser. WWW '19, 2019, p. 3469–3475.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 5998–6008.

[36] H. Zhang, X. Zhang, Q. Jiang, B. Zheng, Z. Sun, W. Sun, and C. Wang, "Trajectory similarity learning with auxiliary supervision and optimal matching," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 2020, pp. 3209–3215.

[37] P. Han, J. Wang, D. Yao, S. Shang, and X. Zhang, "A graph-based approach for trajectory similarity computation in spatial networks," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 556–564.

[38] H. Su, S. Liu, B. Zheng, X. Zhou, and K. Zheng, "A survey of trajectory distance measures and performance evaluation," *VLDB J.*, vol. 29, no. 1, pp. 3–32, 2020.

[39] M. Kaya and H. S. Bilge, "Deep metric learning: A survey," *Symmetry*, vol. 11, no. 9, p. 1066, 2019.

[40] J. Sun, G. Li, and N. Tang, "Learned cardinality estimation for similarity queries," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD'21. Association for Computing Machinery, 2021, p. 1745–1757.

[41] G. Moerkotte, T. Neumann, and G. Steidl, "Preventing bad plans by bounding the impact of cardinality estimation errors," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 982–993, 2009.

[42] Y. Zheng, X. Xie, and W. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.
[43] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi-passenger demand using streaming data," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 3, pp. 1393–1402, 2013.
[44] W. Pei, D. M. J. Tax, and L. van der Maaten, "Modeling time series similarity with siamese recurrent networks," 2016.
[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
[46] W. Wang, R. C.-W. Wong, and M. Xie, "Interactive search for one of the top-k," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1920–1932.