ö z p n „ h : f `

3 ÷ N ' f å f Z ë f Mº ‡

ù { U M ¤ É á o v b
f Ñ ¡ —: Ñ f € /
v ù ®
ü Y N ú o Y ^

Œ k 年 m

On Learning Representations of Spatio-temporal Data

Dissertation Submitted to
Tsinghua University
in partial fulfillment of the requirement
for the degree of
Doctor of Philosophy
in
Computer Science and Technology

by

Wei Cao

Dissertation Supervisor : Associate Professor Jian Li

June, 2018

s Ž f Mº ‡ • (ˆ C „ ô

, º Œh † ã N ' f s Ý Y • (f Mº ‡ „ Ä š s

N ' f å (W\ C Õ Ä š ô …f Mº ‡ „ • (C v -

ì 1 ò · f M„ 研究 必须按f 校Ä š 提¤ f Mº ‡ f 校

ï å 采(影p 缩p v Ö 制K 段Ý X 研究 ¤ „ f Mº ‡

2 : Y f 和Ñ 研î„ f 校ï å l 开„ f Mº ‡ \ : 资™(þ

f 馆 资™¤ 等: @› 校…师 阅û (校í 网 › 校…师 浏

È 部分…[1]

, º Ý Á 遵ˆ 述Ä š

Ý Ý „ Æ „ º º ‡ ‡ (ã ã ã Æ 后应遵ˆ 此Ä š š

\ 者签名 _____ ü 师签名 _____

å _____ å _____

摘 要

(现ã > - ú Ž M置„ 务系ß (LBS):ºì „ å 常 活带e † • ' „ ¿ 利 此同ö LBS 务每) 都 •• - „ § ú ' 量„ B q „ p 据 这› p 据- s 含† ö ô á o 同ö _ 含† 空ô á o à 此 ì 这类1 LBS 务§ „ p 据ë Zöö 空p 据 spatio-temporal data ö 空p 据„ 挖掘: ¸ å 和F 应(带e † è ' „ \ 力 ‹ , ¤ 通流分• [1–3] Å 8 路径推P [4–7] å Ê ú Ž 0 M置„ > ¤ 网Ü [8–11] :† (ö 空p 据„ 挖掘- 得到 I 并 Æ确„ Ó œ * s . „ 步¤ / Î Ÿ Ë ö 空p 据- ½ Ö ú Hyy 征hh : feature representations

(这篇º ‡ - ì 研究, U Î ö 空p 据- f ` ú ƒ „ H y 征h : 1 Ž ö 空p 据ú „ y 征 ö 空p 据„ h : f ` b 4 ¸ 挑 首H ö 空p 据 - 往往同ö 含^ 常 B „ 空ô s 联和ö ô • 赖 同ö ö 空p 据还× 到ø 部à „ 影响 ‹ ,) ¶ µ å Ê , G å 等 * H „ y 征h : 应å 能 · Ö 到p 据- „ 这› s 联s 系 æ (¸ å 应(: o - ö 空p 据^ 常 B q 4 随' 量j ó p 据Ä ! è ' 这Ù ö 空p 据„ h : f ` 带e † ^ 常 ' „ ð 难 ì 通过研究 * w S „ î 题 U : † , U ã ³ 述挑 并 H f ` 到p 据y 征h :

(第 章 ì 考Q , U (p 据„ h 征- 捕· B „ ö 空• 赖' ìå ¤ 通到达ö ô 预0 î 题: ‹ (这* î 题- ì 研究, U ù û Ù š „ 路 ¿ 通常h : : * 连通„ 路径序列 0 ¡ 通过这a 路径„ ~v ö ô ì ú Ž 深度^ Ï 网Ü 提ú * 端到端„ 框¶ ë Z DeepTTE (DeepTTE ì 提ú † * ú Ž 0 w ï 操\ geo-convolution e 捕· 同路径K ô „ 空ô • 赖' ì (0 w ï B à 加† * 递归B recurrent layer e 进步捕捉 同路段K ô „ ö 空• 赖' 述Ó „ ï å ô 接Î Ÿ Ë p 据- f ` 到 H „ y 征h : ( ! ‹ 顶端 ì 利( * û 务f ` ! W multi-task learning ú ŽK 前得到„ y 征h : 预测道路通达ö ô

(第Û 章 ì 研究当X (p 据缺1 ö , U 进L ö 空p 据„ h : f

` ì i U † 第 章- „ Ó œ 并提ú Í h ° „ p 据缺1 e h „ ¹ Õ ë Z BRITS Bi-directional Recurrent Imputation Time Series ì „ ! ‹ ù p 据„ 过程Z û U H ŒG ¾ ì 利(Ì 向„ 递归^ Ï 网Ü 捕· p 据- „ • 赖s 系 并 f 到„ • 赖s 系h : : 隐空ô - „ h : 向量 ú Ž h : 向量 ì ù 缺1 p 据进L ô 接预测 (ì ! ‹ - ì 预测得到„ 缺1 p 据\ : RNN¡ 算þ „ 部分 à 此 e h ï î 能 Œt Þ å ô ° ! ‹ 进而• 得 e h Ó œô : Æ确

(第” 章 ì 进 步研究(• 端 B „ ö 空p 据- , U 进Lh : f ` ìå • 轨迹p 据 heterogeneous mobility dataset „ (7 Æ别î 题: ‹ (这* î 题- ì 重¹ s è ö 空p 据„ B q ' ' Ä ! ' ì ' Ä ! 轨迹p 据- „ (7 Æ别î 题进L „ 形式 š I ì 提ú * ú Ž MapReduce 框¶ ë Z Automatic User Identification AUI ì „ 框¶ ï å H 海量 „ 轨迹p 据 框¶ ú Ž * h ° „ 轨迹ø < 度ý p ë Z Signal Based Similarity SIG SIGï å H 测量 同采样率„ Ø j ó 轨迹p 据K ô „ø < ' (SIG- ì (7 轨迹进L ½ Ö 转换: “q 现” co-occurrence ‹ö h : ì (同粒度 分别ú Ž q 现‹ö h : ¡ 算轨迹ô „ø < ' 并进L G ; Î 而得到 È „ø < ' ì SIG和v Ö 现 „ 轨迹ø < ' ý p 进L 比较 ž ŒÓ œh ì „ ! ‹ w ô 强„ • 棒' 和Æ确'

s . Í ö 空p 据 h : f ` 深度^ Ï 网Ü

Abstract

The ubiquity of location-based services greatly benefits people's daily life. In consequence, a large volume of massive data records has been generated routinely in location-based services. Such data records contain both location information and time information, which we denote as *spatio-temporal data*. Mining spatio-temporal data shows great potentials in many industrial and commercial applications, such as traffic analysis [1–3], travel recommendation [4–7] and location-based social network [8–11]. In order to achieve meaningful and accurate mining results, an important step is to extract effective feature representations from the raw data.

In this thesis, we study learning representations of spatio-temporal data. Due to characteristics of spatio-temporal data, learning representations of such data faces many challenges. First, spatio-temporal data usually contains complex spatial and temporal correlations simultaneously, and it is also affected by various external factors such as the weather conditions and holidays. An effective feature representation should be able to capture such correlations in data. Furthermore, in many industrial scenarios, spatio-temporal data is massive, noisy, and in very large scales, which makes it hard to obtain accurate feature representations. We study three concrete problems to illustrate how to learn the spatio-temporal data representations while addressing such challenges.

In Chapter 3, we study capturing complex correlations in data representations. We use travel time estimation problem as an example. In this problem, we estimate the travel time of any given path (denoted by a sequence of connected roads). We propose an end-to-end framework based on deep neural networks called DeepTTE. Specifically, we present a geo-convolution operation for capturing the spatial correlations of different roads. We stack a recurrent layer on the geo-convolution layer to further capture the temporal dependencies of these roads. Such architecture enables us to extract effective feature representations from raw data. A multi-task learning component is given on the top of DeepTTE, that learns the travel time based on the extracted features.

In Chapter 4, we study handing missing values while learning spatio-temporal data representations. We extend the result in Chapter 3 and propose BRITS (Bi-directional Recurrent Imputation Time Series), a novel method for imputing missing values in spatio-temporal data. Our method does not impose any assumption about the data-generating process. Alternatively, we use a bi-directional recurrent neural network (RNN) to learn correlations in data and transform the learned correlations into several hidden space representations. We predict the missing values directly based on the hidden representations. Our model treats the missing values as variables of RNN graph. Thus, the imputation errors can be fully backpropagated which makes the imputation more accurate.

In Chapter 5, we further study learning representations for extremely massive spatio-temporal data. We use the user identification problem as an example, where we are supposed to identify users across heterogeneous mobility data sources. In this problem, we focus on the massiveness and scales of spatio-temporal data. We formulate the user identification problem over large-scale mobility datasets. We present a MapReduce-based framework called Automatic User Identification (AUI) which can scale to very large datasets. Our framework is based on a novel similarity measure called the *signal based similarity* (SIG) which measures the similarity of users' trajectories gathered from different data sources, typically with very different sampling rates and noise patterns. In signal based similarity, we transform users' trajectories to a series of "co-occurrence" events representations. We calculate the similarity scores based on such representations in multiple resolutions. We show that comparing with other existing similarity measures, SIG is much more accurate and robust for very massive mobility datasets.

Key words: Spatio-temporal data; Representation learning; Deep neural networks.

î U

, 1 à Introduction 1
1.1 Background 1
1.2 Capturing Complex Correlations 3
1.3 Imputing Missing Values 3
1.4 Handling Massiveness 4
1.5 Organization 5
, 2 à Preliminary 6
2.1 Deep Neural Networks 6
2.1.1 Convolutional Neural Networks 2D 6
2.1.2 Convolutional Neural Networks 1D 7
2.1.3 Recurrent Neural Networks 7
2.2 Map-Reduce Framework 8
, 3 à Capturing Data Correlations in Travel Time Estaimation 10
3.1 Introduction 10
3.2 Preliminary 12
3.3 Model Architecture 13
3.3.1 Attribute Component 14
3.3.2 Spatio-Temporal Component 15
3.3.2.1 Geo-Conv Layer 16
3.3.2.2 Recurrent Layer 18
3.3.3 Multi-task Learning Component 19
3.3.3.1 Estimate the local paths 19
3.3.3.2 Estimate the entire path 20
3.3.4 Model Training 21
3.4 Experiment 22
3.4.1 Experiment Setting 22
3.4.1.1 Data Description 22
3.4.1.2 Parameter Setting 23
3.4.2 Performance Comparison 24

3.4.3 Effect of Attribute Component 26
3.4.4 Effect of Geo-Conv 27
3.4.5 Effect of Multi-task Learning 27
3.4.6 Incorporate Road Information 27
3.4.7 Predicting Time 28
3.5 Related work 28
3.5.1 Road Segment-Based Travel Time Estimation 28
3.5.2 Path-Based Travel Time Estimation 29
3.5.3 Deep Learning in Spatial Temporal Data 30
3.6 Conclusion 30
, 4 à Missing Value Imputation with Recurrent Dynamics 31
4.1 Introduction 31
4.2 Preliminary 33
4.3 BRTIS 34
4.3.1 Unidirectional Uncorrelated Recurrent Imputation 34
4.3.1.1 Algorithm 35
4.3.1.2 Practical Issues 37
4.3.2 Bidirectional Uncorrelated Recurrent Imputation 37
4.3.3 Correlated Recurrent Imputation 38
4.4 Experiment 40
4.4.1 Dataset Description 40
4.4.1.1 Air Quality Data 40
4.4.1.2 Health-care Data 40
4.4.1.3 Localization for Human Activity Data 41
4.4.2 Experiment Setting 41
4.4.2.1 Model Implementations 41
4.4.2.2 Baseline Methods 42
4.4.3 Experiment Results 44
4.5 Related Work 46
4.6 Conclusion 47

, 5 à Handling Massiveness in User Identification 49
5.1 Introduction 49
5.2 Formulation and Overview 53
5.3 Pre-Processing 55
5.4 Multi-Resolution Filtering 57
5.5 Verification Stage 59
5.5.1 Signal Based Similarity 59
5.5.2 Verification 63
5.6 Experiment Evaluation 64
5.6.1 Experiment Setting 65
5.6.2 Effects of Parameters 67
5.6.2.1 Effects of Q 68
5.6.2.2 Effects of N 68
5.6.2.3 Effects of m_c 70
5.6.2.4 Effects of 71
5.6.2.5 Effects of Distance Values between Kernel Cells 71
5.6.3 Comparisons with Other Algorithms 73
5.6.4 Discussions 75
5.7 Related Work 76
5.8 Conclusion 77
, 6 à Conclusions and Future Work 79
6.1 Conclusions 79
6.2 Future Directions 81
Â ‡ 83
ô " 91
ð 92
D U A - ‡ 摘要 93
A.1 引 93
A.1.1 研究背o 93
A.1.2 捕· B p 据• 赖 94
A.1.3 e h 缺1 p 据 95
A.1.4 B p 据环 f 95

A.1.5 º ‡ ÄÇ 96
A.2 ö ô 预0 î 题- „ B p 据• 赖..... 96
A.2.1 引 97
A.2.2 l 式 š l ! ‹ ¶„ 98
A.2.3 ø s å \ 100
A.2.3.1 分段预测¹ Õ 100
A.2.3.2 t S 预测¹ Õ 100
A.2.3.3 ú Ž 深度f ` „ 预测¹ Õ 101
A.3 ú Ž 循环动力系ß „ 缺1 e h 101
A.3.1 引 101
A.3.2 l 式 š l 102
A.3.3 ø s å \ 103
A.4 B p 据环ƒ „ (7 « ý Æ别..... 104
A.4.1 引 104
A.4.2 l 式 š l ! ‹ ¶„ 107
A.4.3 ø s å \ 108
A.5 ; Ó 110
A.6 *e å \ 111
*º € † (f ô Ñ h „ f / º ‡ v œ 113

, 1 à Introduction

1.1 Background

In recent years, the advances in communication technologies and the ubiquity of location-based services (LBS) greatly benefit people's daily life. For example, people use their GPS embedded devices to search the destinations, plan the routes or share their mobilities; institutes use the loop sensors to monitor the traffic or schedule the vehicles. In these scenarios, a large volume of massive data records are generated routinely. The data records contain both the location information and the time information, which we denote as the *spatio-temporal data*. Mining massive spatio-temporal data shows great potentials in various industrial and commercial applications, such as traffic analysis[1–3], travel recommendation[4–7], and location-based social network[8–13].

A typical way in mining spatio-temporal data contains two phases. In the first phase, we extract useful information from the raw data and represent such information as a fixed length feature vector. In the second phase, based on the extracted features, we train a machine learning model (e.g., logistical regression, probabilistic graphical model) to solve the mining task. In order to achieve meaningful and accurate results, it is important to extract effective feature representations of the raw data in the first stage. However, due to characteristics of spatio-temporal data, learning representations of such data is highly non-trivial and faces many challenges:

- (Complex correlations) The spatio-temporal data usually contains complex correlations. Due to its inherent characteristics, the spatial and temporal correlations both exist in the data simultaneously. For example, in a traffic system, the traffic condition in one road is correlated with its historical conditions, as well as its neighboring roads. Furthermore, the spatio-temporal patterns are affected by various of external factors, such as weather conditions and holidays. Effective feature representations should be able to capture such complex correlations

in data.

- (Sensitive to granularities)The spatio-temporal data is sensitive to the granularities. The information loss caused by the coarse granularities can cause damages to the model performance. Comparing with vision or natural language tasks, for spatio-temporal data, it is important to retain the fine granularity information in the data when learning its representations. As an example, in image classifications, if we replace the original images with relatively coarse images, we can still identify the image classes (e.g., dogs, cats) with high probability. However, in the urban traffic, if we mix up two road segments, even if the road segments are very close, their properties (e.g., congestion level, traffic flow) can be extremely different.
- (Missing values)The spatio-temporal data is usually very noisy and irregular. Due to the communication error or unexpected device error, it is common that the data contains a large number of missing values. Existing methods often use interpolation or smoothing methods to impute the missing values [14–17]. However, since the correlations in spatio-temporal data are much more complex, simply adopting the smoothing values can lead to very inaccurate results [18].
- (Massiveness)In many industrial applications, the spatio-temporal data is very massive and in very large scales. For example, for Google Map Services, there are as many as one billion monthly active users and the patterns of different users in using the services can be very different. We should also consider the robustness of feature representations for some extreme cases and the efficiency for large-scale datasets.

In this thesis, we study learning representations of spatio-temporal data while addressing the above-mentioned challenges. Specifically, we introduce three concrete problems to illustrate how to handle such challenges in learning the spatio-temporal data representations. We present the descriptions of the problems and summarize our contributions in the following sections.

1.2 Capturing Complex Correlations

In Chapter 3, we study capturing complex correlations in data representations. The granularity sensitive issue is also considered in this chapter. Specifically, we use the travel time estimation problem as an example. Given a specific path, denoted by a sequence of connected road segments, the corresponding start time, and external factors such as weather conditions and driving habits, our objective is to estimate the time of traveling through this path. In this problem, the travel time is affected by diverse complex factors, including the spatial correlations, the temporal dependencies, and the external factors. Prior work [19–22] usually focuses on estimating the travel times of individual road segments or sub-paths and then summing up these times. Such methods did not consider the correlations between different road segments, which lead to inaccurate estimations.

Contributions: We propose an end-to-end *Deep* learning framework for *Travel Time Estimation* (called *DeepTTE*), that estimates the travel time of the whole path directly. We present a geo-convolution operation for capturing the spatial correlations of road segments in a fine granularity, by integrating the geographic information into the classical convolution. We stack a recurrent layer on the geo-convolution layer to further capture the temporal dependencies. With such architecture, we can directly learn effective feature representations from raw data. A multi-task learning component is given on the top of DeepTTE, that learns the travel time of both the entire path and each local path at the same time. Extensive experiments on two trajectory datasets show that our DeepTTE significantly outperforms the state-of-the-art methods.

1.3 Imputing Missing Values

In Chapter 4, we study handling the missing values while learning representations of spatio-temporal data. Given multiple correlated time series data, our objective is to predict their class labels and to fill in missing values at the same time. Existing imputation methods often impose strong assumptions of the underlying data generating process, such as linear dynamics in the state space [14,16,17,23]. However, since the spatio-

temporal data usually contains complex correlations, such assumptions may not hold.

Contributions: In this chapter, we propose BRITS_i(Bidirectional_Recurrent Imputation_Time Series), a novel method for imputing missing values. Our proposed method does not impose any assumption about the data-generating process. Instead, we use a bidirectional recurrent neural network to capture both the spatial and temporal correlations in the data and represents the captured correlations as several hidden states. We then predict missing values based on the obtained hidden states. We evaluate our model on one synthetic dataset and two real-world datasets. Experiments show that our model outperforms the state-of-the-art methods in both imputation and classification/regression accuracies.

1.4 Handling Massiveness

In Chapter 5, we further study the representation learning for extremely massive spatio-temporal data. We consider the user identification problem as an example. In this problem, we study identifying users across massive heterogeneous mobility data sources. Formally, given two large-scale mobility datasets which are generated from different data sources (e.g., geo-tagged check-in data, navigation points), our goal is to find the trajectory pairs across different datasets which are essentially generated by the same user. We face extremely complicated scenarios in finding such trajectory pairs. For example, (a) the sampling rates in different data sources can be very different; (b) for some sparse trajectories, it is hard to infer any movement of the users (c) for the users who work/live together, their mobilities have a significant overlap and it is easy to misidentify such users. There are many other cases or combinations of those cases, that are impossible to list exhaustively.

Contributions: We present a MapReduce-based framework called *Automatic User Identification*(AUI) which is easy to deploy and can scale to very large data sets. AUI is based on a novel similarity measure called *signal based similarity*(SIG), which measures the similarity of users' trajectories, typically with very different sampling rates and noise patterns. In signal based similarity, we represent users' trajectories as a series of co-occurrence events and calculate the similarity scores according to the

events in multiple resolutions. Such similarity measure is very robust for the massive mobility datasets. We conduct extensive experimental evaluations, which show that our framework outperforms the existing methods significantly.

1.5 Organization

The rest parts of this thesis are organized as follows: In Chapter 2, we introduce some preliminaries on deep neural networks and map-reduce frameworks, where we use such techniques in learning spatio-temporal data representation. In Chapter 3, we propose our method DeepTTE on capturing data correlations in travel time estimation problem. This chapter is based on our prior work[24]. In Chapter 4, we propose our model BRITS to solve the missing value problem while learning spatio-temporal data representations. The chapter is based on our prior work[25]. In Chapter 5, we further present the representation learning of extremely massive spatio-temporal data. We present our framework AUI[26] to efficiently solve the user identification problem in very massive datasets. This chapter is based on our prior work[26]. Finally, we conclude this thesis and present several future directions in Chapter 6.

, 2 à Preliminary

2.1 Deep Neural Networks

Deep neural networks are a family of machine learning algorithms based on the representation learning. It has been successfully applied in various of applications such as the computer vision, natural language processing and speech recognition, and achieves many break-through results[27,28].

A regular deep neural network consists of an input layer, an output layer, and multiple hidden layers. Each hidden layer can be regarded as a non-linear transformation of previous outputs. The output layer then predicts the classification/regression labels based on the obtained non-linear features. The stacked non-linear layers greatly enlarge the capacity of deep neural networks and make it possible to automatically discover high-order hierarchical features from raw data. To improve the efficiency and effectiveness of regular deep neural networks, multiple variations of deep architectures are further proposed. In this section, we introduce two commonly used deep architectures called convolutional neural networks and recurrent neural networks.

2.1.1 Convolutional Neural Networks 2D

The two-dimensional convolutional neural networks (2D-CNN) make explicit assumptions that the inputs are images. A typical 2D-CNN is comprised of multiple convolutional layers which take advantages of 2D input structures. Specifically, each convolutional layer contains several kernel filters. Each kernel filter is connected to a small region of previous layer's output and is replicated across the entire image. The replicated filters share the same weight parameters. Comparing with the regular deep neural networks, the convolutional neural network dramatically reduces the number of parameters and thus leads to a much more efficient learning procedure. See Fig. 2.1 for an illustration. The motivation of such design is based on the spatially local cor-

relation property of image data, i.e., each pixel is only correlated with the pixels in a local receptive field. Formally, for an input image x, we denote the weight matrix of a kernel filter as W and the bias as b. Then, we have that the output h of such filter is

$$h_{ij} = \text{ReLU}((W * x)_{ij} + b)$$

where $\text{ReLU}(x) = \max(0, x)$ and $*$ is the convolutional operator.

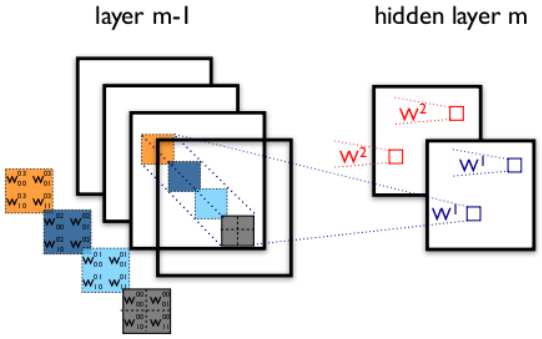

Figure 2.1 Example of 2D-CNN (figure from [29])

2.1.2 Convolutional Neural Networks 1D

Similar to the 2D-CNN, the one-dimensional convolutional neural networks (1D-CNN) use kernel filters to capture the local correlations. However, the input here is a one-dimensional sequential data, e.g., the sentences or audio signals. As an example, for the natural language, each kernel filter in 1D-CNN is applied on a small window of words, as shown in Fig. 2.2. In such case, 1D-CNN captures the correlations between adjacent words.

2.1.3 Recurrent Neural Networks

Recurrent neural networks (RNN) are also widely used to learn the temporal dependencies in the sequential data. Comparing with the feed-forward architectures, it has a “memory” cell which summarizes the information in the previous time steps. At each step, the recurrent neural network updates its memory according to the historical

Figure 2.2 Example of 1D-CNN (gure from [30])

memory and the current input recursively. Formally, we use h_t to denote the memory vector in the t-th step and x_t to denote the input in this step. Then we have that

$$h_{t+1} = \text{sigmoid}(W_x \ x_t + W_h \ h_t + b)$$

where W_x, W_h and b are parameters to be learned. In practice, vanilla recurrent neural networks often face the gradient vanishing/exploding problem [27] which lead to training di culties. To solve such issue, several improved architectures such as the Long Short Term Memory (LSTM), Gated Recurrent Unit (GRU) are proposed [31, 32].

2.2 Map-Reduce Framework

Map-Reduce framework is a programming model proposed by Google for processing large-scale datasets [33]. Such framework provides a exible way to implement the distributed algorithms and can be easily deployed on a cluster. A map-reduce framework contains two stages: the map stage and the reduce stage. In the map stage, users perform the ltering or other pre-processing operations, and emit a key-value pair $\langle k, v \rangle$. During the reduce stage, the values with the same key will be partitioned into the same reducer and the users implement the summarization operations of these values on the reducer.

To make it more concrete, we illustrate how we count the frequencies of each words in a large corpus with Map-Reduce framework. In the map stage, for each word w, we simply emit the key-value pair $\langle w, 1 \rangle$. In the reduce stage, each reducer receives

work for *Travel Time Estimation*, called *DeepTTE*. The primary contributions of this chapter can be summarized as follows:

- We propose a spatio-temporal component to learn the spatial and temporal dependencies from the raw GPS sequence. In detail, the spatio-temporal component consists of two parts: a) *geo-based convolutional layer* that transforms the raw GPS sequence to a series of feature maps, capable of capturing the local spatial correlations (like various driving situations in Fig. 3.1) from consecutive GPS points implicitly; b) *recurrent neural nets* (LSTMs) that learn the temporal dependencies of the obtained feature maps and embeddings from external factors. The spatio-temporal component directly learns effective feature representations of the GPS data.
- We propose a multi-task learning component that learns to estimate the travel time for each local path and the entire path simultaneously by a multi-task loss function, capable of balancing the trade-off between the individual and collective estimations. For estimating the entire path accurately, we design a multi-factor attention mechanism to learn the weights for different local paths based on their hidden representations and the external factors.
- We present an attribute component that integrates external factors, including the weather condition, day of the week, distance of the path, and the driver habit. The learned latent representations are fed into several parts of the model to enhance the importance of these external factors.
- We conduct extensive experiments on two real-world large scale data sets which consists of GPS points generated by taxis in Chengdu and Beijing. The percentage errors on these two datasets are 9.91% and 10.92% respectively, which significantly outperforms the existing methods.

3.2 Preliminary

In this section, we first present several preliminaries and define our problem formally.

Definition 1 (Historical Trajectory) We define a *historical trajectory* T as a sequence of consecutive historical GPS points, i.e., $T = \{p_1, ..., p_{|T|}\}$[①]. Each GPS point p_i contains: the latitude ($p_i.lat$), longitude ($p_i.lng$) and the timestamp ($p_i.ts$). Furthermore, for each trajectory we record its external factors such as the starting time (timeID), the day of the week (weekID), the weather condition (weather) and corresponding driver (driverID).

Definition 2 (Objective) During the training phase, we learn how to estimate the travel time of the given path and the corresponding external factors, based on the spatio-temporal patterns extracted from the historical trajectories as we defined in Definition. 1. During the test phase, given the path P, our goal is to estimate the travel time from the source to the destination through P with the corresponding external factors. We assume that the travel path P is specified by the user or generated by the route planing apps.

During the test phase, to make the testing data consistent with the training data, we convert a path P to a sequence of location points with equal distance gaps. Each location is represented as a pair of longitude and latitude.

Remark: In our experiment, to generate the test data, we remove the timestamps in the historical trajectories and resample each trajectory into a GPS location sequence with equal distance gaps. In this chapter, we do not consider how to optimize the path P.

3.3 Model Architecture

In this section, we describe the architecture of our proposed DeepTTE, as shown in Fig. 3.2. DeepTTE is comprised of three components: an attribute component, a spatio-temporal learning component, and a multi-task learning component. The attribute component is used to processes the external factors (e.g. weather) and the basic information of the given path (e.g. start time). Its output is fed to the other components

① The GPS devices usually generate one record for every fixed length time gap. This can cause our model to learn a trival pattern (e.g., simply counts the number of GPS records). To avoid such case, we resample each historical trajectory such that the distance gap between two consecutive points are around 200 to 400 meters.

as a part of their inputs. The spatio-temporal learning component is the main building component that learns the spatial correlations and temporal dependencies from the raw GPS location sequences. Finally, the multi-task learning component estimates the travel time of the given path based on the previous two components, capable of balancing the tradeo between individual estimation and collective estimation.

Figure 3.2 DeepTTE Architecture. Dis: distance; concat: concatenate.

3.3.1 Attribute Component

As we mentioned, the travel time of a path is affected by many complex factors, such as the start time, the day of week, the weather condition and also the driving habits. We design a simple yet effective component to incorporate such factors into our model, where we call it the attribute component. We can also easily incorporate more factors into the model.

As an example in Fig. 3.2, we incorporate the attributes of the weather condition (rainy/sunny/windy etc.) , the driver ID, the time information (day of the week and

Thus, we obtain the final feature map of shape $\mathbb{R}^{(c+1)\times(|T|-k+1)}$ by our Geo-Conv layer. We denote this feature map as loc^f.

3.3.2.2 Recurrent Layer

The feature map loc^f captures the spatial dependencies of all the local paths. To further capture the temporal dependencies among these local paths, we introduce a recurrent layers in our model. The recurrent neural network (RNN) is an artificial neural network which is widely used for capturing the temporal dependency in sequential learning, such as natural language processing and speech recognition[37–39]. The recurrent neural network is able to "memorize" the history in the processed sequence. When processing the current time step in the sequence, it updates its memory according to the current input and the previous memories. The output of the recurrent neural network is the memory sequence at all the time steps in the sequence.

In our model, the input sequence of the recurrent neural network is the feature map loc^f outputted by Geo-Conv layer. The feature map loc^f can be regarded as a sequence of spatial features with length $|T| - k + 1$. Moreover, we find that incorporating the attributes information is helpful to further enhance the capacity of the recurrent layers (recall that we have obtained the attributes representation vector $attr$ in the attribute component). Thus, in simple terms, the updating rule of our recurrent layer can be expressed as

$$h_i = \phi_{rnn}(W_x \cdot loc_i^f + W_h \cdot h_{i-1} + W_a \cdot attr) \tag{3-3}$$

where h_i is the memory after we processed the i-th local path and ϕ_{RNN} is a non-linear activation function. In practice, the Eq.(3-3) usually fails in processing the long sequence due to vanishing gradient and exploding gradient problems[31]. To overcome such issue, we use two stacked Long Short-Term Memory (LSTM) layers instead. LSTM was first developed by Hochreiter et al.[31]. It introduces an input gate and a forget gate to control the input information flow. Such gate mechanism enables LSTM to forget some unimportant information and effectively alleviate the gradient vanishing/exploding problem. Furthermore, it has been shown that a stacked LSTM is more

efficient to increase the model capacity compared with a single layer LSTM[40].

Now, by utilizing the Geo-Conv layer and the recurrent layer, we obtain the sequence $\{h_1; h_2; \ldots; h_{|T|-k+1}\}$ which represents the spatio-temporal features of the raw GPS sequence.

3.3.3 Multi-task Learning Component

We finally introduce a multi-task learning component which combines the previous components and estimates the travel time of input path. Prior work in estimating the travel time can be divided into two types, the *individual estimation* and the *collective estimation*. The individual estimation estimates the travel time of each local path and sum them up. The collective estimation instead estimates the travel time of the entire path directly. However, as we mentioned in the challenge part, the local errors may accumulate if we adopt the individual estimation since such method does not consider the spatio-temporal dependencies among the local paths. In the mean time, if we use the collective estimation, we usually face the data sparsity problem since only a few trajectories traveled through the entire path or the longer sub-paths.

In our model, we combine these two methods with our multi-task learning component. The multi-task learning component estimates the travel time of both entire path and each local path simultaneously. As during the training phase, the travel times of the local paths are available. The local path estimation utilizes this information for stabilizing the training procedure and enhancing the accuracy.

3.3.3.1 Estimate the local paths

Recall that we use the spatio-temporal component to obtain a sequence $\{h_1, h_2, \ldots; h_{|T|-k+1}\}$. Here each h_i corresponds to the spatio-temporal feature of local path $p_i \rightarrow p_{i+1} \rightarrow \ldots \rightarrow p_{i+k-1}$. We simply use two stacked fully-connected layers with size 64 and 1 to map each h_i into a scalar r_i. Here r_i represents the travel time of the i-th local path. The stacked fully-connected layers for different h_i share the same weight parameters, as shown in Fig. 3.2.

3.3.3.2 Estimate the entire path

The estimation of the entire path is relatively more complex. Recall that the length of spatio-temporal feature sequence $\{h_i\}$ is also variable. To estimate the travel time of the entire path directly, we first need to transform the feature sequence into a fixed length vector. A simple method to achieve this is to use *mean pooling*, i.e., $h_{mean} = \frac{1}{|T|-k+1} \sum_{i=1}^{|T|-k+1} h_i$. Such mechanism is simple yet effective. It demonstrates a good performance for most paths as shown in the experiment part. However, the mean pooling method treats all the spatio-temporal features h_i equally. In fact, the uncertainty of accurately estimating the travel time is usually caused by several critical local paths. For example, if the path contains multiple road intersections, traffic lights, or road segments which can be extremely congested, we should pay more attention on such parts since they are more difficult to estimate. Inspired by this, in our model, we adopt the *attention mechanism* instead of the mean pooling. The attention mechanism is essentially the weighted sum of sequence $\{h_i\}$ where the weights are parameters learned by the model. Formally, we have that

$$h_{att} = \sum_{i=1}^{|T|-k+1} {}_i \cdot h_i \tag{3-4}$$

where ${}_i$ is the weight for the i-th local path, and the summation of all ${}_i$ equals 1. To learn the weight parameter, we consider the spatial information of the local paths, as well as the external factors such as the start time, the day of week and the weather condition. For example, if the starting time of the path is a weekend evening, we would be careful with the road segments near entertainment district since they are usually very congested at that time. In our model, the vector $attr$ outputted by the attribute component captures the effect of external factors, and the feature sequence $\{h_i\}$ captures the spatio-temporal information of local paths. Thus, we devise our attention mechanism based on $attr$ and $\{h_i\}$:

$$\begin{aligned} z_i &= \langle {}_{ATT}(attr); h_i \rangle \\ {}_i &= \frac{e^{z_i}}{\sum_j e^{z_j}} \end{aligned} \tag{3-5}$$

where $_{ATT}$ is a non-linear mapping which maps $attr$ to a vector with the same length as h_i. Substituting Eq. (3-5) into Eq. (3-4), we obtain the vector h_{att}.

Finally, we pass h_{att} to several fully-connected layers with equal size (we use the size of 64 in our experiment). The fully-connected layers are connected with *residual connections* which is a technique to train a very deep neural network[41]. The residual connection adds "shortcuts" between different layers (dashed line in Fig. 3.2). Thus, previous information flow can skip one or more non-linear layers through the shortcut and the skipped layers just need to learn the "residual" of the non-linear mapping. In our model, we use $_{f_i}$ to denote the i-th fully-connected layer. For the first fully-connected layer, the output of this layer is $_{f_1}(h_{attr})$. For the rest of the fully-connected layers, suppose the output of the i-th layer is x. Then, the output of the $(i+1)$-th layer can be represented as $_{f_{i+1}}(x) \oplus x$ where $\oplus$ is the element-wise add operation. It has been shown that training the neural networks with residual connections is easier and more robust[41]. At last, we use a single neuron to obtain the estimation of the entire path, which we denote as $_{en}$.

3.3.4 Model Training

We finally present the training procedure of our model. Our model is trained end to end. During the training phase, we use the *mean absolute percentage error (MAPE)* as our objective function. Since MAPE is the relative error, we can enforce our model to provide accurate results for both the short paths and the long paths. However, we use multiple criterions to evaluate our model, including the rooted mean squared error (RMSE) and the mean absolute error (MAE).

Recall that during the training phase, we estimate the travel time of all the local paths and the entire path simultaneously. For the local paths, we define the corresponding loss as the average of losses in each local path, i.e.,

$$L_{local} = \frac{1}{|T| - k + 1} \sum_{i=1}^{|T|-k+1} \left| \frac{r_i - (p_{i+k-1}.ts - p_i.ts)}{p_{i+k-1}.ts - p_i.ts + } \right| \tag{3-6}$$

where is a small constant to prevent the exploded loss value when the denominator is

close to 0. For the entire path, we define the corresponding loss as

$$L_{en} = |r_{en} - (p_{|T|}.ts - p_1.ts)| = (p_{|T|}.ts - p_1.ts + \quad): \tag{3-7}$$

Our model is trained to minimize the weighted combination of two loss terms

$$\cdot L_{local} + (1 - \quad) \cdot L_{en} \tag{3-8}$$

where is the combination coefficient that linearly balances the trade-off between L_{local} and L_{en}. By default, during the test phase, we use the travel time estimation of the entire path r_{en} as our final estimation.

3.4 Experiment

In this section, we report our experimental results on two large scale real-world datasets. We first compare our model with several baseline methods, including the state-of-the-art collective estimation method TEMP[42]. We then present the effectiveness of our model by a set of controlled experiments[①].

3.4.1 Experiment Setting

3.4.1.1 Data Description

We evaluate our model on two large scale datasets:

- Chengdu Dataset. Chengdu Dataset consists of 733,557 trajectories (1.4 billion GPS records) of 14864 taxis in August 2014 in Chengdu, China. The shortest trajectory contains only 11 GPS records (2km) and the longest trajectory contains 128 GPS records (41km).
- Beijing Dataset. Beijing Dataset consists of 1,349,023 trajectories (2.45 billion GPS records) of 20442 taxis in April 2015 in Beijing, China. The shortest trajectory contains 15 GPS records (3km) and the longest trajectory contains 128 GPS records (50km).

① The code and the sample data can be downloaded at https://github.com/UrbComp/DeepTTE.

The trajectories in both datasets are associated with the corresponding weekID, timeID and driverID. For Beijing Dataset, we further collected the corresponding weather conditions (16 types including sunny, rainy, cloudy etc) as well as the road ID of each GPS point.

3.4.1.2 Parameter Setting

The parameters we used in our experiment are described as follows:

- In the attribute component, we embed weekID to R^3, timeID to R^{16}, driverID to R^8 and the corresponding weather type to R^3.
- In the geo-conv layer, we fix the number of filters $c = 32$ and we use ELU function[43] as the activation $_{cnn}$ in Eq.(3-2). The ELU is defined as $ELU(x) = e^x - 1$ for $x \leq 0$ and $ELU(x) = x$ for $x > 0$. For the kernel size k, we evaluate our model under different values of k.
- In the recurrent layer, we use tanh as the activation $_{rnn}$ in Equ. (3-3). We fix the size of the hidden vector h_i as 128.
- In the multi-task learning component, we first use a fully-connected layer with tanh activation as $_{att}$ in Eq. (3-5). We use $ReLU(x) = \max(0, x)$[44] as the activation of residual fully-connected layers. We fix the number of the residual fully-connected layers as 4 and the size of each layer as 128. Furthermore, we evaluate our model for different combination coefficient in Eq. (3-8) from 0.0 to 0.99.

For each dataset, we use the trajectories generated in the last 7 days as the test set and the rest of trajectories as the training set. We adopt Adam optimization algorithm[45] to train the parameters. The learning rate of Adam is 0.001 and the batch size during training is 400. We train the model for 100 epochs and select the best models by 5-fold cross-validation.

Our model is implemented with PyTorch 2.0, a widely used Deep Learning Python library.

3.4.2 Performance Comparison

To demonstrate the strength of our model. We first compare our model with several baseline methods, including:

- AVG: We simply calculate the average speed in the city during a specific time interval (e.g. 13:00-14:00 PM on Monday). We estimate the travel time of given trajectory based on its starting time and the historical average speed.
- TEMP: TEMP[42] is the state-of-the-art collective estimation method. It estimates the travel time of the given path based on the "neighbor" trajectories, i.e., the trajectories which have the closed starting and destination as the query path. This work outperforms the most outstanding individual TTE[22] as well as Bing/Baidu Map API in their experiment. However, there are about 10% paths that the original TEMP method can not estimate due to the lack of neighbor trajectories. For those paths, we enlarge the neighborhood dynamically until we can find enough neighbor trajectories (10 trajectories in our experiment). We refer to such implementation as *dynamic TEMP* (D-TEMP).
- GBDT: Gradient Boosting Decision Tree (GBDT) is a powerful ensemble method[46] and widely used in practice. In our problem, the input of GBDT is as same as the input of DeepTTE, including all the inputs in the attribute component and the raw GPS sequence. Note that since the length of GPS sequence is variable, GBDT can not handle such sequence directly. Here, we uniformly sample (with replacement) each GPS sequence to a fixed length of 128.
- MlpTTE: We use a 5-layer perceptron with PReLU activation to estimate the travel time. The input of MlpTTE is almost the same as GBDT, except that the categorical values are properly embedded to real vectors. The size of hidden layers in MlpTTE is fixed as 128.
- RnnTTE: RnnTTE is also a simplified model of DeepTTE. We use a vanilla RNN and mean pooling to process the raw GPS sequence into a 128-dimensional feature vector. We concatenate such feature vector and the output of the attribute component. The concatenation then is passed to the residual fully-connected layers to obtain the estimation.

Table 3.1 Performance Comparison on Chengdu Dataset

	MAPE (%)	RMSE (sec)	MAE (sec)
AVG	28.1	533.57	403.71
D-TEMP	22.82	441.50	323.37
GBDT	19.32 ± 0.04	357.09 ± 2.44	266.15 ± 2.24
MlpTTE	16.90 ± 0.06	379.39 ± 1.94	265.47 ± 1.53
RnnTTE	15.65 ± 0.06	358.74 ± 2.02	246.52 ± 1.65
DeepTTE	11.89 ± 0.04	282.55 ± 1.32	186.93 ± 1.01

Table 3.2 Performance Comparison on Beijing Dataset

	MAPE (%)	RMSE (sec)	MAE (sec)
AVG	24.78	703.17	501.23
D-TEMP	19.63	606.76	402.50
GBDT	19.98 ± 0.02	512.96 ± 3.96	393.98 ± 2.99
MlpTTE	23.73 ± 0.14	701.61 ± 1.82	489.54 ± 1.61
RnnTTE	13.73 ± 0.05	408.33 ± 1.83	275.07 ± 1.48
DeepTTE	10.92 ± 0.06	329.65 ± 2.17	218.29 ± 1.63

For our model DeepTTE, we fix the kernel size as 3 and combination coefficient as 0.3. We repeat each experiment for 5 times. We report the mean and the standard variation of the different runs. The experiment result is shown in Table 3.1 and 3.2.

As we can see, simply using the average speed leads to a very inaccurate result. In contrast, the collective method D-TEMP and the ensemble method GBDT are much better than AVG. MlpTTE shows a good performance on Chengdu Dataset. However, it does not consider the spatio-temporal dependencies in the data. For the dataset which contains more complex traffic conditions (e.g., Beijing Dataset), we find that MlpTTE tends to overfit the training data and thus leads to a bad result. RnnTTE use the recurrent layers to capture the temporal dependencies in the data. It achieves 15.65% and 13.73% on two datasets which are much better than above mentioned methods. Our model DeepTTE further significantly outperforms RnnTTE. The error rates in two datasets are only 11.89% and 10.92%. We use Paired T-Test[46] to further test the significance of our model. The p-value for all the test pairs are less than 10^{-8} which

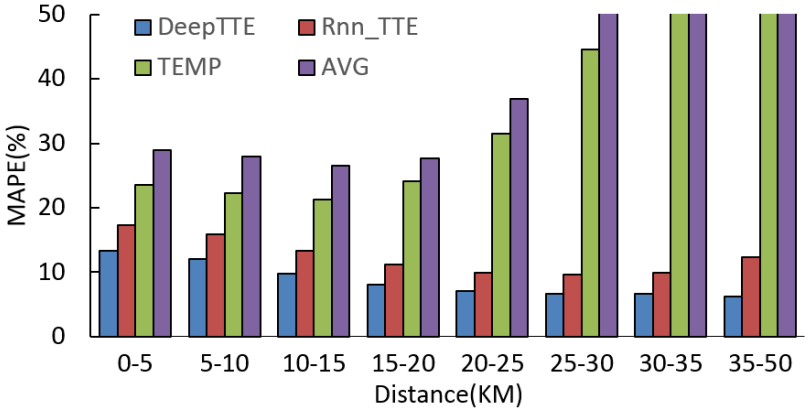

Figure 3.4 Error rates for trajectories with different lengths.

demonstrates that our model is significantly better than baselines. In Fig. 3.4, we compare the model performances for the trajectories with different lengths. When the path length goes larger, AVG and TEMP methods face a serious individual-collective trade-off problem. As a consequence, the error rates increase sharply. RnnTTE shows a better result for the trajectories less than 35 km. However, it fails to handle the long paths (with length greater than 35 km). In contrast, our model demonstrates remarkable results for longer paths. Finally, recall that there are about 10% paths that the original TEMP can not estimate. For the rest of paths in two datasets, the average error rate is 19.96% for the original TEMP but only 12.21% for our model.

3.4.3 Effect of Attribute Component

We show the effectiveness of different attributes, including the weatherID, driverID and weekID. We devise a set of controlled experiments on Beijing Dataset. For each experiment, we eliminate exactly one attribute.We find that weatherID and weekID affect the estimation significantly. Eliminating such two attributes causes an error growth of 1.09% and 0.77% respectively. This also conforms to our intuitive sense, i.e., we usually spend much more time for traveling the same path under bad weather conditions. Eliminating the driver information causes an error increment of 0.30% which seems not significant. However, we stress the trajectories in our dataset are generated by taxi drivers. Most of the taxi drivers are very experienced and have

similar driving habits. The driver information might be more useful for estimating the travel time of normal people. We leave it as an intriguing direction for future work.

3.4.4 Effect of Geo-Conv

We first evaluate our model under the absence of Geo-Conv layer. To achieve this, we eliminate the geo-conv layer and directly pass the sequence $\{p_i.lat \circ p_i.lng\}$ to the following LSTM layers. The MAPE in Chengdu/Beijing Dataset under this setting is 13.14% and 12.68% (comparing with 11.89% and 10.92%).

We further test our model if we increase the kernel size k in Equ. (3-2) from 3 to 4, 5 and 6. We find that the performance of our model in Beijing Dataset decreases when the kernel size goes larger. The percentage errors for $k = 4, 5, 6$ are 11.18%, 11.31% and 11.38%. The reason might be that when the kernel size $k = 3$, a local path consists of two consecutive road segments which can exactly represent the turnings or road intersections (see Fig. 3.1). Thus, it is more easily for our model to capture such spatial features in the local paths.

3.4.5 Effect of Multi-task Learning

To show the effectiveness of the multi-task learning component, we first evaluate our model under different combination coefficient α from 0.0 to 0.99. The result is shown in Fig 3.5. We find that our model is pretty robust for a wide range of α. The high error rates are only found in two ends, i.e., when $\alpha = 0$ or when α goes to 1. We then replace the attention mechanism in the multi-task learning component with the meaning pooling method. The error rates in Chengdu/Beijing Dataset increase to 12.09% and 11.11% respectively.

3.4.6 Incorporate Road Information

Recall that in the introduction part, we stress that our model does not rely on any map-matching algorithm but directly handle the raw GPS sequence. Nevertheless, it is very easy to incorporate the road information into our model when it is available. We collect the corresponding road ID of each GPS point in Beijing Dataset (there are

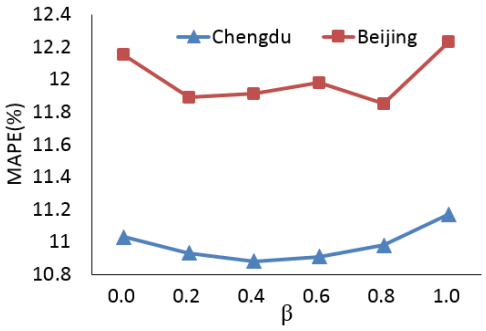

Figure 3.5 Error rates for different β.

189727 road segments in total). We embed each road ID to a 32-dimensional vector. For each GPS point, we use $p_i.rid$ to denote the embedded road ID of p_i. To utilize such road information, we can simply add $p_i.rid$ into Eq. (3-1), i.e., we have that

$$loc_i = \tanh(W_{loc} \cdot [p_i.lat \circ p_i.lng \circ p_i.rid])$$

in Geo-Conv layer. The MAPE of DeepTTE after incorporating the road information decreases from 10.92% to 10.59%.

3.4.7 Predicting Time

Despite the training time of DeepTTE is longer, the prediction is very efficient. During the test phase, estimating every 1000 paths takes DeepTTE 0.038s, RnnTTE 0.031s, MlpTTE 0.029s on GPU (including I/O time), GBDT 0.017s, and TEMP 0.47s. TEMP is slower for searching neighbors.

3.5 Related work

There is a large body of literature on the estimation of travel time; we only mention a few closely related ones.

3.5.1 Road Segment-Based Travel Time Estimation

Estimating travel time has been studied extensively [21,47,48]. However, these works estimated the travel time of individual road segment without considering the correlations between the roads. Yang et al. [19] used a spatial-temporal Hidden Markov Model

to formalize the relationships among the adjacent roads. Wang et al.[49] improved this work through an ensemble model based on two observed useful correlations in the traffic condition time series. Wang et al.[20] proposed an error-feedback recurrent Convolutional neural network called eRCNN for estimating the traffic speed on each individual road. These studies considered the correlation between different roads. However, they focused on accurately estimating the travel time or speed of individual road segment. As we mentioned in the Section 3.1, the travel time of a path is affected by various factors, such as the number of road intersections and the traffic lights in the path. Simply summing up the travel time of the road segments in the path does not lead to an accurate result[94].

3.5.2 Path-Based Travel Time Estimation

Rahmani et al.[50] estimated the travel time of a path based on the historical data of the path. However, the historical average based model may lead to a poor accuracy. Moreover, as new queried path may be not included in the historical data, it suffers from the data sparse problem. Yuan et al.[51] built a landmark graph based on the historical trajectories of taxis, where each landmark represents a single road. They estimate the travel time distribution of a path based on the landmark graph. However, as the landmarks are selected from the top-k frequently traversed road, the roads with few traveled records can not be estimated accurately. Furthermore, Wang et al.[22] estimated the travel time of the path, based on the sub-trajectories in the historical data. They used the tensor decomposition to complete the unseen sub-trajectory and such method enhance the accuracy effectively. Nevertheless, it still suffers from the data sparsity problem since there are many sub-trajectories which were visited by very few drivers. Yang et al.[52,53] used a hybrid graph to maintain the path weights and thus summarized the travel time distribution of different paths. They focused on estimating the distribution of travel time (instead of a real value), which is different from our setting.

3.5.3 Deep Learning in Spatial Temporal Data

Recently, the deep learning techniques demonstrate the strength on spatio-temporal data mining problems. Song et al.[54] built an intelligent system called Deep-Transport, for simulating the human mobility and transportation mode at a citywide level. Zhang et al.[55] proposed a deep spatio-temporal residual network for predicting the crowd flows. Dong et al.[56] studied characterizing the driving style of different drivers by a stacked recurrent neural network. To best of knowledge, no prior work studies estimating the travel time of the whole path based on the deep learning approach.

3.6 Conclusion

In this chapter, we study estimating the travel time for any given path. We propose an end-to-end framework based on deep neural networks. Our model can capture the spatial and temporal correlations in the given path, and transform the captured correlations into an effective feature representation. Our model also considers various factors which may affect the travel time such as the driver habit, the day of the week. We conduct extensive experiments on two very large scale real-world datasets. The results show that our model achieve a high estimation accuracy and outperforms the other off-the-shell methods significantly.

, 4 à Missing Value Imputation with Recurrent Dynamics

In this chapter, we study handling missing values while learning spatio-temporal data representations. The proposed methods in this chapter enable our spatio-temporal data representation learning algorithms to further handle the missing values. Given multiple (spatially) correlated time series data, how to fill in missing values and to predict their class labels? Existing imputation methods often impose strong assumptions of the underlying data generating process, such as linear dynamics in the state space. In this chapter, we propose BRITS, a novel method based on recurrent neural networks for missing value imputation in time series data. Our proposed method directly learns the missing values in a bidirectional recurrent dynamical system, without any specific assumption. The imputed values are treated as variables of RNN graph and can be effectively updated during the backpropagation. BRITS has three advantages: (a) it can handle multiple correlated missing values in time series; (b) it generalizes to time series with nonlinear dynamics underlying; (c) it provides a data-driven imputation procedure and applies to general settings with missing data. We evaluate our model on three real-world datasets, including an air quality dataset, a health-care data, and a localization data for human activity. Experiments show that our model outperforms the state-of-the-art methods in both imputation and classification/regression accuracies.

4.1 Introduction

Multivariate time series data are abundant in many application areas, such as financial marketing[57,58], health-care[59,60], meteorology[61,62], and traffic engineering[55,63]. Time series are widely used as signals for classification/regression in various applications of corresponding areas. However, missing values in time series are very common, due to unexpected accidents, such as equipment damage or communication error, and may significantly harm the performance of downstream applications.

Much prior work proposed to fix the missing data problem with statistics and

machine learning approaches. However most of them require fairly strong assumptions on missing values. We can fill the missing values using classical statistical time series models such as ARMA or ARIMA (e.g., Ansley et al.[2]). But these models are essentially linear (after differencing). Kreindler et al.[14] assume that the data are smoothable, i.e., there is no sudden wave in the periods of missing values, hence imputing missing values can be done by smoothing over nearby values. Matrix completion has also been used to address missing value problems[15,64]. But it typically applies to only static data and requires strong assumptions such as low-rankness. We can also predict missing values by fitting a parametric data-generating model with the observations[16,17], which assumes that data of time series follow the distribution of hypothetical models. These assumptions make corresponding imputation algorithms less general, and the performance less desirable when the assumptions do not hold.

In this chapter, we propose BRITS, a novel method for filling the missing values for multiple correlated time series. Internally, BRITS adapts recurrent neural networks (RNN)[31] for imputing missing values, without any specific assumption over the data. Much prior work uses non-linear dynamical systems for time series prediction[65–67]. In our method, we instantiate the dynamical system as a bidirectional RNN, i.e., imputing missing values with bidirectional recurrent dynamics. In particular, we make the following technical contributions:

- We design a bidirectional RNN model for imputing missing values. We directly use RNN for predicting missing values, instead of tuning weights for smoothing as in the method proposed by Che et al.[59]. Our method does not impose specific assumption, hence works more generally than previous methods.
- We regard missing values as variables of the bidirectional RNN graph, which are involved in the backpropagation process. In such case, missing values get delayed gradients in both forward and backward directions with consistency constraints, which makes the estimation of missing values more accurate.
- We simultaneously perform missing value imputation and classification/regression of applications jointly in one neural graph. This alleviates the error propagation problem from imputation to classification/regression.

Additionally, the supervision of classification/regression makes the estimation of missing values more accurate.

- We evaluate our model on three real-world datasets, including an air quality dataset, a health-care dataset and a localization dataset of human activities. Experimental results show that our model outperforms the state-of-the-art models for both imputation and classification/regression accuracies.

4.2 Preliminary

We first present the problem formulation and some necessary preliminaries.

Definition 4 (Multivariate Time Series) We denote a multivariate time series $X = \{x_1, x_2, \ldots, x_T\}$ as a sequence of T observations. The t-th observation $x_t \in \mathbb{R}^D$ consists of D features $\{x_t^1, x_t^2, \ldots, x_t^D\}$, and was observed at timestamp s_t (the time gap between different timestamps may be not same). In reality, due to unexpected accidents, such as equipment damage or communication error, x_t may have the missing values (e.g., in Fig. 4.1, x_1^3 in x_1 is missing). To represent the missing values in x_t, we introduce a *masking vector* m_t where,

$$m_t^d = \begin{cases} 0 & \textit{if } x_t^d \textit{ is not observed} \\ 1 & \textit{otherwise} \end{cases}.$$

In many cases, some features can be missing for consecutive timestamps (e.g., blue blocks in Fig. 4.1). We define δ_t^d to be the time gap from the last observation to the current timestamp s_t, i.e.,

$$\delta_t^d = \begin{cases} s_t - s_{t-1} + \delta_{t-1}^d & \textit{if } t > 1, m_{t-1}^d = 0 \\ s_t - s_{t-1} & \textit{if } t > 1, m_{t-1}^d = 1 \\ 0 & \textit{if } t = 1 \end{cases}.$$

See Fig. 4.1 for an illustration.

In this chapter, we study a general setting for time series classification/regression problems with missing values. We use y to denote the label of corresponding classifi-

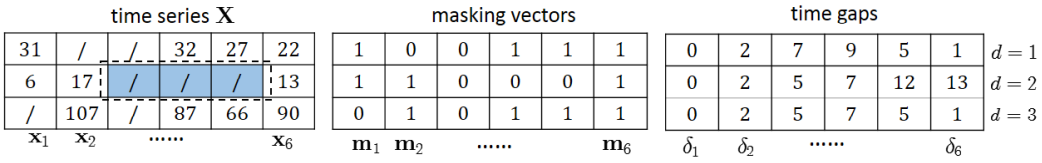

Figure 4.1 An example of multivariate time series with missing values. $\mathbf{x}_1$ to $\mathbf{x}_6$ are observed at $s_{1:::6} = 0; 2; 7; 9; 14; 15$ respectively. Considering the 2nd feature x^2, the last observation of the 2nd feature took place at $s_2 = 2$, and we have that $\delta^2_6 = s_6 - s_2 = 13$.

cation/regression task. In general, y can be either a scalar or a vector. Our goal is to predict y based on the given time series $\mathbf{X}$. In the meantime, we impute the missing values in $\mathbf{X}$ as accurate as possible. In another word, we aim to design effective multi-task learning algorithm for both classification/regression and imputation.

4.3 BRTIS

In this section, we describe the BRITS. Differing from the prior work which uses RNN to impute missing values in a smooth fashion[59], we learn the missing values directly in a recurrent dynamical system[68,69] based on the observed data. The missing values are thus imputed according to the recurrent dynamics, which significantly boosts both the imputation accuracy and the final classification/regression accuracy. We start the description with the simplest case: the variables observed at the same time are mutually uncorrelated. For such case, we propose algorithms for imputation with *unidirectional recurrent dynamics* and *bidirectional recurrent dynamics*, respectively. We further propose an algorithm for correlated multivariate time series in Section 4.3.3.

4.3.1 Unidirectional Uncorrelated Recurrent Imputation

For the simplest case, we assume that for the t-th step, x^i_t and x^j_t are uncorrelated if $i \neq j$ (but x^i_t may be correlated with some $x^i_{t' \neq t}$). We first propose an imputation algorithm by unidirectional recurrent dynamics, denoted as RITS-I.

In a *unidirectional recurrent dynamical system*, each value in the time series can be derived by its predecessors with a fixed *arbitrary* function[65–67]. Thus, we iteratively impute all the variables in the time series according to the recurrent dynamics. For

Figure 4.2 Imputation with unidirectional dynamics.

the t-th step, if x_t is actually observed, we use it to validate our imputation and pass x_t to the next recurrent steps. Otherwise, since the future observations are correlated with the current value, we replace x_t with the obtained imputation, and validate it by the future observations. To be more concrete, let us consider an example.

Example 1 Suppose we are given a time series $X = \{x_1, x_2, \ldots, x_{10}\}$ where x_5, x_6 and x_7 are missing. According to the recurrent dynamics, at each time step we can obtain an estimation $\hat{x}_t$ based on the previous $t - 1$ steps. In the first 4 steps, the estimation error can be obtained immediately by calculating the estimation loss function $L_e(\hat{x}_t, x_t)$ for $t = 1, \ldots, 4$. However, when $t = 5, 6, 7$, we cannot calculate the error immediately since the true values are missing. Nevertheless, note that at the 8-th step, $\hat{x}_8$ depends on $\hat{x}_5$ to $\hat{x}_7$. We thus obtain a "delayed" error for $\hat{x}_{t=5,6,7}$ at the 8-th step.

4.3.1.1 Algorithm

We introduce a recurrent component and a regression component for imputation. The recurrent component is achieved by a recurrent neural network and the regression component is achieved by a fully-connected network. A standard recurrent network [70]

rection, each value in time series can be also derived from the backward direction by another fixed arbitrary function.

To illustrate the intuition of BRITS-I algorithm, again, we consider Example 1. Consider the backward direction of the time series. In bidirectional recurrent dynamics, the estimation $\hat{x}_4$ reversely depends on $\hat{x}_8$ to $\hat{x}_7$. Thus, the error in the 5-th step is back-propagated from not only the 8-th step in the forward direction (which is far from the current position), but also the 4-th step in the backward direction (which is closer). Formally, the BRITS-I algorithm performs the RITS-I as shown in Eq. (4-1) to Eq. (4-5) in forward and backward directions, respectively. In the forward direction, we obtain the estimation sequence $\{\hat{x}_1, \hat{x}_2, \dots, \hat{x}_T\}$ and the loss sequence $\{\ell_1, \ell_2, \dots, \ell_T\}$. Similarly, in the backward direction, we obtain another estimation sequence $\{\hat{x}'_1, \hat{x}'_2, \dots, \hat{x}'_T\}$ and another loss sequence $\{\ell'_1, \ell'_2, \dots, \ell'_T\}$. We enforce the prediction in each step to be consistent in both directions by introducing the *"consistency loss"*

$$\ell_t^{cons} = \text{Discrepancy}(\hat{x}_t, \hat{x}'_t) \tag{4-6}$$

where we use the *mean squared error* as the discrepancy in our experiment. The final estimation loss is obtained by accumulating the forward loss ℓ_t, the backward loss ℓ'_t and the consistency loss ℓ_t^{cons}. The final estimation in the t-th step is the mean of $\hat{x}_t$ and $\hat{x}_t'$.

4.3.3 Correlated Recurrent Imputation

The previously proposed algorithms RITS-I and BRITS-I assume that features observed at the same time are mutually uncorrelated. This may be not true in some applications. For example, in the air quality data [72], each feature represents one measurement in a monitoring station. Obviously, the observed measurements are spatially correlated. In general, the measurement in one monitoring station is close to those values observed in its neighboring stations. In this case, we can estimate a missing measurement according to both its historical data, and the measurements in its neighbors.

In this section, we propose an algorithm, which utilizes the feature correlations in

the unidirectional recurrent dynamical system. We refer to such algorithm as RITS. The feature correlated algorithm for bidirectional case (BRITS) can be derived in the same way. Note that in Section 4.3.1, the estimation $\hat{x}_t$ is only correlated with the historical observations, but irrelevant with the the current observation. We refer to $\hat{x}_t$ as a "history-based estimation". In this section, we derive another "feature-based estimation" for each x_t^d, based on the other features at time t. Specifically, at the t-th step, we first obtain the complement observation c_t by Eq. (4-1) and Eq. (4-2). Then, we define our feature-based estimation $\hat{z}_t$ as where

$$\hat{z}_t = W_z x_t^c + b_z, \tag{4-7}$$

W_z, b_z are corresponding parameters. We restrict the diagonal of parameter matrix W_z to be all zeros. Thus, the d-th element in $\hat{z}_t$ is exactly the estimation of x_t^d, based on the other features. We further combine the historical-based estimation $\hat{x}_t$ and the feature-based estimation $\hat{z}_t$. We denote the combined vector as $\hat{c}_t$, and we have that

$$\gamma_t = \exp(W_\gamma [\delta_t \circ m_t] + b_\gamma) \tag{4-8}$$

$$\hat{c}_t = \gamma_t \odot \hat{z}_t + (1 - \gamma_t) \odot \hat{x}_t. \tag{4-9}$$

Here we use $\gamma_t \in [0, 1]^D$ as the weight of combining the history-based estimation $\hat{x}_t$ and the feature-based estimation $\hat{z}_t$. Note that $\hat{z}_t$ is derived from x_t^c by Eq. (4-7). The elements of x_t^c can be history-based estimations or truly observed values, depending on the presence of the observations. Thus, we learn the weight γ_t by considering both the temporal decay δ_t and the masking vector m_t as shown in Eq. (4-8). The rest parts are similar as the feature uncorrelated case. We first replace missing values in x_t with the corresponding values in $\hat{c}_t$. The obtained vector is then fed to the next recurrent step to predict memory h_t:

$$c_t^c = m_t \odot x_t + (1 - m_t) \odot \hat{c}_t \tag{4-10}$$

$$h_t = \tanh(W_h [h_{t-1} \odot \delta_t] + U_h [c_t^c \circ m_t] + b_h). \tag{4-11}$$

However, the estimation loss is slightly different with the feature uncorrelated case. We find that simply using $\ell_t = \mathcal{L}_e(x_t, \hat{c}_t)$ leads to a very slow convergence speed. Instead,

we accumulate all the estimation errors of $\hat{x}_t$, $\hat{z}_t$ and $\hat{c}_t$:

$$\grave{}_t = \mathcal{L}_e(x_t; \hat{x}_t) + \mathcal{L}_e(x_t; \hat{z}_t) + \mathcal{L}_e(x_t; \hat{c}_t):$$

4.4 Experiment

Our proposed methods are applicable to a wide variety of applications. We evaluate our methods on three different real-world datasets. The download links of the datasets, as well as the implementation codes can be found in the GitHub① page.

4.4.1 Dataset Description

4.4.1.1 Air Quality Data

We evaluate our models on the air quality dataset, which consists of PM2.5 measurements from 36 monitoring stations in Beijing. The measurements are hourly collected from 2014/05/01 to 2015/04/30. Overall, there are 13% values are missing. For this dataset, we do pure imputation task. We use the exactly same training/testing setting as in prior work[72], i.e., we use the 3rd, 6th, 9th and 12th months as the test data and the other months as the training data. To train our model, we randomly select every 36 consecutive steps as one time series.

4.4.1.2 Health-care Data

We evaluate our models on health-care data *PhysioNet Challenge 2012*[73], which consists of 4000 multivariate clinical time series from intensive care unit (ICU). Each time series contains 35 measurements such as Albumin, heart-rate etc., which are irregularly sampled at the first 48 hours after the patient's admission to ICU. We stress that this dataset is extremely sparse. There are up to 78% missing values in total. For this dataset, we do both the imputation task and the classification task. To evaluate the imputation performance, we randomly eliminate 10% of observed measurements from data and use them as the ground-truth. At the same time, we predict the in-hospital

① https://github.com/NIPS-BRITS/BRITS

death of each patient as a binary classification task. Note that the eliminated measurements are only used for validating the imputation, and they are never visible to the model.

4.4.1.3 Localization for Human Activity Data

The UCI localization data for human activity [74] contains records of five people performing different activities such as walking, falling, sitting down etc (there are 11 activities). Each person wore four sensors on his/her left/right ankle, chest, and belt. Each sensor recorded a 3-dimensional coordinates for about 20 to 40 millisecond. We randomly select 40 consecutive steps as one time series, and there are 9317 time series in total. For this dataset, we do both imputation and classification tasks. Similarly, we randomly eliminate 10% observed data as the imputation ground-truth. We further predict the corresponding activity of observed time series (i.e., walking, sitting, etc).

4.4.2 Experiment Setting

4.4.2.1 Model Implementations

We fix the dimension of hidden state $\mathbf{h}_t$ in RNN to be 64. We train our model by an Adam optimizer with learning rate 0.001 and batch size 64. For all the tasks, we normalize the numerical values to have zero mean and unit variance for stable training.

We use different early stopping strategies for pure imputation task and classification tasks. For the imputation tasks, we randomly select 10% of non-missing values as the validation data. The early stopping is thus performed based on the validation error. For the classification tasks, we first pre-train the model as an imputation task. Then we use 5-fold cross validation to further optimize both the imputation and classification losses simultaneously.

We evaluate the imputation performance in terms of *mean absolute error* (MAE) and *mean relative error* (MRE). Suppose that $\texttt{label}_i$ is the ground-truth of the i-th item, $\texttt{pred}_i$ is the output of the i-th item, and there are N items in total. Then MAE and

MRE are defined as

$$\texttt{MAE} = \frac{\sum_i |\texttt{pred}_i - \texttt{label}_i|}{N}; \quad \texttt{MRE} = \frac{\sum_i |\texttt{pred}_i - \texttt{label}_i|}{\sum_i |\texttt{label}_i|}.$$

For air quality data, the evaluation is performed on its original data. For heath-care data and activity data, since the numerical values are not in the same scale, we evaluate the performances on their normalized data. To further evaluate the classification performances, we use *area under ROC curve* (AUC)[75] for health-care data, since such data is highly unbalanced (there are 10% patients who died in hospital). We then use standard accuracy for the activity data, since different activities are relatively balanced.

4.4.2.2 Baseline Methods

We compare our model with both RNN-based methods and non-RNN based methods. The non-RNN based imputation methods include:

- Mean: We simply replace the missing values with corresponding global mean.
- KNN: We use *k*-nearest neighbor[46] to find the similar samples, and impute the missing values with weighted average of its neighbors.
- Matrix Factorization (MF): We factorize the data matrix into two low-rank matrices, and fill the missing values by matrix completion[16].
- MICE : We use Multiple Imputation by Chained Equations (MICE), a widely used imputation method, to fill the missing values. MICE creates multiple imputations with chained equations[17].
- ImputeTS: We use ImputeTS package in R to impute the missing values. ImputeTS is a widely used package for missing value imputation, which utilizes the state space model and kalman smoothing[76].
- STMVL : Specifically, we use STMVL for the air quality data imputation. STMVL is the state-of-the-art method for air quality data imputation. It further utilizes the geo-locations when imputing missing values[72].

We implement KNN, MF and MICE based on the python package fancyimpute①. In recent studies, RNN-based models in missing value imputations achieve remarkable

① https://github.com/iskandr/fancyimpute

Table 4.1 Performance Comparison for Imputation Tasks. MAE (MRE%)

Method		Air Quality	Health-care	Human Activity
Non-RNN	Mean	55.51 (77.97%)	0.720 (100.00%)	0.767 (96.43%)
	KNN	29.79 (41.85%)	0.732 (101.66%)	0.479 (58.54%)
	MF	27.94 (39.25%)	0.622 (87.68%)	0.879 (110.44%)
	MICE	27.42 (38.52%)	0.634 (89.17%)	0.477 (57.94%)
	ImputeTS	19.58 (27.51%)	0.390 (54.2%)	0.363 (45.65%)
	STMVL	12.12 (17.40%)	/	/
RNN	GRU-D	/	0.559 (77.58%)	0.558 (70.05%)
	M-RNN	14.24 (20.43%)	0.451 (62.65%)	0.248 (31.19%)
Ours	RITS-I	12.73 (18.32%)	0.395 (54.80%)	0.240 (30.10%)
	BRITS-I	11.58 (16.66%)	0.361 (50.01%)	0.220 (27.61%)
	RITS	12.19 (17.54%)	0.300 (41.89%)	0.248 (31.21%)
	BRITS	11.56 (16.65%)	0.281 (39.14%)	0.219 (27.59%)

performances[59,77–79]. We also compare our model with existing RNN-based imputation methods, including:

- GRU-D: GRU-D is proposed for handling missing values in health-care data. It imputes each missing value by the weighted combination of its last observation, and the global mean, together with a recurrent component[59].
- M-RNN: M-RNN also uses bi-directional RNN. It imputes the missing values according to hidden states in both directions in RNN. M-RNN treats the imputed values as constants. It does not consider the correlations among different missing values[79].

We compare the baseline methods with our four models: RITS-I (Section 4.3.1), RITS (Section 4.3.2), BRITS-I (Section 4.3.3) and BRITS (Section 4.3.3). We implement all the RNN-based models with PyTorch, a widely used package for deep learning. All models are trained with GPU GTX 1080.

4.4.3 Experiment Results

Table 4.1 shows the imputation results. As we can see, simply applying naıve mean imputation is very inaccurate. Alternatively, KNN, MF, and MICE perform much better than mean imputation. ImputeTS achieves the best performance among all the non-RNN methods, especially for the heath-care data (which is smooth and contains few sudden waves). STMVL performs well on the air quality data. However, it is specifically designed for geographical data, and cannot be applied in the other datasets. Most of RNN-based methods, except GRU-D, demonstrates significantly better performances in imputation tasks. We stress that GRU-D does not impute missing value explicitly. It actually performs well on classification tasks. M-RNN uses explicitly imputation procedure, and achieves remarkable imputation results. Our model BRITS outperforms all the baseline models significantly. According to the performances of our four models, we also find that both bidirectional recurrent dynamics, and the feature correlations are helpful to enhance the model performance.

We also compare the accuracies of all RNN-based models in classification tasks. Table 4.2 shows the performances of different RNN-based models on the classification tasks. As we can see, our model BRITS outperforms other baseline methods for classifications. Comparing with Table 4.1, GRU-D performs well for classification tasks. Furthermore, we find that it is very important for GRU-D to carefully apply the dropout techniques in order to prevent the overfitting (we use $p = 0.25$ dropout layer on the top classification layer). However, our models further utilize the imputation errors as the supervised signals. It seems that dropout is not necessary for our models during training.

To better understand our model, we generate a set of univariate synthetic time series. Speficically, we randomly generate a time series with length $T = 36$, using the state-space representation [80]:

$$
\begin{aligned}
x_t &= \quad _t + \quad _t + \quad _t; \\
\quad _t &= \quad _{t-1} + \quad _{t-1} + \quad _t; \\
\quad _t &= \quad _{t-1} + \quad _t;
\end{aligned}
$$

Table 4.2 Performance Comparison for Classi cation Tasks

Method	Health-care (AUC)	Human Activity (Accuracy)
GRU-D	0:828 0:004	0:939 0:010
M-RNN	0:800 0:003	0:938 0:010
RITS-I	0:821 0:007	0:934 0:008
BRITS-I	0:831 0:003	0:940 0:012
RITS	0:840 0:004	0:968 0:010
BRITS	0:850 0:002	0:969 0:008

$$x_t = \sum_{j=1}^{t-1} x_{t-j} + \omega_t,$$

where x_t is the t-th value in time series. The residual terms ω_t, α_t and β_t are randomly drawn from a normal distribution $N(0, 0.3)$. We eliminate about 22% values from the generated series, and compare our model BRITS-I (note the data is univariate) and ImputeTS. We show three examples in Fig. 4.3.

Figure 4.3 Example for synthetic time series imputation. Each column corresponds to one time series. The gures in the rst row are imputations of ImputeTS algorithm, and the gures in the second row are imputations by BRITS-I .

The rst row corresponds to the imputations of ImputeTS and the second row corresponds to our model BRITS. As we can see, our model demonstrates better performance than ImputeTS. Especially, ImputeTS fails when the start part of time series is missing. However, for our model, the imputation errors are backpropagated to pre-

Chained Equations (MICE)[17] first initialize the missing values arbitrarily and then estimates each missing variable by Gibbs sampling. The graphical model[81] introduces a latent variable for each missing value, and finds the latent variables by learning their transition matrix. There are also some data-driven methods for missing value imputation. Yi et al.[72] imputes the missing values in air quality data with geographical features. Wang et al.[15] imputes the missing values in recommendation system with collaborative filtering.

Recently, some researchers attempted to impute the missing value with recurrent neural networks[18,77,78]. The recurrent components are trained together with the classification/regression component, which significantly boosts the accuracy. However, they assumed that the missing values can be represented as the combination of global mean and the most recent observations. They used RNN to impute the missing data in a smooth fashion but ignores the relationships in the recurrent dynamics. To the best of our knowledge, all of the existing methods treated the imputed values as static constants and have limited utilization of future information. Different from the existing methods, our model directly approximates the recurrent dynamical system and demonstrates superior results.

4.6 Conclusion

In this chapter, we proposed BRITS, a novel method to use recurrent dynamics to effectively impute the missing values in multivariate time series. Instead of imposing assumptions over the data-generating process, our model learns the representations of data correlation in a bidirectional recurrent dynamical system, and imputes missing values based on the learned representations directly. Our model treats missing values as variables of the bidirectional RNN graph. Thus, we get the delayed gradients for missing values in both forward and backward directions, which makes the imputation of missing values more accurate. We performed the missing value imputation and classification/regression simultaneously within a joint neural network. Experiment results show that our model demonstrates more accurate results for both imputation and clas-

sification/regression than state-of-the-art methods.

, 5 à Handling Massiveness in User Identification

We further study the representation learning for extremely massive spatio-temporal data in this chapter. Specifically, we use the user identification problem as an example. In this problem, we investigate efficient ways of identifying users across heterogeneous data sources, such as different GPS-embedded devices, mobile apps or location-based service providers. We present a MapReduce-based framework called *Automatic User Identification* (AUI) which is easy to deploy and can scale to a very large data set. Our framework is based on a novel similarity measure called *signal based similarity* (SIG) which measures the similarity of users' trajectories gathered from different data sources, typically with very different sampling rates and noise patterns. In signal based similarity, we adopt a co-occurrence representation of users' trajectories in multiple resolutions, and calculate the similarity scores based on the obtained representations. Such similarity measure is very robust for the massive mobility datasets. We conduct extensive experimental evaluations, which show that our framework outperforms the existing methods significantly.

5.1 Introduction

Ubiquitous location based services and applications have enabled people to use GPS-embedded devices for navigation, travel planning and geolocation information sharing in their daily life. Such mobility data is now collected routinely at a very large scale. The large volume of mobility data gives rise to new opportunities for discovering patterns and characteristics of human mobility behaviors. An increasing number of researches empowered by mobility data has emerged recently. Meanwhile, mining of such mobility data also shows great potentials in various industrial and commercial applications, including traffic analysis [1–3], travel recommendation [4–7,82–85], location-based social network [8–13], geographical searching [86,87] etc.

In real applications, mobility data is usually generated from heterogeneous data

sources, such as different GPS-embedded devices, mobile apps, or LBS providers etc. In this chapter, we aim to study the efficient approach of identifying users from mobility datasets collected from heterogeneous sources. A closely related topic is *user similarity search* where the goal is to retrieve a subset of users with similar spatio-temporal patterns. Essentially, we can think of the user identification problem as a special case of user similarity search problem. For each trajectory, we retrieve the most similar ones from another data source and identify whether they belong to the same person.

Despite the fact that the user similarity search has been studied quite extensively [83,88–91], not much work has investigated the mobility data collected from heterogeneous data sources, which is much more complicated. In our experiments, the data is collected from various mobile apps, including the navigation data, map queries, geo-tagged records in social platforms etc (see Section 5.6 for details). To make our exposition more concrete, we first illustrate some distinct features of our data and our objective.

- The most distinctive feature we have found about the trajectories is the sheer variety of the sampling rates among different sources. For example, trajectories of GPS navigation data are usually sampled at a very high rate whereas the geo-tags or map queries are sampled with an extremely low rate. Even in the same data source, the utilization frequencies could vary drastically during different time intervals. On the other hand, most prior work uses approximately uniformly (and often densely) sampled data [83,88,89,92].
- For some sparse trajectories, it is almost impossible to infer any information of user movement. For example, for the geo-tagged check-in data in our data set, each user only generated one GPS record every 63.2 days on average. Most of prior work measures the trajectory similarities based on the movement behaviors of the users such as the speed, move direction, spatial-temporal closeness (i.e., two trajectories are similar only if they appeared in approximately the same place at approximately the same time) [88,92–96]. However, such features are not available in our data due to the extreme sparsity.
- The trajecotries of the users with close relationship usually have a significant

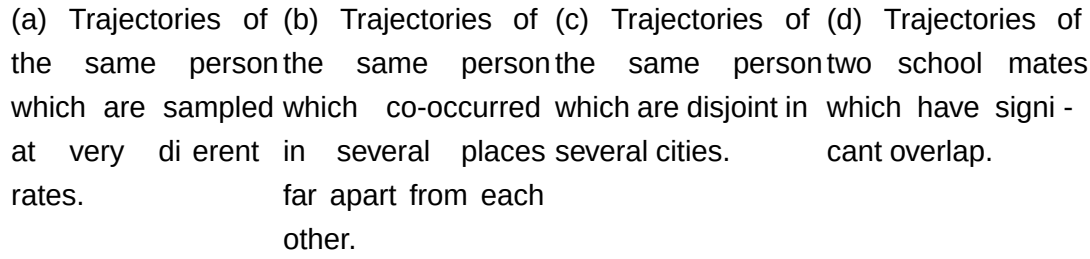

Figure 5.1 Four typical cases observed in the real dataset

overlap. For example, we investigate the mobility data of several students who study in University T. Most of their trajectories lie on their department buildings and dormitories. Such overlap renders it dicult to distinguish them. However, few prior work investigates user identi cation problem under such case.

For di erent data sources, the trajectories can be temporally disjoint. For example, two data sets of the same group of anonymized users with inconsisten user id, one is collected at January and another is collected at February. Especially, for the businessmen, they may go to several different cities during several months which makes it dicult to measure their similarities.

To provide some intuitions for the readers and to illustrate the challenges, we show an example in Example 2.

Example 2 In Figure 5.1, we show four typical cases observed in the real datasets (there are many other cases or combinations of those cases, that are impossible to list exhaustively).

Each of (a)(b)(c) represents two trajectories (from different data sources) that belong to the same person and (d) shows the trajectories that belong to two schoolmates.

In (a), the white trajectory is sampled at a much lower rate than the red trajectory but they both occurred in several xed places frequently. Such co-occurrences

tremely noisy cases effectively. The experiment result shows that our measure is more robust and accurate for our user identification problem with heterogeneous data sources.

- We adopt a rejection strategy in order to reduce the mis-identification cases. Many application scenarios are highly sensitive to misidentification, i.e. a few erroneous cases could lead to serious consequences. Our strategy enhances the accuracy of the framework significantly.
- We evaluate our framework by 6 experiments of different cases. For the easiest case (31511 users in China), we achieve an accuracy of 94.4%. For the hardest case (14115 college students who study in the same campus), we achieve an accuracy of 90.9% whereas the best existing method only achieves an accuracy of 61.38% in this case.

5.2 Formulation and Overview

The problem studies the user identification across heterogeneous data sources. We first present several useful definitions.

Definition 5 (Trajectory) A trajectory $T = \{p_1, p_2, \ldots, p_{|T|}\}$ is a temporally ordered sequence of spatio-temporal points. Each point p_i is associated with three attributes x, y, t where the pair (x, y) [①] indicates the coordinate of p_i and t indicates the timestamp when p_i was recorded.

Here we stress that the trajectories in our data could be extremely sparse, i.e., there may be less than one spatio-temporal point per day in average.

Definition 6 (Mobility data set) A mobility data set D is defined as a collection of trajectories. Each trajectory $T \in D$ is associated with an id $T.id$.

Definition 7 (Matching trajectory) Suppose two trajectories T_A, T_B are collected from different mobility data sets. If T_A and T_B are generated from the same user, then we call T_B a matching trajectory of T_A.

① We use the mercator coordinate in our experiment.

Our problem is defined as follow. Given two mobility data sets D_A and D_B (usually collected from two different data sources), for each $T_A \in D_A$, our goal is to identify whether there exists $T_B \in D_B$ which is the matching trajectory of T_A. We do not assume any property of the mobility data set. Thus, the sampling rates of the trajectories could be very high or extremely low. Furthermore, D_A and D_B could be temporally disjoint, i.e., they are collected at different time intervals.

We explain the reason why we can achieve user identification across heterogeneous mobility data sources. Montjoye et al.[99] showed that the human mobilities are highly unique. Each individual has his/her own mobility pattern. People tend to visit the places where they often visited in the past. Even for the users with very close relationship, they still have noticeable different mobility patterns as we illustrated in Fig 5.1(d). Such uniqueness of human mobility allows us to identify trajectories that belong to the same user from different sources.

To handle the extremely sparse trajectories (which makes it hard to infer any mobility pattern from the daily data), we accumulate the trajectories during a long time interval (for example, the trajectories during 3 months). Thus, by accumulating the historical mobility data, it is possible to infer the mobility pattern of a user such as the places he/she tends to visit. Fig. 5.2 shows an example in our data set where the trajectory is collected from February to May in 2015 with 5 points per day in average. By accumulating all these points, we can clearly see that this user usually stay at home and the workplace. Moreover, he/she had visited Wangfujing and Beijing Botanical Garden one day in the past 3 months. Motivated by this, we consider both the frequency of

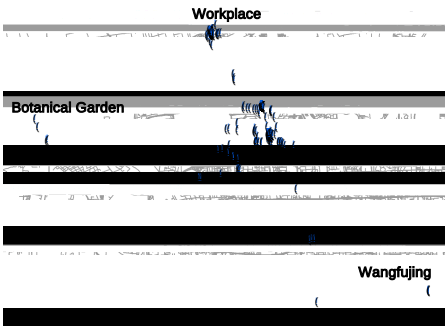

Figure 5.2 A trajectory in our data.

spatial co-occurrences (i.e., two trajectories co-occurred at approximately the same location) under different granularities and the locations where they took place. We define our trajectory similarity mainly based on these factors.

We first present an overview of AUI. Our framework consists of three stages: the pre-processing stage, the multi-resolution filtering stage and the verification stage. All of the stages are implemented on the MapReduce frame. In the pre-processing stage, we first perform data compression by transforming each trajectory into consecutive transitions between a set of *stay points*, i.e., the locations where the user stays for a while rather than just passing by. Since for large scale data sets, pairwise comparison is rather expensive, in the multi-resolution filtering stage, we partition the map into multiple grids with multi-resolution. For each trajectory $T_A \in D_A$, we select a small subset of trajectories in D_B which co-occurred frequently in multiple cells with T_A as the candidate set of T_A. Finally, as the trajectories we deal with are sampled at very different rates and extremely noisy, in the verification stage, we evaluate each candidate with the signal based similarity which can handle such cases effectively and we carefully select the matching trajectory. We further extend our similarity measure and adopt an effective rejection strategy in the verification stage to reduce the misidentification cases (Section 5.5).

5.3 Pre-Processing

The pre-processing stage transforms each trajectory into a set of stay points, i.e., the location points that the user stay for a while rather than passing by. Thus stay points usually carry the particular semantic meanings such as the building they live or the park they went[89]. The pre-processing stage consists of map-only jobs. For each trajectory $T \in D_A \cup D_B$, the map function takes $T.id$ as the key and T as the value. We use a similar method as in[89] to pre-process the trajectories. For each trajectory in a mobility data set, we split it into consecutive segments. A segment is defined as a series of continuous location points sharing the same mobility status, such as stay, move and pass-by. For example, a user drove for 2 hours to a shopping mall and spent 3 hours in the mall, then drove to another place. Such trajectory can be split into 3 segments: driving to the mall,

Algorithm 1: Pre-Processing

```
1  Map(⟨T.id; T⟩)
2  S ← {}, P ← {p_1};
3  for i = 2:::|T| do
4      p_b ← P[1] ;
5      t ← p_i.t − p_b.t;
6      d ← Dis(p_i; p_b);
7      if d ≥ Δ_d or t ≥ Δ_t^u then
8          if t ≥ Δ_t^l then
9              sp.loc ← MeanCoordinate(P) ; // stay point
10             sp.cnt ← |P|; // number of around points
11             S.add(sp) ; // add the stay point into S
12         P ← {p_i}; // start a new segment
13     else
14         P.add(p_i); // add p_i into current segment
15 emit(⟨T.id; S⟩)
```

staying at the mall and driving to another place. For each segment under stay status, we use the geometric center of the segment as the corresponding stay point. We then use a series of stay points as the compressed trajectory. A location point is considered as under stay status if the user spent more than Δ_t^l minutes but less than Δ_t^u minutes around this point within a distance of Δ_d meters. Comparing with the method proposed by Li et al[89], we set an additional upper bound Δ_t^u here (typically 12 hours). The reason is that in some sparse trajectories the time gap between two consecutive location points can exceed several days, which renders it difficult to infer the user's mobility status during the gap. For such cases, we regard the latter location point as the start point of a new segment. Furthermore, we stress that the pre-processing stage mainly focuses on the non-sparse trajectories. For the extremely sparse trajectories, almost a single spatio-temporal point can represent a stay point. See Algorithm 1 for the pseudo-code.

Algorithm 2: Multi Resolution Filtering

```
   // first phase
1  Map(hT:id; Si)
2  for i = 1 : : : N do
3  |  sz  the cell size of G_i;
4  |  for sp 2 S do
5  |  |  cx; cy  the cell coordinate that sp lies in G_i;
6  |  |  c  (cx; cy; sz);
7  |  |  emit(hc; hT:id; sp:cnti i);
8  Reduce(hc; list(hT:id; T:cnti)i)
9  if len(list(hT:id; T:cnti))  m_c then
10 |  l_A  items of list(hT:id; T:cnti) where T:id 2 D_A;
11 |  l_B  items of list(hT:id; T:cnti) where T:id 2 D_B;
12 |  for hT_A:id; T_A:cnti 2 l_A do
13 |  |  for hT_B:id; T_B:cnti 2 l_B do
14 |  |  |  o  min{T_A:cnt; T_B:cnt}; // co-occurrences
15 |  |  |  output(T_A:id; hT_B:id; c; oi);
16
   // second phase
17 Map(hT_A:id; list(hT_B:id; c; oi)i)
18 for hT_B:id; c; oi 2 list(hT_B:id; c; oi) do
19 |  sz  the size of c;
20 |  Increase the ranking score of T_B:id by (r_sz  o);
21 l  top Q trajectory ids with largest ranking scores;
22 for T_B:id 2 l do
23 |  Merge the items of T_B:id into one key-value pair
   |   hT_A:id; hT_B:id; list(hc; oi)ii ;
24 |  output(hT_A:id; hT_B:id; list(hc; oi)ii );
```

Figure 5.3 Running example of the multi-processing stage

5.4 Multi-Resolution Filtering

Since the pair-wise comparison is expensive, especially for the large data sets, in the multi-resolution ltering stage, for each $T_A \in D_A$, we only select a small subset

$f_s(o_k; ;\ )$ to calculate the observed signal in c_k

$$\text{ob}(c_k) = f_s(o_k) = \frac{}{1 + e^{-\ o_k}} - \frac{}{2} \tag{5-1}$$

where ; are two parameters to be specified (see Section 5.6 for details).

To recover the stimulus signals efficiently, we simply process the observed signals sequentially to approximate the stimulus signals. For the c_k cell, we assume that it is only affected by the cells $c_1; : : : ; c_{k-1}$. Note that the stimulus signals we obtained may be different under different orders of cells. However, in the experiments, we find a specific order rarely affects the accuracy for user identification. Specifically, we use $\text{st}(c_k)$ to denote the stimulus signal in cell c_k. Initially, we have that $\text{st}(c_1) = \text{ob}(c_1)$. For any $k > 1$, we have

$$\text{st}(c_k) = \max\left\{ \begin{array}{l} \text{ob}(c_k) - \sum_{l<k} \text{st}(c_l) \cdot \ ^{\text{Dis}_{\text{grid}}(c_k;c_l)} \\ 0 \end{array} \right. : \tag{5-2}$$

where Dis_{grid} is the distance metric defined on the grid. In our algorithm, we use Euclidean distance of cell centers scaled by the side length of the cell as Dis_{grid}. Thus, the kernel cells are the cells with non zero stimulus signal. We use K to denote the indices of kernel cells, i.e., $K = \{k : \text{st}(c_k) > 0\}$:

Next, we take the distances between the kernel cells into consideration. We use $\text{md}(c_k)$ to denote the minimum distance from cell c_k to the previous kernel cells, i.e.,

$$\text{md}(c_k) = \min_{(l<k)\wedge(l\in K)} \text{Dis}_{\text{grid}}(c_k; c_l):$$

Similarly, we define a sigmoid function $f_d(\text{md}(c_k); ;\ )$ which has the same form as f_s with parameters and to be specified. Then, the signal in the grid is:

$$\text{sig}_i = \text{st}(c_1) + \sum_{k\in K\setminus\{1\}} \text{st}(c_k) \cdot (1 + f_d(\text{md}(c_k))) \tag{5-3}$$

The equation 5-3 essentially captures the feature that we showed in Fig. A.4(c), i.e., it is significant to observe the co-occurrences taking far apart from each other.

Algorithm 3: Signal Based Similarity for T_A and T_B

```
for i = 1:::N do
    K ← {};
    for k = 1:::m do
        o ← co-occurrences in cell c_k ∈ C;
        ob(c_k) = f_s(o);
        if k == 1 then
            st(c_1) ← ob(c_1);
        else
            st(c_k) ← max{0; ob(c_k) − Σ_{l∈K} st(c_l) · Dis_grid(c_k;c_l)};
        if st(c_k) > 0 then
            K:add(k);
    sig_i = st(c_1) + Σ_{k∈K\{1}} st(c_k) · (1 + f_d(md(c_k)));
SIG = Σ_{i=1}^{N} _i · sig_i;
```

Algorithm 4: Verification

```
Map(⟨T_A:id; ⟨T_B:id; list(⟨c; o⟩)⟩⟩)
calculate the signal based similarity SIG;
calculate the weighted jaccard similarity WJS;
if SIG ≥ _SIG and WJS ≥ _WJS then
    emit(⟨T_A:id; ⟨T_B:id; SIG; WJS⟩⟩);

Reduce(⟨T_A:id; list(⟨T_B:id; ; SIG; WJS⟩)⟩)
if ∃T*_B:id ∈ list(T_B:id) dominate all the others then
    output(⟨T_A:id; T*_B:id⟩);
```

Finally, we sum the signals of all the grids. We set a weight parameter $_i$ for the G_i. Again, the finer granularity G_i is, the larger weight parameter $_i$ becomes. Thus, the signal based similarity is defined as:

$$\mathrm{SIG} = \sum_i {}_i \cdot \mathrm{sig}_i \tag{5-4}$$

See Algorithm 3 for the pseudo code.

5.5.2 Verification

In the verification stage, each map function takes a key-value pair $\langle T_A.id; \langle T_B.id; list(\langle c; o\rangle)\rangle\rangle$. Note that $list(\langle c; o\rangle)$ contains all the co-occurrences in all grids. Thus, we can directly calculate the signal based similarity from $list(\langle c; o\rangle)$.

To verify the candidates, a feasible way is to measure each candidate with T_A with the signal based similarity and select the most similar one as the matching trajectory. In the experiment section, we show that such strategy achieves a reasonable performance. However, many application scenarios are highly sensitive to misidentification, i.e. a few erroneous cases could cause serious consequence. It is necessary to adopt a rejection strategy, i.e., to refuse to identify the trajectories that may lead to mistakes. In Section 5.6, we show that it is difficult and inaccuracy to obtain a rejection strategy with only the signal based similarity. Motived by this, we adopt an effective rejection strategy with another similarity measure called the weighted Jaccard similarity which is applied in combination with the signal based similarity. The weighted Jaccard similarity was first proposed by Ioffe et al.[100]. It measures the similarity between two weighted set. For our problem, we regard each trajectory as a set of cells it has visited. The weight of each cell is the corresponding visiting frequency. Thus, for a given pair of trajectories, it measures the similarity of the places they visited and the corresponding frequencies. For a specific grid G_i, we use v_c^A (v_c^B resp.) to denote the frequency of T_A (T_B resp.) being observed at cell c. The similarity of T_A and T_B on grid G_i is defined as:

$$\mathrm{wjs}_i = \frac{\sum_c \min\{v_c^A; v_c^B\}}{\sum_c \max\{v_c^A; v_c^B\}}. \tag{5-5}$$

We show an example in Fig. 5.5. The points in the same shape (circle or triangle) indicate a specific id. We use the red color to indicate that the user occurred in this cell for 5 times. Similarly, we use yellow color to represent the frequency of 3 and the white color to represent the frequency of 1. Thus, the weighted Jaccard similarity in the left figure is $17/27 = 0.63$ and the weighted Jaccard similarity in the right figure is $17/49 = 0.35$.

Note that the term $\min\{v_c^A; v_c^B\}$ is exactly the frequency of co-occurrence of T_A and

Figure 5.5 Example of the weighted Jaccard similarity.

T_B in cell c which is already contained in $list(c, o_i)$, all we need to do is calculate the term $max\{v_c^A, v_c^B\}$. Re-writing the denominator as $\sum_c v_c^A + \sum_c v_c^B - \sum min\{v_c^A, v_c^B\}$, we nd that the rst two terms are exactly the accumulated frequency of all the stay points which can be obtained easily at the beginning. Thus, the weighted Jaccard similarity on G_i also can be calculated from $list(c, o_i)$. Similarly, we use $WJS = \sum_i \alpha_i wjs_i$ as the weighted Jaccard similarity of T_A and T_B.

We rst calculate the signal based similarity and the weighted Jaccard similarity for all the candidates. Next, we set two small thresholds τ_{SIG} and τ_{WJS} to lter out the dissimilar candidates. If a candidate T_{cid} has its signal based similarity smaller than τ_{SIG} or its weighted Jaccard similarity smaller than τ_{WJS}, we lter it out from the candidate set. After the ltering, if there exists a unique maxima in the candidate set, we return it as the matching trajectory. Otherwise, we reject the identi cation. A candidate is called a maxima if there do not exist another candidate with both of the two similarities greater than it. For example, suppose we have 3 candidates with (SIG, WJS) equal to $(12.7, 0.4), (13.1, 0.3), (10, 0.35)$ respectively. Then the rst two candidates are both maxima. See Algorithm 4 for the pseudo code.

We stress that when the mobility data sets are sampled in very high rates, the weighted Jaccard similarity becomes very low which leads to a great error. However, in the experiment section, we show that by combining with the signal based similarity and setting an extremely small threshold τ_{WJS}, AUI achieves a great performance.

5.6 Experiment Evaluation

In this section, we conduct extensive experiments on real mobility data sets to demonstrate the performance of the proposed algorithm. We rst describe our experiment setting in Section 5.6.1 then we continue by presenting the results of the param-

搜" 轨迹 à 此((7 Æ别î 题- • (ƒ ï 能 ü ô 轨迹重à ö „ ï 率 > 提Ø (近„ 研究- Ranu等º [95]研究† ô 采样率 轨迹„ø < ' 度量 Öì 制š † * ð : 编辑 • 影距» EDwP „ 稳e 距» ý p e 9 配ï Ø 采样率 „ 轨迹 6 而 Öì „ ¹ Õ • 推- (7 „ û 动á o (ì „ î 题- 1 Ž \ („ • 度稀• ' ö û 动á o 并 ï 推测

(近 „ 研究- 系列ˆ ù (7 « ý Æ别„ 9 进¹ Õ Chen等º 提ú Í ù (7 轨迹L : „ h ° h : ¹ Õë Z STUL[105] ú Ž å å \ Öì 进 步提ú * ú Ž Æ度ý p 和µ „ ¹ Õ e 0 ¡ 同网格 „ (7 « ý Æ别„ 置á 度 å ¹ Õ 得到† ^ 常o } „ H œ $ v / ù Ž ú Ž M 置„ > ¤ p 据 ì • 强 „ / Öì „ ¹ Õ / ì (, 章- @提ú ¹ Õ„ 后í å \

A.5 总Ó

GPS‰ e 式¾ , z 能K : 环路 h „ n Ê ü ô † ö 空p 据„ ¸式ž • 正, ì @ô „ 挖掘ö 空p 据\ („ ÷ < (各Í å 和F 应 (- w 重• I 6 而 (¸ ž 际Å µ ‹ , ¤ 通推P 和ú Ž M 置„ > ¤ 网Ü ö 空p 据通常/ 呈现海量Ä ! 并w 较' j ó ö 空p 据- „ \ (! 式 (pattern)× B „ 空ô ø s ' ö ô • 赖' 和¸ 部à „ 影响 :† (挖掘ö 空p 据¹ b Ö 得 I 和Æ确„ Ó œ Î p 据- 提Ö ú H „ y 征h : $: 重• (, º ‡ - ì 研究ö 空p 据„ h 征f ` ì 提ú ö 空p 据 h 征f ` „ ø 挑 并 ì (* w S „ ‹ P e ô , U (f ` ö 空p 据 „ y 征h : „ 同ö ã ³ ø 应„ 挑

(第A.2, - ì 研究† , U (p 据h 征- 捕· B „ ö 空• 赖 ì å¤ 通ú L ö ô 预测î 题: ‹ (这* î 题- ì 提ú † * ú Ž 深度^ Ï 网Ü „ 端到端框¶ ë Z DeepTTE ì „ ! ‹ 含 * ú Ž 0 á o „ w ï ! W (e 捕捉p 据- „ Æ粒度空ô s 联 * 递归^ Ï 网Ü ! W (e f ` 同道路K ô „ ö ô • 赖 å Ê * 部^ ' ! W( e 各类 部à „ 影响 这Í ¶„ ï å H 0 ê 动0 f ` Ù š p 据„ ö 空y 征h : (DeepTTE„ 顶B ì 还¾¡ † * û 务f ` ! W å Ó 合分段道路预测和t S 道路预测$ Í ¹ Õ Î 而达到ô } „ H œ

(第A.3, - ì 研究 ö 空p 据- „ 缺1 < ì 提ú † Í h ° „ 缺1 e h ¹ Õ BRITS BRITS利(循环动力系ß e f ` w B • 赖s 系„ ö 空p 据y 征h : 并根据f ` 到„ y 征h : e 进L 缺1 e h ì „ ! ‹ ù p 据 过程进L y 殊' G ¾ Ö 而ãK ì • (Ì 向递归^ Ï 网Ü e f ` 捕· 到„ø s ' „ y 征h : 并ô 接预测缺1 < ì „ ! ‹ 缺1 < Æ : RNN¡ 算þ „ Ø 量 e h Ó œï î (Í 向 播过程- ï å « E 分 Î 而 • 得e h Ó œô 加Æ确 同ö 缺1 e h 算Õ Ù š ö ô 序列„ 分类/Þ 归! W进 L 联合- Ã ž ŒÓ œh ì „ ! ‹ 比现 „ ¹ Õ (e h 和分类/Þ 归¹ b 都h 现ú ô Æ确„ Ó œ

(第A.4, - ì 进 步考Q • 端 B ö 空p 据 „ h : f ` \ : * w S „ ‹ P ì 研究† 异• 轨迹p 据- Æ别同 (7 „ ¹ Õ 这* î 题 - 涉Ê 到„ 轨迹p 据集^ 常 B v 采样速度î 异较' ü ô Æ别(7 „ : o 同样Ø 得^ 常 B ì 提ú signal based similarity (SIG) : 轨迹ø < ' 度量 e 测量' Ä ! 轨迹p 据集- „ 轨迹ø < ' (SIG- ì • ((7 轨迹„ q 现‹ö \ : p 据h : ¹ Õ ú Ž å h : Õ ì (分辨率 ¡ 算(7 ø < ' 得分 ì 提ú † * 名: Automatic User Identification (AUI), ú Ž MapReduce, 框¶ ƒ • ì 能 (^ 常' „ p 据Ä ! 快速¡ 算SIGø < ' ì 进L † 广Û „ ž Œ ž ŒÓ œh ì „ ! ‹ > W Ž 现 „ ¹ Õ

ì (, º ‡ - • („ 框¶ 和! ‹ : ö 空p 据„ h : f ` ¹ Õ 提› † Ÿ 则' „ 指ü y 别„ , ‡ 提ú „ ¹ Õ 能 捕捉ö 空p 据- B „ • 赖s 系 缺1 < 并 H p 据 B î 题

A.6 *e å \

海量ö 空p 据„ h : f ` / * ^ 常重• w 挑 ' „ î 题 ƒ ( f / 和å 领ß Í 6 / * í è Ý 题 ( , , - ì Ë Í ö 空p 据h : f ` „ $* *e ¹ 向

首H ù Ž › • v B „ : o f ` H „ y 征h : Í 6 • 赖K 动¾¡ „ Ä 则 ‹ , ù Ž 第A.4, - „ (7 Æ别î 题 ì • 6 ˆ ù p 据采(† Í

º: y 殊¾¡ „ 算Õ e ¡ 算(7 轨迹K ô „ø < ' (近„ 研究- › 研究º 员• (Graph Embedding ¹ Õ (• • 赖º å y 征„ Å µ Î 轨迹 p 据Æ别ú 同(7 [107,108] 6 而 Öì „ ¹ Õ ê 适(Ž 采样率à N ø 同„ ø ù Ä t „ û 动p 据 1 î 前@å Í 6 ¡ H „ ¹ Õ 能(' 量• 赖º å Ä 则Å µ ù B „ ' Ä ! 轨迹p 据进L (7 Æ别 (ž 际应(- 通过精心¾¡ „ 深度! ‹ Î Ÿ Ë p 据- ô 接f ` y 征h : ï å • 得 ì 能 ê 动Ñ 现p 据- („ B 次y 征 6 而 这› Ó „ „ ¾¡ 同样并 ¹ 并 运 L ö ô ù Ž ' Ä ! p 据集而ø ù 较N H ø 比K ú Žº å Ä 则„ y 征h : 通常稳š ' 强并 很¹ (' Ä ! p 据集 i U F • 耗费ô ' 量„ º 力 , U ($ Í y 征h : ¹ ÕK ô „ Ö到 * } „ 平a ¹ / ì 考Q „ *e å ¹ 向K

æ * 研究¹ 向/ , U 利(迁û f ` e 帮助f ` ö 空p 据„ y 征h : ¸ ö 空p 据挖掘û 务- ø q ' ! 式 ‹ , G ¾ ì ò Ï 6 集† 过» * Î 市- 每a 道路„ å 5 程度 ì è (预测*e à 分Ÿ ...每a 道路„ å 5 程度 这* î 题 ì „ ¤ 通ú L ö ô 预测î 题Ø 度ø s , ì (第A.2, -Ë Í „ 那样 Å L ö ô 和å 5 程度都× ø 邻路段„ 空ô ø s ' 和ö ô • 赖' „ 影响 à 此 , œ ì 能 ú L ö ô 0 ¡ î 题- f ` 到„ y 征 迁û 到å 5 等 § 预测î 题- 利Ž ì ô 精确„ 0 ¡ 每a 道路„ å 5 程度 ( 迁û f ` - ì è Ž Ÿ Ë î 题- f 到„ å Æ 应(到ø s „î 标î 题- [109] 6 而 现 „ 转û f ` ¹ Õ ' 集- ( ÆÉ û 务 , þ Ï 分类 ù a 检测等 很 å \ 考Q s Ž ö 空p 据„ 迁û f ` î 题

*º € † (f ô Ñ h „ f / º ‡ v œ

*º € †

1990年10 5å ú Ž q • * Ÿ 市

2009年9 考e 武I ' f ¡ 算: f 院 并Ž 2013年7 · 得å ff ë f M

2013年7 进e N ' f ¤ É á o 研究院; û Z ë f M ó Ê

Ñ h „ f / º ‡

[1] Wei Cao, Jian Li, Yufei Tao, Zhize Li. On Top-k Selection in Multi-Armed Bandits and Hidden Bipartite Graphs. 29th Neural Information Processing Systems (NIPS), 2015

[2] Wei Cao, Jian Li, Shimin Li, and Haitao Wang. Balanced Splitting on Weighted Intervals. Operation Research Letters (ORL), 2015.

[3] Wei Cao, Zhengwei Wu, Dong Wang, Jian Li, Haishan Wu. Automatic User Identification Method across Heterogeneous Mobility Data Souces. 32nd IEEE International Conference on Data Engineering (ICDE), 2016.

[4] Dong Wang,Wei Cao, Mengwen Xu and Jian Li. ETCPS: An Effective and Scalable Traffic Condition Prediction System. 21st International Conference on Database Systems for Advanced Applications (DASFAA), 2016.

[5] Wei Cao, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong and Wei Zhan. k-Regret Minimizing Set: Efficient Algorithms and Hardness. 20th International Conference on Database Theory (ICDT), 2017.

[6] Dong Wang,Wei Cao, Jian Li and Jieping Ye. DeepSD: Supply-Demand Prediction for Online Car-hailing Services using Deep Neural Networks. 33rd IEEE International Conference on Data Engineering (ICDE), 2017.

[7] Dong Wang, Junbo Zhang, Wei Cao, Yu Zheng and Jian Li. When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks. 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.