A dissimilarity measure estimation for analyzing trajectory data

Reza Arfa¹, Rubiyah Yusof²*, Parvaneh Shabanzadeh³

¹²³Centre for Artificial Intelligence and Robotics, Malaysia-Japan International Institute of Technology (MJIIT), Universiti Teknologi Malaysia, 54100, Kuala Lumpur, Malaysia

*rubiyah.kl@utm.my

Received: April 1; Accepted: September 23; Published: October 4, 2019

Abstract. Quantifying dissimilarity between two trajectories is a challenging problem yet it is a fundamental task with a wide range of applications. Existing dissimilarity measures are computationally expensive to calculate. We proposed a dissimilarity measure estimate for trajectory data based on deep learning methodology. One advantage of the proposed method is that it can get executed on GPU, which can significantly reduce the execution time for processing a large number of data. The proposed network is trained using synthetic data. A trajectory simulator that generates random trajectories is proposed. We used a publicly available dataset to evaluate the proposed method for the task of trajectory clustering. Our experiments show the performance of the proposed dissimilarity estimation method is comparable with well-known methods while our method is substantially faster to compute.

Keywords: Trajectory Analysis, Dissimilarity Measure, Deep Learning, Bidirectional LSTM.

1. Introduction

With the advancement of surveillance devices and object tracking algorithms, object’s trajectory can easily and accurately be obtained. A trajectory is a sequence which is produced by tracking the location and the dynamics of a moving object. Despite its simplicity, trajectory is a powerful motion descriptor that can explain the object’s moving pattern and activity. Comparing two trajectories and quantifying their (dis)-similarity is a fundamental research problem with a wide range of applications including animals movement analysis [1], trajectory clustering [2], scene modelling [3] and trajectory retrieval [4].

Defining a function to quantify the similarity between two trajectories is a challenging problem mainly because trajectories are sequences with varying length. A great number of similarity measures have been proposed for comparing trajectory data such as Dynamic Time Warping (DTW) [5], longest common subsequences (LCSS) [6], edit distance on real se-
Even though these methods have been shown to perform well in different trajectory analysis tasks, they suffer from being computationally expensive to calculate. These methods usually are $O(L^2)$ complex, where $L$ being the maximum number of observations in trajectories to be compared [8]. Therefore, it has been argued that similarity-based trajectory analysis systems are not scalable to medium and large datasets [9].

In this paper, we propose a novel approach for quantifying the dissimilarity between two trajectories. Our method is based on deep learning methodology, where a single neural network is used to estimate different dissimilarity measures. One advantage of the proposed method is that it runs on GPU, which makes the calculation much faster by parallelizing the computation. Moreover, the proposed network can estimate different similarity measures simultaneously. Since the weights are shared in the calculation, estimating different similarity measures does not add significant computational time. To train the network we proposed a simulation-training strategy where a trajectory simulation algorithm generates trajectories to train the network. Few real-world data are then used to fine-tune the network weights.

The rest of the paper is organized as follow. In the next section, related works are presented. In section 3 we discuss the detail of the proposed method. The experimental results and discussion are presented in section 4. Finally, in section 5 we provided the conclusion and future research direction.

2. Related work

Quantifying similarity between two trajectories is a fundamental task with dozens of applications. As mentioned earlier, the main challenge when comparing two trajectories is the potential length differences between them. Many studies addressed this problem by introducing length independent similarity measures which can compare sequences with varying length. These similarity measures can be categorized into warping-based methods and shape-based methods [8].

Warping-based methods search for an optimal alignment between two trajectories that minimizes the cumulative distance between the matched points. Dynamic Time Warping (DTW) [5] is among the early warping distance used to compare trajectories. Since DTW tries to match all existing points, this method is sensitive to outliers and noisy observations. Longest common subsequences (LCSS) [6] address this problem by allowing samples to be unmatched. Similar to DTW and LCSS, Piciarelli and Foresti [10] proposed a distance accounted for temporal drift. Unlike LCSS and DTW, their distance measure can be used to compare incomplete trajectories. This particularly important for online comparison of trajectories, where trajectories are developing and still not fully observed.

Many distance measures are based on a modification of edit distance (ED). Chen et al. [11] proposed Edit Distance with Real Penalty (ERP) which allows for amplitude scale and global spatial shift. Chen et al. [7] proposed Edit Distance on Real Sequences (EDR). Similar
to LCSS, EDR uses a predefined threshold for sample matching. Compare to LCSS, EDR is invariant in both scale and global spatial shift.

Shape-based distances are another class of distance measure capable of comparing sequences with varying length. Shaped-based distances use geometric features of the trajectories to compare two sequences. Hausdorff distance is a well-known shaped-based measure. The original form of Hausdorff distance, however, is defined for comparing two unordered sets. Trajectories, on another hand, are sequences of observation where the order between observations is of great importance. Atev et al. [12] proposed a modified Hausdorff distance which takes the order between observations into the account. Their proposed distance takes the effect of outliers in the objective function as well. Alt [13] proposed a Directed Hausdorff Distance (DHD) which intuitively measures the degree to which a trajectory resembles some part of other trajectories. Laxhammar and Falkman [14] extended DHD for online trajectory comparison and anomaly detection.

Another well-known class of shape-based methods are based on Fréchet distance, known also as “walking-dog distance”. Imagine a man walking on a path while his dog on a leash walking on a separate one. The Fréchet distance is the length of shortest leash required for both to traverse their separate paths. Unlike Hausdorff distance, Fréchet distance takes the ordering of observations along the curve into the account.

An exact Fréchet distance for trajectory analysis is proposed by Alt et al. [15]. The complexity of their method, however, is $O(n^2 \log(n^2))$ which is substantially slower than warping-based methods. Eiter et al. [16] proposed a discrete Fréchet distance that approximates the distance for polygonal curves. Discrete Fréchet distance is closely related to warping-based methods. The time complexity of their model is quadratic.

Most distance measures use dynamic programming techniques at some point. This affects the overall computational cost, where the computational cost of most of these methods is at least quadratic. Being computational expensive is particularly a challenging problem when one is dealing with medium to large dataset. In most trajectory analysis systems, it is usually required to calculate the pairwise distance several times. For instance, in trajectory clustering or path modelling systems, a $N \times N$ dissimilarity matrix needs to be calculated, where $N$ is the total number of trajectories in a dataset. Similarly, in anomaly detection or trajectory retrieval systems a query trajectory needs to be compared against all previous $N$ trajectories.

Some studies have suggested using Graphic Processing Units (GPUs) to speed up these methods by parallelizing the calculation [17-20]. Since the pairwise distances which are used to compare trajectories involve recursive calculation and dynamic programming, the parallelization degree is very limited [19, 21]. Therefore, current GPU solutions do not address the scaling problem of distance-based methods to larger datasets [22].

In terms of performance, each dissimilarity measure has its advantages where it outper-
forms the others in a certain scenario. There have been some studies comparing these distances against different datasets [23, 24]. The accuracy of the similarity measures, however, are often task and dataset dependent. A fusion of different similarity measures often produces the best result [25-27]. Combining similarity measures, however, requires more calculation which increases the execution time.

In this study, we proposed to use deep learning methodology to estimate pairwise distances. We used Long Short-Term Memory (LSTM) to account for potentially different trajectory length. Given two trajectories, the goal is to find a nonlinear estimate for the true dissimilarity measures. To train the network, we used synthetic trajectories generated from our trajectory simulation algorithm. The pairwise dissimilarity between these synthetic trajectories is calculated and given to the network as the target values.

3. Methodology

Consider two trajectories $T_x = \langle r_{x,0}, ..., r_{x,|x|} \rangle$ and $T_y = \langle r_{y,0}, ..., r_{y,|y|} \rangle$ where $r_{x,i}$ and $r_{y,j}$ denoting $i$th and $j$th observation and $|x|$ and $|y|$ denoting the length of $T_x$ and $T_y$ respectively. The problem of dissimilarity measure is to define a function $d(T_x, T_y) \in \mathbb{R}$, where it satisfies $d(T_x, T_x) = 0$, $d(T_y, T_x) = d(T_x, T_y)$ [8]. Intuitively, this function produces a small value when two trajectories are similar. As two trajectories become more dissimilar, $d(T_x, T_y)$ produces larger values.

Our approach is based on estimating available dissimilarity measures using deep learning representation. In other words, instead of defining a mathematical function, $d(T_x, T_y)$, a neural network learns a non-linear model, $d_{\text{model}}(T_x, T_y)$, that estimates mathematical dissimilarity measures. This mathematical dissimilarity measure can be any trajectory measure discussed in section 2 including DTW, LCSS, and MoH.

Figure 1 presents an overview of the proposed framework for training and testing phases. Two raw trajectories are first preprocessed to ensure that the length of trajectories is less than a predefined size, $\ell_{\text{max}}$. If the length of a trajectory is longer than $\ell_{\text{max}}$, observations will be uniformly removed until the trajectories length is less than $\ell_{\text{max}}$. Before trajectories are passed into the network, they are normalized by zero-padding technique [28]. This technique extends a trajectory length by simply concatenating 0 to the end of the trajectory. A neural network takes two normalized trajectories and returns the dissimilarity between them. In the training phase, the dissimilarity between two trajectories is calculated and used as the target value.
3.1. Trajectory simulation

Trajectories obtained from different traffic scenarios can be substantially different from each other. There are many factors that affect this variation including different movement pattern, video sampling rate, camera setup, etc. Obtaining real-world data that cover all the different scenarios is a hard task. Training a neural network, however, requires a large number of data. Instead of collecting real-world data from different sources that cover all available scenarios, we use randomly simulated trajectories to train our network.

To generate random trajectories, we divided the scene into $M \times Q$ cells. When an object enters a grid cell, $C_{ij}$, that grid affects the flow of the object by manipulating the object’s directional speed. Figure 2 illustrates an example of a trajectory of an object arriving at a grid cell $C_{ij}$ from the right grid cell.

Let the control flow for the $k^{th}$ object in the grid $C_{ij}$ to be denoted as $U_{ijk}$. This vector encodes both directional component and velocity. Let directional component and velocity to be $v_{ijk}$ and $v_{ijk}$ respectively. Although $v_{ijk}$ can be of any arbitrary direction, for simplicity we quantized this value into four directions: north (N), south (S), west (W), and east (E). For an object $k$ entering the grid $C_{ij}$, the control flow is randomly generated as follow:

$$U_{ijk} \sim \mathcal{N}(v_{ijk}|\mu_v, \sigma_v, C_{ij}, k) \times P(v_{ijk}|V, C_{ij}, k)$$  \hspace{1cm} (1)
where $\mu_v$ and $\sigma_v$ are the average and standard deviation of velocity. The second term of the right-hand side generates a random direction, $v_{ij}$. In the case of discrete directions, this term is reduced to categorical distribution and the variable $v_{ij}$ can take one of the quantized directions given the probability distribution for different directions. In our case where we used four directions, the event probabilities in cell $C_{ij}$ is a four-dimensional probability simplex defined by $p_{ij}^E \geq 0, p_{ij}^W \geq 0, p_{ij}^N \geq 0, p_{ij}^S \geq 0$ where $p_{ij}^E + p_{ij}^W + p_{ij}^N + p_{ij}^S = 1$.

Figure 2: An example of simulated trajectory arriving from east to a cell $C_{ij}$ with direction probability distribution given in the right-hand side of the figure.

Now consider $r_{k,t} = [x'_k(t) \ y'_k(t)]^T$ to be the observed position of the object at that time. In addition, let $X_{k,t} = [x_k(t) \ y_k(t) \ v_x(t) \ v_y(t)]^T$ to be the state vector which represents the position and the speed of the $k$th moving object at time $t$. The state at time $t$ depends on the previous state and the flow control of each grid cell. Flow control, $U_{ijk}$, is randomly generated using equation (1) once the object enters the grid cell. This vector will be kept the same while the $k$th object resides in that cell. The motion of the object at the time $t$ considering it is in $C_{ij}$ can be defined as:

$$X_{k,t+1} = AX_{k,t} + BU_{ijk}$$

$$r_{k,t} = CX_{k,t} + R_t$$

(2)

where $R_t \in \mathbb{R}^{2x1} \sim \mathcal{N}(0, \sigma_R)$ is used to simulate measurement noise, $A$ is state transition matrix, $B$ can be interpreted as a matrix that combines the effect of grid cell’s flow control with the current state of object’s motion, and $C$ is output matrix which extracts the observed position information from its state. These matrices are given as follow:

$$A = \begin{bmatrix}
1/\alpha & 0 & \Delta t & 0 \\
0 & 1/\alpha & 0 & \Delta t \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}, B = (1 - a) \begin{bmatrix}
\Delta t & 0 \\
0 & \Delta t \\
0 & 1 \\
0 & 0
\end{bmatrix}, C = \begin{bmatrix}1 & 0 & 0 & 0 \end{bmatrix}$$

(3)

where $a \in (0,1)$ controls the effect of flow control on the object movement.
The initial values for the states in equation (2), \( X_{k,0} \), is randomly generated from a source grid cell, \( C_{ij}^{\text{source}} \). Source grid cells are predefined cells where objects are first entering the scene. The initial state is generated as follow:

\[
X_{k,0} \sim \mathcal{N}(\mu_{X_0}, \sigma_{X_0}, C_{ij}^{\text{source}}, k)
\]  

(4)

where \( \mu_{X_0} = [x_0 \ y_0]^T \) is the most probable spot where object appears for the first time and \( \sigma_{X_0} \) denotes the uncertainty of this area.

Our proposed trajectory generation process used to simulate random trajectories is given in Algorithm 1. Figure 3 shows some randomly generated trajectories using the proposed trajectory generator algorithm.

**Algorithm 1** random trajectory generator

**Inputs:**
- Scene template:
  - \( \mathcal{N}(\mu_{X_0}, \sigma_{X_0}, C_{ij}^{\text{source}}) \) for all source grid cells, \( \mathcal{P}(\mu, \sigma, C_{ij}) \) and \( \mathcal{N}(\mu_{\sigma}, \sigma_{\sigma}, C_{ij}) \) for all non-source grid cells, scene width \( (W) \), and scene height \( (H) \)
  
**Outputs:**
- \( T_x = (r_{x,0}, \ldots, r_{x,|t|}) \)
  1. Initialize \( T_x =<> \), \( U_{ijk} = [0 \ 0]^T \), \( t=0 \)
  2. Choose a source grid cell and draw \( X_{k,0} \) from equation (4)
  3. calculate \( r_{k,0} \) using \( X_{k,0} \) and equation (2) and append it to \( T_x \)
  4. \( t := t + 1 \)
  5. while True do:
  6. calculate \( X_{k,t+1} \) and \( r_{k,t} \) using equation (2)
  7. if \( x_k(t) > W \) or \( x_k(t) < 0 \) or \( y_k(t) > H \) or \( y_k(t) < 0 \) then
  8. break
  9. end if
  10. append \( r_{k,t} \) it to \( T_x \)
  11. if sample enters new cell then
  12. sample \( U_{ijk} \) using equation (1)
  13. end if
  14. \( t := t + 1 \)
  15. if \( t > \ell_{\text{max}} \) then
  16. return <>
  17. end if
  18. end while
  19. return \( T_x \)
3.2. Network architecture

Our proposed network architecture is based on multilayer Long Short-Term Memory (LSTM) networks. LSTM is an improvement over Recurrent Neural Networks (RNNs). RNNs are neural networks that hidden units form a directed cyclic connection, allowing it to capture temporal dependencies. These types of networks are shown to be very successful for sequential data such as natural language processing [29] and speech recognition [30]. At each time step $t$, the state is a function of the previous state, $h_{t-1}$, and current input, $S_t$:

$$h_t = f_\theta(h_{t-1}, S_t)$$  \hfill (5)

In practice, RNN does not perform well in learning long-term dependencies [31]. To overcome this problem various modification of RNN have been proposed. One popular modification of RNN is LSTM [32] which has been successfully applied to problems in many domains [33-35]. LSTM include input gate, forget gate, and output gate which controls what information needs to be stored and what needs to be deleted. Therefore, unlike normal recurrent neural network LSTM can retain long term temporal dependency. A popular version of LSTM cell [36] is illustrated in Fig. 4. LSTM blocks are described with the following equations:

$$i_t = \sigma(W_i x_t + W_f h_{t-1} + W_c c_{t-1} + b_i)$$  \hfill (6)

$$f_t = \sigma(W_i x_t + W_f h_{t-1} + W_c c_{t-1} + b_f)$$  \hfill (7)
where $\sigma(\cdot)$ is sigmoid function, and $i$, $o$, $f$, and $c$ are input gate, output gate, forget gate, and cell activation vector respectively. $W_i$s and $b_i$s are weight matrices and bias terms respectively.
layers are shared. The output is then connected to the first dense layer denoted as Dense layer 1 in the figure. The network is followed by another stack of BLSTM and a dense layer. The final layers of the proposed network are an LSTM layer followed by a densely connected layer which is connected to the output layer.

Figure 5: Network architecture

The number of nodes in each hidden layers is set by using cross-validation technique where it is chosen from 16, 32, 64, 128, and 256. First, we set the number of nodes of all hidden layers to 16. Beginning from the first hidden layer, we evaluated the effect of selecting a different number of nodes for that layer while keeping the number of nodes in other layers fixed. The number of nodes which produces the minimum mean square error (MSE) in the validation set is chosen for that layer. We repeat this procedure a few times. Table 1 summarises the number of nodes we used for different hidden layers of the proposed network architecture.
Table 1: Number of nodes in different hidden layers of the proposed network

| Layer Name               | Number of Hidden Layers |
|--------------------------|-------------------------|
| LSTM layer 1_y, LSTM layer 1_x | 32                      |
| LSTM layer 2_y, LSTM layer 2_x | 32                      |
| Dense layer 1             | 64                      |
| LSTM layer 3              | 32                      |
| LSTM layer 4              | 32                      |
| Dense layer 2             | 64                      |
| LSTM layer 5              | 64                      |
| Dense layer 3             | 64                      |

4. Experimental results and discussion

In this section, we present the experimental result that we conducted to evaluate the proposed method. We first introduce the dataset that we used in this study. We then present the training strategy which is used to learn the network’s weights. The experimental results related to clustering performance is discussed in Section 4.3. Finally, the results of the execution time is discussed in Section 4.4.

All experiments in this study were executed on a computer equipped with an AMD Ryzen 5 1600 processor, an NVIDIA GeForce GTX 1080 GPU, and 16-GB memory. The system was running on Linux Ubuntu.

4.1. Dataset

To verify our approach, we used Lankershim dataset [40]. Lankershim is part of Next Generation Simulation (NGSIM) program provided by the U.S. Federal Highway Administration (FHWA). The dataset provides trajectories of moving vehicles on Lankershim Boulevard in the Universal City neighborhood of Los Angeles, CA on June 16, 2005. The data are placed into 8:30 am to 8:45 am and 8:45 am to 9:00 am subsets. We only used the trajectories took place near an intersection and removed trajectories outside of this area (Fig. 6). We also removed the trajectories which are shorter than 10 observations. We ended up with 1095 and 1117 trajectories in 8:30 am to 8:45 am and 8:45 am to 9:00 am subsets respectively.
4.2. Training

We trained the network’s weights in two stages. At the first stage, we used synthetic data generation strategy discussed in section 3.1. Then we fine-tuned the weights using a subset of Lankershim dataset.

For the first stage, we generated a total number of 1,000,000 pairs of random trajectories using different scene templates. All trajectories are then preprocessed with strategy mentioned in section 3. We kept 80% of the data for training and the remaining 20% for validation purpose. DTW, LCSS, and MoH similarities for each pair were calculated and used as target values of the network. The network is trained using stochastic gradient descent (SGD) optimizer. We start with learning rate at 0.01. Each time the validation error reaches a plateau, we decrease the learning rate by the magnitude of 10 and continue the training procedure. We initialize the network’s weights by following [41] and we set the batch size to 512. TensorFlow is used to implanted and train the network.

In the second stage, we fine-tune the network’s weights using 8:30 am to 8:45 am subset in Lankershim dataset. For each pair of trajectories, we calculated DTW, LCSS, and MoH similarities. Similar to the first stage of training, we used SGD optimizer. We set the learning rate to $10^{-4}$ and fine-tune the network for 2 epochs.

4.3. Trajectory clustering

We evaluate the trained networks for the task of trajectory clustering on Lankershim dataset. Our clustering approach is based on clustering data using the similarity matrix [3]: pairwise similarities between all samples are calculated and stored in a similarity matrix. Then, a standard distance-based clustering algorithm such as agglomerative or k-means is used to
cluster the trajectories based on the similarity matrix.

We used Correct Clustering Rate (CCR) to evaluate the clustering performance [4, 23, 24]. Let G and C to denote the ground truth and predicted cluster labels. Given the label assignment between the ground truth labels and the predicted cluster labels, the CCR is defined as

\[
CCR = \frac{1}{N} \sum_{i=1}^{K} p_i
\]  

(12)

where N and K are the total number of trajectories and the number of clusters respectively. Finally, \( p_i \) is given by [24]:

\[
p_i = \begin{cases} 
|c_i \cap g_m| ; & \text{given } c_i \in C \text{ assigned to } g_m \in G \\
0 ; & \text{otherwise}
\end{cases}
\]  

(13)

We used 8:45 am to 9:00 am subset in Lankershim dataset to evaluate the clustering performance of the trained network. Since the ground truth labels are not provided in Lankershim dataset, we manually labeled the trajectories into 19 activities. These activities are typical ones happen in intersections where vehicles go straight through, turn left and turn right.

We compared our method with DTW, LCSS, and MoH. To cluster the trajectories we used k-means, spectral clustering, and agglomerative for both our method and the competitor methods. For the proposed network we used the trained network without being fine-tuned and fined-tuned networks with 30%, 60% and 100% of 8:30 am to 8:45 am subset of Lankershim dataset. The average CCR of clustering algorithms for each distance method is reported in Table 2. The CCR for the network which is trained using only simulated trajectories is promising. For instance, the returned EMoH can achieve a 0.925 correct clustering ratio while the network is not yet fine-tuned with real trajectories. The results also indicate that estimated similarities after fine-tuning stage achieve a better clustering performance. For instance, the EMoH produced by the network can achieve 0.939, 0.941, and 0.971 correct clustering rate when it is fined-tuned with 30%, 60%, and 100% of trajectories in 8:30 am to 8:45 am subset. After the fine-tuning stage, the clustering performance of estimated distance measures produced by the network is comparable with the clustering results produced by exact dissimilarity measures.

The resulting clusters founded by the proposed method using EMoH distance is shown in Fig. 7. In this figure, the colors indicate the predicted cluster label. The predicted clusters are typical activities found in an intersection: crossing the intersection, turning left, and turning right.
Table 2: Clustering accuracy for different similarity measures and clustering technique

| Similarity Measures | Percentage of 8:30 am to 8:45 am subset used to fine-tune the network | Correct Clustering Rate (CCR) |
|---------------------|------------------------------------------------------------------------|------------------------------|
| DTW                 | -                                                                      | 0.864                        |
| LCSS                | -                                                                      | 0.931                        |
| MoH                 | -                                                                      | 0.974                        |

| Estimated DTW (EDTW) | Percentage of 8:30 am to 8:45 am subset used to fine-tune the network | Correct Clustering Rate (CCR) |
|----------------------|------------------------------------------------------------------------|------------------------------|
|                      | without fine-tuning                                                   | 0.834                        |
|                      | 30%                                                                    | 0.851                        |
|                      | 60%                                                                    | 0.847                        |
|                      | 100%                                                                   | 0.861                        |

| Estimated LCSSS (ELCSS) | Percentage of 8:30 am to 8:45 am subset used to fine-tune the network | Correct Clustering Rate (CCR) |
|-------------------------|------------------------------------------------------------------------|------------------------------|
|                         | without fine-tuning                                                   | 0.894                        |
|                         | 30%                                                                    | 0.922                        |
|                         | 60%                                                                    | 0.933                        |
|                         | 100%                                                                   | 0.933                        |

| Estimated MoH (EMoH)   | Percentage of 8:30 am to 8:45 am subset used to fine-tune the network | Correct Clustering Rate (CCR) |
|------------------------|------------------------------------------------------------------------|------------------------------|
|                        | without fine-tuning                                                   | 0.925                        |
|                        | 30%                                                                    | 0.939                        |
|                        | 60%                                                                    | 0.941                        |
|                        | 100%                                                                   | 0.971                        |

Figure 7: Predicted clusters where EMoH is used as similarity measure

4.4. Execution time

One advantage of the proposed model compared to traditional distance-based models is being faster to calculate the pairwise similarities. We compared the average execution time for es-
estimated similarities (denoting as EDTW, ELCSS, and EMoH) and exact similarities. To this end, we recorded the average time that is required to obtain pairwise similarity between all trajectories in 8:45 am to 9:00 am subset.

The results are shown in Table 3. As the result suggests the proposed method is substantially faster than any individual similarity function: while it takes almost 1 hour to calculate the similarity matrix using DTW, MoH, and LCSS for 8:45 am to 9:00 am subset, it only takes 22 seconds to obtain EDTW, EMoH, and ELCSS for the same subset. Beside, our model estimates all three distances without substantially increasing the execution time: estimating three similarity matrices instead of one, adds only 0.001 milliseconds to the execution time.

Table 3: Average execution time for calculating pairwise dissimilarity

| Similarity Measures | DTW  | LCSS | MoH  | EDTW | ELCSS | EMoH | EDTW+ | ELCSS+ | EMoH |
|---------------------|------|------|------|------|-------|------|-------|--------|------|
| Average Execution Time [ms] | 1.896 | 1.874 | 1.85 | 0.030 | 0.030 | 0.030 | 0.031 |

5. Conclusion

This paper proposed a deep learning architecture to estimate pairwise dissimilarity of two trajectories. The network accepts two raw trajectories with varying length and returns estimate for DTW, LCSS, and MoH dissimilarities simultaneously. A trajectory simulation is proposed to generate synthetic trajectories to be used to train the network. Compared to traditional similarity measure, the proposed method achieved a similar performance while the execution time is substantially reduced.

References

[1] M. Teimouri, U. Indahl, H. Sickel, and H. Tveite, "Deriving Animal Movement Behaviors Using Movement Parameters Extracted from Location Data," *ISPRS International Journal of Geo-Information*, vol. 7, no. 2, p. 78, 2018.

[2] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of Vehicle Trajectories," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 647-657, 2010.

[3] B. T. Morris and M. M. Trivedi, "Trajectory Learning for Activity Understanding: Unsupervised, Multilevel, and Long-Term Adaptive Approach," *IEEE Transactions on*
[4] H. Weiming, L. Xi, T. Guodong, S. Maybank, and Z. Zhongfei, "An Incremental DPMM-Based Method for Trajectory Clustering, Modeling, and Retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 5, pp. 1051-1065, 2013.

[5] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for datamining applications," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2000, pp. 285-289: ACM.

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

[7] L. Chen, M. T. Özuş, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, Baltimore, Maryland, 2005, pp. 491-502, 1066213: ACM.

[8] P. C. Besse, B. Guillouet, J.-M. Loubes, and F. Royer, "Review and perspective for distance-based clustering of vehicle trajectories," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3306-3317, 2016.

[9] W. Xiaogang, M. Keng Teck, N. Gee-Wah, and W. E. L. Grimson, "Trajectory analysis and semantic region modeling using a nonparametric Bayesian model," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008, pp. 1-8.

[10] C. Piciarelli and G. L. Foresti, "On-line trajectory clustering for anomalous events detection," *Pattern Recogn. Lett.*, vol. 27, no. 15, pp. 1835-1842, 2006.

[11] L. Chen and R. Ng, "On the marriage of Lp-norms and edit distance," presented at the Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, Toronto, Canada, 2004.

[12] S. Atev, O. Masoud, and N. Papanikolopoulos, "Learning traffic patterns at intersections by spectral clustering of motion trajectories," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 4851-4856: IEEE.

[13] H. Alt, "The computational geometry of comparing shapes," in *Efficient Algorithms*: Springer, 2009, pp. 235-248.

[14] R. Laxhammar and G. Falkman, "Online Learning and Sequential Anomaly Detection in Trajectories," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 6, pp. 1158-1173, 2014.

[15] H. Alt and M. Godau, "Computing the Fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 5, no.
[16] T. Eiter and H. Mannila, "Computing discrete Fréchet distance," Citeseer1994.

[17] T. Helms, L. Moldenhauer, and A. M. Uhrmacher, "GPU-BASED CALCULATION OF TRAJECTORY SIMILARITIES," 2014.

[18] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Nienattrakul, "Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs," in 2010 IEEE International Conference on Data Mining, 2010, pp. 1001-1006.

[19] L. Xiao, Y. Zheng, W. Tang, G. Yao, and L. Ruan, "Parallelizing Dynamic Time Warping Algorithm Using Prefix Computations on GPU," in 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013, pp. 294-299.

[20] H. Zhu, Z. Gu, H. Zhao, K. Chen, C. Li, and L. He, "Developing a pattern discovery method in time series data and its GPU acceleration," Big Data Mining and Analytics, vol. 1, no. 4, pp. 266-283, 2018.

[21] D. Bednárek, M. Brabec, and M. Krulis, "On Parallel Evaluation of Matrix-Based Dynamic Programming Algorithms," 2015.

[22] P. Huang and B. Yuan, "Mining Massive-Scale Spatiotemporal Trajectories in Parallel: A Survey," Cham, 2015, pp. 41-52: Springer International Publishing.

[23] B. Morris and M. Trivedi, "Learning trajectory patterns by clustering: Experimental studies and comparative evaluation," in Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, 2009, pp. 312-319.

[24] Z. Zhang, H. Kaiqi, and T. Tieniu, "Comparison of Similarity Measures for Trajectory Clustering in Outdoor Surveillance Scenes," in Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, 2006, vol. 3, pp. 1135-1138.

[25] K. Buza, A. Nanopoulos, and L. Schmidt-Thieme, "Fusion of similarity measures for time series classification," presented at the Proceedings of the 6th international conference on Hybrid artificial intelligent systems - Volume Part II, Wroclaw, Poland, 2011.

[26] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," Data Mining and Knowledge Discovery, journal article vol. 29, no. 3, pp. 565-592, May 01 2015.

[27] T. Górecki, "Classification of time series using combination of DTW and LCSS dissimilarity measures," Communications in Statistics - Simulation and Computation, vol. 47, no. 1, pp. 263-276, 2018/01/02 2018.

[28] H. Weiming, X. Xuejuan, F. Zhouyu, D. Xie, T. Tieniu, and S. Maybank, "A system for
learning statistical motion patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 9, pp. 1450-1464, 2006.

[29] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[30] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, 2013, pp. 6645-6649: IEEE.

[31] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157-166, 1994.

[32] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735-1780, 1997.

[33] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[34] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with lstm recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3547-3555.

[35] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[36] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *2013 IEEE workshop on automatic speech recognition and understanding*, 2013, pp. 273-278: IEEE.

[37] M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney, "Translation modeling with bidirectional recurrent neural networks," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 14-25.

[38] Y. Fan, Y. Qian, F.-L. Xie, and F. K. Soong, "TTS synthesis with bidirectional LSTM based recurrent neural networks," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[39] J. P. Chiu and E. Nichols, "Named entity recognition with bidirectional LSTM-CNNs," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 357-370, 2016.

[40] "NGSIM: Next Generation Simulation," ed. [www.ngsim.fhwa.dot.gov](http://www.ngsim.fhwa.dot.gov): FHWA, U.S. Department of Transportation, 2008.

[41] H.-G. Zimmermann, C. Tietz, and R. Grothmann, "Forecasting with Recurrent Neural Networks: 12 Tricks," in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin
