Noname manuscript No.
(will be inserted by the editor)

HyperETA: An Estimated Time of Arrival Method based on Hypercube Clustering

Oscar LiJen Hsu

the date of receipt and acceptance should be inserted later

Abstract The Estimated Time of Arrival (ETA) that predict the travel time of a given GPS trajectory has been extensively used in route planning. Deep learning has been widely applied to ETA prediction. However, prediction tasks involve some challenges, such as small data size, low GPU's precision, high training loss, and low accuracy. Herein, we present a new machine-learning algorithm called HyperETA for ETA prediction. HyperETA is based on a extraordinary clustering method, called Hypercube Clustering. We conducted experiments to compare HyperETA with a deep-learning-based method called DeepTTE by using taxi trajectories as a benchmark. Two variations of both methods were evaluated. The results indicated that HyperETA outperformed the deep-learning approach in terms of prediction accuracy.

Keywords Estimated Time of Arrival · ETA · trajectories · Deep Learning

1 Introduction

Deep learning, a relatively new paradigm in artificial intelligence, has attracted substantial attention from the research community owing to its remarkable potential compared with traditional techniques. Compared with traditional machine learning (ML) methods, deep learning can be used to model sophisticated functions through layers of nonlinear transformations trainable from the beginning to the end. Methods based on deep learning models have considerably advanced the state of the art in diverse domains.

The success of deep learning has attracted the attention of researchers from different domains, who have explored its use for solving various problems. The problem considered in this paper is the estimated time of arrival (ETA) for a given path, which is a classic problem pertaining to travel planning, car navigation, and traffic dispatch. Numerous variables must be considered to predict the ETA

Oscar LiJen Hsu
Taiwan
E-mail: techrxiv@olife.org

accurately, and scholars have investigated different methods for improving the average accuracy of ETA predictions.

Several deep-learning-based methods have been proposed for solving the ETA problem [21][19][11][16][5][6]. A seminal method is DeepTTE [21], which uses historical trajectories as training data and predicts the ETA of a given trajectory. DeepTTE is a hybrid model [3] that integrates recurrent neural network and convolutional neural network (CNN) models. It has been cited by many researchers, used for comparison, and extended.

We present a novel machine learning algorithm called HyperETA for predicting the ETA of a given trajectory. HyperETA is based on a hypercube-clustering method [9], [10], which measures the similarities among the data of various trajectories. In a previous study, hypercube clustering outperformed other algorithms such as TraClus [12] and grid clustering. The primary concept of hypercube clustering is to represent trajectories by using hypercubes, a type of data structure that is more robust against noise in trajectory data. In the training phase, HyperETA builds a trajectory model from historical trajectories. During the prediction phase, a given trajectory is transformed into hypercubes; the trajectory model is used to estimate the hypercubes' times, and the ETA is computed by summing up these times. Furthermore, in the proposed method, the parameters are set automatically, thus eliminating the need for guesswork or trial and error. Users can adjust the parameters by using metaparameters.

We conducted experiments to compare HyperETA with a deep-learning-based method called DeepTTE by using Cheng-Du taxi trajectories as a benchmark. The results demonstrated that HyperETA outperformed DeepTTE in terms of prediction accuracy.

The primary contributions of this study can be summarized as follows:

1. We designed a new ETA method based on hypercube clustering[9][10] to increase the accuracy of ETA prediction.
2. We designed an algorithm for setting parameters automatically.
3. We conducted experiments using a real-world dataset comprising GPS points generated by taxis in Cheng-du. The percentage error of the proposed method on this dataset was significantly lower than that of existing methods.
4. We empirically compared HyperETA with DeepTTE, a popular deep-learning-based method.
5. We discuss a few problems associated with deep learning, including data size, GPU's precision, training loss, and low accuracy.

The remainder of this paper is organized as follows. Section 2 introduces the related work, including DeepTTE and hypercube clustering. Section 3 presents the HyperETA algorithm. Section 4 describes the experiments and results. The final section presents our conclusion and an outline for future work.

2 RELATED WORKS

2.1 Deep Learning Methods

Deep neural networks (DNNs) are considered the state-of-the-art among ML techniques [15]. DNNs have been applied in many domains [20]. For example, NVIDIA's researchers used DNNs in hand gesture recognition [13]; they used their super-advanced machine, four super GPU cards, and the latest software they developed for deep learning.

Training DNNs to obtain a good model involves stringent requirements. DNNs often require vast sources of data to learn appropriate abstractions [20]. Consequently, massive computing power is required for data processing. Accordingly, expensive GPUs and compatible mainframes are required. Such high levels of computing power are essential for tuning the parameters that define deep learning models, such as learning rates, number of neural network layers, and network architecture. The parameters are determined intuitively and through trial-and-error approaches; they must be determined as fast as possible.

One limitation of DNNs is their high level of opacity [15]. The multilayer mathematical neuronal structure of DNNs makes it difficult to interpret them (loosely defined as the science of comprehending what a model does [7]) and explain why certain inputs lead to certain outputs. Therefore, DNNs are typically executed as black boxes [22] due to this lack of transparency.

Several studies have used deep learning to forecast travel times. For example, [21] used geo-convolutional neural networks in DeepTTE. DeepTTE has been cited by many researchers, in addition to being compared with and extended [19] to predict bus travel times.

Graph convolutional neural networks (GCNNs) have been widely used to predict ETA [11][16][5][6], where “Graph” represents the required roadmaps. All of these methods accept spatiotemporal data. In addition, a GCNN-based origin–destination method [11] was proposed for ETA estimation. An origin–destination method is not concerned with paths. Instead, it aims to determine ETA only depends on two points. In [16], deep learning techniques such as GCNN, LSTM (Long Short-Term Memory), and GRU (Gated Recurrent Unit) were used to predict ETA. In [5], a hybrid spatiotemporal GCNN was used to predict ETA and possible congestion. The experimental dataset was not published online. In [6], a multilayer perceptron (MLP) was used to predict ETA.

A GCNN requires road network information to reduce the ETA to some graph problem, such as TSP (Travelling Salesman Problem). Such a requirement may not be achievable in many applications. The present study focused on methods that do not use road network information. Therefore, GCNN methods were not considered in this study.

For most of the papers published thus far, the software has not been published online, and specifications of the computer language and hardware environment have not been provided. In some of these studies, the experimental data used are not freely available. Therefore, these data could not be involved in our experiments. Some of the works require roadmaps, which is not the focus of the proposed method.

2.2 Non–Deep Learning Methods

Trajectory clustering methods entail finding similar trajectories; hypercube clustering is one of such methods. A trajectory may be presented as a series of GPS waypoints or a series of handwritten position marks. By design, methods based on curve similarity, such as dynamic time warping (DTW) [8], should not be applied

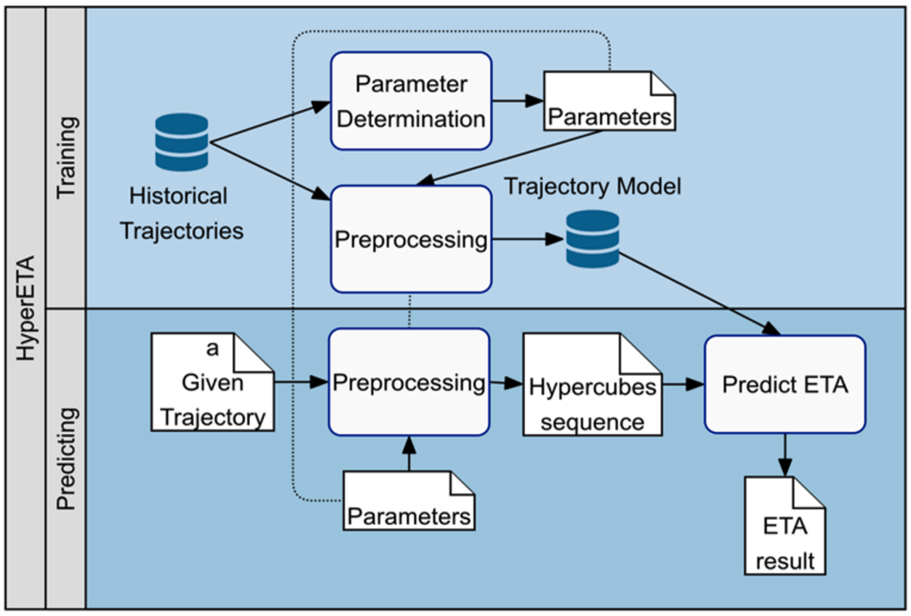

Fig. 1 HyperETA architecture

to scenarios involving multiple common subtrajectories. DTW can be used to measure the shape similarity between two trajectories; however, it cannot be used to find common subtrajectories. The computational cost of this approach is high because of the direct processing of GPS points without the use of a points-reduction method. Moreover, DTW is easily influenced by noisy data, resulting in large Euclidean distances and eventual breakdown. Subtrajectory clustering [12] divides a trajectory into line segments; however, the time cost of finding common subtrajectories can be high under conditions that involve intricate lines. Most of these methods consider only spatial data; that is, they disregard spatiotemporal data [4]. TraClus [12] is a famous classical method in trajectory clustering [1][2][18] and processes spatiotemporal data well. TraClus was specifically designed for trajectory clustering and widely used for the purpose. However, hypercube clustering[9] outperformed TraClus. HyperETA is based on hypercube clustering.

3 ARCHITECTURE OF HYPERETA

In this section, we describe the architecture of HyperETA. Fig. 1 presents a schematic of the architecture, including the training process and the prediction process. In the training process, the trajectory model and parameters are determined. In the prediction process, the ETA of a given trajectory is predicted. The dotted lines represent that the connected components are identical.

The following subsections describe the three essential steps of the proposed method, namely parameter determination, preprocessing, hypercube intersections, and ETA prediction.

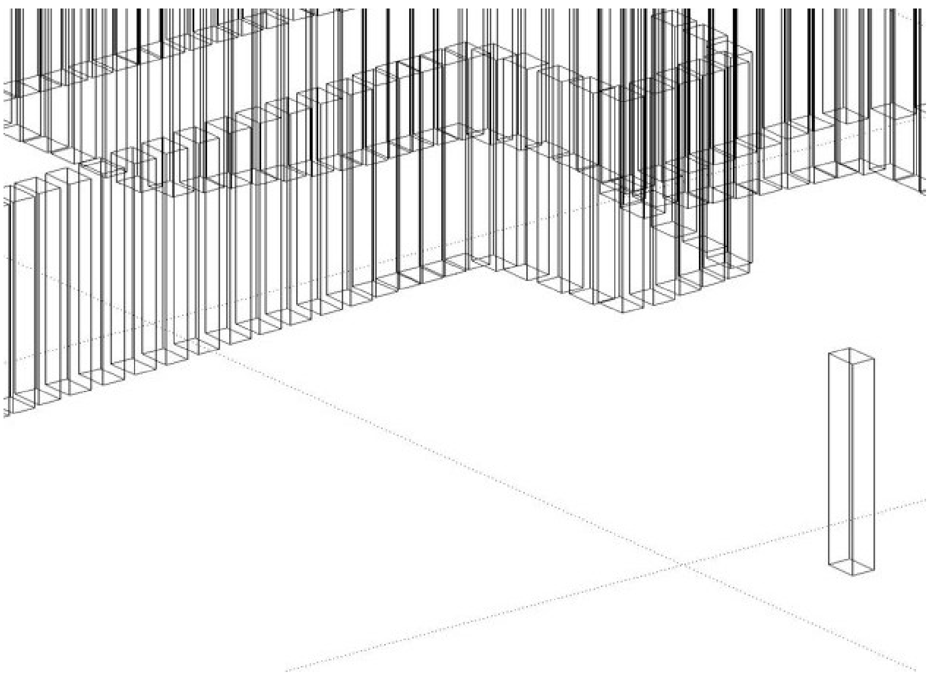

Fig. 2 Conceptual plot of hypercube series following preprocessing

3.1 Parameter Determination

Algorithm 1 Parameter Determination

```
Input: a sequence of number $\mathbf{S}$. A ratio $\gamma$. A multiplier $\mathbf{m}$.
Output: Threshold $\epsilon$.
1: $\mathbf{D}$ = a set of intervals of each sequence $\in \mathbf{S}$
2: find minimal $d' \in \mathbf{D}$ such that $\frac{|\{d<d' | d\in\mathbf{D}\}|}{|\mathbf{D}|} > \gamma$
3: $\epsilon = d' \times \mathbf{m}$
4: return $\epsilon$
```

We designed Algorithm 1 to automatically find the parameters for the preprocessing and ETA prediction steps. Users can input a sequences of latitude or longitude or time to obtain a output, ϵ_y or ϵ_x or τ, respectively.

where γ and $\mathbf{m}$ are metaparameters. The default values of γ and $\mathbf{m}$ were 0.8 and 5, respectively, representing 80% of hypercubes that involve at least five points.

3.2 Preprocessing

A trajectory is transformed into a hypercube sequence such that each hypercube includes a part of the trajectory. The trajectory points are sequentially scanned and segmented based on the cube range (ϵ_x, ϵ_y, τ). The position of a cube is the average of internal points. The direction of a cube is computed according to the first and last points. Direction is the fourth dimension that transforms cubes into hypercubes. In other words, a hypercube is a cube with direction. If we visualize the cubes that transform from a trajectory, it will look like Fig 2.

Algorithm 2 Preprocessing

```
Input: Trajectory $T = \{p_1, p_2, \dots, p_m\}$. Size threshold $\epsilon_x$, $\epsilon_y$. Time threshold $\tau$.
Output: Hypercubes series $C = \{c_1, \dots\}$
1: initialize a Hypercube $c \in C$
2: for $i = 1, \dots, m$ do
3:     add $p_i$ into $c$ until the range $(\epsilon_x, \epsilon_y, \tau)$ can not involve $p_i$ and new $c \in C$ will be initialed.
4: end for
5: return $C$
```

Algorithm 2 outlines the details of this transformation. Input T is a trajectory in which p_i.t is always larger than $p_{i-1}.t$, which is ordered temporally; ϵ_x and ϵ_y indicate the geographic size of a Hypercube; and τ is the temporal size. Initially, c is created, and then trajectory points are sequentially inserted into c. A new Hypercube is created if one of the criteria in (1) the range (ϵ_x, ϵ_y, and τ) is violated by the newly inserted p_i.

$$\begin{aligned} \max_{p \in c_j} |p_i.x - p.x| &\leq \epsilon_x \\ \max_{p \in c_j} |p_i.y - p.y| &\leq \epsilon_y \\ \max_{p \in c_j} |p_i.t - p.t| &\leq \tau. \end{aligned} \tag{1}$$

The direction of Hypercube c is represented by angle $c.\theta$, the definition of which is 2, and p_1 and p_n are respectively the first and last points inserted into c.

$$c.\theta = \arctan\left(\frac{p_n.y - p_1.y}{p_n.x - p_1.x}\right) \tag{2}$$

Furthermore, the preprocessing step helps prevent some fundamental problems. For a non-fixed-point interval trajectory, this step transforms the points into fixed-range hypercubes. Moreover, it can reduce computational costs by reforming the basic units of computing.

3.3 Hypercube Intersections

This step checks the intersection relation between two Hypercube sequences in four dimensions. It will be performed at the next step, ETA prediction. First, the algorithm checks the intersection between two sequences in temporal space. If there is no overlapping in temporal space, there will be no intersection, and the checking only costs constant time. Further examination begins at a temporal intersection if there is one. The examination includes temporal intersections, geographic intersections, and similarity in the relative direction of Hypercubes. The average time complexity with this step is O(n), where n is the number of Hypercubes.

Algorithm 3 shows the Hypercubes-Intersection method. If there is no temporal intersection, an empty set is output and only cost constant time complexity. The input is two sequences, C_a and C_b (in temporal order) and the output is information pertaining to the intersection. Line 1 check whether C_a and C_b have intersections in the time domain.

Algorithm 3 Hypercubes-Intersection method

```
Input: Hypercube series $C_a = \{c_{a,1}, c_{a,2}, \dots\}, C_b = \{c_{b,1}, c_{b,2}, \dots\}$.
Output: a bipartite graph $G_{AB} = (C_a, C_b, E_{AB})$.
1: D = { ($c_a$,$c_b$), $c_a \in C_a$ and $c_b \in C_b$ | $c_a$ have time intersection with $c_b$. }
2: for all ($c_a$,$c_b$) ∈ D do
3:     if $f(c_a, c_b)$ then
4:         Add $(c_a, c_b)$ to $E_{AB}$.
5:     end if
6: end for
7: return $G_{AB}$
```

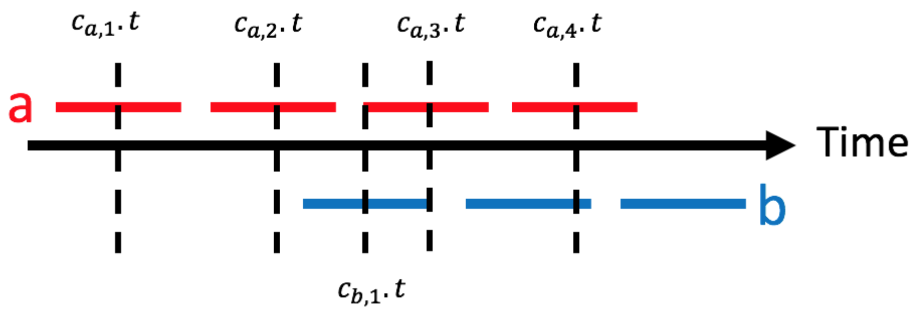

Fig. 3 Two trajectories with intersection in time dimension

Fig 3 presents an example of a temporal intersection. If there is a temporal intersection, then the algorithm uses the fact that the points in a trajectory are always in ascending temporal order so as to accelerate the checking process. The algorithm sequentially checks the temporal intersections between Hypercubes. A simple experiment that comparing this time-sequence technique with no time-sequence technique has been done and used Geolife dataset as a benchmark. Both methods got exactly the same result but different time cost. The no time-sequence technique edition used 1189 seconds to find every intersection. The time-sequence technique edition used only 425 seconds. The time-sequence technique edition is 2.8 times faster. The worse case time complexity of this technique is still O(n^2), where n is a number of cubes. However, the best case is O(1).

Only Hypercubes with temporal intersections are checked for geographic intersections and having a similar direction. LINE 3 is used to check two Hypercubes(they have a temporal intersection). $f(c_a, c_b)$ is Equation (3). For direction checking, we define that two sub-trajectories have no intersection relation if their difference in angle exceeds threshold ϕ. LINE 4 keeps the relation in E_{AB}.

The output is information related to Hypercube intersections, which are kept in a bipartite graph data structure, $G_{AB} = (A, B, E_{AB})$, where each vertex in A is a Hypercube in c_a, each vertex in B is a Hypercube in c_b, and $E_{AB} = \{(c_a, c_b) | c_a \in C_a \text{ and } c_b \in C_b\}$. Each edge in G_{AB} represents a pair of common sub-trajectories; because with proper normalization, it can be shown that two sub-trajectories satisfy the definition of common sub-trajectories.

$$f(c_i, c_j) = \begin{cases} & |c_i.x - c_j.x| < \epsilon_x \text{ and} \\ true, & |c_i.y - c_j.y| < \epsilon_y \text{ and} \\ & \angle(c_i.\theta, c_j.\theta) < \phi, \\ false, & \text{otherwise.} \end{cases} \tag{3}$$

3.4 ETA Prediction

This step is based on the techniques proposed in the previous subsections. Algorithm 4 is the primary method for predicting ETA.

Algorithm 4 ETA Prediction

```
Input: $C_{given}$. Trajectories Model TM. $\epsilon_x, \epsilon_y, \tau, \theta$.
Output: TotalTime.
1: TotalTime = 0
2: for $\forall c \in C_{given}$ do
3:     $C_\bigcap$ = Hypercube_Intersection($c$, TM, $\epsilon_x, \epsilon_y, \tau, \theta$)
4:     segment_Time = Min_DTW_Cube( $c$, $C_\bigcap$ )
5:     TotalTime = TotalTime + segment_Time
6: end for
7: return TotalTime
```

First, the input is a given hypercube series and a trajectory model (TM). The model includes a hypercube series, original trajectories, and a mapping table that maps the hypercubes to the original trajectories. The remaining inputs ϵ_x, ϵ_y , and τ are determined by Algorithm 1. The input θ is the angle difference between two hypercubes that can be tolerated by Hypercube_Intersection(). The output is the predicted ETA of C_{given}.

Lines 2–6 predict the ETA of each given hypercube. In Line 3, the Hypercube_Intersection() function is Algorithm 3. In Line 4, we further apply DTW [17] to find the most similar subtrajectory in the set $C_\bigcap$ to increase accuracy. The next GPS point on the subtrajectory is counted as well. The time cost of the subtrajectory is accumulated into TotalTime.

4 EXPERIMENT

4.1 Experimental Design

Data Description We evaluated the proposed method on a dataset of taxi trajectories in Cheng-du[21] . This dataset was determined to contain 19,400 trajectories

(712 000 GPS records) of 4565 taxi drivers in August 2014 in Cheng-du, China. The shortest trajectory contained only 15 GPS records (2.5 km), and the longest trajectory contained 119 GPS records (40 km). Moreover, we found a few abnormal GPS subtrajectories in the training and testing data during the experiments. These abnormal data were removed from the test data in the experiments.

We compared the following methods in our experiments:

1. DeepTTE, which uses a deep learning framework, PyTorch, to predict ETA.
2. DeepTTE-GPU, which uses GPUs(graphics processing units) to accelerate the computational speed, which is common in PyTorch.
3. HyperETA-noDTW, which is the first HyperETA before the application of DTW.
4. HyperETA, which is the primary method developed in this study.

Criteria We derived mean absolute percentage error (MAPE) values to evaluate the prediction results. MAPE expresses the accuracy as a ratio, as presented in (4):

$$MAPE = 100\% \times \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - F_i}{A_i}\right| \tag{4}$$

where A_i is the actual value and F_i is the forecast value.

Mean absolute error (MAE) and root mean square error (RMSE) were also used in the experiments. MAE is an arithmetic average of the absolute errors; that is, the average error between the actual value and forecast value. Regarding RMSE, large errors emerge from small errors.

Lab Environment The software and hardware environments in which the experiments were performed are outlined as follows:

- CPU: Intel Core i7-4790 3.6GHz
- Memory: DDR3-1600 16GB Non-ECC
- Storage: Intel SSD 535 series 240GB
- OS: Ubuntu Linux 19.10 64-bit

For DeepTTE and DeepTTE-GPU, the following environments were used:

- Python 2.7 + PyTorch 1.4 + CUDA 10.1 + cudnn 8.0.3
- GPU: GEFORCE GTX750Ti-DC2OC-4GD5

For HyperETA and HyperETA-noDTW:

- Python 3.7.7

Meta-parameter setting In this experiment, the metaparameters γ and m were set to 0.8 and 5, respectively, for Algorithm 4 of HyperETA. The setting represents that most of the hypercubes, approximately 80%, would involve at least five GPS points. θ was set to 10 deg.

The software used in the experiments has been released on Github, including HyperETA[1] and DeepTTE[2].

[1] `https://github.com/oscarhsu/HyperETA`

[2] `https://github.com/oscarhsu/DeepTTE`

Table 1 PERFORMANCE COMPARISON

TEST DATA	MAPE(%)	RMSE	MAE(sec)
DeepTTE	37.36	695.70	553.36
DeepTTE-GPU	39.12	649.88	509.30
HyperETA-noDTW	21.53	434.12	279.68
HyperETA	*20.52	*419.23	*269.72

TRAINING LOSS	MAPE(%)	RMSE	MAE(sec)
DeepTTE	16.88	*142.61	91.03
DeepTTE-GPU	25.75	468.41	381.49
HyperETA-noDTW	3.14	191.31	57.51
HyperETA	*3	189.92	*55.25

4.2 Accuracy Comparison

The experimental results are listed in Table 1.

The **Test Data** columns present the data obtained by the algorithms. HyperETA significantly outperformed DeepTTE and DeepTTE-GPU on these data. In HyperETA, the MAPE rate was only 20.52%; moreover, this algorithm had the most favorable RMSE and MAE values. HyperETA outperformed HyperETA-noDTW, indicating that HyperETA coupled with the DTW technique increased accuracy. The error rate of the deep-learning-framework-based DeepTTE was 33.63%. The error rate of DeepTTE-GPU, which uses a GPU instead of a CPU, was higher at 39.12%.

The Training loss columns present the data predicted by each algorithm for model training. Intuitively, the results should be perfect with near-zero error because the data were already learned. However, DeepTTE yielded distinct results, with a significant error rate of 16.88%. For DeepTTE-GPU, the error rate was terribly 25.75%, which was worse than that of the CPU version. HyperETA and HyperETA-noDTW yielded normal results, and their error rates were only 3% and 3.14%, respectively, which were considerably lower than those of DeepTTE and DeepTTE-GPU. The RMSE of HyperETA with training data was slightly higher than that of DeepTTE, but the MAE was smaller, similar to the other experimental values. This indicates that HyperETA exhibited some differentials in predicting long trajectories. However, the differentials occupied only small parts of the long trajectories, explaining why the MAPE of HyperETA was considerably more outstanding than those of the other techniques.

Distance Fig. 4, Fig. 5, and Fig. 6 present a comparison of the model performance levels for trajectories of different lengths. The benchmark is exactly the same as "TEST DATA" in Table 1, but the test dataset is separated into five datasets according to the distance. HyperETA outperformed DeepTTE for every path length. As illustrated in Fig. 4, most of the MAPE values of DeepTTE were higher than 22% but those of HyperETA were lower than 16.2%. After 5–10 distances, the MAPEs of HyperETA decreased significantly. As indicated in Fig. 5, the RMSEs of DeepTTE increased with distance. However, HyperETA exhibited an opposite phenomenon, where the RMSE values decreased. As shown in Fig. 6, the predicted MAE of DeepTTE (in seconds) increased with distance. However, the prediction quality of HyperETA remained steady, regardless of the distance.

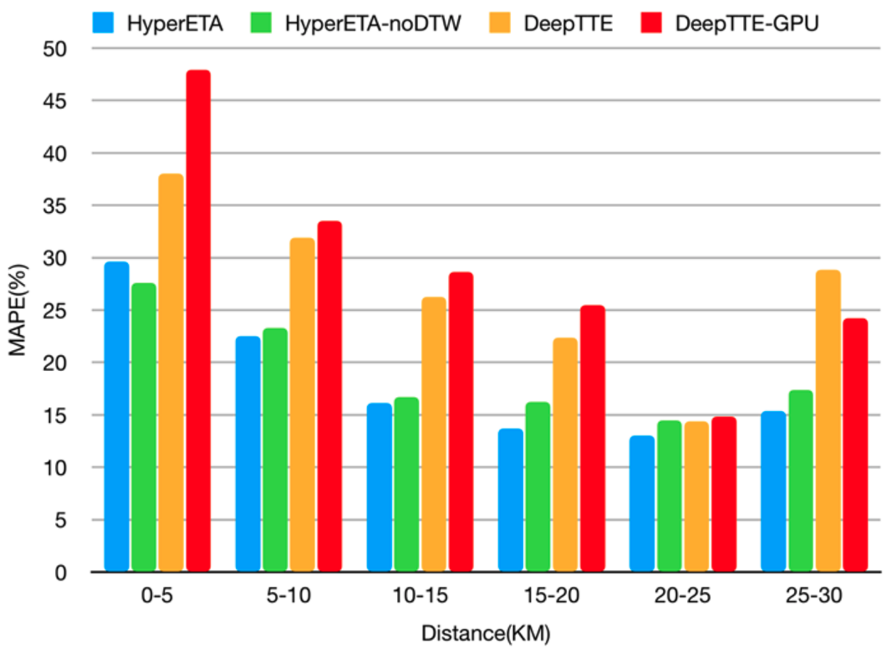

Fig. 4 Error rates of trajectories having different lengths

For most trajectory lengths, the experimental values yielded by HyperETA were superior to those yielded by HyperETA-noDTW. However, HyperETA-noDTW yielded lowest MAPE, RMSE, MAE values for the distances 0–5.

4.3 Discussion

Data Size HyperETA outperformed DeepTTE. DeepTTE requires massive training data to achieve high accuracy. HyperETA offered a higher precision than DeepTTE with a small quantity of training data. This demonstrates that deep learning requires a considerable quantity of labeled data to solve problems; it is impractical if the data volume is limited.

GPU DeepTTE-GPU yielded poorer results than those of DeepTTE because the GPU precision was low. In this study, we used a GTX750Ti, a consumer-level GPU, which is inferior to the supercomputing GPU, for example, ”TITAN V”. In further words, consumer-level GPUs can only process single-precision floating-point numbers, but supercomputing GPUs can process double-precision floating-point numbers. This considerably influences accuracy because each value in the feature vectors in DeepTTE is normalized to a floating-point number between 0 and 1. Narang[14] described the issue of the single-precision format (FP32) and how it influences gradient descent in backpropagation. PyTorch-1.6 uses a certain method to increase computational speed, which allows it to have the speed of FP16 while maintaining the accuracy of FP32. The default precision of a CPU is FP64 (i.e., the double floating-point format). Furthermore, the GPU in this study

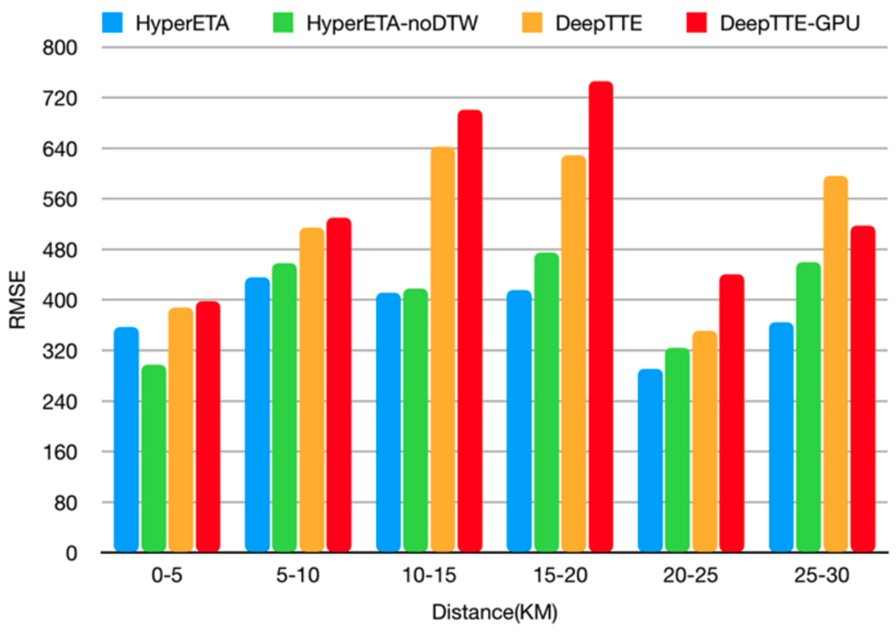

Fig. 5 RMSEs of trajectories with different lengths

was simultaneously used as the display card in the experiments, which affected its computational speed. So, to use an additional supercomputing GPU will benefit deep learning. However, doing so would not have helped in this study, because the small mainframe used in the study, the size of which was half that of a standard desktop, the mainframe had only one PCI-E slot. These practical problems indicate that to accelerate deep learning by using a GPU, large and expensive equipment is required, which is impractical for executing deep learning on small devices such as Internet of Things(IoTs) or mobile devices.

Training loss The training data predicted by DeepTTE were significantly poor, indicating that DeepTTE performed poorly at the fundamental level. The training data were already input in the training stage.

5 CONCLUSION

In this study, we proposed a novel approach called HyperETA for estimating the ETA for a given trajectory. HyperETA significantly outperformed DeepTTE and DeepTTE-GPU. Our research was a preliminary study comparing Hypercube clustering with deep learning. Numerous tasks are involved in the problems we studied, and they cannot be completed in one study. In future works, we will increase the robustness of the method proposed herein, including the reimplementation of HyperETA on a GPU. The implementation of HyperETA on a GPU can accelerate the computational speed. The tensor of PyTorch can be used in the implementation to quickly transfer programming functionalities from the CPU to the GPU.

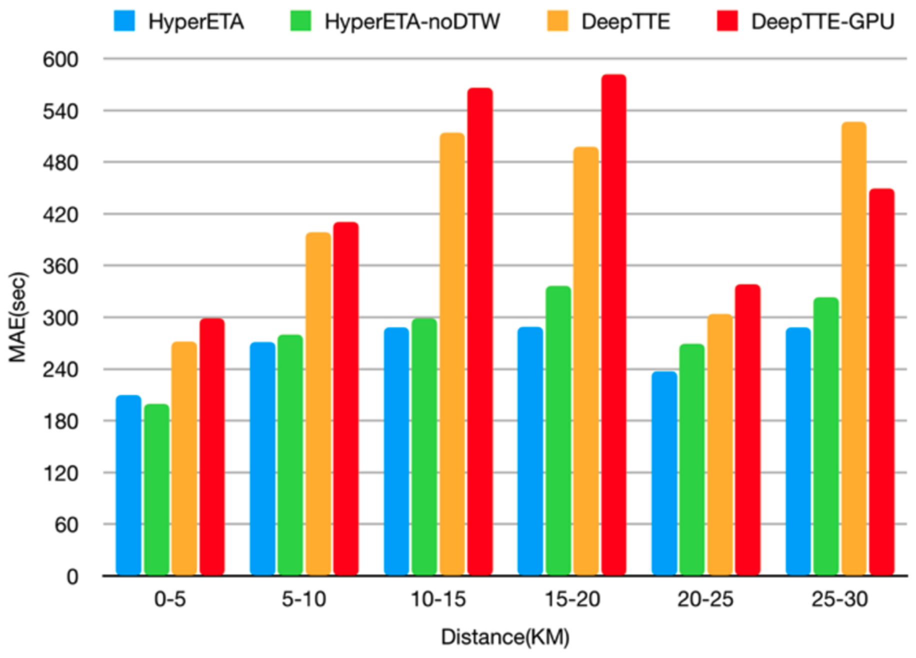

Fig. 6 MAEs of trajectories with different lengths

Conflict of Interest: The authors declare that they have no conflict of interest.

References

1. Atluri, G., Karpatne, A., Kumar, V.: Spatio-temporal data mining: A survey of problems and methods. ACM Comput. Surv. **51**(4), 83:1–83:41 (2018). URL `http://doi.acm.org/10.1145/3161602`
2. Bian, J., Tian, D., Tang, Y., Tao, D.: A survey on trajectory clustering analysis (2018)
3. Chen, Q., Wang, W., Wu, F., De, S., Wang, R., Zhang, B., Huang, X.: A survey on an emerging area: Deep learning for smart city data. IEEE Transactions on Emerging Topics in Computational Intelligence **3**(5), 392–410 (2019). DOI 10.1109/TETCI.2019.2907718
4. Cheng, R., Emrich, T., Kriegel, H.P., Mamoulis, N., Renz, M., Trajcevski, G., Zufle, A.: Managing uncertainty in spatial and spatio-temporal data. In: Data Engineering (ICDE), 2014 IEEE 30th International Conference on, pp. 1302–1305 (2014)
5. Dai, R., Xu, S., Gu, Q., Ji, C., Liu, K.: Hybrid spatio-temporal graph convolutional network: Improving traffic prediction with navigation data. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 3074–3082 (2020)
6. Fu, K., Meng, F., Ye, J., Wang, Z.: Compacteta: A fast inference system for travel time prediction. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 3337–3345 (2020)
7. Gilpin, L.H., Bau, D., Yuan, B.Z., Bajwa, A., Specter, M., Kagal, L.: Explaining explanations: An overview of interpretability of machine learning. In: 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA), pp. 80–89 (2018). DOI 10.1109/DSAA.2018.00018
8. ten Holt, G.A., Reinders, M.J.T., Hendriks, E.A.: Multi-dimensional dynamic time warping for gesture recognition (2007)

9. Hsu, O.L., Lee, C.: Common sub-trajectory clustering via hypercubes in spatiotemporal space. IEEE Access **8**, 23369–23377 (2020). DOI 10.1109/ACCESS.2020.2968150
10. Hsu, O.L.J., Lee, C.R.: Approach for finding ridesharing paths in spatiotemporal space. In: L. Berntzen (ed.) SMART 2016: The Fifth International Conference on Smart Systems, Devices and Technologies(include URBAN COMPUTING 2016), pp. 37–43. IARIA (2016)
11. Hu, J., Yang, B., Guo, C., Jensen, C.S., Xiong, H.: Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 1417–1428 (2020). DOI 10.1109/ICDE48307.2020.00126
12. Lee, J.G., Han, J., Whang, K.Y.: Trajectory clustering: A partition-and-group framework. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 593–604. ACM, New York, NY, USA (2007). URL `http://doi.acm.org/10.1145/1247480.1247546`
13. Molchanov, P., Yang, X., Gupta, S., Kim, K., Tyree, S., Kautz, J.: Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4207–4215 (2016). DOI 10.1109/CVPR.2016.456
14. Narang, S., Diamos, G., Elsen, E., Micikevicius, P., Alben, J., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G.: Mixed precision training. In: Proc. 6th Int. Conf. on Learning Representations (ICLR) (2018)
15. Ong, Y., Gupta, A.: Air5: Five pillars of artificial intelligence research. IEEE Transactions on Emerging Topics in Computational Intelligence **3**(5), 411–415 (2019). DOI 10.1109/TETCI.2019.2928344
16. Qiu, J., Du, L., Zhang, D., Su, S., Tian, Z.: Nei-tte: intelligent traffic time estimation based on fine-grained time derivation of road segments for smart city. IEEE Transactions on Industrial Informatics **16**(4), 2659–2666 (2019)
17. Rouanet, P.: Dynamic time warping python module (2019). URL `https://github.com/pierre-rouanet/dtw`
18. Tampakis, P., Pelekis, N., Doulkeridis, C., Theodoridis, Y.: Scalable distributed subtrajectory clustering. CoRR **abs/1906.06956** (2019). URL `http://arxiv.org/abs/1906.06956`
19. Tran, L., Mun, M.Y., Lim, M., Yamato, J., Huh, N., Shahabi, C.: Deeptrans: a deep learning system for public bus travel time estimation using traffic forecasting. Proc. VLDB Endow. **13**(12), 2957–2960 (2020). DOI 10.14778/3415478.3415518. URL `https://doi.org/10.14778/3415478.3415518`
20. Veres, M., Moussa, M.: Deep learning for intelligent transportation systems: A survey of emerging trends. IEEE Transactions on Intelligent Transportation Systems **21**(8), 3152–3168 (2020). DOI 10.1109/TITS.2019.2929020
21. Wang, D., Zhang, J.B., Cao, W., Li, J., Zheng, Y., Aaai: When will you arrive? estimating travel time based on deep neural networks. In: AAAI-18, pp. 2500–2507 (2018). URL `<GotoISI>://WOS:000485488902071`
22. Zeng, Z., Miao, C., Leung, C., Jih, C.J.: Building more explainable artificial intelligence with argumentation. In: Proc. 23rd AAAI/SIGAI Doctoral Consortium, p. 8044–8045 (2018)