基于海量时空交通数据的学习与预测

（申请清华大学工学博士学位论文）

培养单位：交叉信息研究院
学　　科：计算机科学与技术
研 究 生：王栋
指导教师：李建助理教授

二〇一七年六月

Learning and Prediction over Massive Spatio-Temporal Traffic 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

Dong Wang

Dissertation Supervisor : Assitant Professor Jian Li

June, 2017

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

（保密的论文在解密后应遵守此规定）

作者签名：______________	导师签名：______________
日　　期：______________	日　　期：______________

摘　要

随着经济的高速发展，城市人口持续增长。而城市基础设施和环境容量有限，这给人们的日常出行带来了诸多不便，尤其是在交通拥堵和打车难的问题上尤为严重。低交通效率、高能源消耗和环境污染等问题也被相继引发，这些已经成为了城市进一步发展的瓶颈。

与此同时，随着GPS嵌入设备和移动端应用被人们广泛使用，各种数据海量产生，包括车辆轨迹，用户行踪和在线叫车数据等等。这些在城市中收集到的海量数据包含了关于城市的宝贵信息。这为建立更好的智能交通系统（ITS）[1]提供了新的机会，从而减轻交通拥堵，提高人们在日常通勤中的生活质量。例如，根据交通条件的预测和给定路径的行驶时间估计，ITS可以给人们推荐更好的路线图，从而规避拥塞的道路。同时，通过预测通勤需求，ITS可以提前派出车辆来平衡供需，减少出租车空载造成的能源消耗。在本论文中，我们着重使用深度学习和机器学习算法基于时空交通数据解决ITS中的三个重要预测问题。

我们研究的第一个问题是实时预测在线叫车的供需。我们基于一个新颖的深度学习结构提出一个端到端的框架DeepSD。我们的方法只需要少量手工特征，可自动从历史订单中发现复杂的供需模式。此外，我们的框架高度灵活和可扩展，预测精确（2016年滴滴算法大赛中获得了第2名）。

我们研究的第二个问题是实时交通路况预测。我们提出了一种预测城市道路交通状况的集成交通状况预测系统(ETCPS)。我们在交通路况时间序列中观测到了两个有效的相关性，并基于观测提出了两个模型。我们通过精心集成这两个模型来取得精准预测结果，且系统容易扩展到大规模数据。

我们研究的第三个问题是对于给定路径和相应的出发时间，估计司机行驶完路径所需时间。我们提出了一个端到端框架DeepTTE，基于深度递归神经网络对整条道路建模预测。我们考虑了时空相关和一些可能影响行驶时间的因素，取得精确预测结果（2017年DataCastle出行时间预测竞赛中获得第3名）。

关键词：深度学习；机器学习；时空交通数据

Abstract

The rapid economic development enables a continuing expansion of both urban population and volume of vehicles. However, the limitation of urban infrastructure and environment capacity causes inconvenience in people's daily commutes. Such inconvenience manifests in many ways, especially in traffic congestion and difficulty in getting cabs. It leads to low traffic efficiency, high energy consumption and environmental pollution, which have become bottlenecks of urban development.

At the same time, the wide usage of GPS embedded devices and mobile apps has produced a variety of massive data (e.g., vehicle mobility, traffic patterns, and online car-hailing data). The massive data collected in urban spaces contains valuable information about a city. Analyzing such data with machine learning and deep learning methods brings new opportunities for building a better intelligent transportation system (ITS)[1] to alleviate the traffic congestion and improve the human life quality in daily commute. The ITS recommends better route plans to people to avoid the congested roads, based on the prediction of traffic conditions and travel time estimation of given path. Meanwhile, by forecasting the commute demands, the ITS can dispatch the taxis to balance the supply-demand in advance and reduce the gas consumptions of no-load taxis. In this thesis, we focus on solving three important prediction problems in the ITS using spatio-temporal traffic data.

The first problem we study in this thesis is to predict the real-time car-hailing supply-demand. This is one of the most important component of an effective scheduling system. We present an end-to-end framework called *Deep Supply-Demand (DeepSD)* using a novel deep neural network structure. Our approach can automatically discover complicated supply-demand patterns from the car-hailing service data while only requires a minimal amount hand-crafted features. Moreover, our framework is highly flexible and extendable to utilize multiple data sources. Our model has achieved competitive prediction result (No.2 among 1648 teams in Di-tech Algorithm Competition

2016).

The second problem we study in this thesis is the real-time prediction of the traffic condition. For this problem, we propose an *Ensemble based Traffic Condition Prediction System (ETCPS)* for predicting the traffic conditions of any roads in a city, based on the current and historical GPS data collected from floating vehicles. We observe two useful correlations in the traffic condition time series, and propose two different models called *Predictive Regression Tree (PR-Tree)* and *Spatial Temporal Probabilistic Graphical Model (STPGM)* based on the two observations. Our best quality prediction is achieved by a careful ensemble of the two models. Our system provides high-quality prediction and can easily scale to very large datasets.

The third problem we study in this thesis is estimating the travel time estimation for the given path. We propose an end-to-end framework for Travel Time Estimation, called *DeepTTE*. Our model estimates the travel time of the whole path directly, based on deep recurrent neural networks. In our model, we consider the spatial and temporal dependency in the path as well as various factors which may affect the travel time such as the driver's habit, the day of the week, etc. Our model has achieved competitive prediction result (No.3 among 1578 teams in Travel Time Estimation Competition in DataCastle 2017).

Key words: Deep Learning; Machine Learning; Spatio-temporal traffic data

目 录

第 1 章 Introduction 1
1.1 Background and Challenges 1
1.2 Supply Demand Prediction for Online Car-hailing Services 4
1.3 Traffic Condition Prediction 5
1.4 Travel Time Estimation 5
1.5 Organization 6
第 2 章 Preliminary 7
2.1 Tree-based Methods 7
2.1.1 Decision Tree 7
2.1.2 Random Forest 8
2.1.3 Gradient Boosting 8
2.2 Deep Learning 9
2.2.1 Basic Structure 9
2.2.2 Embedding Method 11
2.2.3 Recurrent Neural Network (RNN) 13
2.2.3.1 Long Short-Term Memory (LSTM) 13
2.2.4 Residual Network 14
第 3 章 Related work 16
3.1 Supply Demand Prediction for Online Car-hailing Services 16
3.1.1 Taxi Route Recommendation 16
3.1.2 Taxi Demand Prediction 16
3.2 Traffic Condition Prediction 17
3.3 Travel Time Estimation 19
3.3.1 Road Segment-Based Travel Time Estimation 19
3.3.2 Path-Based Travel Time Estimation 19
3.4 Deep Learning in Spatio-temporal Data 20

第 4 章 Supply-Demand Prediction for Online Car-hailing Services using Deep Neural Networks 22
4.1 Introduction 22
4.2 Formulation and Overview 27
4.3 Basic Version 28
4.3.1 Identity Part 29
4.3.2 Order Part 30
4.3.3 Environment Part 32
4.3.4 Block Connections 33
4.4 Advanced Version 34
4.4.1 Extended supply-demand block 34
4.4.1.1 First Stage 35
4.4.1.2 Second Stage 36
4.4.2 Last Call Block and Waiting Time Block 38
4.4.3 Extendability 40
4.5 Experiments 41
4.5.1 Data Description 41
4.5.1.1 Error Metrics 42
4.5.2 Model Details 43
4.5.2.1 Embedding 43
4.5.2.2 Activation Function 43
4.5.2.3 Optimization Method 43
4.5.2.4 Platform 44
4.5.3 Performance Comparison 44
4.5.4 Effects of Embedding 47
4.5.5 Effects of Environment Part 49
4.5.6 Effects of Residual 50
4.5.7 Combining Weights of Different Weekdays 50
4.5.8 Extendability 51
4.6 Conclusion 51

第 5 章 Traffic Condition Prediction System 53
5.1 Introduction 53
5.2 Problem Statement 55
5.2.1 Road Network 55
5.2.2 Traffic Condition 56
5.3 Useful Observations 58
5.4 System Overview 59
5.5 Predicting The Traffic Condition with PR-Tree 60
5.5.1 Description of PR-Tree 60
5.5.2 Training PR-Trees 61
5.6 Predicting Traffic Condition with STPGM 64
5.6.1 States of STPGM 64
5.6.2 Parameter Learning 65
5.6.3 Prediction 66
5.7 Model Extensions 67
5.8 Experiments 67
5.8.1 Experiment Setting 68
5.8.2 Pre-processing 68
5.8.3 Performance Evaluation 69
5.9 Conclusion 72
第 6 章 Estimating Travel Time Based on Recurrent Neural Networks 74
6.1 Introduction 74
6.2 Problem Definition 77
6.3 Model Architecture 78
6.3.1 Attribute Component 79
6.3.2 Sequence Learning Component 80
6.3.2.1 Sampling Method 80
6.3.2.2 Pooling Method 81
6.3.2.3 Discussion 82
6.3.3 Residual Component 82
6.3.4 Auxiliary Component 83
6.3.5 Model Training 84
6.3.5.1 Discussion 84

6.4 Experiments 85
6.4.1 Experiment Setting 85
6.4.1.1 Data Description 85
6.4.1.2 Parameter Setting 85
6.4.1.3 Experiment Environment 86
6.4.2 Performance Comparison 86
6.4.3 Effects on the Sampling Rate 88
6.4.4 Effects of the Auxiliary Component 88
6.4.5 Effects of the Attribute Component 89
6.5 Conclusion 89
第 7 章 Conclusion 91
7.1 Summary of the Thesis 91
7.2 Future Directions 94
参考文献 95
致　谢 102
声　明 103
附录 A 中文摘要 104
A.1 引言 104
A.1.1 研究背景及挑战 104
A.1.2 在线叫车服务中的供需预测 106
A.1.3 交通路况预测 107
A.1.4 出行时间预估 107
A.1.5 论文架构 108
A.2 相关工作 108
A.2.1 在线约车平台的供需预测 108
A.2.1.1 出租车路径推荐 108
A.2.1.2 出租车需求预测 109
A.2.2 交通路况预测 109
A.2.3 行驶时间估计 110
A.2.3.1 基于路段的行驶时间估计 110
A.2.3.2 基于路径的行驶时间估计 111
A.2.4 时空数据上的深度学习 111

A.3 基于深度神经网络的在线约车平台供需预测 112
A.3.1 引言 112
A.3.2 公式化和概述 116
A.3.3 基础框架 117
A.4 道路路况预测系统 118
A.4.1 引言 118
A.4.2 系统概述 120
A.5 基于循环神经网络的出行时间预测 121
A.5.1 引言 121
A.5.2 问题定义 123
A.5.3 模型架构 125
A.6 结论 125
A.6.1 本文总结 125
A.6.2 未来方向 127
个人简历、在学期间发表的学术论文与研究成果 128

第 1 章 Introduction

1.1 Background and Challenges

Nowadays, the rapid expansion of many large cities has lead to the population explosion. As a consequence, the commute demands in these cities increase sharply. People suffer the traffic congestion and the difficulty in getting cabs. According to the 2015 Urban Mobility Scorecard from the Texas Transportation Institute[2], traffic congestion problem caused urban Americans to travel an extra 6.9 billion hours and purchase an extra 3.1 billion gallons of fuel for a congestion cost of $160 billion, for the 471 urban areas of the USA in 2014.

Meanwhile, the location based services (LBS) and GPS embedded devices become ubiquitous. Such GPS embedded devices and mobile apps affect people's daily life profoundly. For example, people use their smart phones to plan their routes, call for car-hailing services, find the trip partners, and search the destinations, etc. Large volume of location based data is generated by these devices and apps routinely, including the online car-hailing orders, GPS trajectories, map queries, and geo-tagged check-in data etc.

Analyzing such spatio-temporal traffic data with machine learning and deep learning methods brings new opportunities for building a better intelligent transportation system (ITS)[1] to alleviate the traffic congestion and improve the human life quality in daily commute. The ITS recommends better route plans to people to avoid the congested roads, based on the prediction of traffic conditions and travel time estimation of given path. Meanwhile, by forecasting the commute demands, the ITS can dispatch the taxis to balance the supply-demand in advance and reduce the gas consumptions of no-load taxis. In this thesis, we focus on solving three important prediction problems in the ITS using spatio-temporal traffic data.

- **Online Car-hailing Supply-Demand Prediction.** Online car-hailing apps/platforms (such as Uber, Didi, and Lyft) have emerged as a novel and popular means to provide on-demand transportation service via mobile apps. By incentivizing private cars owners to provide car-hailing services, it enlarges the transportation capacities of the cities. As more passengers and more drivers use the service, it becomes increasingly important to predict the supply-demand of online car-hailing services, based on which the scheduling system can dispatch the drivers in advance to minimize the waiting time of passengers and maximize the driver utilization.

- **Traffic Condition Prediction.** It has been shown by many studies[3–6] that a well performed traffic condition prediction system plays an essential role in improving the traffic efficiency. For example, the governments can use it as a reference, when they make decisions about changes to traffic regulations (e.g., change a normal lane to a bus lane), or constructions of additional roads (e.g., add extra lanes); it can also give suggestions to the civil engineers when they plan for construction zones (e.g., how a short-term construction would impact traffic)[7].
- **Travel Time Estimation.** Estimating the travel time for a given path is a fundamental problem in route planning, navigation, and traffic dispatching. An accurate estimation of travel time helps people better planning their routes and avoiding congested roads, which in turn helps to alleviate traffic congestion. Almost all the electronic maps and online car-hailing services provide the travel time estimation in their apps, such as Google Map, Uber, Didi, etc.

Although such problems are widely studied[3–6], there still exist a large number of challenges due to the massiveness and irregularity of the corresponding traffic data. To be more concrete, we first illustrate the challenges of learning and prediction over such data.

- The traffic data usually contains both spatial (locations) and temporal (timestamps) attributes, which we also refer to as *spatio-temporal data*. Such spatio-temporal data contains spatial and temporal correlation patterns at the same time. For example, the traffic condition of a road is affected by its previous conditions as well as the conditions of its adjacent roads. To capture the spatial correlation and temporal correlation pattern at the same time is not easy.

- The patterns in spatio-temporal data always vary dynamically due to different geographic locations and time intervals. For example, in the morning the car-hailing demand tends to surge in the residential areas whereas in the evening the demand usually tends to surge in the business areas. Furthermore, the supply-demand patterns under different days of a week can be extremely different. Prior work usually builds several sub-models for different days of the week[8–11]. Such implementations, on one hand, make the model tedious, on the other hand, each sub-model only utilizes a small part of data which may suffer from the lack of training data.
- Moreover, the spatial temporal data usually contains multiple attributes. For example, in the online car-hailing supply-demand problem, order data contains attributes such as the timestamp, passenger ID, start location, destination etc, as well as several "environment" factors, such as the traffic condition, weather condition etc. These attributes together provide a wealth of information for prediction. However, it is nontrivial how to use all the attributes in a unified model. Currently, the most standard approach is to come up with many "hand-crafted" features (i.e., feature engineering), and fit them into an off-the-shelf learning algorithm such as logistic regression or random forest. However, feature engineering typically requires substantial human efforts and there is little general principle how this should be done.
- Finally, the spatial temporal data is usually quite massive. For example, in this thesis, the trajectory data we use in the experiment part of the travel time estimation problem (Chapter A.5) contains 9,653,822 trajectories and 1.4 billion GPS records in total. To deal with such massive data efficiently, we usually need to take advantages of big data platforms, such as Hadoop, Spark. Moreover, compared with the traditional machine learning methods, deep learning techniques show great potential for mining massive data. However, to the best of our knowledge, there is no standard deep learning model to deal with such massive, noisy, and multi-attribute spatio-temporal data we mentioned above.

In this thesis, we focus on solving three important prediction problems in the ITS using carefully devised machine learning and deep learning models to address the above challenges. We present the concrete problems and our contributions in the following sections.

1.2 Supply Demand Prediction for Online Car-hailing Services

In Chapter A.3, we study the problem of predicting the real-time car-hailing supply-demand, which is one of the most important component of an effective scheduling system. Our objective is to predict the gap between the car-hailing supply and demand in a certain area in the next few minutes. Based on the prediction, it is possible to balance the supply-demands in advance by dispatching the cars and dynamically adjusting the price. After observing the data, we find that the car-hailing supply-demand varies dynamically due to different geographic locations and time intervals. Furthermore, the supply-demand patterns under different days of a week can be extremely different. It is difficult to predict such heterogeneous data.

Contributions: We present an end-to-end framework called *Deep Supply-Demand (DeepSD)* using a novel deep neural network structure. Our approach can automatically discover complicated supply-demand patterns from the car-hailing service data while only requires a minimal amount hand-crafted features. Moreover, our framework is highly flexible and scalable. Based on our framework, it is very easy to utilize multiple data sources (e.g., car-hailing orders, weather and traffic data) to achieve a high accuracy. We conduct extensive experimental evaluations, which show that our framework provides more accurate prediction results than the existing methods. Our experimental results also show that embedding method can “cluster” the areas with similar supply-demand patterns, and enable different areas to share historical records, which improves data utilization and prediction accuracy.

1.3 Traffic Condition Prediction

In Chapter A.4, we study the problem of predicting the traffic condition of each road after a few minutes, when the current and historical traffic conditions of the road network are given. Ubiquitous location based services enable us to collect a large volume of traffic data from GPS-embedded devices. Our prediction is based on such GPS data. Despite there exist several researches and products for traffic prediction based on the GPS data, most of them only focused on the arterial roads and did not consider the urban roads. After observing the data, we find that by transforming the traffic condition time series into two different forms of time series (expectation-reality gap and first order difference of traffic condition series), the new time series reveal very strong autocorrelations. We hope these observations can provide useful insight in further study of the travel condition prediction problem and related problems.

Contributions: We propose an *Ensemble based Traffic Condition Prediction System (ETCPS)* for predicting the traffic conditions of any roads in a city based on the current and historical GPS data collected from floating vehicles. We have observed two useful correlations in the traffic condition time series, which are the bases of our design. In order to exploit these two correlations for prediction, we propose two different models called *Predictive Regression Tree (PR-Tree)* and *Spatial Temporal Probabilistic Graphical Model (STPGM)*. Our best quality prediction is achieved by a careful ensemble of the two models. Our system provides high-quality prediction and can easily scale to very large datasets.

1.4 Travel Time Estimation

In Chapter A.5, we study the problem of travel time estimation for a given path, driver and start time. Although the problem has been widely studied in the past years, providing an accurate travel time is still a challenging problem. Prior work usually focuses on estimating the travel times of individual road or sub-paths, and then summing up these estimated travel times. However, such approach leads to an inaccurate estimation, since the travel time is not only affected by the traffic condition, but also affected by the number of road intersections or traffic lights in the path, and the estimation errors for individual road may accumulate. Furthermore, the travel time of a given path for a specific driver is also affected by the driving style of the driver.

Contributions: We propose an end-to-end framework for Travel Time Estimation called *DeepTTE*. Our model estimates the travel time of the whole path directly, based on deep recurrent neural networks, which can easily process variable-length GPS trajecto. In our model, we consider the spatial and temporal dependency in the path as well as various factors which may affect the travel time such as the driver's habit, the day of the week etc. We conduct extensive experiment on a large scale dataset. The experiment result shows that our model significantly outperforms the other existing methods. To the best of our knowledge, this is the first time that the trajectory is taken as a whole to estimate the travel time for arbitrary origins and destinations.

1.5 Organization

The rest of the thesis is organized as follows: In Chapter 1, we provide an introduction to this research work giving the background, challenges and the main contributions of this thesis. In Chapter 2, we provide some background on the machine learning and deep learning methods used in the rest of this thesis. We cover topics such as tree-based models (including decision tree, random forest and gradient boosting), and deep learning (including basic structure, embedding method, recurrent neural network, and residual network). In Chapter A.2, we present a brief overview of the related work. Applications of deep learning techniques in spatio-temporal data are also included. In Chapter A.3, we present an end-to-end framework called DeepSD using a novel deep neural network structure to predict the supply-demand for online car-hailing services. In Chapter A.4, we introduce an Ensemble based Traffic Condition Prediction System (ETCPS) for predicting the traffic conditions of any roads in a city. In Chapter A.5, we present an end-to-end framework for travel time estimation called DeepTTE based on deep recurrent neural networks. Our conclusion and future courses of action are followed in Chapter A.6.

第 2 章 Preliminary

In this chapter, we review some preliminary knowledge that is necessary for latter chapters. We introduce some useful models that we use in the thesis, including the tree based models, and deep learning models.

2.1 Tree-based Methods

Tree-based methods are widely used in spatio-temporal data analysis. As we mentioned in the Chapter 1, the spatio-temporal data is usually highly noisy and irregular. The tree-based methods such as random forest and gradient boosting can be easily used to deal with such non-structured data and obtain a reasonable performance. In this section, we briefly introduce the decision tree method, random forest and gradient boosting.

2.1.1 Decision Tree

The decision tree method is a non-parametric supervised learning method used for classification and regression[12]. The goal of decision tree method is to build a model that can determine the best decisions. It predicts the value (class) of a target variable (class) by learning simple decision rules inferred from the data features.

To be more concrete, we assume the data is associated with n features. Each time, we select one feature and partition data into small chunks according to the value of the feature based on the certain criterions. Each chunk represents a node. The feature and split-point are chosen to achieve the best fit. We continue partition these chunks into smaller chunks, until some stopping rule is applied. The common criterions are Gini Index (CART method)[13], Information divergence (ID3 method[12]), Information Gain Ratio (C4.5 method[14]).

2.1.2 Random Forest

Random forest[15] is an ensemble of classification or regression trees. These trees are generated through changing the training set and feature set using the same strategy as bagging. The new training sets are created by re-sampling n times from the original data set. The new feature sets are created by re-sampling m times from the original feature set. Prediction is made by aggregating (majority vote for classification or averaging for regression) the predictions of the ensemble. Random forest generally exhibits a substantial performance improvement over the single tree classifier such as CART and C4.5.

Random forest increases the classification or regression accuracy by decreasing the variance of the classification or regression errors. In another word, it taps on the instability of a classifier or regression. "Instability" of a classifier or regression means that a small change in the training samples may result in comparatively great changes in accuracy[13].

2.1.3 Gradient Boosting

Gradient boosting[16–18] is also an ensemble of classification or regression trees. For a given data set, it builds M models. Each model may be very simple (like decision tree), which we call it weak learner. As a kind of boosting methods, gradient boosting builds the model in a forward stage-wise manner, which is quite different from bagging methods (such as random forest). In each stage, it introduces a weak learner to compensate the "shortcomings" of existing weak learners.

To be more concrete, we define the training data as (x_i, y_i), where $i \in [1, N]$. The loss in using $F(x)$ to predict y on the training data is defined as $L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$ where $L(f)$ is a derivable loss function. The goal is to minimize $L(f)$ with respect to f, where here $f(x)$ is constrained to be a sum of trees. Forward stage-wise boosting is a greedy strategy. In each stage $1 \leq m \leq M$ of gradient boosting, we assume that there is some imperfect model f_m. The gradient boosting constructs a new weak learner T_m that can maximally reduces the loss $L(f) = \sum_{i=1}^{N} L(y_i, f_m(x_i) + T(x_i))$. Then, the new model is defined as $f_{m+1}(x) = f_m(x) + T_m(x)$. To construct the weak learner $T_m(x)$, we calculate the negative gradient vector $\partial L(y_i, f(x_i))/\partial f(x_i)$ and fit such vector to obtain $T_m(x)$[13].

2.2 Deep Learning

The deep learning method[19,20] becomes the hottest topics in machine learning area since 2006 and it has already been successfully applied in computer vision, speech recognition, natural language processing etc. The traditional machine learning methods usually require a large quantity of carefully hand-crafted features. However, feature engineering typically requires substantial human efforts and there is little general principle how this should be done. Alternatively, the deep learning methods learn to represent the data by combining the simple features into more sophisticated features with its deep architecture automatically. Thus, the deep learning methods are capable of achieving more accurate results with less human effort.

Currently, most of deep learning architectures focus on the image[21,22], text[23,24] and speech data[25–27]. Compared with such data, the spatio-temporal traffic data has both spatial and temporal patterns and it usually contains multiple attributes. Moreover, we usually have to handle city-level data which is in a very large scale. Due to the characteristic of spatio-temporal traffic data, few work utilizes deep learning methods in spatio-temporal traffic data analysis and prediction. We carefully devise several deep architectures to solve some classical problems using such data (Chapter A.3, A.5)[28]. Our results demonstrate the strength of deep learning in prediction over massive spatio-temporal traffic data. In this section, we first illustrate the basic structure in deep learning and then continue by introducing several useful deep learning architectures in spatio-temporal traffic data learning and prediction.

2.2.1 Basic Structure

A deep learning model is a neural network which consists of many *layers* of *non-linear* information processing stages and *hierarchical architectures*[20,29,30]. Such structure benefits from joint learning of representations with increased levels of ab-

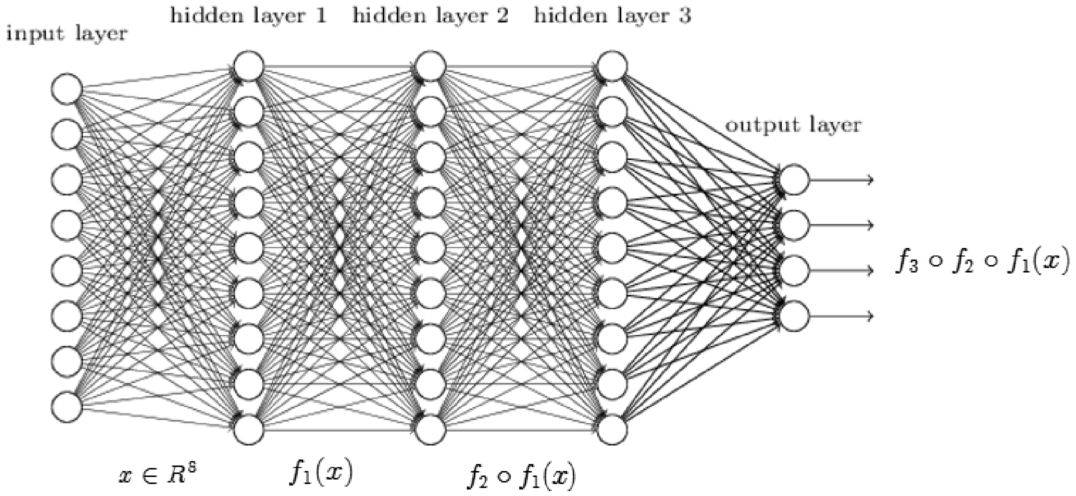


Figure 2.1 Illustration of a neural network (picture from[32])

straction and classification/regression. Formally, a layer can be viewed as a non-linear mapping $f(.)$. For the input data x, the layer maps it into $f(x;\theta)$ where θ is the parameter of this layer. We show a simple example in Fig. 2.1. We use $f_i(.;\theta_i)$ to represent the i-th layer in the model. The lower layers extract the simple features from the original data and the deeper layers combine them into more complicated features[31]. The final output can be represented as the compound function $\hat{y} = f_1 \circ \ldots \circ f_n(x)$. The loss function is denoted as $E(y,\hat{y})$, where y is the label of input x.

We stress that all the parameters are initialized randomly. To train the model, we first calculate the loss function E and update the parameters in the last layer θ_n with corresponding gradient $\frac{\partial E}{\partial \theta_n}$. We then calculate the gradient of θ_{n-1} by the chain rule

$$\frac{\partial E}{\partial \theta_{n-1}} = \frac{\partial E}{\partial \theta_n} \cdot \frac{\partial \theta_n}{\partial f_{n-1}} \cdot \frac{\partial f_{n-1}}{\partial \theta_{n-1}}$$

and update the corresponding parameters accordingly. We call such method *back-propagation*. Moreover, to enhance the learning ability of deep learning model-s , several improved architectures were devised, such as Convolutional Neural Network (CNN)[33,33], Recurrent Neural Network (RNN)[34] and Residual Neural Network (ResNet)[35]. We introduce three useful architectures in spatio-temporal traffic data analysis in Section 2.2.2 to Section 2.2.4.

2.2.2 Embedding Method

Embedding method is a feature learning technique[36,37]. It is widely used in deep learning models, especially in natural language processing (NLP) tasks[36,38,39]. It is a parameterized function which maps the categorical values to the real numbers.

Specifically, neural networks treat every input as a real value. A simple way to transform categorical values to real numbers is *one-hot representation*[40]. For example, suppose the value of a categorical feature is 3 and the corresponding vocabulary size (highest possible value) is 5. Then, its one-hot representation is $(0, 0, 1, 0, 0)$. However, using such representation can be computationally expensive when the vocabulary size is huge. Moreover, such representation does not capture the similarity between different categories.

The embedding method overcomes such issues by mapping each categorical value into a low-dimensional space (relative to the vocabulary size). For example, the categorical value with one-hot representation equal to $(0, 0, 1, 0, 0)$ can be represented as the form of $(0.2, 1.4, 0.5)$. Formally, for each categorical feature, we build an embedding layer with parameter matrix $W \in R^{I \times O}$. Here I is the vocabulary size of input categorical value and O is the dimension of the output space (which we refer to as the embedding space). For a specific categorical value $i \in [I]$, we use $\mathsf{onehot}(i) \in R^{1 \times I}$ to denote its one-hot representation. Then, its embedded vector $\mathsf{embed}(i) \in R^{1 \times O}$ is equal to $\mathsf{onehot}(i)$ multiply the matrix W, i.e., the i-th row of matrix W. We usually have that $O \ll I$. Thus, even the vocabulary size is very large, we can still handle these categorical values efficiently.

Furthermore, an important property of embedding method is that the categorical values with similar semantic meaning are usually very close in the embedding space[41]. Here we give an example of word embedding, which embeds words into real vector, $W : words \rightarrow \mathbb{R}^n$. As shown in the figure 2.2, we visualize the embedded words with t-SNE, a sophisticated technique for visualizing high-dimensional data. The words with similar semantic meaning are very close, such as the words representing numbers are embedded in the left, and the words representing jobs are embedded in the right region. Another important property word embeddings exhibit is that analo-

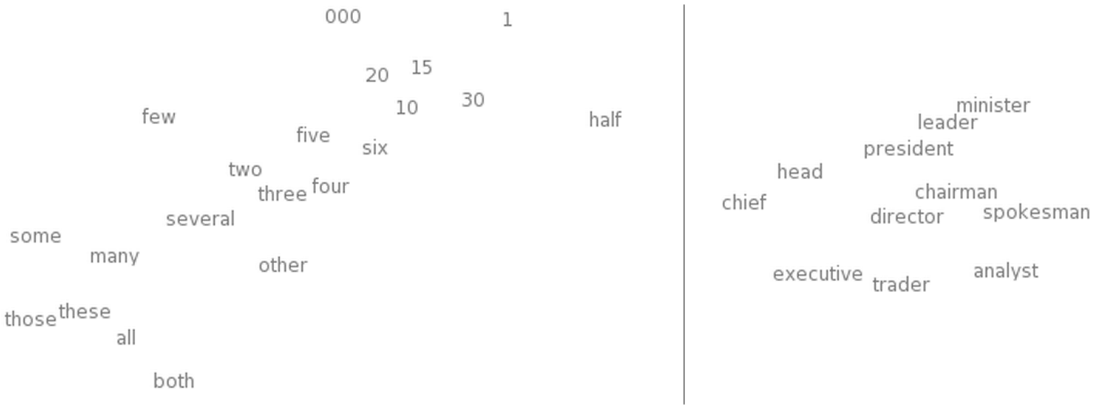


Figure 2.2 T-SNE Visualizations of Word Embeddings. (picture from[41])

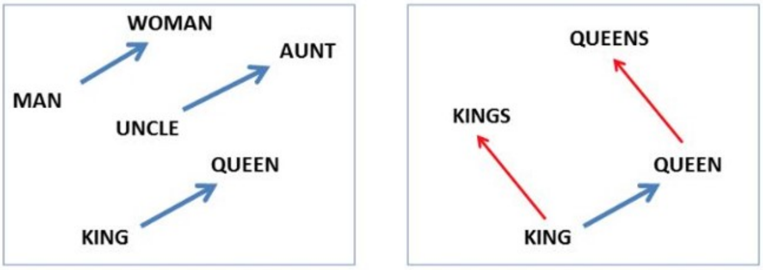


Figure 2.3 Male-female Difference Vector (picture from[42])

gies between words seem to be encoded. For example, the male-female difference vector between "man" and "woman" seems almost the same as that between "king" and "queen", as shown in Fig 2.3.

In general, these properties are considered as side effects. The neural network does not explicitly map the categorical values with similar semantic meanings to the a similar position in the vector space. These properties more or less popped out of the optimization process. It seems that neural networks can learn better ways to represent data, automatically.

In our problem of supply demand prediction for online car-hailing services in Chapter A.3, we find that if two different areas share similar supply-demand patterns, then their area IDs are close in the embedding space. See Section 4.5 for the details. We stress that the parameter matrix W in the embedding layer is optimized with other parameters in the network. We do not train the Embedding Layers separately.

2.2.3 Recurrent Neural Network (RNN)

Recurrent neural network (RNN) is an artificial neural network which contains an internal state and a directed cycle between units[34]. It is suitable for capturing the temporal dependency and has been used successfully in sequential learning such as the natural language processing, speech recognition, etc[38,43,44]. Especially, an internal state can be viewed as the "memory" of previous time steps. When RNN calculates a new internal state, it captures the temporal dependency with all the previous input sequences. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. Formally, RNN takes a sequence $\{x_t\}$ as input. In each iteration, RNN calculate the "hidden state" (memory)

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

where $W_x$fiW_h is the parameters to be learned.

However, vanilla RNN failed in processing long sequences due to vanishing gradient and exploding gradient problems[45]. To overcome such issue, Long Short-Term Memory was developed[45].

2.2.3.1 Long Short-Term Memory (LSTM)

Long Short Term Memory network (LSTM) is a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber[45], and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

An LSTM contains several LSTM units. Each LSTM unit (See Fig. 2.4 for an illustration) contains a memory cell and three gates which are used to control the flow of information in/out of their memory. Mathematically, given the input vector $x = \{x_0, x_1, \ldots, x_n\}$, and denoting the output as $y = \{y_0, y_1, \ldots, y_n\}$, the expected output (the internal states of LSTM) are updated as follows:

$$i_t = \sigma(W_{xi}x_t + W_{yi}y_{t-1} + W_{ci}c_{t-1} + b_i),$$

$$f_t = \sigma(W_{xf}x_t + W_{yf}y_{t-1} + W_{cf}c_{t-1} + b_f),$$

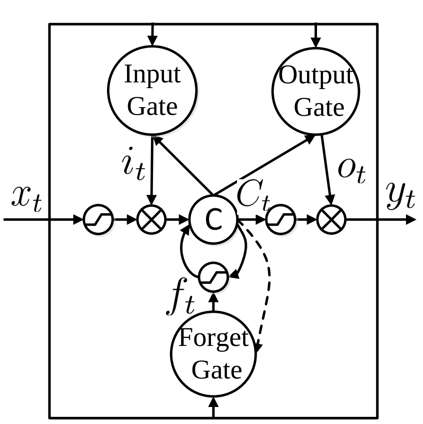


Figure 2.4 LSTM

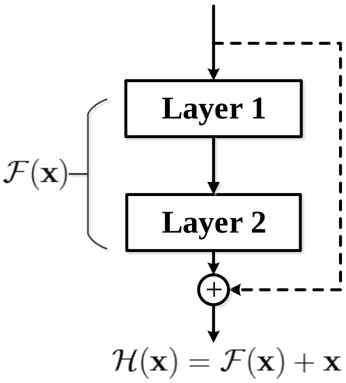

Layer 1
Layer 2
$\mathcal{F}(\mathbf{x})$
$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$

Figure 2.5 Residual Network

$$
\begin{aligned}
c_t &= f_t \otimes c_{t-1} + i_t \otimes tanh(W_{xc}x_t + W_{yc}y_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{yo}y_{t-1} + W_{co}c_{t-1} + b_o), \\
y_t &= o_t \otimes tanh(c_t)
\end{aligned}
$$

where σ denotes the logistic sigmoid function and $\otimes$ denotes element-wise multiplication.

2.2.4 Residual Network

Many non-trivial tasks have greatly benefited from very deep neural networks, which reveals that network depth is of crucial importance[46–48]. However, an obstacle to train a very deep model is the gradient *vanishing/exploding* problem, i.e., the gradient vanishes or explodes after passing through several layers during the backpropagation[20,49]. To overcome such issue in deep neural networks, He et al.[35] proposed a new network architecture called the residual network (ResNet). ResNet makes it possible (easier) to train a very deep convolutional neural network successfully.

The residual learning adds the *shortcut connections* (dashed line in Fig. 2.5) and *direct connections* (solid line in Fig. 2.5) between different layers. Thus, the input vector can be directly passed to the following layers though the shortcut connections. For example, in Fig. 2.5, we use $\mathbf{x}$ to denote the input vector and $\mathcal{H}(\mathbf{x})$ to denote the desired mapping after two stacked layers. In the residual network, instead of learning the mapping function $\mathcal{H}(\mathbf{x})$ directly, we learn the residual mapping $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$ and broadcast $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ to the following layers. It has been shown that optimizing the residual mapping is much easier than optimizing the original mapping[35], which is the key to the success of deep residual network.

第 3 章 Related work

In this chapter, we review the existing related work for the three problems that we study in this thesis respectively. There is a large body of literature on learning and prediction over massive spatio-temporal data and we only mention a few closely related ones. At the end of this chapter, we review the existing work which study the prediction with spatio-temporal data using deep learning.

3.1 Supply Demand Prediction for Online Car-hailing Services

3.1.1 Taxi Route Recommendation

The taxi route recommendation aims to predict the best routes for drivers in order to maximize their utilization. Yuan et al.[9] presented an algorithm to suggest the taxi drivers with locations towards which he/she is most likely to pick up a passenger soon. They used a Poisson model to predict the probability of picking up a passenger for each parking place. In their work, the pick-up locations are fixed in advance. Our work aims to predict the supply-demand gap in every area. Wang et al.[50] investigated the problem of recommending a cluster of roads to the taxi drivers. They used a single hidden layer neural network with carefully selected hand-crafted features. Our work uses a deep neural network with little hand-crafted features. Ge et al.[51] provided a cost-efficient route recommendation algorithm which can recommend a sequence of pick-up locations. They learnt the knowledge from the historical data of the most successful drivers to improve the taxi driver utilization of remaining ones. However, such problem setting is much different from ours.

3.1.2 Taxi Demand Prediction

The taxi demand prediction studies the problem of forecasting the demands in every pick up location. Moreira-Matias et al.[52] combined the Poisson Model and AutoRegressive Moving Average (ARMA) model to predict the demand in each taxi stand. Again, they only considered the demands in several fixed locations. Moreover, in their work they treated the data in each taxi stand separately. Such implementation suffers from the lack of training data. In a recent work, Chiang et al.[10] proposed a generative model, called Grid-based Gaussian Mixture Model, for modeling spatio-temporal taxi bookings. Their approach was able to predict the demand of taxis in any time interval for each area in the city. Nevertheless, on one hand, they treated the orders in weekdays and weekends separately. On the other hand, in their approach, the total amount of taxi bookings was decided by a Poisson model in advance. When the real-time taxi demand changed rapidly, their approach may lead to a large prediction error.

We stress that prior work only studied the demand prediction but ignored the supply. In the real applications such as taxi route recommendation, taxi dispatching etc, it is important to predict the equilibrium of the supply-demand. Moreover, none of these work studied incorporating the environment data such as the weather or traffic conditions to enhance the prediction accuracy. In Di-tech Prediction Competition 2016①, the champion team proposed an accurate algorithm based on Gradient Boosting Descent algorithm ② and 1534 carefully hand-crafted features. Besides the basic features (such as the area ID, day of week, the previous gap and corresponding statistics), they also considered very detailed features such as average/standard deviation of waiting time, calling time of different passengers during different time intervals, the ratio of car-hailing supply/demand of different areas etc. Despite the proposed model achieves a remarkable performance, designing such features are highly non-trivial and requires enormous amount of human effort.

3.2 Traffic Condition Prediction

Most of prior works use the probabilistic models to predict the traffic conditions. Hunter et al.[53] formulated the traffic condition prediction in the arterial network to a maximum likelihood problem and estimated the travel time distributions based on the

---

① http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016&&locale=en

② https://github.com/Microsoft/LightGBM/

observed route travel times. Yeon et al.[54] estimated traffic conditions on a freeway using Discrete Time Markov Chains (DTMC). However these works assumed that the travel times on different road segments are independent without considering the correlation between the traffic conditions on different roads which may lead to incorrect prediction in the urban area[55].

To capture the correlations between road segments, Hofleitner et al.[56] formulated the transitions between states among adjacent road segments as a dynamic Bayesian network model and predicted the traffic conditions by an EM approach. However, it did not consider the efficiency on the large scale data.Yuan et al.[9] built a landmark graph based on the trajectories of taxis, where each node (entitled a landmark) indicates a road segment each edge indicates the aggregation of taxis' commutes between two landmarks. They formulated the correlations and estimated the edge travel time distributions based on the landmark graph. However, as the landmarks are selected from the top-*k* frequently traversed road segments, many of road segments with sparse records can not be predicted.

The most related work with our model was proposed by Yang et al.[57]. They proposed an algorithm called STHMM which is a spatio temporal hidden markov model. They further presented an effective method to deal with the sparsity in the data. However, they did not consider the heterogeneity of transition patterns in different time intervals. In our experiment section (Section 5.8), we show that our model outperform STHMM in both the efficiency and accuracy. We stress that Chu et al.[58] considered the transition patterns in different time intervals and proposed a time-vary dynamic network. However their goal is to reveal the causal structure in a ring road system which differs from ours.

Furthermore, we stress two recent related works[5,59]. Wang et al.[59] presented an efficient algorithm to estimate the travel time of any path, based on sparse trajectories generated by taxi in recent time slots and in history, by using the tensor decomposition. Instead of predicting the traffic conditions, they studied the estimation of travel time for given travel paths in the current time slot. Asghari et al.[5] estimated the travel time distributions based on the historical sensor data. As their work studied the algorithm to find the most reliable route for the travel planning, it has a related but different scope.

3.3 Travel Time Estimation

3.3.1 Road Segment-Based Travel Time Estimation

Estimating travel time has been studied extensively[60–62]. However, these works estimated the travel time of individual road segment without considering the correlations between the roads. Yuan et al.[57] used a spatial-temporal Hidden Markov Model to formalize the relationships among the adjacent roads. Wang et al.[63] improved this work through an ensemble model based on two observed useful correlations in the traffic condition time series. Wang et al.[64] 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. 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[65].

3.3.2 Path-Based Travel Time Estimation

Rahmani et al.[66] 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.[9] 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.[59] 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. Dai et al.[67] proposed a new paradigm for path cost distribution estimation. Given a departure time and a query path, they showed how to select an optimal set of weights with associated sub-trajectories that cover the query path and compute the cost distribution of the query path using the joint distribution. As they focused on estimating the uncertainty of travel cost, it has a related but different scope. In a very recent travel time estimation competition ①, the champion team used a series of standard machine learning models such as the random forest, the multi-layer perceptron, the LASSO etc., as the base estimators. They use Gradient Boosting method to combine the estimation results of different estimators and use the combined result as the final result. However, we stress that in practice, devising many machine learning models is very tedious and hard to maintain. Instead, we only use a single end-to-end framework.

3.4 Deep Learning in Spatio-temporal Data

Recently, the deep learning techniques demonstrate the strength on spatio-temporal data mining problems. An increasing number of researchers studied applying the deep learning technique to prediction problems[68–71]. However, few work studied the prediction with spatio-temporal data using deep learning. Lv et al.[72] studied predicting the traffic flow with deep neural networks. They adopted a *stack autoencoder* to train the network layer by layer greedily. They showed that the deep model is more accurate comparing with the baseline methods. Zhang et al.[73] designed a novel architecture called *DeepST* to predict the crowd flow. Their model learned the spatio-temporal patterns by a sequence of convolutional neural networks. They proposed improved *DeepST* to *ST-ResNet* in[74] by using residual learning to construct a much deeper networks, and proposing a parametric-matrix-based fusion mechanism for modeling both spatial and temporal dependencies. Song et al.[75] built an intelligent system called DeepTransport, for simulating the human mobility and transportation mode at a citywide level. Dong et al.[76] studied characterizing the driving style of differen-

---

① The competition information and data can be found in https://github.com/DeepTTE/DeepTTE t drivers by a stacked recurrent neural network. Ma et al.[77] proposes a CNN-based method that learns traffic as images and predicts large-scale traffic speed. To the best of our knowledge, applying the deep learning technique to enhance car-hailing supply-demand prediction accuracy has not been studied so far, and no prior work studies estimating the travel time of the whole path based on the deep learning approach.

第 4 章 Supply-Demand Prediction for Online Car-hailing Services using Deep Neural Networks

The online car-hailing service has gained great popularity all over the world. As more passengers and more drivers use the service, it becomes increasingly more important for the the car-hailing service providers to effectively schedule the drivers to minimize the waiting time of passengers and maximize the driver utilization, thus to improve the overall user experience. In this chapter, we study the problem of predicting the real-time car-hailing supply-demand, which is one of the most important component of an effective scheduling system. Our objective is to predict the gap between the car-hailing supply and demand in a certain area in the next few minutes. Based on the prediction, we can balance the supply-demands by scheduling the drivers in advance. We present an end-to-end framework called *Deep Supply-Demand (DeepSD)* using a novel deep neural network structure. Our approach can automatically discover complicated supply-demand patterns from the car-hailing service data while only requires a minimal amount hand-crafted features. Moreover, our framework is highly flexible and extendable. Based on our framework, it is very easy to utilize multiple data sources (e.g., car-hailing orders, weather and traffic data) to achieve a high accuracy. We conduct extensive experimental evaluations, which show that our framework provides more accurate prediction results than the existing methods. ①

4.1 Introduction

Online car-hailing apps/platforms have emerged as a novel and popular means to provide on-demand transportation service via mobile apps. To hire a vehicle, a passenger simply types in her/his desired pick up location and destination in the app and sends the request to the service provider, who either forwards the request to some drivers close to the pick up location, or directly schedule a close-by driver to take the

---

① This work has been published in ICDE 2017[28].

order. Compared with the traditional transportation such as the subways and buses, the online car-hailing service is much more convenient and flexible for the passengers. Furthermore, by incentivizing private cars owners to provide car-hailing services, it promotes the sharing economy and enlarges the transportation capacities of the cities. Several car-hailing mobile apps have gained great popularities all over the world, such as Uber, Didi, and Lyft. Large number of passengers are served and volume of car-hailing orders are generated routinely every day. For example, Didi, the largest online car-hailing service provider in China, handles around 11 million orders per day all over China. ①

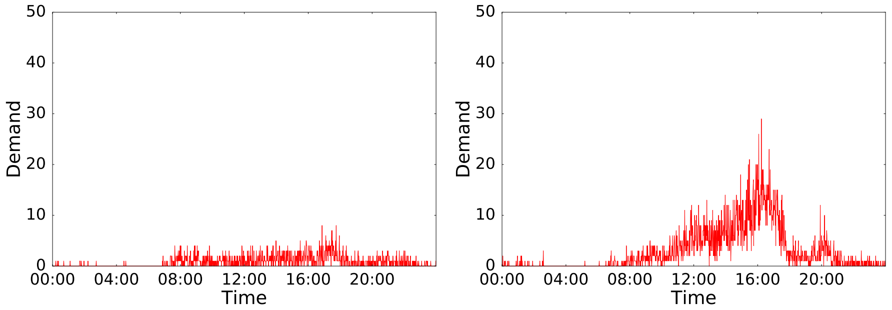
As a large number of drivers and passengers use the service, several issues arise: Sometimes, some drivers experience a hard time to get any request since few people nearby call the rides; At the same time, it is very difficult for some passengers to get the ride, in bad weather or rush hours, because the demand in the surrounding areas significantly exceeds the supply. Hence, it is a very important yet challenging task for the service providers to schedule the drivers in order to minimize the waiting time of passengers and maximize the driver utilization. One of the most important ingredient of an effective driver scheduler is the *supply-demand prediction*. If one could predict/estimate how many passengers need the ride service in a certain area in some future time slot and how many close-by drivers are available, it is possible to balance the supply-demands in advance by dispatching the cars, dynamically adjusting the price, or recommending popular pick-up locations to some drivers.

In this chapter, we study the problem of predicting the car-hailing supply-demand. More concretely, our goal is to predict the gap between the car-hailing supply and demand (i.e., $\max(0, \text{demand} - \text{supply})$) for a certain area in the next few minutes. Our research is conducted based on the online car-hailing order data of Didi. To motivate our approach, we first present some challenges of the problem and discuss the drawback of the current standard practice for such problem.

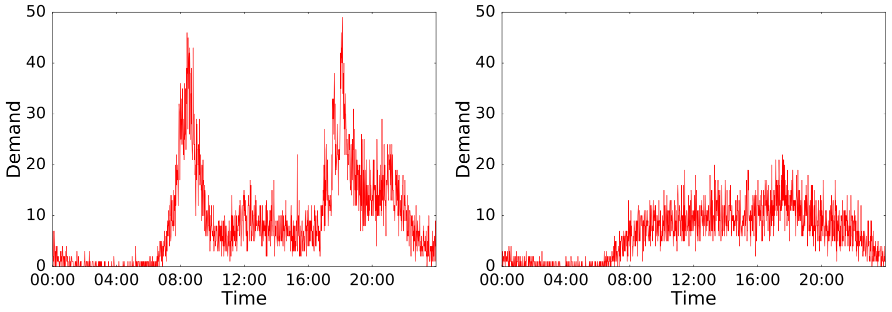
- The car-hailing supply-demand varies dynamically due to different geographic locations and time intervals. For example, in the morning the demand tends to

---

① Homepage: http://www.xiaojukeji.com/en/index.html



(a) First area on March 9th　(b) First area on March 13th



(c) Second area on March 9th　(d) Second area on March 13th

Figure 4.1　Car-hailing demands under four different situations.

surge in the residential areas whereas in the evening the demand usually tends to surge in the business areas. Furthermore, the supply-demand patterns under different days of a week can be extremely different. Prior work usually distinguishes different geographic locations, time intervals or days of week and build several sub-models respectively [8–11]. Treating the order data separately and creating many sub-models are tedious, and may suffer from the lack of training data since each sub-model is trained over a small part of data.

- The order data contains multiple attributes such as the timestamp, passenger ID, start location, destination etc, as well as several "environment" factors, such as the traffic condition, weather condition etc. These attributes together provide a wealth of information for supply-demand prediction. However, it is nontrivial how to use all the attributes in a unified model. Currently, the most standard approach is to come up with many "hand-crafted" features (i.e., feature engineering), and fit them into an off-the-shelf learning algorithm such as logistic regression or random forest[78]. However, feature engineering typically requires substantial human efforts (it is not unusual to see data science/ machine learning practitioners creating hundreds different features in order to achieve a competitive performance) and there is little general principle how this should be done. Some prior work only keeps a subset of attributes for training, such as the timestamp, start location and drops other attributes[8,10,11,52,79]. While this makes the training easier, discarding the attributes leads to the *information loss* and reduces the prediction accuracy.

To provide some intuitions for the readers and to illustrate the challenges, we provide an example in Fig.A.1.

Example 1： Fig. A.1 shows the demand curves for two areas on March 9th (Wednesday) and March 13th (Sunday). From the figure, we can see very different patterns under different timeslots for the two areas. For the first area, few people require the car-hailing services on Wednesday. However, the demand increased sharply on Sunday. Such pattern usually occurs in the entertainment area. For the second area, we observe a heavy demand on Wednesday, especially during two peak hours around 8 o'clock and 19 o'clock (which are the commute times for most people during the weekdays). On Sunday, the demand of car-hailing services on this area reduced significantly. Moreover, the supply-demand patterns change from day to day. There are many other complicated factors that can affect the pattern, and it is impossible to list them exhaustively. Hence, simply using the average value of historic data or empirical supply-demand patterns can lead to quite inaccurate prediction results, which we show in our experiments (see Section 4.5). □

To address the above challenges, we propose an end-to-end framework for supply-demand prediction, called *Deep Supply-Demand (DeepSD)*. Our framework is based on the deep learning technique, which has successfully demonstrated its power in a number of application domains such as vision, speech and natural language processing[38,43,44]. In particular, we develop a new neural network architecture, that is tailored to our supply-demand prediction task. Our model demonstrates a high prediction accuracy, requires little hand-crafted feature, and can be easily extended to incorporate new dataset and features. A preliminary version of our model achieved the 2nd place among 1648 teams in the Didi supply-demand prediction competition.① Our technical contributions are summarized below:

- We proposed an end-to-end framework based on a deep learning approach. Our approach can automatically learn the patterns across different spatio-temporal attributes (e.g. geographic locations, time intervals and days of week), which allows us to process all the data in a unified model, instead of separating it into the sub-models manually. Compared with other off-the-shelf methods (e.g., gradient boosting, random forest[13]), our model requires a minimal amount feature-engineering (i.e., hand-crafted features), but produces more accurate prediction results.
- We devise a novel neural network architecture, which is inspired by the *deep residual network* (ResNet) proposed very recently by He et al.[35] for image classification. The new network structure allows one to incorporate the "environment factor" data such as the weather and traffic data very easily into our model. On the other hand, we can easily utilize the multiple attributes contained in the order data without much information loss.
- We utilize the *embedding* method[36,38], a popular technique used in natural language processing, to map the high dimensional features into a smaller subspace. In the experiment, we show that the embedding method enhances the prediction accuracy significantly. Furthermore, with embedding, our model also automatically discovers the similarities among the supply-demand patterns of different areas and timeslots.
- We further study the *extendability* of our model. In real applications, it is very common to incorporate new extra attributes or data sources into the already

---

① http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016. The preliminary model we used for the competition was almost the same as the basic version of our model described in Section A.3.3. Our final model, described in Section 4.4, further refines the basic model by introducing a few new ideas, and is more stable and accurate. We are currently in an effort of deploying the model and incorporate it into the scheduling system in Didi.

trained model. Typically we have to re-train the model from the scratch. However, the residual learning component of our model can utilize these already trained parameters by a simple *fine tuning* strategy. In the experiment, we show that the fine-tuning can accelerate the convergence rate of the model significantly.

- Finally, we conduct extensive experiments on a large scale real dataset of car-hailing orders from Didi. The experimental results show that our algorithm outperforms the existing method significantly. The prediction error of our algorithm is 11.9% lower than the best existing method.

4.2 Formulation and Overview

We present a formal definition of our problem. We divide a city into N non-overlapping square areas $a_1, a_2, \ldots, a_N$ and each day into 1440 timeslots (one minute for one timeslot). Then we define the car-hailing orders in Definition 1.

Definition 1 (Car-hailing Order)： A car-hailing order o is defined as a tuple: the date when the car-hailing request was sent $o.d$, the corresponding timeslot $o.ts \in [1, 1440]$, the passenger ID $o.pid$, the area ID of start location $o.loc_s \in [N]$ and the area ID of destination $o.loc_d \in [N]$. If the a driver answered the request, we say it is a *valid order*. Otherwise, if no driver answered the request, we say it is an *invalid order*.

Definition 2 (Supply-demand Gap)： For the d-th day, the supply-demand gap of the time interval $[t, t + C)$ in area a is defined as the total amount of invalid orders in this time interval. We fix the constant C to be 10 in this chapter① and we denote the corresponding gap as $\text{gap}_a^{d,t}$.

We further collected the weather condition data and traffic condition data of different areas which we refer to as the *environment data*.

Definition 3 (Weather Condition)： For a specific area a at timeslot t in the d-th day, the weather condition (denoted as wc) is defined as a tuple: the weather type (e.g.,

---

① The constant 10 (minutes) is due to the business requirement. It can be replaced by any other constant.

sunny, rainy, cloudy etc.) wc.*type*, the temperature wc.*temp* and the PM2.5 wc.*pm*. All areas share the same weather condition at the same timeslot.

Definition 4 (Traffic Condition)： The traffic condition describes the congestion level of road segments in each area: from Level 1 (most congested) to Level 4 (least congested). For a specific area a at timeslot t in the d-th day, the traffic condition is defined as a quadruple: the total amount of road segments in area a under four congestion levels.

Now, we can define our problem as below.

Problem Suppose the current date is the d-th day and the current time slot is t. Given the past order data and the past environment data, our goal is to predict the supply-demand gap $\text{gap}_a^{d,t}$ for every area a, i.e., the supply-demand gap in the next 10 minutes.

This chapter is organized as follows. We first show a basic version of our model in Section A.3.3. The basic version adopts a simple network structure and only uses the order data in the current day. In Section 4.4, we present an advanced version which is an extension of the basic version. The advanced version utilizes more attributes in the order data and it further incorporates the *historical order data* to enhance the prediction accuracy. In Section 4.5 we conduct extensive experiment evaluations. Finally, we briefly review some related work in Section A.2.1 and conclude this chapter in Section 4.6.

4.3 Basic Version

We first present the basic version of our model in this section. In Section 4.4, we extend the basic version with a few new ideas, and present the advanced version of our model. The basic model consists of three parts. Each part consists of one or more blocks (recall that the block is the base unit of our model). In Section 4.3.1, we first process the "identity features" (area ID, timeslot, day of week) in the identity part. Next in Section 4.3.2, we describe the order part which processes the order data. The order part is the most important part of our model. In Section 4.3.3, we present the environment part. The environment part processes the weather data and traffic data.

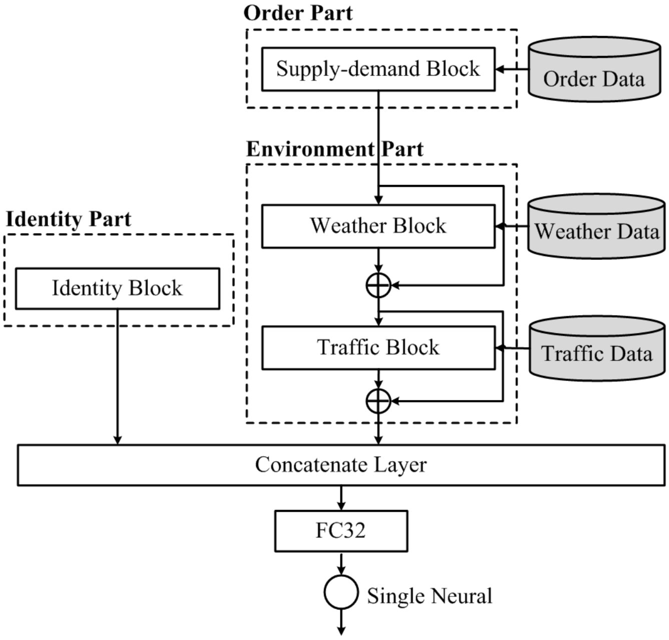


Figure 4.2 Structure of basic DeepSD

Finally, in Section 4.3.4, we illustrate how we connect different blocks. The structure of our basic model is shown in Fig. A.2.

4.3.1 Identity Part

The identity part consists of one block called the identity block. We call the features which identify the data item we want to predict as the “identity features”. The identity features include the ID of area AreaID, the timeslot TimeID and the day of week (Monday, Tuesday, ..., Sunday) WeekID. For example, if we want to predict the supply-demand gap of area a in the time interval $[t, t+10)$ in the d-th day and that day is Monday, then we have that AreaID = a, TimeID = t and WeekID = 0.

Note that the features in the identity block are categorical. As we mentioned in Section 2, we can either use the one-hot representation or embedding representation to transform the categorical values to real numbers. In our problem, since the vocabularies of AreaID and TimeID are very large, the one-hot representation leads to a high cost. Moreover, the one-hot representation treats the different areas or timeslots independently. However, we find that different areas at different time can share

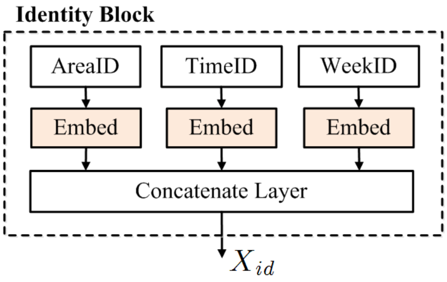


Figure 4.3 Identity Block

similar supply-demand patterns, especially when they are spatio-temporally close. For example, the demands of car-hailing are usually very heavy for all the areas around the business center at 19:00. Clustering these similar data items helps enhance the prediction accuracy. In our model, we use the embedding method to reduce the feature dimensions and discover the similarities among different areas and timeslots.

Formally, the structure of the identity part is shown in Fig. 4.3. We use three Embedding Layers to embed AreaID, TimeID and WeekID respectively. We then concatenate the outputs of three Embedding Layers by a *Concatenate Layer*. The Concatenate Layer takes a list of vectors as the input and simply outputs the concatenation of the vectors. We use the output of the Concatenate Layer as the output of the identity block, denoted as X_{id}. Furthermore, we stress that prior work[9,10,57] also clusters the similar data items to enhance the prediction accuracy. However, they treat the clustering stage as a separate sub-task and they need to manually design the distance measure, which is a non-trivial task. Our model is end-to-end and we can optimize the embedding parameters together with other parameters in the neural network. Hence we do not need to design any distance measure separately. The parameters are optimized through backpropagation towards minimizing the final prediction loss.

4.3.2 Order Part

The order part in the basic version consists only one block called the supply-demand block. The supply-demand block can be regarded as a three layer perceptron, which processes the order data. For a specific area a, to predict the supply-demand gap $\text{gap}_a^{d,t}$ of the time interval $[t, t + 10)$ in the d-th day, we consider the order set with

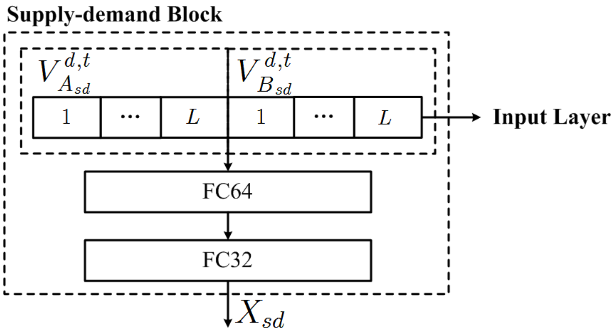


Figure 4.4 Structure of Supply-demand Block

timestamp in $[t-L,t)$ of the d-th day, which we denote as $S^{d,t}$. Here L is the window size which is specified as 20 minutes in the experiment section (Section 4.5). We then aggregate $S^{d,t}$ into a *real-time supply-demand vector*.

Definition 5 (Real-time supply-demand vector): For a specific area a, we define the real-time supply-demand vector in the d-th day at timeslot t as $V_{sd}^{d,t}$. $V_{sd}^{d,t}$ is a $2L$-dimensional vector, which consists of two parts. We denote the first L dimensions of $V_{sd}^{d,t}$ as $V_{A_{sd}}^{d,t}$. The ℓ-th dimension of $V_{A_{sd}}^{d,t}$ is defined as:

$$V_{A_{sd}}^{d,t}(\ell) = |\{o \mid o \text{ is valid} \land o \in S^{d,t} \land o.ts = t-\ell\}|$$

In another word, $V_{A_{sd}}^{d,t}(\ell)$ describes the amount of valid orders at $t-\ell$ in the current day. Similarly, we define the remaining part as $V_{B_{sd}}^{d,t}$ which corresponds to the invalid orders in the previous L minutes. □

We use $V_{sd}^{d,t}$ as the *Input Layer* of the supply-demand block. We then pass $V_{sd}^{d,t}$ through two *Fully-Connected* (abbr. FC) layers. A Fully-Connected Layer with input $\mathbf{x}$ is defined as

$$\mathrm{FC}_{\mathrm{sz}}(\mathbf{x}) = f(\mathbf{x}\cdot W + \mathbf{b})$$

where sz is the corresponding output size, W, $\mathbf{b}$ are the parameters and f is the activation function which we specify in Section 4.5. We use FC_{64} as the first Fully-Connected Layer and the FC_{32} as the second Fully-Connected Layer. The output of the supply-demand block is the output of FC_{32}, denoted as X_{sd}. See Fig 4.4 for illustration.

4.3.3 Environment Part

In the environment part, we incorporate the information from the weather data through adding the weather block to the network and the traffic data through the traffic block.

For the weather condition, we first create a *weather condition vector* $V_{wc}^{d,t}$. We show the structure of the weather block in Fig. 4.5. The vector $V_{wc}^{d,t}$ consists of L parts. For a specific $\ell \in [L]$, we have the weather condition wc at timeslot $t - \ell$ in the d-th day and we embed the weather type wc.*type* into a low dimensional space. Then the ℓ-th part of $V_{wc}^{d,t}$ is defined as the concatenation of the embedded weather type wc.*type*, the temperature wc.*temp* and the PM 2.5 wc.*pm*. Furthermore, note that the weather block also receives the output of the supply-demand block X_{sd} through a direct connection. We concatenate X_{sd} and $V_{wc}^{d,t}$ by a Concatenate Layer and pass the output of the Concatenate Layer through two Fully-Connected layers FC_{64} and FC_{32}. We denote the output of FC_{32} as R_{wc}. Then, the output of the weather block X_{wc} is defined as:

$$X_{wc} = X_{sd} \oplus R_{wc}$$

where $\oplus$ is the element-wise add operation and X_{sd} is obtained through the shortcut connection.

Note that the structure we used here is similar with ResNet as we mentioned in Section 2.2.4. However, there are two main differences between our model and ResNet. First, instead of adding shortcut connections between layers, we add the shortcut connections between different blocks. Second, in ResNet, a layer only receives the input from previous layers through a direct connection whereas in our model a block receives the inputs from both the previous block and the dataset. Such structure on one hand is more suitable for handling the data from multiple sources. On the other hand, we show that in Section 4.5.8, such structure is highly extendable. We can easily incorporate new datasets or attributes into our model based on such structure.

For the traffic condition, recall that at each timeslot the traffic condition of a specific area can be represented as the total amount of road segments in four different congestion levels. We thus create a *traffic condition vector* $V_{tc}^{d,t}$ with L parts. Each part

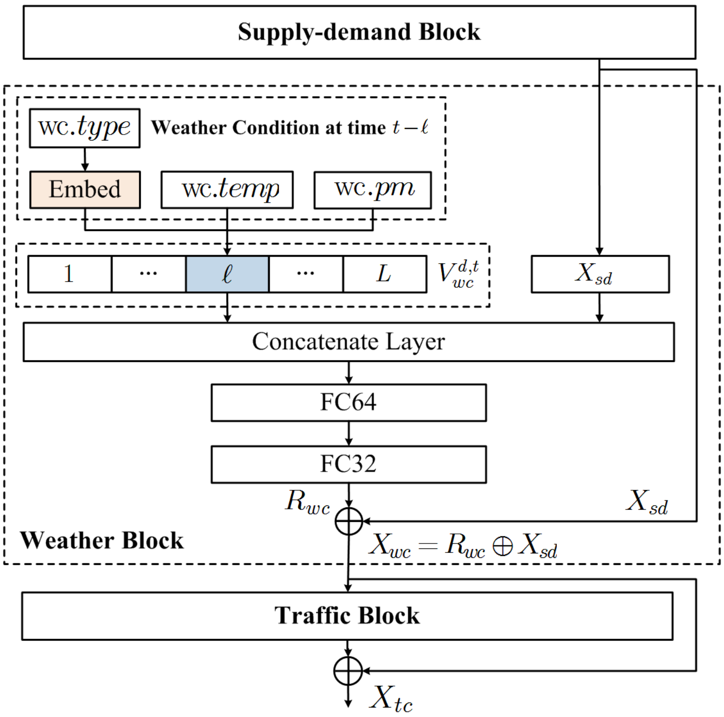


Figure 4.5 Weather Block and Traffic Block

consists of four real values corresponding to the traffic condition at that time slot. We construct the traffic block in the same way as we construct the weather block. Then, we use $X_{tc} = X_{wc} \oplus R_{tc}$ as the output of the traffic block, as shown in Fig. 4.5.

4.3.4 Block Connections

We then connect all the blocks. Note that the supply-demand block, the weather block and the traffic block are already connected through the residual learning. The output vector of these stacked blocks is X_{tc}. We then concatenate the output of the identity block X_{id} and X_{tc} with a Concatenate Layer. We append a Fully-Connected Layer FC_{32} and a single neuron after the Concatenate Layer. The single neuron finally outputs the predicted supply-demand gap with the linear activation function, as shown in Fig. A.2. We stress that our model is end-to-end, once we obtain the predicted value, we can calculate the loss based on the loss function and update each parameter with its gradient through backpropagation.

We further illustrate the intuition of our model. To predict the supply-demand gap, the most relevant and important data is the car-hailing order data. We use the supply-demand block to learn the useful feature vector from the order data. In our model, the learnt feature corresponds to the output of the supply-demand block, X_{sd}. The environment data can be regarded as the supplementary of the learnt features. Thus, we add the weather block to extract the residual R_{wc} and adjust the previous learnt features by adding R_{wc} to X_{sd}. The same argument holds for the traffic block.

4.4 Advanced Version

In this section, we present an advanced version of our model. Compared with the basic model, the advanced model replaces the *order part* in Fig. A.2 with an *extended order part* as shown in Fig. 4.6, which is composed of three blocks. The first block *extended supply-demand block* extends the original *supply-demand block* with a well-designed structure. Such structure enables our model to learn the dependence of the historical supply-demand over different days automatically, which we present in Section 4.4.1. In Section 4.4.2, we present the remaining two blocks, the *last call block* and the *waiting time block*, which have the same structure as the extended supply-demand block. Compared with the basic version where we only use the number of orders, the new blocks contains passenger information as well.

4.4.1 Extended supply-demand block

Recall that in the basic version, we use the real-time supply-demand vector $V_{sd}^{d,t}$ to predict the supply-demand gap. In the extended order block, we further incorporate the historical order data to enhance the prediction accuracy, i.e., the car-hailing orders with date prior to the d-th day. We present the extended supply-demand block in two stages. In the first stage, we obtain an *empirical supply-demand vector* in time interval $[t-L,t)$ in the d-th day. Such empirical supply-demand vector is an estimation of $V_{sd}^{d,t}$ based on the historical order data. In the second stage, we use the real-time supply-demand vector and the empirical supply-demand vector to construct our extended supply-demand block.

4.4.1.1 First Stage

We first extract the empirical supply-demand vector in $[t - L, t)$ in the d-th day, denoted as $E_{sd}^{d,t}$. It has been shown that due to the regularity of human mobility, the patterns in the traffic system usually show a strong periodicity in time on a weekly basis[6,8–10,52,63]. However, for different days of week, the supply-demand patterns can be very different. For example, in Huilongguan, a district in Beijing where most of IT employees live, the demand of car-hailing services in Monday morning is usually much more than that in Sunday morning. Motivated by this, we first consider the historical supply-demands in different days of week. Formally, we use $\mathcal{M}$ to denote all the Mondays prior to the d-th day. For each day $m \in \mathcal{M}$, we calculate the corresponding real-time supply-demand vector in that day, denoted as $V_{sd}^{m,t}$ as we defined in Definition 5. We average the vectors $V_{sd}^{m,t}$ for all $m \in \mathcal{M}$. We call such average the *historical supply-demand vector on Monday*, denoted as $H_{sd}^{(\texttt{Mon}),d,t}$. Thus, we have that,

$$H_{sd}^{(\texttt{Mon}),d,t} = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} V_{sd}^{m,t}.$$

Similarly, we define the historical supply-demand vector on the other days of week: $H_{sd}^{(\texttt{Tue}),d,t}, H_{sd}^{(\texttt{Wed}),d,t}, \ldots, H_{sd}^{(\texttt{Sun}),d,t}$.

The empirical supply-demand vector $E_{sd}^{d,t}$ is defined as a weighted combination of $\{H_{sd}^{(\texttt{Mon}),d,t}, \ldots, H_{sd}^{(\texttt{Sun}),d,t}\}$. We refer to the weight vector as *combining weights of different weekdays*, denoted as p. In our model, such weight vector p is automatically learnt by the neural network according to the current AreaID and WeekID. The network structure is shown in Fig. 4.7. We first embed the current AreaID and WeekID into a low-dimensional space. We concatenate the embedded vectors and pass it into a Softmax Layer. A Softmax Layer takes the concatenation $\mathbf{x}$ as the input and outputs the weight vector p by

$$p^{(i)} = \frac{e^{\mathbf{x} \cdot W_{\cdot i}}}{\sum_j e^{\mathbf{x} \cdot W_{\cdot j}}}, \forall i = 1 \ldots 7$$

where $W_{\cdot j}$ is the j-th column of the parameter matrix W in the Softmax Layer. Then,

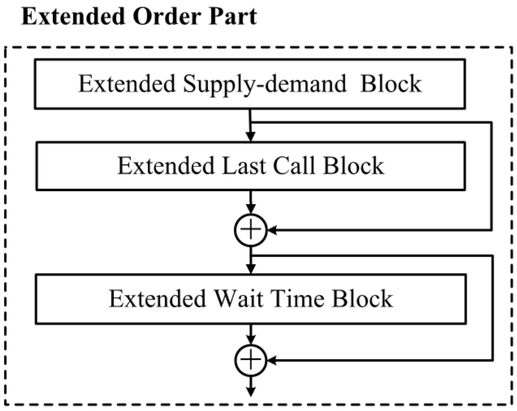


Figure 4.6 Extended Order Part

we have that

$$E_{sd}^{d,t} = p^{(1)} \cdot H_{sd}^{(\mathtt{Mon}),d,t} + \ldots + p^{(7)} \cdot H_{sd}^{(\mathtt{Sun}),d,t}. \tag{4-1}$$

We stress that most of prior work simply distinguish the historical data in weekdays and weekends separately[8–10,79,80]. However, on one hand, such method may suffer from the lack of training data. We only utilizes part of the data when we calculate the historical supply-demand vector. On the other hand, different areas can show different dependences over days of week. For example, in our experiment (Section 4.5), we find that for some areas, the supply-demands in Tuesdays are very different from the other days of week. Thus, to predict the supply-demand in Tuesday, we mainly consider the historical data in the past Tuesdays. For some other areas, the supply-demands in all the days of week are very similar. In this case, taking all the historical data into consideration leads to a more accurate result. Obviously, simply separating the historical data in weekdays and weekends can not such patterns.

4.4.1.2 Second Stage

Next, we use the obtained empirical supply-demand vector and real-time supply-demand vector to construct our block. First, using the same method as we obtain $E_{sd}^{d,t}$, we calculate another empirical supply-demand vector in time interval $[t-L+10, t+10)$ in the current day, denoted as $E_{sd}^{d,t+10}$. Note that $E_{sd}^{d,t+10}$ is the empirical estimation of

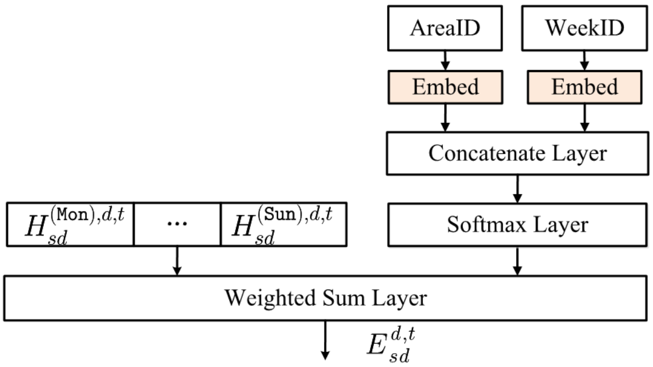


Figure 4.7 Historical supply-demand vector H_{sd}

the real-time supply-demand vector $V_{sd}^{d,t+10}$. If we can estimate $V_{sd}^{d,t+10}$ accurately, we can easily predict the currently supply-demand gap.

In our model, we use the empirical estimations $E_{sd}^{d,t}$, $E_{sd}^{d,t+10}$ and the real-time supply-demand vector $V_{sd}^{d,t}$ to estimate $V_{sd}^{d,t+10}$. We first use the Fully-Connection Layers to project these three vectors onto the same low-dimensional space (in our experiment we fix the dimensionality to be 16). We denote the projected vectors as $\mathsf{Proj}(V_{sd}^{d,t})$, $\mathsf{Proj}(E_{sd}^{d,t})$ and $\mathsf{Proj}(E_{sd}^{d,t+10})$. Instead of estimating $V_{sd}^{d,t+10}$ directly, we estimate the projection of $V_{sd}^{d,t+10}$. We denote the estimated projection as $\hat{\mathsf{Proj}}(V_{sd}^{d,t+10})$ and we have that,

$$\hat{\mathsf{Proj}}(V_{sd}^{d,t+10}) = \mathsf{Proj}(V_{sd}^{d,t}) - \mathsf{Proj}(E_{sd}^{d,t}) + \mathsf{Proj}(E_{sd}^{d,t+10}).$$

Finally, we concatenate $\mathsf{Proj}(V_{sd}^{d,t})$, $\mathsf{Proj}(E_{sd}^{d,t})$, $\mathsf{Proj}(E_{sd}^{d,t+10})$, $\hat{\mathsf{Proj}}(V_{sd}^{d,t+10})$, with a Concatenate Layer and pass it through two Fully-Connected layers FC_{64} and FC_{32}. We use the output of FC_{32} as the output of the extended supply-demand block. See Fig. 4.8 for an illustration.

We explain the reason that we estimate $V_{sd}^{d,t+10}$ in such way. The vector $\mathsf{Proj}(V_{sd}^{d,t}) - \mathsf{Proj}(E_{sd}^{d,t})$ indicates how the real-time supply-demand of $[t-L,t)$ deviates from its empirical estimation. We thus estimate $\mathsf{Proj}(V_{sd}^{d,t+10})$ by adding such deviation to the projection of empirical estimation $\mathsf{Proj}(E_{sd}^{d,t+10})$. Moreover, the projection operation on one hand reduce the dimension of each supply-demand vector from $2L$ to 16. On the other hand, we find that using the projection operation in our experiment makes our

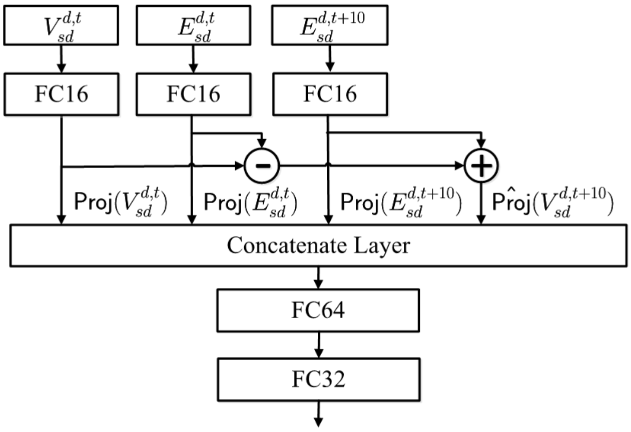

$V_{sd}^{d,t}$
$E_{sd}^{d,t}$
$E_{sd}^{d,t+10}$
FC16
FC16
FC16
$\mathsf{Proj}(V_{sd}^{d,t})$
$\mathsf{Proj}(E_{sd}^{d,t})$
$\mathsf{Proj}(E_{sd}^{d,t+10})$
$\hat{\mathsf{Proj}}(V_{sd}^{d,t+10})$
Concatenate Layer
FC64
FC32

Figure 4.8 Extended Supply-demand Block

model more stable.

4.4.2 Last Call Block and Waiting Time Block

In this section, we present two additional blocks, called the last call block and the waiting time block. Note that the order data contains multiple attributes. However, when calculating the supply-demand vector, we did not consider the attribute *o.pid*. Thus, the supply-demand vector $V_{sd}^{d,t}$ does not contain any “passenger information”. From $V_{sd}^{d,t}$, we can not answer the questions such as “how many unique passengers did not get the rides in the last 5 minutes” or “how many passengers waited for more than 3 minutes” etc. However, we find that the passenger information is also very important to supply-demand gap prediction. For example, if many passengers failed on calling the rides or waited for a long time, it reflects that the current demand exceeds the supply significantly which can lead to a large supply-demand gap in the next few minutes. We use the last call block and the waiting time block to provide the passenger information. Both of these two blocks have the same structure as the extended supply-demand block. In another word, we just replace the real-time supply-demand vector $V_{sd}^{d,t}$ in the extended supply-demand vector with the *real-time last call vector* and *real-time waiting time vector*.

For the last call block, we define the *last call vector* as follows.

Definition 6 (Real-time last call vector)： For a specific area a at timeslot t in the d-th day, we first pick out the *last call orders* in $[t-L,t)$ for all passengers, (i.e. for a specific passenger pid, we only keep the last order sent by pid), and denote the order set as $S_L{}^{d,t}$. Then, the real-time last call vector $V_{lc}^{d,t}$ is defined as a $2L$-dimensional vector. We denote the first L dimensions as $V_{A_{lc}}^{d,t}$. For the ℓ-th dimension of $V_{A_{lc}}^{d,t}$, we have that

$$
\begin{aligned}
V_{A_{lc}}^{d,t}(\ell) = |\{pid \mid \exists o \in S_L{}^{d,t}\ s.t.\ o \text{ is valid} \wedge o.pid = pid \\
\wedge\ o.ts = t-\ell\}|
\end{aligned}
$$

$V_{Alc}(\ell)^{d,t}$ describes the amount of passengers whose last call is at $t-\ell$ and she/he successfully got the ride. Similarly, we define $V_{B_{lc}}^{d,t}$ which corresponds to the passengers who did not get the rides. □

We explain the reason that we define the real-time last call vector. In our data, we find that if a passenger failed on calling a ride, she/he is likely to send the car-hailing request again in the next few minutes. Especially, the last calls near timeslot t are highly relevant to the supply-demand gap in $[t,t+10)$.

Based on $V_{lc}^{d,t}$, we can further obtain the empirical last call vector $E_{lc}^{d,t}$ with the same way as we obtain $E_{sd}^{d,t}$. We thus construct the extended real-time last call block with the same structure as the extended supply-demand block.

For the waiting time block, we define the *real-time waiting time vector* $V_{wt}^{d,t} \in R^{2L}$ in the same way as we defining $V_{sd}^{d,t}$ and $V_{lc}^{d,t}$.

Definition 7 (Real-time waiting time vector)： For a specific area a at timeslot t in the d-th day, we define the real-time waiting time vector as $V_{wt}^{d,t}$. The ℓ-th dimension in the first part $V_{A_{wt}}^{d,t}$ (first L dimensions) is the total number of passengers who waited for ℓ minutes (from her/his first call in $[t-L,t)$ to the last call) and did get the rides at last. Similarly, we define the second part $V_{B_{wt}}^{d,t}$ which corresponds to the wait time of passengers who did not get the ride.

We thus construct the extended waiting time block with the same structure of the extended supply-demand block.

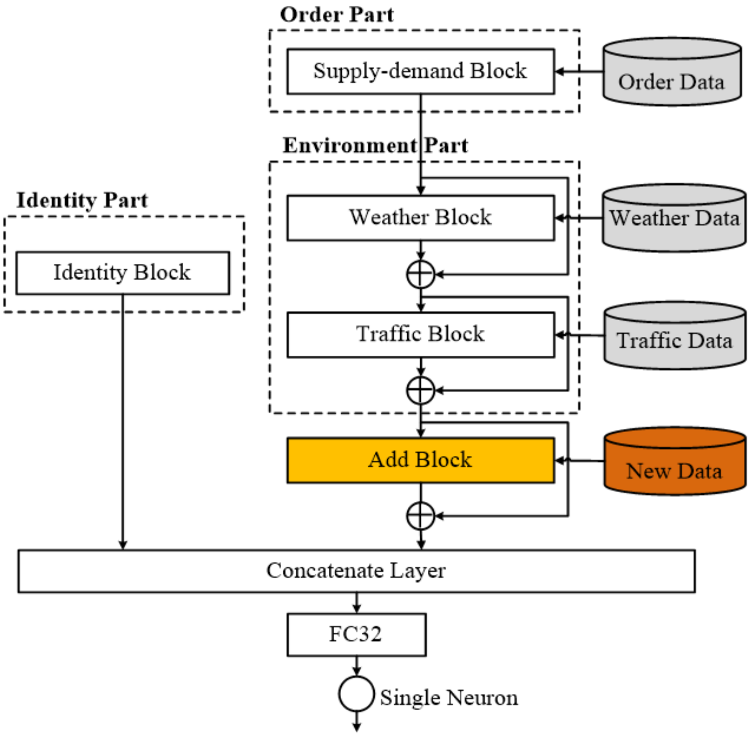


Figure 4.9 Incorporating extra feature

Finally, we connect the supply-demand block, the last call block and the waiting time block through residual learning, as shown in Fig. 4.6. These three blocks together form the extended order part in the advanced model. We use the extended order part to replace the original order part and we thus obtain the advanced version of DeepSD.

4.4.3 Extendability

Finally, in this section we present the extendability of our model. In real applications, it is very common to incorporate new extra attributes or data sources into the previous model. For example, imagine that we have already trained a model based on the order data and the weather data. Now we obtained the traffic data and we want to incorporate such data to enhance the prediction accuracy. Typically, we have to discard the already trained parameters and re-train the model from beginning. However, our model makes a good use of the already trained parameters. In our model, such scenario corresponds to that we have trained a model with the order block and the weather block.

As we show in Fig. 4.9, to incorporate the new data (such as POI data, traffic accident data, and special event data from social network), we construct the “Add Block” (we can design a totally new block for the new data, or a block similar to weather block) and connect the “Add Block” with previous blocks through the residual link. Instead of re-training the model from the scratch, we use the already trained parameters as the initialized parameters and keep optimizing the parameters of the new model through back propagation. We refer to such strategy as *fine-tuning*. In the experiment (Section 4.5), we show that the fine-tuning accelerates the convergence rate significantly and makes our model highly extendable.

4.5 Experiments

In this section, we report our experimental results on a real dataset from Didi. We first describe the details of our dataset in Section 4.5.1 and the experimental setting in Section 4.5.2. Then, we compare our models with several other most popular machine learning algorithms in Section 4.5.3. In Section 4.5.4 to Section 4.5.6, we show the effects of different components in our model. The advanced DeepSD can automatically extract the weights to combine the features of different days of a week. We present some interesting properties of the weights in Section 4.5.7. Finally, we show some results about the extendability of our model in Section 4.5.8.

4.5.1 Data Description

In our experiment, we use the public dataset released by Didi in the Di-tech supply-demand prediction competition① .

The order dataset contains the car-hailing orders from Didi over more than 7 weeks of 58 square areas in Hangzhou, China. The city map is subdivided with Geohash 5 by Didi in the competition. Geohash is a geocoding system. It is a hierarchical spatial data structure which subdivides space into buckets of grid shape, which is one of the many applications of Z-order curve, and generally space-filling curves. Each area

---

① http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016&&locale=en

(grid) is about $4.9km \times 4.9km$ large. The order dataset consists of $11,467,117$ orders. Each order includes the passenger id, the driver id, the area ids of the start location and the destination, and the time of oder was sent. During our experiment, all the passenger ids and driver ids were anonymized. We only have the Geohash 6 area ids of start location and destination, instead of the GPS location. There are only car-hailing orders from part of the city (37 areas) are given in the competition data. The gaps in our dataset are approximately power-law distributed. The largest gap is as large as 1434. On the other hand, around 48% of test items are supply-demand balanced, i.e., gap = 0. Auxiliary information include weather conditions (weather type, temperature, PM 2.5) and traffic conditions (total amount of road segments under different congestion levels in each area).

The training data is from 23th Feb to 17th March (24 days in total). To construct the training set, for each area in each training day, we generate one training item every 5 minutes from 0:20 to 24:00. Thus, we have 58(areas) × 24(days) × 283(items) = $393,936$ training items in total. Due to the restriction of test data, we set the window size $L = 20$.

The test data is from 18th March to 14th April (28 days in total). During the test days, the first time slot is 7:30 and the last time slot is 23:30. We select one time slot t every 2 hours from the first time slot unit the last time slot, i.e., t = 7:30, 9:30, 11:30, ..., 23:30. For each time slot t, we generate one test item. We use T to denote the set of test items.

4.5.1.1 Error Metrics

We evaluate the predicted results using the *mean absolute error* (MAE) and the *root mean squared error* (RMSE). Formally, we use $\text{pred}_a^{d,t}$ to denote the predicted value of $\text{gap}_a^{d,t}$. Then, the mean absolute error and the root mean squared error can be computed as follows:

$$\mathsf{MAE} = \frac{1}{|T|} \sum_{(a,d,t)\in T} \left|\text{gap}_a^{d,t} - \text{pred}_a^{d,t}\right|$$

Table 4.1 Embedding Setting

Embedding Layers	Setting	Occurred Parts
Embedding of AreaID	$\mathbf{R}^{58} \to \mathbf{R}^{8}$	Identity Part, Extended Order Part
Embedding of TimeID	$\mathbf{R}^{1440} \to \mathbf{R}^{6}$	Identity Part
Embedding of WeekID	$\mathbf{R}^{7} \to \mathbf{R}^{3}$	Identity Part, Extended Order Part
Embedding of wc.*type*	$\mathbf{R}^{10} \to \mathbf{R}^{3}$	Environment Part

$$\mathsf{RMSE} = \sqrt{\frac{1}{|T|} \sum_{(a,d,t) \in T} \left(\mathrm{gap}_a^{d,t} - \mathrm{pred}_a^{d,t}\right)^2}.$$

4.5.2 Model Details

We describe the model setting in this section.

4.5.2.1 Embedding

Recall that we map all the categorical values to a low-dimensional vector via embedding (in Section 4.3.1 and Section 4.3.3). The detailed settings of different embedding layers are shown in Table 4.1.

4.5.2.2 Activation Function

For all Fully-Connected layers, we use *Leaky Rectified Linear Unit (LReLU)*[81] as the corresponding activation function. An LReLU function is defined as:

$$\mathsf{LReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0.001 \cdot x & \text{if } x \leq 0 \end{cases}$$

For the final output neuron, we simply use the linear activation.

4.5.2.3 Optimization Method

We apply the *Adaptive Moment Estimation (Adam)* method[82] to train our model. Adam is a robust mini-batch gradient descent algorithm. We fix the batch size to be

64. To prevent overfitting, we further apply the *dropout* method[83] with probability 0.5 after each block (except the identity block).

4.5.2.4 Platform

Our model is trained on a GPU server with one GeForce 1080 GPU (8GB DDR5) and 24 CPU cores (2.1GHz) in Centos 6.5 platform. We implement our model with Theano 0.8.2, a widely used Deep Learning Python library[84].

4.5.3 Performance Comparison

We train both the basic model and advanced model for 50 epochs. We evaluate the model after each epoch. To make our model more robust, our final model is the average of the models in the best 10 epochs.

To illustrate the effectiveness of our model, we further compare our model with several existing methods. The parameters of all the models are fine-tuned through the grid search.

- **Empirical Average**: For a specific t in area a, we simply use the *empirical average gap* $\frac{1}{|D_{train}|}\sum_{d \in D_{train}} \text{gap}_a^{d,t}$ as the prediction for the supply-demand gap in time interval $[t, t+10)$.
- **LASSO[13]**: The Lasso is a linear model that estimates sparse coefficients. It usually produces better prediction result than simple linear regression. Since LASSO can not handle the categorical variables, we transform each categorical variable to the one-hot representation. We use the LASSO implementation from the scikit-learn library[85].
- **Gradient Boosting Decision Tree**: Gradient Boosting Decision Tree (GBDT) is a powerful ensemble method which is widely used in data mining applications. In our experiment, we use a fine-tuned and efficient GBDT implementation XGBoost[78].
- **Random Forest**: Random Forest (RF) is another widely used ensemble method which offers comparable performance with GBDT. We use the RF implementation from the scikit-learn library[85].

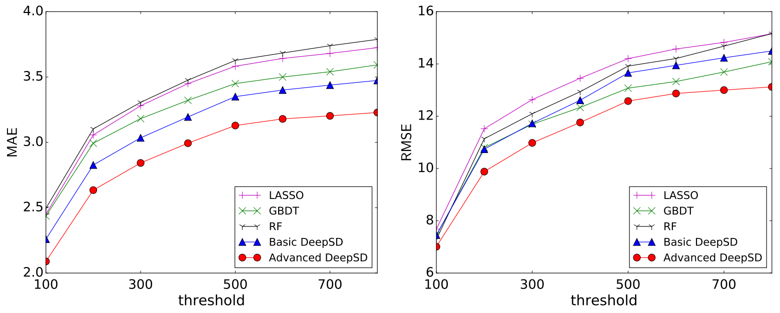
Table 4.2 Performance Comparison

Model	Error Metrics	
	MAE	RMSE
Average	14.58	52.94
LASSO	3.82	16.29
GBDT	3.72	15.88
RF	3.92	17.18
Basic DeepSD	3.56	15.57
Advanced DeepSD	**3.30**	**13.99**

For fair comparisons, we use the same input features for the above methods (except empirical average) as those used in DeepSD, including:

- AreaID, TimeID, WeekID
- Real-time supply-demand vector $V_{sd}^{d,t}$; Historical supply-demand vector of different days of week $H_{sd}^{(\texttt{Mon}),d,t}, \ldots, H_{sd}^{(\text{Sun}),d,t}$.
- Real-time last call vector $V_{lc}^{d,t}$; Historical last call vector of different days of week $H_{lc}^{(\texttt{Mon}),d,t}, \ldots, H_{lc}^{(\text{Sun}),d,t}$.
- Real-time waiting time vector $V_{wt}^{d,t}$; Historical wait time vector of different days of week $H_{wt}^{(\texttt{Mon}),d,t}, \ldots, H_{wt}^{(\text{Sun}),d,t}$.
- Weather conditions; Traffic conditions.

Table 4.2 shows the comparison results. From Table 4.2, we can see that the empirical average gap is much larger than that of the other methods. By carefully tuning the parameters, LASSO provides a much better prediction result than the empirical average. GBDT achieves the best prediction accuracy among all existing methods, for both MAE and RMSE. The overall error of the RF is somewhat worse than that of LASSO. Our models significantly outperform all existing methods. Basic DeepSD only uses the real-time order data, yet already outperforms the other methods even when they use more input features. The advanced DeepSD achieves the best prediction re-



(a) MAE (b) RMSE

Figure 4.10 Accuracy under different thresholds

Table 4.3 Effects of Embedding

Representation	Basic DeepSD			Advanced DeepSD		
	MAE	RMSE	Time (per epoch)	MAE	RMSE	Time (per epoch)
One-hot	3.65	16.12	26.4s	3.42	14.52	49.8s
Embedding	3.56	15.57	22.8s	3.30	13.99	34.8s

sults for both MAE and RMSE, which demonstrates its prediction power. The RMSE of the advanced DeepSD is 11.9% lower than the best existing method.

In Fig. 4.10, we further enumerate a threshold and compare the models under different thresholds. For a specific threshold, we evaluate the models on a subset of test data which has the gaps smaller than the threshold. Basic DeepSD shows a comparable result with GBDT for RMSE. However, for MAE, Basic DeepSD is significantly better than GBDT. For all the thresholds, Advanced DeepSD gives out the best result for both evaluations.

Fig. 4.11 shows the prediction curves of the advanced model and that of GBDT (which performs the best among all other methods). The figure shows that GBDT is more likely to overestimate or underestimate the supply-demand gap under rapid variations. See the curves in the circles in the figure. Our model provide a relatively more accurate prediction result even under very rapid variations.

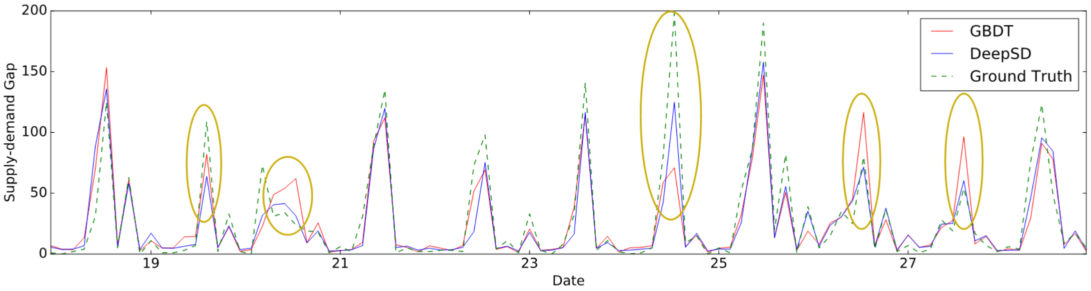


Figure 4.11　Comparison of GBDT and DeepSD. See the curves in the circles, where the ground truth changes drastically.

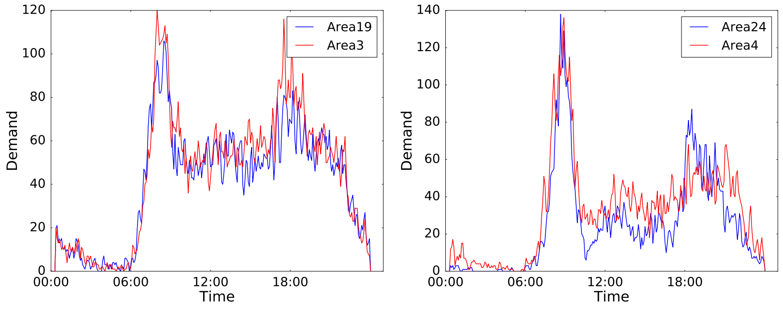
Table 4.4　Distance of embedded areas

AreaID \ AreaID	3	4	19	24
3	0.00	82.37	**10.16**	115.99
4	82.37	0.00	75.77	**26.67**
19	**10.16**	75.77	0.00	133.98
24	115.99	**26.67**	133.98	0.00

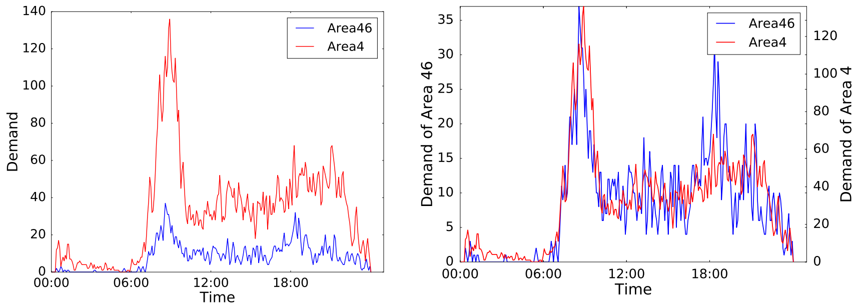
4.5.4　Effects of Embedding

Our model uses the embedding representation instead of one-hot representation for the categorical values. To show the effectiveness of embedding, we list in Table 4.3 the errors of different models with both embedding representation and one-hot representation respectively. The experimental results show that utilizing the embedding methods improves both the time-cost and the accuracy.

Moreover, recall that in Section 4.3.1, we claim that the embedding technique can cluster the data with similar supply-demand patterns to enhance the prediction accuracy. To verify this, we consider the embedded vectors of different areas. We compare the supply-demand curves of different areas. We find that if two area IDs are close in the embedding space, their supply-demand patterns are very similar. As an example, we show the pairwise Euclidean distances among four different areas in the embedding space in Table 4.4. We can see that in the embedding space, Area 3 is very



(a) Area 3 and Area 19 (b) Area 4 and Area 24

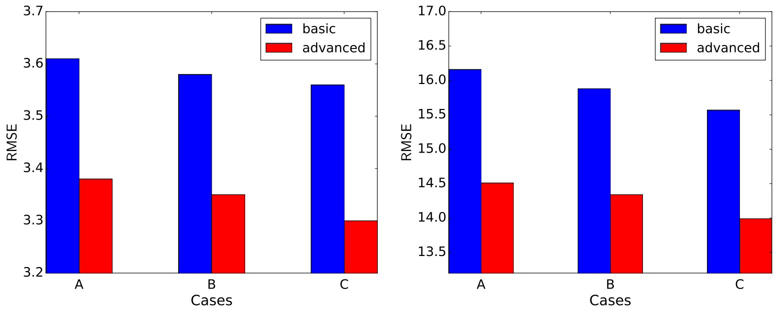


(c) Supply-demand before scaling (d) Supply-demand after scaling

Figure 4.12 Effects of Embedding. (a) and (b): Areas that have similar patterns are also closer in Euclidean distance in the embedding space. (c) and (d): Areas 46 and 4 have similar demand pattern, but at different scales.

close to Area 19 and Area 4 is very close to Area 24. We plot the car-hailing demand curves in 1st March in these areas, as shown in Fig. 4.12(a) and Fig. 4.12(b). From the figure we can see that for the areas which are close in the embedding space, their demand curves are very similar. Meanwhile, for the areas which are far apart from each other, the corresponding demand curves are very different.

More importantly, in the experiment, we find that our model is able to discover the supply-demand similarity under different scales. In another word, our model discovers the similarity of supply-demand “trends” regardless of the scales. For example, Fig. 4.12(c) shows the demand curves of Area 4 and Area 46. The demands in these two areas are in different scales and the demand curves do not even overlap. However, the distance of these two areas obtained by our model in the embedding space is only 13.34. Actually, if we plot two demand curves under the same scale (as shown in Fig.



(a) MAE　　(b) RMSE

Figure 4.13　Effects of the Environment Part

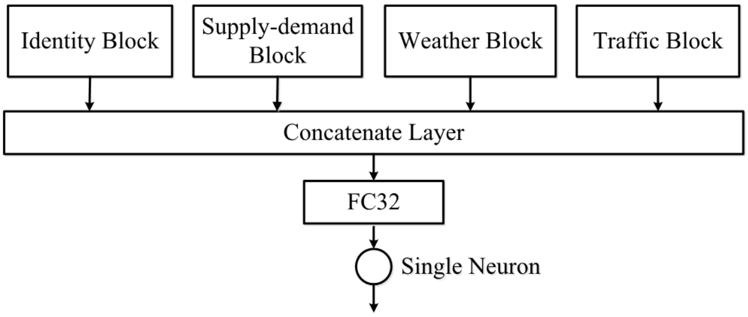


Figure 4.14　The network structure of Basic DeepSD without Residual Learning

4.12(d)), we can see that the curves are very similar, i.e., they have similar supply-demand trends.

4.5.5　Effects of Environment Part

In our model, we incorporate the environment data (e.g., weather, traffic) to further improve the prediction accuracy. To show the effectiveness of supplementary part, we compare the performances of the models under different cases. In Case A, we only use the order part/extended order part. In Case B, we further incorporate the weather block. In Case C, we use all the blocks as we presented in this chapter. Fig. 4.13 shows the prediction accuracies under different cases. Clearly, incorporating the environment data further reduce the prediction error for both the basic and advanced versions of DeepSD.

Table 4.5 Effects of Residual Learning

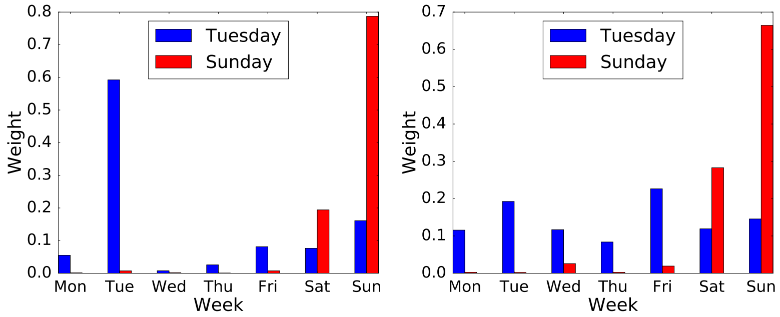
Model	With Residual Learning		Without Residual Learning	
	MAE	RMSE	MAE	RMSE
Basic DeepSD	3.56	15.57	3.63	16.40
Advanced DeepSD	3.30	13.99	3.46	15.06

4.5.6 Effects of Residual

We adopt the residual learning technique to connect different blocks. To show the effects of residual learning, we eliminate all the shortcut/direct connections and simply concatenate all the blocks by a Concatenate Layer. We show the structure of basic DeepSD without residual learning in Fig 4.14. The advanced DeepSD without residual learning can be constructed in the same way. The experimental results are shown in Table 4.5. We find that the residual learning improves the prediction accuracy effectively. In contrast, simply concatenating different blocks leads to a larger error.

4.5.7 Combining Weights of Different Weekdays

Our DeepSD model learns the relative importance for different days of a week, and use a weight vector to combine the features for different days. Specifically, from the current AreaID and WeekID, we obtain a 7-dimensional vector p, which indicates the weights of different days of week (See Equ.(4-1)). We visualize the weight vectors in two different areas at different days of week, as shown in Fig. 4.15. The blue bars correspond to the weight vector at Tuesday, and the red bars correspond to the weight vector at Sunday. As we can see, the weight vector on the Tuesday is extremely different from that on the Sunday. If the current day is Sunday, the weight is only concentrated on the weekends. This also explains the effectiveness of distinguishing the data in weekdays and weekends which is used in prior work[8–10,80]. However, even for the same day of week, the weights in different areas can be different. For example in Fig. 4.15(a), the weight of Tuesday is significantly higher than the other days whereas



(a) Area 1　　(b) Area 26

Figure 4.15　Weight vectors combining different days of a week.

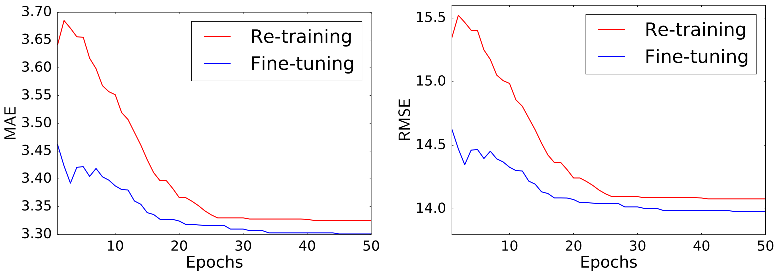
in Fig. 4.15(b) the weight of all the days are relatively uniform.

4.5.8 Extendability

As we claimed in Section A.3.1, our model is highly extendable. When introducing new attributes, we can utilize the previous trained model instead of re-training from the beginning. For example, we first train an advanced DeepSD model without the weather block and the traffic block. Now, as the weather data and the traffic data become available, we want to incorporate them to improve the prediction accuracy. For our model, we only need to add the weather block and the traffic block on top of the previous (trained) model and keep fine-refining the parameters. Fig 4.16 shows the training curves of re-training and fine-tuning respectively. The experimental result shows that refining the parameters when incorporating new extra attributes effectively accelerates the convergence rate.

4.6 Conclusion

In this chapter, we study the problem of predicting the real-time car-hailing supply-demand. We propose an end-to-end framework called *Deep Supply-Demand (DeepSD)*, based on a novel deep neural network structure. Our approach automatically discovers the complicated supply-demand patterns in historical order, weather and traffic data, with minimal amount of hand-crafted features. We conduct extensive



(a) MAE　　(b) RMSE

Figure 4.16　Convergence results of re-training and the fine-tuning method..

experiments on a real-word dataset from Didi. The experimental results show that our model outperforms the existing methods significantly. Furthermore, our model is highly flexible and extendible. We can easily incorporate new data sources or attributes into our model without re-training. We extend the future direction of this work in the Chapter 7.2.

第 5 章 Traffic Condition Prediction System

Real-time prediction of the traffic condition is an important ingredient for a variety of applications. In this chapter, we propose an *Ensemble based Traffic Condition Prediction System (ETCPS)* for predicting the traffic conditions of any roads in a city based on the current and historical GPS data collected from floating vehicles. We have observed two useful correlations in the traffic condition time series, which are the bases of our design. In order to exploit these two correlations for prediction, we propose two different models called *Predictive Regression Tree (PR-Tree)* and *Spatial Temporal Probabilistic Graphical Model (STPGM)*. Our best quality prediction is achieved by a careful ensemble of the two models. Our system provides high-quality prediction and can easily scale to very large datasets. We conduct extensive experimental evaluations with a large GPS data set collected from more than 12,000 taxis in Beijing during two months. The experimental results demonstrate the effectiveness, efficiency, and scalability of our system. ①

5.1 Introduction

Real-time prediction of the traffic condition becomes increasingly important. A well-performed traffic condition prediction system is the fundamental ingredient of various real applications. Examples include the traffic management[86], routing service[9], taxi ride sharing[87] etc. Such problem has been widely studied in recent years[3,5,57,59]. Generally, given the current and historical traffic conditions of the road network, our goal is to predict the traffic condition of each road after a few minutes or hours.

Most prior works on traffic condition prediction are based on the data generated by the road side loop sensors. However, such loop sensors are usually expensive and only embedded in highways and part of urban main roads. Alternatively, ubiquitous location based services enable us to collect a large volume of traffic data from GPS-

---

① This work has been published in DASFAA 2016[63].

embedded devices. Such GPS data provides valuable information for analyzing and predicting the traffic conditions. Despite there exist several researches and products for traffic prediction based on the GPS data, most of them only focused on the arterial roads and did not consider the urban roads.

In this chapter, we study the efficient and scalable models for traffic condition prediction based on the GPS data collected from floating vehicles (taxis in our data). To make our exposition more concrete, we first illustrate several challenges in our problem.

- Large volume of GPS data has been generated routinely, especially for some metropolises such as New York or Beijing. Most prior works are based on probabilistic graphical models[55,56,88]. The state spaces explode in these algorithms under very large scale datasets. Thus, it takes a very long time to run the algorithms.
- The traffic conditions and their transition patterns (i.e., the patterns in which the traffic condition varies) for each road vary significantly under different time intervals. For example, if the traffic is in a jam during a peak hour, it usually lasts for a long time. However, if such congestion happens in a non-peak hour, the traffic usually becomes light soon. Such traffic pattern is changing over time. Prior works based on the Markov Chain and Hidden Markov Model (HMM)[55,57,88] can not capture such feature since the states of transition matrices are not related with time.
- The taxis sometimes slow down or even stop for picking or attracting the passengers. It is hard to distinguish whether such low travel speed is due to the congestion of the traffic. Such records may lead to erroneous estimations of the traffic condition.

To address the above challenges, we propose the *Ensemble based Traffic Condition Prediction System (ETCPS)*. Our system combines two different models called *Predictive Regression Tree (PR-Tree)* and *Spatial Temporal Probabilistic Graphical Model (STPGM)*. We summarize our technical contributions below:

- We present two useful observations in the traffic condition time series which are the bases of our design. We first present the correlations between the gaps of the traffic condition and its expected traffic condition. Then, we show the autocorrelations in the first order difference of the traffic condition series (See Section 5.3).

- We propose a regression tree-based model called PR-Tree. PR-Tree can effectively capture the proposed correlations and thus predict the traffic conditions with a high accuracy. PR-Tree is very efficient on large scale datasets. Given a training set with 10^5 roads, it only takes 3.26 minutes to train a PR-Tree and the prediction of PR-Tree is real-time (See Section 5.5).
- We propose a probabilistic graphical model called STPGM. STPGM can capture the correlations between adjacent roads. It formulates the state transitions in different time intervals separately. Thus, the state space for STPGM is much smaller than the prior works[55,56,88]. On the other hand, STPGM captures different traffic patterns in different time intervals. We show that in the experiment STPGM is more efficient and accurate than the algorithms in prior works (See Section 5.6).
- We propose a prediction system called ETCPS which combines PR-Tree and STPGM. We evaluate our model with real dataset which consists of GPS points generated by over 12,000 taxis collected in two months. It provides an experimental evidence that ETCPS is efficient, scalable in terms of supporting large size road networks, and achieves a high-quality prediction (See Section 5.7).

5.2 Problem Statement

5.2.1 Road Network

We are given a data set consisting of GPS records of taxis. The GPS records of the j-th taxi is represented by $Tr_j = \{p_1, p_2, \ldots, p_{|Tr_j|}\}$. Each p_i represents a GPS record $(cid, time, location, speed)$ indicating the id of the j-th car, the time stamp when the record is generated, the latitude and longitude of the current location and the instantaneous speed respectively. We define a real urban road network as a directed graph

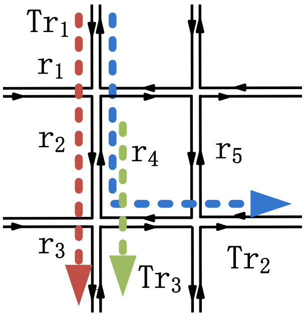


Figure 5.1 Time cost

Traj	Time (Intv)	Road Seg	Speed (km/h)	Traj	Time (Intv)	Road Seg	Speed (km/h)
Tr_1	34	r_1	56	Tr_2	34	r_1	60
Tr_1	35	r_2	60	Tr_2	35	r_2	58
Tr_1	35	r_3	61	Tr_2	35	r_4	58
Tr_3	35	r_2	15	Tr_2	36	r_5	60
Tr_3	35	r_3	60				

Figure 5.2 Time cost (Million seconds)

$G = (V, E)$ where V is the set of nodes representing the terminal points of road segments and E is the set of road segments. A road segment r_i is a directed edge associated with a start point v_s, an end point v_e with length l_i. See Figure 5.1 for an illustration. Utilizing the technique of map-matching[89], each GPS record p_i on the trajectory Tr_j can be located to a road segment r_i in which the car j is traveling on.

5.2.2 Traffic Condition

We define the traffic condition for a road segment r_i during a specific period as below. Given a GPS data set collected during D days, we split the period of D days into several intervals, and each time interval spans λ minutes. We assume that the traffic condition of a specific road segment remains unchanged in one interval. Such assumption is widely used in the transportation literature[3,57].

As each day has $M = \frac{60 \cdot 24}{\lambda}$ time intervals, for a GPS data set collected during D days, there are $T = M \cdot D$ time intervals. The t-th interval is $[t \cdot \lambda, (t + 1) \cdot \lambda)$. For example, if we set $\lambda = 15, D = 31$, then we have $M = 96$, $T = 2976$, and the interval 34 is a time period from $8 : 30$ to $8 : 45$ in the first day.

By mapping each GPS record to a road segment, we consider the average speed of all the records observed in the t-th interval on a road segment. For example, in Table 5.2, the observed average speed for r_2 in the 35-th interval is $(60+58+15)/3$. However, some taxis may run at a very low speed or even stop for *boarding* or *balling* when the road is not congested. We regard such records as the noise which is eliminated in the pre-processing stage (see Section 5.8 for details). Then, the *traffic condition* of a road segment r_i in the t-th interval is defined as the average speed of all the GPS records observed in this road segment during the t-th interval, denoted as o_t^i. Note that for some road segments, there may not exist any GPS record in the t-th interval and thus we can not define the corresponding traffic condition. We explain how we deal with such case in Section 5.8 . Currently, we simply assume o_t^i is well-defined for all i and t. Moreover, we use $\mathrm{Org}^i = \{o_1^i, \ldots, o_T^i\}$ to denote the traffic condition time series of road segment r_i.

Expected Traffic Condition Note that the traffic conditions usually have the "daily pattern". For example, a road segment is usually in a jam during 6:00-9:00 each day whereas from 9:00 to 11:00 it is usually light. For the t-th interval, we define $t \bmod M$ as its *daily index*, i.e., it is the $t \bmod M$-th interval in its corresponding day. For example, if we set $M = 96$, then the 226-th interval represents the time period from $8:30$-$8:45$ in the third day and its daily index is $226 \text{ mod } 96 = 34$. Let $A_t^i = \{o_{t'}^i | t' \equiv t \text{ mod } M\}$ be the set of traffic conditions observed in road segment r_i during the t mod M-th interval for all days. For example, in Table 5.2, the 34-th interval is a time period from $8:30$ to $8:45$ on the first day. Then, A_{34}^i is the set of traffic conditions of the road segment r_i in all days from $8:30$ to $8:45$. We call the mean of A_t^i the *expected traffic condition* of r_i in time interval t, denoted as $a_t^i = \sum_{a \in A_t^i} a / |A_t^i|$. Essentially, the expected traffic condition a_t^i indicates the value that traffic conditions are usually around, in the t mod M-th interval of a day. We use $\mathrm{Avg}^{\mathrm{i}} = \{a_1^i, \ldots, a_T^i\}$ to denote the expected traffic condition time series of the road segment r_i. Note that $\mathrm{Avg}^{\mathrm{i}}$ is a periodic series and once we have the training data, a_t^i is always available for all $t \in Z$.

Problem Definition Given the historical traffic conditions before time interval T, $\mathrm{Org}^{\mathrm{i}} = \{o_1^i, \ldots, o_T^i\}$ for all i, our goal is to predict the traffic condition on the $T+1$-th interval o_{T+1}^i or even longer for each road segment r_i. For convenience, for any t, we use p_t to denote the predicted traffic condition in the time interval t.

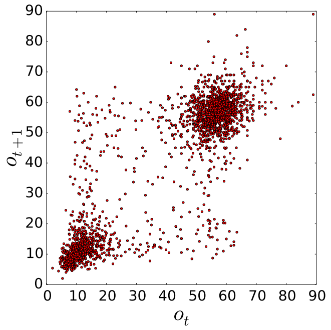


Figure 5.3　o_t and o_{t+1}

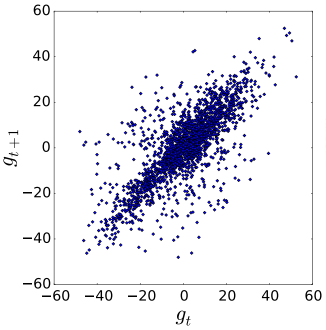


Figure 5.4　g_t and g_{t+1}

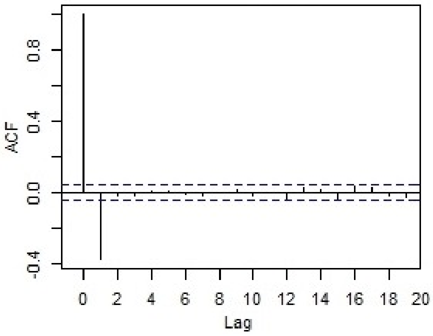


Figure 5.5　ACF of Diff(Org)

5.3　Useful Observations

Most of prior works predict the future traffic conditions directly based on the traffic condition time series. However, it is difficult to extract the patterns in the traffic condition time series Org^i. We find that by transforming the Org^i into two different forms of time series, the new time series reveal very strong autocorrelations. We hope these observations can provide useful insight in further study of the travel condition prediction problem and related problems.

Expectation-reality gap The traffic condition time series of the same road segment in each day usually exhibits strong periodic pattern which we refer to as the “daily pattern”. We eliminate the daily pattern from the traffic condition series by subtracting the corresponding expected traffic condition from each of the traffic conditions. Specifically, we set $g_t^i = o_t^i - a_t^i$ and we thus obtain a new series $\text{Gap}^{\text{i}} = \{g_t^i | t = 1, \ldots, T\}$. Intuitively, if $g_t < 0$, it means that the traffic condition in the time interval t is more congested than usual. We find that there exists a strong correlation between g_{t+1} and g_t. Fig. 5.3 and Fig. 5.4 show the scatter diagram of (o_t, o_{t+1}) and (g_t, g_{t+1}) of a specific road segment respectively. As we can see, by transforming the traffic condition series Org^{i} to the gap series Gap^{i}, we essentially extract the “pattern” of the traffic condition series.

First order difference of traffic condition series We use $\delta_t^i = o_t^i - o_{t-1}^i$ to represent the first order difference of traffic condition series, denoted as Diff(Org). We use ACF (Auto Correlation Function) to analyze the autocorrelation in the time series of

δ_t^i. The autocorrelation of a random process describes the correlation between values of the process at different times with a time lag τ. Given a time series and time lag, ACF returns a value between $+1$ (total positive correlation) and -1 (total negative correlation) inclusive. If the absolute value of ACF is beyond ± 0.05, we usually think the time series is autocorrelated at time lag τ. In Fig. 5.5, we show the ACF value of the time series δ_t of a random road segment. The horizontal axis represents the time lag τ, and vertical axis represents the ACF value at lag τ. As the ACF value at lag $\tau = 1$ is far beyond the threshold -0.05, we conclude that there exists a correlation between δ_t and δ_{t+1}.

5.4 System Overview

The framework of our proposed traffic condition prediction system is illustrated in Fig. A.3. We develop a system that utilizes the historical and real time taxi GPS records to estimate the current travel condition and predict the travel conditions in the next time intervals. It is composed of four major components: Pre-processing, Predictive Regression Tree Model (PR-Tree) , Spatial Temporal Probabilistic Graphical Model (STPGM) and Ensemble.

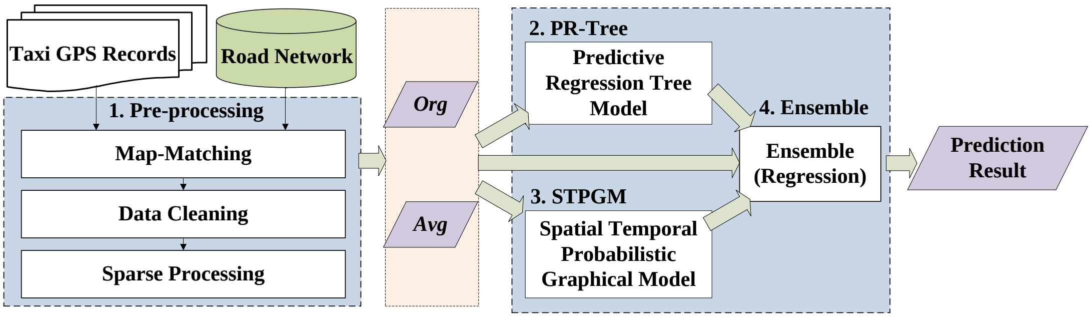


Figure 5.6 Overview of system framework.

In the pre-processing phase, first, we map match the GPS trajectories to road networks using the ST-Matching algorithm[57]. Then, we eliminate the records which are under boarding or balling state. We then deal with the sparsity issue that no GPS record is observed for some roads during some time intervals. With the pre-processing, we thus obtain two time series Org and Avg as defined in Section 5.2. The details are presented in the experiment part (Section 5.8). In Section 5.3, we illustrate two useful observations. Next, in Section 5.5, we use a regression tree based model called PR-Tree to predict the future traffic conditions based on our observed correlations. We further adopt a probabilistic graphical model called STPGM in Section 5.6 which captures both our observations and the correlations between the road segments. Finally, we combine two models in the ensemble stage as shown in Section 5.7. We show that combining two different models enhances the accuracy of the prediction in Section 5.8.

5.5 Predicting The Traffic Condition with PR-Tree

In this section , we define a regression tree based model called PR-Tree to predict the traffic condition of each road segment individually. We first describe the structure of PR-Tree in detail and how we predict the traffic condition on this tree in Section 5.5.1. Then in Section 5.5.2, we present the training algorithm of PR-Tree.

5.5.1 Description of PR-Tree

Recall that the time series Gap shows a strong autocorrelation as we claimed in Section 5.3. We can thus approximate g_{t+1} by an estimation $\hat{g}_{t+1}$ based on g_t and predict the traffic condition in the $t + 1$-th interval by $p_{t+1} = a_{t+1} + \hat{g}_{t+1}$ (the expected traffic condition a_{t+1} is always available as we claimed in Section 5.3). From Fig. 5.4, it is reasonable to set $\hat{g}_{t+1} = \theta \cdot g_t$ since the scatter diagram shows a nearly linear correlation. However, we find that the ratio g_{t+1}/g_t varies when g_t takes different values. For example, if g_t is close to -10, g_{t+1} is usually around 1.2 times g_t whereas if g_t is close to -8, g_{t+1} is usually around 1.4 times g_t. Motivated by this, instead of estimating g_{t+1} by $\theta \cdot g_t$, we use a proper function $R(g_t)$ and estimate g_{t+1} by $g_t \cdot R(g_t)$.

Structure To learn a proper function R, we propose a regression tree based model called PR-Tree. Specifically, PR-Tree splits the input space into several subspaces. Each subspace is associated with an output parameter θ. Given the input g_t, we find the subspace corresponding to g_t and return the corresponding θ as $R(g_t)$. Formally, each

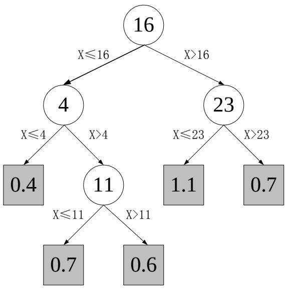


Figure 5.7 An example of PR-Tree

inner node of PR-tree has a splitting value and each leaf node has an output parameter θ. To find the corresponding subspace of g_t, we search on PR-Tree as follows. Initially, the current node is the root of PR-Tree. If g_t is less than or equal to the splitting value of the current node, we search the left child recursively. Otherwise, we search the right child. We perform such search until it reaches a leaf node and returns the corresponding θ on the leaf node as $R(g_t)$. For simplicity, we use R to represent the corresponding PR-Tree.

We show an example of a PR-Tree in Figure 5.7. The PR-Tree contains four inner nodes (the splitting value of these nodes are $\{4, 11, 16, 23\}$), and five leaf nodes (their values are $\{0.4, 0.7, 0.6, 1.1, 0.7\}$). We take $g_t = 5$ as the input. As the splitting value of the root node is 16 and $g_t \leq 16$, we search its left child recursively and finally reach a leaf node with output parameter $\theta = 0.7$.

Prediction To predict the traffic condition in the time interval $t + 1$, we simply set $\hat{g}_{t+1} = R(g_t) \cdot g_t$ and predict o_{t+1} by $p_{t+1} = a_{t+1} + R(g_t)$. Fig. 5.7 shows an example. Given the current traffic condition $o_t = 45$, assuming the expected traffic condition on t and $t + 1$ are $a_t = 40$, $a_{t+1} = 43$, we get $g_t = o_t - a_t = 5$. By taking g_t as the input of PR-Tree, we get $R(g_t) = 0.7$. Then, we estimate o_{t+1} by $a_{t+1} + R(g_t) \cdot g_t = 46.5$.

5.5.2 Training PR-Trees

First, we present the objective for training PR-Trees. Recall that we predict o_{t+1} as $p_{t+1} = a_{t+1} + R(g_t) \cdot g_t$. Given the training set $\mathrm{Org}^{\mathrm{i}} = \{o^i_1, \ldots, o^i_T\}$, our goal is to minimize the squared error $\sum_{t\in[1,T)} (p_{t+1} - o_{t+1})^2$. Equivalently, we need to find an

Algorithm 1 PR-Tree Splitting (Split)

```
Require: Node $root$, Training sequence $TR$, cross validation sequence $CV$
Ensure: Update the PR-Tree.
$e_{TR} = f(TR, \mathsf{out}(TR))$
$e_{min} = \infty$
for $i = 1, \ldots, |TR| - 1$ do
    $TR_l \leftarrow$ first $i$ elements in $TR$
    $TR_r \leftarrow T \backslash TR_l$
    if $f(TR_l, \mathsf{out}(TR_l)) + f(TR_r, \mathsf{out}(TR_r)) < e_{min}$ then
        $e_{min} = f(TR_l, \mathsf{out}(TR_l)) + f(TR_r, \mathsf{out}(TR_r))$
        $TR_l^* = TR_l$ , $TR_r^* = T \backslash TR_l^*$        ▷ update the best $TR_l$
    end if
end for
If $e_{min} > e_S - \gamma$ return
$root.lc \leftarrow$ a new node corresponds to $TR_l^*$        ▷ split $root$
$root.rc \leftarrow$ a new node corresponds to $TR_r^*$        ▷ split $root$
if $best_{CV} > Q(CV)$ then        ▷ qualify the splitted PR-Tree
    $best_{CV} = Q(CV)$        ▷ update the global best value
    $\mathsf{Split}(root.lc, TR_l^*, CV)$, $\mathsf{Split}(root.rc, TR_r^*, CV)$
    set the splitting value of $root$ as $max_{s \in TR_l^*}\ s.u$        ▷ inner node
else
    $root.lc$ = None, $root.rc$ = None
    set the output value of $root$ as $\mathsf{out}(TR)$        ▷ leaf node
    return
end if
```

optimal PR-Tree (function R^*) that

$$R^* = \underset{R}{\operatorname{argmin}} \sum_{t \in [1,T)} (g_{t+1} - R(g_t) \cdot g_t)^2 \tag{5-1}$$

Our training algorithm is slightly different from the standard regression tree training algorithm. To train the PR-Tree, given the time series Gap $= \{g_1^i, \ldots, g_T^i\}$, we construct another sequence $S = \{(u, v)|u = g_t, v = g_{t+1}, \forall t = [1, T)\}$. Each element $s \in S$ indicates a pair of values (g_t, g_{t+1}). We use $s.u$ to denote the first value in pair s and $s.v$ to denote its second value. We sort S by increasing order of $s.u$. For any subsequence $S_x \subset S$ and any PR-Tree R, we define the cost of S_x as $Q(S_x) = \sum_{s \in S_x} (s.v - R(s.u) \cdot s.u)^2$. which represents the squared error if we use PR-Tree R to fit the set S_x.

Our training algorithm works as follows. During the training phase, each node corresponds to a subsequence of $S_x \subset S$. For a specific node, if it is an inner node, we use S_l, S_r to denote the corresponding subsequences of its left child and its right child respectively. Then, its splitting value is $\max_{s \in S_l} s.u$. Otherwise, it is a leaf node. We define $f(S_x, \alpha) = \sum_{s \in S_x} (s.v - \alpha \cdot s.u)^2$. The output θ of this leaf node is $\operatorname{argmin}_\alpha f(S_x, \alpha)$, denoted as $\mathsf{out}(S_x)$.

Initially, we have a singleton tree. There is the only one node which corresponds to S. We split the PR-Tree recursively. For each node, there is a best splitter S_l^*, i.e.,

$$S_l^* = \operatorname*{argmin}_{S_l} \{f(S_l, \mathsf{out}(S_l)) + f(S \backslash S_l, \mathsf{out}(S \backslash S_l))\}.$$

We enumerate the first i elements of S_x as S_l ($S_r = S \backslash S_l$) to search the best splitter S_l^* (line 3 to line 10 in Algorithm 1). Note that since S is sorted and $f(S_l, \alpha)$ is the sum of quadratic terms which is still quadratic. To obtain the best splitter S_l^*, we can maintain the coefficients of $f(S_l, \alpha)$ and the minimum of the quadratic term can be calculated in $O(1)$ time. Each time when we enumerate a new subsequence, we only need to update the coefficients. Thus, we can obtain the best splitter in $O(|S|)$ time efficiently. We denote $S_r^* = S_x \backslash S_l^*$. If $f(S_l^*, \mathsf{out}(S_l^*)) + f(S_r^*, \mathsf{out}(S_r^*)) < f(S_x, \mathsf{out}(S_x)) - \gamma$, we split the current node into two child nodes with subsequences S_l^* and S_r^* respectively where γ is a threshold to be specified (line 12 to line 13). Otherwise, we terminate the recursion (line 11).

The readers may notice that such splitting procedure may cause a serious overfitting problem, i.e., the PR-Tree keeps splitting until each node only contains a very short subsequence. To remedy this issue and reduce the generalization error, we split S into two parts, the training part TR and the cross validation part CV. We use TR to train PR-Tree, each time when a node is split, we qualify the current PR-Tree on the cross-validation set CV and check whether if $Q(CV)$ decreases (line 14). If the qualification on CV does not decrease, we undo the splitting operation (line19 to line21) and terminate the recursion. Otherwise, we continue the splitting operation (line 15 and line 17) and split its children nodes recursively (line 16). See Algorithm 1 for the pseudo code.

5.6 Predicting Traffic Condition with STPGM

Despite that the PR-Tree performs well in most of our data (which we show in Section 5.8), it does not consider the correlations between the road segments. Some roads are easily affected by its neighbors, the congestions of its neighbors usually lead to the congestion of itself in the next few time intervals. For such roads, PR-Tree does not perform well. Motivated by this, we propose a probabilistic graphical model called STPGM which is used in combination with the PR-Tree in our system.

We first construct a *spatial temporal probabilistic graph (STPG)* G_p which corresponds to a road network G. If a vehicle can travel from the road segment r_i to the road segment r_j (or from r_j to r_i) directly, we say that r_i and r_j are adjacent. We construct a vertex v_i in G_p which corresponds to a road segment r_i in G. We add an edge between v_i and v_j if and only if the road segments r_i and r_j are adjacent. For a specific v_i, we use $Neib(v_i)$ to denote all the adjacent vertices of v_i. Intuitively, the adjacent road segments affect each other much more significantly than the other road segments. Thus, each edge in G_p represents a “strong effectiveness” in the road network.

5.6.1 States of STPGM

We first discretize the traffic conditions into different states. Recall that as we claimed in Section A.4.1, the traffic conditions and the transition patterns are very different not only at different road segments, but also at different time intervals. However, for a specific road segment, we find that the traffic conditions and transition patterns are usually similar for the time intervals with the same daily index. For example, if the traffic is congested in $8:00$, it usually stays congested in next several time intervals.

However, if the traffic is congested in 10 : 00, the traffic becomes light in the next few minutes with a large probability. Motivated by this, we consider different time intervals separately and use the same state sets for the time intervals with the same daily index.

For a specific road segment r_i, instead of clustering all of its traffic conditions in series $\mathrm{Org^i}$ (which are widely used in the prior works[9,56,57,88]), we consider the traffic conditions under different daily index separately. Formally, we consider a specific daily index $l \in [M]$. Recall that $A^i_l = \{o^i_t | t \equiv l \bmod M\}$. We cluster the traffic condition set A^i_l into k clusters with K-Mediods where k is a parameter to be specified (see Section 5.8 for details). For example, if the daily index l corresponds to 8 : 30-8 : 45 in a day, then we cluster the traffic conditions for all days during 8 : 30-8 : 45. We use the center $c^i_{x,l}$ of each cluster to represent a state, and denote the set of the centers as $C^i_l = \{c^i_{1,l}, \ldots, c^i_{k,l}\}$. The state of the traffic condition in the time interval t is represented by its nearest center in $C^i_{[t \bmod M]}$, denoted as s^i_t. We show an example of a random selected road segment r_i where $C^i_{25} = \{44, 48, 52, 58\}$ and $C^i_{74} = \{15, 25, 32, 38\}$ (km/h). The time interval 25 corresponds to 6 : 00-6 : 15 where the traffic is usually light and the time interval 74 corresponds to 18 : 30-18 : 45 where the traffic is usually heavy.

5.6.2 Parameter Learning

We predict the traffic condition of a specific vertex (corresponds to a road segment) v_i based on the historical traffic conditions of itself and its neighbors. We assume that the traffic condition of v_i in the time interval $t + 1$ is only related with the traffic conditions of v_i and $Neib(v_i)$ in the time interval t.

Formally, consider a vertex v_i. Let $\{v_i\} \cup Neib(v_i) = \{v_{i_1}, \ldots, v_{i_n}\}$ and the corresponding states in time interval t are $\{c^i_{x_i,t}, c^{i_1}_{x_{i_1},t}, c^{i_2}_{x_{i_2},t}, \ldots, c^{i_n}_{x_{i_n},t}\}$. Our goal is to learn the transition probability for all the possible states in $C^i_{(t+1) \bmod M}$, i.e.,

$$\begin{aligned} & P(s^i_{t+1} = c^i_{x_i,t+1} | s^{i_1}_t = c^{i_1}_{x_{i_1},t}, s^{i_2}_t = c^{i_2}_{x_{i_2},t}, \ldots, s^{i_n}_t = c^{i_n}_{x_{i_n},t}) \\ = & \frac{P(s^i_{t+1} = c^{x_i}_{i,t+1}, s^{i_1}_t = c^{i_1}_{x_{i_1},t}, \ldots, s^{i_n}_t = c^{i_n}_{x_{i_n},t})}{P(s^{i_1}_t = c^{i_1}_{x_{i_1},t}, \ldots, s^{i_n}_t = c^{i_n}_{x_{i_n},t})} \end{aligned} \tag{5-2}$$

For the prediction, it is unnecessary to compute the the denominator, which we show in Section 5.6.3. As for the numerator, the state space in Equation 5-2 explodes exponentially whereas the training data is relatively limited. It is not sufficient to estimate the numerator precisely. Thus, we approximate the numerator of Equation 5-2 by

$$P(s^i_{t+1} = c^{x_i}_{i,t+1}) \prod_{j=1}^{n} P(s^t_{i_j} = c^{x_{i_j}}_{i_j,t} | s^i_{t+1} = c^i_{x_i,t+1}) \tag{5-3}$$

where $P(s^t_{i_j} = c^{x_{i_j}}_{i_j,t} | s^i_{t+1} = c^i_{x_i,t+1})$ indicates that given the observed state in the time interval $t + 1$, the probability that the previous state of v_{i_j} is $c^{x_{i_j}}_{i_j,t}$.

We define the indicator function $I(s^i_t, c^i_{x,t})$ which indicates that whether the state of the road segment r_i in the time interval t equals $c^i_{x,t}$. We use $N = \sum_{t' \equiv t \bmod M} I(s^i_{t'}, c^i_{x,t})$ to represent the total days that the state of the road segment r_i in the $t \bmod M$-th interval of each day is $c^i_{x,t}$. Then, we calculate the probability $P(s^i_t = c^i_{x,t})$ by the frequency $P(s^i_t = c^i_{x,t}) = N/D$. Similarly, for the term $P(s^t_{i_j} = c^{x_{i_j}}_{i_j,t} | s^i_{t+1} = c^i_{x_i,t+1})$, we have

$$P(s^t_{i_j} = c^{x_{i_j}}_{i_j,t} | s^i_{t+1} = c^i_{x_i,t+1}) = \frac{\sum_{t' \equiv t \bmod M} (I(s^i_{t'+1}, c^i_{x_i,t+1}) \cdot I(s^{i_j}_{t'}, c^{i_j}_{x_{i_j},t}))}{\sum_{t' \equiv t \bmod M} I(s^{i_j}_{t'+1}, c^{i_j}_{x_{i_j},t+1})}. \tag{5-4}$$

Thus, we get the approximation of the numerator of Equation 5-2.

5.6.3 Prediction

Suppose the traffic conditions of the road network in time interval t are observed. We first construct the states for each road segment r_i. To predict the traffic condition of a road segment r_i, after obtaining the states of v_i and $Neib(v_i)$ in the time interval t, we use Equation 5-2 to infer the probability of each state for v_i in the time interval $t + 1$. Then, we select the state with the largest probability as the predicted state and the corresponding cluster center as the predicted traffic condition. Note that as the denominator of Equation 5-2 is a constant value when the states of v_i and $Neib(v_i)$ in the time interval t are given, it is actually unnecessary to compute this denominator.

5.7 Model Extensions

Ensemble We find that in the experiment, the performances of PR-Tree and STPGM differ in different roads. Some roads are rarely affected by their neighbors, such as the arterial roads. For such roads, PR-Tree outperforms STGPM. However, as PR-Tree does not consider the correlations of the roads, STPGM performs better than PR-Tree for the roads which are highly affected by its neighbors, especially the roads that only few GPS records are observed. Our prediction for traffic condition in the $t+1$-th interval is a linear combination of the previous traffic condition o_t^i, the prediction obtained by PR-Tree and STPGM. The weights of the linear combination is obtained by linear regression. We show that in the experiment, by combining the models, our system achieves a higher accuracy for the prediction.

Alternate of the input series In fact, both the PR-Tree and STPGM are the models which capture the correlations in a time series. Recall that in the PR-Tree model, we use the time series Gap as the input. In STPGM, we use the traffic condition time series Org as the input. Essentially, we can use the any time series related with the traffic as the input of both models and predict the traffic condition in a proper way. For example, if we use the Org as the input of a PR-Tree, we actually try to approximate o_{t+1}^i by $o_t^i \cdot \theta(o_t^i)$ and we predict the traffic condition directly use $\theta(o_t^i)$. Similarly, we can use the Gap as the input of STPGM. Besides the proposed two series, we can also use the first order difference of Org (i.e., Diff(Org) as defined in Section 5.3) as our input or the traffic conditions filtered with Kalman filtering. The details are presented in Section 5.8.

5.8 Experiments

In this section, we evaluate the effectiveness and efficiency of the proposed models.

5.8.1 Experiment Setting

Data Set In all experiments, we use the real dataset which consists of GPS records collected from 12,000 taxis from November 1st to December 31st in 2012 ① .The GPS data are map matched[89] [90] to road network ② of Beijing. During our experiment, all the driver id were anonymized by recoding. We evaluate our algorithms on the data of November and December respectively. For each month, we divide the data set into the training set (1st - 24th), and the test set (25th - the last day). We distinguish two cases in our experiments: the standard case and the sparse case. For the standard case, we select 10812 road segments which contains more than 140 GPS records per day in average. In the sparse case, we select 101672 road segments in which the GPS records occurred in more than 10 time intervals per day in average. In all experiments, we focus on the time period from $6:00$ to $24:00$ in each day since there are only few GPS records observed during $00:00$ to $6:00$.

Measurement We evaluate the performances of our models on the test data set by Mean Absolute Error (MAE), Mean Relative Error (MRE) and Mean Squared Error (MSE), i.e., MAE $= \frac{1}{|E|}\sum_{i=1}^{|E|}\sum_{t=1}^{T}|p_t^i - o_t^i|$, MRE $= \frac{1}{|E|}\sum_{i=1}^{|E|}\sum_{t=1}^{T}|p_t^i - o_t^i|/o_t^i$, MSE $= \frac{1}{|E|}\sum_{i=1}^{|E|}\sum_{t=1}^{T}(p_t^i - o_t^i)^2$. Recall that we evaluate our algorithms on the datasets of November and December respectively. For convenience, for each model, we use the mean of the errors on the two months as the final error. All the experiments are implemented parallelly with Python 2.7 and run on a service on Open Stack (Intel X-eon E312 CPU of 16 cores with 2.1GHz for each core and 32GB memory on Ubuntu 14.04LTS operate system).

5.8.2 Pre-processing

Data Cleaning In the data cleaning phase, we eliminate the GPS records for taxis which slow down or even stop for picking or attracting passengers. We distinguish two cases of such records. One is *boarding*, i.e., the passengers get on or get off the taxi. The other is *balling*, i.e., the taxis slow down or stop to attract guests who need

---

① This data can be downloaded in http://www.datatang.com/data/45888

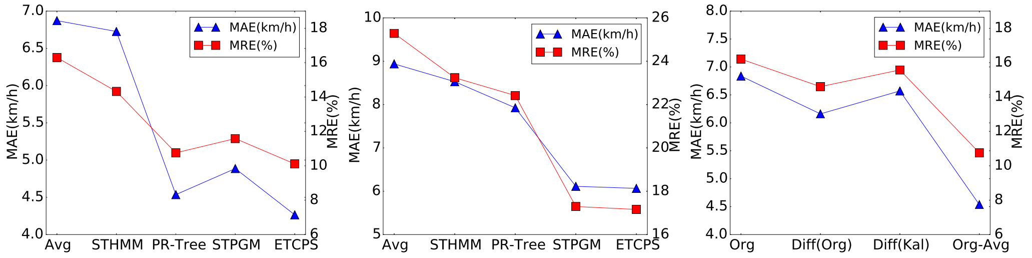
② This data can be downloaded in http://www.datatang.com/data/45422 taxis. For the boarding state, the speed of the taxi usually varies sharply in a short time. Therefore, once we detect such sharp variation of the speed, we eliminate such GPS records. To handle the balling state, for a specific road, we check the speeds of all taxis in this road in a specific time t. If the speeds of most taxis are relatively high, only few of the taxis are driving at a very low speed , we think such taxis are on the balling state and we eliminate the corresponding GPS records.

Deal with Sparsity Recall that as we claimed in Section 5.2, some road segments may not contain any GPS record during the time interval t for some $t \in [T]$. Thus, the corresponding traffic condition o_t^i is not defined. To solve this issue, for the road segment r_i, if the GPS record set observed in the time interval t is not empty, we define $\bar{o}_t^i$ as the average speed of the GPS records in the t-th interval. Otherwise, we have $\bar{o}_t^i = -1$. Let $\overline{A_t^i} = \{\bar{o}_{t'}^i | t' \equiv t \bmod M \wedge \bar{o}_{t'}^i \neq -1\}$ indicate the traffic conditions during the t mod M-th interval in each day. We define $\bar{a}_t^i$ as the mean of $\overline{A_t^i}$ and the series Bias $= \{b_t = \bar{o}_t^i - \bar{a}_t^i | \forall \bar{o}_t^i \neq -1\}$. Then, for each pair of adjacent elements in Bias, we perform the linear interpolation to obtain the undefined b_i. For example, if $Bias = \{b_1 = 3, b_4 = 4.5, b_7 = 10.5\}$, we obtain a series $\{b_1 = 3, b_2 = 3.5, b_3 = 4, b_4 = 4.5, b_5 = 6.5, b_6 = 8.5, b_7 = 10.5\}$ after performing linear interpolation. Finally, we have that the traffic condition o_t^i is obtained by $o_t^i = \bar{a}_t^i + b_t$.

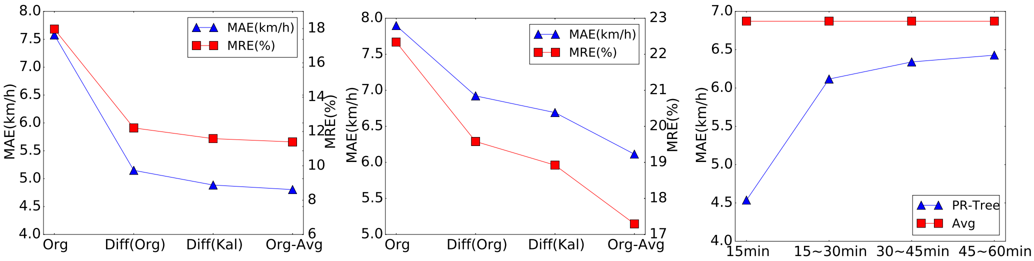
5.8.3 Performance Evaluation

Performances of different models We present the evaluations of our models. We first compare our model with the baseline Avg, i.e., predict the traffic condition o_t^i by its expected value a_t^i. Furthermore, in the recent work, Yang et al.[57] proposed STHMM for traffic condition prediction which is based on a spatial temporal hidden markov model. We compare STHMM with our models as well.

The results are shown in Fig. 5.8(a) and Fig. 5.8(b). As we can see, the baseline (Avg) performs worst in both cases. Despite that STHMM outperforms Avg in both cases, both of our models PR-Tree and STPGM perform better than STHMM in our data set. Moreover, in the standard case, PR-Tree performs better than STPGM as shown in Fig. 5.8(a) whereas in the sparse case STPGM performs better. By combining



(a) All models (standard) (b) All models (sparse) (c) PR-Tree (standard)



(d) STPGM (standard) (e) STPGM (sparse) (f) Predict longer

Figure 5.8 Performance Analysis.

PR-Tree and STPGM, our system ETCPS achieves the best performance in both two cases.

Verifying the Observed Patterns Recall that as we claimed in Section 5.7, any time series related with traffic can be taken as the input of both PR-Tree and STPGM, and predict the traffic condition in the proper way. To illustrate the effects of the observations which we proposed in Section 5.3, we design four different experiments with different time series and evaluate each experiment on PR-Tree and STPGM respectively. The first two time series are Org and Gap = Org − Avg, as we used in Section 5.5 and Section 5.6. Then, we use the first order difference of Org as the input time series, denoted as Diff(Org). The t-th element in Diff(Org) is $o_{t+1} - o_t$. Furthermore, since the raw GPS records usually contain the noise such as the GPS drift, we use Kalman filtering to process the traffic condition series Org. We take the first order difference of the processed time series as the input as well, denoted as Diff(Kal).

We show the experimental results in Fig. 5.8(c), Fig. 5.8(d) and Fig. 5.8(e). Both PR-Tree and STPGM perform badly if we use Org as input directly. However, by using Diff(Org) and Gap instead, the performances improve significantly which verifies our

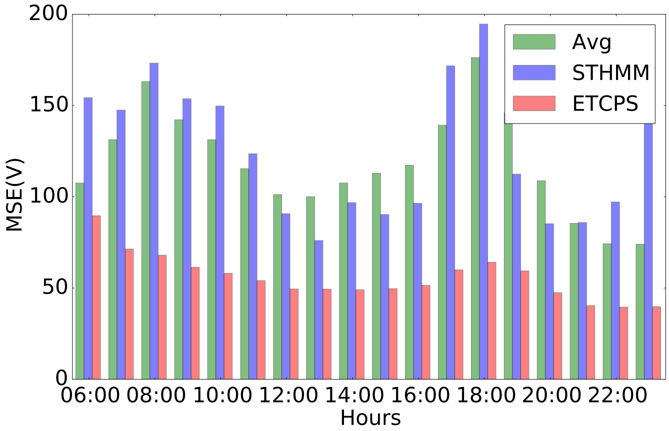


Figure 5.9 MSE varies over day

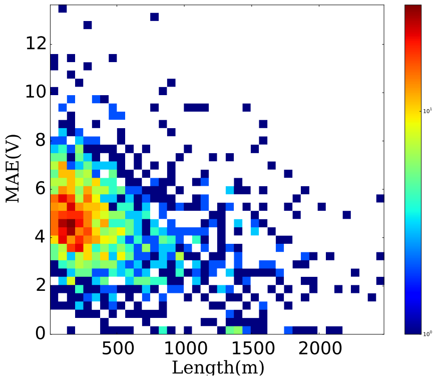


Figure 5.10 MAE varies over road length

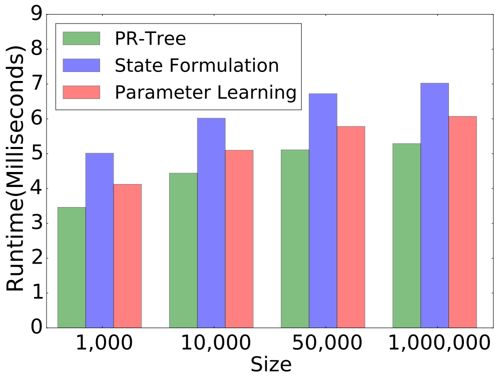


Figure 5.11 MSE varies over day

observations.

Predict longer time intervals The PR-Tree model can be also used to predict the traffic conditions in the longer term. Given observations in interval t denoted as o_t, we first obtain the predicted traffic condition p_{t+1} and we take p_{t+1} as the "true traffic condition" in the time interval $t + 1$ and obtain p_{t+2}. Iteratively, we obtain the prediction after m time intervals p_{t+m}. In Fig. 5.8(f), we show the performance of PR-Tree in predicting the traffic condition in the next 0 to 60 minutes and comparing with the Avg method. As m increases, the performance becomes worse, but it is still better than Avg.

Effects of time and road length

Fig.5.9 shows the effectiveness of our prediction across time. We plot the average

Size	PR-Tree	State Formulation	Parameter Learning
10^3	0.04	1.72	0.22
10^4	0.46	17.46	2.09
$5 * 10^4$	2.14	88.1	10.09
10^5	3.26	176.6	19.61

Figure 5.12 table: Time cost (Minutes)

mean squared error of travel speed (MSE) for the baseline Avg, STHMM and ETCPS respectively during different hours for all days. The result shows that our system outperforms both the baseline and STHMM.

To illustrate the effectiveness of the road length, in the Fig. 5.10, we show the relation between MAE and the length of road segments. The result shows that the road segments with longer length tend to have smaller MAE, i.e., our prediction performs better for the road segments with longer lengths.

Running time Since the predictions of both PR-Tree and STPGM are simple which can be done in real time, we only present the running time for training our models in Fig. 5.11 and Tab. 5.12. From Table 5.12, we can see that the training time cost of PR-Tree is very small. It takes only 3.26 minutes to process 10^5 roads. However, STPGM takes a much longer time to train as shown in Table 5.12. Especially for the state formulation phase, clustering the traffic conditions is time costing. It takes 176.6 minutes to process the state formulation phase for 10^5 roads. We stress that SHTMM applies a complicated state formulation algorithm and the state space is much larger than STPGM. In our data set, the time consuming of SHTMM is 1718 ms per road whereas even for STPGM, it only takes 13.3ms per road to train the model.

5.9 Conclusion

We study the effective and scalable methods for traffic condition prediction. We propose an Ensemble based Traffic Condition Prediction System (ETCPS) which combines two novel models called Predictive Regression Tree (PR-Tree) and Spatial Temporal Probabilistic Graphical Model (STPGM). Our model is based on two useful observed correlations in the traffic condition data. Our system provides high-quality prediction and can easily scale to very large datasets. We conduct extensive experiments to evaluate our proposed models. The experimental results demonstrate that comparing with the existing methods, ETCPS is more efficient and accurate. We extend the future direction of this work in the Chapter 7.2.

第 6 章 Estimating Travel Time Based on Recurrent Neural Networks

The travel time estimation is an important yet challenging problem. It is a fundamental ingredient of many location-based services such as navigation, route planning systems etc. In this chapter, given a path and the corresponding start time, we study the problem of estimating the time for traveling the path. Prior work usually focuses on estimating the travel times of individual road segments or sub-paths, and then summing up these estimated travel times. However, such approach leads to an inaccurate estimation, since the travel time is also affected by the number of road intersections or traffic lights in the path, and the estimation errors for individual road may accumulate. We propose an end-to-end framework for Travel Time Estimation called *DeepTTE*. Our model estimates the travel time of the whole path directly, based on deep recurrent neural networks. In our model, we consider the spatial and temporal dependency in the path as well as various factors which may affect the travel time such as the driver's habit, the day of the week etc. We conduct extensive experiment result on a large scale dataset. The experiment result shows that our model significantly outperforms the other existing methods.

6.1 Introduction

Estimating the travel time for a given path is a fundamental problem in route planning, navigation, and traffic dispatching. An accurate estimation of travel time helps people better planning their routes. Almost all the electronic maps and online car-hailing services provide the travel time estimation in their apps, such as Google Map, Uber, Didi, etc. When a user searches the routes to the destination, the map app provides several candidate routes with estimated travel times (and possibly other measures such as gas consumptions, tolls) for the user to decide. Although the problem has been widely studied in the past years, providing an accurate travel time is still a

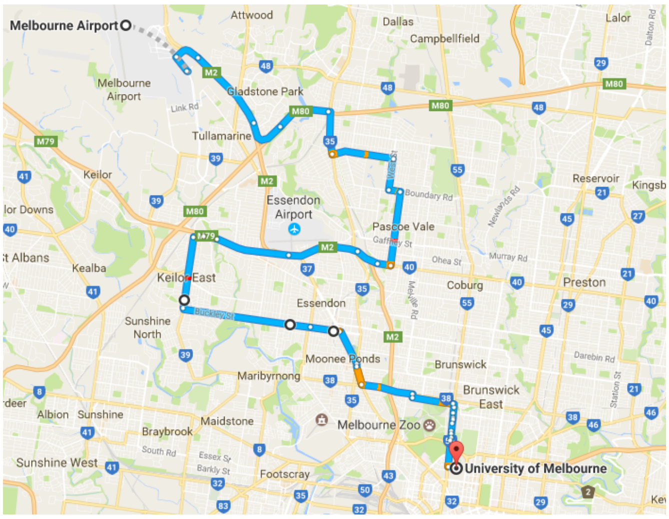


Figure 6.1　An illustration of the query path.

challenging problem. Prior work usually focuses on estimating the travel speed/time of an individual road segment[57,62,64]. However, 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 etc. Simply summing up the travel time of the road segments in the path does not lead to an accurate estimation, as the errors may accumulate[65]. Alternatively, some work decomposes the path into several longer sub-trajectories instead of the road segments and estimates the travel time based on the sub-trajectories[59]. Although such method enhances the estimation accuracy, it suffers from the data sparsity problem since there are many sub-trajectories that were visited by very few drivers.

In this chapter, we view a path as a sequence of location points (see Fig. A.4 for an illustration), and we learn to estimate the travel time from historical trajectories based on deep learning approach. To make our exposition more concrete, we first present some challenges in our problem.

- To estimate the travel time, we have to consider the spatial and temporal dependences in the given trajectory at the same time. Prior work usually formalizes such dependence by discretizing the trajectory into several grids[74] or road segments[57]. However, on one hand discretizing the GPS points into grids causes the information loss due to the coarse granularities. On the other hand, inferring the travel time based on the individual road segments misses the effects of road

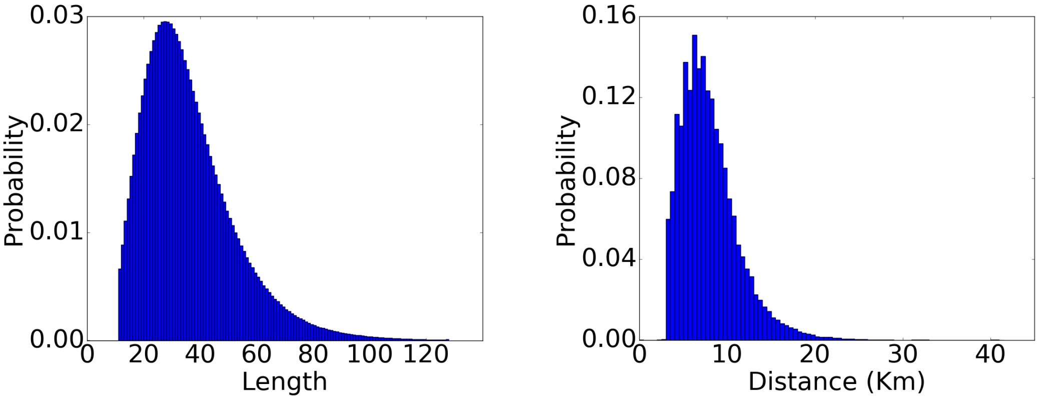


Figure 6.2 Length and Distance Distribution

intersections and traffic lights. Few work studies capturing the spatio-temporal dependence of GPS points directly.

- The travel time of a specific path can be very different at different time intervals. For example, in the peak hours, it usually takes much longer time than that in non-peak hours. Even for a fixed time interval, different days of the week reveal very different distributions of travel times. Prior work usually builds several sub-models for different days of the week[8,9]. Such implementations, on one hand, make the model tedious, on the other hand, each sub-model only utilizes a small part of data which may suffer from the lack of training data.
- Different drivers have different driving habits. The experienced drivers are usually very familiar with the traffic conditions in the city and drive very fast. On the contrary, the new drivers usually drive relatively slow which leads to a longer travel time under the same condition. Most of the prior work does not consider the effects of drivers (the driver information is available in our dataset) when estimating the travel time.
- Different historical trajectories have very different values of length (i.e., the number of points) or distance. Fig. A.5 shows the distribution of the values of trajectory length and distance in our dataset. It is not easy to process the variable-length trajectories directly with the traditional machine learning models such as Random Forest, Gradient Boosting, etc.

To address the above challenges, we propose an end-to-end framework, based on deep recurrent neural networks. The primary contribution of this chapter can be summarized as follows:

- We design an end-to-end framework for Travel Time Estimation, called *DeepTTE*, based on deep recurrent neural networks. We incorporate various factors which may affect the travel time (e.g., the driver, the day of the week, and the time interval, etc.) in a unified model, instead of building several sub-models manually.
- We devise a novel neural network architecture which can easily process variable-length GPS trajectories. Furthermore, by carefully designing the input sequence, our model effectively captures the spatial and temporal dependence in the trajectory simultaneously without much information loss.
- We further extend our model to a *multi-task learning* model by introducing an auxiliary component. The auxiliary component estimates the travel time between each pair of adjacent GPS points which we take as the auxiliary output. We show that the auxiliary component effectively improves the model performance.
- We conduct extensive experiments on a large scale data set which consists of GPS points generated by over $14,684$ taxis collected in one month in Chengdu, China. Our model achieves a high-quality prediction result with the error rate of 12.74% which significantly outperforms several other off-the-shelf machine learning algorithms, as shown in Section 6.4.

This chapter is organized as follows. In Section A.5.2, we formally define our problem and present several preliminaries of our model. In Section A.5.3, we describe our model architecture in detail. We conduct extensive experiments to show the strength of our model in Section 6.4. Finally, we present some related work and conclude this chapter in Section A.2.3 and Section 6.5.

6.2 Problem Definition

Definition 8 (Historical Trajectory)： We define a *historical trajectory* T as a sequence of consecutive historical GPS points, i.e., $T = \{p_1, \ldots, p_{|T|}\}$. 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 corresponding driver ID which we denote as driverID.

We then illustrate the our objective.

Definition 9 (Objective)： Given the path S, the corresponding driver ID and the start time, our goal is to estimate the travel time from the source to the destination through S. We assume that the travel path S is specified by the user or generated by the route planing apps. S to a sequence of location points by sampling. Each location is represented as a pair of longitude and latitude.

Remark: In our experiment, we remove the timestamp in the historical trajectories and use such trajectories as the test data. During the training phase, we learn how to estimate the travel time of the given path, based on the historical trajectories as we defined in Definition 10. During the test phase, to make the testing data consistent with the training data, we convert the path S to a sequence of location points by sampling. In this chapter, we do not consider how to optimize the path S.

6.3 Model Architecture

We first describe the architecture of our model as shown in Fig. A.6. Our model consists of four parts: the *attribute* component, the *sequence learning* component, the *residual* component, and the *auxiliary* component.

The *attribute* component processes the attributes of the driver ID, the current day of the week, and the timeslot of the start time. The *sequence learning* component processes the GPS location sequence. The *residual* component utilizes the outputs of the first two components to estimate of the given path. Finally, we introduce an *auxiliary* component which estimates the travel times between consecutive GPS points and helps improve the model performance.

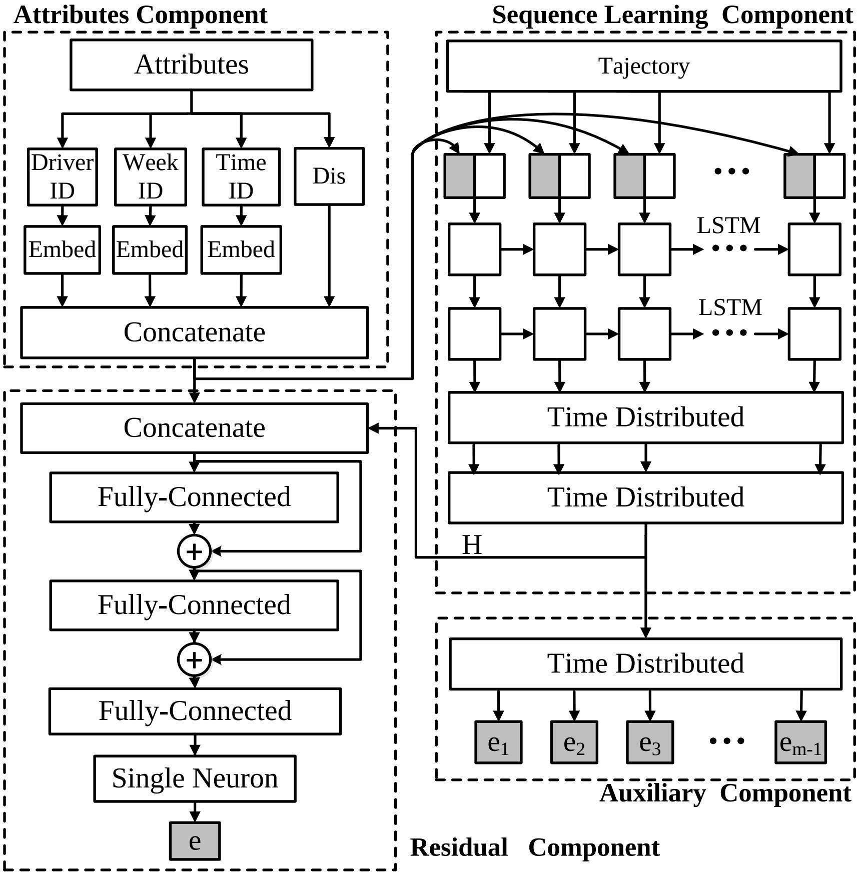


Figure 6.3 Overview of DeepETT.

6.3.1 Attribute Component

In the attribute component, we first process the attributes of the driver ID, the day of the week and the timeslot of travel start① . We use driverID, weekID and timeID to denote these three attributes respectively. Since these attributes are categorical, to feed the attributes into the neural network, we need to transform them into real values. We use the embedding method[38] to process each categorical attribute. The embedding method maps each categorical value $v \in [V]$ to a low-dimensional space $\mathbb{R}^{E\times 1}$ by a matrix $W \in \mathbb{R}^{V\times E}$ (we refer to such space as the embedding space). An important property of embedding method is that the categorical values with similar semantic meaning are usually very close in the embedding space. Thus, the embedding method helps us dis-

---

① We divide one day into 1440 timeslots.

cover the similarities in the data[38]. In our case, for example, for some specific paths, driving at 7:00 A.M. usually takes similar time with that at 18:00 P.M. since they are both peak hours. Thus, we use the embedding method to reduce the computational cost and help us discover such similarities in the data automatically.

We further consider the attribute of *travel distance*. We denote $\Delta d_{p_a \to p_b}$ as the travel distance from GPS point p_a to p_b, i.e., $\Delta d_{p_a \to p_b} = \sum_{i=a}^{b-1} \mathsf{Dis}(p_i, p_{i+1})$ where Dis is the geographic distance between two GPS points. Then, we concatenate the embedded vectors of driverID, weekID and timeID together with the travel distance $\Delta d_{p_1 \to p_n}$ to form the output of the attribute component, which we denote as *attr*.

6.3.2 Sequence Learning Component

In this part, we demonstrate how the sequence learning component extracts the compressed information from the trajectory using a novel architecture.

Recall that each trajectory T is represented by a sequence of GPS points $\{p_1, \ldots, p_{|T|}\}$. Since different trajectories have different lengths, to handle the sequences with variable lengths, we propose two candidate processing methods.

6.3.2.1 Sampling Method

We randomly sample each trajectory to a fixed length m, as shown in Fig. 6.4. We denote the indices of the sampled points as an ordered list L and the sampled trajectory as T'. Furthermore, to make sure that the source and the destination are included in T', we have that $L_1 = 1$ and $L_m = |T|$. Thus, the sampled trajectory T' can be represented as $\{p_{L_1}, \ldots, p_{L_m}\}$.

Recall that the trajectory contains both the temporal and spatial dependencies. To capture the spatio-temporal dependency, we use two stacked LSTM layers to process the GPS point sequence. A direct way is to take the longitude and the latitude of point p_{L_i} to the LSTM layers at each time step i. However, discovering the relative position and the distance between two GPS points is not easy for neural networks. Instead, for each time step i, we concatenate the coordinates of two consecutive points $p_{L_i}, p_{L_{i+1}}$, the travel distance between p_{L_i} and $p_{L_{i+1}}$ (which we denote as $\Delta d_{p_{L_i} \to p_{L_{i+1}}}$) and the output

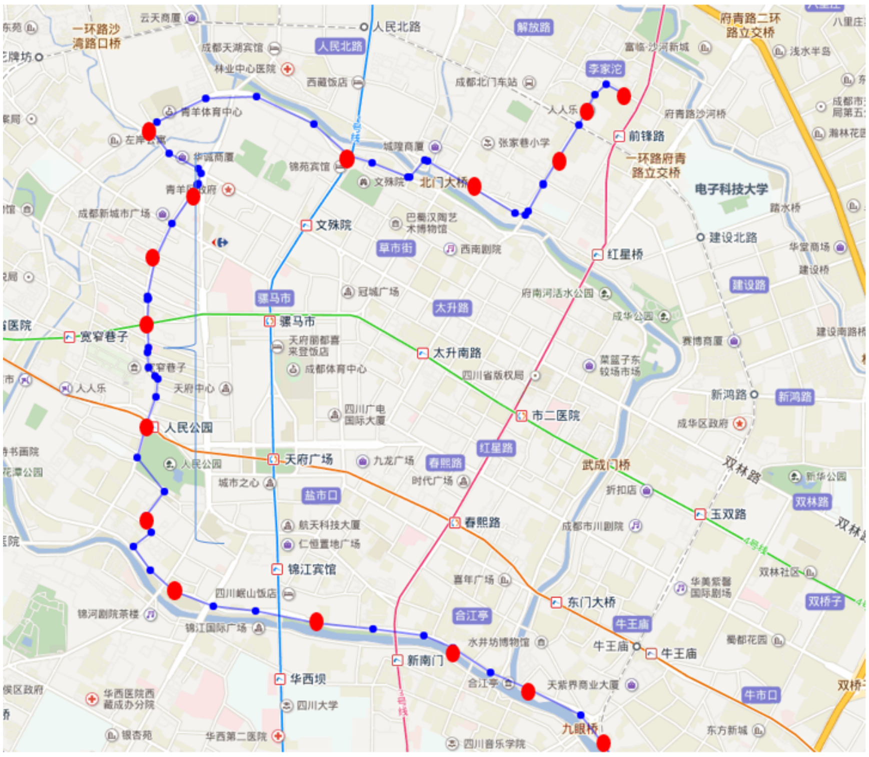


Figure 6.4　Sampling m Points from Trajectory.

feature vector *attr* obtained from the attribute component. We use such concatenation $x_i = (lat_i, lon_i, lat_{i+1}, lon_{i+1}, \Delta d_{p_{L_i} \to p_{L_{i+1}}}, attr)$ as the input of LSTM layers.

We pass the input sequence to two stacked LSTM layers and obtain the output sequence $\{y_1, y_2, \ldots, y_{m-1}\}$. Then, we use two stacked *time-distributed (fully-connected)* layers to map each y_i to a hidden state vector h_i. A time-distributed fully-connected layer takes a sequence of vectors and maps each vector to an output vector using the same mapping function. We finally concatenate the hidden state sequence $\{h_1, h_2, \ldots, h_{m-1}\}$ into a vector H, where H indicates the representation vector of the trajectory.

6.3.2.2　Pooling Method

Alternatively, since the recurrent neural network can process the variable length sequences, we can also use the original trajectory T directly without sampling. In such implementation, instead of using the concatenate layer, we use a *mean pooling* layer as our merge layer, i.e., $H = \frac{1}{|T|-1} \sum_{i=1}^{|T|-1} h_i$ and obtain the representation vector in the same way. We compare these two different sequence processing implementations in the experiment part.

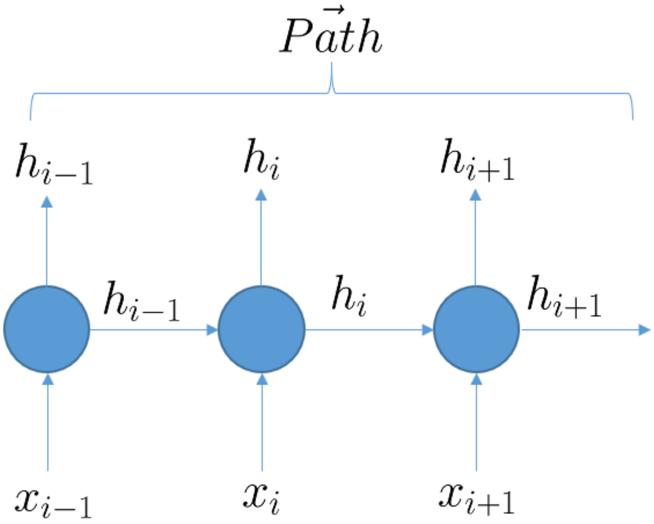


Figure 6.5 Learn the presentation of the whole trajectory.

6.3.2.3 Discussion

When a new point pair $x_i = (p_i, p_{i+1})$ given to a LSTM neural, we deal it with the first $i-1$ points pairs together, instead of viewing it as a isolated pair. To be more concrete, as shown in the Fig. 6.5, when the LSTM neural transform the x_i to h_i, it will also consider the abstract of the first $i-1$ points pairs, e.g. h_{i-1}. Finally, we concatenate the $\{h_1, \ldots, h_m\}$ together as a high dimension feature vector. Such vector is a presentation of the whole trajectory.

6.3.3 Residual Component

The residual component simply concatenates all the obtained feature vectors from sequence learning component and the attribute component. It thus forms a high-dimensional vector which indicates the representation of the trajectory and the related information (driver ID, the day of the week, etc.). Then, we use three residual layers to extract the feature vectors from this high-dimensional vector. Finally, we use a single neuron with linear activation to obtain the estimated travel time of the given path.

6.3.4 Auxiliary Component

In fact, the residual component is already available to estimate the travel time of the given path. However, it only considers the "global information"of the whole trajectory but ignores the travel time of sub-trajectories which we refer to as the "local information". Note that during the training phase, since the time stamps of all the GPS points $p.ts$ are available, we can easily infer the local information of the trajectory.

To utilize the local information, a feasible way is to estimate the travel time between each pair of consecutive GPS points and sum up the travel time of all pairs. However, on one hand, the travel time between the consecutive points is usually very small but has a large variance. Estimating such small values is difficult and leads to a very inaccurate result. On the other hand, since we only care the estimation accuracy of the whole trajectory, it is not necessary to accurately estimate the travel time of each individual pair.

Instead, we introduce an auxiliary component to make use of the local information, and we take the estimated travel time between the point pairs as the "auxiliary output". Formally, we denote the travel time of a sub-trajectory $p_a \rightarrow p_b$ as $\Delta t_{p_a \rightarrow p_b}$. For convenience, suppose we adopt the sampling trick in the sequence learning component. Based on the first three components, the auxiliary component receives the hidden state sequence $\{h_1, \ldots, h_{m-1}\}$ from the sequence learning component. Then it maps the hidden state sequence into a real value sequence $\{e_1, e_2, \ldots, e_{m-1}\}$ using a time-distributed fully-connected layer. Each real value e_i corresponds to the estimated travel time $\Delta t_{p_{L_i} \rightarrow p_{L_{i+1}}}$ where L is the indices of sampled points as we defined in Section 6.3.2. We use the estimated travel time sequence $\{e_1, e_2, \ldots, e_{m-1}\}$ as the auxiliary output of our neural network. The auxiliary output is trained together with the estimated travel time of the whole trajectory. See Section 6.3.5 for more details.

6.3.5 Model Training

Our model is trained end-to-end. We use the *mean absolute percentage error (MAPE)* as our objective function. Formally, suppose we adopt the sampling trick in the sequence learning component. Then, the loss function of sequence learning component is defined as

$$\text{loss}_{seq} = |e - \Delta t_{p_1 \to p_{L_m}}| / \Delta t_{p_1 \to p_{L_m}}.$$

For the auxiliary component, the loss function is the average MAPE loss of each time step, i.e.,

$$\text{loss}_{aux} = \frac{1}{m-1} \sum_{i=1}^{m-1} \frac{|e_i - \Delta t_{p_{L_i} \to p_{L_{i+1}}}|}{\Delta t_{p_{L_i} \to p_{L_{i+1}}} + \epsilon}. \tag{6-1}$$

Note that in Eq. (6-1) we use a small factor ϵ to prevent the exploded loss values when $\Delta t_{p_{L_i} \to p_{L_{i+1}}} \to 0$.

The loss function we used during the training phase is defined as the weighted sum of loss_{seq} and loss_{aux}, i.e.,

$$\text{loss} = \text{loss}_{seq} + \alpha \cdot \text{loss}_{aux} \tag{6-2}$$

where α is the weight factor which is specified in the experiment section. Note that the auxiliary loss is just used to improve the accuracy, we still use loss_{seq} to evaluate our model performance.

Thus, we can train our model by the standard backpropagation and gradient descent method.

6.3.5.1 Discussion

Learning to estimate the travel time of the whole trajectory and the travel time sequence of consecutive location pairs at the same time is one of the advantages of using deep learning model. Even though our original problem is to estimate the travel time of the whole path, we convert it to a *multi-task learning* problem, and optimize multiple objectives with a shared neural network architecture. This technique allows us to fully utilize the information contained in the training data. Multi-task learning has been widely studied in the computer visions and the natural language processing problems[91–93]. It is unclear how to implement such multi-task learning using traditional machine learning techniques such as the random forest, or gradient boosting, in our setting.

6.4 Experiments

In this section, we report our experimental results on the real world dataset. We first describe the experimental setting and the details of dataset in Section 6.4.1. We then compare our model with several baseline methods in Section 6.4.2. In Section 6.4.3 to Section 6.4.5 we present the effects of different components and parameters.

6.4.1 Experiment Setting

6.4.1.1 Data Description

Our dataset consists of 1.4 billion GPS records of 14864 taxis from 2014/08/03 to 2014/08/30 in Chengdu, China① . Each record contains three attributes: the longitude, the latitude and the corresponding time stamp. During our experiment, all the driver id were anonymized by recoding. The total number of trajectories is $9,653,822$. The shortest trajectory contains only 11 GPS location points (2km) and the longest trajectory contains 128 GPS location points (41km).

We use the last 7 days (from 24th to 30th) as the test set and the remaining ones as the training set.

6.4.1.2 Parameter Setting

We present the parameter settings of different components.

Attribute Component: We embed driverID to $\mathbb{R}^{1\times16}$, weekID to $\mathbb{R}^{1\times3}$ and timeID to $\mathbb{R}^{1\times4}$.

---

① The dataset and the corresponding code can be downloaded at https://github.com/DeepTTE/DeepTTE

Sequence Learning Component: We set the dimension of each LSTM internal state as 128 and the dimensions of two time-distributed fully-connected layers as 128 and 64 respectively. We use leaky relu function (LReLU)[81] as the activation of the time-distributed fully-connected layers. Moreover, we test our model under different sampling rates (i.e., the number of sampled points) when we adopt the sampling trick. See Section 6.4.3 for more details.

Residual Component: We set the dimension of all three residual layers as 128. The activation functions of the residual layers are all leaky relu function.

Furthermore, in the auxiliary loss in Eq.(6-1), we set $\epsilon = 10$. In Eq. (6-2), we set the weight factor α as 3.0. We fix the batch size of our model as 512 and we adopt *Adam*[82] optimizer with learning rate 0.001 to train our model. Our model is trained by 40 epochs.

To evaluate our model, we use 5-fold cross-validation in the training set. For each fold, we select the best model based on the validation. We thus obtain 5 best models. To estimate the travel time on the test set, we use each selected model to obtain an estimation respectively and average the estimations as our final result. The final estimation is evaluated by MAPE as we mentioned in Section 6.3.5.

6.4.1.3 Experiment Environment

Platform: Our model is trained on the server with one GeForce 1080 GPU. We implement our model with Keras 0.8.2 (Theano backend), a widely used Deep Learning Python library.

6.4.2 Performance Comparison

To demonstrate the strength of our model on estimating the travel time, we compare our model with several popular machine learning methods. Since part of the baseline methods cannot process the sequences with variable lengths, to make a fair comparison, we first sample each trajectory to a fixed length of 30. The methods are shown as follow:

Table 6.1 Performance Comparison of Baseline Methods

Model	MAPE
Gradient Boosting	20.32%
MLP-3 layers	16.17%
MLP-5 layers	15.75%
Vanilla RNN	18.85%
DeepTTE	**13.14%**

1. **G**radient Boosting Decision Tree (GBDT): Gradient Boosting Decision Tree is a powerful and widely used ensemble method[13]. To estimate the travel time using GBDT, we concatenate the all the attributes contained in our attribute component and the input sequence in our sequence learning component. We use the concatenated vector as the input of GBDT. In our experiment, we use XGBoost, a widely used GBDT library[78]. The optimal parameters are achieved by the grid search.
2. **Multi-Layer Perceptron (MLP)**: A multi-layer perceptron is a fully-connected neural network with multiple hidden layers. We test our data with a 3-layer MLP (with 3 hidden layers) and a 5-layer MLP respectively. The input vector of MLP is the same as the input of GBDT. The dimension of each hidden layer is fixed as 128 and the activation is leaky relu.
3. **Vanilla RNN**: We further compare our model with a vanilla RNN architecture. Each time step, the vanilla RNN receives the coordinates of p_{L_i} and $p_{L_{i+1}}$ as well as the corresponding travel distance between them. Similarly, we build an attribute component. We concatenate the output the attribute component and RNN, and we pass the concatenation to a 3-layer perceptron to obtain the estimated travel time.

We show the experiment result in Table 6.1. As we can see, the GBDT results in a large error of 20.32%. We stress that although GBDT is a powerful and widely used method, it cannot capture the temporal dependency in the data. Furthermore, GBDT relies on carefully hand-crafted features. However, extracting useful features from the GPS point sequence is not easy. For vanilla RNN, it considers the temporal

Table 6.2　Performance of Different Sampling Capacity

Sampling Capacity	MAPE	Time (per epoch)
DeepTTE-10	15.45%	674s
DeepTTE-30	13.14%	1729s
DeepTTE-70	13.02%	3879s
DeepTTE-100	**12.74%**	5484s
DeepTTE-Var	12.87%	5841s

dependency between GPS location points but it failed to process the long sequence due to the gradient vanishing problem, as we mentioned in Section A.5.1.

6.4.3 Effects on the Sampling Rate

We compare the performances of DeepTTE when we use different sampling rates (i.e., the length m of sampled trajectories as we defined in Section 6.3.2). We use x to denote the lengths of sampled trajectories and DeepTTE-x to denote the corresponding model. We further compare the performance of DeepTTE if we use the pooling trick, which we denote as DeepTTE-Var. The experiment result is shown in Table 6.2.

The result shows that enlarging the sampling rate increases the estimation accuracy of our model. When the sampling rate is 100, our model achieves the best performance of 12.74%. However, using large sampling rate also increases the training time. For DeepTTE-100, it takes about 1.5 hours to train a single epoch. Choosing a proper sampling rate is a trade-off between the speed and accuracy.

DeepTTE-Var achieves the estimation accuracy of 12.85% which is slightly worse than DeepTTE-100. Although it utilizes all the information of the original trajectory T (recall that the max length of T is 128), it causes the information loss when we simply averaging the hidden state by the mean pooling layer.

6.4.4 Effects of the Auxiliary Component

Recall that in our model, we introduce an auxiliary component to estimate the travel time between consecutive GPS points. To verify the effectiveness of the auxil-

Table 6.3 Effects of Attribute Component

Experiment Setting	MAPE
DeepTTE-30	**13.14**%
Eliminate driverID	13.37%
Eliminate weekID	13.58%
Eliminate both	13.59%

iary component, we eliminate the auxiliary component in DeepTTE-30 and train our model under the same condition. The experiment result shows that the estimation error increases from 13.14% to 13.95% dramatically.

Furthermore, if we directly predict the travel time between the consecutive GPS points and use the summation as our estimation, the MAPE is as high as 28.44%.

6.4.5 Effects of the Attribute Component

To show the effects of the attribute component (driver ID, day of the week), we compare the performance of DeepTTE-30 under three different settings. In the first setting, we eliminate the day of the week in the attribute component. In the second setting, we eliminate the driver ID. In the last setting, we eliminate both two attributes and only keep the start time and the travel distance in the attribute component. The experiment result is shown in Table 6.3.

The result shows that eliminating any attribute in our model leads to a reduction of estimation accuracy. As we can see, dropping out the day of the week increases the estimation loss dramatically. Moreover, if we eliminate the driver ID, the MAPE loss increases from 13.14% to 13.37% which verifies that the driving habits affects the travel time estimation, as we mentioned in Section A.5.1.

6.5 Conclusion

In this chapter, we study estimating the travel time of a given path. We propose an end-to-end framework based on deep recurrent neural networks. Our model effectively captures the spatial and temporal dependencies in the given path at the same time. Furthermore, our model considers various factors which may affect the travel time such as the drivers, the day of the week etc. We conduct extensive experiments on a very large scale real-world dataset. The experimental result shows that our model achieve a high estimation accuracy and outperforms the other off-the-shelf methods significantly.

第 7 章 Conclusion

Currently, the wide usage of location based services and GPS embedded devices have changed people's life style. For example, people use their smart phones to plan their routes, call for car-hailing services, find the trip partners and search the destinations etc. A variety of massive spatial temporal data is generated routinely (e.g., vehicle mobility, traffic patterns, online car-hailing data and geo-tagged check-in data etc). The rapid expansion of both urban population and volume of vehicles has lead to the commute demands in these cities increase sharply. People suffer the traffic congestion and the difficulty in getting cabs. Analyzing the large volume of location based data brings new opportunities for discovering valuable information. It enables the governments to do the traffic analysis and urban planning, which in turn can alleviate the traffic congestion and difficulty in taking cabs. Motivated by this, an increasing number of papers empowered recently.

In this chapter, we will first summarize the contributions of this thesis, and discuss how the frameworks/systems provided in this thesis can be applied to problems outside the scenarios considered. Then, we will talk about the future research directions of the prediction over massive spatio-temporal traffic data.

7.1 Summary of the Thesis

In this thesis, we focus on learning and prediction over the massive spatio-temporal traffic data.Three specific problems are investigated, including car-hail supply demand prediction, traffic condition prediction and travel time estimation. These three problems are quite related, but different. They are the base of building a better intelligent transportation system (ITS) to alleviate the traffic congestion and improve the user experience in daily commute. The ITS recommend people better route plans to avoid the congestions roads, based on the prediction of traffic conditions and travel time estimation of given path. Meanwhile, by forecasting the commute demands, the

ITS can dispatch the taxis to balance the supply-demand in advance and reduce the gas consumptions of no-load taxis.

In the Chapter A.3, we study the problem of predicting the real-time car-hailing supply-demand. We propose an end-to-end framework called *Deep Supply-Demand (DeepSD)*, based on a novel deep neural network structure. Our approach automatically discovers the complicated supply-demand patterns in historical order, weather and traffic data, with minimal amount of hand-crafted features. We presented two versions of DeepSD: a basic version and an advanced version. The basic version simply use the real-time car-hailing order data whereas the advanced version further incorporate the historical car-hailing data with a more complex structure. We conduct extensive experiments on a real-word dataset from Didi. The experimental results show that our model outperforms the existing methods significantly. Furthermore, our model is highly flexible and extendible. We can easily incorporate new data sources or attributes into our model without re-training.

In the Chapter A.4, we study the effective and scalable methods for traffic condition prediction. We propose an Ensemble based Traffic Condition Prediction System (ETCPS) which combines two novel models called Predictive Regression Tree (PR-Tree) and Spatial Temporal Probabilistic Graphical Model (STPGM). Our model is based on two useful observed correlations in the traffic condition data. Our system provides high-quality prediction and can easily scale to very large datasets. We conduct extensive experiments to evaluate our proposed models. The experimental results demonstrate that comparing with the existing methods, ETCPS is more efficient and accurate. In the future, we plan to infer the traffic conditions by incorporating more features from heterogeneous data sources, such as the weather condition, POI information etc. Next, we will focus on the efficient way to deal with road segments which have extremely sparse trajectory records. Furthermore, we plan to try different ensemble methods to combine the different models in order to enhance the performance of the prediction.

In the Chapter A.5, we study estimating the travel time of a given path. We propose an end-to-end framework for $\underline{T}$ravel $\underline{T}$ime $\underline{E}$stimation called *DeepTTE* based on deep recurrent neural networks. Our model effectively captures the spatial and temporal dependencies in the given path at the same time. Furthermore, our model considers various factors which may affect the travel time such as the drivers, the day of the week etc. We conduct extensive experiments on a very large scale real-world dataset. The experimental result shows that our model achieve a high estimation accuracy and outperforms the other off-the-shell methods significantly.

Note that, the frameworks/systems provided in this thesis can be easily applied to problems outside the scenarios we considered. For example, in the Chapter A.3, we proposed an end-to-end framework called *Deep Supply-Demand (DeepSD)* to predict the supply-demand for online car-hailing services. In fact, supply-demand prediction is widely studied in many fields, such as the logistics transportation, the commodity retailing, the photovoltaic power generation. To be more concrete, we take the logistics transportation as an example. The logistics companies, such as UPS, FedEX, and EMS, need to maintain networks of warehouses for each city, to pick-up or deliver packages. Each warehouse is responsible for a certain area. However, the number of packages need to be handled varies dynamically due to different warehouses and time intervals. For example, some large warehouses deliver at least $10,000$ packages each day, whereas some warehouses in small-scale only deliver at most 50 packages. Furthermore, the number of packages need to be handled under different time intervals of a day can be extremely different. Obviously, an accurate prediction of the number of packages to be handled for different warehouses and time intervals helps scheduling the employees efficiently, which can greatly improve productivity and save cost. In fact, such problem is quite similar to the supply-demand prediction for online car-hailing services. Currently, we are cooperating with a major logistics transportation company in China. We help them to predict the number of packages to be handled for different warehouses in each time intervals a month in advance. The framework we propose in the Chapter A.3 seems also quite useful in this project.

7.2 Future Directions

There are still a few aspects which can be further optimized. For example, more data sources should be involved in the prediction framework/system. The spatio-temporal traffic data is affected by the complex environments, especially the unexpected events (e.g. extreme weather, traffic accidents, activities, etc.). Event detection based on the search data or the social network data should be added to the prediction system to improve the prediction accuracy in the exceptional situations.

Another interesting avenue would be to explore the feature vectors we extract from trajectories with recurrent neural network. In the chapter A.5, we discuss using the recurrent neural network to represent the whole GPS trajectories as a high dimension feature vector. Although, in that work we use the vector to estimate the travel time of the path, we think the information that the feature vector contains is far more than that. For example, the feature vector may contain the information of driving style of the driver, habit of the trajectory owner, and traffic condition of the roads that the trajectory passed etc. Based on the trajectory feature vector, we can learn the similarity between different trajectories to detect the social relationships between the owners of trajectories.

Finally, in this thesis, we focus on the prediction in the spatio-temporal data. In general, the prediction results provide a reference for decision making. In the next step, we plan to using deep reinforcement learning models to help make decision in the spatio-temporal field. For example, in chapter A.3, we can design a deep reinforcement learning based architecture to decide how to schedule the drivers or recommend a suitable price for dynamic pricing.

参考文献

[1] Papadimitratos P, De La Fortelle A, Evenssen K, et al. Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation. IEEE Communications Magazine, 2009, 47(11).

[2] TTI. Texas Transportation Institute: 2015 Urban Mobility Scorecard and Appendices. https://mobility.tamu.edu/ums/report/, 2015.

[3] Zheng W, Lee D H, Shi Q. Short-term freeway traffic flow prediction: Bayesian combined neural network approach. Journal of transportation engineering, 2006, 132(2):114–121.

[4] Kim W, Natarajan S, Chang G L. Empirical analysis and modeling of freeway incident duration. Proceedings of Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on. IEEE, 2008. 453–457.

[5] Asghari M, Emrich T, Demiryurek U, et al. Probabilistic estimation of link travel times in dynamic road networks. Proceedings of Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2015. 47.

[6] Liu W, Zheng Y, Chawla S, et al. Discovering spatio-temporal causal interactions in traffic data streams. Proceedings of Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011. 1010–1018.

[7] Xu J, Deng D, Demiryurek U, et al. Mining the situation: Spatiotemporal traffic prediction with big data. IEEE Journal of Selected Topics in Signal Processing, 2015, 9(4):702–715.

[8] Hu H, Li G, Bao Z, et al. Crowdsourcing-based real-time urban traffic speed estimation: From trends to speeds. Proceedings of 32nd International Conference on Data Engineering. IEEE, 2016. 883–894.

[9] Yuan N J, Zheng Y, Zhang L, et al. T-finder: A recommender system for finding passengers and vacant taxis. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(10):2390–2403.

[10] Chiang M F, Hoang T A, Lim E P. Where are the passengers?: a grid-based gaussian mixture model for taxi bookings. Proceedings of Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2015. 32.

[11] Lichman M, Smyth P. Modeling human location data with mixtures of kernel densities. Proceedings of Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014. 35–44.

[12] Quinlan J R. Induction on decision tree. Machine Learning, 1986, 1(1):81–106.

[13] Friedman J, Hastie T, Tibshirani R. The elements of statistical learning, volume 1. Springer series in statistics Springer, Berlin, 2001.

[14] Salzberg S L. C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. Machine Learning, 1994, 16(3):235–240.

[15] Breiman L. Random forests. Machine learning, 2001, 45(1):5–32.

[16] Friedman J H. Stochastic gradient boosting. Computational Statistics & Data Analysis, 2002, 38(4):367–378.

[17] Friedman J H. Greedy function approximation: a gradient boosting machine. Annals of statistics, 2001. 1189–1232.

[18] Breiman L. Arcing the edge. Technical report, 1997.

[19] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press, 2016.

[20] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553):436–444.

[21] Ciregan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification. Proceedings of 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2012. 3642–3649.

[22] Weng J J, Ahuja N, Huang T S. Learning recognition and segmentation of 3-d objects from 2-d images. Proceedings of Proceedings of the 4th International Conference on Computer Vision. IEEE, 1993. 121–128.

[23] Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks. Proceedings of Advances in neural information processing systems, 2014. 3104–3112.

[24] Gers F A, Schmidhuber E. LSTM recurrent networks learn simple context-free and context-sensitive languages. IEEE Transactions on Neural Networks, 2001, 12(6):1333–1340.

[25] Mesnil G, Dauphin Y, Yao K, et al. Using recurrent neural networks for slot filling in spoken language understanding. IEEE/ACM Transactions on Audio, Speech and Language Processing, 2015, 23(3):530–539.

[26] Waibel A, Hanazawa T, Hinton G, et al. Phoneme recognition using time-delay neural networks. IEEE transactions on acoustics, speech, and signal processing, 1989, 37(3):328–339.

[27] Hinton G, Deng L, Yu D, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 2012, 29(6):82–97.

[28] Wang D, Cao W, Li J, et al. DeepSD: Supply-Demand Prediction for Online Car-hailing Services using Deep Neural Networks. Proceedings of 33nd International Conference on Data Engineering. IEEE, 2017. 243–254.

[29] Arel I, Rose D C, Karnowski T P. Deep machine learning-a new frontier in artificial intelligence research [research frontier]. IEEE Computational Intelligence Magazine, 2010, 5(4):13–18.

[30] Bengio Y, et al. Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2009, 2(1):1–127.

[31] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86(11):2278–2324.

[32] Nielsen M A. Neural Networks and Deep Learning. Determination Press, 2015.

[33] LeCun Y, Bengio Y, et al. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 1995, 3361(10):1995.

[34] Schmidhuber J. Learning complex, extended sequences using the principle of history compression. Neural Computation, 1992, 4(2):234–242.

[35] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. Proceedings of Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. 770–778.

[36] Bengio Y, Ducharme R, Vincent P, et al. A neural probabilistic language model. Journal of machine learning research, 2003, 3(Feb):1137–1155.

[37] Roweis S T, Saul L K. Nonlinear dimensionality reduction by locally linear embedding. science, 2000, 290(5500):2323–2326.

[38] Mesnil G, He X, Deng L, et al. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. Proceedings of INTERSPEECH, 2013. 3771–3775.

[39] Graves A, Jaitly N. Towards End-To-End Speech Recognition with Recurrent Neural Networks. Proceedings of ICML, volume 14, 2014. 1764–1772.

[40] Harris D, Harris S. Digital design and computer architecture. Elsevier, 2012.

[41] Turian J, Ratinov L, Bengio Y. Word representations: a simple and general method for semi-supervised learning. Proceedings of Proceedings of the 48th annual meeting of the association for computational linguistics, 2010. 384–394.

[42] Mikolov T, Yih W t, Zweig G. Linguistic Regularities in Continuous Space Word Representations. Proceedings of Proceedings of NAACL-HLT, 2013. 746–751.

[43] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. Proceedings of Advances in neural information processing systems, 2012. 1097–1105.

[44] Graves A, Mohamed A r, Hinton G. Speech recognition with deep recurrent neural networks. Proceedings of 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, 2013. 6645–6649.

[45] Hochreiter S, Schmidhuber J. Long Short-Term Memory. Neural Computation, 1997, 9(8):1735.

[46] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions. Proceedings of Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015. 1–9.

[47] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556, 2014..

[48] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. 2014, 79(10):1337–1342.

[49] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks, 2015, 61:85–117.

[50] Wang R, Chow C Y, Lyu Y, et al. Taxirec: recommending road clusters to taxi drivers using ranking-based extreme learning machines. Proceedings of Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2015. 53.

[51] Ge Y, Xiong H, Tuzhilin A, et al. An energy-efficient mobile recommender system. Proceedings of Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2010. 899–908.

[52] Moreira-Matias L, Gama J, Ferreira M, et al. Predicting taxi–passenger demand using streaming data. IEEE Transactions on Intelligent Transportation Systems, 2013, 14(3):1393–1402.

[53] Hunter T, Herring R, Abbeel P, et al. Path and travel time inference from GPS probe vehicle data. NIPS Analyzing Networks and Learning with Graphs, 2009, 12:1.

[54] Yeon J, Elefteriadou L, Lawphongpanich S. Travel time estimation on a freeway using Discrete Time Markov Chains. Transportation Research Part B: Methodological, 2008, 42(4):325–338.

[55] Ramezani M, Geroliminis N. On the estimation of arterial route travel time distribution with Markov chains. Transportation Research Part B: Methodological, 2012, 46(10):1576–1590.

[56] Hofleitner A, Herring R, Abbeel P, et al. Learning the dynamics of arterial traffic from probe data using a dynamic Bayesian network. Intelligent Transportation Systems, IEEE Transactions on, 2012, 13(4):1679–1693.

[57] Yang B, Guo C, Jensen C S. Travel cost inference from sparse, spatio temporally correlated time series using markov models. Proceedings of the VLDB Endowment, 2013, 6(9):769–780.

[58] Chu V W, Wong R K, Liu W, et al. Causal structure discovery for spatio-temporal data. Proceedings of International Conference on Database Systems for Advanced Applications. Springer, 2014. 236–250.

[59] Wang Y, Zheng Y, Xue Y. Travel time estimation of a path using sparse trajectories. Proceedings of Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014. 25–34.

[60] Zhong S, Ghosh J. A new formulation of coupled hidden Markov models. Dept. Elect. Comput. Eng., Univ. Austin, Austin, TX, USA, 2001..

[61] Sevlian R, Rajagopal R. Travel Time Estimation Using Floating Car Data. arXiv preprint arXiv:1012.4249, 2010..

[62] Pan B, Demiryurek U, Shahabi C. Utilizing real-world transportation data for accurate traffic prediction. Proceedings of Proceedings of the 2012 IEEE 12th International Conference on Data Mining. IEEE, 2012. 595–604.

[63] Wang D, Cao W, Xu M, et al. ETCPS: An Effective and Scalable Traffic Condition Prediction System. Proceedings of International Conference on Database Systems for Advanced Applications. Springer, 2016. 419–436.

[64] Wang J, Gu Q, Wu J, et al. Traffic Speed Prediction and Congestion Source Exploration: A Deep Learning Method. Proceedings of Proceedings of the 2016 IEEE 16th International Conference on Data Mining. IEEE, 2016. 499–508.

[65] Jenelius E, Koutsopoulos H N. Travel time estimation for urban road networks using low frequency probe vehicle data. Transportation Research Part B: Methodological, 2013, 53:64–81.

[66] Rahmani M, Jenelius E, Koutsopoulos H N. Route travel time estimation using low-frequency floating car data. Proceedings of 16th International Conference on Intelligent Transportation Systems. IEEE, 2013. 2292–2297.

[67] Dai J, Yang B, Guo C, et al. Path cost distribution estimation using trajectory data. Proceedings of the VLDB Endowment, 2016, 10(3):85–96.

[68] Grover A, Kapoor A, Horvitz E. A deep hybrid model for weather forecasting. Proceedings of Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Mining. ACM, 2015. 379–386.

[69] Wu S, Ren W, Yu C, et al. Personal recommendation using deep recurrent neural networks in NetEase. Proceedings of 32nd International Conference on Data Engineering. IEEE, 2016. 1218–1229.

[70] Ding X, Zhang Y, Liu T, et al. Deep learning for event-driven stock prediction. Proceedings of Proceedings of the 24th International Joint Conference on Artificial Intelligence. IJCAI, 2015. 2327–2333.

[71] Mocanu E, Nguyen P H, Gibescu M, et al. Demand forecasting at low aggregation levels using Factored Conditional Restricted Boltzmann Machine. Proceedings of 2016 Power Systems Computation Conference, 2016. 1–7.

[72] Lv Y, Duan Y, Kang W, et al. Traffic flow prediction with big data: a deep learning approach. IEEE Transactions on Intelligent Transportation Systems, 2015, 16(2):865–873.

[73] Junbo Z, Yu Z, Dekang Q, et al. DNN-Based Prediction Model for Spatio-Temporal Data. Proceedings of Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2016. 92.

[74] Zhang J, Zheng Y, Qi D. Deep spatio-temporal residual networks for citywide crowd flows prediction. Proceedings of Proceedings of the 31st AAAI Conference on Artificial Intelligence, 2017.

[75] Song X, Kanasugi H, Shibasaki R. Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level. Proceedings of Proceedings of the 25th International Conference on Artificial Intelligence. IJCAI, 2016. 2618–2624.

[76] Dong W, Li J, Yao R, et al. Characterizing Driving Styles with Deep Learning. arXiv preprint arXiv:1607.03611, 2016..

[77] Ma X, Dai Z, He Z, et al. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. Sensors, 2017, 17(4):818.

[78] Chen T, Guestrin C. Xgboost: A scalable tree boosting system. Proceedings of Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016. 785–794.

[79] Zheng X, Liang X, Xu K. Where to Wait for a Taxi? Proceedings of Proceedings of the ACM SIGKDD International Workshop on Urban Computing. ACM, 2012. 149–156.

[80] Wang H, Calabrese F, Di Lorenzo G, et al. Transportation mode inference from anonymized and aggregated mobile phone call detail records. Proceedings of 13th International Conference on Intelligent Transportation Systems. IEEE, 2010. 318–323.

[81] Clevert D A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, 2015..

[82] Kingma D, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014..

[83] Srivastava N, Hinton G E, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014, 15(1):1929–1958.

[84] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints, 2016, abs/1605.02688.

[85] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2011, 12:2825–2830.

[86] Leontiadis I, Marfia G, Mack D, et al. On the effectiveness of an opportunistic traffic management system for vehicular networks. Intelligent Transportation Systems, IEEE Transactions on, 2011, 12(4):1537–1548.

[87] Ma S, Zheng Y, Wolfson O. T-share: A large-scale dynamic taxi ridesharing service. Proceedings of 29th International Conference on Data Engineering. IEEE, 2013. 410–421.

[88] Kwon J, Murphy K. Modeling freeway traffic with coupled HMMs. Technical report, Technical report, Univ. California, Berkeley, 2000.

[89] Lou Y, Zhang C, Zheng Y, et al. Map-matching for low-sampling-rate GPS trajectories. Proceedings of Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2009. 352–361.

[90] Yuan J, Zheng Y, Zhang C, et al. An interactive-voting based map matching algorithm. Proceedings of Proceedings of the 2010 Eleventh International Conference on Mobile Data Management. IEEE Computer Society, 2010. 43–52.

[91] Li Y, Tian X, Liu T, et al. Multi-task Model and Feature Joint Learning. Proceedings of Proceedings of the 24th International Conference on Artificial Intelligence. IJCAI, 2015. 3643–3649.

[92] Zhang Z, Luo P, Loy C C, et al. Facial landmark detection by deep multi-task learning. Proceedings of European Conference on Computer Vision. Springer, 2014. 94–108.

[93] Liu P, Qiu X, Huang X. Recurrent Neural Network for Text Classification with Multi-Task Learning. Proceedings of Proceedings of the 25th International Joint Conference on Artificial Intelligence. IJCAI, 2016. 2873–2879.

[94] Xu M, Wang D, Li J. DESTPRE: a data-driven approach to destination prediction for taxi rides. Proceedings of International Joint Conference on Pervasive and Ubiquitous Computing. ACM, 2016. 729–739.

致　谢

首先，我要感谢并感谢我的导师李建博士的宝贵支持和指导。从我博士生涯的一开始，他就充分提供了探索不同研究问题的自由，不断激励和挑战我，把自己的观点和理解提升到现在这个层次。我非常感谢他的指导和建议。他的对科研的热情，充沛的精力和工作的动力一直是我们灵感的源泉。在他指导下的这段时间里我得到了很多的帮助。

我要感谢曹玮和ADL组其他成员（许梦雯、黄棱萧、刘宇、金逸飞、付昊、金恺等等）跟我一起讨论问题，与我分享他们的知识和经验，并给我提供了宝贵意见和鼓励。

我也要感谢父母的爱和支持。他们总是无条件的支持我，相信我和鼓励我。我也要感谢我的亲戚朋友，他们的关爱让我温暖、充满动力。

最后，我要感谢交叉信息院所有的老师，没有交叉信息院提供的良好的学习环境和科研资源，我不会如此顺利地完成我的科研工作。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：____________ 日　期：____________

附录 A 中文摘要

A.1 引言

A.1.1 研究背景及挑战

目前，许多大型城市的快速扩张带来了城市人口的迅速膨胀。随之而来的是这些城市中交通需求的急剧增加。人们面临着交通拥堵和打车难的问题。德州交通研究所2015年发表的《Urban Mobility Scorecard》显示，在2014年美国471城市地区中，交通挤塞问题共导致居住在这些城市的美国人额外花费了69亿小时，多消耗了31亿加仑燃料，造成的直接经济损失高达1600亿美元。并且交通拥堵情况保持上升趋势，严重阻碍了城市的发展。

同时，基于位置的服务(LBS)和GPS嵌入式设备变得无处不在。这样的GPS嵌入式设备和移动应用程序深深地影响了人们的日常生活。例如，人们使用他们的智能手机来规划他们的路线，在线叫车服务，寻找旅行伙伴，搜索目的地等。大量的基于位置的数据由这些设备和应用程序每天都被生成，包括在线订单，GPS轨迹，地图查询和带地理标记的签到数据等。这些在城市中收集到的海量数据包含了关于城市的宝贵信息。用机器学习和深度学习方法分析这些数据为建立更好的智能交通系统（ITS）[1] 提供了新的机会，从而减轻交通拥堵，提高人们在日常通勤中的生活质量。举例来说，根据交通条件的预测和给定路径的行驶时间估计，ITS可以给人们推荐更好的路线图，从而规避拥塞的道路。同时，通过预测通勤需求，ITS可以提前派出车辆来平衡供需，减少出租车空载造成的能源消耗。在本论文中，我们着重使用时空交通数据解决ITS中的三个重要预测问题。

- **在线叫车服务的供需预测** 在线叫车应用程序/平台（如Uber，Didi和Lyft）已经成为通过移动应用程序提供按需交通服务的新颖和流行的方式。通过鼓励私家车主提供叫车服务，扩大了城市的交通能力。随着越来越多的乘客和更多司机使用这项服务，预测在线叫车服务的供需变得越来越重要，基于此，调度系统可以提前派遣驾驶员，以最大限度地减少乘客的等待时间并最大限度地提高司机利用率。

- **交通路况预测** 许多研究表明，良好的交通状况预测系统在提高交通效率方面起着至关重要的作用。例如，政府可以在对交通规则的变更作出决定（例如，将公共汽车通道改为普通车道）或建设其他道路（例如增加车道）时，将其作为参考。当他们计划施工区时，也可以向土木工程师提出建议（例如，短期施工如何影响交通）[7]。
- **出行时间预估。** 估计给定路径的旅行时间是路线规划、导航和流量调度中的一个根本问题。准确估计旅行时间有助于人们更好地规划路线，避免拥挤的道路，从而有助于减轻交通拥堵。几乎所有的电子地图和在线叫车服务都在他们的应用程序中提供了旅行时间估计，例如谷歌地图，优步打车，滴滴打车等。当用户搜索到达目的地的路线时，地图应用程序提供了多个候选路线，其中估计出行时间（以及可能的其他措施，如燃气消耗，费用）供用户决定。

虽然这些问题得到广泛的研究[3–6]，但由于相应交通数据的海量规模和异构性，我们仍然要面临着大量的挑战。接下来，我们进一步说明学习和预测这些数据时所遇到挑战的特点。

- 交通数据通常包含空间（位置）和时间（时间戳）属性，我们也将其称为时空数据。这种时空数据同时包含了空间上和时间上的相关模式（correlation pattern）。例如，一条道路的交通路况状况受其自身之前的路况的影响，同时也受到与其相邻的一些道路的路况影响。同时捕获空间相关性和时间相关模式并不容易。
- 对于不同的地理位置和时间段，时空交通数据中的模式总是动态变化。例如，在早上，住房区的打车需求通常急剧上升；而在晚上，商务区域的打车需求往往会增加。此外，一周不同星期日的供求模式可能会有很大差异。以前的工作通常区分不同的地理位置，时间段或星期几，分别构建几个子模型[8–11]。将不同的订单分开处理并建立很多子模型，不仅是的模型变得繁琐，而且不同子模型都是只能通过一小部分数据进行训练，这样使得模型可能会受到缺乏训练数据的困扰。
- 此外，时空交通数据通常包含多个属性。例如，在线叫车需求预测问题中，订单数据包含时间戳，乘客ID，起始位置，目的地等多个属性，以及诸如交通状况，天气状况等几“环境”因素。这些属性一起为供需预测提

供了大量信息。然而，想在统一模型中使用所有属性并不容易。目前最标准的方法是人工抽取许多特征（即特征工程），并将其交给一些现成的学习算法，如逻辑回归或随机森林。然而，特征工程通常需要大量的人力和领域知识，而且对于如何进行特征抽取几乎没有一般性的指导原则。

- 最后，时空交通数据的规模通常相当庞大。例如，在本文中，我们在出行时间预估问题的实验部分中使用的轨迹数据总共包含$9,653,822$个轨迹和14亿个GPS记录。为了有效地处理这样的大规模数据，我们通常需要利用Hadoop，Spark等大型数据平台。此外，与传统的机器学习方法相比，深度学习技术显示出挖掘海量数据的巨大潜力。然而，据我们所知，并没有一个标准的深度学习模型来处理我们上面提到的这种大规模、多噪音和多属性的时空数据。

在本文中，我们着重解决ITS中的三个重要预测问题，使用精心设计的机器学习和深度学习模型来应对上述挑战。我们接下来具体介绍问题和我们在个问题中的贡献。

A.1.2　在线叫车服务中的供需预测

在第 A.3章节中，我们研究了预测实时在线叫车供需的问题，这是在线叫车平台中一个有效调度系统的最重要的组成部分之一。我们的目标是预测在未来几分钟内某一地区的在线叫车业务的供应与需求之间的差距。根据预测，我们可以通过提前调度车辆和动态调整价格来平衡供需。通过对数据的观察，我们发现汽车供应量在不同的地理位置和时间段内呈现动态变化的特征。此外，一周内不同日子的供求分布可能非常不同。预测这种异质数据是很有挑战性的。

贡献： 我们基于一种新颖的深层神经网络结构提出了一种称为*Deep Supply-Demand(DeepSD)* 的端到端框架。我们的方法只需要最少量的手工提取特征，就可以自动从在线叫车服务数据中发现复杂的供需模式。此外，我们的框架是高度灵活和可扩展的。基于我们的框架，可以很容易利用多个不同数据源中获取的数据（例如，在线叫车订单，天气信息和交通数据）来实现提高预测精度。我们对模型进行广泛的实验评估，实验结果表明我们的框架可以提供比现有方法更准确的预测结果。

A.1.3 交通路况预测

在第A.4章中，我们研究在给定道路网络的当前和历史交通状况下，预测几分钟后每条道路交通状况的问题。基于位置服务的广泛使用使我们能够从GPS嵌入式设备中收集大量的交通数据。尽管已经存在很多基于GPS数据的交通预测的研究和产品，但大部分仅集中在被频繁经过的高速公路或主干道路预测上，很多城市中的道路并不在他们的预测之列。通过对数据的观察，我们发现将原始的交通路况时间序列转换为两种不同形式的时间序列（期望现实差距和一阶差分序列）后，新的时间序列显示出了非常强的自相关性。我们希望这些观察能够为进一步研究交通路况预测及相关问题提供帮助。

贡献： 基于从浮动车辆收集的当前和历史GPS数据，我们提出了一种集成交通状况预测系统（ETCPS），用于预测城市道路交通状况。我们在交通状况时间序列中观察到两个有用的相关性，这是我们后续模型设计的基础。基于这两个观测到的相关性，我们提出了两种称为预测回归树（*R-Tree*）和时空概率图形模型（*STPGM*）的不同模型。我们通过将两个模型精心整合来取得了最好的预测结果。我们的系统预测结果精确，并可以轻松扩展到非常大的数据集上。

A.1.4 出行时间预估

在第A.5章中，我们研究了给定路径和开始时间的出行时间估计问题。以前的工作通常专注于估计单独道路或子路径的行驶时间，然后将这些估计的行驶时间进行加和得到路径行驶时间。然而，出行时间也受路径中道路交叉路口或红绿灯数量的影响，并且路段上的估计误差会累积，因此这种方法的估计值可能并不准确。此外，给定道路和出发时间的行驶时间同样也受不同司机的驾驶风格影响。

贡献： 我们提出了一个端到端的出行时间预测框架，叫做*DeepTTE*。我们的模型基于深层循环神经网络直接估计整个路径的出行时间。在我们的模型中，我们考虑了路径中的空间和时间依赖性以及可能影响影响出行时间的各种因素，如驾驶习惯，星期几等。我们在大规模数据集上进行了广泛的实验验证。实验结果表明，我们的模型显著优于其他现有方法。

A.1.5 论文架构

论文的其余部分组织如下：在第A.1结中，我们将介绍本研究工作的背景，挑战和主要贡献。在第A.2结中，我们将简要介绍相关工作，以及深度学习技术在时空数据中的应用。在第A.3结中，我们提出一个名为DeepSD的端到端框架，基于一种新颖的深层神经网络结构来预测在线叫车服务的供需。在第A.4结中，我们介绍了一个集合交通状况预测系统（ETCPS），用于预测城市道路交通状况。在第A.5结中，我们提出了基于深度循环神经网络的称为DeepTTE的出行时间估计的端到端框架。最后，我们将在第A.6结中给出本文的总结和下一步计划。

A.2 相关工作

在本结中，我们分别回顾了我们在本论文中研究的三个问题的现有相关工作。已存在大量的有关时空数据的学习和预测的文献，我们这里只提一些密切相关的文献。在本结末尾，我们回顾了利用深度学习研究时空数据预测的现有工作。

A.2.1 在线约车平台的供需预测

A.2.1.1 出租车路径推荐

出租车路线推荐旨在为司机预测能够最大化车辆利用率的路线。Yuan等[9]提出了向出租车司机提出他/她最有可能能够接到乘客位置的一个算法。他们使用泊松模型来预测每个停车地点接到乘客的概率。在他们的工作中，接客地点是在固定的集合范围内。我们的工作旨在预测所有地区的供需缺口。Wang等人[50]调研了向出租车司机推荐一组道路的问题。他们使用了一个隐藏层神经网络和精心挑选的人工抽取特征。我们的工作使用了一个只需要很少的人工抽取特征的深层神经网络。Ge等人[51]提供了一种具有高成本效益的路线推荐算法，可以推荐一系列的接客位置。他们从最成功的司机的历史行驶轨迹中学到知识，以改善其他司机的的士司机的车辆利用率。然而，这个问题设定与我们的区别很大。

A.2.1.2 出租车需求预测

出租车需求预测问题研究每个接客地点需求的预测问题。Moreira-Matias等人[52]结合泊松模型和自回归滑动平均（ARMA）模型来预测每个出租车站的需求。他们还是只考虑了几个固定接客地点（出租车站）的需求。此外，在他们的工作中，他们分别处理了每个出租车站的数据。这种实施缺乏训练数据。在最近的一项工作中，Chiang等人[10]提出了一种生成模型，称为基于网格的高斯混合模型，用于对时空出租车预订建模。他们的方法能够在城市的每个地区的任何时间间隔内预测出租车的需求。然而，一方面，他们对工作日和周末的订单分开对待。另一方面，在他们的方法中，出租车预订的总量由泊松模型（Poisson）提前决定。当实时出租车需求迅速变化时，他们的方法可能会导致大的预测误差。

我们强调以前的工作只研究了需求预测，而忽视了供应。在出租车路线推荐，出租车调度等实际应用中，重要的是预测供需平衡。此外，这些工作都没有研究加入环境数据，如天气或交通条件，以提高预测精度。在2016年滴滴预测大赛中①，冠军队使用了梯度提升算法②，并提出了1534个精心设计的特征。除了基本特征（如区号，星期几，之前的供需差距和相应的统计数据）外，还考虑了非常详细的特征，如不同时间段内不同乘客的等待时间、呼叫时间，不同区域的汽车供应/需求比例等的平均值和标准差。尽管他们的模型取得了显着的表现，但是设计这些特征是非常不容易的，并且需要大量的人力。

A.2.2 交通路况预测

在本节中，我们将回顾相关的现有工作。大多数以前的工作使用概率模型来预测交通状况。Hunter等人[53] 将主干路网中的交通状况预测表示为最大似然问题，并根据观察到的历史路线行驶时间估计行驶时间分布。Yeon等人使用离散时间马尔科夫链(DTMC)，在高速公路上估算交通状况[54]。然而，这些工作假设不同路段上的旅行时间是独立的，而不考虑不同道路上的交通状况之间的相关性，这可能导致城市地区的错误预测。

为了捕捉路段之间的相关性，Hofleitner等人[56]将相邻路段之间的状态转移

---

① http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016&&locale=en

② https://github.com/Microsoft/LightGBM/

表示为了动态贝叶斯网络模型，并通过EM方法预测了交通状况。然而，它并没有考虑到算法在大规模数据上的效率。Yuan等[9]根据出租车的轨迹建立了一个地标图，每个节点（即地标）表示路段，每条边表示两个地标之间的出租车往来集合。他们使用地标图表示相关性，并估计了边上的行驶时间分布。然而，由于地标是从前k个最经常被通过的路段中选出的，因此许多具有稀疏记录的路段无法预测。

与我们的模型最相关的工作是由Yang等提出的[57]。他们提出了一种时空隐马尔可夫模型，称为STHMM的算法。他们进一步提出了一种处理数据稀疏性的有效方法。然而，他们并没有考虑不同时间间隔内的转移模式的异质性。在我们的实验部分中，我们展示了我们的模型在效率和准确性方面都优于STHMM。我们强调Chu等人[58]考虑了不同时间间隔的转换模式，并提出了时变动态网络。然而，他们的目标是揭示环路系统中的因果结构,这与我们的目标不同。

此外，我们强调最近的两个相关的工作[5,59]。 Wang等[59]提出了一种有效的算法，通过使用张量分解，基于出租车在最近时间段和历史上产生的稀疏轨迹来估计任意路径的行进时间。他们不是预测交通状况，而是研究当前时间段内给定路径的行驶时间估计。 Asghari等人[5]根据历史传感器数据估计行驶时间分布。由于他们的工作研究的是为旅行计划找到最可靠路线的算法。这与我们的工作具有相关但不同的研究范围。

A.2.3　行驶时间估计

A.2.3.1　基于路段的行驶时间估计

行驶时间估计已被广泛研究[60–62]。然而，这些工作估计了个别路段的行驶时间，而不考虑道路之间的相关性。 Yuan等人[57]使用时空隐马尔可夫模型来表示相邻道路之间的关系。 Wang等人[63]通过基于交通条件时间序列中两个观察到的有用相关性的集合模型改进了这项工作。 Wang等人[64]提出了一种称为eRCNN的误差反馈循环卷积神经网络，用于估计每条道路上的交通速度。这些研究考虑了不同道路之间的相关性。但是，他们专注于准确估计单个路段的行驶时间或速度。路径的行驶时间受各种因素的影响，例如道路交叉口的数量和路径中的交通灯。简单地将路段中路段的行驶时间加和并不会导致准确的结

果[65]。

A.2.3.2 基于路径的行驶时间估计

Rahmani等人[66]根据路径的历史数据估计路径的行车时间。然而，基于历史平均值的模型可能导致较差的准确性。此外，由于新的查询路径可能不包括在历史数据中，所以它受数据稀疏问题的影响。 Yuan等人[9]建立了一个基于出租车历史轨迹的地标图，每个地标代表一条路段。他们根据地标图估计路径的旅行时间分布。然而，由于地标是从前k个经常被通过的道路中选出的，因此无法准确估计出行记录很少的道路。此外，Wang等人[59]根据历史数据中的子轨迹估计路径的行进时间。他们使用张量分解来分解子轨迹，这种方法有效提高了准确性。尽管如此，它仍然受到数据稀疏问题的困扰，因为许多子轨迹只被非常少的司机行驶过。

Dai等人[67]提出了路径成本分布估算的新范例。给定出发时间和查询路径，他们展示了如何为使用覆盖查询路径的相关子轨迹选择最优权重集，并使用联合分布计算查询路径的成本分布。由于他们专注于估计出行成本的不确定性，因此它与我们的问题相关但有所不同。在最近的出行时间估计大赛① 找到，冠军队采用随机森林，多层感知器，LASSO等一系列标准机器学习模型作为基准估计。他们使用梯度提升方法组合不同估计方法的估计结果作为最终结果。但是，我们强调在实践中设计这么多的机器学习模型非常繁琐，很难维护。相反，我们只使用一个端到端的框架。

A.2.4 时空数据上的深度学习

最近，深度学习技术展现了它在时空数据分析问题的实力。越来越多的研究人员研究了将深度学习技术应用于预测问题中[68–71]。然而，很少有工作研究使用深度学习解决时空数据上的预测。 Lv等人[72]研究了使用深层神经网络预测交通流量。他们采用堆自编码器以贪心的方式逐层训练网络。他们表明，与基线方法相比，深层模型可以获得更为准确的结果。 Zhang等人[73]设计了一种称为*DeepST*的新颖的架构来预测人潮流。他们的模型通过一系列卷积神经网络

---

① 比赛信息和数据可以在https://github.com/DeepTTE/DeepTTE

学习了时空模式。他们在[74]中对*DeepST*改进，并提出*ST-ResNet*。改进主要体现在：通过使用残差学习来构建更深层次的网络，并提出了一种基于参数矩阵的融合机制，用于建模空间和时间依赖。

Song等人[75]建立了一个名为DeepTransport的智能系统，用于模拟全市范围内的人员流动和交通模式。 Dong等人研究了通过堆叠式循环神经网络来描绘不同司机的驾驶风格。据我们所知，迄今尚未研究应用深度学习技术来提高在线约车供需预测的准确性，而且还没有以前的工作研究根据深度学习方法估计整个路径的行驶时间。

A.3　基于深度神经网络的在线约车平台供需预测

在线叫车服务在世界各地受到广泛的欢迎。随着越来越多的乘客和司机使用这项服务，一个行之有效的调度策略对在线叫车服务提供商而言变得越来越重要。有效的调度可以尽量减少乘客的等待时间和最大限度地提高司机利用率，从而改善整体用户体验。

在本章中，我们研究预测实时在线叫车供需问题，这是有效调度系统中最重要的组成部分之一。我们的目标是在未来几分钟内预测某一地区的在线叫车的供应与需求之间的差距。基于预测，我们可以提前安排司机来平衡供需。我们基于一种新颖的深层神经网络结构，提出一个名为*Deep Supply-Demand (DeepSD)*的“端到端”的框架。我们的方法只需要少量的人工抽取特征，就可以自动从在线叫车数据中发现复杂的供需模式。此外，我们的框架是高度灵活和可扩展的。基于我们的框架，我们可以很容易利用多个数据源（例如，在线叫车订单，天气和交通数据）来实现高精度。我们进行广泛的实验评估，实验表明我们的框架的预测结果比现有方法更准确。①。

A.3.1　引言

在线叫车应用程序/平台已经成为通过手机应用程序提供按需交通服务的新颖和流行方式。当需要要雇用车辆时，乘客只需在应用程序中输入她/他所需的接送位置和目的地，并将请求发送给服务提供商，服务提供商将请求

---

① 这个工作已经发表在了ICDE 2017[28]

转发到靠近乘客接送位置的一些司机，或直接安排一个附近的司机来接受订单。与传统的交通工具如地铁，公共汽车等相比，在线约车服务更加方便灵活。此外，在线约车服务通过鼓励私家车主提供汽车服务，可以促进分享经济，扩大城市交通能力。几款在线约车手机应用在世界各地受到广泛欢迎，如Uber，Didi和Lyft。每天有大量的乘客使用在线约车服务，同时产生了大量的订单。例如，中国最大的在线约车服务提供商Didi在中国每天处理大约11万在线叫车订单。①

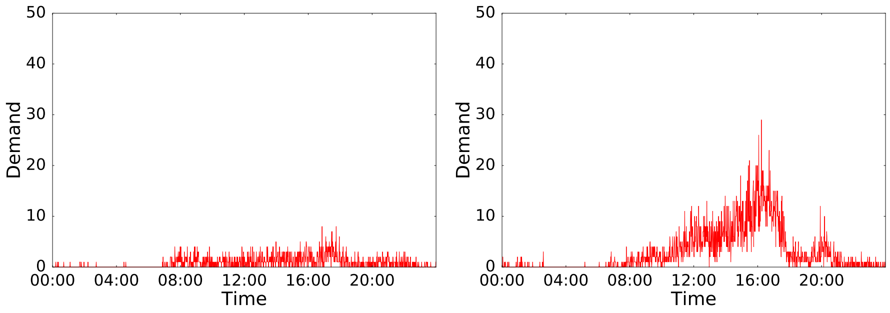
由于大量的司机和乘客使用这项服务，目前出现了以下几个问题：有时候，因为附近想要约车的人很少，有些司机很难收到任何约车请求;同时，一些乘客在恶劣的天气或高峰时段很难约到车，因为周边地区的需求显着超过供应。因此，这就要服务提供平台通过提前调度司机，以尽量减少乘客的等待时间，并最大限度地提高司机利用率。这是一个非常重要而又具有挑战性的任务。 供需预测是有效的司机调度程序的最重要的组成部分之一。如果可以预测/估计在未来某个时间段有多少乘客需要约车服务，以及附近有多少空闲司机可以使用，在线约车平台就可以通过提前调度车辆，动态调整价格或推荐某些司机的常用接乘客地点来提前平衡供应和需求。

在本结中，我们研究在线约车的供需预测问题。更具体地说，我们的目标是预测在接下来的几分钟的某个地区，在线约车供应和需求之间的差距（即$\max(0, \text{需求量} - \text{供应量})$）。我们的研究是在Didi的在线约车订单数据上进行的。为了引出我们的方法，我们首先提出这个问题的一些挑战，并讨论这个问题目前标准做法的缺点。

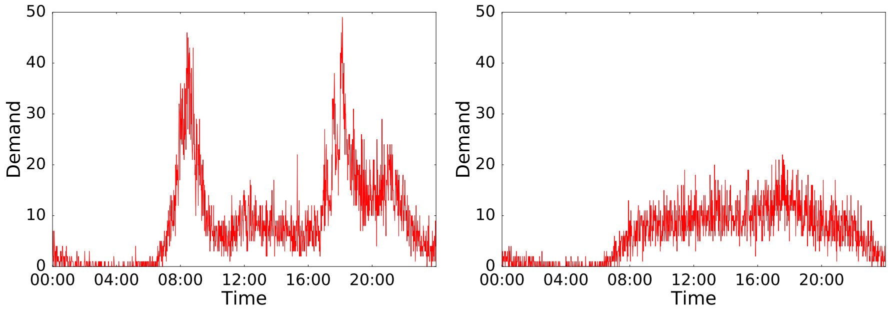
- 对于不同的地理位置和时间段，在线约车的供应需求动态变化。例如，在早上，住宅区的约车需求有所增加，而在晚上，商务领域的约车需求往往会增加。此外，一周不同星期日的供求模式可能会有很大差异。以前的工作通常区分不同的地理位置，时间段或星期几，分别构建几个子模型[8–11]。将不同的订单分开处理并建立很多子模型，不仅使得模型变得繁琐，而且不同子模型都是只能通过一小部分数据进行训练，这样使得模型可能会受到缺乏训练数据的困扰。
- 订单数据包含时间戳，乘客ID，起始位置，目的地等多个属性，以及诸

---

① 主页：http://www.xiaojukeji.com/en/index.html



(a) 3月9日区域一　　(b) 3月13日区域一



(c) 3月9日区域二　　(d) 3月13日区域二

图 A.1　在四种不同情况下的在线叫车需求。

如交通状况，天气状况等几类“环境”因素。这些属性一起为供需预测提供了大量信息。然而，想在统一模型中使用所有属性并不容易。目前标准的方法是人工抽取许多特征（即特征工程），并将其交给一些现成的学习算法，如逻辑回归或随机森林。然而，特征工程通常需要大量的人力和领域知识（通常数据科学、机器学习从业者需要抽取数百个不同的特征才能实现有竞争力的表现），而且对于如何进行特征抽取几乎没有一般性的指导原则。一些以前的工作只保留一部分的属性用于训练，如时间戳、起始位置，而丢弃了其他属性 [8,10,11,52,79]。虽然这使得训练变得更加容易，但丢弃一部分属性会导致信息丢失并降低预测精度。

为了给读者提供一些直觉理解，并展示出挑战，我们在图A.1中提供了一个例子。

例 1： 图A.1显示3月9日（星期三）及3月13日（星期日）两个地区的需求曲线。从图中可以看出，在两个不同时期呈现出非常不同的模式。对于区域一，

星期三很少有人需要在线约车服务。不过星期天的需求急剧上升。这种模式通常发生在娱乐区。对于区域二，我们观察到星期三的约车需求旺盛，特别是在八点钟左右的两个高峰时段和晚上7点钟（这是平日的大多数人的通勤时间）。星期天，这一区域的约车需求大幅减少。而且，供需模式随着时间一直在改变。还有许多其他复杂的因素可能会影响供需模式，很难详细列出。因此，简单地使用历史数据或经验供需模式的平均值可能会导致相当不准确的预测结果，相关结果我们将在实验部分中展示。 □

为了解决上述挑战，我们提出了一种称为*Deep Supply-Demand (DeepSD)*的供需预测的端到端框架。我们的框架是基于深度学习技术。深度学习技术已经在诸如视觉，言语和自然语言处理等许多应用领域成功地展示了它的力量[38,43,44]。特别是，我们开发出一种新的神经网络架构，它是针对我们的供需预测任务量身定做的。我们的模型只需要很少的手工特征提取就可以表现出高预测精度，并且可以轻松地扩展来利用新的数据集和特征。我们模型的初步版本在Didi供需预测大赛的1648个团队中取得了第二名的成绩①。我们用于比赛的初步版本模型与我们在第A.3.3结部分描述的基本模型几乎相同。我们在后续章节描述了我们的最终模型。最终模型通过引入一些新的想法进一步优化了基本模型，并且更加稳定和准确。我们目前正在努力部署该模型，并将其纳入Didi的调度系统。我们的技术贡献总结如下：

- 我们提出了一种基于深度学习方法的端到端框架。我们的方法可以自动学习不同时空属性（例如地理位置，时间间隔和星期几）的模式，这使我们能够以统一模型处理所有数据，而不是为它们分别建立子模型。与其他现成的方法（例如梯度提升，随机森林[13]）相比，我们的模型需要最小量的特征工程（即手工抽取特征），但是可以产生更准确的预测结果。
- 我们设计了一种新颖的神经网络架构，它受到Kaiming He等人最近为解决图像分类问题提出的深度残差网络 *(ResNet)* 的启发。新的网络结构可以很容易地将天气和流量数据等“环境因素”数据整合到我们的模型中。另一方面我们可以轻松地利用订单数据中包含的多个属性，而不会有太多的信息丢失。

---

① http://research.xiaojukeji.com/competition/main.action?competitionId=DiTech2016

- 我们利用*Embedding*方法[36,38]（一种自然语言处理中使用的流行技术）将高维特征映射到较小的子空间。在实验中，我们显示Embedding方法显着提高了预测精度。此外，通过Embedding，我们的模型还可以自动发现不同地区和时间段的供需模式之间的相似性。
- 我们进一步研究了我们的模型的可扩展性。在实际应用中，将新的外部属性或数据源加入到已经训练的模型中是很常见的。通常我们必须从头开始重新训练模型。然而，我们的模型的残差学习模块可以通过简单的微调*(fine tuning)*策略来利用这些已经训练的参数。在实验中，我们显示微调可以显着加快模型的收敛速度。
- 最后，我们在大规模实际数据集上——来自Didi的在线约车订单，进行了广泛的实验。实验结果表明我们的算法显著优于现有方法。我们的算法的预测误差比现有最佳方法低 11.9%。

A.3.2 公式化和概述

我们给出我们问题的正式定义。我们将一个城市划分为N不重叠的正方形区域$a_1, a_2, \ldots, a_N$，将每天分成1440个时间片（一分钟一个时间片）。然后我们在定义1中定义约车订单。

定义 1 (约车订单)： 约车订单o被定义为一个元组：发送约车请求的日期$o.d$，相应的时间段$o.ts \in [1, 1440]$，乘客ID $o.pid$,起始位置的区域ID $o.loc_s \in [N]$，和目的区域ID $o.loc_d \in [N]$。如果司机应答了这个约车请求，我们称这是一个有效的命令。否则，如果没有司机回答请求，我们称这是一个无效订单。

定义 2 (供需缺口)： 对于第d天，区域a中的时间段$[t, t+C)$的供需缺口被定义为该时间段内无效订单的总量。在本章中，我们将常量C定为10①，我们将相应的时间段表示为$\text{gap}_a^{d,t}$。

我们进一步收集了不同地区的天气状况数据和交通状况数据，我们将这些数据称作为环境数据。

---

① 常数10（分钟）是根据业务需求定的。它可以被任何其他常数代替

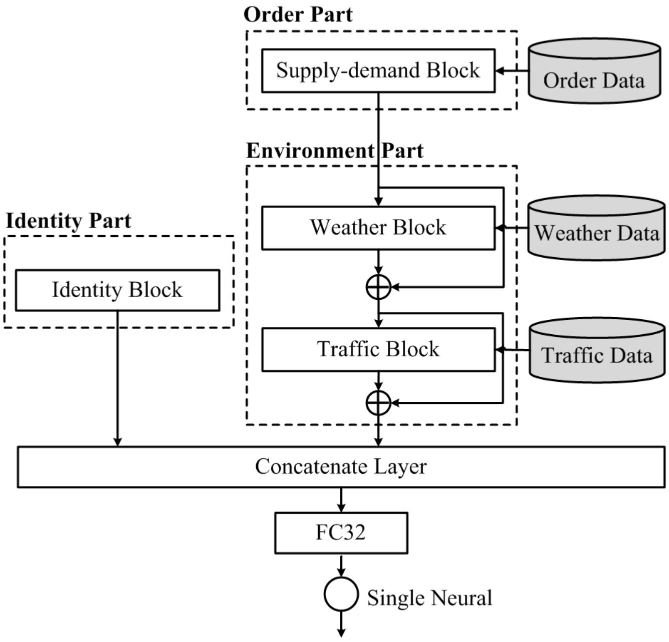


图 A.2 Structure of basic DeepSD

定义 3 (天气状况)： 对于特定区域a在第d天的第t时间段,天气状况（记作wc）被定义为一个元组：天气类型（例如，晴天，下雨多云等） wc.$type$，温度wc.$temp$，和PM2.5 wc.pm。所有区域在同一时间段内具有相同的天气条件。

定义 4 (交通状况)： 交通状况描述了每个区域的道路段拥挤程度：从1（最拥挤）到4（最不拥挤）。对于特定地区a在第d 天的第t时段，交通状况被定义为四维向量：区域a在四个拥塞程度之下的路段总数。

现在我们可以定义我们的问题如下。

问题假设当前日期是第d天，而当前时间段是第t时段。给定过去的订单数据和过去的环境数据，我们的目标是预测每个区域a的供需缺口$\mathrm{gap}_a^{d,t}$，即在未来10分钟内的供应需求缺口。

A.3.3 基础框架

在这结中，我们给出模型基础版本的框架结构。我们的基础模型包含三部分。每一个部分由一个或者更多的模块（模块是我们模型的基础组成单元）。模型中，我们首先通过标识部分处理“标识信息”（如区域ID，时间片，一周中

的第几天等）。然后我们通过订单部分处理订单数据。订单部分是我们模型最重要的部分。接着，我们使用环境部分来处理天气数据和交通数据。最终，我们将不同的模块通过残差连接的方式连接起来。我们模型的结构如图A.2所示。

A.4 道路路况预测系统

交通状况的实时预测是各种应用的重要组成部分。在本结中，我们提出了一种基于从浮动车辆收集的当前和历史GPS数据来预测城市道路交通状况的集成交通路况预测系统 *(ETCPS)*。我们将在交通路况时间序列中观察到两个有用的相关性作为我们设计的基础。我们基于这两个相关性提出两种不同的模型称为预测回归树 *(R-Tree)*和空间时间概率图形模型*(STPGM)*。我们最好的预测是通过两个模型的精心整合来实现的。我们的系统不仅可以提供高质量的预测，并且可以轻松扩展到非常大的数据集。我们在两个月内从北京的12000多辆出租车收集到的GPS数据集上进行了广泛的实验评估。实验结果表明我们系统的有效性，有效性和可扩展性。①

A.4.1 引言

交通状况的实时预测正变得越来越重要。有效的交通状况预测系统是各种实际应用的基本组成要素。这些应用包括：交通管理系统[86]，路径规划服务[9]，出租车拼车[87]等。这些问题近年来得到广泛的研究。一般来说，我们的目标是在给定到道路路网的当前和历史交通状况的情况下，预测未来15分钟后每条道路的交通状况。

之前的关于交通状况预测的大多数工作都是基于路侧环路传感器产生的数据。然而，这种环形传感器通常非常昂贵的，并且仅安装在了高速公路和部分城市主干道。其实，无处不在的基于位置的服务使我们能够从GPS嵌入式设备收集大量的流量数据。这样的GPS数据为分析和预测交通状况提供了重要信息。尽管已经有多项研究和产品是基于GPS数据来进行交通预测的,但他们大部分仅集中在高速公路、主干道路的路况预测上，并未考虑城市道路。

---

① 这个工作已经发表在了DASFAA 2016[63]。

在本结中，我们基于从浮动车辆（我们的数据中使用的是出租车）上收集到的GPS数据，研究交通路况预测的有效和可扩展的模型。为了使我们的阐述更加具体，我们首先说明我们问题中的几个挑战。

- 大量GPS数据已经在不断地生成，特别是对于一些大都会城市，如纽约或北京。以前的工作大多都是基于概率图形模型[55,56,88]。这些工作中算法的状态空间在非常大规模的数据集下会遇到爆炸的问题，这就导致了运行算法需要很长时间。
- 每条道路的交通路况及其转移规律（既：交通路况的变化规律）会随着时间的不同显著变化。比如，高峰期的交通拥堵通常会持续很长一段时间；而如果交通拥堵发生在非高峰期，通常道路很快就会变回通畅。这种交通规律正在随着时间的推移而变化。以前的基于马尔科夫链和隐马尔可夫模型的研究工作[55,57,88] 由于转移矩阵的状态与时间无关，无法捕获此特征。
- 出租车有时会减速甚至停下来接或吸引乘客。我们很难区分这样的低行驶速度是否是由于交通拥堵所致。这些记录可能导致对交通路况的错误估计。

为了解决上述挑战，我们提出了集成交通路况预测系统 *(ETCPS)*。我们的系统结合了两种不同的模型，分别被称为预测回归树 *(R-Tree)*和空间时间概率图形模型*(STPGM)*。我们的技术贡献总结如下：

- 我们在交通路况时间序列中观察到了两个有用的相关性，我们将此作为我们设计的基础。我们首先介绍交通路况与其预期交通路况之间差值所构成时间序列中的相关性。然后，我们将展示在交通路况时间序列的一阶差分中显示出的自相关性。
- 我们提出一种称为PR-Tree的基于回归树的预测模型。 PR-Tree可以有效地学习我们之前观测到的相关性，从而以高精度预测交通路况。PR-Tree在大规模数据集上非常高效。给定包含10^5条道路的训练集，PR-Tree的训练时间只需要3.26分钟，而且预测是实时的。
- 我们提出了一种称为STPGM的概率图形模型。STPGM可以学习相邻道路之间的相关性。它可以分开表达在不同的时间段内的状态转移。因此，STPGM的状态空间远小于以前的工作[55,56,88]。另一方面，STPGM可

以对不同时间段学习不同的转移规律。我们实验部分展示了，STPGM比以前工作中的算法更高效也更准确。

- 我们提出一种称为ETCPS的预测系统，它结合了PR-Tree和STPGM。我们在真实数据集上对我们的模型进行了评估，其中包括两个月内收集的超过12,000个出租车产生的GPS点。实验结果表明，ETCPS可以高效的支持大型道路路网，扩展性很好，并且可以提供准确的准确。

A.4.2　系统概述

我们提出的交通状况预测系统的框架如图A.3所示。我们开发了利用历史和实时出租车GPS记录来估计当前道路路况和预测下一个时间段道路路况的系统。它由四个主要部分组成：预处理，预测回归树模型（PR-Tree），空间时间概率图形模型（STPGM）和集成。

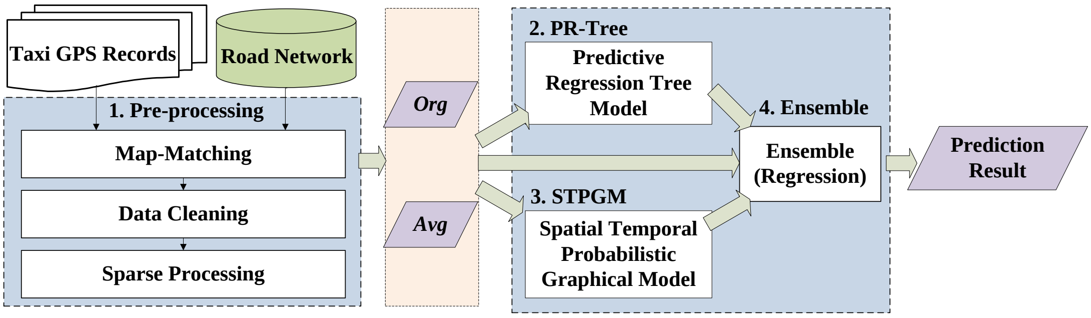


图 A.3　系统框架概述

在预处理阶段，首先，我们使用ST-Matching算法将GPS轨迹映射到道路网络[57]。然后，我们消除在乘客上车和下车状态下的记录（这样的记录通常会导致对交通状况的错误估计）。然后，我们处理稀疏问题，在一段时间间隔内没有观察到一些道路的GPS记录。通过预处理，我们得到两个时间序列Org和Avg。细节在实验部分中给出。接下来，我们使用基于回归树的模型PR-Tree来根据我们观察到的相关性来预测未来的交通路况。我们进一步采用称为STPGM的概率图形模型，它捕捉了我们的观察结果和路段之间的相关性。最后，我们结合了集合阶段的两个模型。我们表明，结合两个不同的模型提高了预测的准确性。

A.5 基于循环神经网络的出行时间预测

出行时间预估同样是一个重要且具有挑战性的问题。它是许多基于位置的服务（例如导航，路线规划系统等）的基本要素。在这个问题中，对于给定路径和相应的出发时间，我们估计司机行驶完路径所需时间。先前的工作通常集中在估计各路段或子路径的行驶时间，然后对这些估计的行驶时间进行加和。然而，这种方法会导致不准确的估计，因为出行时间也受路径中道路交叉路口或红绿灯数量的影响，并且路段上的估计误差会累积。我们提出了一个预测出行时间的端到端框架，叫做DeepTTE。我们基于深度循环神经网络来直接预估整条道路的出行时间。在我们的模型中，我们考虑路径中的空间和时间依赖性以及可能影响出行时间的各种因素，例如驾驶习惯，星期几等。我们在大规模数据集上进行了扩展实验。行驶，我们的模型相较于其他现存的方法，取得了很有竞争力的预测结果（在2017年DataCastle的出行时间预测竞赛的1578个团队中获得了第3名的成绩）。

A.5.1 引言

估计给定路径的行驶时间是路线规划，导航和流量调度中的一个基本问题。精确估计行驶时间可以帮助人们更好地规划路线。几乎所有的电子地图和在线叫车服务都会在他们的应用程序中提供旅行时间估计，例如Google地图，Uber，Didi等。当用户搜索到达目的地的路线时，地图应用程序提供了几条候选路线，估计行驶时间（以及可能的其他消耗，如汽油消耗，收费等）供用户进行决策。虽然这个问题在过去几年得到广泛的研究，但提供准确的行驶时间仍然是一个具有挑战性的问题。之前的工作通常侧重于估计单个道路路段的行驶速度/时间[57,62,64]。然而，路径的行驶时间受多种因素影响，如路径中所包含的路段数和交通灯数。简单的将每个单独路段的行驶时间相加起来并不能得到一个准确的时间估计，因为误差会来每条路段上累积[65]。另一种选择是，一些工作将道路分解成稍长一些的子轨迹而不是道路段，并基于子轨迹估计行驶时间[59]。虽然这种方法提高了估计的准确性，但是由于存在很多子轨迹只被非常少的司机行驶经过，所以这个方法受到数据稀疏问题的困扰。

在本结中，我们将路径视为一系列位置点（示例参见图A.4），我们使用

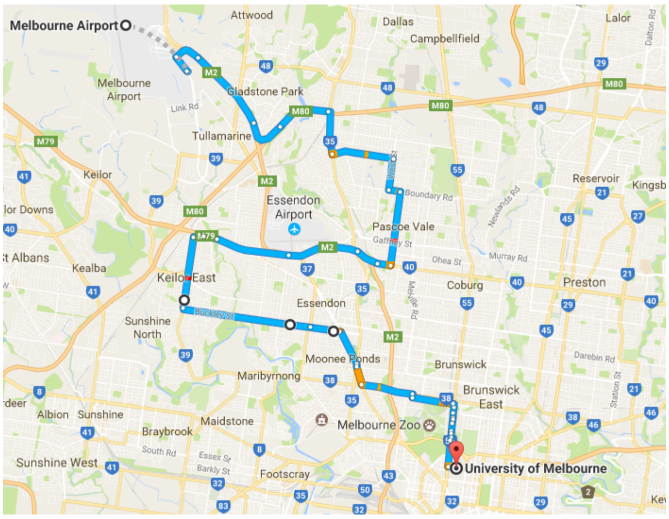


图 A.4　查询路径示例

基于深度学习方法学习从历史轨迹中估计行驶时间。为了使我们的阐述更加具体，我们首先给出我们问题中的一些挑战。

- 为了估计行驶时间，我们必须同时考虑给定轨迹中的空间和时间依赖性。之前的工作通常通过将轨迹离散成很多网格[74]或路段[57]来形式化这种依赖。然而，一方面将GPS点离散成网格会由于粗粒度导致的信息丢失。另一方面，根据单独道路路段推断出行驶时间将丢失道路交叉口和红绿灯的影响。很少有工作研究直接捕捉GPS点间的时空依赖性。
- 在不同的时间段内，相同路径的行驶时间可能非常不同。例如，在高峰时段，行驶通过同一段路径通常需要比非高峰时间要花费更长的时间。即使在某个固定的时间段，相同路径的行驶时间在一周中的不同日子呈现非常不同的分布。以前的工作通常会在一周的不同日子里建立几个子模型[8,9]。一方面，这种实现使得模型变得繁琐，另一方面，由于每个子模型仅能利用一小部分数据，这使得子模型可能遭受缺乏训练数据的问题。
- 不同的司机有不同的驾驶习惯。经验丰富的司机通常对城市的交通状况非常熟悉，驾驶速度非常快。相反，新手司机通常驾驶相对较慢，导致在相同条件下行驶时间较长。大多数以前的工作在估计行驶时间时不考虑司机的影响（司机信息在我们的数据集中可用）。
- 不同的历史轨迹具有非常不同的长度值（即，记录点数）或距离。图A.5显示了我们的数据集中轨迹长度和距离的分布。直接用传统的机器学

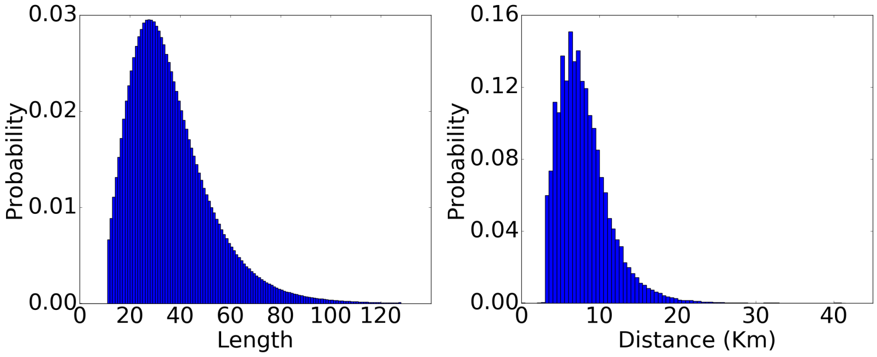


图 A.5 轨迹长度和距离分布

习模型（如随机森林，渐变增强等）处理可变长度的轨迹是不容易的。

为了解决上述挑战，我们提出了一种基于深度循环神经网络的端到端框架。本结的主要贡献可概括如下：

- 我们基于深层次的神经网络设计了一个端到端的框架来估计行驶时间，称为DeepTTE。我们结合了可能影响行驶时间的各种因素（例如，司机，星期几和时间间隔等）在统一的模型中，而不是手动构建几个子模型。
- 我们设计出一种新颖的神经网络架构，可以轻松处理可变长度的GPS轨迹。此外，通过仔细设计输入序列，我们的模型可以有效地捕捉了轨迹中的空间和时间依赖性，同时没有太多的信息丢失。
- 我们在大型真实数据集上进行了广泛的实验，其中包括在中国成都一个月内收集的超过$14,684$辆出租车所产生的GPS记录点。我们的模型实现了高质量的预测结果，平均误差率为12.74%，显着优于其他标准的机器学习算法。

A.5.2 问题定义

Definition 10 (**历史轨迹**)： 我们将历史轨迹定义为是历史上一段连续的GPS点序列，即$T = \{p_1, \ldots, p_{|T|}\}$。所有GPS点$p_i$包含：维度($p_i.lat$), 经度($p_i.lng$)和时间戳($p_i.ts$)。而且，对于每一条轨迹，我们记录下其对应的司机的ID号，记为driverID。

我们接下来讲述我们的问题目标。

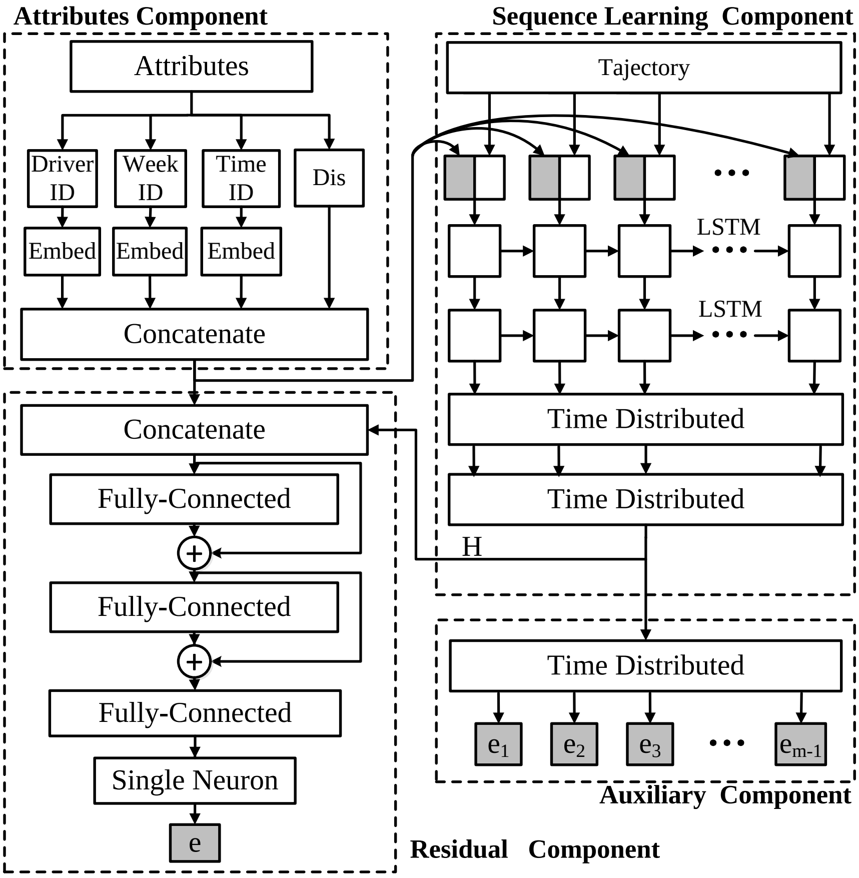


图 A.6 DeepETT概述

Definition 11 (**问题目标**)： 给定路径S和相关的司机、出发时间，我们的目标是估计在给定出发时间该司机走完S所需的时间。我们假设路径S是由用户指定或者路径规划软件生成的。S由一系列的路径上采样的定位点表示。所有定位点为经纬度点对。

注意： 在我们的实验中，我们将时间戳从历史数据中移除，使用移除时间后的历史轨迹来作为测试数据。在训练阶段，如定义10中所描述，我们通过历史轨迹数据学习估计给定轨迹的行驶时间。在测试阶段，为了使查询路径与训练数据中使用的轨迹数据统一，我们将查询路径表示为一系列的路径上采样的定位点。

A.5.3 模型架构

我们首先在图A.6中描述我们模型的架构。我们的模型由四部分组成：属性部分，序列学习部分，残差部分，辅助部分。

其中，属性部分负责处理诸如司机ID，出发时间所在时间段和周几等属性信息。序列学习部分处理GPS位置轨迹。残差部分使用前两个部分的输出估计给定路径和出发时间的路径行驶时间。最后，我们引入辅助模型来估计相邻两个GPS点之间的出行时间，来提高模型的表现。

A.6 结论

目前，基于位置的服务和GPS嵌入式设备的广泛应用改变了人们的生活方式。例如，人们使用智能手机来规划他们的路线，在线约车服务，寻找旅行伙伴，搜索目的地等。各种时空数据（例如，车辆移动性，流量模式，在线汽车数据和地理标记的入住数据等）每天被大量地生成。随着城市人口和车辆数量的快速扩张，城市的通勤需求急剧增加，人们遇到了交通拥堵和打车难等问题。分析大量的基于位置的数据为发现有价值的信息带来了新的机会。基于这些数据，使政府能够进行交通分析和城市规划，从而减轻交通拥堵和打车难问题。因此，这方面的研究工作最近越来越多的涌现了出来。

在本章中，我们将首先总结本文，并讨论本文中提供的框架/系统如何应用于所考虑的场景之外的问题。然后，我们将讨论大规模时空交通数据预测的未来发展方向。

A.6.1 本文总结

在本论文中，我们重点研究了大量时空交通数据的学习和预测。研究了三个具体问题，包括网约车供应需求预测，交通状况预测和旅行时间估计。这三个问题是相关的，但有在各自的特点上有所不同。它们都是时空交通区域中最基本的，经典的和重要的预测问题。同时，它们分别基于网格（在线租车服务的供需预测），道路段（交通状况预测）和序列（旅行时间），分别在时空交通区域中呈现预测问题的三个方面估计），这是地理聚合的三种常见形式。我们利用深度学习应用在其中两个问题当中并取得了非常显著的结果。

在第A.3章中，我们研究了预测实时在线叫车供需的问题。我们基于一种新颖的深层神经网络结构，提出了一种名为*Deep Supply-Demand (DeepSD)*的端到端框架。我们的方法只需要极少的手工提取特征，就可以自动在历史订单，天气和交通数据中发现复杂供需模式。我们在滴滴的真实数据集上进行了实验。实验结果表明，我们的模型显著优于现有方法。此外，我们的模型是高度灵活和可扩展的。我们可以轻松地将新的数据源或属性合并到我们的模型中，而无需重新训练。我们正在努力将我们的预测模型应用到滴滴的调度系统。

在第A.4章中，我们研究了有效的且可扩展的交通路况预测方法。我们提出了一种集成交通状况预测系统(ETCPS)，它结合了两种新型的模型：预测回归树(R-Tree)和时空概率图形模型(STPGM)。我们的模型基于交通状况数据中两个有用的观察到的相关性。我们的系统提供了高质量的预测，同时可以轻松扩展到非常大的数据集。我们进行广泛的实验来评估我们提出的模型。实验结果表明，与现有方法相比，ETCPS更加高效准确。未来，我们计划通过整合异构数据源的更多特征，如天气状况，POI信息等，来推断交通状况。接下来，我们将重点关注处理具有非常稀疏轨迹记录的路段的有效方式。此外，我们计划尝试不同的集合方法来组合不同的模型，以提高预测的性能。

在第A.5章中，我们研究对给定路径的估计行驶时间。我们提出了一种基于深度循环神经网络的端到端框架。我们的模型同时有效地捕获了给定路径中的时空依赖关系。此外，我们的模型考虑了可能影响旅行时间的各种因素，如司机，星期几等。我们在大规模的真实数据集上进行广泛的实验。实验结果表明，我们的模型实现了高估计精度，并且显著优于其他现有标准方法。

请注意，本论文提出的框架/系统可以很容易地应用于我们考虑的场景之外的问题。例如，在第A.3章节中，我们提出了一个名为*Deep Supply-Demand(DeepSD)*的端到端框架来预测在线叫车服务的供需。事实上，供需预测在物流运输，商品零售，光伏发电等诸多领域得到广泛的研究。更具体地说，以物流运输为例。物流公司，如UPS，FedEX和EMS，需要在每个城市维护一个物流网络，以提供收发包裹服务。每个网点负责某一块区域。然而，由于不同的网点和时间段，需要处理的包裹数量动态变化。例如，一些大型网点每天至少要处理$10,000$个包裹，而一些小规模的网点一天最多处理50个包裹。此外，一天的不同时间段内需要处理的包裹数量也非常不同。显然，对于不同的

网点和时间需要处理包数量的准确预测，可以有效地调度员工，从而大大提高生产力并节省成本。事实上，这个问题与在线约车服务的供需预测非常相似。DeepSD很容易适用于这种供应需求预测问题。

A.6.2　未来方向

本文的主体目标已经实现，但还有一些的方面可以进一步优化。例如，我们可以在预测框架/系统中引入更多的数据源来提高预测准确率。时空交通数据受复杂环境影响，特别是意外事件（如极端天气，交通事故，活动等）。基于搜索数据或社交网络数据的事件检测应添加到预测系统中，以提高异常情况下的预测精度。

另一个有趣的方向将是探索从循环神经网络的轨迹中提取的特征向量。在第A.5章中，我们讨论了使用循环神经网络将整个GPS轨迹表示为高维特征向量。虽然在这项工作中，我们使用向量来估计路径的行进时间，但是我们认为特征向量所包含的信息远远超过这一点。例如，特征向量可以包含驾驶员的驾驶风格，轨迹所有者的习惯以及轨迹通过的道路的交通状况等信息。基于轨迹特征向量，我们更好的进行不同轨迹之间的相似性检测。

最后，在本论文中，我们重点关注时空数据中的预测。一般来说，预测结果为决策提供参考。在下一步中，我们计划使用深度强化学习模型来帮助在时空领域做出决定。例如，在第A.3章中，我们可以设计一个基于强化学习的体系结构，以决定如何调遣司机或为动态定价推荐合适的价格。

个人简历、在学期间发表的学术论文与研究成果

个人简历

1990 年 8 月 29 日出生于江苏省徐州市。

2008 年 9 月考入中国矿业大学计算机科学与技术学院， 2012 年 7 月本科毕业并获得工学学士学位。

2012 年 9 月免试进入清华大学交叉信息研究院攻读博士学位至今。

发表的学术论文

[1] Wang D, Cao W, Li J, et al. DeepSD: Supply-Demand Prediction for Online Car-hailing Services using Deep Neural Networks. Proceedings of 33nd IEEE International Conference on Data Engineering, 2017, 243–254. (EI源刊, SCI源刊.)

[2] Wang D, Cao W, Xu M, et al. ETCPS: An Effective and Scalable Traffic Condition Prediction System. Proceedings of International Conference on Database Systems for Advanced Applications, 2016, 419–436. (EI收录, 检索号：20161502216093.)

[3] Xu M, Wang D, Li J, et al. DESTPRE: A Data-Driven Approach to Destination Prediction for Taxi Rides. Proceedings of International Joint Conference on Pervasive and Ubiquitous Computing, 2016, 729-739. (EI收录, 检索号：20164302929103.)

[4] Cao W, Wu Z, Wang D, et al. Automatic User Identification Method across Heterogeneous Mobility Data Sources. Proceedings of 32nd IEEE International Conference on Data Engineering, 2016. (EI收录, 检索号：20163202690741, SCI收录, WOS：000382554200083.)