



DIFFERENTIABLE GENERATIVE MODELS FOR TRAJECTORY DATA ANALYTICS

LI XIUCHENG

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

July 2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

[July 10, 2019]

[李修成]

.....

Date


.....

[LI Xiucheng]

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

[July 10, 2019]

[]

.....

Date

.....

[Cong Gao]

Authorship Attribution Statement

(B) This thesis contains material from [2] paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as LI, Xiucheng, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. Deep representation learning for trajectory similarity computation. IEEE 34th International Conference on Data Engineering (ICDE), 617-628 (2018). DOI: 10.1109/ICDE.2018.00062.

The contributions of the co-authors are as follows:

- Prof Cong provided the initial project direction, discussed the idea with me and edited the manuscript drafts.
- I designed the model, conducted all the experiments and finished the manuscript draft.
- Doctor Zhao and Prof Jensen helped edit the manuscript.
- A/Prof Wei gave some comments on our completed manuscript draft.

Chapter 4 is published as LI, Xiucheng, Gao Cong, Aixin Sun, and Yun Cheng. Learning Travel Time Distribution with Deep Generative Model. ACM The World Wide Web Conference (WWW), 1017-1027 (2019). DOI: 10.1145/3308558.3313418.

The contributions of the co-authors are as follows:

- Prof Cong discussed the idea with me and edited the manuscript drafts.
- I designed the model, conducted all the experiments and finished the manuscript draft.
- A/Prof Sun helped edit the manuscript.
- Mr Cheng gave some comments on the completed manuscript draft.

[July 10, 2019]

[李 修 成]

.....
Date

.....
[LI Xiucheng]

Acknowledgments

I would like to express my very great appreciation to my supervisor Prof. Cong Gao for his patient guidance, critiques, and carefulness. I also would like to thank Prof. Li Yi from SPMS, I gained a lot of mathematical insights and training from his MAS 723 Course–High Dimensional Probability, Christian S. Jensen and Prof. Aixin Sun from whom I learned a lot in paper writing.

I would like to say thanks to my family members for your continuous support these years, and my girlfriend Tao Lingling, I feel so lucky to meet you in such a good time!

The great thanks also go to our lab members in Data Management and Analytics Lab (DMAL) including Chen Lisi, Feng Kaiyu, Ji Zongcheng, Feng Shanshan, Zhao Kaiqi, Guo Tao, Chen Zhida, Liu Yiding, Chen Jinyao, Chen Yue, Chen Yile, Wang Zheng, Biaoge (Li Jing), Wang Yequan, Huang Keke, Grace Lee, Su Hao, Tianshu Liu, Guo Shengnan and the friends from SPMS–Guo Zhenyang and Yang Chengran, you guys make my daily life rich and joyful. I would also like to extend my thanks to the technician Mr Loo Kian Hock (Thomas) in DMAL for coordinating resources and offer me supports.

Contents

Acknowledgments	i
List of Figures	vi
List of Tables	viii
Summary	ix
1 Introduction	1
1.1 Background	1
1.2 Research Problems	2
1.2.1 Trajectory Representation Learning	2
1.2.2 Travel Time Distribution Learning	5
1.2.3 Spatial Transition Learning	7
1.3 Methodology Overview	9
1.3.1 Methodology of Trajectory Representation Learning	9
1.3.2 Methodology of Travel Time Distribution Learning	9
1.3.3 Methodology of Spatial Transition Learning	10
1.4 Summary of Contributions	10
1.5 Definitions and Notation	11
2 Literature Review	14
2.1 Trajectory Data Mining	14
2.2 Trajectory Similarity Computation	17
2.3 Travel Time Prediction	18
2.4 Spatial Transition Modeling	20
2.5 Representation Learning	21
2.6 Deep Probabilistic Generative Models	22

3	Trajectory Representation Learning	25
3.1	Overview and Contributions	25
3.2	Problem Statement and Preliminaries	26
3.2.1	Problem Statement	26
3.2.2	Preliminaries of sequence encoder-decoders	27
3.3	Proposed Method	28
3.3.1	Motivations and challenges	29
3.3.2	Handling varying sampling rates and noise	29
3.3.3	Learning consistent representations	31
3.3.4	Complexity of similarity computation	35
3.4	Experiments	35
3.4.1	Experimental setup	35
3.4.2	Parameter settings and training details	36
3.4.3	Performance evaluation	37
3.4.4	Scalability	43
3.4.5	Evaluation on the loss function	44
3.4.6	Effect of the cell size and the hidden layer size	44
3.4.7	Effect of the training data size	46
3.5	Summary	46
4	Travel Time Distribution Learning	48
4.1	Overview and Contributions	48
4.2	Problem Statement and DeepGTT Overview	49
4.3	DeepGTT Modeling	51
4.3.1	Road Segment Representation $\boldsymbol{\rho}_i$	51
4.3.2	Real-time Traffic Representation \mathbf{c}	53
4.3.3	Road Segment Speed v_i	54
4.3.4	Aggregating Road Segment Speeds	55
4.3.5	Variational Loss and Learning Algorithm	56
4.3.6	Prediction	57
4.4	Experiments	58

4.4.1	Experimental setup	58
4.4.2	Evaluation on travel time estimation	61
4.4.3	Performance on route recovery	62
4.4.4	The impact of real time traffic	63
4.4.5	Time Cost Study	64
4.5	Summary	66
5	Spatial Transition Learning	67
5.1	Overview and Contributions	67
5.2	Problem Statement	68
5.3	The Proposed Method – DeepST	68
5.3.1	The generative process	69
5.3.2	The representation of past traveled route	70
5.3.3	The representation of destination	70
5.3.4	Inference with VAEs	72
5.3.5	Route prediction and likelihood score	76
5.3.6	The complexity analysis	77
5.4	Experiments	77
5.4.1	Experimental setup	77
5.4.2	The most likely route prediction	82
5.4.3	Evaluation on route recovery	85
5.4.4	Scalability	86
5.4.5	Parameter sensitivity study	86
5.5	Summary	86
6	Conclusions and Future Work	88
6.1	Conclusions	88
6.2	Future Work	90
	Appendix - List of Publications	92
	References	93

List of Figures

1.1	Examples of the challenge in quantifying trajectory similarity.	4
1.2	(a) The travel time distributions of two routes \mathbf{r}_A and \mathbf{r}_B , $\mathbb{E}[t_A] = 26$, $\mathbb{E}[t_B] = 30$. (b) Example of route recovery from sparse trajectory.	6
1.3	Road network.	8
3.1	Sequence encoder-decoder model. The model reads the sequence \mathbf{x} and outputs the sequence \mathbf{y} , where EOS is a special token indicating the end of a sequence and \mathbf{v} is the representation of \mathbf{x} . The hidden state \mathbf{h}_t captures the sequential information in $[\mathbf{x}, y_1, y_2, \dots, y_{t-1}]$	27
3.2	T_b and $T_{b'}$ are two trajectories generated from an underlying route \mathbf{r} . After transforming the coordinates to cells, their corresponding sequences are \mathbf{y}, \mathbf{y}' respectively. The sample points in the two trajectories interleave on the route.	31
3.3	Creating two sub-trajectories T_a and $T_{a'}$ from trajectory T_b by alternately taking points from it.	38
3.4	(a)-(c) k -nn results when varying the dropping rate for $k = 20, 30, 40$. (d)-(f) k -nn results when varying the distorting rate for $k = 20, 30, 40$. . .	42
3.5	(a) k -nn query efficiency versus database size using the Porto dataset. (b) k -nn query efficiency versus database size using the Harbin dataset. . . .	43
3.6	The impact of training dataset size on the model using the Porto dataset.	46
4.1	The graphical model of the generative process in which $\boldsymbol{\rho}_i \in \mathbb{R}^{ \boldsymbol{\rho}_i }$, $v_i \in \mathbb{R}$, $\mathbf{c} \in \mathbb{R}^{ \mathbf{c} }$ are latent variables and r_i, t are observed scalars.	50
4.2	The parameterization of $p(\boldsymbol{\rho}_i r_i)$	52

4.3	The real-time traffic condition C at 7am and 11pm, where the cell value indicates the average speed in that local area.	53
4.4	The spatial distribution of the GPS points.	59
4.5	The distributions of duration (minutes) and travel distance (km)	60
4.6	(a) MAE changes against travel distance, and over the training dataset size, respectively.	62
4.7	Accuracy comparison.	64
4.8	The travel time distributions of route \mathbf{r}_{102} on Jan 03 (a) and Jan 17 (b), both of them are Saturday.	65
4.9	Time cost of all methods grows linearly with the training data size. However, DeepGTT has a much smaller constant term than its competitors.	65
5.1	The graphical model of the generative process in which $\mathbf{c} \in \mathbb{R}^{ \mathcal{c} }$ is latent variable and represents the real-time traffic, \mathbf{d} is the trip destination.	69
5.2	The graphical model of the complete generative process in which $\mathbf{c} \in \mathbb{R}^{ \mathcal{c} }$, $\boldsymbol{\pi} \in \mathbb{R}^k$ are latent variables, $M \in \mathbb{R}^{2 \times k}$, $S \in \mathbb{R}_+^{2 \times k}$, $W \in \mathbb{R}^{n_x \times k}$ are parameters to be learned, and $\boldsymbol{\eta} \in \mathbb{R}^k$ is hyperparameter; $\boldsymbol{\pi}$ can be interpreted as the destination allocation indicator, the nonzero dimension of $\boldsymbol{\pi}$ indicates the proxy which the trip is allocated.	71
5.3	The inference framework.	74
5.4	The spatial distribution of the GPS points: (a) Chengdu with size 10×10 (km); (b) Harbin with size 28×30 (km).	78
5.5	The distributions of travel distance (km) and number of road segments: (a)-(b) for Chengdu; (c)-(d) for Harbin.	79
5.6	The route prediction accuracy of different methods versus travel distance.	84
5.7	Training time versus training data size on Harbin dataset.	85

List of Tables

1.1	Example of trips on road network.	7
1.2	Frequently used notation.	13
3.1	Dataset statistics.	36
3.2	Mean rank versus the database size using the Porto and Harbin datasets.	38
3.3	Mean rank versus d_1 using the Porto and Harbin datasets.	39
3.4	Mean rank versus d_2 using the Porto and Harbin datasets.	39
3.5	Mean cross-distance deviation for varying dropping rate d_1 and distorting rate d_2	41
3.6	Mean rank and training time (hours) for the model equipped with loss functions $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_3 + \text{CL}$ using the Porto dataset.	44
3.7	The impact of the cell size on the model.	45
3.8	The impact of the hidden layer size on the model.	45
4.1	Dataset statistics.	59
4.2	Performance comparison of different methods.	61
5.1	Dataset statistics.	78
5.2	CNN architecture	82
5.3	Overall performance.	83
5.4	Route recovery accuracy versus sampling rate.	84
5.5	The sensitivity to k on Chengdu dataset.	86
5.6	The sensitivity to k on Harbin dataset.	86

Summary

With the proliferation of GPS-enabled devices, trajectory data is being generated at an unprecedented speed. The trajectories are typically represented as sequences of discrete sample points, which carry rich spatiotemporal information. Mining patterns and distilling knowledge from such a large amount of trajectory data could potentially help address many real-world problems and improve our daily life experience. For instance, accurately and efficiently quantifying the similarity between two trajectories is a foundation for many trajectory based applications, such as tracking migration patterns of animals, mining hot routes in cities, trajectory clustering and moving group discovery.

In this dissertation, we seek to effectively and efficiently distill knowledge from trajectory data with differentiable generative models. We develop three flexible generative models with efficiency in mind. The resulting models not only are capable of revealing the useful patterns underlying the data, but also admit end-to-end training. Moreover, our methods scale to real-world large-scale trajectory datasets easily. Specifically, we explore three important research problems arising in big trajectory data analytics: 1) learning representation for trajectory similarity computation; 2) learning the travel time distribution for any route on the road network; 3) spatial transition learning on the road network.

In the study of trajectory representation learning, we propose the first deep learning approach – **t2vec** – to learning representations of trajectories that is robust to low data quality, thus supporting accurate and efficient trajectory similarity computation and search. Experiments show that our method is capable of higher accuracy and is at least one order of magnitude faster than the state-of-the-art methods for k-nearest trajectory search.

In the study of travel time distribution learning, we develop a novel deep generative model – **DeepGTT** – to learn the travel time distribution for any route on the route network by conditioning on the real-time traffic. **DeepGTT** interprets the generation of travel time using a three-layers hierarchical probabilistic model, and describes the generation process in a reasonable manner rather than simply learning by brute force, and thus it not only produces more accurate results but also is quite data-efficient. A variational loss is further derived and the entire model is fully differentiable, which makes the model easily scale to large data sets.

In the study of spatial transition learning on the road network, we present a novel deep probabilistic model – **DeepST** – which unifies three explanatory factors, the past

traveled route, the impact of destination and real-time traffic for the route decision. **DeepST** explains the generation of next road link by conditioning on the representations of the three explanatory factors. To enable effectively sharing the statistical strength, we propose to learn representations of k -destination proxies with an adjoint generative model. To incorporate the impact of real-time traffic, we introduce a high-dimensional latent variable as its representation whose posterior distribution can then be inferred from observations. An efficient inference method is developed within the Variational Auto-Encoders framework to scale **DeepST** to large-scale data sets.

Chapter 1

Introduction

In this chapter, we first present the background on trajectory data analytics in Section 1.1. Then, we introduce three trajectory analytics problems studied in this dissertation. For each problem, we motivate it with real world applications and discuss the corresponding challenges in Section 1.2. The high level ideas behind our proposed methods are briefly reviewed in Section 1.3. We summarize our main contributions in Section 1.4, and this Chapter is ended by talking about the key definitions and notation used throughout the dissertation in Section 1.5.

1.1 Background

With the ubiquitous availability of GPS-enabled devices, trajectory data is being generated at an unprecedented speed. The trajectories are usually recorded as sequences of location and timestamp tuples, which carry rich spatial and temporal information. Mining patterns and discovering knowledge from such archived trajectory data could potentially help us better understand our urban living spaces and benefit our daily life. For this reason, great efforts in trajectory data mining have been made during the past decades. For instance, predicting the next visiting location of a user [MPTG09, CYZ13, CLY14] enables the personalized commercial advertisements delivery; building the digital maps automatically from the vehicle trajectory data [CLH⁺16, LBE⁺12] could save huge labor investment; mining hot routes in cities [CSZ11a] could help the government to understand the commute patterns of the citizens, so as to provide better transportation service in the future; mining the GPS records generated by the studied wild animals could help the scientists track the migration patterns of the animals [LHJ⁺11], and consequently create a friendly living space for both human beings and wild creature; estimating travel time of a given route on a road network [IS11, ZN13] could help us perform route planing, taxis dispatching, and ride-sharing, etc.

Despite these successes, it is still challenging to extract the complex hidden patterns from the archived trajectory data for the traditional trajectory data mining techniques such as frequent pattern mining-based methods [CSZ11a, LHJ⁺11], or simple statistical models [IS11, ZN13, MPTG09, CYZ13]. Moreover, these traditional techniques also struggle in handling the massive amount of data. As an example, to measure the similarity of two trajectories, the traditional methods [YJF98, VKG02, CN04, CÖO05] usually relies on the pairwise matching and dynamic programming, which lead to not only inaccurate results but also a quadratic computation cost (see Section 1.2.1). Over the past years, the differentiable models (widely known as deep learning in the literature) have proved its capability in learning abstract representations from raw data. The variables of interest introduced in such models are differentiable with respect to the loss functions, and thus they conform to the usage of the numerical optimization algorithms [BCN18] and have the potential to handle the large-scale datasets. The differentiable models have achieved great successes in image classification, speech recognition, machine translation, and the challenging game play [HZRS16, CvMG⁺14, SVL14, VKK⁺15, SHM⁺16]. In contrast, they have been relatively less explored in spatiotemporal data analytics.

To overcome the challenges of traditional trajectory data mining techniques and take advantage of the power of differentiable models, in this dissertation, we seek to effectively and efficiently distill knowledge from trajectory data with the differentiable generative models. We develop three flexible generative models with efficiency in mind. The resulting models not only are capable of revealing the useful patterns underlying the data, but also admit end-to-end training, and thus scale to real-world large-scale trajectory datasets easily. Specifically, we explore three important research problems arising in big trajectory data analytics with the differentiable generative models: 1) learning representation for trajectory similarity computation; 2) learning the travel time distribution for a trip; 3) spatial transition learning on the road networks. In the first study, we propose a seq2seq-based generative model (t2vec) whereas we developed two novel deep probabilistic generative models (DeepGTT and DeepST) in the last two studies. Generally, we can categorize the trajectory data mining tasks into five genres: clustering, classification, prediction, pattern mining, and outlier detection. Our first study is related to clustering and classification, while the second and third study fall into the prediction and pattern mining, respectively, which we will give a discussion in Section 2.1.

1.2 Research Problems

1.2.1 Trajectory Representation Learning

Given a collection of historical trajectories, we aim to learn a representation $\mathbf{v} \in \mathbb{R}^n$ (n is the dimension of a Euclidean space) for each trajectory T such that the representation

can reflect the underlying route of the trajectory for computing trajectory similarity. The similarity of two trajectories based on the learned representations must be robust to non-uniform, low sampling rates and noisy sample points.

1.2.1.1 Motivations

Quantifying the similarity between two trajectories is a fundamental research problem and is a foundation for many trajectory based applications, such as tracking migration patterns of animals [LHJ⁺11], mining hot routes in cities [CSZ11b], trajectory clustering [HPL15] and moving group discovery [JYZ⁺08, LCJT13]. The importance of measuring trajectory similarity has also been recognized by researchers, and a number of classic methods have been proposed, such as dynamic time wrapping (DTW) [YJF98], longest common subsequence (LCSS) [VKG02], edit distance with real penalty (ERP) [CN04], and edit distance on real sequences (EDR) [CÖO05].

These existing methods usually try to form a pairwise matching the sample points of two trajectories and identify the best alignment using dynamic programming. More specifically, these pairwise point-matching methods implicitly partition the space into cells according to a threshold ϵ and match two points if they fall into the same cell. Dynamic programming is then used to find an alignment that minimizes a matching cost.

1.2.1.2 Challenges

The pairwise point-matching methods for computing trajectory similarity often suffer in three scenarios: the first is when the sampling rates of trajectories are non-uniform. Sampling rates often vary across devices due to different device settings, battery constraints, and communication failures. Even for the same device, the sampling rates may vary. For example, a taxi driver may alter the default device sampling rate to reduce the power consumption [ZZXZ12]. This may result in sampling points alternating between sparse and dense episodes.

This poses challenges to the existing methods—if two trajectories represent the same underlying route, but are generated at different sampling rates, it is difficult for these methods to identify them as similar trajectories. The second scenario where it is challenging to align the sample points of similar trajectories is when the sampling rate is low. For example, the sampling rates for trajectories generated from online “check-ins” (e.g., Foursquare, Facebook), geo-tagged tweets, geo-tagged photo albums, and call detail records are low and inherently non-uniform [RDT⁺15]. Third, the performance of these methods may be degraded when the sample points are noisy. Such noise may occur in GPS points when moving in urban canyons.

Moreover, the existing methods rely on local point matching and identify an optimal alignment using dynamic programming, which leads to quadratic computational complexity $O(n^2)$, where n is the mean length of the trajectories. We argue that a good trajectory similarity measure should be both accurate and scalable to large datasets. We illustrate the problems caused by non-uniform and low sampling rates with two examples.

Example 1: Consider the two trajectories $T_a = [a_1, a_2, a_3]$ and $T_b = [b_1, \dots, b_6]$ in Figure 1.1a that are sampled from the same underlying route with different sampling rate. Assuming that the cell threshold ϵ is 1, a pairwise point-matching method such as EDR will match (a_1, b_1) and (a_3, b_6) while the remaining points will be unmatched, which yields a cost (edit distance) of 5. Although the two trajectories represent the same underlying route, they differ when compared using the pairwise point-matching methods.

□

Example 2: To see the challenge of low sampling rates, consider the trajectory in Figure 1.1b consists of sample points $[a_1, \dots, a_5]$. Due to the low sampling rate in the middle part of the trajectory, the pairwise point-matching methods may falsely match the two trajectories because four point pairs are matched.

□

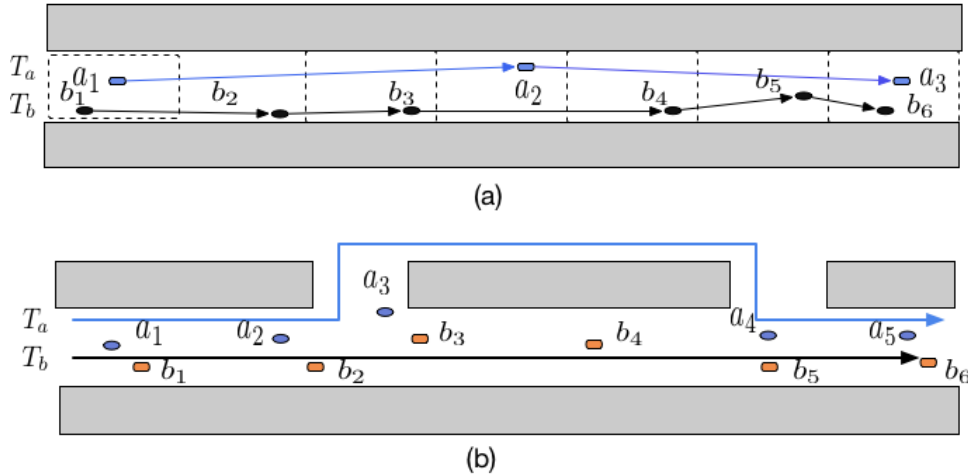


Figure 1.1: Examples of the challenge in quantifying trajectory similarity.

The essential problem underlying the two examples is how to infer the true route from the observed trajectory points in the presence of non-uniform sampling rates, low sampling rates, and noise. As shown in Figure 1.1b, it is hard to tell which route, e.g., $\mathbf{r}_A, \mathbf{r}_B$ is represented by the set of sample points $[a_1, \dots, a_5]$.

Recent studies [ZZXZ12, BRR14] reveal that the transiting patterns between certain locations are often highly skewed, i.e., some routes are more likely to be traveled than others, and these transition patterns, which are accumulated in the spatiotemporal

databases [ZZXZ12, BRR14, SZW⁺13a], hold the potential to help quantify trajectory similarity. To harness such transition patterns, Su et al. [SZW⁺13a] proposed an Anchor Points based Method (APM). APM first learns the transfer relationships among a fixed set of spatial objects (such as Points of Interest), called anchor points, from dense (i.e., with high sampling rate) historical trajectories by using Hidden Markov Models (HMM). Then sparse trajectories are calibrated to the anchor points, such that the existing pairwise point-matching methods, e.g., DTW, EDR, LCSS, can be employed to compute similarities of trajectories more accurately. APM requires the availability of a large amount of POIs and suffers from the limitations inherent in HMM like requiring explicit dependency assumptions to make inferences tractable [GFGS06]. Moreover, even after being trained on a historical dataset, APM still cannot reduce the $O(n^2)$ time complexity of the pairwise point-matching methods.

1.2.2 Travel Time Distribution Learning

Given a road network $G(V, E)$ and a historical trajectory dataset, we aim to learn the travel time distribution $p(t|\mathbf{r}, C)$, for a given route \mathbf{r} in the road network, conditioned on the real-time traffic condition indicator C (we defer to explain this C in Chapter 3).

1.2.2.1 Motivations

Estimating travel time of a given route on a road network is an essential task for many applications, including route planning, taxi dispatching, and ride-sharing. Many online web apps (e.g., Google Map and Uber) provide such service. Due to its great importance, significant research efforts have been made towards the accurate estimation of travel times [IS11, WKKL16, WZC⁺18, WFY18]. However, existing studies only focus on the estimation of *expected travel time*. We argue that it is much more helpful or even essential to estimate *travel time distribution* (the probability density function) of a given route, based on real-time traffic conditions. We illustrate this point through an example.

Example 3: Mary has a time budget of 35 minutes to reach airport from home. She queries for the best route, and there are two candidates — route A and route B , denoted by \mathbf{r}_A and \mathbf{r}_B , respectively. The travel time distributions of \mathbf{r}_A and \mathbf{r}_B are depicted in Figure 1.2a, with the expectation $\mathbb{E}[t_A] = 26$ (minutes), and $\mathbb{E}[t_B] = 30$ (minutes). If the system recommends route solely based on the expected travel time, then \mathbf{r}_A is the best choice. However, as shown in the figure, \mathbf{r}_A has a much larger variance. Therefore, Mary would have a high risk of missing her flight if taking \mathbf{r}_A . In contrast, the probability density function of t_B is much more concentrated, and it is much safer to take \mathbf{r}_B . \square

This example shows that it is more sensible to make decision based on the probability distribution of the travel time, instead of mere its expectation. In fact, estimating

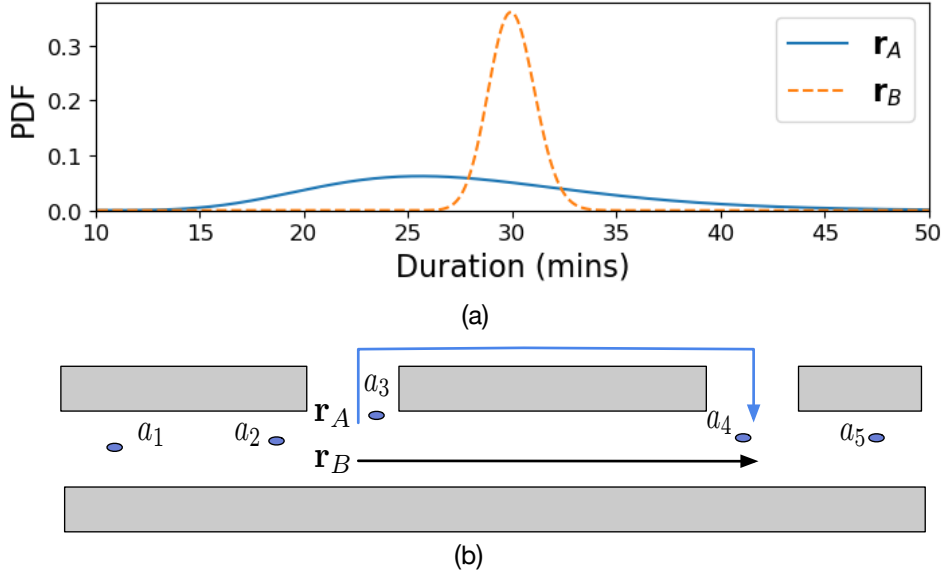


Figure 1.2: (a) The travel time distributions of two routes \mathbf{r}_A and \mathbf{r}_B , $\mathbb{E}[t_A] = 26$, $\mathbb{E}[t_B] = 30$. (b) Example of route recovery from sparse trajectory.

travel time distribution also addresses the issue of route recovery from the sparse trajectory [WMS⁺16].

Example 4: Figure 1.2b shows two possible routes $\mathbf{r}_A, \mathbf{r}_B$ from a_3 to a_4 (due to the low sampling rate) in an observed trajectory $T_a = [a_1, \dots, a_5]$. As the sampling points usually carry timestamps, we could get the travel time cost from a_3 to a_4 , denoted by t . The problem of inferring the most likely route can be expressed as

$$\operatorname{argmax}_{\mathbf{r}} \mathbb{P}(\mathbf{r}|t), \quad \mathbf{r} \in \{\mathbf{r}_A, \mathbf{r}_B\}.$$

Using the Bayes Rule, we have $\mathbb{P}(\mathbf{r}|t) \propto p(t|\mathbf{r})\mathbb{P}(\mathbf{r})$. That is, we need to access the value of travel time distribution $p(t|\mathbf{r})$ at any $t > 0$. \square

1.2.2.2 Challenges

In this study, we aim to learn the *travel time distribution* $p(t|\mathbf{r})$ for a given route \mathbf{r} , with the consideration of real-time traffic conditions. This problem is challenging due to a few reasons. First, data sparsity remains a key challenge. Even though we have access to a large number of trajectories, the trajectories demonstrate a skewed distribution in the space. Roads in central business district are frequently visited, while roads in rural or remote areas are rarely covered. It is reported that about 10% of paths cannot be estimated due to lack of neighbor trajectories [WKKL16]. Second, travel time

Table 1.1: Example of trips on road network.

Route	Destination	Frequency
$r_3 \rightarrow r_2 \rightarrow r_4$	C	400
$r_3 \rightarrow r_2 \rightarrow r_5 \rightarrow r_{10}$	C	100
$r_1 \rightarrow r_2 \rightarrow r_5 \rightarrow r_6$	A	100
$r_1 \rightarrow r_2 \rightarrow r_7 \rightarrow r_8$	B	100
$r_1 \rightarrow r_2 \rightarrow r_5 \rightarrow r_6 \rightarrow r_9$	B	100

heavily depends on real-time traffic conditions which are dynamic. Almost all existing methods [IS11, LAS15, WKKL16, WMS⁺16, WZC⁺18] assume that traffic conditions in the same time slot (*e.g.*, 8:00am-9:00am on every Saturday) are temporally-invariant, when estimating travel times. The assumption does not hold because traffic conditions can be influenced by many random variables such as the daily activities of people, road works, and accidents. Considering real-time traffic conditions makes data sparsity issue even worse. Further, travel time is also influenced by spatial characteristics of a road *e.g.*, number of traffic lights, number of lanes, and speed limit. To the best of our knowledge, no existing study has incorporated these heterogeneous influencing factors (both spatial and temporal) into a single model for travel time estimation.

1.2.3 Spatial Transition Learning

In this study, we investigate the problem, given the origin and destination, of predicting the most likely traveling route on a road network as well as outputting a probability value to indicate the likelihood of a route that being traveled.

1.2.3.1 Motivations

In taxi dispatch system, the origin and destination are usually given before the start of a trip, and thus predicting the most likely route could help us better arrange the taxi sharing by picking up the potential passengers that are waiting on or nearby the most likely traveled route. Furthermore, if we could score the likelihood of a route being traveled, we can also recover the underlying route from sparse trajectories. As mentioned in Example 4 in Section 1.2.2, the route recovery problem [WMS⁺16, BRR14] also requires us to evaluate the spatial transition likelihood of a route being traveled— $\mathbb{P}(\mathbf{r})$. If we treat a_3 , a_4 as the origin and destination respectively, there will be two candidate routes \mathbf{r}_A , \mathbf{r}_B , and we could score the likelihood of them to help infer the truly traveled route.

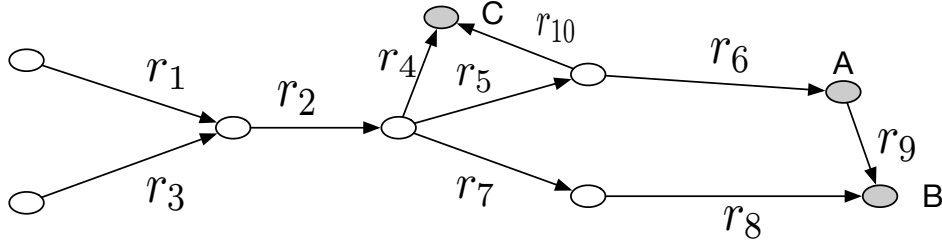


Figure 1.3: Road network.

1.2.3.2 Challenges

The problem being addressed can be summarized as modeling the spatial transition patterns of real-world trips on the road network. Several proposals [ZZXZ12, SZW⁺13a] have revealed that the transition patterns of vehicles are often highly skewed: some routes are more likely to be traveled than others. However, to reliably model such transition patterns, we argue that three key explanatory factors need to be carefully taken into consideration. We illustrate these key factors with an example; assuming we have observed 8 trips described in Table 1.1, over a road network showed in Figure 1.3.

First, the spatial transition patterns demonstrate strong sequential property. Consider that we try to predict the next transition of a vehicle driving on r_2 . According to the historical trips, $\mathbb{P}(r_4|r_2) = 4/8$ is greater than $\mathbb{P}(r_5|r_2) = 3/8$ and $\mathbb{P}(r_7|r_2) = 1/8$. We have a high confidence to predict that the vehicle will transit to r_4 . However, if we also know that the traveled road sequence is $r_1 \rightarrow r_2$, the prediction will then favor r_5 over r_4 . Second, the trip destination has a global impact on the transition. Now consider that a vehicle is driving on r_5 and the trip destination is C . Based on the historical trips it will have a higher probability of transiting to r_6 than transiting to r_{10} since $\mathbb{P}(r_6|r_5) = 2/3 > \mathbb{P}(r_{10}|r_5) = 1/3$. But if we take into consideration the trip destination, we would predict the next road to be r_{10} as $\mathbb{P}(r_{10}|r_5, C) = 1$. Third, the route choices are also influenced by the real-time traffic. The vehicle drivers tend to choose those less congested routes rather than the shortest one. Again, assuming that a vehicle is on r_2 and the trip destination is B . The historical trips show that it has equal probability of transiting to r_5 or r_7 as $\mathbb{P}(r_5|r_2, B) = \mathbb{P}(r_7|r_2, B) = 1/2$. However, if the traffic on r_7, r_8 is more congested than that on r_5, r_6, r_9 , the driver is more likely to choose r_5 instead of r_7 even if the route $r_7 \rightarrow r_8$ is shorter.

It is challenging to unify all these factors into a single model. To capture the sequential property, the authors [SZW⁺13a] propose to model the spatial transition patterns using the Hidden Markov Model (HMM) which requires explicit dependency assumptions to make inference tractable [GFGS06]. Wu *et al.* [WCS⁺17] consider the impact of destination on the route decision by treating each destination separately, thus failing to

share statistical strength across trips with nearby destinations. In addition, they assume that accurate ending streets of the trips are available, and use the corresponding road segments to help make decision. However, in some applications we may only have the rough destination coordinates at hand *e.g.*, the driver may not end a trip on the exact street as the user requested in the taxi dispatch system. To incorporate the influence of traffic when learning the transition patterns, [LAS15, WMS⁺16] assume that traffic conditions in the same time slot (*e.g.*, 7:00am-8:00am every weekday) are temporally-invariant, which is not real-time traffic and may not hold. To our knowledge, no existing work on spatial transition learning is based on real-time traffic.

1.3 Methodology Overview

1.3.1 Methodology of Trajectory Representation Learning

In the study of *Trajectory Representation Learning*, we propose a novel approach, called **t2vec** (trajectory to **vector**), to inferring and representing the underlying route information of a trajectory based on deep representation learning. The learned representation is designed to be robust to non-uniform and low sampling rates, and noisy sample points for trajectory similarity computation. This is achieved by taking advantage of the archived historical trajectory data and a new deep generative framework seq2seq.

The high level idea is to seek a model to maximize the probability of recovering route \mathbf{r} by conditioning on the observed trajectory T

$$\text{maximize } \mathbb{P}(\mathbf{r}|T).$$

This is accomplished by compressing the trajectory T into a vector \mathbf{v} , and then recovering route \mathbf{r} from \mathbf{v} . Such a method actually associate every observation T to a latent route \mathbf{r} . If there are multiple possible routes that might generate observation T , the model will be trained to recover the most likely one, and thus the most likely route information will be encoded into \mathbf{v} .

1.3.2 Methodology of Travel Time Distribution Learning

In the study of *Travel Time Distribution Learning*, we approach travel time distribution learning from deep probabilistic generative perspective [KW13, RMW14]. Specifically, we propose a model named **DeepGTT** (**Deep Generative Travel Time**), a three-layer hierarchical probabilistic model [GCSR95]. Each layer in **DeepGTT** captures underlying dependencies among the relevant random variables in a principled way.

In the first layer, we present two techniques, spatial smoothness embeddings and amortization [DHNZ95], to share statistical strength among different road segments to

address the data sparsity challenge. A representation ρ_i is learned for each road segment r_i . We further propose a convolutional neural network based representation learning component to address the second challenge. This learning component enables the encoding of dynamically changing real-time traffic conditions into a vector \mathbf{c} . In the middle layer, a nonlinear factorization model is developed to combine the road segment representation ρ_i and traffic condition \mathbf{c} to generate the travel speed v_i for road segment r_i . This auxiliary random variable v_i allows us to separate the static spatial road features from the dynamically changing traffic conditions, so as to address the third challenge. In the end layer, an attention mechanism [BCB14a] based function is proposed to generate the entire route travel time t by adaptively aggregating the road segment speed v_i . We derive a variational low bound [BKM16] to train the entire model in an end-to-end fashion.

1.3.3 Methodology of Spatial Transition Learning

In the study of *Spatial Transition Learning*, we again approach the spatial transition modeling problem from the generative perspective. We develop a novel deep probabilistic generative model, **DeepST** (**Deep Spatial Transition**), which unifies the aforementioned three key explanatory factors, sequential property, the impact of destination and real-time traffic into a single model, for learning the spatial transition patterns on the road network.

DeepST explains the generation of a route by conditioning on the past traveled road sequence, destination and real-time traffic representations. The past traveled road sequence is squeezed by a Recurrent Neural Net (RNN) which does not make the explicit dependency assumption. To incorporate the impact of destination, we propose to learn k -destination proxies with an adjoint generative model instead of treating the destinations separately. The benefits of introducing the destination proxies are two-folds: 1) the proposed method will be robust to the inaccuracy of destinations; 2) it allows effectively sharing the statistical strength across trips. The destination proxies are learned jointly with the generation of routes which permits end-to-end training. To account for the influence of real-time traffic, we propose to learn the real-time traffic representation with a latent variable, whose posterior distribution can then be inferred from the observations. In the end, we develop an efficient inference method to learn the posterior distributions of the latent variables of **DeepST** based on observations; the inference method is developed within the Variational Auto-Encoders (VAEs) framework and is fully differentiable, enabling **DeepST** to scale to large-scale datasets.

1.4 Summary of Contributions

To summarize, we have made the contributions as follows.

Firstly, we propose a seq2seq-based generative model to learn trajectory representations, for the fundamental research problem of trajectory similarity computation. The trajectory similarity based on the learned representations is robust to non-uniform, low sampling rates and noisy sample points. Our solution computes the similarity of two trajectories in linear time. For the purpose of learning consistent representations, we develop a new spatial proximity aware loss function and a cell representation learning approach that incorporate the spatial proximity into the deep learning model. To further speed up training, we propose an approximate loss function based on Noise Contrastive Estimation.

Secondly, for the first time, we develop a novel deep probabilistic generative model, named **DeepGTT**, to learn travel time distributions. **DeepGTT** is designed to be data-efficient. The model utilizes spatial smoothness embeddings and amortization to model road segments, and a convolutional neural network based representation learning component to capture real-time traffic conditions. The carefully designed hierarchical architecture allows us to separate the dynamically changing traffic conditions from the static spatial features, and thus enables incorporating the heterogeneous influencing factors (both spatial and temporal) into a single model for travel time learning. We develop an attention mechanism based function to collectively aggregate road segment speeds to generate the observations. Then, a variational low bound is derived to enable the model to be trained in an end-to-end fashion. As **DeepGTT** optimizes the complete distribution rather than a single mean value, it could incorporate more variability for more accurate prediction.

Thirdly, we develop a novel deep probabilistic generative model – **DeepST** – to learn the spatial transition patterns, which simultaneously takes into consideration the transition sequential property, the impact of destinations and real-time traffic. We propose a novel adjoint generative model to learn the k -destination proxies, which enables effectively sharing statistical strength across trips and the resulting model is robust to inaccurate destinations. We develop an efficient inference method that scales the model to large-scale datasets within the VAEs framework.

1.5 Definitions and Notation

We end this chapter by presenting the key definitions and notation used throughout the dissertation as follows.

Definition 1 *Underlying Route.* *An underlying route of a moving object is a continuous spatial curve (e.g., in the longitude-latitude domain), indicating the exact path taken by the object.*

The underlying route is only a theoretical concept as location acquisition techniques do not record moving locations continuously.

Definition 2 *Road Network.* A road network is represented as a directed graph $G(V, E)$, in which V, E represents the vertices (crossroads) and edges (road segments) respectively.

Definition 3 *Route.* A route $\mathbf{r} = [r_i]_{i=1}^n$ is a sequence of adjacent road segments, where $r_i \in E$ represents the i -th road segment in the route.

Definition 4 *GPS Trajectory.* A GPS trajectory T is a sequence of sample points $\langle p_i, \tau_i \rangle_{i=1}^{|T|}$ from the underlying route of a moving object, where p_i, τ_i represents the i -th GPS location and timestamp respectively.

Definition 5 *Trip.* A trip \mathbf{T} is a travel along a route \mathbf{r} in the road network starting at time s . We use $\mathbf{T}.\mathbf{r}$ and $\mathbf{T}.s$ respectively, to denote the traveled route and starting time of trip \mathbf{T} .

We denote scalars, vectors and matrices/sets using lower-case, bold-case and upper-case letters, *e.g.*, x, \mathbf{x} and X , respectively (the only exceptions are T and \mathbf{T} which are used to represent a trajectory and trip, respectively). We denote the elements of \mathbf{x} by x_i . We use $\mathbb{P}(\cdot)$ to denote the probability mass for the discrete random variables, and use $p(\cdot)$ or $q(\cdot)$ to denote the probability density function for the continuous random variables. To ease of reference, frequently used notation is given in Table 1.2.

Table 1.2: Frequently used notation.

Symbol	Definition
T (or T_a)	Trajectory
$ T $	Length of T
\mathbf{T}	Trip
\mathbf{r}	Underlying route
\mathbf{x}	A sequence of tokens
x_t	Token at position t
$\mathbf{x}_{1:t}$	A sequence of tokens from position 1 to t
\mathbf{v}	Embedded vector
\mathbf{h}	Hidden state (vector)
\mathcal{V}	Vocabulary
d_1 (d_2)	Dropping rate (distorting rate)
r_i	The i -th road segment in route \mathbf{r}
$\boldsymbol{\rho}_i \in \mathbb{R}^{ \rho_i }$	The learned representation of r_i
$v_i \in \mathbb{R}$	Travel speed of r_i
C	Real-time traffic indicator
$\mathbf{c} \in \mathbb{R}^{ \mathbf{c} }$	The learned real-time traffic representation
\mathbf{d}	Destination coordinate
$\boldsymbol{\pi}$	Destination allocation indicator

Chapter 2

Literature Review

In this chapter, we first briefly review the researches on trajectory data mining and analyze the issues of these proposals to introduce our three research problems, namely, *learning representation for trajectory similarity computation*, *travel time distribution learning* and *spatial transition learning for a trip* in Section 2.1. Then we survey the related work on trajectory similarity computation (the first study), travel time prediction (the second study), spatial transition modeling (the third study), representation learning (the first study), and deep probabilistic generative models (the second and third studies), in Section 2.2, 2.3, 2.4, 2.5, 2.6 respectively. In particular, we emphasize the differences between the related proposals and our proposed methods.

2.1 Trajectory Data Mining

Trajectory data mining is regarding discovering knowledge and useful information from trajectory data. A trajectory is usually recorded as a sequence of location and time-stamp tuples, which carry rich spatial and temporal semantics information. The easy availability of GPS-enabled devices has led to a massive amount of trajectory data being accumulated into the databases. Mining patterns and discovering knowledge from the archived trajectory databases could potentially help us to improve our urban living experience and, eventually, build the smart cities. Predicting the next visiting location of a user [MPTG09, CYZ13, CLY14] enables the personalized commercial advertisements delivery. Building the digital maps automatically from the vehicle trajectory data [CLH⁺16, LBE⁺12] could save huge labor investment. Mining hot routes in cities [CSZ11a] could help the government to understand the commute patterns of the citizens, so as to provide better transportation service in the future. Mining the GPS records generated by the studied wild animals could help the scientists track the migration patterns of the animals [LHJ⁺11], in order to create a friendly living space for both human being and wild creature. Estimating travel time of a given route on a road

network [IS11, ZN13] could help us perform route planing, taxis dispatching, and ride-sharing. Generally, we can categorize the trajectory data mining tasks into five genres: clustering, classification, prediction, pattern mining, and outlier detection.

Clustering. Trajectory clustering aims at partition trajectories into clusters such that the trajectories falling into the same cluster are closer based on a user specified discrepancy measure. This discrepancy measure is crucially important as different measures might yield distinct results. The widely-adopted trajectory similarity measures in practice during the past decades include DTW [YJF98], ERP [CN04], EDR [CÖO05], etc. Such methods try to form a pairwise matching the sample points in the trajectories and searching the optimal alignment using the dynamic programming. In addition, the proposal [ASKP03] models the trajectories as sequences of transitions between hidden positions and the HMM (Hidden Markov Model) which best fits the observed trajectories is used to represent a cluster.

Classification. The objective of trajectory classification is to classify trajectories into several pre-defined classes. We usually have a training dataset in which each trajectory is labeled and then we are asked to assign the unseen trajectories into the fixed classes. Trajectory classification finds its application in transportation mode detection [LCH⁺17], human gesture recognition [RDT⁺15], user-identity linkage [FZW⁺19]. There are generally two sort of methods adopted in trajectory classification, feature-based methods and distance-based methods. The traditional feature-based methods first extract a set of discriminative features from the raw trajectories and then train an existing supervised-classification model such as SVM (Support Vector Machine), Decision Trees to fit the training data pairs with the extracted features. The commonly used discriminative features include average speed, average acceleration, vehicle-direction, etc. Unlike the feature-based methods, the distance-based methods do not require the step of discriminative feature extraction. Instead, they rely on a pre-specified distance function to measure the similarity between two trajectories. The aforementioned trajectory similarity discrepancy methods such as DTW, ERP, EDR can be used. We would like to highlight that both the trajectory clustering and trajectory classification involve the employment of trajectory similarity function, in which our proposal *t2vec* is applicable.

Prediction. Trajectory prediction attempts to predict a quantity or symbol information relevant to trajectory data. Many applications such as travel time prediction, future location prediction and trip purpose identification fall into this genre. Travel time prediction aims to predicting the expected travel time of a route on a road network. Many proposals [IS11, ZN13, WFY18] treat it as a regression problem. The proposals on location prediction are usually based on Markov Models, sequential rules or Recurrent Neural Networks. The representative work [AMSS11, MRM12] using Markov Models assumes the transition of trajectories follows Markov property. Studies [MPTG09, YLWT11] based on sequential rules define a trajectory as an ordered sequence of locations, and

discovering the frequent patterns with the sequential pattern mining methods. Since the Markov Models is limited by the Markov assumption and the sequential rules cannot reveal the non-linear patterns, in recent years, several proposals [LWWT16, FLZ⁺18] try to approach the future location prediction problem with the Recurrent Neural Networks. Our second study travel time distribution learning falls into this genre.

Pattern Mining. Trajectory pattern mining seeks to discover the common movement patterns hidden in trajectories, and produce when and where the patterns occur as well as the individuals that are involved. The movement patterns include repetitive or periodic pattern mining, group pattern mining, and transition pattern mining. Repetitive pattern mining is concerned with discovering the regular patterns such as the movement of a commuter, which is repeated periodically. A common adopted approach [CMC07] is to consider the trajectory as a sequence of locations and employ the frequent pattern mining methods on the sequences to discover the repeated patterns. Group pattern mining [JYZ⁺08, WORN11] seeks to find the movement patterns involving a group of objects that move together. Transition pattern mining aims at revealing the spatial transition behavior of moving objects recorded by the trajectories. An example is the route decision of taxi drivers, the proposals [ZZXZ12, SZW⁺13a] have revealed that the transition patterns of vehicles are often highly skewed: some routes are more likely to be traveled than others. We focus on the spatial transition learning on the road network in our third study.

Outlier Detection. The goal of trajectory outlier detection [LHL08, LQZH12] is to find the trajectories that do not conform with the general behaviors of the trajectory dataset. Trajectory outlier detection is closely related to trajectory clustering. Actually, one of the most common used outlier detection approaches is based on the trajectory clustering technique. We first perform clustering on the trajectory dataset, the trajectories far away from the representatives (centriods) are regarded as the outliers. Hence, one important step is to choose an appropriate trajectory similarity measure, which is also relevant to our first study. Another approach on outlier detection is to interpret the generation of trajectory dataset using a probabilistic generative models, the trajectories that got lower likelihood scores will be considered as anomalies.

Despite the successes achieved by the aforementioned work in trajectory data mining, it is still challenging to extract the complex hidden patterns from the archived trajectory data for the traditional trajectory data mining techniques such as frequent pattern mining-based methods [CSZ11a, LHJ⁺11], or simple statistical models [IS11, ZN13, MPTG09, CYZ13]. Moreover, these traditional techniques also struggle in handling the massive amount of data, e.g. to measure the similarity of two trajectories, the traditional methods [YJF98, VKG02, CN04, CÖO05] usually relies on the pairwise matching and dynamic programming, which lead to not only inaccurate results but also a quadratic computation cost. For these reasons, we attempt to go beyond the traditional

trajectory data mining techniques by exploring the power of differentiable generative models. Specifically, we explore three important research problems arising in big trajectory data analytics with the differentiable generative models: 1) learning representation for trajectory similarity computation (**Clustering** and **Classification**); 2) learning the travel time distribution for a trip (**Prediction**); 3) spatial transition learning on the road networks (**Pattern Mining**), whose related literature reviews are given as follows.

2.2 Trajectory Similarity Computation

Computing the similarity between two trajectories is fundamental functionality in many spatiotemporal data analysis tasks. Not surprisingly, the problem of accurately and efficiently measuring the similarity of trajectories has been studied extensively [YJF98, VKG02, CN04, CÖO05]. DTW [YJF98] was a first attempt at tackling the local time shift issue for computing trajectory similarity. ERP [CN04], EDR [CÖO05], DISSIM [FGT07], and the model-driven approach MA [SAM⁺13] were developed to further improve the ability of capturing the spatial semantics in trajectories. Wang et al. [WSZ⁺13] studied the effectiveness of these similarity methods according to their robustness to noise, varying sampling rates, and shifting. All of these methods focus on identifying the optimal alignment based on sample point matching, and thus they are inherently sensitive to variation in the sampling rates. To solve this issue, APM [SZW⁺13b] and EDwP [RDT⁺15] are proposed. As discussed in the introduction, APM solves this issue by learning transition patterns of anchor points from historical trajectories. To compute the similarity of two trajectories, EDwP computes the cheapest set of replacement and insertion operations using linear interpolation to make them identical. Our solution **t2vec** is very different from APM and EDwP in that we aim to learn a vector that represents a trajectory and to then compute similarity using the new representation. Our study of *Trajectory Representation Learning* is also related to the inference of hidden routes from partial observations. The problem was originally studied under the name map matching [NK09] when the sampling rates are relatively high. Zheng et al. [ZZXZ12] first studied the problem in the very low sampling rates cases, and Banerjee et al. [BRR14] further explored it using Bayesian posterior inference to estimate the top- k most likely routes. Our first study differs from these two in that our ultimate goal is to learn representations of the trajectories rather than solely inferring the most possible routes.

Moreover, all the aforementioned existing measures for trajectory similarity are based on the dynamic programming technique to identify the optimal alignment which leads to $O(n^2)$ computation complexity. Given the complexity, it will be computationally expensive if we want to apply these similarity measures to cluster a large trajectory database. In contrast, our method **t2vec** has a linear time complexity $O(n + |\mathbf{v}|)$ to measure the

similarity of two trajectories, which is able to support analysis on big trajectory data, such as clustering trajectories. **t2vec** can also offer near-instantaneous response times that support interactive use, while the competition cannot.

2.3 Travel Time Prediction

The road segment-based methods first estimate travel time on each individual road segment independently by using the speeds information collected by loop detectors. Then travel time on a given route is estimated by summing up the estimated times of the road segments traversed in the given route. Such methods fail to consider delays caused by traffic lights, and left/right turns, resulting in large estimation errors.

To address the weaknesses of road segment-based methods, route-based methods attempt to estimate the travel time of a route as a whole. There are two main threads for route-based methods, namely, nearest neighbors search [TJ08, WKKL16] and trajectory regression [IS11, ZN13, WFY18]. Nearest neighbors search estimates travel time of a route by averaging travel times of the historical trajectories that have the closed origin and destination with the query route. Trajectory regression methods, on the other hand, treat travel times of the road segments as unknown variables and solve a regression problem using the historical trajectories.

Several works also attempt to estimate the travel time distribution for a given route on road network. Hunter et al. [HHR⁺13] investigate travel time distribution of a path in arterial networks using Gaussian Markov Random Field. Wu et al. [WMS⁺16] propose a model named **STRS**, that seeks to recover the route from sparse trajectories. **STRS** comprises of a travel time distribution estimation component and a spatial transition inference component. Its travel time distribution estimation component first learns the mean value of travel time by trajectory regression, and then empirically study the relationship between the variance and mean value to derive the distribution. **STRS** has been shown to be the state-of-the-art route recovery algorithm. Raghu et al. [GSA14] explore the prediction limits of different methods by estimating the entropy of the travel time distributions.

Almost all existing studies make the assumption that traffic conditions in the same time slot are temporally-invariant. In our study of *Travel Time Distribution Learning*, we relax this assumption by learning travel time distribution conditioned on the real-time traffic. Wang et al. [WZX14] also attempt to estimate the travel time of a path based on the real-time traffic. They split the query path into multiple sub-paths and search them from the real-time trajectories. However this method seriously suffers from data sparsity issue, even tensor decomposition was used to mitigate it.

Very recently, two deep learning based travel time estimation models, **DeepTTE** [WZC⁺18] and **WDR** [WFY18] are proposed. **DeepTTE** is a Recurrent Neural Net-based model.

In the training stage, it first maps each GPS coordinate in a trajectory into a high-dimensional vector, then it performs 1D-Convolution over the obtained vector sequence with a sliding window, by which they wish the spatial correlation between spatially neighboring points could be captured. In this manner, the original trajectory is transformed into a sequence of high-dimensional vectors and it compresses this sequence with a RNN to get an output. In the end, it matches the output of RNN with the observed travel time to yield the training loss. To account for the impact of real-time traffic, it discretizes the temporal dimension (weather information) into slots that are treated as attributes, and it learns each attribute with a vector representation which is fed into the model. WDR shares the similar idea with DeepTTE in the sense that they all compress a sequence of spatial object (sub-paths, road links) representations with a RNN model. In addition to RNN, WDR also employs a logistic regression model and MLP (Multiple Layer Perceptron) in its model design. A regressor on the top of them combines all their outputs to make the final prediction.

Our method DeepGTT differs from them in three aspects. First, both DeepTTE and WDR only learn to predict the expectation of travel time, whereas our model outputs a probability distribution. Second, DeepTTE and WDR both adopt the assumption that traffic conditions in a time slot are temporally-invariant (except that DeepTTE takes weather into consideration), while we learn the travel time distribution conditioned on the real-time traffic. Third, both of them employ an RNN to learn the travel time by brute force, which unavoidably leads to a data-hungry model. In contrast, our method DeepGTT is built on deep probabilistic generative models which allow us to interpret the generation of travel time in a reasonable manner. As a result, our model DeepGTT not only produces more accurate results but also is data-efficient.

Li et al. [LFW⁺18] study the problem of estimating travel time of a *trip* defined by its origin, destination and departing time, and propose a multi-task-based model MURAT. MURAT partitions the space and time into discrete cells and slots respectively. It learns each cell, slot, and road link on the road networks a high-dimensional representation, respectively, by enforcing spatial and temporal smoothness. Given an origin-destination pair, it first locates their corresponding cell, slot and neighboring road link, whose representations are used to predict the travel time as well as other path summary (e.g. the number of road links, traffic lights) with a multi-task loss function. We highlight that this problem actually is different from ours as we learn the travel time distribution for an entire *route*. Note that, a *trip* can be achieved through multiple routes. In many applications such as trip planning, ride-sharing, travel time estimation between two locations does not help much without knowing the routes to travel.

2.4 Spatial Transition Modeling

The spatial transition modeling on the road network in the literature arises in the following area: sparse trajectory similarity computation, route recovery from sparse trajectories, and future movement prediction.

The low-sampling-rated trajectories bring difficulty for the similarity computation, as an observed sparse trajectory could have multiple possible underlying routes. As mentioned in Section 2.2, Su *et al.* [SZW⁺13a] proposed to capture the spatial transition patterns using HMM. In our proposed **t2vec** (Chapter 3), we attempted to encode into a compact representation the most likely route information of a trajectory.

Route recovery attempts to infer the most likely route between two road segments that are not adjacent on the road network. Zheng *et al.* [ZZXZ12] modeled the spatial transition probability between adjacent road segments with one-order Markov model. Banerjee *et al.* [BRR14] explored the problem with Gibbs sampling, in which the spatial transition probabilities were also modeled with the Markov model albeit high-order ones were employed. Wu *et al.* [WMS⁺16] proposed to use the inverse reinforcement learning to capture the spatial transition patterns. The problem we consider in *Spatial Transition Learning* actually is more general than route recovery from sparse observations, in the sense that **DeepST** can be applied for route recovery while the vice versa may not hold, for two reasons: first, the route recovery attempts to infer the most likely routes for already observed trajectories, in which the travel time is available to help the inference, while in our problem setting we do not require the travel time to be known in advance; second, the route recovery problem assumes the accurate destination road segments are known, whereas **DeepST** is applicable even if only the rough destination coordinates are available.

Xue *et al.* [XZZ⁺] attempted to predict the destination of a trajectory based on its already observed partial sub-trajectory. They partitioned the space into cells and modeled the transition probability between adjacent cells with the first-order Markov Model. Zhao *et al.* [ZZZ⁺18] revisited the problem with RNN. Li *et al.* [LAS15] developed a Bayesian model which is capable of predicting the future movement of a vehicle on the road network. The spatial transition was again modeled with the first-order Markov Model. As the Markov Model requires explicit dependency assumptions and struggles in accounting for the long range dependency, Wu *et al.* [WCS⁺17] explored to model the trajectories with RNN. They treated each road link as a token and squeezed a sequence of road links with a RNN to make the prediction for future movement. They assumed that the exact destination road segments of trips are known and learned the representations of them to help route decision. In contrast, we only assume the rough destination addresses to be available and handle them with a novel adjoint generative model. Several proposals [LWWT16, FLZ⁺18] consider the problem of the next location prediction,

without the constraint of road networks, and thus they are not directly applicable to our scenario.

In addition, the existing methods either ignore the influence of traffic [ZZXZ12, SZW⁺13a, XZZ⁺, WCS⁺17, BRR14] or simply assume that the traffic conditions in the same periodic time slot (*e.g.*, 7-8am weekend) are temporally-invariant [LAS15, WMS⁺16]. To our knowledge, no existing work on spatial transition modeling is based on the real-time traffic.

2.5 Representation Learning

Learning representations for specific tasks has been a longstanding open problem in machine learning. A good representation of data makes it easier to build machine learning models in the downstream tasks. The rapid increase in activity on representation learning is spurred by the revival of deep neural networks. As surveyed in [BCV13], in its early stage, representation learning achieved remarkable successes in the research fields—**Speech Recognition, Object Recognition**. Speech Recognition is one of early application scenarios of neural networks. The development of representation learning has an important new impact on it. Microsoft has published a new version of speech system in 2012 based on deep representation learning [SLY11], which has managed to reduce the word error rate on four major benchmarks by about 30%. Object Recognition. The beginning of deep representation learning in 2006 broke the supremacy of SVMs on the MNIST digit image classification task [HOT06]. Since then the deep representation learning has moved from simple digit images to the object recognition in natural images. It brought down the state-of-the-art error rate from 26.1% to 15.3% on the ImageNet dataset [KSH12].

Inspired by the success of word2vec [MCCD13], the idea of learning general representation has been extended to paragraphs [LM14], social networks [PAS14, TQW⁺15, FCK⁺18], POIs [FLZ⁺15, FCAC17] etc. To capture the sequential order information emerging in the sequence processing tasks, Recurrent Neural Networks (RNNs) based encoder-decoder models have been developed, such as sequence to sequence learning [CvMG⁺14, SVL14, VKK⁺15], and skip-thought vectors [KZS⁺15]. Our model **t2vec** is based on the general sequence encoder-decoder framework. However, these existing sequence encoder-decoder models were initially proposed for natural language processing to deal with discrete tokens (*i.e.*, words, punctuations). Our scenarios is different in that the tokens inherently share the spatial proximity relation. As discussed in the insightful review [BCV13], an appropriate choice of objective is critically important in learning a useful representation for a given task. Simply adopting the loss functions of the original sequence encoder-decoder model makes it impossible to capture this spatial proximity inherent in the trajectory data. Our model **t2vec** therefore differs from the above sequence

encoder-decoders in two ways: i) we design a spatial proximity aware loss function and a cell representation pretraining approach to incorporate the spatial proximity into the deep representation model, and ii) we also propose an approximate loss function based on Noise Contrastive Estimation to accelerate the training.

2.6 Deep Probabilistic Generative Models

Deep probabilistic generative models are the hybrids of deep learning and Bayesian inference, which share the great advantages of two worlds. On the one hand, probabilistic generative models [GCSR95] allow us to specify the relationships among random variables to incorporate our prior knowledge in a flexible way. On the other hand, the automatic-differentiation [BPRS17] permits us to perform inference on large-scale datasets.

Probabilistic generative models have been widely adopted in text data mining [BNJ03], online recommendation [GHB13, GCB14, YCL14], spatial data analytics [YCH⁺11, HAG⁺12, YCM⁺] as well as air quality inference [CLL⁺14b, CLL⁺14a]. They are usually trained with sampling-based methods [PNI⁺08, ADFDJ03] which have the drawback of slow convergence, thus limits their usage to relative small datasets [HBWP13]. The emergence of VAE (Variational Auto-Encoders) [KW13, RMW14] makes training large-scale generative models possible by marrying Bayesian inference and the automatic-differentiation technique—the core ingredient in deep learning. Since then, deep generative probabilistic models have been growing rapidly and achieved a range of state-of-the-art results in semi-supervised learning [KMRW14], multiple instances scene understanding [EHW⁺16], high quality images and audio synthesis [YYSL16, vdOVK17], large-scale online recommendation [LKHJ18]. In contrast, deep probabilistic generative models have been little explored in spatial-temporal data analysis field. To our knowledge, we develop the first two deep generative models (DeepGTT and DeepST) for travel time distribution and spatial transition learning respectively. To make it easy to understand our two proposals, we provide a review on the technique details of Variational Inference and Variational Auto-Encoders as follows.

The probabilistic methods or Bayesian generative methods provide us a principled way to explain the generation of observed data. They permit us to incorporate our prior knowledge regarding data into the model design. Specifically, they offer us two ways to achieve this 1) introducing appropriate latent variables \mathbf{z} to serve as explanatory factors; 2) describing the generation of observed data by specifying proper generative process based on our domain knowledge. Generally, the probabilistic methods can be formulated as follows: we first draw a latent variable \mathbf{z} from a prior distribution $p(\mathbf{z})$, and then relate \mathbf{z} to observation \mathbf{x} through a conditional distribution $p(\mathbf{x}|\mathbf{z})$; lastly, we intend to infer the posterior distribution $p(\mathbf{z}|\mathbf{x})$, which will be used in the prediction stage.

One of main challenges in adopting probabilistic methods lies on the posterior distribution $p(\mathbf{z}|\mathbf{x})$ inference. Using the Bayes rule, we have

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z})p(\mathbf{x}|\mathbf{z})}{p(\mathbf{x})},$$

and the difficulty here is the computation of marginal distribution $p(\mathbf{x}) = \int p(\mathbf{z}, \mathbf{x})d\mathbf{z}$ is intractable in high dimensional space. The traditional inference method Markov Chain Monte Carlo (MCMC) is usually only applicable to small datasets and simple models [KTR⁺17]. An alternative method, Variational Inference (VI), turns the posterior inference into an optimization problem. In comparison to MCMC, VI is much more efficient by taking advantage of numerical optimization algorithms.

In VI, we posit a family of densities \mathcal{Q} and then search the optimal posterior approximation $q^*(\mathbf{z}) \in \mathcal{Q}$ that is closest to $p(\mathbf{z}|\mathbf{x})$ measured by KL divergence, *i.e.*

$$q^*(\mathbf{z}) = \operatorname{argmin}_{q \in \mathcal{Q}} \operatorname{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})).$$

Extending the KL divergence term we get

$$\begin{aligned} \operatorname{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{q(\mathbf{z})} [\log q(\mathbf{z})] - \mathbb{E}_{q(\mathbf{z})} [\log p(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{q(\mathbf{z})} [\log q(\mathbf{z})] - \mathbb{E}_{q(\mathbf{z})} [\log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x}), \end{aligned}$$

from which we can conclude

$$\mathbb{E}_{q(\mathbf{z})} [\log p(\mathbf{z}, \mathbf{x})] - \mathbb{E}_{q(\mathbf{z})} [\log q(\mathbf{z})]$$

is a lower bound of the log-likelihood $\log p(\mathbf{x})$ since KL divergence is nonnegative. This lower bound is commonly known as Evidence Lower Bound (ELBO) in the VI literature, maximizing ELBO is equivalent to minimizing $\operatorname{KL}(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x}))$ or maximizing the log-likelihood $\log p(\mathbf{x})$.

In the past decades, the most widely adopted VI is mean-field VI due to its simplicity. Mean-field VI assumes that the approximated posterior distribution $q(\mathbf{z})$ admits a factorized form as $q(\mathbf{z}) = \prod_{j=1}^{|\mathbf{z}|} q(z_j)$, *i.e.*, all z_j are mutually independent and each z_j is governed by its own factor distribution $q(z_j)$, whose parameters are referred to as variational parameters. Mean-field VI requires us to specify the parametric form for each factor distribution $q(z_j)$, and derive their parameter iterative equations by hand. The drawback is that it constrains us to build models within only a small fraction of probability distributions; otherwise, no parameter iterative equation exists. This implies the resulting models may lack the flexibility to explain the observed data. Moreover, the optimization of mean-field VI often relies on the Coordinate Ascent algorithm which also struggles when the datasets are very large [BKM16].

To address the drawbacks of mean-field VI, Variational Auto-Encoders (VAEs) combine the automatic-differentiation [BPRS17], the core ingredient of deep learning, with variational inference, which yields a flexible yet efficient inference framework for probabilistic generative methods. VAEs replace $q(\mathbf{z})$ with $q(\mathbf{z}|\mathbf{x})$ and rewrite ELBO as

$$\text{ELBO} = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} [\log p(\mathbf{x}|\mathbf{z})] - \text{KL}(q(\mathbf{z}|\mathbf{x}) || \log p(\mathbf{z})), \quad (2.1)$$

and parameterize $q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$ with neural networks, which are referred to as inference network and generative network respectively. More specifically, VAEs assume $q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$ follow two parametric distributions, and fit the parameters of the two distributions with two neural networks. The inference network takes a datapoint \mathbf{x} as input and produces a corresponding latent variable \mathbf{z} ; while the generative network takes \mathbf{z} as input and tries to decode \mathbf{x} . The ELBO serves as a loss function for both inference network and generative network, which is estimated with the Monte Carlo method by drawing L samples, $\mathbf{z}^{(l)}$, from $q(\mathbf{z}|\mathbf{x})$

$$\text{ELBO} \approx \frac{1}{L} \sum_{l=1}^L \log p(\mathbf{x}|\mathbf{z}^{(l)}) - \text{KL}(q(\mathbf{z}|\mathbf{x}) || \log p(\mathbf{z})), \quad (2.2)$$

where the KL term has analytic solution for the Gaussian prior and posterior. The parameters of the two networks are optimized by stochastic gradient descent algorithms which scale to large-scale datasets easily. Once the two networks are optimized on the training dataset, the inference network $q(\mathbf{z}|\mathbf{x})$ will act to define the approximated posterior distribution for any unseen datapoint \mathbf{x} .

Chapter 3

Trajectory Representation Learning

In this chapter, we study the problem of learning representation for trajectory similarity computation¹. We propose the first deep learning approach, **t2vec**, based on seq2seq to learning representations of trajectories that is robust to low data quality, thus supporting accurate and efficient trajectory similarity computation and search. This chapter is organized as follows. We present the overview and summary of contributions in Section 3.1 (the motivation can be found in Section 1.2.1). The problem definition and preliminaries are given in Section 3.2. Section 3.3 presents the details of our method. The experimental results are presented in Section 3.4. We conclude this chapter in Section 3.5.

3.1 Overview and Contributions

In this study, we propose a novel approach, called **t2vec** (**t**rajectory to **v**ector), to inferring and representing the underlying route information of a trajectory based on deep representation learning. The learned representation is designed to be robust to non-uniform and low sampling rates, and noisy sample points for trajectory similarity computation. This is achieved by taking advantage of the archived historical trajectory data and a new deep learning framework. With the learned representation, it only takes a linear time $O(n + |\mathbf{v}|)$ ($|\mathbf{v}|$ is the length of vector \mathbf{v}) to compute the similarity between two trajectories, while all the existing approaches take $O(n^2)$ time. To the best of our knowledge, this is the first deep learning based solution for computing the similarity of trajectories.

To learn trajectory representations, it is natural to consider the use of Recurrent Neural Networks (RNNs), which are able to embed a sequence into a vector. However, simply applying RNNs to embed trajectories is impractical. First, the representation obtained using RNNs is unable to reveal the most likely true route of a trajectory when uncertainty arises due to low sampling rates or noise. Second, the existing loss functions

¹This chapter was published as *Deep Representation Learning for Trajectory Similarity Computation* [LZC⁺18].

used to train RNNs fail to consider spatial proximity, which is inherent in spatial data. Thus, they cannot guide the model to learn consistent representations for trajectories generated by the same route. To overcome the first challenge, we propose a sequence-to-sequence (seq2seq) based model to maximize the probability of recovering the true route of trajectory. To contend with the second challenge, we design a spatial proximity aware loss function and a cell pretraining algorithm that encourage the model to learn consistent representations for trajectories generated from the same route. We also propose an approximate loss function using Noise Contrastive Estimation [GH10] to boost the training speed. Overall, the study makes the following contributions:

- We propose a seq2seq-based model to learn trajectory representations, for the fundamental research problem of trajectory similarity computation. The trajectory similarity based on the learned representations is robust to non-uniform, low sampling rates and noisy sample points. Our solution computes the similarity of two trajectories in linear time.
- For the purpose of learning consistent representations, we develop a new spatial proximity aware loss function and a cell representation learning approach that incorporate the spatial proximity into the deep learning model. To further speed up training, we propose an approximate loss function based on Noise Contrastive Estimation.
- We conduct extensive experiments on two real-world trajectory datasets that offer evidence that the proposed method is capable of outperforming the existing trajectory similarity measure techniques in terms of both accuracy and efficiency.

3.2 Problem Statement and Preliminaries

In this section, we present problem statement and preliminaries essential to understand the problem addressed, *i.e.*, the sequence encoder-decoder model used in our solution.

3.2.1 Problem Statement

In practice, an underlying route can be represented by enormous trajectories, depending on the specifics of the moving objects and the sampling strategies used. Each generated trajectory can be considered as a representative of an underlying route. In the rest of this chapter, a trajectory is also referred to as trip, depending on the context.

Problem statement. Given a collection of historical trajectories, we aim to learn a representation $\mathbf{v} \in \mathbb{R}^n$ (n is the dimension of a Euclidean space) for each trajectory T such that the representation can reflect the underlying route of the trajectory for

computing trajectory similarity. The similarity of two trajectories based on the learned representations must be robust to non-uniform, low sampling rates and noisy sample points.

Our proposed method for solving the problem is based on deep representation learning techniques. Specifically, we adapt the sequence encoder-decoder framework for the first time to compute trajectory similarity (the motivation for adapting that particular framework is explained in Section 3.3.1).

3.2.2 Preliminaries of sequence encoder-decoders

We briefly present the sequence encoder-decoder framework. Consider two sequences $\mathbf{x} = \langle x_t \rangle_{t=1}^{|x|}$ and $\mathbf{y} = \langle y_t \rangle_{t=1}^{|y|}$ where each x_t and y_t denotes token (e.g., a word or punctuation mark in a natural language sentence) and $|\mathbf{x}|$ and $|\mathbf{y}|$ represent lengths. We next illustrate how to build the conditional probability $\mathbb{P}(\mathbf{y}|\mathbf{x})$ in the framework.

The sequence encoder-decoder model has two main components—an encoder and a decoder, as depicted in Figure 3.1. The encoder is used to encode sequence \mathbf{x} into a fixed-dimensional vector \mathbf{v} that preserves the sequential information in \mathbf{x} , and then the decoder decodes out sequence \mathbf{y} conditioned on the encoded representation \mathbf{v} . Since Recurrent Neural Networks (RNNs) [WH86, Wer90] accept input in the form of real-valued vectors, a token embedding layer is added to embed the discrete token in a vector. The token embedding layer is form of a feed forward neural network [BDVJ03].

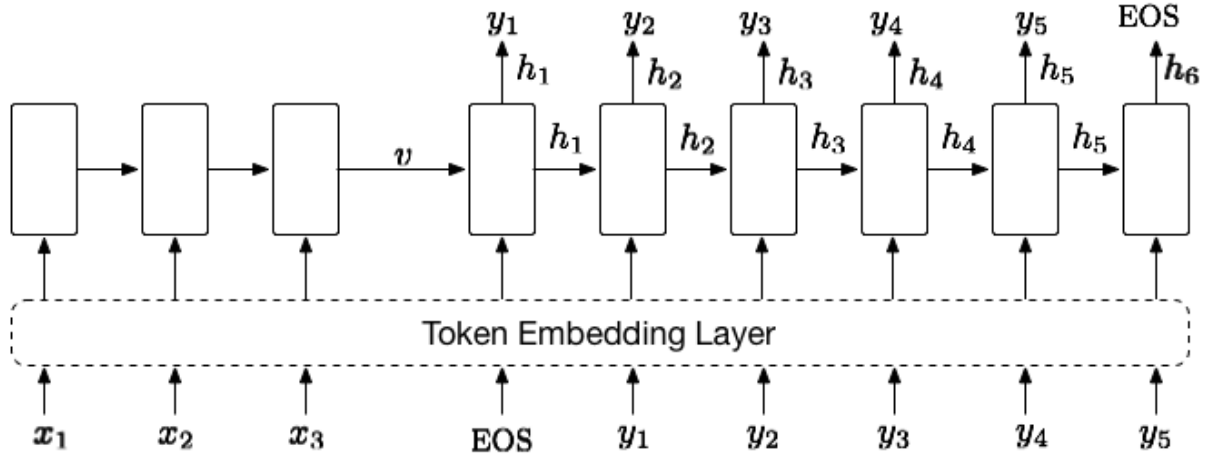


Figure 3.1: Sequence encoder-decoder model. The model reads the sequence \mathbf{x} and outputs the sequence \mathbf{y} , where EOS is a special token indicating the end of a sequence and \mathbf{v} is the representation of \mathbf{x} . The hidden state \mathbf{h}_t captures the sequential information in $[\mathbf{x}, y_1, y_2, \dots, y_{t-1}]$.

To see how the model builds the probability $\mathbb{P}(\mathbf{y}|\mathbf{x})$, we first rewrite it by the chain rule, i.e.,

$$\mathbb{P}(y_1, \dots, y_{|\mathbf{y}|} | x_1, \dots, \mathbf{x}_{|\mathbf{x}|}) = \mathbb{P}(y_1 | \mathbf{x}) \prod_{t=2}^{|\mathbf{y}|} \mathbb{P}(y_t | \mathbf{y}_{1:t-1}, \mathbf{x}),$$

where $\mathbf{y}_{1:t-1}$ represents y_1, y_2, \dots, y_{t-1} . The encoder reads and encodes sequence \mathbf{x} into the fixed-dimensional vector \mathbf{v} . Since \mathbf{v} encodes the sequential information in \mathbf{x} , we have

$$\mathbb{P}(y_t | \mathbf{y}_{1:t-1}, \mathbf{x}) = \mathbb{P}(y_t | \mathbf{y}_{1:t-1}, \mathbf{v}).$$

The decoder builds the probability $\mathbb{P}(y_t | \mathbf{y}_{1:t-1}, \mathbf{v})$ at every position t by squashing \mathbf{v} and $\mathbf{y}_{1:t-1}$ into the hidden state \mathbf{h}_t , which is simply a forward computation. More specifically, \mathbf{h}_t is computed from the output of the previous position \mathbf{h}_{t-1} and token y_{t-1} , as follows:

$$\mathbf{h}_t = \begin{cases} f(\mathbf{v}, \text{EOS}) & t = 1 \\ f(\mathbf{h}_{t-1}, y_{t-1}) & t \geq 2, \end{cases} \quad (3.1)$$

where $f(\cdot, \cdot)$ indicates the RNNs forward computation and EOS is the special token that signals the end of a sequence, which is necessary in order to support variable-length sequences [SVL14].

Note that we recursively compute $\mathbf{h}_t = f(\mathbf{h}_{t-1}, y_{t-1})$: \mathbf{h}_{t-1} encodes the information of \mathbf{v} along with y_1, y_2, \dots, y_{t-2} , and y_{t-1} is further encoded into \mathbf{h}_t along with y_1, y_2, \dots, y_{t-2} . Finally, $\mathbb{P}(y_t | \mathbf{y}_{1:t-1}, \mathbf{v})$ can be modeled as follows.

$$\mathbb{P}(y_t = u | \mathbf{y}_{1:t-1}, \mathbf{v}) = \mathbb{P}(y_t = u | \mathbf{h}_t) = \frac{\exp(W_u^\top \mathbf{h}_t)}{\sum_{v \in \mathcal{V}} \exp(W_v^\top \mathbf{h}_t)}.$$

Here, W^\top is the projection matrix that projects \mathbf{h}_t from the hidden state space into the vocabulary space, W_u^\top denotes its u -th row, and \mathcal{V} is the vocabulary.

3.3 Proposed Method

We first present the underlying motivation for the proposed method and the challenges to be addressed when developing it. Then we describe how to handle non-uniform and low sampling rates, as well as noisy sample points in our model. In Section 3.3.3, we discuss the details of the proposed spatial proximity aware loss function and cell representation learning approach in order to encourage the sequence encoder-decoder to learn consistent representations for trajectories. We discuss the time complexity of using our model to compute trajectory similarity in Section 3.3.4.

3.3.1 Motivations and challenges

Recent work [ZZXZ12, BRR14] reveals that the transition patterns between road network locations are often highly skewed. This implies that some routes have higher probability of being traveled than others. Rich movement patterns have been archived in spatiotemporal databases (we have 20 different data sources at Daisy² and different data sources have different sampling rates), which then afford us the opportunity to learn representations for trajectories that overcome the shortcomings of the point-matching methods.

Recall that we intend to learn a representation $\mathbf{v} \in \mathbb{R}^n$ for a trajectory T that is robust to non-uniform, low sampling rates and noisy sample points when using it to compute trajectory similarity. To learn a representation for sequential data, it is very natural to consider RNNs [WH86, Wer90] since these have been shown successful at handling sequences in natural language processing, including generating sequences [Gra13], neural machine translation [SVL14, VKK⁺15] and paraphrases detection [KZS⁺15]. However, by simply applying RNNs we will not be able to accomplish our goal, which is due to the two difficulties explained in Section 3.1.

To tackle the first difficulty, the desired model must be able to maximize the conditional probability $\mathbb{P}(\mathbf{r}|T)$, i.e., finding the most likely underlying route \mathbf{r} for the given trajectory T . However, the underlying route \mathbf{r} is often not available in practice. To circumvent this, we exploit two observations: i) both a non-uniform, relatively low sampling rate trajectory, denoted by T_a , and a relatively high sampling rate trajectory, denoted by T_b , are paraphrases of their underlying route, and ii) a relatively high sampling rate trajectory T_b is closer to their true underlying route \mathbf{r} than is T_a , and it has lower uncertainty. These observations cause us to replace the objective of maximizing $\mathbb{P}(\mathbf{r}|T_a)$ into the objective of maximizing $\mathbb{P}(T_b|T_a)$ and to build the model using a sequence encoder-decoder framework. The encoder embeds T_a into its representation \mathbf{v} , and the decoder will try to recover its counterpart T_b with relatively high sampling rate conditioned on \mathbf{v} by optimizing its parameters. When the model is trained using the real-world trajectories, the transition patterns hidden in historical data will be learned by the model. To overcome the second difficulty, we propose a new spatial proximity aware loss function and cell (token) representation pre-training method to incorporate spatial proximity into the model. To accelerate the training, we develop an approximate spatial proximity aware loss function based on Noise Contrastive Estimation [GH10].

3.3.2 Handling varying sampling rates and noise

Based on the above analysis, given a collection of sampling rate trajectories $\{T_b^{(i)}\}_{i=1}^N$ (where N is the cardinality of the collection), we create a collection of pairs (T_a, T_b) ,

²<http://www.daisy.aau.dk/>

where T_b is an original trajectory and T_a is obtained by randomly dropping sample points from T_b with dropping rate d_1 . By doing so, each down-sampled T_a is also non-uniformly sampled and thus represents a real-life trajectory with non-uniform and low sampling rate. The start and end points of T_b are preserved in T_a to avoid changing the underlying route of the down-sampled trajectory. To illustrate, consider generating the sub-trajectories for T_b in Figure 1.1b, we randomly drop points in $b_{2:5}$, i.e., all generated sub-trajectories will start with b_1 and end with b_6 . After the generating procedure, we maximize the joint probability of all (T_a, T_b) pairs with the sequence encoder-decoder model:

$$\text{maximize} \quad \prod_{i=1}^N \mathbb{P}(T_b^{(i)} | T_a^{(i)}). \quad (3.2)$$

In the sequence encoder-decoder model, the inputs should be sequences of discrete tokens. Therefore, we need to find a way to map the continuous coordinates (i.e., longitude, latitude) into discrete tokens (analogous to words in natural language). We adopt a simple strategy that is used commonly in spatial data analytics, i.e., we partition the space into cells of equal size [GS95] and treat each cell as a token. All sample points falling into the same cell are then mapped to the same token.

The above helps mainly to overcome the problems of non-uniform and low sampling rates. However, the realistic trajectories also may have noisy sample points. For example, when a GPS receiver is in an urban canyon and satellite visibility is poor, inaccurate locations may result. To eliminate the influence of noisy sample points, we only keep the cells which are hit by more than δ sample points. These cells are referred to as hot cells and form the final vocabulary \mathcal{V} (in the rest of chapter, we will interchangeably use token and cell to refer to an element \mathcal{V}). Sample points are represented by their nearest hot cell. To make the learned representations more robust to the noisy data, we further distort each downsampled T_a based on a distorting rate d_2 to create the distorted variants, i.e., we randomly sample a fraction of the points (size indicated by d_2) that are then distorted. Point (p_x, p_y) is distorted by adding a Gaussian noise with a radius 30 (meters) as follows,

$$\begin{aligned} p_x &= p_x + 30 \cdot d_x, & d_x &\sim \text{Gaussian}(0, 1) \\ p_y &= p_y + 30 \cdot d_y, & d_y &\sim \text{Gaussian}(0, 1). \end{aligned} \quad (3.3)$$

We can optimize the same objective as shown in Equation 3.2 where T_a is both down-sampled and distorted.

3.3.3 Learning consistent representations

The original sequence encoder-decoder does not model the spatial correlation between cells, which is important in order to learn consistent representations for trajectories drawn from the same route. To address this, we propose a novel spatial proximity aware loss function (in Section 3.3.3.1) and a new cell representation pretraining approach that takes into account spatial proximity (in Section 3.3.3.2). To further improve the training, an approximate loss function based on Noise Contrastive Estimation [GH10] is also proposed (in Section 3.3.3.1).

3.3.3.1 Spatial proximity aware loss function

To train a sequence encoder-decoder, we need a loss function to characterize the optimization objective. This is important, as differences in the loss function would encourage the model to learn different representations [BCV13]. When the sequence encoder-decoder is employed in natural language processing, e.g., in neural machine translation [CvMG⁺14, SVL14, BCB14b], Negative Log Likelihood (NLL) loss is chosen to minimize the negative log likelihood function for tokens in the target sentence as follows,

$$\mathcal{L}_1 = -\log \prod_t \mathbb{P}(y_t | y_{1:t-1}, x). \quad (3.4)$$

However, simply adopting this loss function is problematic for the spatiotemporal data. Recall that our original purpose is to maximize $\mathbb{P}(\mathbf{r} | T_a)$. Due to the unavailability of the underlying route \mathbf{r} , we use the original trajectory T_b to represent \mathbf{r} and instead maximize $\mathbb{P}(T_b | T_a)$. In practice, even original trajectories may not cover their underlying route \mathbf{r} well. For example, Figure 3.2 has two trajectories T_b and $T_{b'}$ generated from the same route \mathbf{r} (after transforming the coordinates to cells, their corresponding sequences are \mathbf{y} and \mathbf{y}' , respectively). The sample points of the two trajectories interleave the cells in our space partitioning. Let T_a and $T_{a'}$ denote the sub-trajectories of T_b and $T_{b'}$, respectively. Ideally, the representations learned for T_a and $T_{a'}$ should be similar, as they are both generated from route \mathbf{r} . The NLL loss function in Equation 3.4 differentiates T_b and $T_{b'}$ as two identical target trajectories, so that it cannot discover the similarity between T_a and $T_{a'}$.

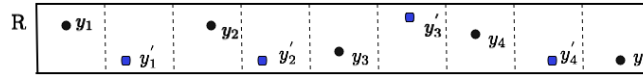


Figure 3.2: T_b and $T_{b'}$ are two trajectories generated from an underlying route \mathbf{r} . After transforming the coordinates to cells, their corresponding sequences are \mathbf{y}, \mathbf{y}' respectively. The sample points in the two trajectories interleave on the route.

The reason is that the loss function in Equation 3.4 penalizes the output cells with equal weight. Intuitively, the output cells that are closer to the target should be more acceptable than those that are far away. For example, if the decoded target cell is y_3 (in Figure 3.2), the loss function penalizes the outputs y'_3 and y_1 equally. This is not a good penalty strategy. Since y'_3 is spatially closer to y_3 , it is more acceptable for the decoder to output y'_3 rather than to output y_1 .

The intuition behind the proposed spatial proximity aware loss function is that we assign a weight for each cell when we try to decode a target cell y_t from the decoder. The weight of cell $u \in \mathcal{V}$ is inversely proportional to its spatial distance to the target cell y_t , so the closer the cell is to y_t the larger weight we will assign to it. The spatial proximity aware loss is given as follows.

$$\mathcal{L}_2 = - \sum_{t=1}^{|y|} \sum_{u \in \mathcal{V}} w_{uy_t} \log \frac{\exp(W_u^\top \mathbf{h}_t)}{\sum_{v \in \mathcal{V}} \exp(W_v^\top \mathbf{h}_t)}, \quad (3.5)$$

where

$$w_{uy_t} = \frac{\exp(-\|u - y_t\|_2 / \theta)}{\sum_{v \in \mathcal{V}} \exp(-\|v - y_t\|_2 / \theta)}$$

is the spatial proximity weight for cell u when decoding target y_t , and $\|u - y_t\|_2$ denotes the Euclidean distance between the centroid coordinates of the cells. Here $\theta > 0$ is a spatial distance scale parameter. A small θ penalizes far away cells heavily, and when $\theta \rightarrow 0$, the loss function will be reduced to the NLL loss function in Equation 3.4. The exponential kernel function is chosen as it decays fast at the tail, which would encourage the model to learn to output cells near the target cell y_t .

Although the spatial proximity aware loss function in Equation 3.5 helps us learn consistent representations for trajectories generated from the same routes, it requires us to sum over the entire vocabulary twice every time we decode a target y_t :

$$\sum_{u \in \mathcal{V}} w_{uy_t} \underbrace{\left(W_u^\top \mathbf{h}_t - \sum_{v \in \mathcal{V}} \exp(W_v^\top \mathbf{h}_t) \right)}_{\log \text{ probability}}. \quad (3.6)$$

Thus, the cost of decoding a trajectory \mathbf{y} is $O(|\mathbf{y}| \times |\mathcal{V}|)$. When the vocabulary size $|\mathcal{V}|$ is large, it will be expensive to train the model.

Approximate spatial proximity aware loss function. To reduce the training cost, we design an approximate spatial proximity aware loss function based on the following two observations: i) most of w_{uy_t} are very small except cells that are close to target cell y_t ; ii) it is not necessary to calculate the exact value of the log probability in Equation 3.6, as long as we can encourage the decoder to assign the probability to the cells that are close

to the target cell. Based on the first observation, we can use just the k nearest cells of y_t , denoted as $\mathcal{N}_k(y_t)$, instead of using the whole vocabulary in the first sum in Equation 3.6. Based on the second observation, we can use Noise Contrastive Estimation (NCE) [GH10] to compute the log probability in Equation 3.6. NCE was developed by Gutmann et al. [GH10] to differentiate data from noise by training a logistic regression. We can use it to approximately maximize the log probability of cells in $\mathcal{N}_k(y_t)$ by randomly sampling a small set of cells from $\mathcal{V} - \mathcal{N}_k(y_t)$ as noise data, denoted as $\mathcal{O}(y_t)$. In our experiments, we find that 500 randomly sampled noise cells can give a very good approximation, and thus the time complexity is reduced from $O(|\mathbf{y}| \times |\mathcal{V}|)$ to $O(|\mathbf{y}|)$. In summary, our approximate spatial proximity aware loss is given as follows.

$$\mathcal{L}_3 = - \sum_{t=1}^{|\mathbf{y}|} \sum_{u \in \mathcal{N}_k(y_t)} w_{uy_t} \left(W_u^\top \mathbf{h}_t - \sum_{v \in \mathcal{NO}} \exp(W_v^\top \mathbf{h}_t) \right), \quad (3.7)$$

where

$$w_{uy_t} = \frac{\exp(-\|u - y_t\|_2 / \theta)}{\sum_{v \in \mathcal{N}_k(y_t)} \exp(-\|v - y_t\|_2 / \theta)}$$

$$\mathcal{NO} = \mathcal{N}_k(y_t) \cup \mathcal{O}(y_t).$$

3.3.3.2 Pre-training cell representations

To further guarantee that the trajectories generated by the same route have close representations in the latent space, we propose a cell representation learning algorithm to pre-train the cells in the embedding layer of the model. The intuition is that the encoder squeezes a sequence of cells covered by the trajectory to get the trajectory representation \mathbf{v} , and thus the representations of two trajectories along the same route will be close in their latent space if we can learn similar representations for cells that are spatially close. For example, if each y_i has a similar cell representation as that of y'_i in Figure 3.2, the trajectory representations of T_a and T'_a will be close in the latent space since they are encoded by the same encoder.

Two straightforward representations exist for the cells, the one-hot representation [BCV13] and the centroid coordinates of the cells (GPS coordinates). However, both representations have limitations. The one-hot representation loses the spatial distance relation of the cells as all the cells are treated independently. As a result, the proposed model may take more training time to discover spatial relations in the cell embedding layer. It would help accelerate the training if the input cell representations provide the prior knowledge of spatial proximity. Next, the centroid coordinates of the cells naturally encode the spatial proximity for the cells but restrict the representations in a two-dimensional space,

which make it difficult for the loss function to further optimize the representations in their parameter space.

Based on the above analysis, we propose to feed the distributed cell representations, which capture the cell spatial proximity relation, to the embedding layer of the model. We achieve this by borrowing the key idea from skip-grams [MCCD13]. The intuition behind skip-grams is that words with similar meanings tend to appear together in the same contexts, and if we use the representation of a word to predict its surrounding words then we can embed the words into a Euclidean space in a way that captures the semantic distances between the words. Towards this end, we create the context for a given cell $u \in \mathcal{V}$ by randomly sampling its neighbor $u' \in \mathcal{N}_k(u)$ (we also only consider its k -nearest neighbors) according to the following cell sampling distribution:

$$\mathbb{P}(u') = \frac{\exp(-\|u' - u\|_2/\theta)}{\sum_{v \in \mathcal{N}_k(u)} \exp(-\|v - u\|_2/\theta)}. \quad (3.8)$$

The cell sampling distribution is similar to the spatial proximity weight in Equation 3.5. Note that their θ values do not have to be equal. For each cell $u \in \mathcal{V}$, the cell sampling distribution tends to sample the cells that are spatially close to it as its context. In this fashion, we are able to create the context for each cell and to learn the cell representation efficiently with the negative sampling algorithm [MSC⁺13] by maximizing the log probability of observing the neighboring cells in its context, $\mathcal{C}(u)$, given cell u :

$$\text{maximize} \quad \sum_{u \in \mathcal{V}} \log \mathbb{P}(\mathcal{C}(u)|g(u)). \quad (3.9)$$

Here, $g(u)$ denotes the representation of cell u .

The learned cell representations will be used to initialize the embedding layer in the model, but we do not fix their values. Thus they can still be further optimized by the loss function in Equation 3.7. The learning algorithm is shown in Algorithm 1.

Algorithm 1: CellLearning (CL)

Input: The dimension of the learned representations d , context window size l

Output: The learned cell representations $g(u)$

```

1 for  $u \in \mathcal{V}$  do
2    $\mathcal{C}(u) \leftarrow \emptyset$ ;
3   while  $|\mathcal{C}(u)| < l$  do
4      $u' \sim \mathbb{P}(u')$  according to Equation 3.8;
5      $\mathcal{C}(u) \leftarrow \mathcal{C}(u) \cup u'$ ;
6 Optimizing the loss function in Equation 3.9;
7 return  $g(u)$  for  $u \in \mathcal{V}$ ;
    
```

3.3.4 Complexity of similarity computation

Our model can be trained completely unsupervised with the SGD (Stochastic Gradient Descent) algorithm. Given a trained model, it only requires $O(n)$ time (as shown in Equation 3.1, where $f(\cdot, \cdot)$ indicates the encoder-RNN and \mathbf{h}_0 is a zero vector) to embed a trajectory into a vector which is fast and can be done using GPUs. Then we can use the Euclidean distance of the two vectors to measure the similarity of two trajectories, with a time complexity of $O(|\mathbf{v}|)$. Therefore the time complexity of measuring the similarity between two trajectories is $O(n + |\mathbf{v}|)$.

3.4 Experiments

We study the effectiveness and scalability of our proposed method on two real-world taxi datasets. The experimental setup and parameter settings are presented in Sections 3.4.1 and 3.4.2 respectively. Then we evaluate the accuracy of different methods using most similar search, cross-similarity, and k -nn queries in Sections 3.4.3.1 to 3.4.3.3, respectively. Scalability is covered in Section 3.4.4. The proposed loss functions and cell learning approach are evaluated in Section 3.4.5. We end by evaluating the impact of the cell size, the hidden state size of the encoder, and the training data size on the learned trajectory representations in Sections 3.4.6 and 3.4.7.

3.4.1 Experimental setup

Dataset. The experiments are conducted on two real-world taxi datasets. The first dataset³ is collected in the city of Porto, Portugal over 19 months and contains 1.7 million trajectories. Each taxi reports its location at 15 second intervals. We remove trajectories with length less than 30, which yields 1.2 million trajectories. The second dataset contains trajectories collected from 13,000 taxis over 8 months in Harbin, China. We select trajectories with length at least 30 and time gaps between consecutive sample points being less than 20 second. This yields 1.5 million trajectories. We partition both sets into training data and testing data based on the starting timestamp of the trajectories. For both sets, the first 0.8 million trajectories are used for training, and the remaining trajectories are used for testing. Statistics of the two sets are shown in Table 5.1.

To create the training trajectory pairs as described in Section 3.3.1, we perform two kinds of transformations, down-sampling and distortion. For each trajectory T_b we first down-sample it with a dropping rate d_1 varied in $[0, 0.2, 0.4, 0.6]$ to create its 4 sub-trajectories T_a . And we further distort each down-sampled T_a based on a distorting rate

³<http://www.geolink.pt/ecmlpkdd2015-challenge>

Table 3.1: Dataset statistics.

Dataset	#Points	#Trips	Mean length
Porto	74,269,739	1,233,766	60
Harbin	184,809,109	1,527,348	121

d_2 (as described in Equation 3.3) varied in $[0, 0.2, 0.4, 0.6]$. As a result, 16 training pairs (T_a, T_b) are created for each original trajectory T_b .

Benchmarking Methods: We compare **t2vec** with three other methods for measuring the trajectory similarity, namely **EDR** [CÖO05], **LCSS** [VKG02], and **EDwP** [RDT⁺15]. **LCSS** and **EDR** are two of the most widely adopted trajectory similarity measures in spatiotemporal data analyses. **EDwP** is the state-of-the-art method for measuring similarity of non-uniform and low sampling rate trajectories. We do not include **DTW** as it has been demonstrated to be consistently inferior to **EDR** in trajectory similarity computation [RDT⁺15]. Moreover, we compare with the vanilla RNN (**vRNN**) [CGCB14a] and the common set representation (**CMS**). The vanilla RNN serves as an embedding baseline method, and the common set representation is used to measure the similarity of two trajectories based on their common set after they have been mapped to cells. We discuss the reasons for comparing with the two baselines in Section 3.4.3.1.

Evaluation Platform: Our method⁴ is implemented in Julia [BEKS17] and PyTorch, and trained using a Tesla K40 GPU. The baseline methods are written in Java⁵. The platform runs the Ubuntu 14.04 operating system with an Intel Xeon E5-1620 CPU.

3.4.2 Parameter settings and training details

Cell size: The default cell size in the experiments is 100 meters. After removing the cells hit by less than 50 points (i.e., $\delta = 50$), we get 18,866 hot cells for the Porto dataset and 22,171 hot cells for the Harbin dataset.

RNN units: In our model, GRU [CGCB14a] with 3 layers is chosen as the computational unit because it has been shown to be as good as LSTM [HS97] in sequence modeling tasks, while it is much more efficient to compute [CGCB14a].

Gradient clipping: Although RNNs tend not suffer from gradient vanishing problem, they may have exploding gradients [PMB13]. Hence, we clip the gradients by enforcing a maximum gradient norm constraint [Gra13], which is set to 5 in our experiments.

Terminating condition: We randomly select 10,000 trajectories as a validation dataset from the test dataset (the selected trajectories are removed from the test dataset). The

⁴<https://github.com/boathit/research-papers/tree/master/t2vec>

⁵The authors of **EDwP** give us access to their compiled jar file.

training is terminated if the loss in the validation dataset does not decrease in 20,000 successive iterations.

In addition, both the hidden layer size in GRU and the dimension of the learned cell representation d are set to 256, the context window size l in the cell learning algorithm is set to 10. For simplicity, θ in Equations 3.5 and 3.8 is fixed at 100 (meters). Parameter k and the size of $\mathcal{O}(y_t)$ in Section 3.3.3 are set to 20 and 500, respectively. We adopt Adam stochastic gradient descent [KB14a] with an initial learning rate of 0.001 to train the model. We evaluate the training time in Sections 3.4.5 and 3.4.6.

To set the parameter ϵ of the baseline methods **EDR** and **LCSS**, we adopt the strategies described in the studies [CÖO05, VKG02] proposing the two methods; the parameters of **vRNN** are set to be the same as our encoder-RNN except that it is trained by predicting the next cell based on the cells that it has already seen.

3.4.3 Performance evaluation

The lack of ground-truth dataset makes it a challenging problem to evaluate the accuracy of trajectory similarity. Two recent studies [SZW⁺13a, RDT⁺15] propose to evaluate the accuracy of methods for computing trajectory similarity using the self-similarity and cross-similarity comparisons and assessments of the precision of finding the k -nearest neighbors. Currently, this is the best evaluation methodology, and we adopt this methodology in the experiments. In addition, we also design a new experiment, called most similar search (which can be considered as a sort of self-similarity measure) to evaluate the effectiveness of different methods, in Section 3.4.3.1.

One of the most important tasks in trajectory analysis is similar trajectory search. To overcome the lack of ground-truth, we design three experiments to evaluate the performance using different methods for this task.

We randomly choose 10,000 trajectories from the test dataset, denoted as Q , and then we choose another m (a parameter to be evaluated) trajectories, denoted by P . For each trajectory $T_b \in Q$, we create two sub-trajectories from it by alternately taking points from it, denoted as T_a and $T_{a'}$ (see Figure 3.3), and we use them to construct two datasets $D_Q = \{T_a\}$ and $D'_Q = \{T_{a'}\}$. We perform the same transformation for the trajectories in P to get D_P and D'_P . Then for each query $T_a \in D_Q$, we retrieve its top- k most similar trajectories from database $D'_Q \cup D'_P$ and calculate the rank of $T_{a'}$. Ideally $T_{a'}$ is ranked at the top since it is generated from the same original trajectory as T_a . The reason for using $D'_Q \cup D'_P$ as the database instead of $D'_Q \cup P$ is that the query trajectory will have similar mean length as the trajectories in the database⁶. Moreover, to evaluate whether RNN encodes two sequences of cells into two similar vectors simply because the two sequences

⁶Similar results were found using database $D'_Q \cup P$.

Table 3.2: Mean rank versus the database size using the Porto and Harbin datasets.

Porto					
DB size	20k	40k	60k	80k	100k
EDR	25.73	50.70	76.07	104.01	130.98
LCSS	31.95	59.20	95.85	130.40	150.67
CMS	62.18	112.84	173.34	231.55	291.26
vRNN	32.73	61.24	100.20	135.22	163.10
EDwP	6.78	11.48	16.08	23.02	28.90
t2vec	2.30	3.45	4.73	6.35	7.67

Harbin					
DB size	20k	40k	60k	80k	100k
EDR	30.37	57.90	85.72	118.02	149.01
LCSS	35.49	63.20	105.46	137.20	160.67
CMS	97.41	141.04	209.37	271.45	316.81
vRNN	34.30	65.24	103.05	140.25	162.10
EDwP	12.80	20.64	29.10	35.20	45.30
t2vec	5.10	7.50	9.62	12.51	15.70

have the same starting or ending cells, or just because they have sufficient numbers of common cells, we include another two baselines, **vRNN** and **CMS**. If the aforementioned reason is true, **vRNN** and **CMS** should also give good performance in the task.

3.4.3.1 Most similar trajectory search

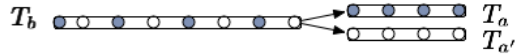


Figure 3.3: Creating two sub-trajectories T_a and $T_{a'}$ from trajectory T_b by alternately taking points from it.

Experiment 1 We first study the performance of the different methods when we increase m , the size of P , from 20,000 to 100,000. Table 3.2 shows the mean rank of the 10,000 queries in D_Q using different methods when using the Porto and Harbin datasets. As the size of P grows, the performance of all methods degrades. **CMS** performs the worst, as it ignores the sequential information in the trajectories. **vRNN** and **LCSS** demonstrate similar performance, and this is reasonable because both methods can be considered as enhanced version of **CMS** that preserve the order of the points in the sequence. **EDwP**

Table 3.3: Mean rank versus d_1 using the Porto and Harbin datasets.

Porto					
d_1	0.2	0.3	0.4	0.5	0.6
EDR	160.03	208.01	235.60	285.10	340.68
LCSS	168.02	173.45	187.60	188.40	192.20
CMS	296.56	317.70	430.00	387.90	446.50
vRNN	173.45	179.58	190.24	200.13	210.20
EDwP	29.10	30.50	31.64	39.67	61.72
t2vec	7.88	8.00	9.48	12.70	15.99

Harbin					
d_1	0.2	0.3	0.4	0.5	0.6
EDR	183.02	231.01	265.50	316.30	380.86
LCSS	178.80	193.50	195.30	208.40	210.30
CMS	330.70	376.20	450.04	460.90	476.50
vRNN	176.45	184.83	191.42	203.13	250.20
EDwP	47.32	49.80	51.39	57.91	81.81
t2vec	15.92	17.21	19.87	21.74	30.95

Table 3.4: Mean rank versus d_2 using the Porto and Harbin datasets.

Porto					
d_2	0.2	0.3	0.4	0.5	0.6
EDR	132.40	133.10	135.60	134.90	139.10
LCSS	210.30	215.70	214.60	215.05	228.03
CMS	296.16	317.27	337.31	327.90	346.05
vRNN	212.16	220	217.30	220.61	235.70
EDwP	30.10	30.16	32.63	31.23	33.53
t2vec	9.10	9.20	9.52	9.49	10.80

Harbin					
d_2	0.2	0.3	0.4	0.5	0.6
EDR	142.44	143.69	145.67	146.90	152.11
LCSS	230.32	235.93	244.65	245.15	251.60
CMS	306.62	327.87	329.31	339.34	349.51
vRNN	222.41	227.20	236.37	250.62	245.75
EDwP	46.41	47.97	49.32	50.68	51.10
t2vec	16.43	17.28	17.52	18.51	21.08

performs the best among all baseline methods. **t2vec** outperforms the other methods significantly, and even when the size of database P reaches 100,000, it gives a low mean rank for the queries.

Experiment 2 Next, we study the impact of down-sampling on the methods with a fixed database size $|D'_Q \cup D'_P| = 100,000$. We vary the dropping rate d_1 from 0.2 to 0.6 and down-sample the trajectories in both D_Q and $D'_Q \cup D'_P$ based on the dropping rate. Table 3.3 depicts the mean rank for the queries in D_Q of the different methods using Porto and Harbin datasets. **EDR** degrades quickly when we increase the dropping rate d_1 , while **LCSS** and **vRNN** are not very sensitive to variations in d_1 . **EDwP** shows relatively consistent performance when the down-sampling rate d_1 varies between 0.2 and 0.5, but when raising d_1 to 0.6, its mean rank increases markedly. This implies that the linear interpolation may not be able to handle a low sampling rate effectively if the dropping rate is large. The reason is that the assumption made by the linear interpolation that the object would move along a straight line between two consecutive sample points is no longer true. **t2vec** consistently outperforms the other methods by a large margin.

Experiment 3 Finally, we evaluate the effect of noise on results. We still fix $|D'_Q \cup D'_P| = 100,000$ and distort the points of trajectories in both query D_Q and database $D'_Q \cup D'_P$ with distorting rate d_2 , as described in Equation 3.3. The results are shown in Table 3.4 using the Porto and Harbin datasets. Unlike with down-sampling, we observe that all methods are not very sensitive to point distortion. Even if the distorting rate is set to 0.6, no method shows obvious performance degradation. We observe that **t2vec** achieves the best performance.

In subsequent experiments, we observe similar results on the two datasets, and we only report the results when using the Porto dataset.

3.4.3.2 Cross-similarity comparison

A good similarity measure should be able to not only recognize the trajectory variants of the same underlying route (self-similarity), but should also preserve the distance between two different trajectories, regardless of the sampling strategy. We adopt an evaluation criterion from the literature [SZW⁺13b, WSZ⁺13], namely cross distance deviation, which is calculated as follows:

$$\frac{|d(T_a(r), T_{a'}(r)) - d(T_b, T_{b'})|}{d(T_b, T_{b'})},$$

where T_b and $T_{b'}$ represent two distinct original rate trajectories, and $T_a(r)$ and $T_{a'}(r)$ are their variants obtained by randomly dropping (or distorting) sample points with the dropping (or distorting) rate r . We randomly select 10,000 trajectory pairs $(T_b, T_{b'})$ from the test dataset to calculate their mean cross distance deviation. A small cross distance deviation indicates that the evaluated distance is much close to the ground truth. The

Table 3.5: Mean cross-distance deviation for varying dropping rate d_1 and distorting rate d_2 .

Porto				
d_1	0.1	0.2	0.4	0.6
t2vec	0.057	0.010	0.016	0.025
EDwP	0.059	0.010	0.024	0.039
EDR	0.130	0.190	0.380	0.580

Porto				
d_2	0.1	0.2	0.4	0.6
t2vec	0.010	0.013	0.018	0.021
EDwP	0.010	0.018	0.031	0.038
EDR	0.012	0.019	0.033	0.039

mean cross distance deviation of the different methods is described in Table 3.5, where we vary the dropping rate d_1 and the distorting rate d_2 . We notice that **t2vec** outperforms the other two methods in terms of cross distance deviation for different dropping and distorting rates.

3.4.3.3 k -nn queries

The similarity measure is a fundamental operation that can be used in many applications, e.g., similarity search, clustering, and classification. In this experiment, we evaluate the performance of different similarity search methods. To contend with the lack of ground-truth, we follow the methodology used in previous work [SZW⁺13a, RDT⁺15]. We first randomly choose 1000 trajectories as query and 10,000 trajectories as the target database from the test dataset. We apply each method to find the k -nearest-neighbors (k -nn) of each query trajectory from the target database as its ground-truth. Next, we transform queries and database trajectories by randomly dropping (resp. distorting) certain sample points according to the dropping (resp. distorting) rate d_1 (resp. d_2). Finally, for each transformed query, we find its k -nn from the target database using each method and then compare the result with the corresponding ground-truth. The rationale behind this methodology is that a robust distance measure should adapt to non-uniform and relatively low sampling rates (resp. distortion) and yield results close to those for relatively high sampling rate (resp. non-distorted) counterparts.

Figure 3.4 shows the precision (the proportion of true k -nn trajectories) of the different similarity methods when the dropping rate (as shown in Figure 3.4(a)-3.4(c)) and the distorting rate (as shown in Figure 3.4(d)-3.4(f)) is varied for $k = 20, 30, 40$. The

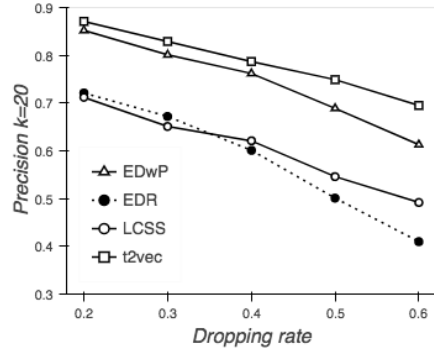
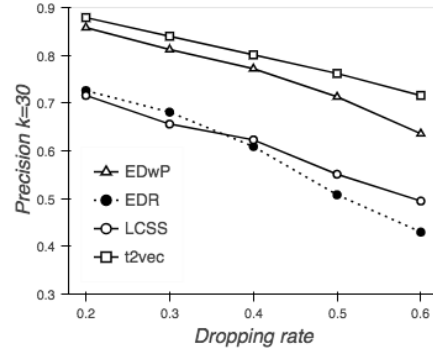
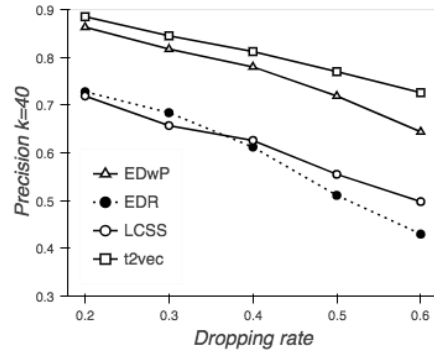
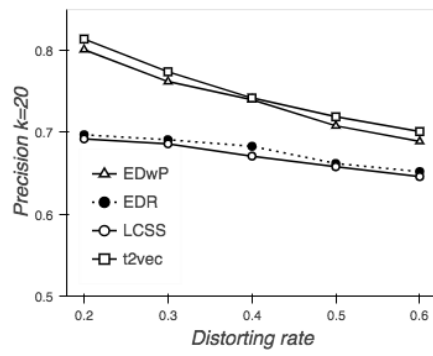
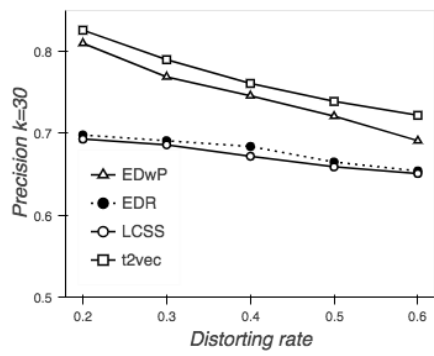
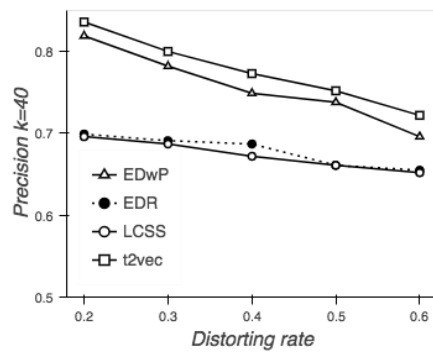
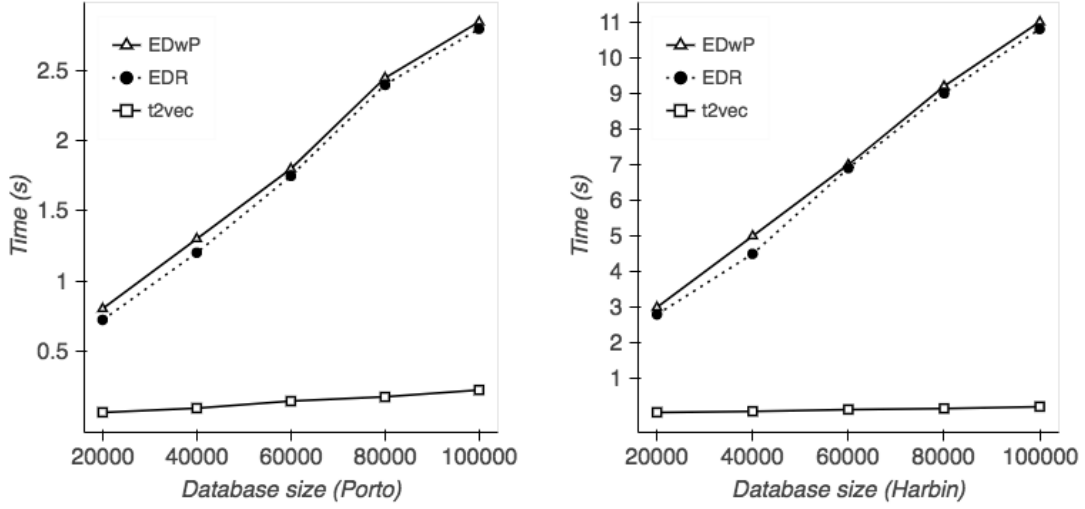

 (a) $k = 20$

 (b) $k = 30$

 (c) $k = 40$

 (d) $k = 20$

 (e) $k = 30$

 (f) $k = 40$

Figure 3.4: (a)-(c) k -nn results when varying the dropping rate for $k = 20, 30, 40$. (d)-(f) k -nn results when varying the distorting rate for $k = 20, 30, 40$.



(a) k -nn query efficiency versus database size using the Porto dataset.

(b) k -nn query efficiency versus database size using the Harbin dataset.

Figure 3.5: (a) k -nn query efficiency versus database size using the Porto dataset. (b) k -nn query efficiency versus database size using the Harbin dataset.

precision of all methods decreases when the dropping rate or distorting rate increases. EDR and LCSS show similar performance, but the precision of EDR drops rapidly when the dropping rate reaches 0.6. EDwP surpasses them by a fair magnitude, and t2vec consistently performs the best.

3.4.4 Scalability

The time complexity of LCSS and EDR for determining the similarity of two trajectories T_a, T_b is $O(|T_a| \times |T_b|)$. EDwP has complexity $O((|T_a| + |T_b|)^2)$. These methods rely on intricate pruning techniques [CÖ05, RDT⁺15] to answer k -nn queries on large datasets. As discussed in Section 3.3.4, once our model has been trained offline, it takes linear time to encode a trajectory into a vector \mathbf{v} , and the encoding process can also be done offline. As an example, we can encode the 1.7 million trajectories in the Porto dataset into vectors within 30 minutes using one Tesla K40 GPU. After encoding the trajectories into vectors offline, its online complexity is $O(|\mathbf{v}|)$. The linear complexity makes t2vec scale well on large datasets. Note that model training is done offline; we will study the training time in Section 3.4.5.

Although it is obvious that our method is much more efficient in terms of the complexity analysis, we conduct an experiment to compare the efficiency of t2vec with those of EDR and EDwP empirically. Figures 3.5(a) and 3.5(b) show that the query time grows with the size of the target database for the k -nn query ($k = 50$) using the Porto and

Table 3.6: Mean rank and training time (hours) for the model equipped with loss functions \mathcal{L}_1 , \mathcal{L}_2 , \mathcal{L}_3 , $\mathcal{L}_3 + \text{CL}$ using the Porto dataset.

Loss	\mathcal{L}_1	\mathcal{L}_2	\mathcal{L}_3	$\mathcal{L}_3 + \text{CL}$
$\text{MR}@d_1 = 0.4$	46.56	21.34	9.70	9.48
$\text{MR}@d_1 = 0.5$	55.72	27.30	13.50	12.70
$\text{MR}@d_1 = 0.6$	68.49	32.01	16.52	15.99
Time	26	120	22	14

Harbin dataset, respectively. Although both EDR and EDwP employ carefully designed pruning and indexing techniques, **t2vec** is at least one order of magnitude faster than both methods. **t2vec** offers near-instantaneous response times that support interactive use and analysis on big trajectory data, such as trajectory clustering, while the competition cannot. A response in less than 200 ms is perceived as instantaneous.

3.4.5 Evaluation on the loss function

In this experiment we evaluate the effectiveness of the proposed loss function and the cell representation learning (CL in Algorithm 1) approach on most similar trajectory search by using the same setting in Section 3.4.3.1. The database size is fixed at $|D'_Q \cup D'_P| = 100,000$. Table 3.6 shows the mean rank (MR) w.r.t dropping rates $d_1 = 0.4, 0.5, 0.6$ and the training time of different loss functions using the Porto dataset. The \mathcal{L}_2 loss, is very expensive to compute, and since the model does not converge even after training for over 5 days (120 hours), we terminate the training process before it converges. \mathcal{L}_3 loss is capable of improving the mean rank significantly when compared to \mathcal{L}_1 . The cell representation learning approach further improves the mean rank and reduces the training time by 1/3.

3.4.6 Effect of the cell size and the hidden layer size

Intuitively, a small cell size provides a higher resolution of the underlying space, but it also generates more cells (tokens), which leads to higher training complexity since the model complexity is linear in the number of tokens [JCMB15]. We evaluate the influence of the cell granularity on the method performance in answering most similar search with $d_1 = 0.5, d_2 = 0.5$. As shown in Table 3.7, a cell size of 100 (resp. 150) gives the best mean rank for the Porto (resp. Harbin) dataset. The smallest cell size (25) performs the worst, which is probably because it has the highest model complexity and thus is much more difficult to train.

Another important parameter that determines the quality of the learned representation \mathbf{v} is the dimension of the hidden layer in the encoder. A high dimension of the

Table 3.7: The impact of the cell size on the model.

Porto				
Cell size	25	50	100	150
MR@d ₁ = 0.5	216.23	15.21	12.70	12.70
MR@d ₁ = 0.6	234.18	19.21	15.99	16.03
MR@d ₂ = 0.5	291.57	9.49	9.49	9.51
MR@d ₂ = 0.6	302.91	10.87	10.80	11.03
Time	37	25	14	8

Harbin				
Cell size	25	50	100	150
MR@d ₁ = 0.5	223.11	24.76	21.74	20.05
MR@d ₁ = 0.6	243.50	36.23	30.95	28.47
MR@d ₂ = 0.5	326.73	21.33	18.51	18.60
MR@d ₂ = 0.6	343.23	26.05	21.08	21.80
Time	48	30	20	12

Table 3.8: The impact of the hidden layer size on the model.

Porto					
v	64	128	256	484	512
MR@d ₁ = 0.5	400.01	50.21	12.70	10.24	11.26
MR@d ₁ = 0.6	431.11	63.71	15.99	16.70	17.42
MR@d ₂ = 0.5	390.27	48.36	9.49	8.01	9.09
MR@d ₂ = 0.6	397.22	50.26	10.80	9.27	10.05

Harbin					
v	64	128	256	484	512
MR@d ₁ = 0.5	70.21	30.24	21.74	21.34	23.57
MR@d ₁ = 0.6	82.33	35.78	30.95	30.98	31.42
MR@d ₂ = 0.5	63.19	27.65	18.51	18.57	19.27
MR@d ₂ = 0.6	65.70	32.18	21.08	23.64	23.06

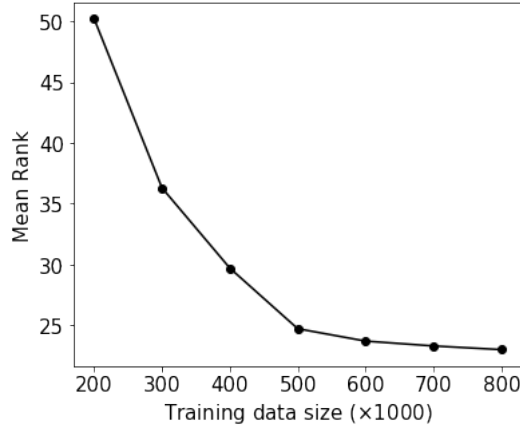


Figure 3.6: The impact of training dataset size on the model using the Porto dataset.

hidden layer is typically much more expressive, but requires more training data to avoid overfitting. Table 3.8 summarizes the impact of the hidden layer size on the most similar trajectory search with the same settings as in the above experiment. It is obvious that increasing $|\mathbf{v}|$ from 64 to 256 significantly enhances the quality of the learned representations, while the performance drops when we increase it further.

3.4.7 Effect of the training data size

In this experiment, we evaluate the effect of training data size on the accuracy of the trajectory similarity search. The setting is similar to the one in Section 3.4.5, with exception that we vary the training data size and fix the dropping rate d_1 at 0.6.

Figure 3.6 shows the effect of the training data size on the most similar search on the Porto dataset. The mean rank drops rapidly as we increase the training data size from 200,000 to 600,000, and the decrease slows down when we continue to enlarge the training data size. When we increase the training data size from 200,000 to 600,000, more transition patterns can be learned for representing the trajectories, and thus the model quality is enhanced significantly. However, when we further increase the training data size, the marginal benefit of using larger training data is less pronounced.

3.5 Summary

We study the problem of learning representation for trajectory similarity computation in this chapter. Trajectory similarity computation is fundamental functionality with many applications such as animal migration pattern studies and vehicle trajectory mining to identify popular routes and similar drivers. While a trajectory is a continuous curve in some spatial domain, e.g., 2D Euclidean space, trajectories are often represented by point

sequences. Existing approaches that compute similarity based on point matching suffer from the problem that they treat two different point sequences differently even when the sequences represent the same trajectory. This is particularly a problem when the point sequences are non-uniform, have low sampling rates, and have noisy points. We propose the first deep learning approach to learning representations of trajectories that is robust to low data quality, thus supporting accurate and efficient trajectory similarity computation and search. Experiments show that our method is capable of higher accuracy and is at least one order of magnitude faster than the state-of-the-art methods for k-nearest trajectory search.

Chapter 4

Travel Time Distribution Learning

In this chapter, we explore the problem of learning travel time distribution of a trip ¹. We develop a novel deep generative model – **DeepST** – to learn the travel time distribution for any route by conditioning on the real-time traffic. **DeepST** interprets the generation of travel time using a three-layer hierarchical probabilistic model and describes the generation process in a reasonable manner, and thus it not only produces more accurate results but also is more data-efficient. This chapter is organized as follows. The overview and summary of contributions are presented in Section 4.1 (the motivation can be found in Section 1.2.2). The problem statement and the overview of **DeepGTT** are presented in Section 4.2. Section 4.3 describes the details of our method. The experimental results are presented in Section 4.4. We summarize this chapter in Section 4.5.

4.1 Overview and Contributions

In this study, we approach travel time distribution learning from deep generative perspective [KW13, RMW14]. Specifically, we propose a model named **DeepGTT** (Deep Generative Travel Time), a three-layer hierarchical probabilistic model [GCSR95]. Each layer in **DeepGTT** captures underlying dependencies among the relevant random variables in a principled way. In the first layer, we present two techniques, spatial smoothness embeddings and amortization [DHNZ95], to share statistical strength among different road segments to address the data sparsity challenge. A representation ρ_i is learned for each road segment r_i . We further propose a convolutional neural network based representation learning component to address the second challenge. This learning component enables the encoding of dynamically changing real-time traffic conditions into a vector \mathbf{c} . In the middle layer, a nonlinear factorization model is developed to combine the road segment

¹This chapter was published as *Learning Travel Time Distributions with Deep Generative Model* [LCSC19].

representation ρ_i and traffic condition \mathbf{c} to generate the travel speed v_i for road segment r_i . This auxiliary random variable v_i allows us to separate the static spatial road features from the dynamically changing traffic conditions, so as to address the third challenge. In the end layer, an attention mechanism [BCB14a] based function is proposed to generate the entire route travel time t by adaptively aggregating the road segment speed v_i . We derive a variational low bound [BKM16] to train the entire model in an end-to-end fashion.

To the best of our knowledge, this is the first deep generative model developed for travel time distribution learning. A great advantage **DeepGTT** inherits from probabilistic methods [GCSR95], is that it is quite data-efficient [Bis06, EHW⁺16]; even only trained with a small fraction of data it could produce substantially better results than existing methods trained on much larger datasets. In summary, our contributions are as follows:

- For the first time, we develop a deep generative model, named **DeepGTT**, to learn travel time distributions. **DeepGTT** is designed to be data-efficient. The model utilizes spatial smoothness embeddings and amortization to model road segments, and a convolutional neural network based representation learning component to capture real-time traffic conditions.
- The carefully designed hierarchical architecture allows us to separate the dynamically changing traffic conditions from the static spatial features, and thus enables incorporating the heterogeneous influencing factors (both spatial and temporal) into a single model for travel time learning.
- We develop an attention mechanism based function to collectively aggregate road segment speeds to generate the observations. Then, a variational low bound is derived to enable the model to be trained in an end-to-end fashion. As **DeepGTT** optimizes the complete distribution rather than a single mean value, it could incorporate more variability for more accurate prediction.
- We conduct thorough experiments on a real world large traffic dataset. Experiments show that our model produces substantially better results than state-of-the-art methods in both travel time estimation and route recovery tasks.

4.2 Problem Statement and DeepGTT Overview

In our study, all GPS trajectories are first mapped into the road network to get their underlying routes with a map matching algorithm [NK09]. The travel time distribution of a route $\mathbf{T.r}$ highly depends on the real-time traffic condition and it is difficult to precisely define traffic condition. Intuitively, average speed of (sub-)trajectories could

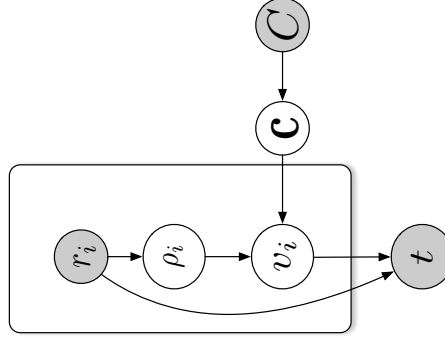


Figure 4.1: The graphical model of the generative process in which $\boldsymbol{\rho}_i \in \mathbb{R}^{|\boldsymbol{\rho}_i|}$, $v_i \in \mathbb{R}$, $\mathbf{c} \in \mathbb{R}^{|\mathbf{c}|}$ are latent variables and r_i, t are observed scalars.

be considered as a measurement or sensing of the real-time traffic condition. Therefore, we propose to use all (sub-)trajectories collected during the time window $[\mathbf{T}.s - \Delta, \mathbf{T}.s)$ as the indicator of real-time traffic condition, denoted by C (or $\mathbf{T}.C$ for the route $\mathbf{T}.\mathbf{r}$). Here, Δ is a specified parameter *e.g.*, 30 minutes. We are now ready to formally define the research problem in this chapter.

Problem Statement. Given a road network $G(V, E)$ and a historical trajectory dataset $\mathcal{D} = \{T^{(m)}\}_{m=1}^M$, we aim to learn the travel time distribution $p(t|\mathbf{r}, C)$, for a given route \mathbf{r} in the road network, conditioned on the real-time traffic condition indicator C .

DeepGTT Overview. The high-level idea of our proposed solution DeepGTT is that we treat the travel time of a route \mathbf{r} as a random variable t and explain the generation of t using a three-layer hierarchical probabilistic model. The graphical model of the generative process is depicted in Figure 4.1.

Intuitively, the travel speed of a road segment r_i depends on two important factors: 1) the static spatial features such as the road types (highway, secondary way), the number of lanes, etc., and 2) the dynamic temporal feature, *i.e.* the real-time traffic condition. In our model, we learn a road representation $\boldsymbol{\rho}_i$ for road segment r_i to capture its static spatial features and use \mathbf{c} to model the real-time traffic condition. The generative process of travel time t for route $\mathbf{r} = [r_i]_{i=1}^n$ is as follows:

- Draw the traffic condition representation $\mathbf{c} \sim p(\mathbf{c}|C)$.
- For the i -th road segment r_i in the route \mathbf{r} ,
 - Draw the road segment representation $\boldsymbol{\rho}_i \sim p(\boldsymbol{\rho}_i|r_i)$.
 - Draw the road segment speed $v_i \sim p(v_i|\boldsymbol{\rho}_i, \mathbf{c})$.
- Draw the travel time $t \sim p(t|\mathbf{r}, \mathbf{v})$.

In this model, $\mathbf{v} = [v_i]_{i=1}^n$ and $\mathbf{c} \in \mathbb{R}^{|\mathbf{c}|}$, $\boldsymbol{\rho}_i \in \mathbb{R}^{|\boldsymbol{\rho}_i|}$ are fixed-length vectors.

Specifically, in the generative process, we first draw the real-time traffic representation $\mathbf{c} \sim p(\mathbf{c}|C)$. For every road segment r_i in route \mathbf{r} we draw its representation $\boldsymbol{\rho}_i \sim p(\boldsymbol{\rho}_i|r_i)$ which only depends on the static spatial features of r_i . The introduction of the auxiliary random variable v_i in the middle layer allows us to model both r_i 's static spatial features and the dynamically changing traffic conditions. In other words, probability distribution $p(v_i|\boldsymbol{\rho}_i, \mathbf{c})$ depends on both road representation $\boldsymbol{\rho}_i$ and traffic condition \mathbf{c} . In the end, we collectively aggregate the road segment speeds v_i to generate the travel time for the entire route \mathbf{r} by drawing t from $p(t|\mathbf{r}, \mathbf{v})$.

4.3 DeepGTT Modeling

We next detail the modeling of road segment representation $\boldsymbol{\rho}_i$ (Section 4.3.1), traffic condition \mathbf{c} (Section 4.3.2), speed (Section 4.3.3), and the aggregation (Section 4.3.4). Section 4.3.5 details the loss function and the learning algorithm. The prediction is discussed in Section 4.3.6.

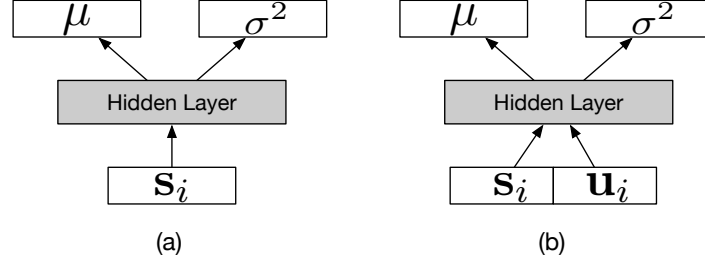
4.3.1 Road Segment Representation $\boldsymbol{\rho}_i$

As aforementioned, travel speed v_i is dominated by two main factors: the static spatial features of road segment r_i and the dynamic temporal feature—real-time traffic condition. We model the impact of static spatial features of r_i with distribution $p(\boldsymbol{\rho}_i|r_i)$, where $\boldsymbol{\rho}_i \in \mathbb{R}^{|\boldsymbol{\rho}_i|}$ is the static road segment representation.

The high-level idea is to encode into $\boldsymbol{\rho}_i$ all spatial features that are indicative to travel speed via modeling $p(\boldsymbol{\rho}_i|r_i)$ conditioned on such spatial features. In our model, we consider the following 5 types of static spatial features:

1. Road types: *e.g.*, primary, secondary, tertiary, residential, etc.;
2. Number of lanes: how many marked traffic lanes;
3. Whether it is a one way or not;
4. The road shape, *e.g.*, straight, curve;
5. Spatial smoothness, *e.g.*, congestion propagation makes neighboring roads likely to have similar congestion level.

To overcome data sparsity issue, we present two techniques—amortization and spatial smoothness embeddings. These two techniques enable the sharing of statistical strength among different road segments and impose spatial smoothness on modeling $p(\boldsymbol{\rho}_i|r_i)$.


 Figure 4.2: The parameterization of $p(\boldsymbol{\rho}_i|r_i)$.

Amortization. The main idea of amortization [DHNZ95] is to use a function $f(\cdot)$ to map a given observation to a set of parameters that are shared across all data-points. This allows us to share statistical strength among different road segments, in the sense that the knowledge learned from frequently traveled road segments is shared to those rarely traveled. In our setting, the observation corresponds to the (1)-(3) types of categorical features.

For easy exposition, let us use $n^{(t)}$, $n^{(l)}$, $n^{(o)}$ to represent the number of possible values in the first 3 types of categorical features, respectively. Then we use $\phi^{(t)}$, $\phi^{(l)}$, $\phi^{(o)}$ to map a road segment to its corresponding feature value. For example, assuming there are 5 road types in total and r_1 belongs to the third road type, then we have $n^{(t)} = 5$ and $\phi^{(t)}(r_1) = 3$.

We introduce three spatial feature embedding matrices, $S^{(t)}$, $S^{(l)}$, and $S^{(o)}$, with size $n^{(t)} \times d^{(t)}$, $n^{(l)} \times d^{(l)}$, and $n^{(o)} \times d^{(o)}$ respectively, for the three types of spatial categorical features. Here, $d^{(t)}$, $d^{(l)}$, and $d^{(o)}$ are their embedding dimensions. Each row in $S^{(\cdot)}$ corresponds to an attribute embedding, *e.g.*, $S^{(t)}[1]$ (the first row of $S^{(t)}$) is the embedding of the first type of road and $S^{(t)}[\phi^{(t)}(r_i)]$ is the embedding of r_i 's road type. For road segment r_i we define \mathbf{s}_i as the concatenation of its spatial feature embeddings, *i.e.*

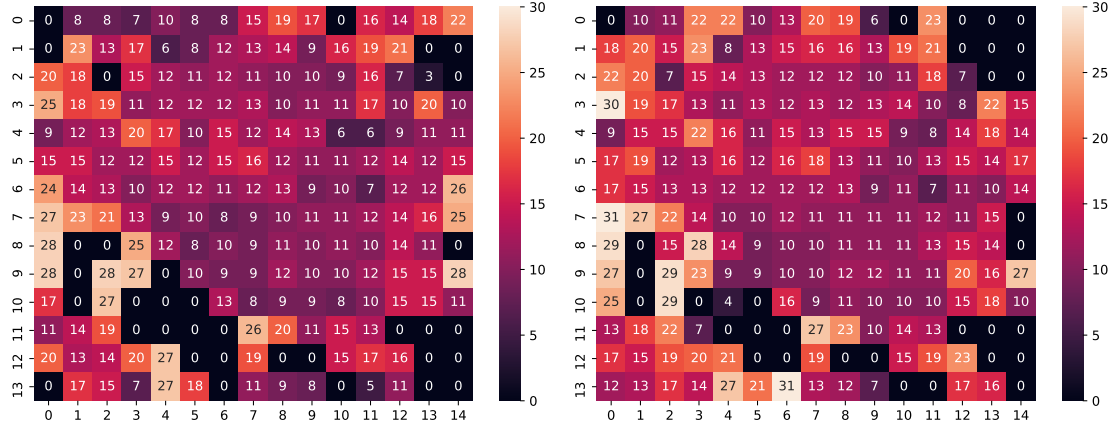
$$\mathbf{s}_i = [S^{(t)}[\phi^{(t)}(r_i)], S^{(l)}[\phi^{(l)}(r_i)], S^{(o)}[\phi^{(o)}(r_i)]] \in \mathbb{R}^{d^{(t)}+d^{(l)}+d^{(o)}}.$$

In the spirit of amortization, we model the distribution $p(\boldsymbol{\rho}_i|r_i)$ with a high dimensional Gaussian distribution whose parameters are the outputs of a determined function taking \mathbf{s}_i as input

$$p(\boldsymbol{\rho}_i|r_i) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{s}_i), \text{diag}(\boldsymbol{\sigma}^2(\mathbf{s}_i))). \quad (4.1)$$

Here, $\boldsymbol{\mu}(\mathbf{s}_i)$ and $\boldsymbol{\sigma}^2(\mathbf{s}_i)$ are parameterized by two MLPs (Multi-Layer Perceptron) with shared input and hidden layers as shown in Figure 4.2a.

Spatial Smoothness Embeddings. Amortization allows us to share the statistical strength of the categorical features among different road segments. However, it does not consider the impact of road shape and spatial smoothness *i.e.* the last 2 types of road features. To remedy this, for every r_i we add an additional embedding $\mathbf{u}_i \in \mathbb{R}^{|\mathbf{u}_i|}$. This


 (a) Real-time traffic condition C at 7am

 (b) Real-time traffic condition C at 11pm

 Figure 4.3: The real-time traffic condition C at 7am and 11pm, where the cell value indicates the average speed in that local area.

embedding aims to capture the complex factors (*e.g.*, road shape) that are specific to the road segment. Moreover, to impose spatial smoothness, we run DeepWalk [PAS14] on the road network $G(V, E)$ to initialize the embeddings \mathbf{u}_i . Then we concatenate \mathbf{s}_i and \mathbf{u}_i as the input of the MLPs to parameterize the probability distribution $p(\boldsymbol{\rho}_i|r_i)$ as following (also shown in Figure 4.2b).

$$p(\boldsymbol{\rho}_i|r_i) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{s}_i, \mathbf{u}_i), \text{diag}(\boldsymbol{\sigma}^2(\mathbf{s}_i, \mathbf{u}_i))). \quad (4.2)$$

The embeddings $S^{(t)}$, $S^{(l)}$, $S^{(o)}$, \mathbf{u}_i and parameters of the two MLPs will be learned from data.

4.3.2 Real-time Traffic Representation \mathbf{c}

In the generative process, we assume there exists a distribution $p(\mathbf{c}|C)$ from which we could draw a random variable \mathbf{c} that captures the real time traffic condition. To realize this, we need to first find a way to construct C which could roughly reveal the real time traffic condition. To this end, we partition the space into cells and estimate the average vehicle speed in a cell by using real time trajectories. Each cell is considered as a sensing of the local traffic congestion level in that area. Figure 4.3(a) and 4.3(b) show the constructed C at 7am and 11pm respectively on the same example day, and the cell size is $2\text{km} \times 2\text{km}$.

To model $p(\mathbf{c}|C)$, a straightforward approach is to reshape C into a vector to get \mathbf{c} and then build the condition probability table with each entry corresponding to a possible \mathbf{c}

value. However, such a table representation has several drawbacks. First, the space cost is prohibitively expensive. Even if we restrict the average speed into an integer with k possible values, there will be $k^{|c|}$ states for \mathbf{c} in total. Second, the table representation is sensitive to the spatial distribution of vehicles at that time. If there is no sensing vehicle passing the cell at that time, the cell value becomes zero, *e.g.*, $C[0, 13]$, $C[0, 14]$ are 18, 22 at 7am and they become zero at 11pm as shown in Figure 4.3. This is unreasonable because 7am is typically a peak hour in a day. Third, the table representation could not reveal similarity between two similar states. Even if only one cell value changes in C , the resulting state \mathbf{c} will be considered to be different from the original one.

To overcome the above shortcomings, we propose to use a CNN (Convolutional Neural Net) to extract the abstract features from the rough estimation speed matrix C . The intuition is that similar traffic conditions generate similar travel times for a given path, which in turn results in close loss. We can then propagate such signals through gradients back to the CNN whose parameters will be tuned to being robust to missing values and being capable of producing similar representations for analogous traffic conditions.

$$\begin{aligned} \mathbf{f} &= \text{CNN}(C) \\ p(\mathbf{c}|C) &= \mathcal{N}(\boldsymbol{\mu}(\mathbf{f}), \text{diag}(\boldsymbol{\sigma}^2(\mathbf{f}))) \end{aligned} \quad (4.3)$$

Equation 5.2 presents the modeling of $p(\mathbf{c}|C)$, where $\boldsymbol{\mu}(\mathbf{f})$, $\boldsymbol{\sigma}^2(\mathbf{f})$ are parameterized by two MLPs with shared hidden layer. The convolutional neural net comprises of three connected convolution blocks and an average pooling layer. Each convolution block consists of three layers: Conv2d \rightarrow BatchNorm2d \rightarrow LeakyReLU.

Note that, the choice of making \mathbf{c} being a random variable instead of being a deterministic one allows us to incorporate more variability in modeling complex traffic conditions and to produce a more reliable model.

4.3.3 Road Segment Speed v_i

Once we get the static road speed representation $\boldsymbol{\rho}_i$ and the dynamic traffic representation \mathbf{c} we could use them to generate road travel speed v_i for r_i . One straightforward method is the linear factorization models [Mur12], *i.e.* travel speed v_i is interpreted as the linear interaction between $\boldsymbol{\rho}_i$ and \mathbf{c} as

$$v_i = \langle \boldsymbol{\rho}_i, \mathbf{c} \rangle, \quad (4.4)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product operation. However, this simple solution is limited by the linear assumption. We consider that travel speed on a road segment is very likely to be the result of nonlinear interaction between the static spatial features and the

dynamic temporal traffic condition. Hence, we propose to model the generation of speed v_i in a nonlinear manner:

$$\begin{aligned} \mathbf{h}_i &= \text{SELU}(W_1 \boldsymbol{\rho}_i + W_2 \mathbf{c}) \\ \mu_i &= \langle \boldsymbol{\theta}_\mu, \mathbf{h}_i \rangle, \quad \sigma_i^2 = \langle \boldsymbol{\theta}_\sigma, \mathbf{h}_i \rangle \\ p(v_i | \boldsymbol{\rho}_i, \mathbf{c}) &= \mathcal{N}(\mu_i, \sigma_i^2). \end{aligned} \quad (4.5)$$

In the above equation, $\text{SELU}(\cdot)$ is the self-normalizing nonlinear activation function [KUMH17] and $W_1, W_2, \boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma$ are parameters to be learned and shared across all road segments. The Gaussian distribution is chosen due to the fact that travel speeds can be perfectly modeled by it [LAS15]. The above generative process generalizes the classical linear factorization models. We actually can recover them by substituting the non-linear activation function SELU with the identity function.

4.3.4 Aggregating Road Segment Speeds

The Gaussian distribution of speed on road segment r_i naturally leads to Inverse-Gaussian distribution for travel time. The probability density function of Inverse-Gaussian distribution is given as follows:

$$p(t; \lambda, \mu) = \sqrt{\frac{\lambda}{2\pi t^3}} \exp \left\{ \frac{-\lambda(t - \mu)^2}{2\mu^2 t} \right\}, \quad (4.6)$$

where $\lambda, \mu > 0$ are parameters *i.e.*, mean μ and variance μ^3/λ . The Inverse-Gaussian distribution describes the first passage time of a Brownian random walk [inv], and thus it is suitable for modeling travel time in our problem setting.

However, we cannot simply sum up the travel time distributions of r_i 's to form the travel time distribution of the entire route \mathbf{r} , because Inverse-Gaussian distribution is not closed under addition. In other words, modeling $t = \sum_i \ell_i / v_i$ leads to an intractable model (here ℓ_i denotes the length of road segment r_i). Similar to [LAS15], in our proposed solution, we consider the average speed of route \mathbf{r} , denoted by $v_{\mathbf{r}}$, as the weighted sum of v_i , *i.e.*

$$v_{\mathbf{r}} = \sum_{i=1}^n w_i v_i, \quad w_i = \frac{\ell_i}{\sum_{j=1}^n \ell_j}.$$

Then we have

$$\mathbb{E}[v_{\mathbf{r}}] = \mathbb{E} \left[\sum_{i=1}^n w_i v_i \right] = \sum_{i=1}^n w_i \mu_i \quad (4.7)$$

$$\text{Var}[v_{\mathbf{r}}] = \text{Var} \left(\sum_{i=1}^n w_i v_i \right) = \sum_{i=1}^n w_i^2 \sigma_i^2. \quad (4.8)$$

Consequently, $P(t|\mathbf{r}, \mathbf{v})$ will be a $\text{IG}(\mu, \lambda)$ with

$$\mu = \frac{\sum_i^n \ell_i}{\mathbb{E}[v_{\mathbf{r}}]}, \quad \lambda = \frac{\mu^3}{(\sum_i^n \ell_i)^2 \text{Var}[v_{\mathbf{r}}]}. \quad (4.9)$$

We scale route variance with the square of route length— $\sum_i^n \ell_i$ in Equation 4.9. This is to take into consideration travel time uncertainty is likely higher for longer route.

Note that Equation 4.8 holds only when v_i 's are mutually independent. This may not be true in reality as speeds of spatially contiguous road segments are highly correlated. On the other hand, this equation offers us the insight that $\text{Var}[v_{\mathbf{r}}]$ can be represented as a weighted sum of $\text{Var}[v_i]$. Next, we present a better way to assign weights, *i.e.*, an attention mechanism-based method to adaptively assign weights for road segments.

The general idea of attention mechanism [BCB14a] is to learn the weights for a collection of inputs under a query vector \mathbf{q} . It assigns a relatively larger weight for the input which is more important under query \mathbf{q} . In our case, the inputs are the hidden states \mathbf{h}_i in Equation 4.5, and the query vector is the traffic representation \mathbf{c} because the uncertainty of a road segment is largely determined by the traffic condition. More formally, we have

$$\begin{aligned} a_i &= \langle W\mathbf{c}, \mathbf{h}_i \rangle \\ \text{Var}[v_{\mathbf{r}}] &= \sum_{i=1}^n \text{softmax}(a_i) \sigma_i^2, \end{aligned} \quad (4.10)$$

where W is the parameter matrix to be learned.

Notably, as we collectively aggregate v_i 's to generate t , we actually amortize the delays caused at the crossroads into their adjacent road segments. Also note that we directly use the parameters of $p(v_i|\boldsymbol{\rho}_i, \mathbf{c})$ to form $p(t|\mathbf{r}, \mathbf{v})$. As a result, v_i becomes the virtual node in the graphical model (see Figure 4.1).

4.3.5 Variational Loss and Learning Algorithm

The generative process implies the following log-likelihood function

$$\begin{aligned} \mathcal{L}(\Theta) = \log \int_{\mathbf{c}} \int_{\boldsymbol{\rho}} d\mathbf{c} d\boldsymbol{\rho} & \prod_{m=1}^M p(\mathbf{c}_m | C_m) \\ & \prod_{i=1}^{n_m} p(\boldsymbol{\rho}_{mi} | r_{mi}) p(v_{mi} | \boldsymbol{\rho}_{mi}, \mathbf{c}_m) p(t_m | \mathbf{r}_m, \mathbf{v}_m), \end{aligned}$$

in which Θ denotes all the parameters involved. Unfortunately, the log-likelihood is intractable due to the appearance of the latent variable integrals. Further, the MCMC based methods [PNI⁺08, ADFDJ03] are not suitable for our case due to their slow convergence speed.

To circumvent this, we make an important observation that travel time t is actually a kind of reflection of the real-time traffic condition indicator C . Therefore, we can view $p(\mathbf{c}|C)$ as the encoder and $p(t|\mathbf{r}, \mathbf{c})$ as the decoder of the real-time traffic condition. In the light of variational auto-encoder [KW13], the log-likelihood for one route could then be approximated as (lower bound)

$$\begin{aligned}\mathcal{L}(\Theta) &\geq \mathbb{E}_{\mathbf{c}, \boldsymbol{\rho}} [\log p(t|\mathbf{r}, \mathbf{c})] \\ &= \mathbb{E}_{\mathbf{c}, \boldsymbol{\rho}} \left[\sum_{i=1}^n \log p(\boldsymbol{\rho}_i|r_i) + \log p(v_i|\boldsymbol{\rho}_i, \mathbf{c}) + \log p(t|\mathbf{r}, \mathbf{v}) \right],\end{aligned}$$

where $\mathbf{c} \sim p(\mathbf{c}|C)$ and $\boldsymbol{\rho}_i \sim p(\boldsymbol{\rho}_i|r_i)$.

This loss can be interpreted as the expected negative reconstruction error. Indeed, if we put a weak prior distribution for \mathbf{c} and $\boldsymbol{\rho}_i$ respectively, we recover the well-known variational loss [BKM16] as

$$\begin{aligned}\mathcal{L}_v(\Theta) &= \mathbb{E}_{\mathbf{c}, \boldsymbol{\rho}} [\log p(t|\mathbf{r}, \mathbf{c})] - KL(p(\mathbf{c}|C)||p(\mathbf{c})) \\ &\quad - \sum_{i=1}^n KL(p(\boldsymbol{\rho}_i|r_i)||p(\boldsymbol{\rho}_i)),\end{aligned}\tag{4.11}$$

where $KL(\cdot||\cdot)$ indicates KL divergence. We now can approximate the variational loss using the Monte Carlo method and propagate the gradients backward using the reparameterization trick [KW13]. In our case, we reparameterize the Gaussian random variable \mathbf{z} (\mathbf{c} and $\boldsymbol{\rho}_i$) as:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2) \Leftrightarrow \mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma}\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1),$$

and calculate the gradient w.r.t the parameters $\boldsymbol{\theta}$ as

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}} [f(\mathbf{z})] = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\epsilon} [f(\boldsymbol{\mu} + \boldsymbol{\sigma}\epsilon)] = \mathbb{E}_{\epsilon} [\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\mu} + \boldsymbol{\sigma}\epsilon)],$$

where $f(\mathbf{z})$ represents the log-likelihood function of the parameters. In this sense, our model becomes the Variational Auto-Encoder (VAE) which is fully differentiable. The learning algorithm is presented in Algorithm 2.

We highlight that it is unnecessary to compute the traffic condition indicator C for every trajectory in training dataset (Line 3 in Algorithm 2). Instead, we discretize the temporal dimension into slots and let the trajectories whose start time fall into the same slot share the same C (see Section 4.4.1).

4.3.6 Prediction

Given the learned parameters Θ , a query route \mathbf{r} along with its real-time traffic condition indicator C , we run the forward computation (Lines 2—5 in Algorithm 2) to predict the travel time distribution $p(t|\mathbf{r}, C)$.

Algorithm 2: Learning Algorithm

Input: Training dataset: $(\mathbf{r}_m, C_m, t_m)_{m=1}^M$
Output: Parameter set: Θ

```

1 while training is true do
2    $\mathcal{B} \leftarrow$  A random minibatch of data;
3    $\mathbf{c}_k \sim p(\mathbf{c}_k | C_k) \quad \forall C_k \in \mathcal{B}, k = 1, \dots, |\mathcal{B}|;$ 
4    $\boldsymbol{\rho}_{ki} \sim p(\boldsymbol{\rho}_{ki} | r_{ki}) \quad \forall \mathbf{r}_k \in \mathcal{B}, \forall r_{ki} \in \mathbf{r}_k, k = 1, \dots, |\mathcal{B}|;$ 
5    $\mathcal{L} \leftarrow$  Calculating the variational loss using Equation 4.11;
6   for  $\boldsymbol{\theta} \in \Theta$  do
7      $\mathbf{g}_{\boldsymbol{\theta}} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}_v(\boldsymbol{\theta});$ 
8      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Gamma(\mathbf{g}_{\boldsymbol{\theta}});$ 
9 return  $\Theta;$ 

```

Specifically, we compute the mean value $\mu(\mathbf{s}_i, \mathbf{u}_i)$ (in Equation 4.2) for each road segment r_i and $\mu(\mathbf{f})$ (in Equation 5.2) for traffic condition indicator C . Different from the training stage where we perform sampling operation to draw instances of $\boldsymbol{\rho}_i$ and \mathbf{c} , we simply feed these mean values into the subsequent computation to generate v_i (Equation 4.5) in prediction. Once we obtain v_i for each r_i , we aggregate them to output the parameters μ, λ (Equation 4.9) of the travel time distribution $p(t|\mathbf{r}, C)$.

4.4 Experiments

We evaluate the effectiveness and scalability of DeepGTT on a real-world taxi dataset, against the state-of-the-art baselines.

4.4.1 Experimental setup

Dataset. The dataset was collected by 13,000 taxis during 28 days in a provincial capital city in China. It contains over 2.9 million trajectories. The sampling time interval between two consecutive points is around 30 seconds, and the map matching accuracy at this sampling rate can be as high as 99% [NK09]. We acquire the road network data from the Open Street Map ². There are 14,497 road segments in total and every road segment has the spatial features: the road type, the number of lanes, one way or not, and road length.

Figure 4.4 shows the spatial distribution of GPS points. As expected, routes at the central area of the city are frequently traveled whereas routes outside of the city are relatively less traveled by the vehicles. The average travel time is 19.4 minutes and

²<https://www.openstreetmap.org>



Figure 4.4: The spatial distribution of the GPS points.

Table 4.1: Dataset statistics.

Measures	min	max	mean
Duration (mins)	7	46	19.4
Distance (km)	1.2	60	11.4

average travel distance is 11.4 kilometers. Table 5.1 reports main statistics of the trips and Figure 4.5 plots distributions of trip duration and travel distance.

We use the first 18 days’ trajectories as training dataset, the middle 3 days’ trajectories as validation set. The remaining 7 days’ trajectories are used for testing. The dataset size of training, validation, and testing is 1.7, 0.3, and 0.9 million respectively.

Baseline Methods. We evaluate DeepGTT on two tasks: (i) travel time estimation, and (ii) route recovery from sparse trajectories.

For travel time estimation, we compare DeepGTT with three baseline methods, namely, DeepTTE [WZC⁺18], WDR [WFY18], and MURAT [LFW⁺18]. For route recovery from sparse trajectories, we demonstrate how we enhance the state-of-art route recovery method STRS [WMS⁺16] by simply changing its travel time distribution component to DeepGTT.

- **DeepTTE** first projects each GPS points in the trajectory into a high dimensional space, in which it performs 1D-Convolution operation; then it compresses the sequence of high-dimensional vectors with a RNN to predict the travel time.
- **WDR** is an ensemble model. In the low level, there are three basic units, namely, a logistic regression model, MLP, and a RNN. A regressor in the top combines the outputs of the three units in order to make the prediction. The RNN unit shares the similar idea with DeepTTE in the sense that it also compresses a sequence of spatial object (road link) representations into a output vector.
- **MURAT** is a multi-task representation learning based travel time estimation model. It only uses the origin-destination, and departing time to make prediction. It

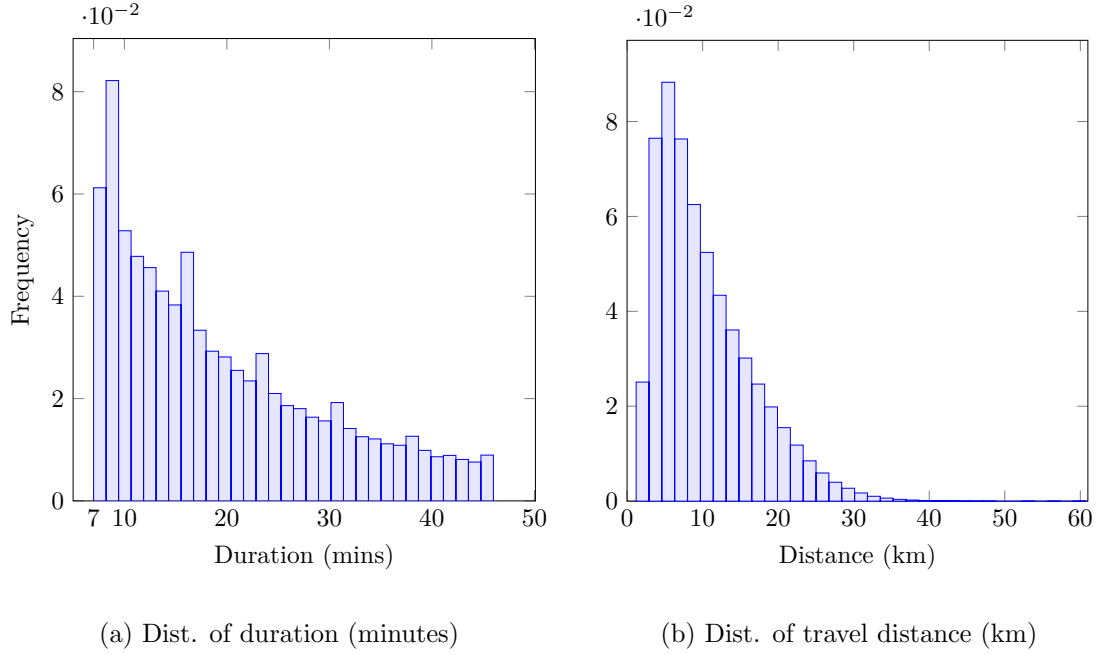


Figure 4.5: The distributions of duration (minutes) and travel distance (km)

partitions the space and time into discrete cells and slots, and learns each cell, slot, and road link on the road networks a high-dimensional representation, respectively. Given an origin-destination pair, it first locates their corresponding cell, slot and neighboring road link, whose representations are used to predict the travel time as well as other path summary with a multi-task loss.

- **STRS** is the state-of-the-art route recovery algorithm, which contains a travel time distribution learning component. The learning component first learns the mean value of travel time by trajectory regression, and then empirically study the relationship between the variance and mean value to derive the distribution.

Parameter settings. Our method³ is implemented with PyTorch 0.4⁴ and Julia 1.0⁵, and trained with a Tesla K40 GPU. The platform runs on Ubuntu 14.04 OS with a Genuine Intel CPU.

The embedding sizes of $d^{(t)}$, $d^{(l)}$, $d^{(o)}$, \mathbf{u}_i are set to 64, 32, 16, 200 respectively. The dimensions of the random variables are set as follows: $|\boldsymbol{\rho}_i| = 256$, $|\mathbf{c}| = 400$, $|\mathbf{f}| = 600$, $|\mathbf{h}_i| = 500$. The space is partitioned into a 138×148 matrix C with cell size $200\text{m} \times 200\text{m}$,

³The code is available at <https://github.com/boathit/deepgtt>

⁴<https://pytorch.org>

⁵<https://julialang.org>

Table 4.2: Performance comparison of different methods.

Method	MURAT	WDR	DeepTTE	DeepGTT
RMSE (sec)	510.23	374.24	337.51	193.04
MAE (sec)	409.24	275.48	241.37	141.75

and $\Delta = 30$ minutes. We discretize the temporal dimension with a slot size 20 (minutes); two trips share the same traffic condition indicator C if their start time fall into the same slot. The batch size $|\mathcal{D}|$ in the training algorithm 2 is 150. The model is optimized by Amsgrad [RKK18, KB14b] with an initial learning rate 0.001 for 10 epochs, and early stopping is used on validation dataset. We select the best hyperparameters for the baseline methods on valuation dataset.

4.4.2 Evaluation on travel time estimation

Overall performance. We evaluate all methods on this task by RMSE (Root Mean Square Error) and MAE (Mean Average Error),

$$\text{RMSE}(\mathbf{t}, \hat{\mathbf{t}}) = \sqrt{\frac{1}{|\mathbf{t}|} \|\mathbf{t} - \hat{\mathbf{t}}\|_2^2}, \quad \text{MAE}(\mathbf{t}, \hat{\mathbf{t}}) = \frac{1}{|\mathbf{t}|} \|\mathbf{t} - \hat{\mathbf{t}}\|_1,$$

where \mathbf{t} , $\hat{\mathbf{t}}$ denote ground truth, estimated value respectively.

Reported in Table 4.2, simple model MURAT leads to relatively large errors, and WDR surpasses it by a fair margin. Among all the baselines DeepTTE achieves the best results. Our model DeepGTT outperforms all the baselines by a large margin, for three reasons: 1) DeepGTT interprets the generation of travel time in a logical way instead of learning by brute force; and 2) DeepGTT makes prediction based on the learned real-time traffic representation, rather than assuming that traffic conditions in the same time slot are temporally-invariant; 3) DeepGTT optimizes the full distribution rather than a single mean value, and thus it could incorporate more variability for more accurate prediction.

Remarkably, DeepGTT reduces MAE to 141.75 seconds for the trips with 19.4 minutes average duration which may potentially facilitate many existing web services.

Impact of travel distance. We now study the impact of travel distance on the performance of different models. To this end, we group the trips in test dataset into subgroups by their lengths (in 5KM step), $[0, 5)$, $[5, 10)$, \dots , $[50, 55)$, and study the performance of different models on these subgroups. Figure 4.6(a) plots MAEs of DeepTTE and DeepGTT against travel distance (we omit other baselines due to their large errors). Not surprisingly, MAEs increase with the travel distance. Longer trips typically involve more road segments and have larger uncertainty. It is worth noting that the performance

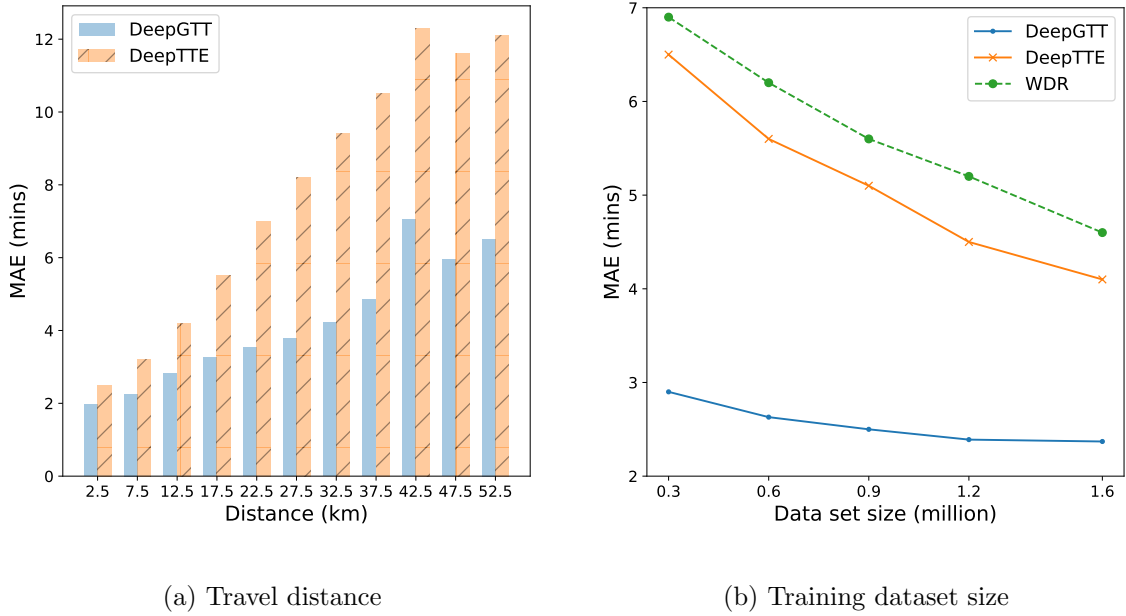


Figure 4.6: (a) MAE changes against travel distance, and over the training dataset size, respectively.

difference between DeepTTE and DeepGTT grows with the travel distance. This result suggests that DeepGTT is more trustful for long trip estimation.

Impact of training data size. One of appealing properties of DeepGTT inherited from probabilistic models is that it is quite data efficient [Bis06, EHW⁺16]. To demonstrate this property, we study the change of MAE with different number of training data points from 0.3 to 1.7 million, reported in Figure 4.6(b). Observe that even trained with only 0.3 million data points, DeepGTT surpasses baseline methods that are trained with the full training data by a notable margin. This can be explained by the fact that DeepGTT interprets the generation of travel time in a more plausible manner, and thus it could reveal the hidden dependencies among relevant variables and explore training data in a more efficient way. The data efficient property enables DeepGTT to be trained in a much faster speed as well, making it attractive to the online web services that require frequent updates.

4.4.3 Performance on route recovery

In this subsection, we demonstrate how DeepGTT could enhance the existing leading route recovery algorithm, in which we need to access $p(t|\mathbf{r})$ for any $t > 0$. As described

in Section 1.2.2, route recovery from sparse trajectories is formulated as

$$\operatorname{argmax}_{\mathbf{r}} p(t|\mathbf{r})\mathbb{P}(\mathbf{r}), \quad \mathbf{r} \in \{\text{candidate routes}\}.$$

The first term $p(t|\mathbf{r})$ describes the probability of a route \mathbf{r} for an observed travel time t , namely, the temporal component. The second term $\mathbb{P}(\mathbf{r})$ describes the probability from the spatial transition perspective, namely, the spatial component. Actually, these are the two main components in the state-of-the-art sparse route recovery algorithm STRS [WMS⁺16].

In this experiment, we replace STRS’s temporal component with DeepGTT and refer to the new model as STRS+. The performance of route recovery is evaluated by route recovery accuracy [WMS⁺16]: the ratio of the length of correctly inferred road segments against the maximum value of the length of ground truth \mathbf{r}_G and the length of inferred route \mathbf{r}_I .

$$\text{accuracy} = \frac{\text{length}(\mathbf{r}_G \cap \mathbf{r}_I)}{\max(\text{length}(\mathbf{r}_G), \text{length}(\mathbf{r}_I))}.$$

We randomly select 10k trajectories from the test dataset and match these trajectories into the map. The matched results are used as ground truth. Then we down-sample these trajectories and recover the routes using STRS, and STRS+, respectively. Figure 4.7 reports the accuracy of STRS and STRS+, over different sampling time intervals (second). As sampling time interval increases, accuracy of both methods drop as expected. The reason is that larger sampling time interval leads to more possible candidate routes to be inferred between two sample points. Observe that accuracy of STRS+ is always higher than that of STRS. In particular, the difference between their accuracies becomes more evident when the sample time interval is larger. This result shows the superiority of STRS+.

4.4.4 The impact of real time traffic

In this study, we relax the assumption that traffic conditions are temporally invariant for the same time slot. Instead, we propose to learn a real-time traffic representation \mathbf{c} , and by conditioning on which we generate the travel time. It is natural to ask the effectiveness of this real time traffic representation \mathbf{c} . We design two experiments to answer this question.

Experiment 1. We fill the traffic condition indicator C with a constant that is specific to the time slot. In this case, our model is forced to “assume” that traffic conditions are temporally invariant. The RMSE, MAE in travel time estimation increases from 193.04 to 272.45, and from 141.75 to 200.02, respectively. In other words, adopting the assumption that traffic conditions are temporally invariant in DeepGTT leads to a 41%

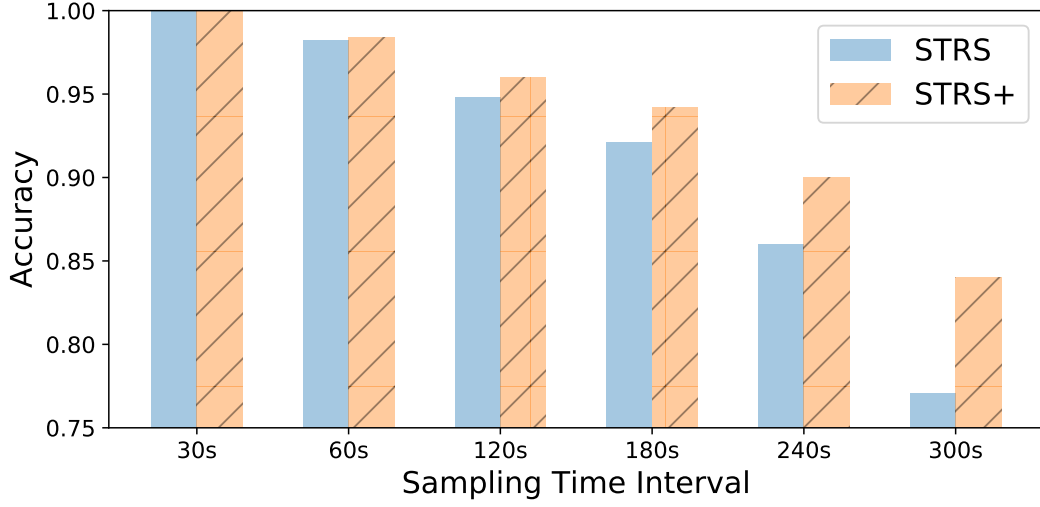


Figure 4.7: Accuracy comparison.

performance drop in terms of MAE. This result implies that an appropriate modeling of $p(\mathbf{c}|C)$ is crucial for accurate travel time estimation.

Experiment 2. As a case study, we randomly select a route \mathbf{r}_{102} with length 21.37km and evaluate the change of its travel time distributions by taking different real-time traffic condition indicators C 's. More specifically, we calculate C by using the (sub-)trajectories in different time slots and feed these C 's into the model while keeping the route unchanged. Figure 4.8a and 4.8b plot the travel time distributions at different time slots on Jan 03, 2015 and Jan 17, 2015 respectively (both are Saturday). Two points are worth noting: I) The travel time distributions at 1:00am and 10:00pm exhibit smaller expectations and variances. This is reasonable because the traffic conditions at these time slots are less congested, compared with peak hours, *e.g.*, 8:00am and 5:00pm. II) Even both 8:00am and 5:00pm are peak hours, the traffic is more congested at 5:00pm on Jan 03, 2015 whereas it is more congested at 8:00am on Jan 17, 2015. Our observations suggest that traffic conditions in the same time slot are not invariant across the temporal dimension. The assumption made in previous studies does not hold [WZC⁺18, WKKL16, LFW⁺18, WFY18, WMS⁺16].

4.4.5 Time Cost Study

The training time complexity of DeepGTT grows linearly with the number of trajectories. Figure 4.9 demonstrates that the training time of different methods vary with the data size. Even though the time cost of both DeepTTE and WDR grows linearly with the size of training data, they have larger constant terms compared with DeepGTT. This is due to their employment of RNN which is more computationally expensive.

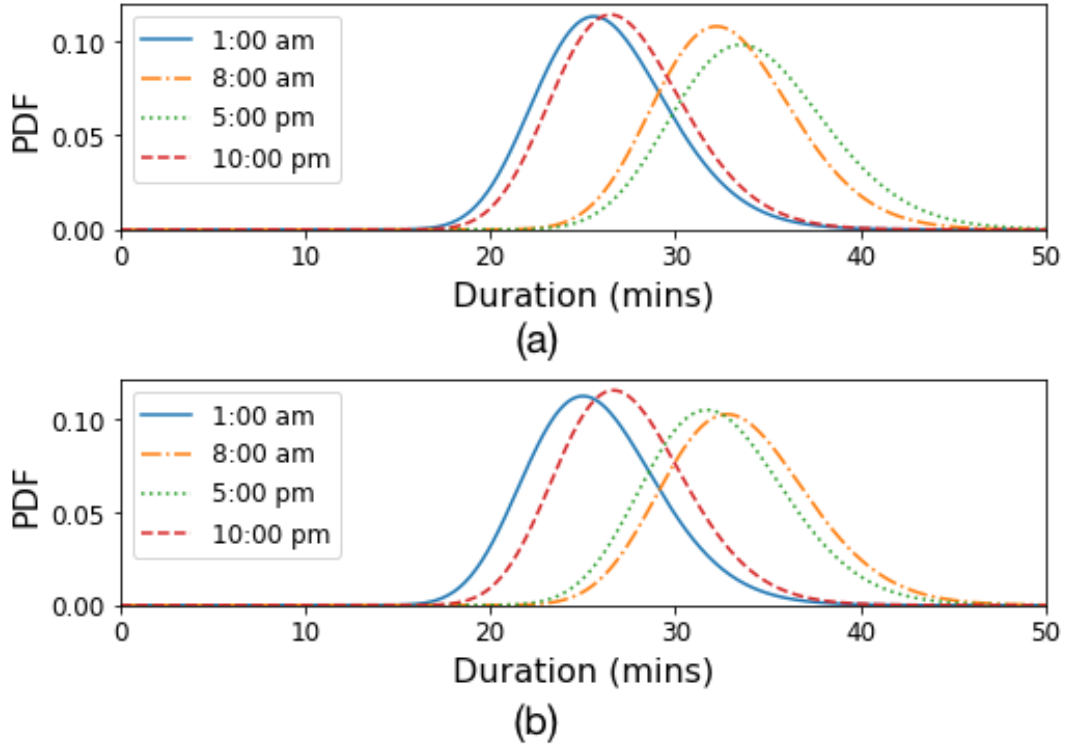


Figure 4.8: The travel time distributions of route r_{102} on Jan 03 (a) and Jan 17 (b), both of them are Saturday.

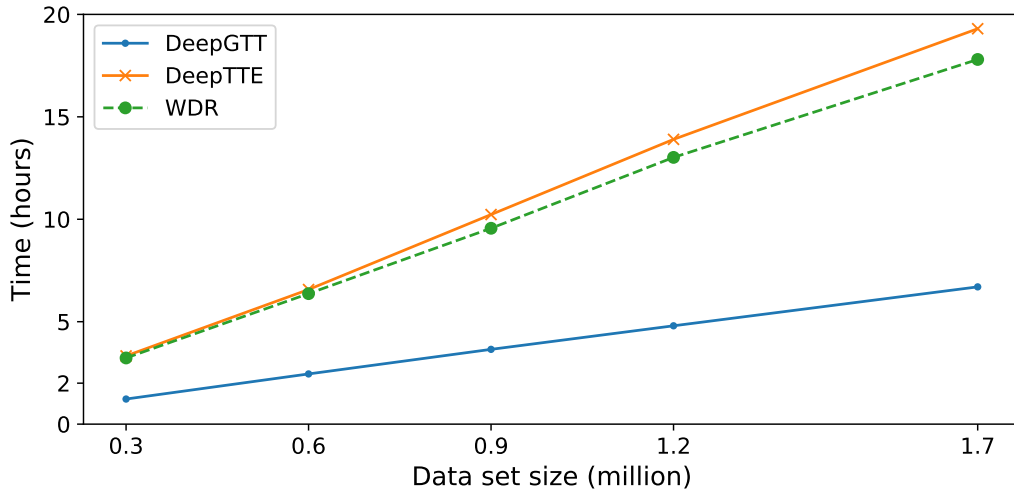


Figure 4.9: Time cost of all methods grows linearly with the training data size. However, DeepGTT has a much smaller constant term than its competitors.

For prediction, DeepGTT can process over 0.9 million trajectories within 81 seconds with a mini-batch size 256. The fast running speed makes it ideal for online deployment.

4.5 Summary

In this chapter, we study the problem of learning the travel time distribution for any route on the road network conditioning on the real time traffic. We interpret the generation of travel time using a three-layers hierarchical probabilistic model, which permits us to separate the statically spatial features from the dynamically changing traffic conditions and describe the generation process in a reasonable manner instead of simply learning by brute force, thus the resulting model is quite data-efficient. A variational loss is further derived and the entire model is fully differentiable, which makes the model easily scale to large data sets. The proposed deep generative model produces substantially better results than the existing methods in both travel time estimation and route recovery from sparse trajectories tasks on a real-world large scale data set.

Chapter 5

Spatial Transition Learning

In this chapter we study the problem of spatial transition learning on the road networks¹. We present a novel deep probabilistic model – **DeepST** – which unifies three key explanatory factors, the past traveled route, the impact of destination and real-time traffic for the route decision. **DeepST** explains the generation of next route by conditioning on the representations of the three explanatory factors. An efficient inference method is developed within the Variational Auto-Encoders framework to scale **DeepST** to large-scale datasets. This chapter is organized as follows. We present the overview and summary of contributions in Section 5.1 (the motivation can be found in Section 1.2.3). Section 5.2 gives the problem statement. The details of **DeepST** are described in Section 5.3. Section 5.4 presents the experimental results conducted on two real-world large-scale trajectory dataset. This chapter is summarized in Section 5.5.

5.1 Overview and Contributions

In this study, we approach the spatial transition modeling problem from the generative perspective. We develop a novel deep probabilistic model–**DeepST** (Deep Probabilistic Spatial Transition), which unifies the aforementioned three key explanatory factors–sequential property, the impact of destination and real-time traffic into a single model, for learning the spatial transition patterns on the road network. **DeepST** explains the generation of a route by conditioning on the past traveled road sequence, destination and real-time traffic representations. The past traveled road sequence is squeezed by a Recurrent Neural Net (RNN) which does not make the explicit dependency assumption. To incorporate the impact of destination, we propose to learn K -destination proxies with an adjoint generative model instead of treating the destinations separately. The benefits of introducing the destination proxies are two-folds: 1) the proposed method will be robust

¹This chapter was published as *Spatial Transition Learning on Road Networks with Deep Probabilistic Models* [LCC20].

to the inaccuracy of destinations; 2) it allows effectively sharing the statistical strength across trips. The destination proxies are learned jointly with the generation of routes which permits end-to-end training. To account for the influence of real-time traffic, we propose to learn the real-time traffic representation with a latent variable, whose posterior distribution can then be inferred from the observations. In the end, we develop an efficient inference method to learn the posterior distributions of the latent variables of DeepST based on observations; the inference method is developed within the Variational Auto-Encoders (VAEs) framework and is fully differentiable, enabling DeepST to scale to large-scale datasets. In summary, our contributions are as follows.

- For the first time, we develop a novel deep probabilistic model–DeepST– to learn the spatial transition patterns, which simultaneously takes into consideration the transition sequential property, the impact of destinations and real-time traffic.
- We propose a novel adjoint generative model to learn the K -destination proxies, which enables effectively sharing statistical strength across trips and the resulting model is robust to inaccurate destinations.
- We develop an efficient inference method that scales the model to large-scale datasets within the VAEs framework.
- We conduct experiments on two real-world datasets to demonstrate the superiority of DeepST over existing methods on two tasks: predicting the most likely routes and route recovery from sparse trajectories.

5.2 Problem Statement

Problem Statement. Given a road network $G(V, E)$ and a historical trajectory dataset $\mathcal{D} = \{T^m\}_{m=1}^M$, as well as the starting time, origin and destination of a trip \mathbf{T} , we aim to predict the most likely traveled route of the trip \mathbf{T} as well as score the likelihood of any route being traveled by conditioning on the real-time traffic.

In this study, we assume that the initial road segment $\mathbf{T}.r_1$ of the trip \mathbf{T} is given; however, for the destination we only assume a rough coordinate \mathbf{d} (denoted by $\mathbf{T}.\mathbf{d}$), *i.e.*, a pair of latitude and longitude, is available.

5.3 The Proposed Method – DeepST

We first present the general idea of our proposed method and the generative process in Section 5.3.1. We then detail the representation learning of past traveled route and destination in Section 5.3.2 and Section 5.3.3, respectively. The developed inference method is presented in Section 5.3.4, and the prediction is discussed in Section 5.3.5. We present the complexity analysis of DeepST in Section 5.3.6

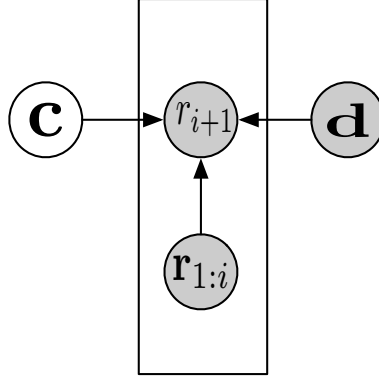


Figure 5.1: The graphical model of the generative process in which $\mathbf{c} \in \mathbb{R}^{|\mathbf{c}|}$ is latent variable and represents the real-time traffic, \mathbf{d} is the trip destination.

5.3.1 The generative process

The high level idea of our proposed method **DeepST** is that we interpret the generation of a route by conditioning on the past traveled road segment sequence, the representations of destination and real-time traffic. In such a manner, **DeepST** simultaneously takes into consideration the aforementioned three key explanatory factors—the sequential property of transition, the global impact of destination and real-time traffic. The graphical model is shown in Figure 5.1 and the generative process of one route is described as follows.

- Draw $\mathbf{c} \sim \text{Normal}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$.
- For $i + 1$ -th road segment, $i \geq 1$
 - Draw $r_{i+1} \sim \mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, \mathbf{d}, \mathbf{c})$.
 - Draw ending indicator $s \sim \text{Bernoulli}(f_s(r_{i+1}, \mathbf{d}))$.
 - If $s = 0$ then continue else end the generation.

We first draw a latent variable $\mathbf{c} \in \mathbb{R}^{|\mathbf{c}|}$ which represents the real-time traffic from an isotropic Gaussian prior distribution. Then the $i + 1$ -th road segment is drawn from $\mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, \mathbf{d}, \mathbf{c})$. Finally, we use a Bernoulli distribution whose parameter is the output of function f_s taking r_{i+1}, \mathbf{d} as input, to decide whether to terminate the generation, and we use the Euclidean distance between the projection point of \mathbf{d} on r_{i+1} and \mathbf{d} to determine the termination of the generation, *i.e.*

$$f_s(r_{i+1}, \mathbf{d}) = \frac{1}{1 + \|\mathbf{p}(\mathbf{d}, r_{i+1}) - \mathbf{d}\|_2},$$

where $\mathbf{p}(\mathbf{d}, r_{i+1})$ denotes the projection point of \mathbf{d} on r_{i+1} . In this study, we propose to model $\mathbb{P}(r_{i+1}|\mathbf{r}_{1:i}, \mathbf{d}, \mathbf{c})$ as a Categorical distribution whose parameter is determined by the past traveled route sequence $\mathbf{r}_{1:i}$, destination \mathbf{d} and real-time traffic \mathbf{c} ,

$$\mathbb{P}(r_{i+1}|\mathbf{r}_{1:i}, \mathbf{d}, \mathbf{c}) = \text{softmax} \left(\alpha^\top f_{\mathbf{r}}(\mathbf{r}_{1:i}) + \beta^\top f_{\mathbf{d}}(\mathbf{d}) + \gamma^\top \mathbf{c} \right),$$

in which $f_{\mathbf{r}}(\mathbf{r}_{1:i}) \in \mathbb{R}^{n_{\mathbf{r}}}$, $f_{\mathbf{d}}(\mathbf{d}) \in \mathbb{R}^{n_{\mathbf{d}}}$ are representations of past traveled route $\mathbf{r}_{1:i}$ and destination \mathbf{d} , respectively; $\alpha \in \mathbb{R}^{n_{\mathbf{r}} \times \mathcal{N}(r_i)}$, $\beta \in \mathbb{R}^{n_{\mathbf{d}} \times \mathcal{N}(r_i)}$, $\gamma \in \mathbb{R}^{|\mathbf{c}| \times \mathcal{N}(r_i)}$ are projection matrices that map the corresponding representations into the road segment space, and $\mathcal{N}(r_i)$ is the number of adjacent road segments of r_i , *e.g.*, $\mathcal{N}(r_2) = 3$ in Figure 1.3a. The projection matrices are shared across all road segments, and since different r_i may have different number of adjacent road segments, we substitute $\mathcal{N}(r_i)$ with $\max_{r \in E} \mathcal{N}(r)$, *i.e.*, the maximum number of neighboring road segments on the road network.

5.3.2 The representation of past traveled route

Most of existing studies [SZW⁺13a, XZZ⁺, BRR14, LAS15] model the transition relationship with Markov Model which requires to make explicit dependency assumption to make inference tractable [GFGS06]. In contrast, the Recurrent Neural Nets are capable of modeling the long range dependency by embedding a sequence of tokens into a vector. As presented in Section 1.2.3, the transition patterns of vehicle on road network may demonstrate strong sequential property, it is desirable to capture such long range dependency when modeling the spatial transition, and thus we adopt the RNNs to squeeze the past traveled route into its representation. Specifically, we update the i -th hidden state as follows

$$\mathbf{h}_i = \begin{cases} \mathbf{0} & i = 1 \\ \text{GRU}(\mathbf{h}_{i-1}, r_{i-1}) & i \geq 2 \end{cases}$$

where \mathbf{h}_1 is initialized as a zero vector and $\text{GRU}(\cdot, \cdot)$ represents the Gated Recurrent Unit updating function [CGCB14b]. Eventually, we choose i -th hidden state \mathbf{h}_i as the representation of past traveled route, *i.e.*, we let $f_{\mathbf{r}}(\mathbf{r}_{1:i-1}) = \mathbf{h}_i$.

5.3.3 The representation of destination

The trip destinations have a global impact on the spatial transition, and thus it is critically important to properly learn the destination representations $f_{\mathbf{d}}(\mathbf{d}) \in \mathbb{R}^{n_{\mathbf{d}}}$ to help the route decision. Previous study [WCS⁺17] assumed that the destination road segments of the trips are available, and learned the representations for these road segments to guide the route decision. However, the exact destination road segment of a trip may not be available at the start of a trip. Furthermore, the method [WCS⁺17] learns representation of each road segment separately cannot effectively share the statistical strength across

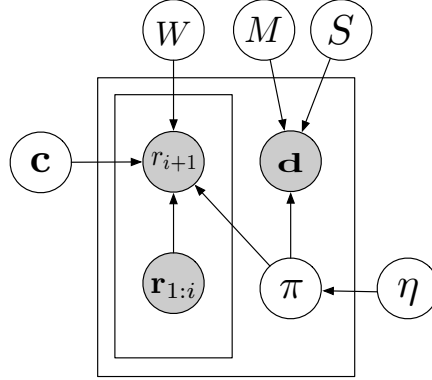


Figure 5.2: The graphical model of the complete generative process in which $\mathbf{c} \in \mathbb{R}^{|\mathcal{C}|}$, $\boldsymbol{\pi} \in \mathbb{R}^k$ are latent variables, $M \in \mathbb{R}^{2 \times k}$, $S \in \mathbb{R}_+^{2 \times k}$, $W \in \mathbb{R}^{n_x \times k}$ are parameters to be learned, and $\boldsymbol{\eta} \in \mathbb{R}^k$ is hyperparameter; $\boldsymbol{\pi}$ can be interpreted as the destination allocation indicator, the nonzero dimension of $\boldsymbol{\pi}$ indicates the proxy which the trip is allocated.

trips with the spatially close destinations. As an example, assuming we model the transition probabilities for the trips shown in Table 1.1; if we treat the destinations separately as does the work [WCS⁺17], we will have

$$\mathbb{P}(r_{10}|r_5, C) = 1/3, \mathbb{P}(r_6|r_5, A) = 1/3, \mathbb{P}(r_6|r_5, B) = 1/3;$$

however the spatial proximity between A and B is small, and if we use one representation AB for all trips driving towards them, we will have

$$\mathbb{P}(r_{10}|r_5, C) = 1/3, \mathbb{P}(r_6|r_5, AB) = 2/3,$$

i.e., the transition patterns can be shared to mutually reinforce the transition probability between r_5 and r_6 across trips.

Intuitively, the trips whose destinations are spatially close should share similar destination representations. More importantly, the learned destination representations are also supposed to be able to effectively guide the spatial transition of their corresponding trips. Separating the destination representations learning and spatial transition learning into two stages prevents end-to-end training, which may lead to a suboptimal solution. Instead, it is desirable to jointly learn the destination representations and model the spatial transition such that the statistical strength could be effectively shared across trips. To summarize, when learning the destination representations we should consider two points: 1) if the spatial proximity between the destinations of two trips is small, the learned destination representations should be similar; 2) the learned representations could effectively guide the spatial transition of their corresponding trips.

Towards this end, we propose to learn representations for the k -destination proxies that are shared by all trips, instead of learning each trip a separate destination representation. Specifically, we simultaneously learn representations of k -spatial locations, referring to as k -destination proxies, and adaptively allocate the trips into one of these destination proxies; the destination proxy representation will be used to guide the spatial transition of trips that are allocated to this proxy. We achieve this by developing an adjoint generative model which explains the generation of the destination coordinates with another latent variable $\boldsymbol{\pi}$. The adjoint generative process is as follows.

- Draw $\boldsymbol{\pi} \sim \text{Categorical}(\boldsymbol{\eta})$;
- Draw $\mathbf{d} \sim \text{Normal}(M\boldsymbol{\pi}, \text{diag}(S\boldsymbol{\pi}))$,

where $\boldsymbol{\eta} \in \mathbb{R}^k$ is a hyperparameter, $\boldsymbol{\pi} \in \mathbb{N}^k$ is a one-hot vector indicating which proxy the trip is allocated. Then we use $\boldsymbol{\pi}$ and M, S to generate the destination coordinate \mathbf{d} , where $M \in \mathbb{R}^{2 \times k}$ is the mean value matrix and $S \in \mathbb{R}_+^{2 \times k}$ is the variance matrix of the k -destination proxies; the operation $M\boldsymbol{\pi}$ (resp. $S\boldsymbol{\pi}$) selects one column of M (resp. S) which corresponds to the mean (resp. variance) of the allocated proxy. We interpret the observation \mathbf{d} via conditioning on $M\boldsymbol{\pi}$ and $S\boldsymbol{\pi}$ with a Gaussian distribution, and hence the resulting method will be robust to the inaccuracy of observations.

In such a fashion, we actually allocate each trip destination \mathbf{d} to a proxy and the trips that are allocated to the same proxy will share the same proxy representation, so as to effectively share the statistical strength across these trips. As our eventual purpose is to obtain the proxy destination representation, we introduce an embedding matrix $W \in \mathbb{R}^{n \times k}$ and let $f_{\mathbf{d}}(\mathbf{d}) = W\boldsymbol{\pi}$. We can see that the introduction of the latent variable $\boldsymbol{\pi}$ enables achieving the aforementioned two points simultaneously. On the one hand, $M\boldsymbol{\pi}$ and $S\boldsymbol{\pi}$ are supposed to be capable of explaining the observations \mathbf{d} , so as to satisfy the spatial constraint. On the other hand, $W\boldsymbol{\pi}$ should be able to yield a useful destination representation that can effectively guide the spatial transition of the trips. Consequently, the complete graphical model becomes the one shown in Figure 5.2. in which $\mathbf{c} \in \mathbb{R}^{|\mathcal{C}|}$, $\boldsymbol{\pi} \in \mathbb{R}^k$ are latent variables and $M \in \mathbb{R}^{2 \times k}$, $S \in \mathbb{R}_+^{2 \times k}$, $W \in \mathbb{R}^{n \times k}$ are parameters to be learned; we can consider $\boldsymbol{\pi}$ as the destination allocation indicator, and the nonzero dimension of $\boldsymbol{\pi}$ indicates the proxy to which the trip is allocated.

5.3.4 Inference with VAEs

The ultimate generative process described in Figure 5.2 implies the following log-likelihood function for one trip,

$$\begin{aligned} \mathcal{L}(\Theta) = \log \sum_{\boldsymbol{\pi}} \int_{\mathbf{c}} p(\mathbf{c} | \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \mathbb{P}(\boldsymbol{\pi} | \boldsymbol{\eta}) \prod_{i=1}^{n-1} \mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, W\boldsymbol{\pi}, \mathbf{c}) d\mathbf{c} \\ + \log \sum_{\boldsymbol{\pi}} \mathbb{P}(\boldsymbol{\pi} | \boldsymbol{\eta}) p(\mathbf{d} | \boldsymbol{\pi}, M, S), \end{aligned} \quad (5.1)$$

where Θ is the collection of all parameters involved. The first term describes the log-likelihood of route generation, while the second term represents the log-likelihood of destination generation. We are required to infer the posterior distributions $p(\mathbf{c}|\mathbf{r}, \mathbf{d})$, $\mathbb{P}(\boldsymbol{\pi}|\mathbf{r}, \mathbf{d})$, and fit the parameters Θ by maximizing the log-likelihood given the observations. The exact computation of the log-likelihood function is intractable as it requires us to integrate (sum) out two latent variables \mathbf{c} , $\boldsymbol{\pi}$ in the high-dimensional space. The sampling-based methods [ADFDJ03] suffer from the slow convergence and are also not suitable in our scenario.

To circumvent this, we develop an approximate inference method within the Variational Auto-Encoders (VAEs) framework [KW13, RMW14]. The general idea behind VAEs is to fit the intractable posterior distributions with appropriate inference neural nets; since the exact computation of log-likelihood is difficult, VAEs resort to maximize the evidence lower bound (ELBO) of the log-likelihood and propagate the gradients backwards to optimize the inference neural nets. To this end, we first derive the ELBO of $\mathcal{L}(\Theta)$ using the Jensen's Inequality as follows,

$$\begin{aligned} \mathcal{L}(\Theta) \geq \mathbb{E}_{q(\mathbf{c}, \boldsymbol{\pi}|\mathbf{r}, \mathbf{d})} \left[\log \frac{p(\mathbf{c})\mathbb{P}(\boldsymbol{\pi}) \prod_{i=1}^{n-1} \mathbb{P}(r_{i+1}|\mathbf{r}_{1:i}, W\boldsymbol{\pi}, \mathbf{c})}{q(\mathbf{c}, \boldsymbol{\pi}|\mathbf{r}, \mathbf{d})} \right] \\ + \mathbb{E}_{q(\boldsymbol{\pi}|\mathbf{r}, \mathbf{d})} \left[\log \frac{\mathbb{P}(\boldsymbol{\pi})p(\mathbf{d}|\boldsymbol{\pi}, M, S)}{q(\boldsymbol{\pi}|\mathbf{r}, \mathbf{d})} \right], \end{aligned}$$

in which $q(\mathbf{c}, \boldsymbol{\pi}|\mathbf{r}, \mathbf{d})$ and $q(\boldsymbol{\pi}|\mathbf{r}, \mathbf{d})$ are the approximated posterior distributions to be optimized. $q(\mathbf{c}, \boldsymbol{\pi}|\mathbf{r}, \mathbf{d})$ represents the joint posterior distribution of \mathbf{c} , $\boldsymbol{\pi}$ given the route \mathbf{r} and destination \mathbf{d} . Since $\boldsymbol{\pi}$ denotes the proxy that \mathbf{d} is allocated to, and this allocation is invariant to the traveled route \mathbf{r} , we factorize the joint distribution as

$$q(\mathbf{c}, \boldsymbol{\pi}|\mathbf{r}, \mathbf{d}) = q(\mathbf{c}|\mathbf{r})q(\boldsymbol{\pi}|\mathbf{d}),$$

and for the same reason, $q(\boldsymbol{\pi}|\mathbf{d}) = q(\boldsymbol{\pi}|\mathbf{r}, \mathbf{d})$. But our scenario slightly differs from the typical VAEs here: because the data to be generated is observable in the typical usage of VAEs when performing inference; however, in our case we seek to predict the most likely route \mathbf{r} which is not available at the time of prediction.

To sidestep this, we note that $q(\mathbf{c}|\mathbf{r})$ actually manages to infer the posterior distribution of traffic condition \mathbf{c} from the transition patterns of \mathbf{r} . Intuitively, the real-time traffic at the start of trip \mathbf{T} can also be measured by the average speed of (sub-)trajectories in the time window $[\mathbf{T}.s - \Delta, \mathbf{T}.s)$, Δ is a specified parameter *e.g.*, 20 minutes. We denote these (sub-)trajectories as C (or $\mathbf{T}.C$), and propose to extract the real-time traffic representation \mathbf{c} from C . To this end, we partition the space into cells and calculate the average vehicle speed in a cell by using the real-time trajectories. However, the raw cell representation is sensitive to the spatial distribution of vehicles, *i.e.*, if there is no sensing

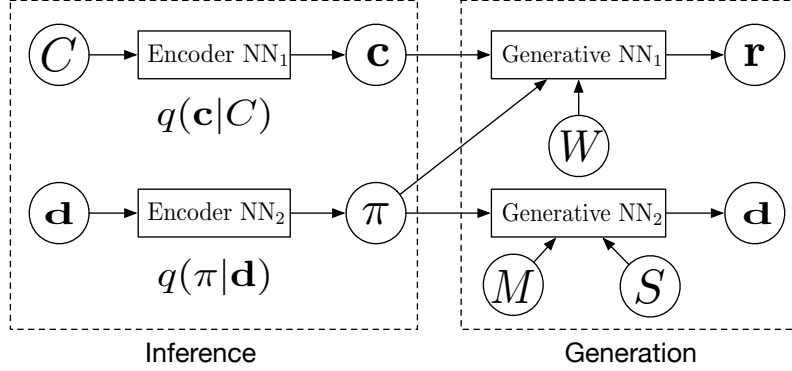


Figure 5.3: The inference framework.

vehicle passing the cell at the moment, the cell value will become zero; furthermore, the raw cell representation cannot reveal the similarity between two analogous traffic conditions, even if only one cell value changes, the representation will be considered to be different from the original one. Therefore, similar to [LCSC19], we use a Convolutional Neural Net (CNN) to extract the high level features from the raw cell representation. More formally, we have

$$\begin{aligned} \mathbf{f} &= \text{CNN}(C) \\ q(\mathbf{c}|C) &= \text{Normal}(\mu(\mathbf{f}), \text{diag}(\sigma^2(\mathbf{f}))) \end{aligned} \quad (5.2)$$

where $\mu(\mathbf{f}), \sigma^2(\mathbf{f})$ are parameterized by two MLPs (Multiple Layer Perceptrons) with shared hidden layer.

Eventually, our inference framework is shown in Figure 5.3. The Encoder NN₁ is the stack of CNN+MLPs, which approximates the posterior distribution $q(\mathbf{c}|C)$; Encoder NN₂ approximates $q(\pi|\mathbf{d})$ and is also parameterized by a MLP. The two encoders are referred to as inference nets, and the parameters set involved are denoted as Φ . For a given trip \mathbf{T} , we feed $\mathbf{T}.C$ and $\mathbf{T}.\mathbf{d}$ into the inference nets to draw the instances of random variable \mathbf{c}, π ; then the sampled instances are fed into the two generative processes represented by Generative NN₁ and Generative NN₂ in Figure 5.3, to compute the subsequent loss. To be specific, with the generated L samples $\{\pi^{(l)}, \mathbf{c}^{(l)}\}_{l=1}^L$ we can estimate the ELBO using the Monte Carlo method as

$$\begin{aligned} \text{ELBO} &\approx \frac{1}{L} \sum_{l=1}^L \sum_{i=1}^{n-1} \log \mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, W \pi^{(l)}, \mathbf{c}^{(l)}) + \frac{1}{L} \sum_{l=1}^L \sum_{i=1}^{n-1} \log p(\mathbf{d} | \pi^{(l)}, M, S) \\ &\quad - \text{KL}(q(\mathbf{c}|C) || \mathbb{P}(\mathbf{c})) - 2\text{KL}(q(\pi|\mathbf{d}) || \mathbb{P}(\pi)), \end{aligned} \quad (5.3)$$

where $\text{KL}(\cdot || \cdot)$ denotes the KL-divergence. Equation 5.3 involves evaluation of log-probability of two distributions, $\mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, \mathbf{d}, \mathbf{c})$ and $p(\mathbf{d} | \pi, M, S)$, note that we ignore the superscript (l) for clarity. The first distribution characterizes the probability of next

possible road r_{i+1} with a softmax distribution, and the log-probability can be calculated as follows,

$$\log \mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, \mathbf{d}, \mathbf{c}) = \log \frac{\exp(\alpha_j^\top \mathbf{h}_i + \beta_j^\top W \boldsymbol{\pi} + \gamma_j^\top \mathbf{c})}{\sum_{j'} \exp(\alpha_{j'}^\top \mathbf{h}_i + \beta_{j'}^\top W \boldsymbol{\pi} + \gamma_{j'}^\top \mathbf{c})},$$

where α_j is the j -th column of α . For the purpose of numerical stability, we use the "log-sum-exp" trick,

$$\log \sum_j \exp(y_j) = \max_j y_j + \log \sum_j \exp(y_j - \max_j y_j),$$

to compute the denominator. The second distribution is the Normal distribution with mean $M\boldsymbol{\pi}$ and variance $S\boldsymbol{\pi}$, whose log-probability calculation is straightforward. When the approximated posterior $q(\mathbf{c}|C)$ and prior $p(\mathbf{c})$ are both Gaussian distributions, the KL-divergence $\text{KL}(q(\mathbf{c}|C)||p(\mathbf{c}))$ has closed form solution. As

$$\begin{aligned} \int q(\mathbf{c}|C) \log p(\mathbf{c}) d\mathbf{c} &= \int \mathcal{N}(\mathbf{c}; \boldsymbol{\mu}, \boldsymbol{\sigma}) \log \mathcal{N}(\mathbf{c}; \mathbf{0}, \mathbf{I}) d\mathbf{c} \\ &= -\frac{|\mathbf{c}|}{2} \log(2\pi) - \frac{1}{2} \sum_{i=1}^{|\mathbf{c}|} (\mu_i^2 + \sigma_i^2), \\ \int q(\mathbf{c}|C) \log q(\mathbf{c}|C) d\mathbf{c} &= \int \mathcal{N}(\mathbf{c}; \boldsymbol{\mu}, \boldsymbol{\sigma}) \log \mathcal{N}(\mathbf{c}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) d\mathbf{c} \\ &= -\frac{|\mathbf{c}|}{2} \log(2\pi) - \frac{1}{2} \sum_{i=1}^{|\mathbf{c}|} (1 + \log \sigma_i^2), \\ \text{KL}(q(\mathbf{c}|C)||p(\mathbf{c})) &= \frac{1}{2} \sum_{i=1}^{|\mathbf{c}|} (1 - \mu_i^2 - \sigma_i^2 + \log \sigma_i^2). \end{aligned}$$

To enable the gradients flowing back into the inference nets, we need to reparameterize the random variables when sampling. For the Gaussian random variable \mathbf{c} , we can reparameterize it as

$$\mathbf{c} \sim \text{Normal}(\mathbf{c} | \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \Leftrightarrow \mathbf{c} = \boldsymbol{\mu} + \boldsymbol{\sigma} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \text{Normal}(\mathbf{0}, \mathbf{I}).$$

As $\boldsymbol{\pi}$ is a discrete random variable which does not admit the above reparameterization trick [KW13, RMW14], we instead resort to the Gumbel-Softmax relaxation [JGP16, MMT16] as

$$\boldsymbol{\pi} = \text{softmax}(\log(\boldsymbol{\phi} + \mathbf{g})),$$

where $\boldsymbol{\phi}$ is the parameter of the posterior distribution $q(\boldsymbol{\pi} | \mathbf{d})$, $\mathbf{g} = -\log(-\log(\mathbf{u}))$, $\mathbf{u} \sim \text{Uniform}(0, 1)$.

Algorithm 3: Learning Algorithm

Input: Training dataset: $(\mathbf{r}_m, C_m, \mathbf{d}_m)_{m=1}^M$
Output: Parameter sets: Θ and Φ

```

1 while training is true do
2    $\mathcal{B} \leftarrow$  A random minibatch of data;
3   for  $b = 1, \dots, |\mathcal{B}|$  do
4      $\mathbf{c}^{(b)} \sim q(\mathbf{c}^{(b)} | C^{(b)})$ ;
5      $\boldsymbol{\pi}^{(b)} \sim q(\boldsymbol{\pi}^{(b)} | \mathbf{d}^{(b)})$ ;
6   ELBO  $\leftarrow$  Calculating the ELBO using Equation 5.3;
7   for  $\theta \in \Theta \cup \Phi$  do
8      $\mathbf{g}_\theta \leftarrow \nabla_\theta \text{ELBO}$ ;
9      $\theta \leftarrow \theta + \Gamma(\mathbf{g}_\theta)$ ;
10 return  $\Theta$  and  $\Phi$ ;

```

The learning algorithm is presented in Algorithm 3. We highlight that it is unnecessary to compute C for each trip in Line 4 of Algorithm 3, as we can discretize the temporal dimension into slots and let the trips whose start times fall into the same slot share one C . Since we take a mini-batch of data to compute the ELBO in Line 6 of Algorithm 3, the number of random variables L can be set to 1 in Equation 5.3 and we are still able to get low-variance gradient estimators in Line 8 of Algorithm 3. We update the parameters of the model with gradient descent in Line 9 of Algorithm 3

5.3.5 Route prediction and likelihood score

Route prediction. In route prediction, given the trained network parameter sets Θ, Φ , the initial road $\mathbf{T}.r_1$, the set (sub-)trajectories $\mathbf{T}.C$ for referring real-time traffic, and destination $\mathbf{T}.\mathbf{d}$ of a trip \mathbf{T} , we run the complete generative process depicted in Figure 5.2 to generate the most probable route for the trip \mathbf{T} . The prediction algorithm is presented in Algorithm 4. Unlike the training stage, the prediction algorithm samples latent variables $\mathbf{c}, \boldsymbol{\pi}$ from the learned posterior distributions (Line 1 – 2); we then use the sampled $\mathbf{c}, \boldsymbol{\pi}$ as well as the learned parameter W to generate the next road link r_{i+1} (Line 4), which is defined as

$$\mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, W\boldsymbol{\pi}, \mathbf{c}) = \text{softmax}(\alpha^\top \mathbf{h}_i + \beta^\top W\boldsymbol{\pi} + \gamma^\top \mathbf{c}).$$

Route likelihood score. Scoring the likelihood of a given route \mathbf{r} is similar to the route prediction except that the route is fixed. Once we draw \mathbf{c} and $\boldsymbol{\pi}$ from the posterior distribution, the likelihood of \mathbf{r} can be calculated as

$$\prod_{i=1}^{|\mathbf{r}|-1} \mathbb{P}(r_{i+1} | \mathbf{r}_{1:i}, W\boldsymbol{\pi}, \mathbf{c}).$$

Algorithm 4: Prediction Algorithm

Input: $(\Theta, \Phi, \mathbf{T}.r_1, \mathbf{T}.C, \mathbf{T}.\mathbf{d})$ **Output:** Generated route: $\mathbf{T}.\mathbf{r}$

```
1  $r_1 \leftarrow \mathbf{T}.r_1, C \leftarrow \mathbf{T}.C, \mathbf{d} \leftarrow \mathbf{T}.\mathbf{d};$ 
2 Draw  $\mathbf{c} \sim q(\mathbf{c}|C);$ 
3 Draw  $\boldsymbol{\pi} \sim q(\boldsymbol{\pi}|\mathbf{d});$ 
4 for  $i \geq 1$  do
5   Draw  $r_{i+1} \sim \mathbb{P}(r_{i+1}|\mathbf{r}_{1:i}, W\boldsymbol{\pi}, \mathbf{c});$ 
6   Draw ending indicator  $s \sim \text{Bernoulli}(f_s(r_{i+1}, \mathbf{d}));$ 
7   if  $s = 0$  then
8     continue;
9   else
10    break;
11 return  $\mathbf{r};$ 
```

5.3.6 The complexity analysis

As our model is trained with SGD method, the convergence time is linearly to the number of trajectories in the training dataset. Given a trained model, the complexity of the route prediction and route likelihood score both are $O(|\mathbf{r}|)$. We empirically study the scalability of DeepST in Section 5.4.4.

5.4 Experiments

We evaluate the effectiveness of DeepST on two real-world large-scale taxi trajectory datasets, against the state-of-art baseline methods in this section. We first present the experimental setup in Section 5.4.1, and then compare DeepST with baseline methods on the most likely route prediction and route recovery, in Section 5.4.2 and Section 5.4.3 respectively. We end this section by evaluating the parameter sensitivity and scalability in Section 5.4.5 and Section 5.4.4, respectively.

5.4.1 Experimental setup

Dataset description. We use two real-world large-scale trajectory datasets in our experiments.

- The first dataset is a **publicly available dataset** released by DiDi Company². It was collected by 33,000 taxi cabs in a provincial capital city, Chengdu, in China.

²<https://outreach.didichuxing.com/research/opendata/en>

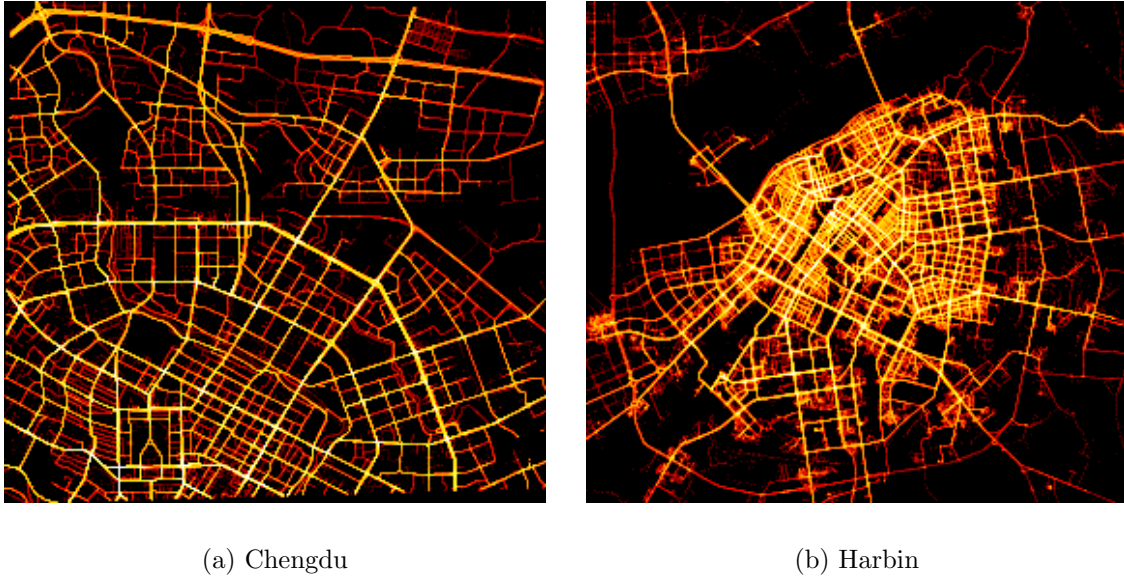


Figure 5.4: The spatial distribution of the GPS points: (a) Chengdu with size 10×10 (km); (b) Harbin with size 28×30 (km).

Table 5.1: Dataset statistics.

City	Chengdu			Harbin		
Measures	min	max	mean	min	max	mean
Distance (km)	1.0	40	4.9	1.1	60	11.4
#road segments	5	85	14	5	102	24

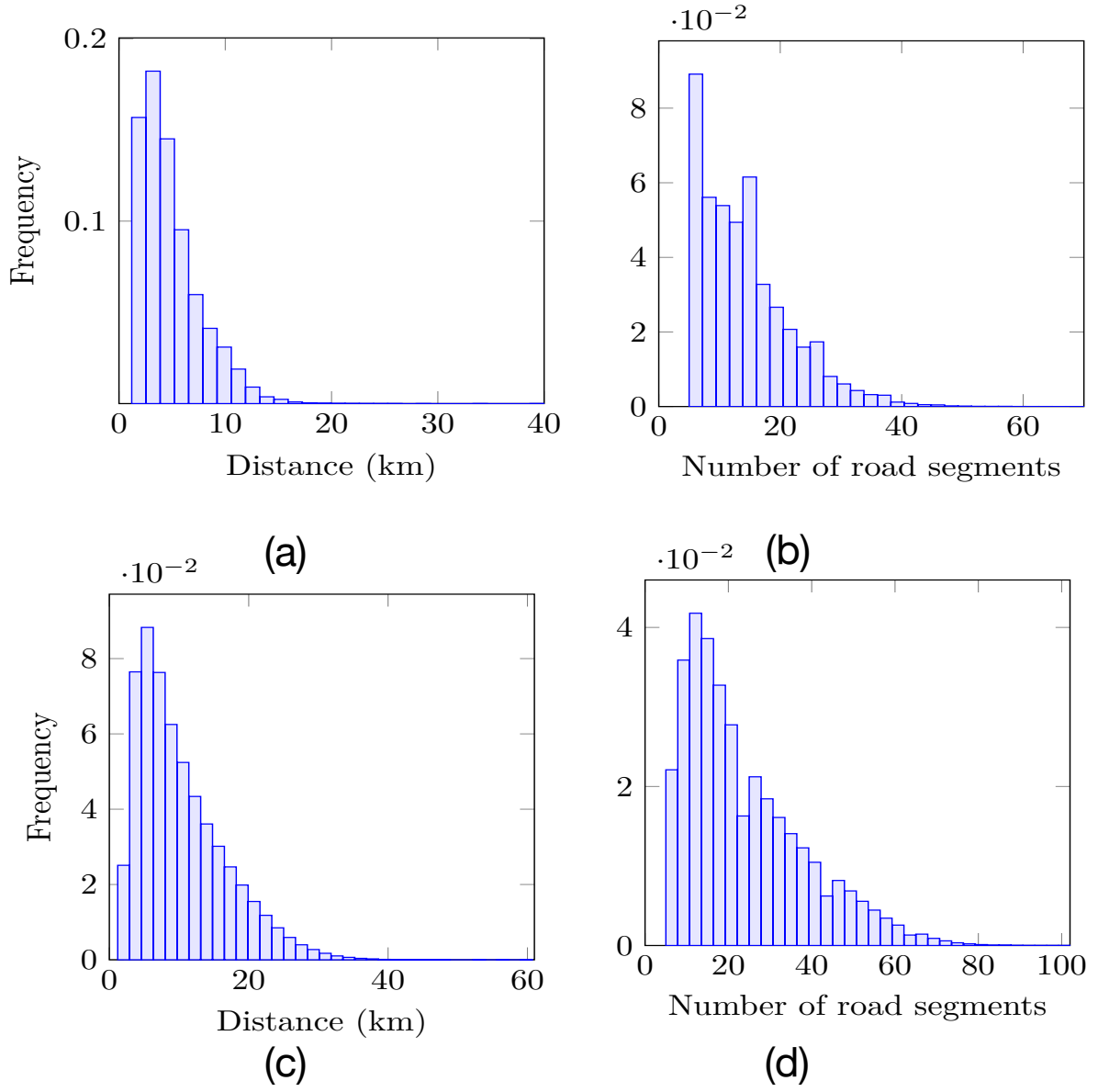


Figure 5.5: The distributions of travel distance (km) and number of road segments: (a)-(b) for Chengdu; (c)-(d) for Harbin.

The sampling rate is around 3 seconds. We include its first 15 days’ data, over 3 million trajectories, in our experiments.

- The second dataset contains over 2.9 million trajectories, which were collected by 13,000 taxi during one month in another provincial capital city, Harbin, in China. The sampling time interval is around 30 seconds per GPS point, and the accuracy of existing map-matching algorithm can be 99% at this sampling-rate [NK09].

Figure 5.4 shows the spatial distributions of the GPS points of the two datasets. The road network data is collected from the OpenStreetMap³, and the Chengdu and Harbin datasets cover 3,185 and 12,497 road segments respectively. Figure 5.5 plots the distributions of travel distance and the number of road segments covered by the trips. Table 5.1 reports the basic statistics of the trips. The mean travel distance and number of road segments are 4.8 (km) and 14 respectively for the Chengdu dataset, and 11.4 (km) and 24 respectively for the Harbin dataset.

For the Chengdu dataset, we use its first 8 days’ trajectories as training dataset, and the next 2 days’ trajectories as validation, and the remaining ones are used for testing. For the Harbin dataset, we use the first 18 days’ trajectories as training dataset, and the next 3 days’ trajectories as validation, and the remaining ones are used for testing. As a result, the dataset size of training, validation, and testing is 1.6 (resp. 1.7), 0.4 (resp. 0.3), and 1.0 (resp. 0.9) million for Chengdu (resp. Harbin) dataset respectively.

Baseline Methods. DeepST is evaluated against baseline methods on two tasks: i) the most likely route prediction, and ii) route recovery from sparse trajectories. For the task of the most likely route prediction, we compare DeepST with DeepST-C, RNN, MMI (the first-order Markov Model), WSP (Weighted Shortest Path), and the state-of-art route decision model CSSRNN [WCS⁺17]. For the task of route recovery from sparse trajectories, DeepST is compared with STRS [WMS⁺16], which has been shown to be superior to other route recovery methods including HRIS [ZZXZ12], InferTra [BRR14] and MPR [CSZ11a].

- DeepST-C is a simplified version of DeepST without considering the impact of real-time traffic.
- RNN is the vanilla RNN that only takes the initial road segment as input. It is a further simplified DeepST by ignoring the impact of both the destination and real-time traffic.

³<https://www.openstreetmap.org>

- **CSSRNN** [WCS⁺17] is the state-of-art route decision model. It considers the road links as tokens and represents them with high-dimensional representations. It squeezes the sequence of representations of past traveled route with a RNN to predict the most likely transition in the next step. In addition, it assumes the last road segments of the trips are known in advance and learns their representations to help model the spatial transition.
- **MMI** models the spatial transition by calculating the first-order conditional probability between adjacent road segments from the historical trips.
- **WSP** always returns the shortest path from the origin road segment to the destination road segment on the weighted road network. The edge weight equals to the mean travel time of corresponding road segment, and the mean travel time is estimated using the entire historical dataset.
- **STRS** [WMS⁺16] is the state-of-art route recovery method comprising of a travel time inference module and a spatial transition inference module. Its spatial transition inference module considers the taxi drivers as experts and learns the transition patterns on the road networks with the inverse reinforcement learning. We substitute its spatial transition inference module with **DeepST** to demonstrate how **DeepST** can be used to enhance its performance.

Platform and Parameter Setting. The platform runs Ubuntu 16.04 OS, and the model is implemented with PyTorch 1.0⁴, and trained with one Tesla P100 GPU around 6 hours (resp. 10 hours) for the Chengdu (resp. Harbin) dataset.

For the Harbin dataset, the number of destination proxies k is 1000; we partition the space into a 138×148 matrix with cell size $200\text{m} \times 200\text{m}$. For the Chengdu dataset, k is set to 500; the space is partitioned into a 87×98 matrix with cell size $100\text{m} \times 100\text{m}$. The CNN in Equation 5.2 comprises of three connected convolution blocks followed by an average pooling layer; each convolution block consists of three layers: Conv2d \rightarrow BatchNorm2d \rightarrow LeakyReLU. The CNN architectures of the two cities are presented in Table 5.2, the two cities share the same architecture except the pooling layer, we use AvgPool2d(7) for Harbin and AvgPool2d(5) for Chengdu. The other parameters are the same for the two datasets, which are described as follows.

The dimension of real-time traffic representation $|\mathbf{c}|$ and the hidden size of all MLPs used are 256, $n_{\mathbf{r}}$, $n_{\mathbf{d}}$ are set as 128. We choose a three-layer stacking GRU with hidden size 256 for all RNNs used in the experiments. The time window size $\Delta = 30$ (minutes), and the temporal dimension is discretized into 20 (minutes) size slots, two trips share the same C if their start times fall into the same slot. The batch size $|\mathcal{B}|$ is 128. The model is trained with Adam [KB14b] for 15 epochs, and early stopping is used on the validation dataset.

⁴<https://pytorch.org>

Table 5.2: CNN architecture

Layer	Type
1	Conv2d(1, 32, (5, 5), stride=2, padding=1)
2	BatchNorm2d(32)
3	LeakyReLU(0.1, inplace=True)
4	Conv2d(32, 64, (4, 4), stride=2, padding=1)
5	BatchNorm2d(64)
6	LeakyReLU(0.1, inplace=True)
7	Conv2d(64, 64, (4, 4), stride=2, padding=1)
8	BatchNorm2d(64)
9	LeakyReLU(0.1, inplace=True)
10	AvgPool2d(7) / AvgPool2d(5)

5.4.2 The most likely route prediction

We evaluate different methods on the most likely traveled route prediction task in terms of two measurements: recall@ n and accuracy. 1) Denoting the ground truth route as \mathbf{r} , we first generate the most likely route $\hat{\mathbf{r}}$ using different methods. Since the generated route $\hat{\mathbf{r}}$ can be arbitrarily long, to give a fair comparison, we truncate $\hat{\mathbf{r}}$ by only preserving the first $|\mathbf{r}|$ road segments, denoting as $\hat{\mathbf{r}}_t$, and recall@ n is defined as the ratio of the length of correctly predicted road segments over the length of ground truth \mathbf{r} , *i.e.*

$$\text{recall@}n = \frac{|\mathbf{r} \cap \hat{\mathbf{r}}_t|}{|\mathbf{r}|}, \quad (5.4)$$

2) accuracy [WMS⁺16] is defined as the ratio of the length of correctly predicted road segments over the maximum value of the length of ground truth \mathbf{r} and the length of predicted route $\hat{\mathbf{r}}$, *i.e.*

$$\text{accuracy} = \frac{|\mathbf{r} \cap \hat{\mathbf{r}}|}{\max(|\mathbf{r}|, |\hat{\mathbf{r}}|)}. \quad (5.5)$$

Overall Performance. Table 5.3 shows the overall performance of different methods on the two datasets. RNN outperforms MMI, as it can capture the long-range dependency in the spatial transition patterns. Both RNN and MMI are worse than WSP in terms of recall@ n and accuracy; because without considering the destinations, the former two methods will always make identical predictions for all trips that start from the same initial road segment. CSSRNN performs much better than RNN, MMI and WSP as it explicitly incorporates the influence of destinations by learning their distributed representations. This indicates that the destinations play a very important role in the route decision. DeepST surpasses CSSRNN by 10.4% (resp. 18.2%) in terms of recall@ n and by 10.1% (resp. 19.5%) in terms of accuracy on the Chengdu (resp. Harbin) dataset, as

Table 5.3: Overall performance.

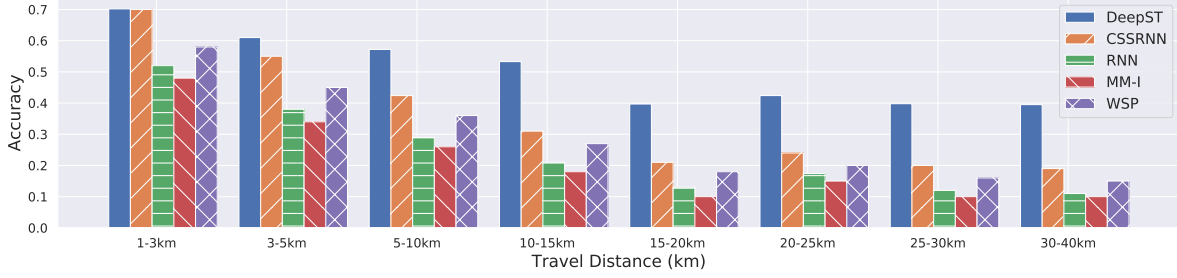
Chengdu						
Method	DeepST	DeepST-C	CSSRNN	RNN	MMI	WSP
recall@ n	0.637	0.626	0.577	0.409	0.318	0.431
accuracy	0.612	0.601	0.556	0.389	0.281	0.431

Harbin						
Method	DeepST	DeepST-C	CSSRNN	RNN	MMI	WSP
recall@ n	0.397	0.385	0.336	0.261	0.202	0.267
accuracy	0.374	0.366	0.313	0.172	0.154	0.267

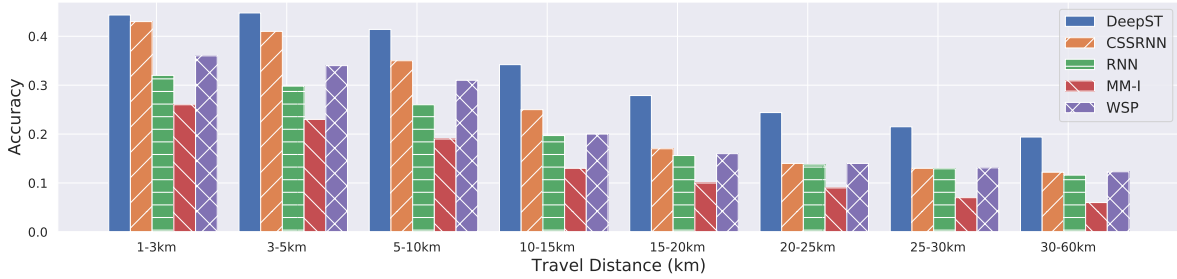
it simultaneously models the spatial transition sequential property, the impact of destinations and real-time traffic in a principled way. All methods show better performance on the Chengdu dataset than on the Harbin dataset. This could be because the mean length of trips in the Harbin dataset is much longer and the road network topological structure of Harbin is more complex as shown in Figure 5.4, and thus the task on the Harbin dataset is more challenging.

Effectiveness of k -destination proxies. As shown in Table 5.3 even without considering the real-time traffic, DeepST-C is able to surpass CSSRNN, which learns destination representations separately, by a fair margin. This verifies our conjecture that learning representations for the destinations separately cannot effectively share the statistical strength across trips and also demonstrates the effectiveness of our proposed k -destination proxies.

Impact of travel distance. We further study the impact of travel distance on the performance of different methods. To this end, we partition the trips in the test dataset by their length (km) into 8 buckets, $[1, 3)$, $[3, 5)$, $[5, 10)$, $[10, 15)$, $[15, 20)$, $[20, 25)$, $[25, 30)$, $[30, -)$ and calculate accuracy of different methods on the buckets. Figure 5.6 shows the accuracy of different methods over the travel distance. For the short trips (<3 km), both DeepST and CSSRNN show much better performance than the other three methods. As the travel distance grows, not surprisingly, the performance of all methods drops. This is because as the travel distance increases, the number of possible routes between the origin and destination grows exponentially, and the task of predicting the most likely route from them becomes more difficult. Nevertheless, DeepST is able to surpass the baseline methods on all buckets. As the travel distance grows up to 10km, the performance gap between DeepST and baseline methods becomes more evident; in particular, DeepST surpasses the best baseline method by almost 50% in terms of accuracy on the Chengdu dataset.



(a) Chengdu



(b) Harbin

Figure 5.6: The route prediction accuracy of different methods versus travel distance.

Table 5.4: Route recovery accuracy versus sampling rate.

Chengdu									
Rate (mins)	1	2	3	4	5	6	7	8	9
STRS	0.98	0.96	0.93	0.90	0.86	0.82	0.79	0.73	0.69
STRS+	0.99	0.98	0.97	0.95	0.93	0.91	0.88	0.83	0.81
δ (%)	1.02	2.08	4.30	5.56	8.14	11.0	11.4	13.7	15.9

Harbin									
Rate (mins)	1	2	3	4	5	6	7	8	9
STRS	0.98	0.95	0.93	0.89	0.85	0.82	0.76	0.70	0.65
STRS+	0.98	0.96	0.95	0.92	0.89	0.86	0.82	0.77	0.75
δ (%)	0.00	1.05	2.15	3.37	4.71	4.88	7.89	10.0	15.4

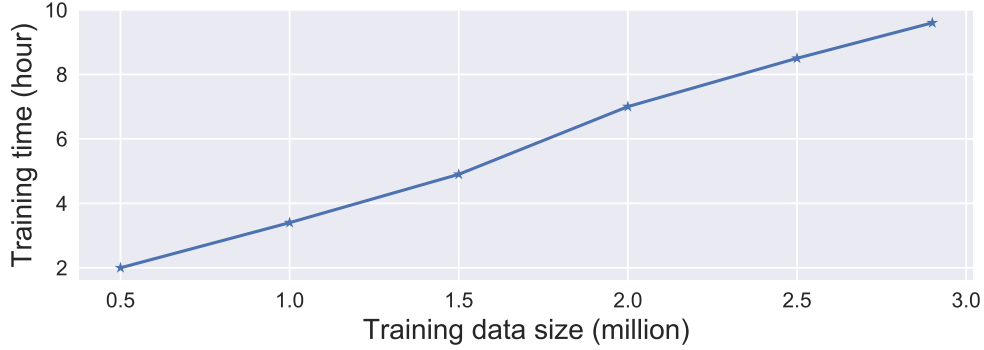


Figure 5.7: Training time versus training data size on Harbin dataset.

5.4.3 Evaluation on route recovery

DeepST is able to score the spatial transition likelihood of any given route, and thus can be used to boost existing route recovery algorithm. As mentioned in Example 4 in Section 1.2.2, the route recovery algorithm attempts to infer the most likely route between two GPS points in a sparse trajectory. We use the state-of-art sparse route recovery algorithm STRS [2] to illustrate how DeepST can be used to enhance the existing route recovery algorithms. As the travel time t during the two points is often available, the problem is expressed as $\operatorname{argmax}_{\mathbf{r}} \mathbb{P}(\mathbf{r}|t)$ [WMS⁺16]. Using the Bayes rule we have,

$$\operatorname{argmax}_{\mathbf{r}} p(t|\mathbf{r})\mathbb{P}(\mathbf{r}), \quad \mathbf{r} \in \{\text{candidate routes}\}.$$

The first term $p(t|\mathbf{r})$ measures the likelihood of a route \mathbf{r} with an observed travel time t , namely, the temporal inference module; the second term $\mathbb{P}(\mathbf{r})$ scores the spatial transition likelihood of a route \mathbf{r} , namely, the spatial inference module. We substitute the spatial inference module $\mathbb{P}(\mathbf{r})$ of STRS with DeepST and the new model is referred to as STRS+. We compare the route recovery accuracy of STRS and STRS+ to see whether DeepST is able to enhance STRS.

To this end, we randomly select 10k trajectories from the test dataset and map these trajectories into the road network as the ground truth with the existing map-matching algorithm [NK09]. Then we downsample the trajectories with different sampling rates and infer the underlying route with STRS and STRS+. Table 5.4 presents the route recovery accuracy (as defined in Equation 5.5) of STRS and STRS+ over the sampling rate. The accuracy increase of STRS+ over STRS is denoted by δ . The superiority of STRS+ becomes obvious as sampling-rate increases and the trajectory becomes more sparse. Note that the need for route recovery is more critical when the trajectory is sparse. As the δ row shows, as the sampling rate grows up to 9 minutes STRS+ outperforms STRS by 15% in terms of accuracy.

Table 5.5: The sensitivity to k on Chengdu dataset.

k	500	1000	1500	2000	2500	3000
recall@ n	0.601	0.637	0.630	0.622	0.611	0.598
accuracy	0.575	0.612	0.610	0.600	0.589	0.570

Table 5.6: The sensitivity to k on Harbin dataset.

k	500	1000	1500	2000	2500	3000
recall@ n	0.362	0.397	0.398	0.386	0.383	0.375
accuracy	0.339	0.374	0.373	0.368	0.362	0.353

5.4.4 Scalability

Figure 5.7 shows the scalability of DeepST on Harbin dataset. It can be seen easily that the training time grows linearly against the training dataset size (the same observation has been made on Chengdu dataset).

5.4.5 Parameter sensitivity study

In this study we propose to learn k -destination proxies to effectively share statistical strength across trips, rather than treating each destination separately. As reported in Section 5.4.2, the k -destination proxies have an important impact on the performance. Hence, we further analyze the impact of k on the performance in this experiment.

Table 5.5 and Table 5.6 report the recall@ n and accuracy on the two datasets respectively as we vary k . The performance is significantly improved when k increases from 500 to 1000. This is because a small k could not provide sufficient number of proxies to guide the spatial transition of vehicles. Both recall@ n and accuracy drop when increasing k to 2000 and beyond. This is because with a large k , no sufficient number of trips is allocated to a proxy, and thus different proxies cannot effectively share the desired statistical strength.

5.5 Summary

In this chapter, we study the problem of predicting the most likely traveling route on the road network between two given locations by considering the real-time traffic. We present a deep probabilistic model – DeepST – which unifies three key explanatory factors, the past traveled route, the impact of destination and real-time traffic for the route decision. DeepST explains the generation of next route by conditioning on the representations

of the three explanatory factors. To enable effectively sharing the statistical strength, we propose to learn representations of k -destination proxies with an adjoint generative model. To incorporate the impact of real-time traffic, we introduce a high dimensional latent variable as its representation whose posterior distribution can then be inferred from observations. An efficient inference method is developed within the Variational Auto-Encoders framework to scale **DeepST** to large-scale datasets. We conduct experiments on two real-world large-scale trajectory datasets to demonstrate the superiority of **DeepST** over the existing methods on two tasks: the most likely route prediction and route recovery from sparse trajectories. In particular, on one public large-scale trajectory dataset, **DeepST** surpasses the best competing method by almost 50% on the most likely route prediction task and up to 15% on the route recovery task in terms of accuracy.

Chapter 6

Conclusions and Future Work

In this dissertation, we propose to study three trajectory analytics problems with differentiable generative models: 1) learning representation for trajectory similarity computation; 2) learning the travel time distribution for a trip; 3) spatial transition learning on the road networks. In the first study, we propose a seq2seq-based generative model whereas we developed two novel deep probabilistic generative models in the last two studies. Hence, we make the conclusions and discuss the future work in this chapter.

6.1 Conclusions

In Chapter 3, we study the problem of *representation learning for trajectory similarity computation*, which is defined as follows. Given a collection of historical trajectories, we aim to learn a representation $\mathbf{v} \in \mathbb{R}^n$ (n is the dimension of a Euclidean space) for each trajectory T such that the representation can reflect the underlying route of the trajectory for computing trajectory similarity. The similarity of two trajectories based on the learned representations must be robust to non-uniform, low sampling rates and noisy sample points.

Measuring the similarity between two trajectories is a fundamental research problem in trajectory data mining, its importance has spurred the study of many discrepancy measurements including, dynamic time wrapping (DTW) [YJF98], longest common subsequence (LCSS) [VKG02], edit distance with real penalty (ERP) [CN04], and edit distance on real sequences (EDR) [CÖO05]. These methods try to form a pairwise the sample points of two trajectories and then identify the best alignment by the dynamic programming. Consequently, these pairwise-matching methods not only tend to yield worse performance in the presence of non-uniform, low-sampling rate and noisy sample points, but also lead to a quadratic computation complexity, which makes them struggle in processing big trajectory datasets.

Motivated by these observations, to the best of our knowledge, we present the first solution for using learning for creating representations of trajectories that are well suited for use in trajectory similarity computation. Specifically, we propose a seq2seq-based differentiable generative model to learn representations for trajectories that enable accurate and efficient trajectory similarity search that is robust to sampling rate variations and noisy sample points. The method is evaluated empirically with favorable results in terms of both accuracy and efficiency. It consistently outperforms the baseline methods by a large margin in the most similarity tasks. The method is at least one order of magnitude faster than the other methods, thus it can support big trajectory analysis and interactive use while the competition cannot.

In Chapter 4, we explore the problem of *learning travel time distribution of a trip*, which is formally defined as follows. Given a road network $G(V, E)$ and a historical trajectory dataset, we aim to learn the travel time distribution $p(t|\mathbf{r}, C)$, for a given route \mathbf{r} in the road network, conditioned on the real-time traffic condition indicator C .

Estimating the travel time of a given path on the road networks is an important task, which finds its applications in route planning, taxi dispatching, and ride-sharing. Due to its importance, significant research efforts have been made towards the accurate estimation of travel times. However, existing studies only focus on the estimation of *expected travel time*, and assume that traffic conditions in the same time slot (*e.g.*, 8:00am-9:00am on every Saturday) are temporally-invariant, when estimating travel times. As we argued in Section 1.2.2, it is more useful or even essential to learn the travel time distributions with the consideration of real-time traffic.

To this end, for the first time, we develop a deep generative model **DeepGTT** for travel time distribution learning. We tackle the data sparsity challenge by presenting two techniques, amortization and spatial smoothness embedding, in the lower layer. We relax the assumption that traffic conditions are temporally-invariant by proposing a convolution neural net based real time traffic representation learning and a hierarchical structure. The introduction of an auxiliary random variable v_i in the middle layer enables separating the static spatial features from the dynamically changing real-time traffic. As a result, we could incorporate these heterogeneous influencing factors into a single model. The further derived variational loss enables the model to be trained in an end-to-end fashion, and makes **DeepGTT** applicable to large-scale data sets. **DeepGTT** interprets the generation of travel time in a plausible manner rather than learning by brute force, thus it produces more accurate results and is data-efficient.

Evaluated on a real-world large-scale data set, we first demonstrate the superiority of **DeepGTT** over existing methods, including two recently proposed deep-neural-net-based approaches, in two tasks: travel time estimation and route recovery. Notably, even trained with a small fraction of data **DeepGTT** is able to produce substantially better results than the state-of-the-art baselines. We further design experiments to verify that

an appropriate modeling of real-time traffic is crucial for accurate travel time distribution prediction, and the assumption made in previous studies that traffic conditions are temporally-invariant does not hold. Finally, we empirically compare the time cost of DeepGTT with the other two deep-neural-net-based approaches, which demonstrates that DeepGTT owns much faster training speed.

In Chapter 5, we consider the problem of *spatial transition learning on the road networks*, which is formally defined as follows. Given a road network $G(V, E)$ and a historical trajectory dataset $\mathcal{D} = \{T^m\}_{m=1}^M$, as well as the starting time, origin and destination of a trip \mathbf{T} , we aim to predict the most likely traveled route of the trip \mathbf{T} as well as score the likelihood of any route being traveled by conditioning on the real-time traffic.

We propose a novel probabilistic generative model to the problem. For the first time, we unify three key explanatory factors—past traveled route, destination and real-time traffic—for the spatial transition modeling with a deep probabilistic model—DeepST. DeepST achieves this by explaining the generation of next route conditioned on the representations of the three explanatory factors in a principled way. The past traveled route is compressed with a RNN, and thus could account for the long range dependency. To enable effectively sharing statistical strength across trips, we propose an adjoint generative process to learn representations of k -destination proxies rather than learning the destination representations separately. The introduction of the latent variable \mathbf{c} allows us to incorporate the impact of real-time traffic by inferring its posterior distribution. Lastly, an efficient inference algorithm is developed within the VAEs framework to scale DeepST to large-scale datasets. The experiments are conducted on two real-world large-scale trajectory datasets to demonstrate the superiority of DeepST over the competitors on two tasks: the most likely route prediction and route recovery from sparse trajectories. Remarkably, on the Chengdu trajectory dataset, DeepST surpasses the best competing method by almost 50% on the most likely route prediction task and up to 15% on the route recovery task in terms of accuracy.

6.2 Future Work

The work of *trajectory representation learning* presented in Chapter 3 sheds light on several new research directions: 1) Employing the learned representations to explore more downstream tasks, e.g., outlier detection, and popular-routes search. 2) Extending the proposed method to more general time series data beyond trajectories. 3) Developing indexing techniques to further speed up the proposed method. The first direction might be achieved by performing trajectory clustering with the representations produced by `t2vec`; the second direction is more challenging since the general time series have more

semantic information to be captured; the third one seeks to further accelerate the computation based on the learned representations, this might be achieved by the combination of Locality-Sensitive Hashing [AI06] and deep learning.

For the future work of *travel time distribution learning* presented in Chapter 4, we would like to enhance the performance of DeepGTT by investigating more rich posterior distribution for the real time traffic representation with the normalizing flows [RM15, vdBHTW18]. In the present solution, DeepGTT assumes that the travel times of longer routes tend to show higher variance which may not hold for highways, and we will attempt to address this in the future work, by designing a road link type aware variance function. It is also desirable to explore the usage of DeepGTT in the problems such as ride-sharing, taxi-dispatching, travel time-related trajectory anomaly detection and time-aware trajectory representation learning [LZC⁺18, YCZB19].

For the *spatial transition learning* presented in Chapter 5, we consider the following future directions. As the past traveled route $\mathbf{r}_{1:i}$ is generated by the model itself on the prediction stage, which may deviate from the ground truth and lead to accumulated errors, we plan to address it by exploring GAN-like loss function [GLZ⁺16] or structure prediction. We also would like to explore the usage of DeepST in popular route discovery, anomaly trips detection, etc.

- **Xiucheng Li**, Gao Cong, Yun Cheng. Spatial Transition Learning on Road Networks with Deep Probabilistic Models. 36th IEEE International Conference on Data Engineering (**ICDE 2020**), Dallas, USA, Texas, April 20-24, 2020.
- **Xiucheng Li**, Gao Cong, Aixin Sun, Yun Cheng. Learning Travel Time Distributions with Deep Generative Model. The World Wide Web Conference (**WWW 2019**), pages 1017-1027, San Francisco, CA, USA, May 13-17, 2019.
- **Xiucheng Li**, Kaiqi Zhao, Gao Cong, Christian S. Jensen, Wei Wei. Deep Representation Learning for Trajectory Similarity Computation. 34th IEEE International Conference on Data Engineering (**ICDE 2018**), pages 617-628, Paris, France, April 16-19, 2018.
- Shanshan Feng, Gao Cong, Arijit Khan, **Xiucheng Li**, Yong Liu, Yeow Meng Chee. Inf2vec: Latent Representation Model for Social Influence Embedding. 34th IEEE International Conference on Data Engineering (**ICDE 2018**), pages 941-952, Paris, France, April 16-19, 2018.
- **Xiucheng Li**, Yun Cheng, Gao Cong, Lisi Chen. Discovering Pollution Sources and Propagation Patterns in Urban Area. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (**KDD 2017**), pages 1863-1872, Halifax, NS, Canada, August 13-17, 2017.

References

- [ADFDJ03] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.
- [AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 459–468, 2006.
- [AMSS11] Akinori Asahara, Kishiko Maruyama, Akiko Sato, and Kouichi Seto. Pedestrian-movement prediction based on mixed markov-chain model. In *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, November 1-4, 2011, Chicago, IL, USA, Proceedings*, pages 25–33, 2011.
- [ASKP03] Jonathan Alon, Stan Sclaroff, George Kollios, and Vladimir Pavlovic. Discovering clusters in motion time-series data. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), 16-22 June 2003, Madison, WI, USA*, pages 375–381, 2003.
- [BCB14a] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.
- [BCB14b] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [BCN18] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

REFERENCES

- [BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.
- [BDVJ03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.
- [BEKS17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.
- [Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.
- [BKM16] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *CoRR*, abs/1601.00670, 2016.
- [BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
- [BPRS17] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:153:1–153:43, 2017.
- [BRR14] Prithu Banerjee, Sayan Ranu, and Sriram Raghavan. Inferring uncertain trajectories from partial observations. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 30–39, 2014.
- [CGCB14a] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [CGCB14b] Junyoung Chung, Çağlar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

REFERENCES

- [CLH⁺16] Chen Chen, Cewu Lu, Qixing Huang, Qiang Yang, Dimitrios Gunopulos, and Leonidas J. Guibas. City-scale map creation and updating using GPS collections. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1465–1474, 2016.
- [CLL⁺14a] Yun Cheng, Xiucheng Li, Zhijun Li, Shouxu Jiang, and Xiaofan Jiang. Fine-grained air quality monitoring based on gaussian process regression. In *ICONIP 2014*, pages 126–134, 2014.
- [CLL⁺14b] Yun Cheng, Xiucheng Li, Zhijun Li, Shouxu Jiang, Yilong Li, Ji Jia, and Xiaofan Jiang. Aircloud: a cloud-based air-quality monitoring system for everyone. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems, SenSys '14, Memphis, Tennessee, USA, November 3-6, 2014*, pages 251–265, 2014.
- [CLY14] Meng Chen, Yang Liu, and Xiaohui Yu. NLPMM: A next location predictor with markov modeling. In *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part II*, pages 186–197, 2014.
- [CMC07] Huiping Cao, Nikos Mamoulis, and David W. Cheung. Discovery of periodic patterns in spatiotemporal sequences. *IEEE Trans. Knowl. Data Eng.*, 19(4):453–467, 2007.
- [CN04] Lei Chen and Raymond Ng. On the marriage of lp-norms and edit distance. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 792–803, 2004.
- [CÖO05] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 491–502, 2005.
- [CSZ11a] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. Discovering popular routes from trajectories. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 900–911, 2011.

REFERENCES

- [CSZ11b] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. Discovering popular routes from trajectories. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 900–911, 2011.
- [CvMG⁺14] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734, 2014.
- [CYZ13] Hong Cheng, Jihang Ye, and Zhe Zhu. What’s your next move: User activity prediction in location-based social networks. In *Proceedings of the 13th SIAM International Conference on Data Mining, May 2-4, 2013. Austin, Texas, USA.*, pages 171–179, 2013.
- [DHNZ95] Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, pages 889–904, 1995.
- [EHW⁺16] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems 29: NIPS 2016, December 5-10, 2016, Barcelona, Spain*, pages 3225–3233, 2016.
- [FCAC17] Shanshan Feng, Gao Cong, Bo An, and Yeow Meng Chee. Poi2vec: Geographical latent representation for predicting future visitors. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 102–108, 2017.
- [FCK⁺18] Shanshan Feng, Gao Cong, Arijit Khan, Xiucheng Li, Yong Liu, and Yeow Meng Chee. Inf2vec: Latent representation model for social influence embedding. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 941–952, 2018.

REFERENCES

- [FGT07] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. Index-based most similar trajectory search. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 816–825, 2007.
- [FLZ⁺15] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. Personalized ranking metric embedding for next new POI recommendation. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2069–2075, 2015.
- [FLZ⁺18] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. Deepmove: Predicting human mobility with attentional recurrent networks. In *Proceedings of the 2018 World Wide Web Conference, WWW 2018, Lyon, France, April 23-27, 2018*, pages 1459–1468, 2018.
- [FZW⁺19] Jie Feng, Mingyang Zhang, Huandong Wang, Zeyu Yang, Chao Zhang, Yong Li, and Depeng Jin. Dplink: User identity linkage via deep neural network from heterogeneous mobility data. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 459–469, 2019.
- [GCB14] Prem K Gopalan, Laurent Charlin, and David Blei. Content-based recommendations with poisson factorization. In *Advances in Neural Information Processing Systems 27: NIPS 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3176–3184, 2014.
- [GCSR95] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 1995.
- [GFGS06] Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 369–376, 2006.

REFERENCES

- [GH10] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 297–304, 2010.
- [GHB13] Prem Gopalan, Jake M Hofman, and David M Blei. Scalable recommendation with poisson factorization. *CORR*, 2013.
- [GLZ⁺16] Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron C. Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4601–4609, 2016.
- [Gra13] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- [GS95] Ralf Hartmut Güting and Markus Schneider. Realm-based spatial data types: the rose algebra. *VLDBJ*, 4(2):243–286, 1995.
- [GSA14] Raghu K. Ganti, Mudhakar Srivatsa, and Tarek F. Abdelzaher. On limits of travel time predictions: Insights from a new york city case study. In *IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014, Madrid, Spain, June 30 - July 3, 2014*, pages 166–175, 2014.
- [HAG⁺12] Liangjie Hong, Amr Ahmed, Siva Gurumurthy, Alexander J. Smola, and Kostas Tsioutsoulis. Discovering geographical topics in the twitter stream. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 769–778, 2012.
- [HBWP13] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

REFERENCES

- [HHR⁺13] Timothy Hunter, Aude Hofleitner, Jack Reilly, Walid Krichene, Jerome Thai, Anastasios Kouvelas, Pieter Abbeel, and Alexandre M. Bayen. Arriving on time: estimating travel time distributions on large-scale road networks. *CoRR*, abs/1302.6617, 2013.
- [HOT06] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.
- [HPL15] Chih-Chieh Hung, Wen-Chih Peng, and Wang-Chien Lee. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *VLDBJ*, 24(2):169–192, 2015.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.
- [inv]
- [IS11] Tsuyoshi Idé and Masashi Sugiyama. Trajectory regression on road networks. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011.
- [JCMB15] Sébastien Jean, KyungHyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1–10, 2015.
- [JGP16] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *CoRR*, abs/1611.01144, 2016.

REFERENCES

- [JYZ⁺08] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S Jensen, and Heng Tao Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1):1068–1080, 2008.
- [KB14a] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [KB14b] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR 2015*, abs/1412.6980, 2014.
- [KMRW14] Diederik P. Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems 27: NIPS 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3581–3589, 2014.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.
- [KTR⁺17] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18:14:1–14:45, 2017.
- [KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems 30: NIPS 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 972–981, 2017.
- [KW13] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *ICLR 2013*, abs/1312.6114, 2013.
- [KZS⁺15] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3294–3302, 2015.

REFERENCES

- [LAS15] Mu Li, Amr Ahmed, and Alexander J. Smola. Inferring movement trajectories from GPS snippets. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM 2015, Shanghai, China, February 2-6, 2015*, pages 325–334, 2015.
- [LBE⁺12] Xuemei Liu, James Biagioni, Jakob Eriksson, Yin Wang, George Forman, and Yanmin Zhu. Mining large-scale, sparse GPS traces for map inference: comparison of approaches. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 669–677, 2012.
- [LCC20] Xiucheng Li, Gao Cong, and Yun Cheng. Spatial transition learning on road networks with deep probabilistic models. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, Texas, USA, April 20-24, 2020*, 2020.
- [LCH⁺17] Guanyao Li, Chun-Jie Chen, Sheng-Yun Huang, Ai-Jou Chou, Xiaochuan Gou, Wen-Chih Peng, and Chih-Wei Yi. Public transportation mode detection from cellular data. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 2499–2502, 2017.
- [LCJT13] Xiaohui Li, Vaida Ceikute, Christian S Jensen, and Kian-Lee Tan. Effective online group discovery in trajectory databases. *IEEE Trans. Knowl. Data Eng.*, 25(12):2752–2766, 2013.
- [LCSC19] Xiucheng Li, Gao Cong, Aixin Sun, and Yun Cheng. Learning travel time distributions with deep generative model. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 1017–1027, 2019.
- [LFW⁺18] Yaguang Li, Kun Fu, Zheng Wang, Cyrus Shahabi, Jieping Ye, and Yan Liu. Multi-task representation learning for travel time estimation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, SIGKDD 2018, London, UK, August, 2018*, 2018.

REFERENCES

- [LHJ⁺11] Zhenhui Li, Jiawei Han, Ming Ji, Lu An Tang, Yintao Yu, Bolin Ding, Jae-Gil Lee, and Roland Kays. Movemine: Mining moving object data for discovery of animal movement patterns. *ACM TIST*, 2(4):37:1–37:32, 2011.
- [LHL08] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico*, pages 140–149, 2008.
- [LKHJ18] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 689–698, 2018.
- [LM14] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196, 2014.
- [LQZH12] Liangxu Liu, Shaojie Qiao, Yongping Zhang, and JinSong Hu. An efficient outlying trajectories mining approach based on relative distance. *International Journal of Geographical Information Science*, 26(10):1789–1810, 2012.
- [LWWT16] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. Predicting the next location: A recurrent model with spatial and temporal contexts. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 194–200, 2016.
- [LZC⁺18] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. Deep representation learning for trajectory similarity computation. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 617–628, 2018.
- [MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013.

REFERENCES

- [MMT16] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016.
- [MPTG09] Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 637–646, 2009.
- [MRM12] Wesley Mathew, Ruben Raposo, and Bruno Martins. Predicting future locations with hidden markov models. In *The 2012 ACM Conference on Ubiquitous Computing, Ubicomp '12, Pittsburgh, PA, USA, September 5-8, 2012*, pages 911–918, 2012.
- [MSC⁺13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.
- [Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. Cambridge, MA, 2012.
- [NK09] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, SIGSPATIAL 2009, November 4-6, 2009, Seattle, Washington, USA, Proceedings*, pages 336–343, 2009.
- [PAS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 701–710, 2014.

REFERENCES

- [PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318, 2013.
- [PNI⁺08] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 569–577. ACM, 2008.
- [RDT⁺15] Sayan Ranu, P Deepak, Aditya D Telang, Prasad Deshpande, and Sriram Raghavan. Indexing and matching trajectories under inconsistent sampling rates. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 999–1010, 2015.
- [RKK18] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. 2018.
- [RM15] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1530–1538, 2015.
- [RMW14] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1278–1286, 2014.
- [SAM⁺13] Swaminathan Sankararaman, Pankaj K Agarwal, Thomas Mølhave, Jiangwei Pan, and Arnold P Boedihardjo. Model-driven matching and segmentation of trajectories. In *21st SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2013, Orlando, FL, USA, November 5-8, 2013*, pages 234–243, 2013.
- [SHM⁺16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

REFERENCES

- [SLY11] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *INTERSPEECH 2011, 12th Annual Conference of the International Speech Communication Association, Florence, Italy, August 27-31, 2011*, pages 437–440, 2011.
- [SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.
- [SZW⁺13a] Han Su, Kai Zheng, Haozhou Wang, Jiamin Huang, and Xiaofang Zhou. Calibrating trajectory data for similarity-based analysis. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 833–844, 2013.
- [SZW⁺13b] Han Su, Kai Zheng, Haozhou Wang, Jiamin Huang, and Xiaofang Zhou. Calibrating trajectory data for similarity-based analysis. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 833–844, 2013.
- [TJ08] Dalia Tiesyte and Christian S. Jensen. Similarity-based prediction of travel times for vehicles traveling on known routes. In *16th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, SIGSPATIAL 2008, November 5-7, 2008, Irvine, California, USA, Proceedings*, page 14, 2008.
- [TQW⁺15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 1067–1077, 2015.
- [vdBHTW18] Rianne van den Berg, Leonard Hasenclever, Jakub M. Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 393–402, 2018.

REFERENCES

- [vdOVK17] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems 30: NIPS 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6309–6318, 2017.
- [VKG02] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 673–684, 2002.
- [VKK⁺15] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2773–2781, 2015.
- [WCS⁺17] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. Modeling trajectories with recurrent neural networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3083–3090, 2017.
- [Wer90] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [WFY18] Zheng Wang, Kun Fu, and Jieping Ye. Learning to estimate the travel time. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 858–866, 2018.
- [WH86] DRGHR Williams and GE Hinton. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.
- [WKKL16] Hongjian Wang, Yu-Hsuan Kuo, Daniel Kifer, and Zhenhui Li. A simple baseline for travel time estimation using large-scale trip data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2016, Burlingame, California, USA, October 31 - November 3, 2016*, pages 61:1–61:4, 2016.

REFERENCES

- [WMS⁺16] Hao Wu, Jiangyun Mao, Weiwei Sun, Baihua Zheng, Hanyuan Zhang, Ziyang Chen, and Wei Wang. Probabilistic robust route recovery with spatio-temporal dynamics. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, SIGKDD 2016, San Francisco, CA, USA, August 13-17, 2016*, pages 1915–1924, 2016.
- [WORN11] Monica Wachowicz, Rebecca Ong, Chiara Renso, and Mirco Nanni. Finding moving flock patterns among pedestrians through collective coherence. *International Journal of Geographical Information Science*, 25(11):1849–1864, 2011.
- [WSZ⁺13] Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq, and Xiaofang Zhou. An effectiveness study on trajectory similarity measures. In *Twenty-Fourth Australasian Database Conference, ADC 2013, Adelaide, Australia, February 2013*, pages 13–22, 2013.
- [WZC⁺18] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. When will you arrive? estimating travel time based on deep neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI 2018, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018.
- [WZX14] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, SIGKDD 2014, New York, NY, USA - August 24 - 27, 2014*, pages 25–34, 2014.
- [XZZ⁺] Andy Yuan Xue, Rui Zhang, Yu Zheng, Xing Xie, Jin Huang, and Zhenghua Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *29th IEEE International Conference on Data Engineering, Brisbane, Australia, April 8-12, 2013, pages = 254–265, year = 2013*.
- [YCH⁺11] Zhijun Yin, Liangliang Cao, Jiawei Han, Chengxiang Zhai, and Thomas S. Huang. Geographical topic discovery and comparison. In *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 247–256, 2011.

REFERENCES

- [YCL14] Quan Yuan, Gao Cong, and Chin-Yew Lin. COM: a generative model for group recommendation. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 163–172, 2014.
- [YCM⁺] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat-Thalmann. Who, where, when and what: discover spatio-temporal topics for twitter users. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013, pages = 605–613, year = 2013*.
- [YCZB19] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pages 1358–1369, 2019.
- [YJF98] Byoung-Kee Yi, HV Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23-27, 1998*, pages 201–208, 1998.
- [YLWT11] Josh Jia-Ching Ying, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S. Tseng. Semantic trajectory mining for location prediction. In *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, November 1-4, 2011, Chicago, IL, USA, Proceedings*, pages 34–43, 2011.
- [YYSL16] Xinchun Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2image: Conditional image generation from visual attributes. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 776–791, 2016.
- [ZN13] Jiangchuan Zheng and Lionel M. Ni. Time-dependent trajectory regression on road networks via multi-task learning. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2013, July 14-18, 2013, Bellevue, Washington, USA., 2013*.

REFERENCES

- [ZXZ⁺18] Jing Zhao, Jiajie Xu, Rui Zhou, Pengpeng Zhao, Chengfei Liu, and Feng Zhu. On prediction of user destination by sub-trajectory understanding: A deep learning based approach. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 1413–1422, 2018.
- [ZZXZ12] Kai Zheng, Yu Zheng, Xing Xie, and Xiaofang Zhou. Reducing uncertainty of low-sampling-rate trajectories. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 1144–1155, 2012.