

A Linear Time Approach to Computing Time Series Similarity based on Deep Metric Learning

Di Yao, Gao Cong, Chao Zhang, Xuying Meng, Rongchang Duan, Jingping Bi

Abstract—Time series similarity computation is a fundamental primitive that underpins many time series data analysis tasks. However, many existing time series similarity measures have a high computation cost. While there has been much research effort for reducing the computational cost, such effort is usually specific to one similarity measure. We propose NEUTS (**N**eural metric learning for **T**ime **S**eries) to accelerate time series similarity computation in a generic fashion. NEUTS computes the similarity of a given time series pair in linear time and generic to handle any existing similarity measures. NEUTS samples a number of seed time series from the given database, and then uses their pair-wise similarities as guidance to approximate the similarity function with a neural metric learning framework. NEUTS features two novel modules to achieve accurate approximation of the similarity function: (1) a local attention memory module that augments existing recurrent neural networks for time series encoding; and (2) a distance-weighted ranking loss that effectively transcribes information from the seed-based guidance. With these two modules, NEUTS can yield high accuracies and fast convergence rates even if the training data is small. Our experiments with five real-life datasets and four similarity measures (Fréchet, Hausdorff, ERP and DTW) show that NEUTS outperforms baselines consistently and significantly. Specifically, it achieves over 80% accuracies in most settings, while obtaining 50x-1000x speedup over brute-force methods and 3x-350x speedup over approximate algorithms for top-k similarity search.

Index Terms—deep metric learning, time series similarity, linear time



1 INTRODUCTION

Computing the similarity between two time series is a fundamental primitive that underpins many searching and mining tasks for time series analysis. Various measures have been proposed to capture the intrinsic structural similarities between time series, such as Dynamic Time Warping (DTW) [41], the Hausdorff distance [4], the Fréchet distance [2], and Edit distance with Real Penalty (ERP) [9]. These similarity measures play a key role in the successes of computational biology [22], self-driving car [29], financial analysis [3] and other tasks. For example, computing the similarity of DNA sequences is critical to functional gene classification which can be used for diagnosing diseases; the similarities between stocks prices are widely used to detect similar changing patterns and fundamental to stock price prediction.

Unfortunately, the high computation cost of existing similarity measures has become the bottleneck for time series analysis at scale. To compute the similarity between two time series, existing techniques often require to align the points in the two time series, accumulate the information among all aligned pairs, and finally produce the distance. Such a process incurs quadratic time complexity and limits

many time series mining algorithms from being applied to large datasets. For instance, it takes us more than 6.5 hours to compute the pair-wise Hausdorff distances for merely ~8000 2D human movement sequences on a server with two Intel Xeon E7 CPUs. As massive time series data are being collected at an unprecedentedly scale in many scenarios, fast time series similarity computation for different measures has become a pressing need.

There are two major difficulties in accelerating time series similarity computation for different measures. The first is the requirement of shifting and scaling time series during the similarity computation process. As a pair of input time series may have completely different lengths, capturing their similarities often requires finding the best alignment with flexible shifting and scaling instead of simple head-to-tail matching. As such, most existing techniques have to employ a scan-and-align mechanism for determining the best matching, and it is hard to decouple the matching process and reduce the time cost. The second is the variations across different similarity measures. Prevailing similarity measures (e.g. DTW, Hausdorff, Fréchet) differ a lot in their definitions and computation mechanisms. It is challenging to design a generic accelerating strategy that accommodates all the existing measures.

There have been considerable research efforts attempting to accelerate time series similarity computation for various tasks. Such efforts generally fall into two categories. The first [10], [19], [20], [39] is to reduce the involved computation at a global level. However, the techniques along this line are often exclusively designed for the top- k similarity search task. Instead of reducing the computation complexity for an ad-hoc pair of time series, they focus on designing indexing and pruning strategies for a given time series database

- Di Yao, Xuying Meng and Jingping Bi are with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
E-mail: yaodi, bjp@ict.ac.cn
- Gao Cong is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore.
- Chao Zhang is with the School of Computational Science and Engineering, Georgia Tech, GA, USA.
- Rongchang Duan is with National Computer Network Emergency Response Technical Team/Coordination Center of China.

Manuscript received April 19, 2005; revised August 26, 2015.
Concise vision of this paper has been accepted in ICDE 2019.

and reducing the computations for top- k similarity search. As such, they cannot be applied for tasks that require the distances between all pairs of time series such as clustering and anomaly detection. The second line aims at directly reducing the time complexity for time series similarity computation with approximations or heuristics. Different strategies have been proposed for different measures, such as locality sensitive hashing (LSH) for the Fréchet distances [15] and constrained strategies [34], [35] for DTW to reduce the computation cost. Unfortunately, these approximation techniques are designed for specific measures, and cannot be generalized to other measures.

In this paper, we propose a general method that drastically accelerates time series similarity computation, which is applicable to any existing measure. Our proposed method, named NEUTS (**N**eural metric learning for **T**ime **S**eries), is an approximate approach based on neural metric learning. NEUTS samples a pool of seed time series from the database and uses their pair-wise similarities as guidance. Specifically, NEUTS learns a neural network that jointly embeds input time series and approximates the distance function. It has the following attractive characteristics:

- **Generic:** Unlike previous methods that are specific to one similarity measure, NEUTS is generic enough to support any existing measure. It can thus be used for accelerating most time series analysis tasks utilizing different similarity measures.
- **Fast:** Given an ad-hoc pair of time series, NEUTS is able to compute their similarity in $O(L)$ time complexity, where L is the length sum of the pair. Practically, we find it 50x-1000x faster than brute-force computations in many cases.
- **Accurate:** NEUTS achieves superb approximation performance in practice. On five real time series datasets, NEUTS achieves over 80% hitting ratio for Fréchet, Hausdorff, DTW and ERP, which outperforms approximation methods [5], [15].
- **Elastic:** NEUTS embeds the time series without losing its inherent structural information which makes it elastic to be extended by other indexing and pruning strategies. In tasks like top- k similarity search, existing indexing methods [10], [20], [39] can be incorporated into NEUTS to further reduce the computational cost.

The core of NEUTS is a deep metric learning framework that uses recurrent neural networks (RNNs) to generate time series embeddings. We sample a pool of seed time series from the database and compute their pair-wise similarities. With the computed seed similarities as guidance, we design a pair-wise loss to optimize the network for fitting seed similarities. NEUTS features two novel modules that encourage the network to approximate the similarity function accurately: (1) *Local attention memory (LAM)*. Vanilla RNNs along with its existing variants (GRU, LSTM) can only model one sequence without considering between-sequence correlations. Our designed LAM units memorize the information from previously processed time series with the attention mechanism, and capture the correlations between training time series to produce better time series embeddings. (2) *Distance-weighted ranking loss*. One difficulty of making use of the seed time series is the dilemma

between efficiency and effectiveness. On one hand, training the network sufficiently would preferably iterate over all pairs of time series. On the other hand, a full enumeration of all pairs of time series incurs expensive computation time. To address this dilemma, we propose a distance-weighted sampling strategy to focus on the more discriminative training pairs. Along with the weighted sampling strategy, our proposed distance-weighted ranking loss learns parameters to conform to the guidance from the seeds. With these two novel modules, NEUTS can yield high accuracies and fast convergence rates with the seed training data.

Our contributions can be summarized as follows:

- 1) We propose a neural metric learning method for accelerating time series similarity computation under different measures. To the best of our knowledge, NEUTS is the first method that supports accelerating different time series similarity measures in a generic way, making it applicable to a wide range of applications.
- 2) We propose the local attention memory unit to model the correlations between locally close time series based on an attention network and external memory tensor. Further, we design a weighted sampling and learning module that better utilizes seed time series as training data. Compared with existing architecture, our deep metric learning model yields faster convergence rates and higher accuracies.
- 3) We conduct extensive experiments on five real time series datasets, ranging from one-dimensional to three-dimensional. The results demonstrate that the proposed model consistently outperforms baselines in both accuracy and efficiency. Better still, it can be extended to zero-shot learning scenarios.

Compared with our preliminary work [40], we have made extensions as follows:

- 1) The problem is generalized to accelerate the similarity computation of general time series, apart from trajectory data. Accordingly, a local attention memory module, which extends the spatial attention memory [40], is designed to capture the correlations between locally close time series.
- 2) We design a pseudo cell state strategy to ensure that the local attention memory module is able to augment other gated recurrent units that do not have an explicit cell state. As a result, a new recurrent unit, namely LAM-augmented GRU, is proposed and evaluated to show the usefulness of local attention memory module.
- 3) Several new experiments have been added to evaluate the performance of NEUTS on time series datasets:
 - Besides the trajectory datasets (2D time series), we tested our model on three real life time series datasets, including two 1D time series datasets and one 3D time series dataset.
 - New experimental results of LAM-augmented GRU were reported in several tasks, such as Top-K similarity search, time series classification and time series clustering.
 - To examine the effectiveness of NEUTS on labeled data, we added a new task, *i.e.*, time series classification. We compared the accuracy of NEUTS with its guided similarity measure.

- We added model adaptation experiments to test the robustness of NEUTS under data scarce scenarios.
- Efficiency analysis of time series clustering on several datasets has been added.

2 RELATED WORK

We provide an overview of existing studies related to NEUTS from three perspectives: (1) time series similarity computation; (2) deep metric learning; and (3) memory networks.

2.1 Time Series Similarity Computation

Similarity Measures. Many algorithms have proposed to measure the similarity of a pair of time series, such as DTW [41], Hausdorff [4], Fréchet [2], ERP [9]. Due to the different length of time series, these similarity measures need perform a head-to-toe alignment procedure, which leads to quadratic or super-quadratic complexity. Therefore, accelerating the similarity computation of time series has become a critical problem for time series analysis.

Accelerating Similarity Computation. Various techniques have been proposed to accelerate time series similarity computation, which can be broadly categorized into two categories. The first category uses indexing and pruning techniques to reduce the involved computations at a global level. Most techniques in this category focus on 2-dimensional time series and employ tree-based index structures [10], [12], [19], [24], [36], such as K-D tree or R-tree, to organize time series data in a hierarchy data structure. Based on the index, bounding-box-based pruning techniques are employed to eliminate unnecessary computations. Thus time series [10], [12] or its sub-segments [20], [39] in a bounding box which are too faraway to belong to the top- k results are pruned. However, the techniques in this category are specifically designed for the top- k similarity search problem. They do not reduce the complexity of computing the similarity between a pair of time series. Hence, they cannot be applied for tasks that require the similarities of all pairs, such as DBSCAN for time series.

The second category aims at designing approximate algorithms to speed up similarity computation for a pair of time series. Most techniques in this category treat each time series as a curve and address the problem from a computational geometry point of view. Focusing on the Hausdorff distance, Farach-Colton *et al.* and Backurs *et al.* [5], [18] proposed embedding-based methods for approximating nearest neighbor search. Salvador *et al.* [1] proposed an approximate algorithm which can fast compute the DTW distance. Thanawin *et al.* [32] proposed a method which omits the square computation step to speed up DTW computation. Li *et al.* [25] proposed a new trajectory similarity measure that computes the similarity of two trajectories based on deep representation learning. Very recently, Driemel *et al.* [15] proposed a locality sensitive hashing (LSH) based algorithm for fast computing the Fréchet and Hausdorff distances. Although these algorithms can achieve high computation efficiency, they suffer from two shortcomings. First, they rely on hand-crafted heuristics and normally cannot obtain satisfactory accuracies in all similarity measures. In our experiments, we observed that these algorithms can generate

poor approximations in many cases. Second, they are all designed for specific measures. It is hard to adapt these techniques for other similarity measures.

2.2 Deep Metric Learning

NEUTS is related to the recent development of deep metric learning, which aims at learning a distance function that measures how similar two objects are, based on neural networks. Bromley *et al.* [6] pioneered deep metric learning and proposed the classic Siamese network for signature verification. Qian *et al.* [31] used precomputed activation features to learn a feature embedding for classification. Pei *et al.* [30] extended the Siamese network to learn a similarity metric for sentences. Our method differs from the above models in two aspects. First, they are all designed for modeling one sequence independently, while ours employs the local attention mechanism to capture the correlations among all the time series. Second, they all use random sampling to generate training samples and could suffer from low convergence for time series data.

2.3 Memory Network

NEUTS is also related to memory networks [21] in the deep learning field. The embryonic form of memory network is proposed to solve the tasks that need to model long term dependency [21]. Weston *et al.* [38] first employed a long-term memory component and defined the read and write operation for Question Answering(QA). Then Sukhbaatar *et al.* [37] extended the architecture to recurrent neural network. In the proposals [8], [38], the memory network was extended to a hierarchical structure. But these memory structures are designed without considering the local information and cannot be used directly for time series modeling.

3 PRELIMINARIES

3.1 Problem Definition

We consider a time series database \mathcal{T} and a similarity function $f(\cdot, \cdot)$. Each time series $T \in \mathcal{T}$ is a sequence of points recording the attributes of an object. Without loss of generality, we consider two-dimensional time series. That is, each time series $T = [X_1^c, \dots, X_t^c, \dots]$ is a sequence of tuples where $X_t^c(x_t, y_t)$ is the t -th attributes of the object. For any two time series $T_i, T_j \in \mathcal{T}$, the function $f(T_i, T_j)$ measures the similarity between T_i and T_j . Here, $f(\cdot, \cdot)$ could be the DTW similarity, the Hausdorff distance, the Fréchet distance, or any other similarity measure. We omit the detailed definitions of these measures due to the space limitation.

Our problem is to compute the similarity for an ad-hoc pair of time series from \mathcal{T} under the similarity function $f(\cdot, \cdot)$. However, for most prevailing similarity measures, computing the similarity between a pair of time series incurs quadratic time complexity. Hence, the research question is: how can we learn an approximate similarity function $g(\cdot, \cdot)$, such that computing $g(T_i, T_j)$ takes linear time while the difference $|f(T_i, T_j) - g(T_i, T_j)|$ is minimized.

At the high level, NEUTS adopts a neural metric learning framework. It randomly samples N time series from \mathcal{T} as

TABLE 1
Notations used in this paper

Notations	Description
\mathcal{T}, \mathcal{S}	A time series database and a pool of N seeds time series which are random sampled from the database.
T	A time series in \mathcal{T} which consist of a sequence of feature tuples.
n	A time series in \mathcal{T} which consist of a sequence of feature tuples.
\mathbf{D}, \mathbf{S}	The distance and similarity matrices of \mathcal{S} which have the same size: $\mathbb{R}^{N \times N}$.
\mathbf{M}	The memory tensor which stores the local information of $P \times Q$ grids.
$\mathbf{E}_i, \mathbf{E}_j$	d -dimensional embedding vectors of T_i and T_j learnt by NEUTS.
X_t	The input of NEUTS at t -time step which contains the coordinate input X_t^c and grid input X_t^g .
$\mathbf{W}, \mathbf{U}, \mathbf{b}$	The linear weights and bias in recurrent units.
$\mathbf{f}_t, \mathbf{i}_t, \mathbf{o}_t$	The forget, input and output gates in LAM-augmented LSTM unit.
$\mathbf{r}_t, \mathbf{u}_t$	The reset and update gates in LAM-augmented GRU units.
\mathbf{s}_t	Novel local gate in LAM-augmented units which controls the <i>read</i> and <i>write</i> operations on \mathbf{M} .
$\mathbf{c}_t, \hat{\mathbf{c}}_t$	The cell state and intermediate cell state in LAM units at t -time step which store the information of the processed $t - 1$ steps.
$\mathbf{h}_t, \mathbf{h}_{t-1}$	The hidden states in RNN units at time step t and $t - 1$.
\mathbf{G}_t	The local information matrix of X_t with shape $\mathbb{R}^{(2w+1)^2 \times d}$, where w is the scanning bandwidth.
\mathbf{A}	The local attention weight that reflects the similarity weights of $\hat{\mathbf{c}}_t$ over \mathbf{G}_t .
$\mathbf{c}_t^{\text{cat}}, \mathbf{c}_t^{\text{his}}$	The intermediate concatenated state and the final historical state in memory <i>read</i> operation at t -time step.
T_a	The anchor time series which is sampled from the seeds for training NEUTS.
$\mathcal{T}_a^s, \mathcal{T}_a^d$	n similar time series and n dissimilar time series of T_a which are sampled from the seeds for fitting the pair-wise similarities.
$\mathbf{S}_a^s, \mathbf{S}_a^d$	The ground truth similarities of both similar and dissimilar pairs of T_a
$\hat{\mathbf{S}}_a^s, \hat{\mathbf{S}}_a^d$	The similarities of T_a which are calculated by NEUTS.

the pool of seeds \mathcal{S} and computes a symmetric $N \times N$ distance matrix \mathbf{D} for \mathcal{S} . Then it transforms the original distance matrix into a normalized similarity matrix \mathbf{S} . Leveraging the matrix \mathbf{S} as guidance, NEUTS further learns a neural network, which maps arbitrary-length time series into low-dimensional space to capture their similarities. More formally, for any two input time series T_i and T_j ($i, j \in [1, \dots, N]$), NEUTS projects them to two d -dimensional vectors \mathbf{E}_i and \mathbf{E}_j , respectively. The learned mapping should be similarity preserving, namely $f(T_i, T_j) \approx g(T_i, T_j)$ where $g(\cdot, \cdot)$ is the similarity between \mathbf{E}_i and \mathbf{E}_j in the embedding space. Figure 1 illustrates the architecture of NEUTS. It consists of two major parts: local attention memory(LAM) augmented RNN encoder and seed-guided metric learning method.

3.2 Overview of NEUTS

LAM Augmented RNN Encoder. NEUTS relies on recurrent neural networks (RNN) to model the time series and takes the last hidden state of RNN as the embedding vector. However, as aforementioned, vanilla RNNs and its variants (GRU, LSTM) capture the information of each sequence independently. For similarity computing, the correlations between time series are critical. It is important to leverage

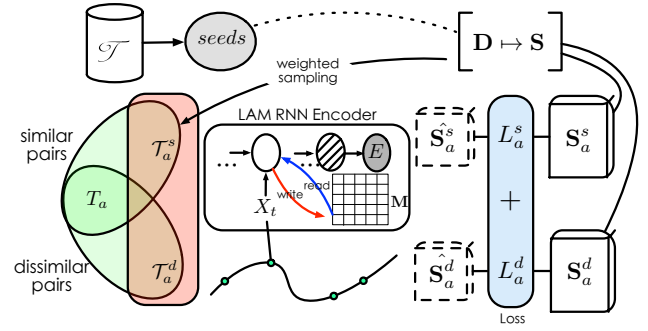


Fig. 1. Architecture of NEUTS. Taking similar and dissimilar pairs of anchor T_{a_i} as input, NEUTS first generates the embedding of each time series, and then fits the pair-wise similarity guided by the ground truth in \mathbf{S} .

the information of locally close time series previously seen to guide the metric learning process. Thus, we design a local attention memory module in NEUTS. It employs a local memory tensor to store the local space information of previously processed time series. The memory tensor underpins *read* and *write* operations over the entire space based on the soft attention mechanism, such that the information of previously seen time series can be encoded and retrieved on demand.

Seed-Guided Neural Metric Learning. Based on LAM-augmented RNN, NEUTS builds a seed-guided metric learning architecture to consume a pair of time series, and learns the network to approximate the similarity matrix \mathbf{S} . Existing metric learning methods employ random sampling approaches to produce training pairs, which implies all time series are equally weighted. But this assumption dose not hold in time series metric learning as it ignores the local proximity between time series. Particularly, we develop a distance-weighted sampling procedure and a ranking loss objective to solve this problem. Unlike previous random sampling, the distance-weighted sampling focuses on the more discriminative training pairs from the seed time series. With the weighted sampling strategy, each seed time series T_a is associated with one similar list \mathcal{T}_a^s and one dissimilar list \mathcal{T}_a^d in the pool. The ranking loss then is used to learn the parameters of the network for fitting the similarities to \mathbf{S} and preserving the ranking order in both similar and dissimilar pairs.

4 LAM AUGMENTED RNN ENCODER

We introduce the Local Attention Memory (LAM) module that augments existing RNN architectures for time series encoding. Below, we first introduce the local attention memory structure. Then we explain the high-level idea of how local information affects the time series similarity computation. After that, we present two novel RNN units, LAM-augmented LSTM and LAM-augmented GRU, which augment existing recurrent neural networks with the LAM. Finally, we detail the *read* and *write* operations of the memory in LAM-augmented recurrent units. For convenience of presentation, we employ two-dimensional time series to explain the idea of the proposed method, which can be easily adapted to other input dimensionality.

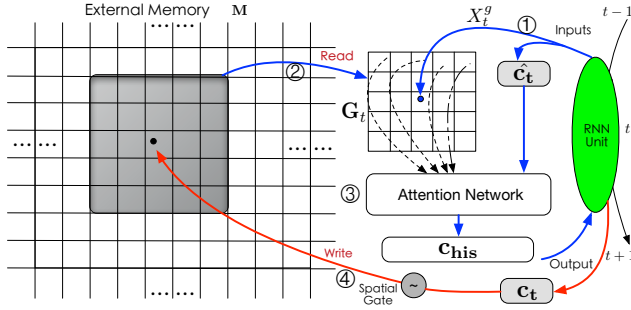


Fig. 2. Illustration of the proposed local attention memory (LAM) with scanning bandwidth $w = 2$. At each time step, LAM takes two inputs, the input grid X_t^g and the intermediate cell state \tilde{c}_t . The reader first scans the memory M to get $(2 * 2 + 1)^2 = 25$ grid embeddings. Then it calculates and outputs the attention cell state c_t^{his} . The writer (red lines) updates M with the cell state c_t in the recurrent unit.

4.1 Grid-Based Memory Tensor

The LAM module is a grid-based memory network. As a preprocessing step, we partition the two-dimensional input space into small 2D grids. Then any time series $T = [X_1^c, \dots, X_t^c, \dots]$ can be mapped into a sequence $T^g = [X_1^g, \dots, X_t^g, \dots]$ where $X_t^g = (x_t^g, y_t^g)$ specifies the grid at t -th position. Figure 2 shows the architecture of proposed LAM module. As shown, the core part of LAM is a memory tensor M . The memory tensor M stores vector representations for all the grids in the space, which enable encoding and retrieving information for previously seen time series. Formally, suppose the entire 2D space is partitioned into $P \times Q$ grids, then the dimensionality of the memory tensor is $\mathbb{R}^{P \times Q \times d}$, where d is the hidden size of the recurrent unit, P and Q are the partition sizes of the two dimensions respectively. Each slice $(p, q, :)$ in M stores the embedding vector of grid (p, q) and all grid embeddings are initialized with 0 before training. As NEUTS continuously processes the time series in the training data, the memory tensor M will be updated accordingly by memory-augmented recurrent units to encode the information in processed time series.

4.2 Local Information Used in NEUTS

Before introducing the memory-augmented recurrent units, we first analyze the definition of similarity measures and explain why M can benefit the similarity computation. As shown in Figure 3, most of the time series similarity measures involve two procedures to generate the similarity values, *i.e.* sequence alignment and summarization. The computational costs in these measures are mainly produced by sequence alignment procedure, which iteratively computes the distance between the point in one sequence and all points in another sequence. For one specific time series T , this procedure is highly coupled with another involved time series, *e.g.* T_1 in Figure 3. Therefore, we have to start the similarity computation of each new time series pairs from scratch, and no precomputation can be done before knowing the exact pair. Therefore, the coupled alignment procedure is a key bottleneck in large time series similarity computation.

In this paper, we propose to use grid-based local memory tensor M to bridge this gap and decouple the sequence

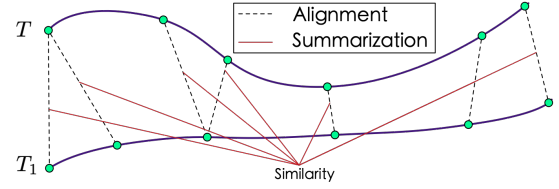


Fig. 3. Illustration of the procedures for similarity computation.

alignment. In our method, NEUTS encodes time series to a fixed-length vector independently and models the correlation of time series pair by sharing the same memory tensor in M . Instead of aligning the points explicitly, NEUTS sequentially takes the local information of each point as the input to generate time series embedding. It makes sure that the points having similar previous sequences also have similar inputs. Taking T and T_1 in Figure 5 as an example, NEUTS selects the same parts of M as the local information inputs when the points are located in the same grid, such as X_t . Then, the encoder decides how to use the local information according to the previous sequence similarity of T and T_1 . Furthermore, both the encoder and the memory tensor are learnable and well-designed. We can optimize their parameters to approximate the exact similarities. More details are discussed in the next sections.

4.3 LAM-Augmented RNN Units

The LAM module allows for memorizing and retrieving information from processed time series. We leverage it to ameliorate standard RNN units. In this way, the RNN encoder captures the information from not only the current time series, but also those similar ones previously processed. In what follows, we show the details of LAM-augmented LSTM and LAM-augmented GRU.

4.3.1 LAM-Augmented LSTM

Figure 4(a) shows the architecture of LAM-augmented LSTM unit and the green parts are the novel LAM module. As shown, at each time step t , the unit takes $X_t = (X_t^c, X_t^g)$ and the previous hidden state \mathbf{h}_{t-1} as the input and outputs \mathbf{h}_t to the next recurrent step. As in LSTM, LAM-augmented LSTM employs gating mechanism to control the operations on the cell state \mathbf{c}_t that stores core information of the processed sequence. The recurrent step is performed as follows:

$$(\mathbf{f}_t, \mathbf{i}_t, \mathbf{s}_t, \mathbf{o}_t)^T = \sigma(\mathbf{W}_g \cdot X_t^c + \mathbf{U}_g \cdot \mathbf{h}_{t-1} + \mathbf{b}_g) \quad (1)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot X_t^c + \mathbf{U}_c \cdot \mathbf{h}_{t-1} + \mathbf{b}_c) \quad (2)$$

$$\hat{\mathbf{c}}_t = \mathbf{f}_t \cdot \mathbf{c}_{t-1} + \mathbf{i}_t \cdot \tilde{\mathbf{c}}_t \quad (3)$$

$$\mathbf{c}_t = \hat{\mathbf{c}}_t + \mathbf{s}_t \cdot \text{read}(\hat{\mathbf{c}}_t, X_t^g, M) \quad (4)$$

$$\text{write}(\mathbf{c}_t, \mathbf{s}_t, X_t^g, M) \quad (5)$$

$$\mathbf{h}_t = \mathbf{o}_t \cdot \tanh(\mathbf{c}_t) \quad (6)$$

where $\mathbf{W}_g \in \mathbb{R}^{4d \times 2}$, $\mathbf{U}_g \in \mathbb{R}^{4d \times d}$, $\mathbf{W}_c \in \mathbb{R}^{d \times 2}$, $\mathbf{U}_c \in \mathbb{R}^{d \times d}$ and d is the hidden state size. All of the gates $(\mathbf{f}_t, \mathbf{i}_t, \mathbf{s}_t, \mathbf{o}_t)$, cell states $(\tilde{\mathbf{c}}_t, \hat{\mathbf{c}}_t, \mathbf{c}_t)$ and hidden states $(\mathbf{h}_t, \mathbf{h}_{t-1})$ have the same shape: $d \times 1$.

To obtain the hidden state \mathbf{h}_t , the unit performs the following steps: (1) Gate operations. By Equation 1, the unit applies a sigmoid function σ on the linear transformation of the coordinate input X_t^c and the previous hidden state \mathbf{h}_{t-1} to obtain the four gates: forget gate \mathbf{f}_t , input gate \mathbf{i}_t ,

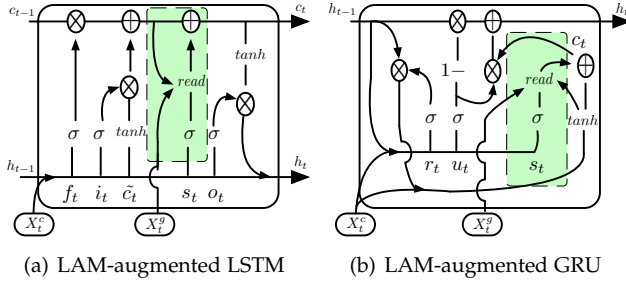


Fig. 4. Memory-augmented RNN units.

local gate s_t and output gate o_t . (2) Cell state operations. By Equations 2 ~ 4, the unit produces cell state of current time-step c_t , based on the f_t , i_t , s_t and the inputs (X_t^c, X_t^g) . (3) Hidden state operations. By Equation 6, the unit generates hidden state h_t and output it to the next recurrent step. After that, the unit updates the memory tensor M by formula 5.

The main novelty of LAM-augmented LSTM lies in Equation 4 that augments \hat{c}_t with historical information by the *read* operations.

4.3.2 LAM-Augmented GRU

Unlike LSTM, standard GRU does not have cell states. To cooperate LAM with GRU, we construct a pseudo cell state c_t . It stores information for the current time step but does not output to the next time step. As shown in Figure 4(b), there are three gates in LAM-GRU: (1) the reset gate r_t , (2) the update gate u_t ; and (3) the local gate s_t . The LAM-GRU unit updates as follows:

$$(r_t, u_t, s_t)^T = W_I \cdot X_t^c + U_h \cdot h_{t-1} + b \quad (7)$$

$$\hat{c}_t = \tanh(W_c \cdot X_t^c + \sigma(r_t) \cdot U_c \cdot h_{t-1}) \quad (8)$$

$$c_t = \hat{c}_t + \sigma(s_t) \cdot \text{read}(\hat{c}_t, X_t^g, M) \quad (9)$$

$$h_t = (1 - \sigma(u_t)) \cdot c_t + \sigma(u_t) \cdot h_{t-1} \quad (10)$$

$$\text{write}(c_t, s_t, X_t^g, M) \quad (11)$$

where $W_I \in \mathbb{R}^{3d \times 2}$, $U_h \in \mathbb{R}^{3d \times d}$. With c_t , the LAM module can also augment to GRU in the same way. The details of *read* and *write* operations are described in the next section.

4.4 Attention-Based Reads and Writes

With the memory tensor M , NEUTS uses the attention mechanism to capture the information in processed time series: (1) the *read* operation retrieves relevant grid embeddings from M to augment encoding the current time series; and (2) the *write* operation attends to relevant grids and updates grid embeddings with the information of the current time series. As shown in Figure 2, we mask the blue and red arrays to show the order of *reads* and *writes* operations. The *read* operations in LAM module consist of three sub-procedures which are shown in Figure 2 ① ~ ③, respectively. The *writes* operation is shown in Figure 2 ④.

4.4.1 Local Memory Reader

The reader retrieves information from the memory and uses the information to augment RNN-based time series encoding. As shown in Figure 2-①, at each step t , the reader takes two inputs: (1) grid input X_t^g ; and (2) intermediate cell state \hat{c}_t . With these two inputs, the reader outputs a vector c_t^{his} ,

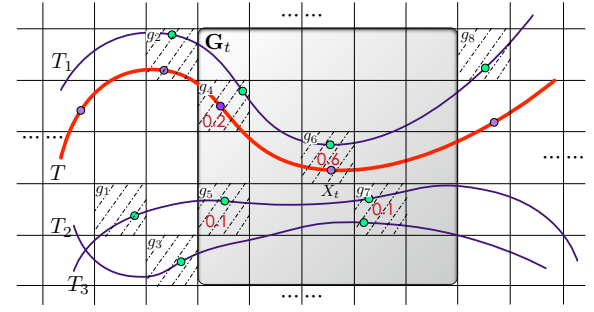


Fig. 5. Example to illustrate local reads and writes. $T_1 \sim T_3$ are previously processed time series; T is the current processing time series and X_t is the current processing input. After processed $T_1 \sim T_3$, grid embeddings of $g_1 \sim g_8$ are updated to non-zero value. By local readers, NEUTS scans the local closets grids of X_t and computes the attention weights of grids for encoding T . The attention weights of non-zero grids $g_4 \sim g_7$ are the red values. After that, the cell state of X_t is used to update the grid embedding of g_6 .

which augments \hat{c}_t with the influence of previously processed time series close to X_t^g , i.e. $c_t^{\text{his}} = \text{read}(\hat{c}_t, X_t^g, M)$.

Illustrated by Figure 2-②, the attentional reader first looks up the grids that are locally close to $X_t^g = (x_t^g, y_t^g)$. Specifically, the reader uses the bandwidth w to perform a memory scan and identifies the grids around (x_t^g, y_t^g) : $\text{scan}(x_t^g) = [x_t^g - w, x_t^g + w]$; $\text{scan}(y_t^g) = [y_t^g - w, y_t^g + w]$. The read grid embeddings are stored in a matrix G_t with shape $\mathbb{R}^{(2w+1)^2 \times d}$. After memory scan, the reader employs the attention network to transform the matrix G_t to an d -dimensional vector, which is shown in Figure 2-③. The attention mechanism is performed as follows:

$$A = \text{softmax}(G_t \cdot \hat{c}_t); \text{mix} = G_t^T \cdot A;$$

$$c_t^{\text{cat}} = [\hat{c}_t, \text{mix}]; c_t^{\text{his}} = \tanh(W_{\text{his}} \cdot c_t^{\text{cat}} + b_{\text{his}})$$

where the matrix W_{his} and b_{his} are the parameters of the attention network. $A \in \mathbb{R}^{(2w+1)^2 \times 1}$ is the attention weight that reflects the similarity of the \hat{c}_t over the historical grid embedding matrix G_t . For example, non-zero attention weights over grids $g_4 \sim g_7$ in Figure 5 are 0.2, 0.1, 0.6, 0.1, which indicates the embedding of current time series is more similar to T_1 than T_2 and T_3 . $\text{mix} \in \mathbb{R}^{d \times 1}$ is the weighted sum of the G_t to concatenate to \hat{c}_t . Finally, a fully connected layer transforms c_t^{cat} to fit the current state and generate the local attention cell state c_t^{his} . Once c_t^{his} is generated, we can combine it with the intermediate cell state \hat{c}_t to get the final cell state c_t by Equation 4.

4.4.2 Local Memory Writer

The grid embeddings in M should be updated during the training process. At each step, the writer directly performs sparse updating of the corresponding entry in the memory M based on s_t :

$$M(X_g)_{\text{new}} = \sigma(s_t) \cdot c_t + (1 - \sigma(s_t)) \cdot M(X_g)_{\text{old}}.$$

By this formula, the grid embeddings are essentially the weighted average of processed pass-by time series. Note that we are using the same local gate s_t for both the reader and writer. It is because s_t reflects not only the confidence level that c_{his} is useful for the current input, but also how much information in the current input is suitable for

updating $M(X_g)$. Another benefit of sharing the same gate is that it limits the number of parameters of our model.

5 SEED-GUIDED NEURAL METRIC LEARNING

In this section, We first describe the metric learning procedures of NEUTS and then present the weighted sampling and optimization method to learn the model parameters.

5.1 Metric Learning Procedures of NEUTS

Figure 1 illustrates our NEUTS model that uses neural networks to embed time series into d -dimensional space and approximates the similarity function. As shown, the core of NEUTS is the local memory-augmented RNN encoder, which generates latent vector representation for any time series. For an input time series, the final hidden state of our RNN encoder is used as its representation.

Given a pool of seed time series \mathcal{S} and their distance matrix \mathbf{D} , NEUTS normalizes \mathbf{D} to a similarity matrix \mathbf{S} and uses the \mathbf{S} as guidance. For any two input time series T_i and T_j ($i, j \in [1, \dots, N]$), the RNN encoder is able to project them to two d -dimensional vectors \mathbf{E}_i and \mathbf{E}_j . Our goal is to learn the network parameters such that similarity between \mathbf{E}_i and \mathbf{E}_j is close to the original similarity $f(T_i, T_j)$. But directly fitting all similarities in \mathbf{S} is intractable and will lead to over-fitting. We need select discriminative pairs for optimization. Formally, the loss function of NEUTS is the weighted sum of MSE of all pairs: $\min \sum_{k=1}^K w_k \cdot (f(T_i, T_j) - g(T_i, T_j))^2$, where K is the number of discriminative pairs and w_k is the weight of pair k .

To this end, we design a similarity-preserving ranking objective in NEUTS. Specifically, for any anchor time series T_a , we will sample a set of similar neighbors \mathcal{T}_a^s , as well as a set of dissimilar neighbors \mathcal{T}_a^d from the seed time series, to form the similar and dissimilar pairs. With the sampled pairs, we design a weighted ranking objective, which encourages the network to learn a regression function for fitting the similarities to \mathbf{S} , as well as preserving the ranking order in both similar and dissimilar pairs. In what follows, we introduce our weighted sampling and optimization procedure.

5.2 Weighted Sampling and Optimization

Our NEUTS model for metric learning is related to the classic Siamese network [6]. The Siamese network uses random sampling method to generate training pairs, and learns a regression function to fit the target measure. However, such a random sampling strategy implies all pairs have the same weight to the total loss. This assumption does not hold for time series metric learning as it ignores the input space proximity between time series. Given one anchor time series, we need to focus on the most similar time series and the most dissimilar ones, because they are more discriminative than others. Directly using random sampling and treating all pairs equally can lead to slow convergence and weak accuracies.

To remedy the above problem, we propose a distance-weighted sampling and optimization strategy. We first

transform the original distance matrix \mathbf{D} into a normalized similarity matrix \mathbf{S} as follows:

$$S_{i,j} = \exp(-\alpha \cdot D_{i,j}) / \sum_{n=1}^N \exp(-\alpha \cdot D_{i,n})$$

where α is a tunable parameter controlling the similarity value distribution. The reason behind the transformation is that the distribution of raw distances often obey to power-law distributions and the magnitude of the distance can span a large range. The transformation is essentially a smoothing operation which brings the similarity values into the range $[0, 1]$ and smooths the distribution.

Inspired by [26], our distance-weighted sampling procedure works as follows. We take time series in the pool of N seeds as anchor time series sequentially. For one anchor time series T_a , we take the corresponding row from the similarity matrix \mathbf{S} as the importance vector \mathbf{I}_a . With the entries in \mathbf{I}_a as importance weights, we sample n distinct time series as similar samples: $\mathcal{T}_a^s = \{T_1^s, \dots, T_n^s\}$. Conversely, we sample another n dissimilar samples $\mathcal{T}_a^d = \{T_1^d, \dots, T_n^d\}$ using the entries in $1 - \mathbf{I}_a$ as importance weights. Then we rank the similar samples with the decrease of its similarity to T_a and rank dissimilar samples with the increase order. Finally, we obtain $2n$ pairs for T_a .

After sampling, we generate the time series embeddings and define the pair-wise similarities for the anchor time series over the similar and dissimilar pairs as follows:

$$\begin{aligned} \hat{\mathbf{S}}_a^s &= \hat{\mathbf{S}}(T_a, \mathcal{T}_a^s) = [g(T_a, T_1^s), \dots, g(T_a, T_n^s)] \\ \hat{\mathbf{S}}_a^d &= \hat{\mathbf{S}}(T_a, \mathcal{T}_a^d) = [g(T_a, T_1^d), \dots, g(T_a, T_n^d)] \end{aligned} \quad (12)$$

where $g(T_i, T_j) = \exp(-\text{Euclidean}(\mathbf{E}_i, \mathbf{E}_j))$ computes the similarity between two time series embeddings, and \mathbf{E} is the embedding of the corresponding time series.

Coupled with weighted sampling, we propose a weighted ranking loss which is motivated by list-wise ranking [7] and Mean Reciprocal Rank [27]. Given a ranked list of n sampled time series, we set their ranking weights as $\mathbf{r} = (1, 1/2, \dots, 1/l, \dots, 1/n)$ and normalize the weights by $\sum_{l=1}^n r_l$. For the n similar pairs, their weights decrease with the ranking order, namely the most similar time series in \mathcal{T}_a^s is regarded as the most important. Thus we define the loss for similar pairs of T_a as:

$$L_a^s = \sum_{l=1}^n r_l \cdot (g(T_a, T_l^s) - f(T_a, T_l^s))^2 \quad (13)$$

where $f(T_i, T_j)$ is the ground truth similarity of (T_i, T_j) .

For dissimilar pairs, it is not reasonable to focus more on fitting the similarity value. Instead, we design a margin loss to separate dissimilar time series from the anchor time series:

$$L_a^d = \sum_{l=1}^n r_l \cdot [\text{ReLU}(g(T_a, T_l^d) - f(T_a, T_l^d))]^2 \quad (14)$$

The $\text{ReLU}(x) = \max(0, x)$ function defines the margin loss as follows: when $g(T_a, T_l^d) - f(T_a, T_l^d) < 0$, $L_a^d = 0$, meaning that dissimilar sample is faraway enough from the anchor time series in the embedding space; when $g(T_a, T_l^d) - f(T_a, T_l^d) > 0$, $L_a^d > 0$, the embeddings should be adjusted to enlarge the embedding-based distance of the dissimilar sample of the anchor. Finally, the loss for the

given \mathcal{S} is the sum of the similar and dissimilar samples over all N seeds.

$$L_{\mathcal{S}} = \sum_{a \in [1, \dots, N]} (L_a^s + L_a^d)$$

Since all the modules and the loss functions are differentiable, all the parameters in NEUTS can be learned in an end-to-end manner. In the training process, we update the parameters with back-propagation through time (BPTT) algorithm and employ Adam optimizer for stochastic optimization.

6 DISCUSSIONS

6.1 Complexity Analysis

The time complexity of NEUTS for computing the similarity of a time series pair includes two parts: the embedding part and the distance computation part. For embedding, the computation is linear to the number of recurrent operations. In one time step, the higher-order term of complexity in classic recurrent units is $(m + 1) * d^2$, where m is the number of gates, *e.g.*, $m = 3$ for LSTM. In LAM units, extra computation cost is involved by a new gate and an attention reader which complexity is also quadratic of d . For a pair of time series, d is a constant and the complexity of distance computation in the embedding space is a constant. The overall time complexity of NEUTS is thus linear in the length of the time series.

For a time series database, the time series embeddings only need to be computed once. When new time series query is conducted, we generate the embedding of new time series and perform search based on the distance of embeddings. So the computation is linear with the size of search space, which makes NEUTS suitable for large dataset.

6.2 Theoretical Explanation

Similarity measures are used for mapping time series to a metric space. In order to make NEUTS general for various distance measures, we employ a recurrent neural network, which is well known to be a sufficient approximator for an arbitrary mapping function [23], to learn an approximate metric spaces via time series embeddings. Under the constrain of pair-wise similarities of seed time series, we employ the objection function, which is the weighted sum of the MSE of discriminative time series pairs, for learning the parameters of the encoder and minimize the difference between the true metric and the learned embedding based metric. We empirically find the difference is small, which demonstrates that NEUTS works well with small size of seeds.

However, the effects of the neural network module is double-edged. While benefiting the generic property, NEUTS has no provable theoretical guarantee of accuracy. We empirically find that NEUTS can learn a similarity preserving metric space with small size of seed time series.

7 EXPERIMENT

We conduct extensive experiments on five real datasets to evaluate the effectiveness and efficiency of the proposed

method. Our experimental evaluation is designed to answer several research questions(RQs).

- **RQ1:** How well do the learned embedding-based similarities perform in downstream tasks?
- **RQ2:** What are the capabilities of the proposed local attention module and the distance weighted sampling?
- **RQ3:** How efficient is NEUTS in accelerating time series computation?
- **RQ4:** What are the influences of different hyper-parameter settings?
- **RQ5:** How robust is NEUTS in zero-shot learning scenario?

7.1 Experimental Settings

7.1.1 Datasets.

Our experiment are based on five public datasets which can be categorized into three groups.¹

- The first group contains two one-dimensional time series, **ElectricDevices(ED)** and **ItalyPowerDemand(IPD)**, that are selected from UCR archives [11]. ElectricDevices consists of 8,100 time series. ItalyPowerDemand consists of 1,000 time series.
- The second group is two-dimensional datasets. We use two public trajectory datasets in two cities: Beijing and Porto. The first dataset [43], referred **Geolife(GL)**, consists of 17,621 trajectories of human mobility from 2007 to 2010. The second dataset [28], referred **Porto**, consists of over 1.7 millions of taxi trajectories from 2013 to 2014. These trajectories are the sequence recording of GPS locations which have two dimensions: latitude and longitude. To moderate the dimensions of **M**, we choose trajectories in the centre area of the city and discretize the area into $50m \times 50m$ grids. Then, we remove the trajectories less than 10 records. After such preprocessing, we obtain 8203 2D time series(trajectories) in Geolife and 601,071 2D time series(trajectories) in Porto.
- The third group is three-dimensional dataset. We observe that some one-dimensional datasets in UCR archives are different sensor's records for an identical time period. Here, we combine three one-dimensional datasets, **uWaveGestureLibraryX**, **uWaveGestureLibraryY** and **uWaveGestureLibraryZ** into one three-dimensional dataset which is denoted as **UWaveGestureLibrary(UGL)**. It consist of over 4,400 time series.

7.1.2 Experimental Tasks

We design several tasks to examine the effectiveness, efficiency and robustness of NEUTS. To evaluate the effectiveness of NEUTS and answer **RQ1** & **RQ2**, we consider three widely used downstream tasks for time series similarity computation:

- **Top- k similarity search.** Given a time series database and a specific time series, it aims to find top K similar time series in the dataset based on a similarity measure.
- **Time series classification.** In this task, every time series has one label. Given a new time series, we utilize 1-NN

1. Code and data available at <https://github.com/yaodi833/NeuTS>

classifier to label it with the label of the most similar training time series.

- Time series clustering. In the experiment, we evaluate the difference between ground truth similarities and embedding-based similarities with three clustering algorithms, *i.e.*, Hierarchical clustering, KMedoids clustering and DBSCAN.

For these tasks, we also test the efficiency of NEUTS and answer **RQ3**. In addition, we evaluate the parameter sensitivity and answer **RQ4** with top- k similarity search.

To test the robustness of NEUTS in data-scarce scenario and answer **RQ5**, we have designed two zero-shot learning tasks.

- Synthetic time series experiment. In this case, we focus on 2D trajectory time series, for which we can easily generate time series synthetically. We first generate synthetic trajectories based on the road networks in Beijing. Then, NEUTS was trained on these synthetic trajectories and tested on real-world data.
- Model adaptation experiment. In this experiments, we conduct two adaptation settings *i.e.* (1) training with time series in **Geolife** and testing on **Proto** (2) training with time series in **ED** and testing on **IPD**. For example, we directly use the models trained on **ED** to calculate the time series similarities of **IPD** dataset and evaluate the performance.

7.1.3 Experimental Protocol

In this section, we show the detailed experimental protocol of different tasks separately.

Top-K similarity search. We perform top- k similarity search on five datasets and evaluate the results under four similarity measures: the Fréchet distance, the Hausdorff distance, Edit distance with Real Plenty(ERP) and Dynamic Time Warping (DTW). The first three are metric, namely the distance is symmetric and satisfies the triangle inequality. We thus learn the models to approximate the metrics directly. However, DTW is not a metric mainly because it does not satisfy the triangle inequality. Experiments on DTW explore the performance of NEUTS on non-metric similarity measure.

The ground-truth of the problem is the exact top- k results based on the accurate similarity. Due to the enormous time series, it is impractical to compute the exact similarity of all time series pairs directly. For datasets that have more than 10000 samples, such as **Porto**, we only sample 10000 time series as the subdatasets for this experiment and pre-compute the pair-wise similarity of them. For long time series, such as **ED** and **UGL**, directly computing the exact pair-wise similarities is extremely time consuming. We downsample the time series in these two datasets with gaps 5 and 15 respectively and shorten the length of them to less than 20. We random choose 20% time series as the seeds and use their similarities to train NEUTS. Additionally, 10% time series are used for tuning parameters and 70% are used for testing. Performance results of this experiment are shown in Section 7.2.1.

Time series classification. We perform the time series classification on three time series datasets *i.e.* **ED**, **IPD** and **UGL**. Time series in these datasets are associated with labels.

For each dataset, we employ 20% times series for training and other 80% for testing. DTW is shown to achieve good accuracy for time series classification. Following the settings in [14], we utilize 1-NN classifier to generate the labels based on the constrained DTW(cDTW) and set the best warpping window as 14%, 0%(directly summarize the distances of each points) and 4% of the time series length for **ED**, **IPD** and **UGL** respectively. After the distance computation, we learn time series embeddings based on cDTW as well. Performance comparison of this experiment is shown in Section 7.2.2.

Time series clustering. To evaluate the effectiveness of NEUTS for computing pair-wise similarities, we conduct time series clustering experiments on these datasets. For the clustering algorithm, we choose three widely used clustering algorithms, *i.e.*, Hierarchical clustering, KMedoids clustering and DBSCAN. These algorithms can be divided into two groups based on whether they need the number of clusters as an input parameter. For Hierarchical clustering and KMedoids clustering which require cluster numbers as a parameter, we take the classification labels as the ground truth and set the number of classes as the cluster numbers. For DBSCAN, we change the cluster parameters and take the clusters of exact distance as the ground truth. Then, the difference between ground truth and embedding-based clusters is evaluated.

DBSCAN [17] which is a widely used density-based algorithm for time series clustering. Based on exact distances and embedding-based distances, we can obtain two set of clusters for one dataset. Then, we evaluate the difference between them under four metrics. The results of this experiments are shown in Section 7.2.3.

Efficiency studies. We evaluate the time cost of NEUTS on both top- k similarity search and time series clustering. This experiment is conducted on a machine with Inter Xeon E5 @2.20GHz CPU and Nvidia P100 GPU. The result of this experiment is detailed in Section 7.3.

Robust studies. Besides effectiveness and efficiency, we have conducted three kinds of experiments to evaluate the robustness of NEUTS. The performances of these experiments are evaluated by top- k similarity search task.

- *Parameter sensitivity.* We examine the influence of seed size, embedding size and scanning bandwidth on NEUTS. Experimental result of this part is shown in Section 7.4.
- *Zero-shot learning.* We conduct two zero-shot learning tasks. For synthetic time series experiment, we test whether NEUTS works well on an area which has no available trajectories but just the road networks. For the model adaptation experiment, we employ the well-trained model on one dataset to generate the embeddings of another dataset and evaluate the performance of these embeddings on top- K similarity search. Result of this experiment is detailed in Section 7.5.
- *Case studies.* In addition, based on the well-trained model on small Proto dataset, we conduct a case study of entire Proto dataset (reported in Section 7.6) to show the performance of NEUTS on large dataset.

7.1.4 Compared Methods

For the studied four measures, we compare NEUTS with seven baselines in top- k similarity search task, which can be roughly divided into three categories:

- *Approximate algorithms*: Except ERP which has no approximate algorithm, each of the three measures has several approximate algorithms to fast compute them. We compare with the approximate algorithms from [15] which is used for computing both Fréchet and DTW distances, and [5] which is used for Hausdorff distance. We call these algorithms as **AP** in general for all the distance measures.
- *Siamese Network*: This category is a metric learning approach based on the Siamese network. We instantiate the Siamese network with both LSTM and GRU backbones, and denote them as **Siamese (LSTM)** and **Siamese (GRU)**, respectively.
- *Ablations*: Finally, we include two kinds of ablations of NEUTS. (1) The weight sampling in NEUTS is replaced by random sampling to test the effectiveness of distance-weighted ranking loss. We denote these two variants as **NT-No-WS (LSTM)** and **NT-No-WS (GRU)**, respectively. (2) We replace the LAM units with GRU and LSTM, denoted as **NT-No-LAM (GRU)** and **NT-No-LAM (LSTM)**, to test the effects of the proposed local attention memory mechanism.

For time series classification, we classify the time series with 1-NN classifier under cDTW distance and embedding-based distance. cDTW with Sakoe-Chiba warpping band is shown to perform well on classifying [16]. Under the supervision of cDTW, we train NEUTS to generate the embeddings and compare the classification performance of using embeddings with using cDTW in 1-NN classifiers.

For time series clustering, we generate the clustering results based on both ground truth distances and embedding-based distances, and compare the differences between them. The intention of trajectory clustering experiments is to evaluate how the learned embedding-based distances approximate to the exact distances. For Hierarchical clustering and KMedoids clustering, we directly set the number of classes as the cluster numbers and compare the similarities between clusters with the classification labels. For DBSCAN, we gradually increase the minimum distance within the cluster of DBSCAN and evaluate the similarity of clusters. If the clusters are similar consistently, we can conclude that the learned embedding-based distances are similar to the exact distances.

7.1.5 Evaluation Metrics

For top- k similarity search, we use three different metrics for performance evaluation. The first is the top- k hitting ratio, which examines the overlap percentage of the top- k results and the ground truth. We report the hitting ratio for both top-10 (HR@10) and top-50 searches (HR@50). The second is the top-50 recall for the top-10 ground truth (R10@50). This one evaluates how many of top 10 ground-truth time series are recovered by the top 50 lists produced by different methods. The third metric is the distortion of average distance for the top-10 results, denoted as δ_{H10} and δ_{R10} . δ_{H10} directly gets the top-10 time series from the

results and δ_{R10} get the top-10 most similar time series from top-50 results. They measure the distortion of average exact distances between the query time series between the top-10 search results and the average exact distance. The smaller the distance is, the stronger a method performs. Noted that this metric is meaningful only in trajectory dataset.

For time series classification, we employ three metrics to evaluate the classification performance of the learned embeddings. The first one is overall accuracy, which tests the correctness of the labels by considering all classes equally. The second one is Kappa value, which evaluates the classification performance by considering the imbalance of classes. The third one is the coverage rate, which counts the percentage of times series having the same labels with cDTW and the learned embeddings.

In time series clustering experiment, we evaluate the difference between the clustering results and the ground truth by five commonly used metrics:

- **Cluster Numbers**. We compare the cluster numbers with the changing of hyper-parameters of DBSCAN.
- **Homogeneity**. It measures that each cluster contains only time series of a single ground truth class.
- **Completeness**. It measures that all time series of a given ground truth are assigned to the same cluster.
- **V-measure** [33]. It is the harmonic mean between homogeneity and completeness, *i.e.*,

$$v = \frac{(1 + \beta) * \text{homogeneity} * \text{completeness}}{\beta * \text{homogeneity} + \text{completeness}}$$

- **Adjusted Random Index**. The random index computes a similarity measure between two clusters by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and ground truth clusterings. Based on the random index, Adjusted Random Index(ARI) can be calculate following this equation:

$$ARI = \frac{RI - Expected_{RI}}{max(RI) - Expected_{RI}}$$

7.1.6 Parameter Settings

The key parameters in NEUTS include: (1) the embedding dimension d ; (2) the scan width w of the attention memory reader. To moderate the workload of parameter tuning, we tune the parameters on Porto and directly use them on other four datasets. We have tuned d by the grid search in range {16, 32, 64, 128, 256}. In general, the performance increases with the d and gradually stabilizes when it is large enough. For w , we tuned it in {0, 1, 2, 3, 4} and found that it had an optimal value as $w = 2$ on both datasets. Finally, we set $d = 128$ and $w = 2$. In addition, we set the batch sizes as 20 and the sampling size n as 10. For the compared methods, we tuned their parameters to obtain the best performance in our datasets. We will also report the parameter study results shortly in Section 7.4. For the trajectory classification experiment, we choose the best wrapping window reported in [13].

7.2 Performance Comparison

7.2.1 Top- k Similarity Search

Table 2 shows the performance of different methods for the top- k similarity search task on five datasets. As shown,

TABLE 2

Performance comparison of five datasets for different methods on four measures. HR is the hitting ratio; R10@50 is the top-50 recall for the top-10 ground truth; δ_{H10} is the distortion of average distance on the top 10 results; δ_{R10} is the distortion of average distance on the top 10 recall in top 50 result. Both δ_{H10} and δ_{R10} are only available on the two trajectory datasets. The ground truth of top-10 average distance of Fréchet and Hausdorff distances are: 1044m and 730m(Geolife); 935m and 679m(Porto); the element unit of $\delta_{H10}/\delta_{R10}$ is meters. The bold values are the best performance of the batch experiments.

		Fréchet				Hausdorff				ERP			DTW		
Data	Method	HR@10	HR@50	R10@50	$\delta_{H10}/\delta_{R10}$	HR@10	HR@50	R10@50	$\delta_{H10}/\delta_{R10}$	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Results on One Dimensional Datasets															
ED	AP	0.2770	0.2731	0.5683	—	0.2771	0.2832	0.4750	—	—	—	—	0.4670	0.4788	0.7090
	Siamese(LSTM)	0.4610	0.5181	0.5842	—	0.2531	0.3166	0.4190	—	0.4013	0.4103	0.5359	0.5180	0.5521	0.7140
	Siamese(GRU)	0.4781	0.5352	0.6109	—	0.2506	0.3215	0.4212	—	0.3974	0.4126	0.5336	0.5328	0.5871	0.7343
	NEUTS (LSTM)	0.5002	0.5723	0.6781	—	0.2863	0.3800	0.4687	—	0.4263	0.4187	0.5498	0.5477	0.6037	0.7996
	NEUTS (GRU)	0.5017	0.6002	0.7088	—	0.2938	0.3920	0.4864	—	0.4131	0.4147	0.5571	0.5343	0.5944	0.7843
IPD	AP	0.2820	0.3111	0.5933	—	0.2950	0.2983	0.5220	—	—	—	—	0.3880	0.3942	0.7137
	Siamese(LSTM)	0.4040	0.5537	0.7901	—	0.1798	0.3200	0.3884	—	0.6938	0.8126	0.9673	0.4923	0.6421	0.8690
	Siamese(GRU)	0.4029	0.5589	0.7862	—	0.1831	0.3298	0.3905	—	0.7033	0.8132	0.9660	0.4970	0.6502	0.8781
	NEUTS (LSTM)	0.4336	0.5821	0.8108	—	0.1910	0.3524	0.4783	—	0.7680	0.8653	0.9897	0.5387	0.6941	0.9290
	NEUTS (GRU)	0.4443	0.6046	0.8256	—	0.1896	0.3610	0.4822	—	0.7577	0.8726	0.9992	0.5123	0.6898	0.9251
Results on Two Dimensional Datasets															
GL	AP	0.2374	0.2542	0.5290	213/87	0.2967	0.3180	0.5363	217/113	—	—	—	0.3870	0.4268	0.7139
	Siamese(LSTM)	0.4631	0.6032	0.8121	162/34	0.3120	0.4236	0.6640	199/69	0.5787	0.7363	0.8964	0.2680	0.4582	0.6172
	Siamese(GRU)	0.4720	0.6200	0.8368	141/30	0.3368	0.4532	0.7047	172/64	0.5831	0.7480	0.9182	0.2817	0.4690	0.6350
	NEUTS (LSTM)	0.4947	0.6786	0.8403	84/18	0.3691	0.4870	0.7416	152/42	0.6137	0.7780	0.9424	0.3067	0.4832	0.6513
	NEUTS (GRU)	0.5172	0.6823	0.8581	79/15	0.3860	0.5136	0.7640	147/36	0.6283	0.7810	0.9513	0.3104	0.4847	0.6581
Porto	AP	0.2542	0.2851	0.5520	208/79	0.2832	0.2966	0.5620	201/86	—	—	—	0.3798	0.4160	0.7010
	Siamese(LSTM)	0.4740	0.5802	0.7970	128/27	0.3834	0.4999	0.7760	165/48	0.4982	0.6893	0.9043	0.3832	0.4804	0.7602
	Siamese(GRU)	0.4983	0.6037	0.8122	116/21	0.4105	0.5309	0.7931	121/27	0.5130	0.7091	0.9120	0.4091	0.5134	0.7801
	NEUTS (LSTM)	0.5225	0.6351	0.8292	89/8	0.4372	0.5714	0.8089	101/15	0.5427	0.7297	0.9277	0.4370	0.5613	0.8396
	NEUTS (GRU)	0.5391	0.6542	0.8402	84/7	0.4532	0.5951	0.8150	97/13	0.5493	0.7439	0.9304	0.4330	0.5527	0.8549
Results on Three Dimensional Datasets															
UGL	AP	0.2431	0.2486	0.5187	—	0.2760	0.2881	0.4963	—	—	—	—	0.3591	0.3760	0.6431
	Siamese(LSTM)	0.4388	0.5323	0.7681	—	0.1466	0.2133	0.3799	—	0.5889	0.6736	0.8947	0.1280	0.1582	0.3120
	Siamese(GRU)	0.4310	0.5428	0.7726	—	0.1471	0.2149	0.3683	—	0.5914	0.6958	0.9173	0.1189	0.1695	0.3221
	NEUTS (LSTM)	0.4693	0.5762	0.8108	—	0.1656	0.2290	0.4067	—	0.6189	0.7213	0.9633	0.1208	0.1787	0.3198
	NEUTS (GRU)	0.4602	0.5881	0.8211	—	0.1770	0.2341	0.4102	—	0.6236	0.7230	0.9750	0.1457	0.2106	0.3482

TABLE 3

Performance Comparison of Ablation Baselines.

Data	Method	Fréchet				Hausdorff				ERP			DTW		
		HR@10	HR@50	R10@50	$\delta_{H10}/\delta_{R10}$	HR@10	HR@50	R10@50	$\delta_{H10}/\delta_{R10}$	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
GL	NT-No-WS (LSTM)	0.4736	0.6353	0.7996	139/27	0.3338	0.4393	0.6273	169/55	0.5880	0.7170	0.8686	0.2591	0.4610	0.6260
	NT-No-WS (GRU)	0.4812	0.6523	0.8181	131/25	0.3483	0.4523	0.6517	163/50	0.5983	0.7223	0.8850	0.2663	0.4731	0.6301
	NT-No-LAM (LSTM)	0.4842	0.6483	0.8198	117/23	0.3574	0.4607	0.7219	157/46	0.6090	0.7537	0.9291	0.2881	0.4792	0.6482
	NT-No-LAM (GRU)	0.4918	0.6549	0.8316	109/21	0.3514	0.4829	0.7384	159/44	0.6093	0.7652	0.9310	0.2990	0.5006	0.6522
	NEUTS (LSTM)	0.4947	0.6786	0.8403	84/18	0.3691	0.4870	0.7416	152/42	0.6137	0.7780	0.9424	0.3067	0.4832	0.6513
Porto	NEUTS (GRU)	0.5172	0.6823	0.8581	79/15	0.3860	0.5136	0.7640	147/36	0.6283	0.7810	0.9513	0.3104	0.4847	0.6581
	NT-No-WS (LSTM)	0.4990	0.5883	0.7981	102/10	0.4190	0.5628	0.7909	140/33	0.5192	0.6920	0.8917	0.3930	0.5013	0.7919
	NT-No-WS (GRU)	0.5162	0.6189	0.8001	98/10	0.4189	0.5765	0.7917	136/30	0.5238	0.7080	0.9082	0.4136	0.5198	0.8086
	NT-No-LAM (LSTM)	0.5154	0.6121	0.8171	92/10	0.4238	0.5691	0.8033	126/16	0.5382	0.7111	0.9107	0.4238	0.5425	0.8148
	NT-No-LAM (GRU)	0.5232	0.6429	0.8226	90/9	0.4260	0.5752	0.8024	124/17	0.5391	0.7193	0.9132	0.4294	0.5532	0.8240
	NEUTS (LSTM)	0.5225	0.6351	0.8292	89/8	0.4372	0.5714	0.8089	101/15	0.5427	0.7297	0.9277	0.4370	0.5613	0.8396
	NEUTS (GRU)	0.5391	0.6542	0.8402	84/7	0.4532	0.5951	0.8150	97/13	0.5493	0.7439	0.9304	0.4330	0.5527	0.8549

NEUTS significantly outperforms all the baseline methods in most of distance metrics. Take the Fréchet distance on Porto dataset as an example. Compared with the approximate algorithms (AP), the two variants of NEUTS, *i.e.*, NEUTS (LSTM) and NEUTS (GRU) achieve more than two time higher hitting ratios, about 62% gain in R10@50, and about 60% reductions in average distance. Such huge improvements is impressive given the fact that NEUTS does not rely on any hand-crafted heuristics but learn the similarity function automatically from seed time series. The superiority of NEUTS over the Siamese network is also obvious in all the four metrics. While both methods employ neural metric learning for approximating the similarity function, NEUTS has two advantages over the Siamese network. First, the weighted sampling and ranking loss can yield more distinguishing time series and training loss compared to Siamese network. Second, the local attention

memory module models the correlation between local close time series, which is very beneficial for generating quality time series embedding.

We find the accuracy of NEUTS is inferior to the approximate methods(AP) in some experiments, such as Hausdorff distance on ItalyPowerDemand, or Hausdorff distance and DTW distance on UWaveGestureLibrary. The reason lies in two aspects: (1) The number of time series in these two dataset are limited, and thus the seed time series number is too small to train a good mapping function for generating high-quality time series embeddings. (2) Compared with Fréchet and ERP measures, Hausdorff and DTW are more sensitive to the number of seed samples. This phenomenon can be observed in Figure 8.

The results of ablation studies are shown in Table 3. Comparing NEUTS with its ablations, one can further see the effectiveness of the two major modules in NEUTS. Continue using the Fréchet distance on the Geolife dataset

TABLE 4
Time series Classification Result.

Methods	Accuracy	Kappa Value	Coverage	# of Classes
ElectricDevices(ED)				
cDTW	0.3583	0.1944	0.6034	5
NEUTS	0.3937	0.2391		
ItalyPowerDemand(IPD)				
cDTW	0.9613	0.9223	0.9825	2
NEUTS	0.9688	0.9374		
UWaveGestureLibrary(UGL)				
cDTW	0.9563	0.9500	0.9560	8
NEUTS	0.9364	0.9273		

as an example. We observe: (1) by including the LAM module, NEUTS improves HR@10 of NT-NO-WS from 0.47 to 0.48; and (2) by including the weighted sampling and optimization module, NEUTS improves the HR@10 of NT-NO-WS from 0.48 to 0.51. The trends are similar for the Porto dataset and other three similarity measures.

7.2.2 Time Series Classification

In this experiment, we evaluate the effectiveness of the learned embeddings for time series classification on three time series datasets *i.e.* **ED**, **IPD** and **UGL**. 1-NN classifier is utilized to obtain the time series labels. To be fair, we also use the same 20% percentage of time series to train NEUTS as the training set of the classifier. The results of this experiment are shown in Table 4.

As shown, the accuracies of NEUTS are comparable with cDTW on the three datasets. It shows that the learned embeddings can recover the metric space well under the supervision of seeds similarities. Moreover, the coverage rates of classification results are over 0.6. It means that the majority labels of time series are the same when using the learned embedding as the distances for classification instead of cDTW.

Among the three datasets, the result of **ED** is worse than others. The accuracy of **ED** is less than 0.4. It is because time series in **ED** are quite similar. The points in a **ED** time series usually have the same value *i.e.*, [-0.19025, ..., -0.17345, -0.19025, ..., -0.19025] and the same values in different time series are also similar. cDTW would be dominated by the alignment distances of these same value points. This is the reason why cDTW performs worse for classifying time series in **ED**. Surprisingly, NEUTS has better performance than cDTW even though it is trained under the supervision of cDTW. The reason may lie in the structure of NEUTS. Due to the recurrent nets and the locality memory, NEUTS can not only catch the relationship across different time series but also capture the intrinsic characters of time series itself.

7.2.3 Time Series Clustering

In this experiment, we aim at exploring the effectiveness of NEUTS for computing pair-wise similarities via time series clustering. For concision, we only show the result of Fréchet distance based clusters on four datasets.

Result of Hierarchical clustering and KMedoids clustering. The two algorithms take the number of clusters as an input. We use the number of classes in the classification task as the cluster numbers. The results are shown in Table 5. For each dataset, the results are reported in three settings:

- 1) Label v.s. Embedding(L v.s. E). We take the classification labels as ground truth and measure the difference be-

TABLE 5
Time series Clustering Result of Hierarchical clustering and KMedoids clustering.

Dataset	Setting	Homo	Comp	V-measure	ARI
Hierarchical Clustering					
ED	L v.s. E	0.0150	0.0361	0.0212	0.0009
	L v.s. F	0.0182	0.0335	0.0236	0.0073
	F v.s. E	0.4201	0.5480	0.4756	0.5850
IPD	L v.s. E	0.0730	0.1545	0.0991	0.0363
	L v.s. F	0.0503	0.1660	0.0772	0.0144
	F v.s. E	0.6680	0.4289	0.5224	0.6368
UGL	L v.s. E	0.4465	0.5354	0.4869	0.3554
	L v.s. F	0.3572	0.4448	0.3962	0.2616
	F v.s. E	0.3645	0.3509	0.3576	0.2432
KMedoids Clustering					
ED	L v.s. E	0.0496	0.0489	0.0493	0.0329
	L v.s. F	0.0447	0.0438	0.0442	0.0300
	F v.s. E	0.6883	0.6915	0.6899	0.6844
IPD	L v.s. E	0.0111	0.0128	0.0119	0.0135
	L v.s. F	0.0004	0.0004	0.0004	-0.0004
	F v.s. E	0.0212	0.0243	0.0226	0.0322
UGL	L v.s. E	0.5225	0.5496	0.5357	0.4367
	L v.s. F	0.2543	0.2807	0.2668	0.2118
	F v.s. E	0.2873	0.2738	0.2804	0.2792

tween embedding-based clusters and the classification labels.

- 2) Label v.s. Fréchet distance(L v.s. F). We take the classification labels as ground truth and measure the difference between the clusters of Fréchet distance and the classification labels.
- 3) Fréchet distance v.s. Embedding (F v.s. E). We take the clusters of Fréchet distance as ground truth and measure the difference between the embedding-based clusters and the Fréchet distance-based clusters.

Compared the results of L v.s. E with L v.s. F, we can easily find that the values of the four metrics are close in both Hierarchical clustering and KMedoids clustering. It indicates that the embedding-based clusters and exact distance-based clusters have similar performance. We note that the metric values on **ED** and **IPD** are relatively small. It is because the pairwise distances (both embedding-based and exact distances) are not enough to figure out the classes without training data.

For evaluating how the embedding based distances approximate to the distance measure, we also test the difference between the embedding-based clusters and the exact distance-based clusters, *i.e.*, F v.s. E. The metric values in this experiment are quite high. It means that the learned embeddings can generate similar clusters comparing to the exact distance. To further explore this similarity, we conduct DBSCAN clustering experiment.

Result of DBSCAN. DBSCAN algorithm has two key parameters: (1) ϵ , the minimum distance within a cluster; (2) m , the minimum number of samples that a cluster contains. For each dataset, we fix m and obtain two sets of cluster results with the increase of ϵ based on exact distances and embedding-based distances, respectively. Then, we measure the difference between these two sets of results on several clustering metrics.

Because the data distributions and sample numbers of these datasets vary a lot, we need to set different parameters(m and ϵ) for these datasets. The parameters for the tested datasets are shown in the axis labels of Figure 6. Taking ElectricDevices as an example, the minimum distance ϵ varies from 0.01 to 0.39 with increase interval 0.02

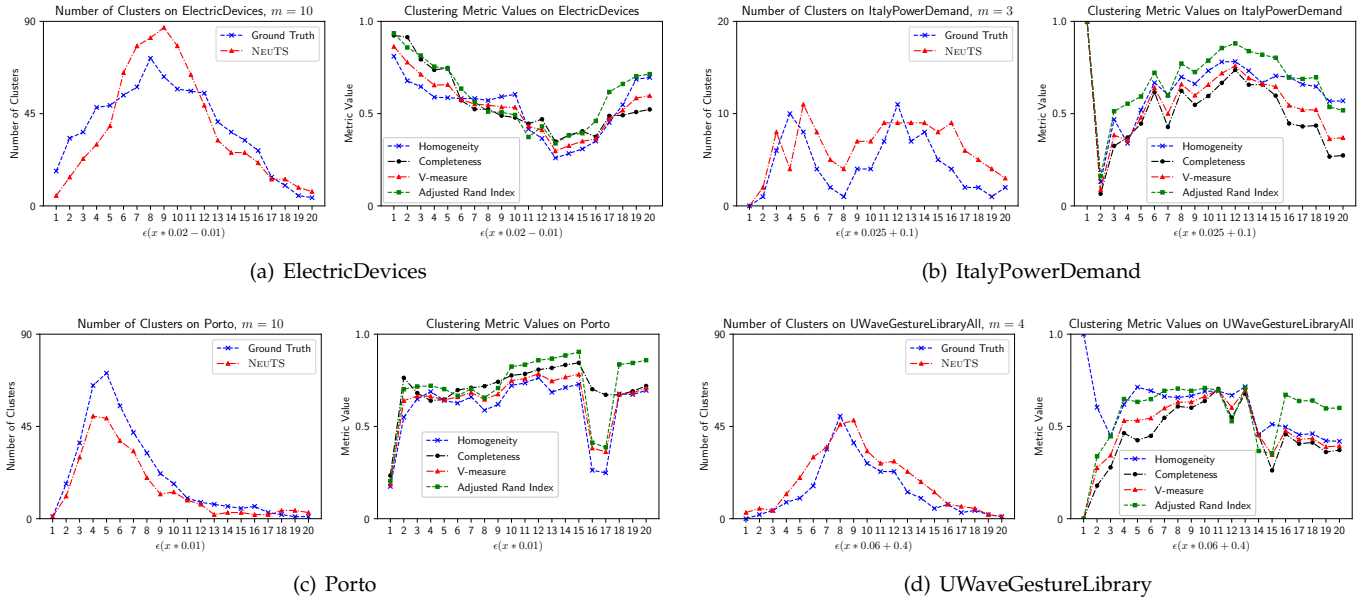


Fig. 6. Time series Clustering Result of DBSCAN.

TABLE 6
Time cost for offline model training.

Methods	Model Train			Embed 200k
	t_{epoch}	$\#_{epoch}$	t_{total}	
Siamese(LSTM)	164s	71	11644s	411s
Siamese(GRU)	153s	65	9945s	386s
NEUTS (LSTM)	285s	15	5130s	639s
NEUTS (GRU)	276s	14	3864s	483s
Ablations Experiments				
NT-NO-LAM (LSTM)	168s	15	2520s	412s
NT-NO-LAM (GRU)	159s	13	2067s	384s
NT-NO-WS (LSTM)	283s	20	5660s	636s
NT-NO-WS (GRU)	271s	18	4878s	496s

and the minimum samples number $m = 10$.

As shown in Figure 6, fixing the minimum points m and increasing the value of ϵ , the cluster numbers of NEUTS are similarly changing as its of ground truth in four datasets. Most values of the evaluation metrics are more than 0.5, which indicates that embedding-based distances generating by NEUTS are approximate to the exact distances. We note that NEUTS does not perform well in some parameter settings, such as $\epsilon = 0.16$ of Porto dataset and $\epsilon = 0.15$ of UWaveGestureLibrary dataset. The reason lies in the distance weighted sampling and the optimization strategy. For an anchor time series, we focus on the top similar and dissimilar time series pairs and assign higher weights on them in the optimization procedure. It may lead to poor performance when ϵ falls in between. Similar trends are observed on other three distance measures(DTW, Hausdorff and ERP).

7.3 Efficiency Study

In this subsection, we study the efficiency of NEUTS. We first report its offline time cost for training and embedding. Then, we report the time cost for online top- k similarity search and time series clustering with learned embeddings. Due to the space limitation, we only show the result under Fréchet distance. This result is similar for other similarity measures and other datasets.

7.3.1 Time Cost of Offline Training and Embedding

Offline Training Time. The offline training time of NEUTS consists of two parts *i.e.* the seeds similarities computation and model training. For the first part, it takes us about 6.5 hours and 10.7 hours to compute the pair-wise similarities of **Geolife** and **Porto** on a high end server with two Intel Xeon E7 CPUs respectively. For the second part, Table 6 only shows the comparison of offline model training time on the Porto dataset under Fréchet distance. For 2,000 training time series, NEUTS converges within 20 epochs. The training time of one epoch is around 5 min for NEUTS, and thus the entire training time of NEUTS is less than 2 hours. As a comparison, the Siamese network takes more than 60 epochs to converge, which is about 3X slower than NEUTS. We also compare the convergence rate of NEUTS and NT-NO-LAM in Figure 7 and observe that: (1)NEUTS has higher convergence rate than NT-NO-LAM because LAM module captures useful information from processed time series; (2)the GRU version of NEUTS is stronger than the LSTM version. The reason is GRU-based NEUTS has less parameters than LSTM, which tends to achieve better convergence.

Offline Embedding Time. Another part of offline training is the embedding procedure which generates the embeddings of time series by using the well-trained model. In this experiment, we set the embedding batch size as 2000 and report the time cost of embedding 200k time series. The results of all neural network based methods are shown in Table 6. We can easily find that LAM units based methods(NEUTS, NT-NO-WS) are little slower than standard units based method(NT-NO-LAM, Siamese). The reason is LAM units need more calculation for finding useful information in memory tensor. For methods using the same kind of units, GRU based methods are faster than LSTM methods because GRU has fewer gates than LSTM and needs fewer matrix multiply operations.

TABLE 7
Time cost for online similarity search without index.

Method	1k	5k	10k	200k
Fréchet				
BruteForce	8.712s	41.876s	84.480s	1639.834s
AP	1.840s	11.319s	23.107s	532.652s
NT-NO-LAM (LSTM)	0.461s	0.471s	0.489s	1.576s
NT-NO-LAM (GRU)	0.432s	0.452s	0.477s	1.554s
NEUTS (LSTM)	0.461s	0.470s	0.490s	1.574s
NEUTS (GRU)	0.432s	0.453s	0.476s	1.553s
Hausdorff				
BruteForce	0.238s	1.416s	2.981s	51.642s
AP	0.127s	0.154s	0.179s	3.426s
NT-NO-LAM (LSTM)	0.026s	0.046	0.072s	1.133s
NT-NO-LAM (GRU)	0.021s	0.041	0.065s	1.128s
NEUTS (LSTM)	0.024s	0.047	0.073s	1.131s
NEUTS (GRU)	0.019s	0.041	0.064s	1.125s
ERP				
BruteForce	0.409s	1.982s	3.807s	73.054s
NT-NO-LAM (LSTM)	0.027s	0.046s	0.081s	1.154s
NT-NO-LAM (GRU)	0.025s	0.039s	0.072s	1.148s
NEUTS (LSTM)	0.026s	0.047s	0.081s	1.152s
NEUTS (GRU)	0.026s	0.040s	0.071s	1.147s
DTW				
BruteForce	0.305s	1.482s	3.070s	59.054s
AP	0.119s	0.142s	0.185s	4.021s
NT-NO-LAM (LSTM)	0.023s	0.044s	0.066s	1.028s
NT-NO-LAM (GRU)	0.019s	0.040s	0.059s	1.024s
NEUTS (LSTM)	0.021s	0.043s	0.067s	1.027s
NEUTS (GRU)	0.019s	0.038s	0.060s	1.023s

7.3.2 Time Cost of Online Similarity Search

Table 7 shows the time cost of NEUTS for performing top- k similarity search with different sizes. Specifically, from the test set of Porto, we randomly sample four sub-corpora with sizes 1K, 5K, 10K, and 200K respectively. Then we use NEUTS to perform top-50 similarity for each time series and re-rank the 50 time series by calculating their accurate distance. Table 7 reports the average time cost for processing one query. We compare it with the BruteForce method that directly computes the exact distances following the definition, the approximate algorithms (AP), as well as the neural network based methods (NT-NO-LAM). We omit the results of NT-NO-WS and Siamese methods because the time cost of NT-NO-WS and Siamese methods are analogous to NEUTS and NT-NO-LAM in online search procedure. As shown in Table 7, for all the four measures, NEUTS provides 50x-1000x speedup over BruteForce and 3x-350x speedup over existing approximate algorithms. The speedup ratios are especially significant for large datasets. The time cost of ablation methods NT-NO-LAM are very close to NEUTS because the embedding time difference of one search time series is small. Between the two variants of NEUTS, the GRU version is faster than the LSTM one by involving fewer gates.

TABLE 8
Time cost for time series clustering.

Exact Distance based Clustering					
Datasets	# t_s	Embed	Dist	Cluster	Total
ElectricDevices	8.1K	—	5830s	0.70s	5830s
ItalyPowerDemand	1K	—	623s	0.02s	623s
Porto	10K	—	7238s	0.71s	7238s
UWaveGestureLibrary	4.4K	—	5321s	0.28s	5321s
Embedding Distance based Clustering					
ElectricDevices	8.1K	16s	3.04s	0.69s	20s
ItalyPowerDemand	1K	3s	0.04s	0.02s	3s
Porto	10K	24s	2.96s	0.73s	27s
UWaveGestureLibrary	4.4K	27s	0.23s	0.22s	28s

TABLE 9
Time cost for time series classification.

Dataset	BruteForce	UCR Suite	NEUTS	# $_{test}$	# $_{train}$
ED	28.4531s	14.8892s	0.8242s	6480	1620
IPD	5.3247s	0.3735s	0.1373s	800	200
UGL	69.4923s	29.0565s	0.6355s	3520	880

7.3.3 Time Cost of Time Series Clustering

In this experiment, we evaluate the time cost of time series clustering. Table 7 reports the average runtime of exact distance based clustering and embedding-based clustering. The runtime of exact distance based clustering consists of two parts: clustering time (denoted by cluster in Table 7) and pair-wise distance computation time (Dist). Apart from the two parts, the runtime of embedding-based clustering includes the time of embedding generation (Embed). Due to the high complexity of exact distance based clustering algorithms, we employ 20 threads to compute pair-wise Fréchet distance of these datasets. In contrast, we use only one thread to compute the embedding based distance. With the setting favoring the exact distance based clustering, embedding distance base clustering still achieves over 100x acceleration comparing to the exact distance based methods.

7.3.4 Time Cost of Time Series Classification

In this experiment, we evaluate the time cost of time series classification. Time series are classified by a 1-NN classifier that uses cDTW or NeuTS to find the most similar time series. The cost mainly lies in querying the time series. We compare three query methods *i.e.* BruteForce, UCR Suite and NEUTS. For BruteForce, we compute cDTW with the query time series and all other time series and select the most similar one. For UCR Suite [32], we extend it to 1-NN time series classification. It employs three bounding techniques *i.e.* LB_{Kim} , LB_{Keogh} and LB_{Keogh-} , to prune irrelevant time series. For NEUTS, we first embed the query time series into an embedding vector and then find the nearest time series in embedding space. NEUTS is an approximate method to cDTW, but its accuracy is comparable to cDTW (as reported in Section 7.2.2). We report the efficiency of the time series classification in Table 9

UCR Suite reduces the time cost of BruteForce significantly. Owing to the three lower bounds, most of time series are pruned with little computation cost. However, it is still slow when the time series dataset is large. NEUTS is much more efficient than UCR Suite. It can achieve about 3-40 times speedup over UCR Suite. The reason is that operations in NEUTS are all based on the embedding vectors. We note that NEUTS introduces the model training and time series similarity computation costs (detailed in Section 7.3.1), which could be done offline, while 1-NN classification based on cDTW does not need a training. Moreover, NEUTS is an approximate method and has no theoretical guarantee. It will be interesting future work to evaluate the performance of NeuTS for 1-NN classification on more time series datasets.

7.4 Parameter Sensitivity Study (RQ4)

In this subsection, we evaluate the model sensitivity on three parameters: the seed size, the scan width w , and the embedding dimension d . Due to the space limitation, we only shows the experimental result on the Porto dataset.

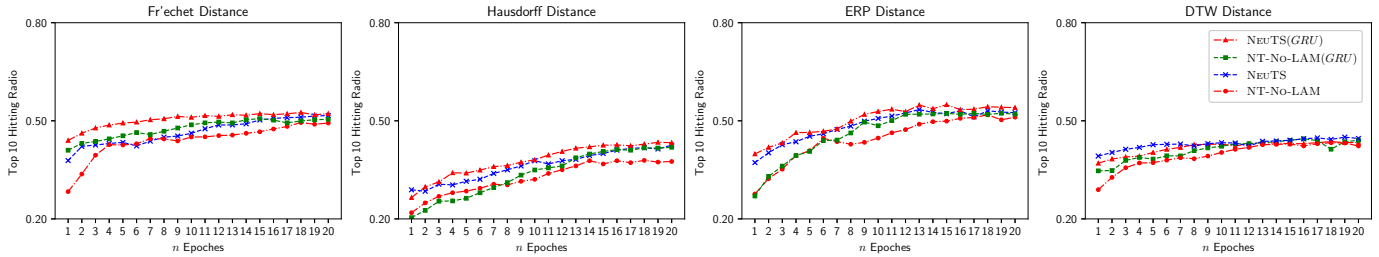


Fig. 7. The convergence curve of NEUTS and NT-NO-LAM on Fréchet, Hausdorff, ERP and DTW with respect to 20 epochs.

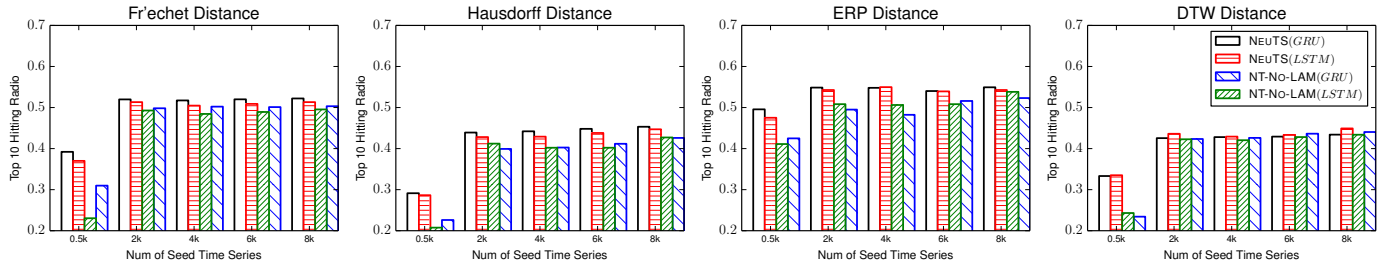


Fig. 8. HR@10 of NEUTS and NT-NO-LAM on Fréchet, Hausdorff and DTW with varying seed size.

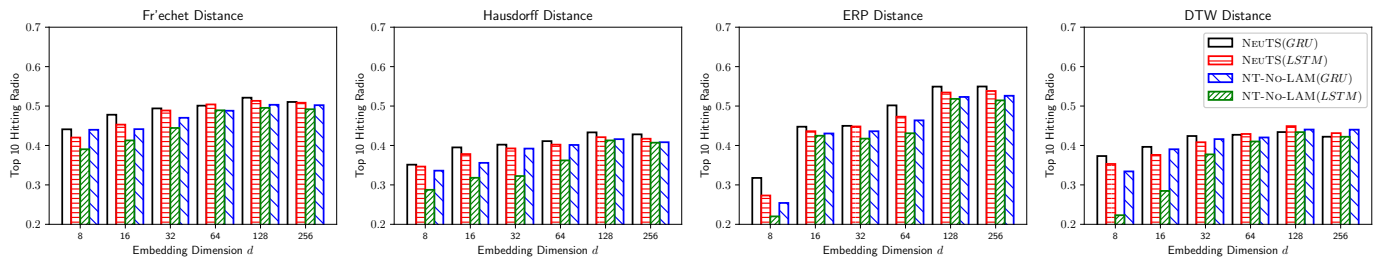


Fig. 9. HR@10 of NEUTS and NT-NO-LAM on Fréchet, Hausdorff and DTW with varying embedding size d .

7.4.1 The effect of seed size

We first study the effect of the number of seed time series on the accuracy of NEUTS. Figure 8 shows the results of NEUTS and its ablation NT-NO-LAM for the four measures as the seed size varies from 500 to 8000 on Porto. As shown, the accuracy of NEUTS becomes relatively stable when there are more than 2000 training time series. Comparing the performance across different methods, LAM-based models outperform the classic LSTM and GRU models. Another interesting observation is that NEUTS is more robust than NT-NO-LAM with sparse seeds. As shown, when the number of training time series is only 500, the performance gaps between NEUTS and NT-NO-LAM are particularly large. This is because LAM employs a memory tensor to memorize useful information from processed time series.

7.4.2 The effect of embedding dimension d

We proceed to study the effect of the embedding dimension d on the performance of NEUTS. Figure 9 illustrates HR@10 as d varies from 8 to 256. As shown, the performance of NEUTS and its variants first increases and then drops slightly. The reason is the parameter d controls the complexity of NEUTS. When d increases, the model enjoys more expressive power to capture the intrinsic structures of time series. However, when d is too large, the model will suffer from over-fitting due to the limited size of seed.

7.4.3 The effect of scan width w

Finally, the scan width w in the LAM module is a key parameter that controls the exploration spread for historical

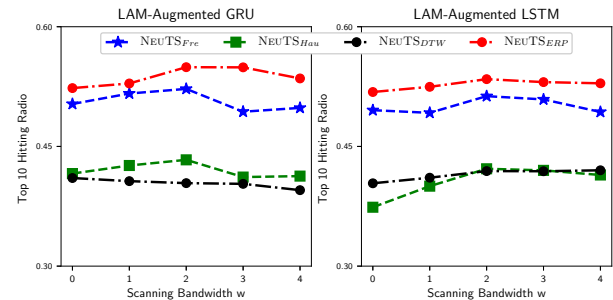


Fig. 10. HR@10 of NEUTS with varying w .

time series. As shown in Figure 10, with the increase of w , the HR@10 for all methods first increases and then slight drops. This phenomenon is attributed to two reasons: (1) as w increases, more information tends to be accessed by the LAM reader, which is useful for encoding the current time series in the initial stage; (2) when w is too large, the information of some non-relevant time series will be inevitably incorporated. Even if the attention mechanism in NEUTS can help reduce this effect, the performance of the model can still be harmed.

7.5 Evaluation on Zero-Shot Learning (RQ5)

This set of experiments is to study the performance of NEUTS for zero-shot learning scenarios. Specifically, the NEUTS model relies on a real time series database and sampling seeds from the database as guidance. It is interesting to examine the question: how does NEUTS perform

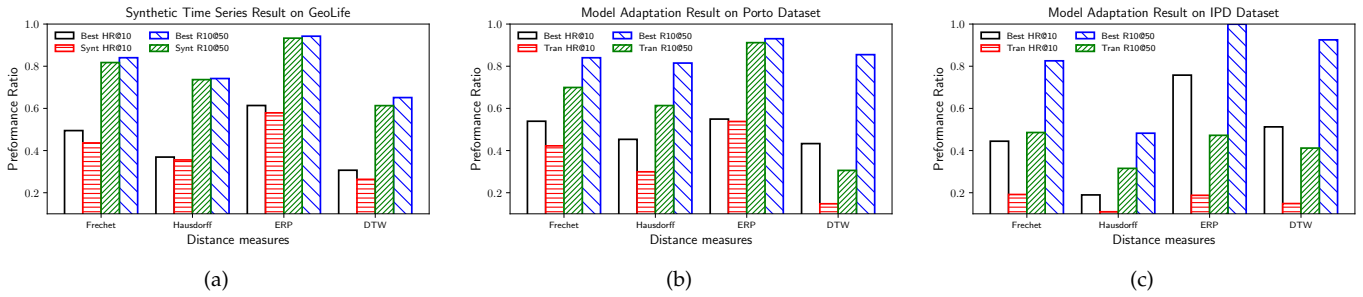


Fig. 11. Illustration of zero-shot learning results on Geolife dataset. *Best* is the best performance NEUTS can achieve on real dataset.

if there are no real seed time series for training NEUTS? For this purpose, we extend NEUTS for the zero-shot learning scenario and design two tasks to evaluate its robustness.

7.5.1 Synthetic Time Series Experiment

This experiment is based on 2D time series. We assume no time series databases are given and there exists only a road network of the target city. Then we generate a bunch of simulated time series as our seeds and train the NEUTS model for computing the similarity for a pair of real time series. Based on the road network in Beijing [42], we generate 6,000 synthetic time series by employing random walk on road node graph and interpolating coordinates between the nodes. Then we take the synthetic time series as the seeds to train NEUTS, and test it with the real time series from Geolife. Figure 7.5(a) shows the HR@10 and R10@50 of NEUTS.

Impressively, even using synthetic time series and their similarities as guidance, NEUTS can still achieve around 0.8 recall for all the four metrics. Such a phenomenon indicates that NEUTS can be applied to scenarios when no real time series databases are available, but similar synthetic data can be generated. Due to the road constraints, synthetic time series of moving objects are easy to obtain. But it is not so convenient to generate synthetic data for general time series. For these time series, NEUTS can not be applied directly.

7.5.2 Model Adaptation Experiment

In this experiment, we first train NEUTS utilizing the similarities of seeds sampled from one dataset, such as **Geolife** and **ED**. With the well-trained model, we directly generate the embeddings of time series in other datasets *i.e.*, **Porto** and **IPD**, respectively. The results are illustrated in Figure 11(b) and Figure 11(c).

The adaptation performances on both datasets are much inferior to the best ones. It demonstrates NEUTS is sensitive to the training data. If the distribution of training data is similar to the test data, NEUTS will perform well. As shown, the adaptation performances of Porto dataset are better than IPD dataset. The reason is that Geolife and Porto are trajectory datasets and they are inherently similar. On the contrary, time series in **ED** and **IPD** are different, which leads to poor performance.

This experiment exposes some limits of NEUTS and can be used to guide its applications. Ideally, if real time series are available for training, we can train NEUTS with them and obtain the best performance. If no training time series are available, we should either generate synthetic data similar

to the testing ones or leverage other datasets that are similar to the testing time series in data distribution.

7.6 Case Studies

Because computing exact pair-wise similarity is very time consuming, we cannot pre-compute all pair-wise similarities and evaluate the performance of NEUTS on entire large time series dataset. In this experiment, we use the entire Porto dataset to perform several case studies to intuitively examine the top- k search results of our model. For this purpose, we randomly choose several time series and retrieve their top-5 neighbors under the Fréchet distance. Due to the space limitation, Table 10 only shows the results of two representative time series: T_{91} and T_{65} . For each query, we plot both the top-3 ground truth time series as well as the top-3 time series retrieved by NEUTS.

The result in Table 10 shows NEUTS is quite effective on both short(T_{91}) and long(T_{65}) time series: the results returned by NEUTS match the ground truth very well, and the distortions of top-5 average distance δ_{H5} are very small. Moreover, by training with the weighted ranking loss, NEUTS preserves the ranking order of time series.

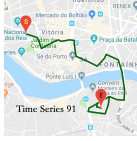
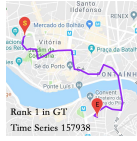
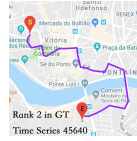
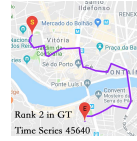
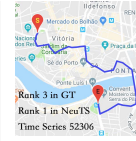
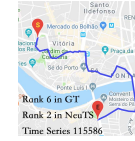

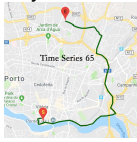


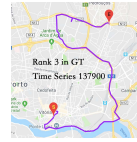



8 CONCLUSION

We proposed a seed-guided neural metric learning approach NEUTS that is fast, accurate, generic and elastic for time series similarity computation. Its novelty lies on two aspects: (1) a local attention memory (LAM) module that can augment existing RNN architectures to capture the correlations between time series; and (2) a distance-weighted ranking loss that effectively leverages seed information to learn quality time series embeddings. Experiments on two real-life datasets have shown that NEUTS can effectively accelerate on four distance measures(Fréchet, Hausdorff, DTW, ERP), while producing more accurate results over state-of-the-art approximate algorithms.

ACKNOWLEDGMENTS

The work was done when Di Yao visited NTU. This work is supported in part by a Singapore MOE Tier-2 grant MOE2016-T2-1-137, a MOE Tier-1 grant RG31/17, a grant from Microsoft, and the National Natural Science Foundation of China(Grant No.61472403, 61303243, 61702470). The authors would also like to thank the anonymous reviewers and associate editor for their constructive comments and suggestions.

TABLE 10
Comparison of top-3 similar time series between ground truth(GT) and NEUTS.

Result of T_{91} : HR@10: 0.7; HR@50: 0.78; H10@R50: 0.9; $\delta_{H5} = 4m$; $\delta_{H10} = 4m$; $\delta_{R10} = 2m$						
Query Time Series	Top-3 ground truth.			Top-3 of NEUTS.		
						
Result of T_{65} : HR@10: 0.4; HR@50: 0.52; H10@R50: 0.8; $\delta_{H5} = 296m$; $\delta_{H10} = 236m$; $\delta_{R10} = 62m$						
Query Time Series	Top-3 ground truth.			Top-3 of NEUTS.		
						

REFERENCES

- [1] P. K. Agarwal, K. Fox, J. Pan, and R. Ying, "Approximating dynamic time warping and edit distance for a pair of point sequences," in *SoCG'16*, 2016, pp. 6:1–6:16.
- [2] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *Int. J. Comput. Geometry Appl.*, vol. 5, pp. 75–91, 1995.
- [3] T. Ando and J. Bai, "Clustering huge number of financial time series: A panel data approach with high-dimensional predictors and factor structures," *Journal of the American Statistical Association*, vol. 112, no. 519, pp. 1182–1198, 2017.
- [4] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of vehicle trajectories," *IEEE Trans. Intelligent Transportation Systems*, vol. 11, no. 3, pp. 647–657, 2010.
- [5] A. Backurs and A. Sidiropoulos, "Constant-distortion embeddings of hausdorff metrics into constant-dimensional L_p spaces," in *APPROX/RANDOM'16*, 2016, pp. 1:1–1:15.
- [6] J. Bromley, I. Guyon, B. Hu, N. Begum, A. Bagnall, and R. Shah, "Signature verification using a siamese time delay neural network," in *NIPS'93*, 1993, pp. 737–744.
- [7] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *ICML'07*, 2007, pp. 129–136.
- [8] S. Chandar, S. Ahn, H. Larochelle, P. Vincent, G. Tesauero, and Y. Bengio, "Hierarchical memory networks," *arXiv:1605.07427*, 2016.
- [9] L. Chen and R. T. Ng, "On the marriage of l_p -norms and edit distance," in *VLDB'04*, 2004, pp. 792–803.
- [10] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD'05*, 2005, pp. 491–502.
- [11] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," July 2015.
- [12] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching trajectories by locations: an efficiency study," in *SIGMOD'10*, 2010, pp. 255–266.
- [13] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The ucr time series archive," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 6, pp. 1293–1305, 2019.
- [14] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1542–1552, 2008. [Online]. Available: <http://www.vldb.org/pvldb/vol1/1454226.pdf>
- [15] A. Driemel and F. Silvestri, "Locality-sensitive hashing of curves," in *SoCG'17*, 2017, pp. 37:1–37:16.
- [16] K. Echihiabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim, "The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art," *VLDB'18*, vol. 12, no. 2, pp. 112–127, 2018.
- [17] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD'96*, 1996, pp. 226–231.
- [18] M. Farach-Colton and P. Indyk, "Approximate nearest neighbor algorithms for hausdorff metrics via embeddings," in *FOCS'99*, 1999, pp. 171–180.
- [19] X. Gong, Y. Xiong, W. Huang, L. Chen, Q. Lu, and Y. Hu, "Fast similarity search of multi-dimensional time series via segment rotation," in *DASFAA'15*, 2015, pp. 108–124.
- [20] M. G. Gowanlock and H. Casanova, "Distance threshold similarity searches: Efficient trajectory indexing on the GPU," *IEEE TPDS.*, vol. 27, no. 9, pp. 2533–2545, 2016.
- [21] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv:1410.5401*, 2014.
- [22] X. Jin, Q. Jiang, Y. Chen, S.-J. Lee, R. Nie, S. Yao, D. Zhou, and K. He, "Similarity/dissimilarity calculation methods of dna sequences: a survey," *Journal of Molecular Graphics and Modelling*, vol. 76, pp. 342–355, 2017.
- [23] V. Kreinovich, "Arbitrary nonlinearity is sufficient to represent all functions by neural networks: A theorem," *Neural Networks*, vol. 4, no. 3, pp. 381–383, 1991.
- [24] T. Lee and S. Lee, "OMT: overlap minimizing top-down bulk loading algorithm for r-tree," in *CAISE'03*, 2003.
- [25] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *ICDE'18*, 2018.
- [26] R. Manmatha, C. Wu, A. J. Smola, and P. Krähenbühl, "Sampling matters in deep embedding learning," in *ICCV'17*, 2017, pp. 2859–2867.
- [27] B. McFee and G. R. G. Lanckriet, "Metric learning to rank," in *ICML'10*, 2010, pp. 775–782.
- [28] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Time-evolving O-D matrix estimation using high-speed GPS data streams," *Expert Syst. Appl.*, vol. 44, pp. 275–288, 2016.
- [29] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang, "Self-driving database management systems," in *CIDR*, 2017.
- [30] W. Pei, D. M. J. Tax, and L. van der Maaten, "Modeling time series similarity with siamese recurrent networks," *arXiv*, vol. abs/1603.04713, 2016.
- [31] Q. Qian, R. Jin, S. Zhu, and Y. Lin, "Fine-grained visual categorization via multi-stage metric learning," in *CVPR'15*, 2015, pp. 3716–3724.
- [32] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *KDD '12*, 2012, pp. 262–270.
- [33] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *EMNLP, J. Eisner, Ed. ACL*, 2007, pp. 410–420.
- [34] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.
- [35] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.
- [36] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis, "Trajectory similarity join in spatial networks," *PVLDB*, vol. 10, no. 11, pp. 1178–1189, 2017.
- [37] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *NIPS'15*, 2015, pp. 2440–2448.

- [38] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *arXiv*, vol. abs/1410.3916, 2014.
- [39] D. Xie, F. Li, and J. M. Phillips, "Distributed trajectory similarity search," *PVLDB'17*, vol. 10, no. 11, pp. 1478–1489, 2017.
- [40] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *ICDE'19*. IEEE, 2019, pp. 1358–1369.
- [41] B. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE'98*, 1998, pp. 201–208.
- [42] X. Zhan, S. V. Ukkusuri, and P. S. C. Rao, "Dynamics of functional failures and recovery in complex road networks," *Physical Review E*, vol. 96, no. 5, p. 052301, 2017.
- [43] 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.



Rongchang Duan He is an engineer at National Computer Network Emergency Response Technical Team/Coordination Center of China (CERT/CC). He received his Ph.D. degree in University of Chinese Academy of Sciences under the supervision of Prof. Zhongcheng Li. His research interest lies on Internet of things and deep learning.



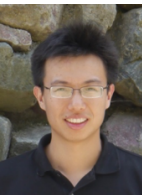
Di Yao He is an assistant professor at Institute of Computing Technology, Chinese Academy of Sciences. He received his Ph.D. degree in University of Chinese Academy of Sciences under the supervision of Prof. Jingping Bi. His research interest lies on spatio-temporal data mining and deep learning.



Gao Cong He received the PhD degree from the National University of Singapore, in 2004. He is a full professor with Nanyang Technological University, Singapore. Before he relocated to Singapore, he worked with Aalborg University, Microsoft Research Asia, and the University of Edinburgh. His current research interests include geo-textual data management and data mining.



Jingping Bi She received her Ph.D. in 2002 from the Institute of Computing Technology, Chinese Academy of Sciences. She is currently a full professor at the Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include network measurement, routing, virtualization, SDN, and big data. She is an IEEE Member.



Chao Zhang He is an assistant professor at the School of Computational Science and Engineering, Georgia Tech. Before joining Georgia Tech, he received his Ph.D. in Computer Science from UIUC under the supervision of Prof. Jiawei Han. His research include spatio-temporal data mining, indexing techniques, and web data mining.



Xuying Meng She is an assistant professor at the Institute of Computing Technology, Chinese Academy of Sciences. She received her Ph.D. degree from the University of Chinese Academy of Sciences in 2018. Her current research interest lies on data mining and security protection of network services.