Summer 2019

Applied deep learning in intelligent transportation systems and embedding exploration

Xiaoyuan Liang
New Jersey Institute of Technology

ABSTRACT

APPLIED DEEP LEARNING IN INTELLIGENT TRANSPORTATION SYSTEMS AND EMBEDDING EXPLORATION

by
Xiaoyuan Liang

Deep learning techniques have achieved tremendous success in many real applications in recent years and show their great potential in many areas including transportation. Even though transportation becomes increasingly indispensable in people's daily life, its related problems, such as traffic congestion and energy waste, have not been completely solved, yet some problems have become even more critical. This dissertation focuses on solving the following fundamental problems: (1) passenger demand prediction, (2) transportation mode detection, (3) traffic light control, in the transportation field using deep learning. The dissertation also extends the application of deep learning to an embedding system for visualization and data retrieval.

The first part of this dissertation is about a Spatio-TEmporal Fuzzy neural Network (STEF-Net) which accurately predicts passenger demand by incorporating the complex interaction of all known important factors, such as temporal, spatial and external information. Specifically, a convolutional long short-term memory network is employed to simultaneously capture spatio-temporal feature interaction, and a fuzzy neural network to model external factors. A novel feature fusion method with convolution and an attention layer is proposed to keep the temporal relation and discriminative spatio-temporal feature interaction. Experiments on a large-scale real-world dataset show the proposed model outperforms the state-of-the-art approaches.

The second part is a light-weight and energy-efficient system which detects transportation modes using only accelerometer sensors in smartphones. Understanding people's transportation modes is beneficial to many civilian applications, such as urban transportation planning. The system collects accelerometer data in an efficient way and leverages a convolutional neural network to determine transportation modes. Different architectures and classification methods are tested with the proposed convolutional neural network to optimize the system design. Performance evaluation shows that the proposed approach achieves better accuracy than existing work in detecting people's transportation modes.

The third component of this dissertation is a deep reinforcement learning model, based on Q learning, to control the traffic light. Existing inefficient traffic light control causes numerous problems, such as long delay and waste of energy. In the proposed model, the complex traffic scenario is quantified as states by collecting data and dividing the whole intersection into grids. The timing changes of a traffic light are the actions, which are modeled as a high-dimension Markov decision process. The reward is the cumulative waiting time difference between two cycles. To solve the model, a convolutional neural network is employed to map states to rewards, which is further optimized by several components, such as dueling network, target network, double Q-learning network, and prioritized experience replay. The simulation results in Simulation of Urban MObility (SUMO) show the efficiency of the proposed model in controlling traffic lights.

The last part of this dissertation studies the hierarchical structure in an embedding system. Traditional embedding approaches associate a real-valued embedding vector with each symbol or data point, which generates storage-inefficient representation and fails to effectively encode the internal semantic structure of data. A regularized autoencoder framework is proposed to learn compact Hierarchical K-way D-dimensional (HKD) discrete embedding of data points, aiming at capturing semantic structures of data. Experimental results on synthetic and real-world datasets show that the proposed HKD embedding can effectively reveal the semantic structure of data via visualization and greatly reduce the search space of nearest neighbor retrieval while preserving high accuracy.

APPLIED DEEP LEARNING IN INTELLIGENT TRANSPORTATION SYSTEMS AND EMBEDDING EXPLORATION

by
Xiaoyuan Liang

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

August 2019

APPROVAL PAGE

APPLIED DEEP LEARNING IN INTELLIGENT TRANSPORTATION SYSTEMS AND EMBEDDING EXPLORATION

Xiaoyuan Liang

Dr. Guiling Wang, Thesis Advisor Date
Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Ali Mili, Committee Member Date
Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Zhi Wei, Committee Member Date
Associate Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Zhu Han, Committee Member Date
Professor of Electrical and Computer Engineering, University of Houston
Professor of Computer Science and Engineering, Kyung Hee University, South Korea

---

Dr. Renqiang Min, Committee Member Date
Researcher, NEC Laboratories America Inc

BIOGRAPHICAL SKETCH

Author:	Xiaoyuan Liang
Degree:	Doctor of Philosophy
Date:	August 2019

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, 2019
- Bachelor Degree of Engineering in Information Security
  Harbin Institute of Technology, 2013

Major: Computer Science

Presentations and Publications:

Xiaoyuan Liang, Martin Renqiang Min, Hongyu Guo, and Guiling Wang, "Learning k-way d-dimensional discrete embedding for hierarchical data visualization and retrieval," *The 28th International Joint Conference on Artificial Intelligence*, Macau, China, 2019 August.

Xiaoyuan Liang, Guiling Wang, Martin Renqiang Min, Qi Yi, and Zhu Han, "A deep spatio-temporal fuzzy neural network for passenger demand prediction," *The SIAM International Conference on Data Mining*, Calgary AB, Canada, 2019 May.

Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han, "A deep reinforcement learning network for traffic light cycle control," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1243-1253, 2019 January.

Xiaoyuan Liang, Guiling Wang and Zhu Han, "A low-cost collaborative indoor localization system based on smartphone platform," *The 14th IEEE International Conference on Green Computing and Communications*, Halifax NS, Canada, 2018 July.

Xiaoyuan Liang, Tan Yan, Joyoung Lee, and Guiling Wang, "A distributed intersection management protocol for safety, efficiency, and driver's comfort," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1924-1935, 2018 June.

Jie Tian, Yi Wang, Xiaoyuan Liang, Guiling Wang, and Yujun Zhang, “WA-MAC: a weather adaptive MAC protocol in survivability-heterogeneous wireless sensor networks,” *Elsevier Ad Hoc Networks*, vol. 67, pp. 40-52, 2017 December.

Xiaoyuan Liang, and Guiling Wang, “A convolutional neural network for transportation mode detection based on smartphone platform,” *The 14th IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, Orlando FL, USA, 2017 October.

Xin Gao, Jie Tian, Xiaoyuan Liang, and Guiling Wang, “ARPP: an augmented reality 3D ping-pong game system on Android mobile platform,” *The 23rd IEEE Wireless and Optical Communication Conference*, Newark NJ, USA, 2017 May.

Jie Tian, Xiaoyuan Liang, and Guiling Wang, “Deployment and reallocation in mobile survivability-heterogeneous wireless sensor networks for barrier coverage,” *Elsevier Ad Hoc Networks*, vol. 36, pp. 321-331, 2016 January.

Xiaoyuan Liang, Jie Tian, Xiaoning Ding, and Guiling Wang, “A risk and similarity aware application recommender system,” *Journal of Computing and Information Technology*, vol. 23 (4), pp. 303-315, 2015 December.

Jie Tian, Xiaoyuan Liang, Tan Yan, Mahesh Kumar Somashekar, Guiling Wang, and Cesar Bandera, “A novel set division algorithm for joint scheduling and routing in wireless sensor networks,” *Springer Wireless Networks*, vol. 21, pp. 1443-1455, 2015 July.

To my lovely wife, Yan! This cannot be this easy without your support!

ACKNOWLEDGMENT

Foremost, I would like to express my sincere gratitude to my advisor, Dr. Guiling Wang for the continuous support and guidance of my Ph.D. study and research. Her patience, enthusiasm, knowledge and insightful vision has provided me endless interesting thoughts in the research and life throughout my whole Ph.D. studies in these years and will continue to influence me in my future life to explore the research and the sea of knowledge.

Besides my advisor, I would like to thank Dr. Zhu Han and Dr. Martin Renqiang Min, for their immense knowledge and critical suggestions, not only on my research, but also the attitude to research. They taught me to focus on the things that will make a real impact, which is a precious experience for me. Thanks also goes to my dissertation committee members, Dr. Ali Mili and Dr. Zhi Wei, for taking time from their busy schedules to review my dissertation and provide constructive comments. Their feedback were important for me to finish this dissertation.

I would also like to thank all my colleagues and collaborators, Dr. Hongyu Guo, Dr. Tan Yan, Dr. Jie Tian, Dr. Xin Gao, Mr. Yuchuan Zhang, and Mr. Xunsheng Du, for their hard work and contribution in finishing the academic papers. Thanks also to all my friends for their time and help.

Lastly, special thanks are given to my parents, Youde Liang and Qingbi Duan, for their upbringing, understanding and support all the time, from when I was born to now, through all my hardship and happiness. Thank my brother, Xiaolin Liang, for his support to the family. Hope his child, Yinuo, has a happy life! All my family members are thanked for their care and support in the past years.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Background on Deep Learning

Deep learning [60, 68, 70, 42] has been successfully applied in many fields, such as computer vision [32, 73, 83] and natural language processing [17, 79, 92, 103]. Deep learning is usually composed of multiple processing layers to learn representations of data with multiple levels of abstraction [60]. Among all DL techniques, Convolutional Neural Networks (CNNs) [59] and Recurrent Neural Networks (RNNs) [98] are undoubtedly two popular models. A CNN is composed of one or more convolutional layers with fully connected layers on top. CNNs are generally the method of choice to process the data by moving windows to capture the obvious features in windows. CNNs [68] are often deployed to model data with spatial feature interaction while RNNs are often used to process data with temporal feature interaction. An RNN is a class of artificial neural networks in which connections between units form a directed cycle. It creates an internal state of the network, which allows it to exhibit dynamic temporal behaviors. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. A special RNN called LSTM is widely adopted to overcome the vanishing gradient problem in traditional RNNs [42].

Reinforcement learning is another category of algorithms in machine learning and is different from supervised learning and unsupervised learning [112, 64]. It interacts with the environment to get rewards from actions. Its goal is to take the action to maximize the numerical rewards in the long run. In reinforcement learning, an agent, the action executor, takes an action and the environment returns a numerical reward based on the action and current state. Combining deep learning and reinforcement learning enables the system to learn complex information from the environment for better decisions. Reinforcement learning is widely applied into games to achieve higher scores than humans [107, 108].

1.2 Passenger Demand Prediction

Accurate future passenger demand prediction is very important in the field of transportation [69, 102]. Knowing the future demands, a Transportation Network Company (TNC) can wisely pre-allocate resources (vehicles and drivers) to meet the demands, such that the best service can be provided to passengers with a minimum waiting time, and unnecessary driving around on road can be prevented, reducing energy consumption and traffic jam. However, passenger demand prediction is very challenging considering the future demands are simultaneously influenced by many factors, including spatial and temporal factors, as well as many discrete external factors, such as weather and being daytime or nighttime [69]. These factors have complex and non-linear interactions with future demands and capturing the interactions in one model to make prediction is very difficult. Moreover, data of external factors are often either inaccurate or too coarse due to data-collection sensors' sparse deployment and unavoidable errors.

To predict passenger demand in the near future, previous studies have proposed various models. One of the most well-known methods uses the Auto-Regressive Integrated Moving Average (ARIMA) [102, 85]. ARIMA-based methods only consider the temporal correlation among the passenger demands, which limits the prediction performance. Recent studies [141, 136] propose deep learning models considering both temporal and spatial feature interactions, which outperform previous methods considering only one type of factor. However, the work handles the spatial and temporal feature interaction sequentially, resulting in information loss. In addition, they ignore other important discrete external factors, e.g., the weather. Although the model proposed in [50] considers the weather impact, it fails to consider inaccuracies within the collected data. The challenges become the accurate passenger demand prediction fusing all the related information into one model.

A deep **Spatio-Temporal Fuzzy Neural Network (STEF-Net)** is proposed to predict the passenger demand in the near future, which fuses all correlated information in one model. This model contains the following contributions: (1) It is the first model that combines a fuzzy neural network and deep learning techniques into one to handle data uncertainty and learn complex interaction among multiple factors. (2) It is composed of a new feature fusion method using a convolution operation, which can preserve the temporal relations of the outputs, capture the spatial information and achieve better performance than commonly used weighted addition. (3) It provides explainable results on when the historical information influences most in the prediction and how the weather can influence the prediction via adopting an attention layer on every time step. (4) Extensive experiments are conducted on real data to show that this model significantly outperforms state-of-the-art models in prediction accuracy.

1.3 Transportation Mode Detection

The information of travelers' transportation modes is important for developing many transportational applications. For example, (1) knowledge of people's transportation modes is useful for improving urban transportation planning [39, 46]. The new method would transform the way how transportation demand information is gathered for supplementing the traditional information acquisition practice based on telephone interviews and questionnaires, which is expensive and time consuming to conduct [110]. (2) The knowledge can also improve the performance of localizing and positioning systems. With the awareness of transportation modes, localizing systems can more precisely narrow down users' location [39]. (3) The information facilitates targeted and customized advertisements and services [110] based on the transportation modes the users are taking. (4) The acquired information can also help improve smartphone users' physical habits for environment protection purpose. For example, the CO_2 footprint as well as the calories burned by individuals can be better monitored with the information [75, 46]. In this way, the data can help users build green transportation habits to protect the environment.

Most previous studies use data from the Global Positioning System (GPS) to detect people's transportation modes [110, 128, 75]. However, GPS-based methods suffer from the following drawbacks [39, 46]: (1) GPS signals are not available everywhere. GPS requires an unobstructed view to satellites, limiting its applicability in metropolitan areas with highrises or in shielded areas; (2) a GPS sensor consumes a significant amount of energy and may rapidly deplete the battery of a mobile device. To address these issues, some existing work uses alternative sensors to detect transportation modes. For example, Jahangiri *et al.* [46] propose leveraging an accelerometer coupled with a gyroscope and a rotation vector sensor to detect five transportation modes. Fang *et al.* [23] use a deep neural network to classify five transportation modes based on the data from the accelerometer, magnetometer and gyroscope. Hemminki *et al.* [39] propose using an accelerometer sensor to detect six transportation modes. However, the detection accuracy of the above work, less than 90%, still needs improvement when only using the accelerometer sensor. In contrast, the proposed approach can detect all common transportation modes, including being stationary, walking, bicycling, taking bus, driving a car, taking subway, and taking train, using data only acquired from accelerometer sensors in smartphones. Thus the challenges exist in how to improve the accuracy with low energy consumption, which is applicable to all smartphones.

The proposed transportation mode detection system on smartphones is composed of three parts, data collection, data pre-processing and deep learning model. The data collection decides what data to collect and how the data are collected. The pre-processing part deals with the fluctuation caused by smartphones' different places and orientation. The core part, deep learning model, is essentially a convolutional neural network on a series of one-dimension acceleration magnitude. This system is the first attempt to adopt a deep convolutional neural network framework on the accelerometer data only to detect transportation modes in near realtime, about 1 second. Different from many existing systems, this system can be widely applied due to the availability of accelerometers in smartphones. The system is energy efficient by only using the accelerometers since accelerometers consumes much less energy than other types of motion sensors [33]. Comparing to the work that only uses accelerometers, the proposed system is robust to the position and orientation of smartphones since the magnitude instead of the vector in three axes is used, which greatly improves the user experience. The proposed system is proven to outperform existing studies and accurately detect transportation modes via extensive experiments when the same window size is taken on the same dataset.

1.4 Traffic Light Control

An efficient intersection management is critical to improve the road safety. The intersection management of busy or major roads is primarily done through traffic lights, whose inefficient control causes numerous problems, such as long delay of travelers and huge waste of energy. Even worse, it may also incur vehicular accidents [86, 71]. Existing traffic light control either deploys fixed programs without considering real-time traffic or considering the traffic to a very limited degree [10]. The fixed programs set the traffic signals equal time duration in every cycle, or different time duration based on historical information. Some control programs [9] take inputs from sensors such as underground inductive loop detectors to detect the existence of vehicles in front of traffic lights. However, the inputs are processed in a very coarse way to determine the duration of green/red lights.

To improve the efficiency of traditional traffic lights, adaptive traffic light control systems are studied in [76, 35, 140], which dynamically adjusts the cycle length of traffic lights based on traffic information estimated from loop detector sensors [116] or cameras. Early work defines the states by the number of waiting vehicles or the waiting queue length [22, 2]. But real traffic situation cannot be accurately captured by the number of waiting vehicles or queue length [29]. With the popularization of VANETs and cameras, more information about roads can be extracted and transmitted via the network, such as vehicles' speed and waiting time [37]. However, more information causes the dramatically increasing number of states. When the number of states increases, the complexity in a traditional reinforcement learning system grows exponentially. With the rapid development of deep learning, deep neural networks have been employed to deal with the large number of states, which constitutes a deep reinforcement learning model [84]. A few recent studies have proposed to apply deep reinforcement learning in the traffic light control problem [63, 121]. But there are two types of main limitation in the existing studies: (1) the traffic signals are usually split into fixed-time intervals, and the duration of green/red lights can only be a multiple of this fixed-length interval, which is not efficient in many situations; (2) the traffic signals are designed to change in a random sequence, which is not a safe or comfortable way for drivers. Thus, the challenges rely on how to control the timing in a whole cycle using the collected road traffic information.

The proposed traffic light control system is a real-time system, which learns the timing in a cycle from the current road traffic information. The core part is a deep reinforcement learning model based on Q learning. The proposed **Deep Dueling Double Q Network (3DQN)** combines deep learning and reinforcement learning into one model and improves the performance with prioritized experience replay, target network, double network and dueling network. This model is the first one to combine dueling network, target network, double Q network and prioritized experience replay into one framework to solve the traffic light control problem, which can be easily applied into other problems. The proposed model is the first one to decide the phases' time duration in a whole cycle instead of dividing the time into segments. Extensive experiments on a traffic micro-simulator, Simulation of Urban MObility (SUMO) [54], show the effectiveness and high-efficiency of the proposed model.

1.5 Embedding Visualization and Retrieval

Data embedding methods have been successfully deployed in many real-world applications, including unsupervised and supervised data visualization [73, 83, 82], natural language understanding [79, 92, 103], computer vision [26], information retrieval [16], bioinformatics analysis [21], and many others. In a neural network, word embedding is designed as real-valued vectors, while each word is assigned an independent integer vectors, such as one-hot vectors.

Existing embedding strategies, however, fail to sufficiently reveal essential semantic structures of the data in the embedded space. Typically, these methods associate a real-valued embedding vector with each symbol or data point, which is equivalent to applying a linear transformation to "one-hot" encoding of discrete symbols or data points. Despite their simplicity, these methods are incapable of encoding the internal semantic structure of data, failing to effectively preserve the interplay of the symbols/data points in the embedded space, such as the hierarchical relationship of the symbols or data samples. Hierarchical clusters of data will allow one to know how the symbols/data points are grouped and how lower layer groups form upper layer clusters. Such structural information is, therefore, critical for data understanding and fast information retrieval. In this field, exploring the semantic neighboring information via visualization is a challenging and promising problem.

The proposed **Hierarchical K-way D-dimensional (HKD)** codes employ an auto-encoder framework to transfer embeddings into discrete codes. The codes are learnt via the reconstruction and KL divergence, which are both associated with decaying weights. The learnt codes guarantee the nearest neighbors share the same codes in the front dimensions and are differentiated in the last few dimensions. The proposed auto-encoder framework is the first one to learn hierarchical discrete embedding, which enables hierarchical data visualization and fast nearest neighbor retrieval in addition to embedding storage efficiency. These salient features make the proposed embedding strategy particularly attractive in practice, where both the computation power and storage resources may not be abundant.

1.6 Structure

The remaining of the thesis is structured as follows, Chapter 2 introduces the related work. Chapter 3 presents the details of STEF-Net for passenger demand prediction. Chapter 4 shows a deep neural network for transportation mode detection on smartphones. Chapter 5 discusses the deep reinforcement learning model, 3DQN, to control traffic lights. Chapter 6 explores the embedding visualization and nearest neighbor retrieval using deep auto-encoders. The dissertation is finally summarized in Chapter 7.

CHAPTER 2

RELATED WORK

This chapter presents the related work in the domain of passenger demand prediction (Section 2.1), transportation mode detection (Section 2.2), traffic light control (Section 2.3) and embedding visualization and retrieval (Section 2.4).

2.1 Passenger Demand/Traffic Flow Prediction

Deep Learning and Fuzzy Learning: Deep learning [60, 69, 67] has been successfully applied in many fields, such as computer vision and natural language processing [32]. Among all deep learning techniques, CNNs [59] and RNNs [98] are two popular models. CNNs [68] are often deployed to model data with spatial feature interaction. A CNN is composed of one or more convolutional layers with fully connected layers on top. It also uses tied weights and pooling layers. In particular, maxpooling is often used in Fukushimas convolutional architecture [27]. This architecture allows CNNs to take advantage of the 2-dimension structure of the input data. RNNs are often used to process data with temporal feature interaction. An RNN is a class of artificial neural networks in which connections between units form a directed cycle. It creates an internal state of the network, which allows it to exhibit dynamic temporal behaviors. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. A special RNN called LSTM is widely adopted to overcome the vanishing gradient problem in traditional RNNs [42]. However, neither CNNs nor LSTMs are perfect models for addressing spatial-temporal problems. Except CNNs and LSTMs, *attention* [123] is another modeling mechanism to capture temporal relevance among sequences, which is common in natural language processing.

To handle imperfect data, fuzzy learning is a powerful tool and shows better performance than deterministic methods [134]. Combining fuzzy theory and neural networks can improve complex data representation with probability distribution over cross-layer units [139]. Even though they have been widely applied in control systems [139] and portfolio management [20, 78]. no existing work applies fuzzy neural networks in demand prediction. Fuzzy learning can be fused into a neural network in two ways: One is to learn different data representation from both fuzzy learning and neural network and then to fuse their outputs. One example of this kind is done by Deng *et al.* [20]. They apply deep neural network and fuzzy learning on the data at the same time and fuse their outputs in a model. The other is to sequentially apply fuzzy and neural network [139].

Passenger Demand/Traffic Flow Prediction: Passenger demand prediction is closely related to traffic flow prediction. Both have the same-format data and are influenced by the same complex factors. Both are reviewed in this section. Traditional approaches to predict future passenger demands only consider temporal information, such as ARIMA [102, 85] or ANN [11]. These studies do not consider the spatial correlation, which cannot accurately predict the future demands.

Recent advances in deep learning [60] motivate researchers to apply deep learning techniques for passenger demand and traffic flow prediction. Recent studies employ CNNs to capture complex spatial feature interaction [136] or RNNs (including LSTMs) to capture temporal feature interaction [138, 133]. Some pioneering work combines CNNs and RNNs to capture both spatial and temporal feature interaction in the data recently. Yao et al. [132] propose a multi-view model, which employs a CNN and an LSTM to capture the spatial and temporal feature interaction sequentially, but not simultaneously, which potentially leads to information loss. The above work either captures only one of the spatial and temporal feature interaction or captures both sequentially. None of the methods fully captures spatial-temporal feature interaction simultaneously.

ConvLSTMs [104] are another deep learning model, which combines CNNs and LSTMs. A ConvLSTM can simultaneously capture the spatial and temporal feature interactions. It replaces the fully connected layer in the traditional LSTM with a convolutional layer, which shows better performance than the traditional LSTM in precipitation nowcasting. The follow-up work by [50] uses ConvLSTMs to predict the passenger demands. The model is composed of a ConvLSTM and a LSTM to process the weather information, the travel time rate and demand intensity, and simply fuses the results from the two networks. However, The model fails to consider the inaccuracy of external data and the inaccuracy of simple prediction result fusion. AttConLSTM, a multi-step model built upon the attention-based encoder-decoder framework for passenger demand prediction is proposed in [141]. However, it fails to consider external factors, which greatly influences passenger demands. Different from all previous models, this dissertation's model captures spatio-temporal feature interaction simultaneously without information loss, employs a fuzzy neural network to handle external data inaccuracy and includes a new and effective feature fusion method based on convolution.

2.2 Transportation Mode Detection

There have been several efforts on activity detection using acceleration data. There are several studies using accelerometers only to detect transportation modes [87, 75, 39, 130]. Hemminki *et al.* [39] collect accelerometer data at the frequency from 60 to 100Hz and divide the data into 1.2-second windows with 50% overlap. They extract 27 features in every window and train an adaptive boosting model to classify the data into six modes, stationary, walk, bus, train, metro and tram. They achieve an accuracy of 80.1%. Manzoni *et al.* [75] collect accelerometer data at the frequency of 25Hz and divide it into windows of 10.24-second length with 50% time overlap. They also extract features from the FFT coefficients in every window and train a decision tree model to classify the data into eight modes, walk, bicycle, bus, car, metro, train, still, and motorcycle. They achieve an accuracy of 82.14%. Yang [130] collects accelerometer data at the frequency of 36Hz and divide the data into 10-second windows with 50% overlap. Features are extracted from both time and frequency domains and a decision tree model is used to classify six transportation modes, sitting, standing, walk, run, bicycle, and car. The accuracy is 90.6%. Compared with the existing studies using acceleration data, the proposed system can provide higher accuracy. Existing studies usually use the traditional machine learning methods to detect transportation modes. In this work, it is shown that CNNs outperform traditional machine learning methods in detecting transportation modes. In addition, among the traditional methods, the random forest performs best in the accuracy metric instead of other ones used in the existing work.

The work in [7] fuses the data from five biaxial accelerometers fixed at five body parts to recognize users' activities, such as walking, sitting, standing and running. A similar work [77] compares the performance of the acceleration data from six body parts in recognizing standing, sitting, walking and so on. Another work [3] presents a system to recognize the sitting, standing, lying and walking by requiring a device fixed at users' waists. But the above work requires extra accelerometers. Kwapisz *et al.* propose a system in Android phones put in the front pants leg pocket to recognize users' 6 activities, such as sitting and walking [58]. Lee *et al.* use an HMM model to classify the activities [61] while Anguita *et al.* use a multiclass hardware-friendly SVM [4]. A fusion system of motion sensors is proposed to recognize physical activities in [105]. However, none of the above work focuses on detecting users' transportation modes using deep learning models.

Some other related work may use other sensors and extra information to detect transportation modes. Specifically, some studies employ deep learning to detect transportation modes [124, 126, 109, 23]. All of these studies use the data from GPS or other sensors, which are not comparable to this study in terms of energy consumption. In addition, Wang *et al.* [126] (74.1% in unknown number of transportation modes) and Fang *et al.* [23] (95.43% in five modes) use the deep neural network with fully-connected layers to perform the detection while Vu *et al.* [124] (93.1% in five modes) and Song *et al.* [109] (83.26% in five modes) employ recurrent neural networks to discover the relation among close samples. The work uses the information from other sensors, such as GPS and Gyro, which is different from this chapter's work only using the accelerometer. None of them considers applying CNNs in detecting transportation modes. The above work merges the motorised transportation modes into one cluster, which provides a coarse-grained service. A model based on sensor data from accelerometer, gyroscope, magnetic field, rotation vector, geomagnetic rotation vector, linear acceleration, and uncalibrated versions where applicable, is proposed in [53] to detect six transportation modes, including walk, bike, MRT, bus, car and stationary. The model employs four separate machine learning methods, Gaussian naïve Bayes, discriminant analysis, SVM and k-NN. It can achieve 96% accuracy on average. Su *et al.* [111] propose an online SVM model to detect six transportation modes on the data from all motion sensors, including accelerometer, gravity sensor, gyroscope, magnetometer, and barometer. They can achieve an accuracy of 97.1%. Reddy *et al.* [96] propose a fusion model with decision trees and Hidden Markov Model (HMM) using GPS and accelerometer data to classify stationary, walk, run, bicycle and motorized transportation. They achieve an accuracy of 93.6%. Feng *et al.* [24] discover that combining GPS and accelerometer can achieve higher accuracy than using GPS or accelerometer data only, and using accelerometer only has a higher accuracy than using GPS only in their model. Stenneth *et al.* [110]

build a random forest model combining GPS and Geographical Information System (GIS) to classify stationary, walk, bicycle, car, bus, and train. They achieve an accuracy of 93.5%. Compared with the existing related work, it is obvious that this dissertation's work provides even a higher accuracy with accelerometer only and it is more efficient with only one low power-consuming sensor.

2.3 Traffic Light Control

Previous work has been done to dynamically control traffic lights. But due to the limited computing power and simulation tools, early studies focus on solving the problem by fuzzy logic [14], linear programming [90], etc. In these studies, road traffic is modeled by limited information, which cannot be applied in large scale.

With the success of deep learning in artificial intelligence, more and more researchers use deep learning to solve transportation problems. Deep learning includes supervised learning, unsupervised learning and reinforcement learning. In the traffic light control problem, since no labels are available and the traffic scenario is influenced by a series of actions, reinforcement learning is a good way to solve the problem and has been applied in traffic light control since 1990s. El-Tantawy *et al.* [22] summarize the methods from 1997 to 2010 that use reinforcement learning to control traffic light timing. During this period, the reinforcement learning techniques are limited to tabular Q learning and a linear function is normally used to estimate the Q value. Due to the technique limitation at the time in reinforcement learning, they usually make a small-size state space, such as the number of waiting vehicles [2, 72] and the statistics of traffic flow [5, 6]. A signal control system is proposed in [94]. The authors use the queue length and current light time as the state and use a linear function to approximate the Q values. A cooperative traffic light control system based on reinforcement learning is proposed in [72]. The authors propose to cluster vehicles and use a linear function to approximate the Q values; however, only the queue information is used in the states. The complexity in a traffic road system can not be actually presented by such limited information. When much useful relevant information is omitted in the limited states, it seems unable to act optimally in traffic light control [29].

With the development of deep learning and reinforcement learning, they are combined together as deep reinforcement learning to estimate the Q value. Some researchers have applied deep reinforcement learning to control the wireless communication [142, 115], but the systems cannot be directly applied in traffic light control scenarios due to different actions and states. Here is the summarization of the recent studies that use the value-based deep reinforcement learning to control traffic lights in Table 2.1. There are three types of limitation in these previous studies. Firstly, most of them test their models in a simple cross-shape intersection with through traffic only [63, 121]. Secondly, none of the previous work determines the traffic signal timing in a whole cycle. Thirdly, deep reinforcement learning is a fast developing field, where a lot of new ideas are proposed in these two years, such as dueling deep Q network [127], but they have not been applied in traffic control. In this dissertation, the following progress is made. Firstly, the intersection scenario contains multiple phases, which corresponds a high-dimension action space in a cycle. Secondly, the proposed model guarantees that the traffic signal time smoothly changes between two neighboring actions, which is precisely defined in the MDP model. Thirdly, the proposed model employs the state-of-the-art techniques in value-based reinforcement learning algorithms to achieve good performance.

2.4 Embedding Visualization and Retrieval

Data embedding methods have been successfully deployed in many real-world applications, including unsupervised and supervised data visualization [73, 83, 82], natural language understanding [79, 92, 103], computer vision [26], information

Table 2.1 Previous Studies Using Value-based Deep Reinforcement Learning

Study	*State*	*Action*	*Reward*	*Time step*	*Note*
Genders *et al.* (2016) [29]	Position, speed	4 phases	Change in cumulative delay	NA	Convolutional neural network
Li *et al.* (2016) [63]	Queue length	2 phases	Difference between flows in two directions	5s	Stacked auto-encoders
Van Der Pol (2016) [121]	Position	2 phases	Teleport, wait time, stop, switch, and delay	1s	Double Q network, Prioritized experience replay
Gao *et al.* (2017) [28]	Position, speed	4 phases	Change in cumulative staying time	6/10s	Convolutional neural network, experience replay

retrieval [16], bioinformatics analysis [21], and many others. The proposed method is built on the success of the recent K-way D-dimensional discrete encoding [13, 106]. These discrete encoding algorithms encode, through deep neural networks, data points with discrete codes, thus being able to significantly reduce the storage space when compared to real-valued embedding.

Conventional methods considering the hierarchical structure in data retrieval are usually based on Huffman coding [44]. But Huffman codes do not contain the semantic information and are not suitable in a large embedding system. In most embedding systems, such as word embedding [31, 92], one-hot vectors are commonly used to denote every symbol and a linear transformation is used to get embedding from vectors. Even the continuous values in the embedding can represent the semantic information in the dataset, the one-hot vectors [36] consumes a large size of storage and the transformation matrix is also large. Some recent work [51, 101, 137] explores character or sub-word based embedding to achieve smaller vocabulary size and better performance. A hash function [113] is proposed to automatically map texts to pre-defined bases with a smaller vocabulary size. But the above work is very limited for fixed characters and sub-words based on a given language, which cannot capture all semantic features in other types of data. In this work, the KD codes are learned for almost any embedding system. The work most related to this work is the KD code learning as introduced in [106, 13]. The work in [106] provides an autoencoder framework to learn the KD codes while [13] proposes an end-to-end KD code learning framework for different tasks, where the KD codes are learned to minimize task-specific losses. Another related work is about the k-ary tree [95], which builds a tree structure for the entities, but k-ary tree requires the children size to be either 0 or k. In this dissertation, the number of children can be any number from 0 to k, which greatly helps to capture the semantic information. In contrast, a novel autoencoder regularizer is introduced to force the autoencoder for the hierarchical structures while generating discrete embedding codes. This framework can be worked on any existing embedding to extract the hierarchical semantic information.

The semantic information learning from high dimension to low dimensions is usually implemented by pairwised methods, such as SNE [40] and t-SNE [73]. However, the computation complexity of t-SNE is very high, which limits their applicability to small datasets. Some improvements are proposed to reduce the complexity, like BarnesHut trees [119], fast multipole methods [30] and many others [120, 131], but the above work does not reach to a linear complexity. Based on t-SNE, pt-SEE [81] is proposed to use exemplars as cluster information to reduce the computation complexity to a linear complexity. pt-SEE only provides the distribution for continuous values in the both high and low dimensions. Thus, it cannot be directly applied in this work. What's more important, pt-SEE equally treats every unit, which cannot be directly used for hierarchical structures. The proposed autoencoder regularization schema builds on the pt-SEE strategy [81], where forming embeddings are encouraged to cluster around the prototypes or exemplars. Nevertheless, this method treats all embedding dimensions equally when computing their distance with the exemplars, thus discards the semantic structures of those data points. The proposed approach here treats each embedding dimensions with different weights which correspond to their closeness with different exemplars. Those distances directly reflect their relationship with other data points.

CHAPTER 3

A DEEP SPATIO-TEMPORAL FUZZY NEURAL NETWORK FOR PASSENGER DEMAND PREDICTION

3.1 Introduction

Accurate future passenger demand prediction is very important in the field of transportation. Knowing the future demands, a TNC can wisely pre-allocate resources (vehicles and drivers) to meet the demands, such that the best service can be provided to passengers with a minimum waiting time, and unnecessary driving around on road can be prevented, reducing energy consumption and traffic jam. However, passenger demand prediction is very challenging considering the future demands are simultaneously influenced by many factors, including continuous spatial and temporal factors, as well as many discrete external factors, such as weather and being daytime or nighttime. These factors have complex and non-linear interactions with future demands and capturing the interactions in one model to make prediction is very difficult. Moreover, data of external factors are often either inaccurate or too coarse due to data-collection sensors' sparse deployment and unavoidable errors. The challenging nature of future passenger demand prediction requires special handling and many existing modeling of spatio-temporal forecasting cannot be directly applied.

To predict passenger demands in the near future, previous studies have proposed various models. One of the most well-known methods uses the ARIMA [102, 85]. ARIMA-based methods only consider the temporal correlation among the passenger demands, which limits the prediction performance. Recent studies [141, 136] propose deep learning models considering both temporal and spatial feature interaction, which outperform previous methods considering only one type of factor. However, they handle the spatial and temporal feature interaction sequentially, resulting in information loss. In addition, they ignore other important discrete external factors, e.g., the weather. Although the model proposed in [50] considers the weather impact, it fails to consider the uncertainty within the collected data.

To tackle this challenging problem with desirable performance, a deep STEF-Net is proposed to predict the passenger demand in a city area. In this network, all related factors are fused to model the complex interaction among them, including spatial-temporal dependencies, external information and temporal relevance, and design an end-to-end learning framework with different neural networks modeling different types of feature interaction. Specifically, the proposed model simultaneously captures the spatial and temporal dependencies via a ConvLSTM. A ConvLSTM replaces the full connection in a traditional LSTM with the convolutional operation such that the spatial and temporal feature interactions can be simultaneously captured and information loss can be avoided compared to sequential processing by stacking convolutional layers and LSTMs. Regarding the uncertainty of external factors, a fuzzy neural network is proposed. A fuzzy neural network, which combines the fuzzy theory and neural networks, can learn the feature representation with high error tolerance and trainable rules. It shows significantly better performance than deterministic neural networks for this data type. A new feature fusion method using convolution is proposed to connect the two separate networks without losing temporal information. An attention layer is further employed to capture the temporal relevance of the high-level fused data, considering the future demands are unequally influenced by past ones. The proposed model is evaluated on the real data from Didi Chuxing, the biggest TNC in China, which is similar to Uber in the United States. The experimental results show that the proposed model outperforms the state-of-the-art models.

In summary, the contributions of this chapter include: (1) This work is the first to combine a fuzzy neural network and deep learning techniques to handle data uncertainty and learn complex interactions among multiple factors, which can achieve better performance than solely using deep learning. (2) A new feature fusion method using convolution is proposed, which can preserve the temporal relation of the outputs, capture the spatial information and achieve better performance than commonly used weighted addition. (3) An attention layer is adopted for the first time on every time step to provide explainable results on when the historical information influences most in the prediction and how the weather can influence the prediction. (4) Extensive experiments are conducted on real data to evaluate the proposed model. The proposed model significantly outperforms state-of-the-art models in prediction accuracy.

3.2 Problem Formulation

Being consistent with existing work, the following definitions are made. Based on the definitions, the problem statement of this chapter is given.

Definition 3.2.1 Region In this chapter, the passenger demands in different areas in a city are predicted. The whole city is partitioned into $W \times H$ equal-size grids. A grid is called a region, which is denoted by r. Let $r_{i,j}$ denote the region with the coordinate i, j, where $i \in [0, W)$ and $j \in [0, H)$.

Definition 3.2.2 Service request A service request s_k made by a passenger is composed of the request ID, pick-up coordinates $s_{k,pc}$ (longitude and latitude), and pick-up time $s_{k,pt}$. (Note that drop-off location and time are not considered for demand predictions.) A service request $s_k = \{s_{k,pc}, s_{k,pt}\}$. A valid request's pick-up location should be in the city. If it is outside the city, it is discarded. The total number of available legal requests is denoted by N.

Definition 3.2.3 Passenger demands Time is divided into equal intervals. The t^{th} time interval, starting from 0, is the interval of $[t \times C, (t+1) \times C)$, where C is a constant representing the interval's time span. The passenger demand of a region $r_{i,j}$ is accumulated in the specific t^{th} time interval based on the requests' pick-up time. The passenger demand in region $r_{i,j}$ at the t^{th} time interval is denoted by $d^t_{i,j}$.

$$d^t_{i,j} = |\{k \in [0, N) : s_{k,pc} \in r_{i,j} \wedge s_{k,pt} \in [t \times C, (t+1) \times C)\}|. \tag{3.1}$$

The whole area's demands are denoted by $\boldsymbol{D}$, which means

$$\boldsymbol{D}^t = \{d^t_{i,j} | \forall i \in [0, W), \forall j \in [0, H)\}. \tag{3.2}$$

It can be imagined that $\boldsymbol{D}^t$ is a demand snapshot of the whole area at the t^{th} time interval, where every pixel is the demand of that particular location.

Definition 3.2.4 External information Let $\boldsymbol{e}^t$ denote the external information set at the t^{th} time interval. The external factors impacting the passenger demands considered in the paper includes the weather, the day in a week and being daytime or nighttime. The process details of the external information will be given in next section.

Problem statement The problem is defined as follows. Suppose the current time interval is t. Given the historical passenger demands and the external information at the t^{th} time interval, the goal is to predict the passenger demands in all regions in the city at the $(t+1)^{\text{th}}$ time interval. Specifically, in this problem, the historical data, demands and external information in the last k time intervals are taken as input and the output is the predicted passenger demands at the $(t+1)^{\text{th}}$ time interval. Let $\hat{\boldsymbol{D}}^{t+1}$ denote the predicted passenger demands at the $(t+1)^{\text{th}}$ time interval. $\hat{\boldsymbol{D}}^{t+1}$ is a function f of the previous k time intervals' data.

$$\hat{\boldsymbol{D}}^{t+1} = f(\boldsymbol{D}^{t-k}, \boldsymbol{D}^{t-k+1}, ..., \boldsymbol{D}^t, \boldsymbol{e}^{t-k}, \boldsymbol{e}^{t-k+1}, ..., \boldsymbol{e}^t). \tag{3.3}$$

This chapter's goal is to minimize the difference between $\hat{\boldsymbol{D}}^{t+1}$ and the true passenger demands $\boldsymbol{D}^{t+1}$.

3.3 STEF-Net

3.3.1 Preliminary Analysis

In this section, a preliminary data analysis is conducted to provide some intuition on how passenger demands are influenced by different factors. A dataset from Didi

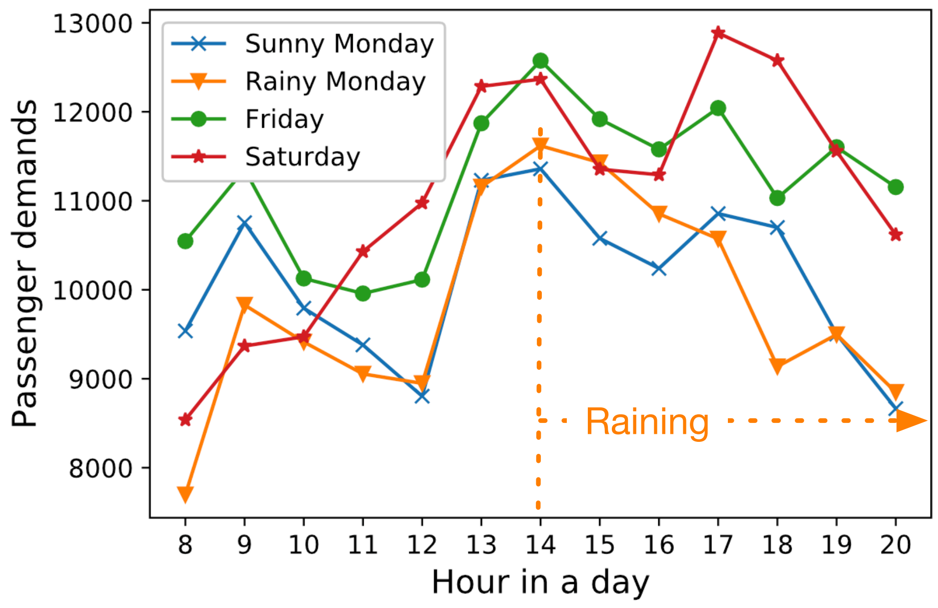


Figure 3.1 Illustration of different hourly passenger demands in different days.

Chuxing, China [1] is used to extract the passenger demand information. The data contains over 5.24 million service requests from 11/01/2016 to 11/30/2016. Figure 3.1 shows the total passenger demands over different hours in different days. Two Mondays and a Friday as representatives of weekdays and a Saturday as that of weekends are picked as examples. A rainy Monday and a sunny Monday are compared to show the impact of weather on passenger demands. Except the rainy Monday, all the other days are sunny. In the figure, the x axis is the hour of the day and the y axis is the passenger demands during the hour. It can be seen that the passenger demands have different patterns in different days, at different time of the day, and under different weathers. For example, at the noon time, the demand drops on Monday and Friday but it increases on Saturday. About the weather factor, on the rainy Monday, it starts to rain at 2pm. Comparing the sunny Monday with the rainy Monday, it is shown that the patterns of passenger demands on the two days before

[1]The source of raw data: DiDi Chuxing GAIA Open Data Initiative. Available at: https://gaia.didichuxing.com

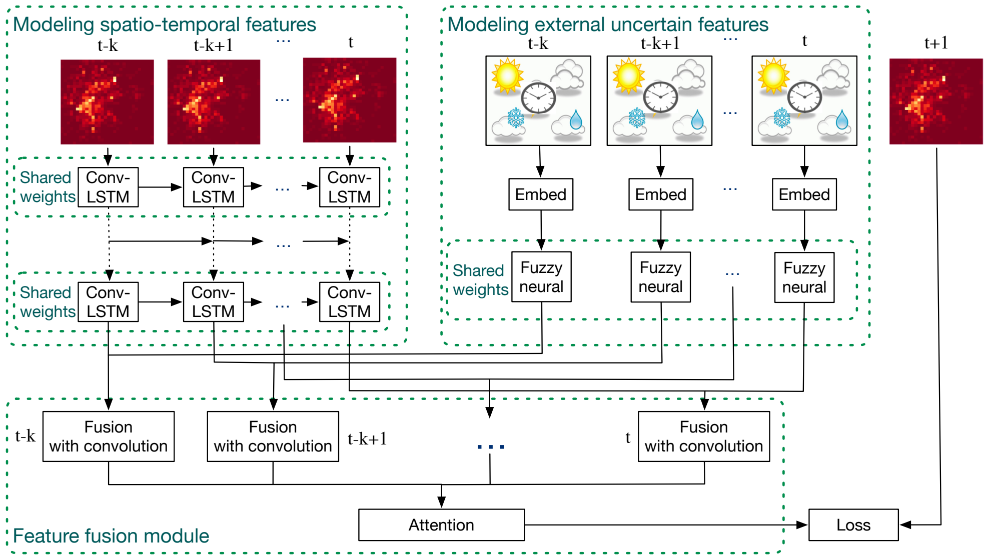


Figure 3.2 Proposed model for passenger demand prediction: STEF-Net.

rain are similar while the patterns become different after rain comes. Specifically, the number of passenger demand keeps decreasing on rainy Monday while it gets increased once on sunny Monday. Figure 3.1 shows the passenger demands are determined by complex interaction among many factors.

3.3.2 Overview of the Deep Learning Model

A deep learning model, STEF-Net, is proposed to predict the passenger demands incorporating the complex interaction between various factors. This model is illustrated in Figure 3.2, which is mainly composed of four components. (1) As shown in the left side of the figure, a stacking ConvLSTM is employed to capture the *spatial-temporal feature interaction* with the passenger demands. The input is the historical passenger demands with location and time information, and the output is the prediction using only spatial-temporal information. (2) As shown in the right side of the figure, in parallel, a fuzzy neural network is employed to capture the *external information's interaction* on passenger demands. The input is the data about external information and the output is the prediction using external information. (3) As shown on the bottom of the figure, the outputs from the stacked ConvLSTMs and the fuzzy neural network are fused into one network to generate the final output. A new feature fusion method using convolution on the data from the same time period is proposed, which keeps the temporal relation of the outputs from the two networks. Considering the passenger demands are unequally influenced by different time intervals, this model further adapts an attention layer on the high-level fused data to capture the temporal relevance. The data are then reshaped into the output format, which matches the regions in a city. (4) As shown in the right corner of the figure, a loss function is employed to measure the difference between the predicted value and true value. A neural network's goal is to minimize the loss defined by an objective function. In the following, every component is presented in detail.

3.3.3 Modeling Spatio-temporal Features

This model enables to simultaneously capture the deep spatial and temporal dependencies in passenger demands by stacking ConvLSTMs. A ConvLSTM is a neural network model that combines convolutional operation and LSTM units, where an LSTM is known to well handle temporal feature interaction without the vanishing gradient problem, while convolutional networks are known to gracefully handle spatial feature interaction. A ConvLSTM uses the convolution operation to replace the full connection in traditional LSTMs. In a traditional LSTM, a cell is composed of input gate i_t, forget gate f_t, memory cell status c_t, output gate o_t, and final state h_t. In a traditional LSTM, all the elements are 1D tensors, which accepts the input from $T \times L$ dimensions and generates outputs into $T \times L'$ dimensions. T is the length of the time sequences, L is the length of one input vector, and L' is the length of one output vector.

A ConvLSTM can be explained as follows. To make it general, let $\mathbf{x}_t$ denote the input at the t^{th} time step in the time sequence. The core idea is to transfer all the inputs, memory cell values, hidden states, and various gates into 3D tensors, where the first two dimensions are considered as the spatial information, rows and columns, and the last dimension is the channels. All the notation of weights and outputs are bold to present they are 3D tensors. The elements in a cell can be calculated as follows,

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}_{xi} * \mathbf{x}_t + \mathbf{W}_{hi} * \mathbf{H}_{t-1} + \mathbf{b}_i\right), \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{xf} * \mathbf{x_t} + \mathbf{W}_{hf} * \mathbf{H_{t-1}} + \mathbf{b}_f), \\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \sigma(\mathbf{W}_{xc} * \mathbf{x}_t + \mathbf{W}_{hc} * \mathbf{H}_{t-1} + \mathbf{b}_c), \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo} * \mathbf{x}_t + \mathbf{W}_{ho} * \mathbf{H}_{t-1} + \mathbf{W}_{co} \circ \mathbf{c}_t + \mathbf{b}_o), \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t).
\end{aligned} \tag{3.4}$$

In the above equations, $*$ is the convolution operation and $\circ$ is the Hadamard (element-wise) product. $\mathbf{W}$ and $\mathbf{b}$ are corresponding weights and biases, respectively, indexed for different calculation. σ and tanh are two activation functions under the input x given by following equations, respectively,

$$\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}}, \\
\tanh(x) &= \frac{e^{x}+e^{-x}}{e^{x}+e^{-x}}.
\end{aligned} \tag{3.5}$$

To adopt a ConvLSTM in this problem, the first two dimensions in the passenger demand data are treated as rows and columns at one time interval. The ConvLSTM in this problem can be considered as a function $R^{T\times W\times H\times L} \rightarrow R^{T\times W\times H\times L'}$, where T, L and L' are the same as those in the traditional LSTM, and W and H are the length of rows and columns, corresponding to the width and height of the grids in the city-wide area in this problem. Several ConvLSTM layers are stacked in the proposed neural network. In the last ConvLSTM layer, the length of the output vector is set 1, which means the output is a 4D tensor with the size of $T \times W \times H \times 1$. This is equivalent to a 3D tensor with the size of $T \times W \times H$.

3.3.4 Modeling External Uncertain Features

Passenger demands are influenced by many external factors in addition to location and time which are handled in the previous section. It is required to identify the factors and obtain corresponding data to train this model. Note that existing data about external factors are likely inaccurate or of coarse granularity, and thus it is proposed for the first time to employ a fuzzy neural network to model the data. In the section, the factor selection and data pre-processing is first present, and then the fuzzy neural network modeling is given.

Factor selection and data pre-processing The proposed model aims to include all highly correlated factors and chooses (1) the weather, (2) the day in a week, and (3) being daytime or nighttime. Obviously, weather greatly influences a passenger's choice between taking Uber and walking, waiting and then taking a bus. When it rains heavily or when it is very cold or very hot, people tend to take a more comfortable way of transportation. In this model, the weather is represented by temperature, dew point, humidity, pressure, wind speed, and weather condition. The first five variables are numerical variables. The last one, weather condition, is represented by ten different categories: clear, partly cloudy, scattered cloud, mostly cloudy, haze, light rain, shower, mist, patches of fog, and fog. The categories are indexed from 1 to 10. The numbers are embedded using one-hot vectors with ten dummy variables. The second factor, the day in a week, refers to the 7 days in a week, which also influences people's daily transportation behavior. This factor is a categorical variable and is represented by a one-hot vector with seven dummy variables. The third factor is whether it is dark outside and it is differentiated by the sunrise and sunset time of the day. In summary, to represent all the three factors, 24 variables are chosen in total in this model. The external information is a 1D tensor.

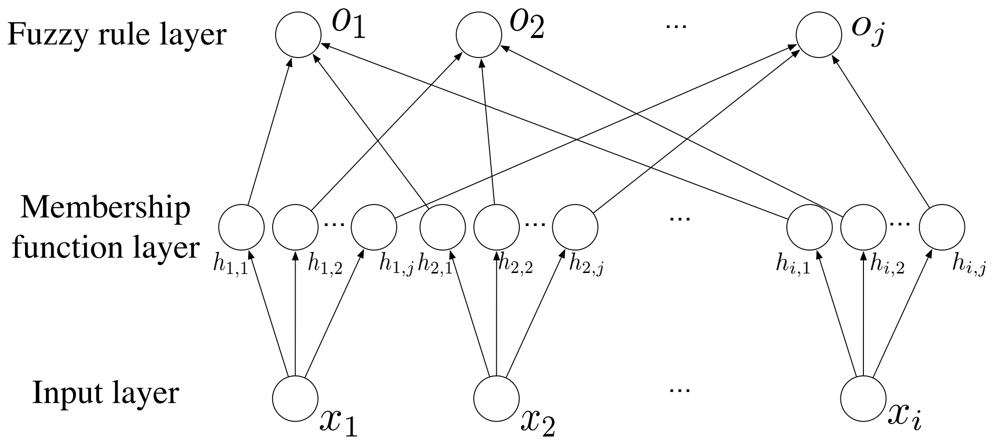


Figure 3.3 Illustration of the fuzzy neural network.

Fuzzy Neural Network Modeling A fuzzy neural network is adopted to learn the representation of the external information. The fuzzy neural network is composed of two hidden layers, membership function layer and logic rule layer. The architecture is shown in Figure 3.3. The membership function layer calculates the degree that an input node belongs to a certain fuzzy set. Let x_i denote the i^{th} element in the input. In the membership function layer, every element is split by multiple Gaussian distributions. Let j index the Gaussian distribution for the i^{th} element. One distribution is denoted by $(\mu_{i,j},\, \delta_{i,j})$. The membership function layer's output is calculated as follows,

$$h_{i,j} = e^{\frac{-(x_i-\mu_{i,j})^2}{\delta_{i,j}^2}}. \tag{3.6}$$

The logic rule layer performs the "AND" fuzzy logic operation as follows,

$$o_j = \prod_i h_{i,j}. \tag{3.7}$$

Through the rule layer, the output can present the probability that it is related to every unit. In the fuzzy neural network, all time intervals' external information shares the same member function layer and logic rule layer.

The output from the fuzzy neural network of one interval's external information is reshaped into two dimensions to match the passenger demands in a whole city, which is $W \times H$. The outputs from the past T time intervals' external information are a 3D tensor with the shape size of $T \times W \times H$.

3.3.5 Feature Fusion Module

In the above two sections, the passenger demand data and external information are mapped into the same feature representation format from two separate networks. They need to be combined to predict the next time interval's demand.

Previous work employs weighted addition [50, 136] to fuse the two components. In this model, convolution is employed to fuse the outputs from two networks. In addition, to keep the temporal feature interaction, the data from the same time interval are fused first and then an attention layer is employed to generate the final output.

The outputs of the passenger demand and external information at the t^{th} are denoted by $\mathbf{O}_{t,p}$ and $\mathbf{O}_{t,e}$, respectively. Let $\mathbf{O}_{t,f}$ denote the output after fusion with convolution denoted by $\oplus$. The illustration of fusion with convolution is shown in Figure 3.4. The calculation can be presented by the following equation,

$$\mathbf{O}_{t,f} = \mathbf{O}_{t,p} \oplus \mathbf{O}_{t,e}. \tag{3.8}$$

The two are concatenated by adding a new dimension, which can be imaged as the channel in a CNN. After concatenation, $\mathbf{O}_{t,f} \in R^{W \times H \times 2}$. To make the output's dimension consistent, a convolutional operation with window size $w \times h$ ($w \ll W \& h \ll H$) with 1 channel is applied, which outputs a 2D tensor with the size of $W \times H$. In this way, the fusion with convolution method only needs $w \times h$ parameters while the weighted addition requires $2 \times W \times H$ parameters in the previous work [50, 136]. In

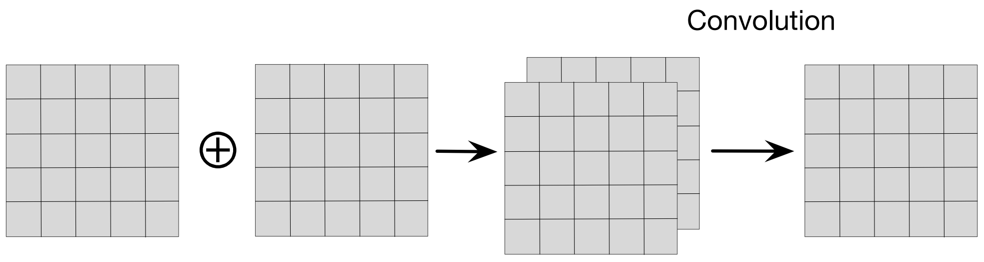


Figure 3.4 The fusion method with convolution.

addition, the convolutional operation can further learn the spatial information on the fused data.

After fusing the data, a bidirectional LSTM and attention are used to further capture the temporal relevance. In the bidirectional LSTM, the data are flattened into one dimension and fed into LSTMs:

$$\overrightarrow{\mathbf{h_t}} = \overrightarrow{LSTM}\left(\mathbf{w_t}, \overrightarrow{\mathbf{h_{t-1}}}\right), \tag{3.9}$$

$$\overleftarrow{\mathbf{h_t}} = \overleftarrow{LSTM}\left(\mathbf{w'_t}, \overleftarrow{\mathbf{h_{t+1}}}\right). \tag{3.10}$$

w_t and w'_t are the weights in the forward and backward LSTMs, respectively. The outputs from the forward and backward are added in an element-wise way:

$$\mathbf{h_t} = \overrightarrow{\mathbf{h_t}} + \overleftarrow{\mathbf{h_t}}. \tag{3.11}$$

The number of units in the LSTM is the same as the number of grids, which is equal to $W \times H$. All the time steps are concatenated into a matrix H:

$$\mathbf{H} = (\mathbf{h_1}, \mathbf{h_2}, ..., \mathbf{h_t}). \tag{3.12}$$

For all time steps' values in every grid, linear transformation and a *softmax* activation function are used to get the attention weights on the time step domain:

$$\mathbf{a} = \text{softmax}(\mathbf{WH}). \tag{3.13}$$

The outputs are the weighted sum of the hidden states and the attention weights in an element-wise way. The passenger prediction is the outputs after being weighted by attention.

3.3.6 Objective Function

In this model, the objective function is defined as the mean square error between the true passenger demands and predicted passenger demands. The model is trained based on mini-batches. Suppose there are m samples in a mini-batch and every sample is indexed by i, the objective function $L(\theta)$ with trainable parameters θ is defined as follows,

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m} \|\hat{\boldsymbol{D}}_i^{(t+1)} - \boldsymbol{D}_i^{(t+1)}\|^2, \tag{3.14}$$

It is the mean square error between the predicted and true passenger demand in a mini-batch. The optimization algorithm in this model is the ADAptive Moment estimation (Adam) [52], which adaptively changes the effective learning rate during training.

The training pipeline of the proposed model, STEF-Net, is presented in Algorithm 1.

3.4 Evaluation

3.4.1 Evaluation Objectives and Metrics

STEF-Net is evaluated by comparing it with state-of-the-art models on real data with regard to the accuracy in passenger demands prediction. The accuracy is measured by two metrics, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The MAE and RMSE are two widely employed metrics to evaluate the performance of a prediction system [136].

Algorithm 1: Training pipeline of STEF-Net

```
Input  : Historical observations: $D^1, D^2, ..., D^t, ....$
         External information: $e^1, e^2, ..., e^t, ....$
Output: A well learnt STEF-Net model
Notation:
$\theta$: parameters in the STEF-Net
$M$: Memory
1  Initialization
2  for $\forall t > k$ do
3      Generate training data D = $[D^{t-k}, D^{t-k+1}, ..., D^t]$
4      Embed external information E = $[e^{t-k}, e^{t-k+1}, ..., e^t]$
5      Generate a training sample <{D, E}, $D^{t+1}$> into memory $M$.
   end
6  Initialize the parameters $\theta$ in the STEF-Net
7  repeat
8      Randomly select a mini-batch of samples $m$ from memory $M$
9      Calculate the cost function $L(\theta)$ in Equation 3.14
10     Update parameters $\theta$ using Adam back propagation
   until convergence criterion met;
```

3.4.2 Dataset

The dataset used for training and testing is from Didi Chuxing [2]. The data contains over 5.24 million non-duplicating service requests from 11/01/2016 to 11/30/2016 in Chengdu City, China. In the dataset, every service request record is composed of the request ID, pick-up time, pick-up coordinates, drop-off time and drop-off coordinates. (Note that drop-off time and drop-off coordinates are not needed here.) The data of the first 23 days is used for training and that of the last 7 days (one week) for testing. The data from the dataset are extracted in a specific area in Chengdu City based on the pick-up coordinates, where the longitude is from 30.60E to 30.73E and the latitude is from 104.00N to 104.15N. The area is about 14.41km $\times$ 14.39km. The whole area is divided into 20$\times$20 same-size grids. The passenger demands are accumulated in every grid from the raw data. The length and width of every grid are both about 700 meters. The time interval is set half an hour. Same as previous studies [136, 141], the passenger demands are scaled into [0,1] using max-min scaling. In the final step, the demand values are recovered by the inverse of max-min scaling.

Regarding the external information, the day of a week is extracted from the pick-up time. The weather information and sunrise/sunset information are crawled from the Weather Underground website [118] using Python. The website provides historical weather information in Chengdu. As presented in previous section, 24 features about the external information are extracted.

3.4.3 Hyperparameters and Development Environment

In this network, 3 ConvLSTMs are stacked and all the ConvLSTMs use 64 filters of size 3$\times$3. A convolutional layer with one filter is added after the stacked ConvLSTMs to convert the data into a 20$\times$20 tensor. The membership layer in the fuzzy neural network is composed of 24$\times$400 units, and the fuzzy rule layer has 400 units. The

[2]The source of raw data: DiDi Chuxing GAIA Open Data Initiative. Available at: https://gaia.didichuxing.com (Accessed on June 26th, 2019).

outputs from the fuzzy neural network are reshaped into 20×20 and the following convolutional layer has one filter with the kernel size of 3×3. In the fusion part, the convolutional layer has one filter with the kernel size of 3×3. The output is reshaped into 20×20 to match the prediction in the whole city. The parameters in the fuzzy neural network are uniformly initialized from 0 to 1. All the other parameters are uniformly initialized.

In this model, to be consistent with the previous studies [50, 141], previous eight time intervals' data are by default used to predict the current time interval's passenger demands, It means the current time interval's passenger demands are predicted based on the historical 4 hours' data. 2 hours' data are also used to predict the demands to see how the time length can influence the prediction accuracy.

The proposed model is developed on the top of Keras [15] with the backend of Tensorflow [1]. The model is running on a desktop with an Intel Xeon 3.10GHz×4 CPU and a GeForce GTX 1050 Ti GPU. The model is trained by mini-batches. Every mini-batch has 16 samples. Every model is trained using 50 epochs and the results are generated after that.

3.4.4 Baselines

For a thorough comparison with existing methods, the proposed model is compared with three categories of methods, time-series (ARIMA), regression-based (Ridge and XGBoost) and neural network-based methods (ST-ResNet, AttConLSTM and FCL-Net). The methods are presented as follows:

- **ARIMA [85]:** ARIMA uses both moving average and autoregressive to predict the next time interval's passenger demands.
- **Ridge linear regression [43]:** Ridge linear regression uses a linear equation to model the relationship between historical features and future passenger demand. All features in this paper are reshaped into a vector and feed vectors into the linear regression.

- **XGBoost [12]:** XGBoost (2016) is a widely used boosting method with a tree structure. All features are also reshaped into vectors to feed the XGBoost model.
- **AttConLSTM[141]:** AttConLSTM (2018) fuses attention and ConvLSTM into an auto-encoder model. It stacks CNNs and ConvLSTMs to encode and decode the passenger demands and extracts passenger demands patterns as references in an attention network.
- **FCL-Net [50]:** FCL-Net (2017) employs ConvLSTM and LSTM to extract information from demands, time and weather. It fuses the outputs from two networks by addition.
- **ST-ResNet [136]:** ST-ResNet (2017) uses ResNet to capture the spatial and temporal information on demands from three categories, recent, near and distant. Only weather information at the current time interval is considered. It fuses data by addition. Because the distant demands require at least three weeks for one sample, the recent and near categories are taken.

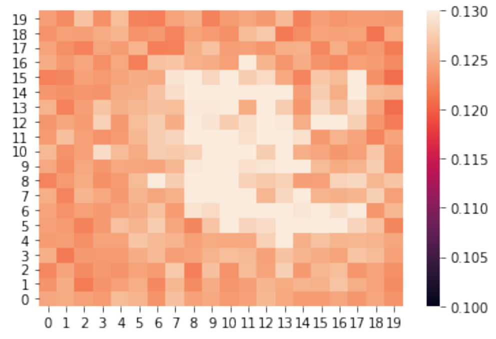
All the models follow the settings in their original papers, and all of them are trained 50 epochs.

3.4.5 Results

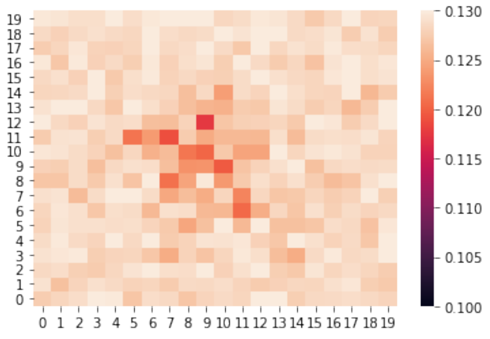
Comparisons with Baselines The comparison results are presented with baselines in Table 3.1. From this table, it is shown that the proposed model outperforms all the others regarding both metrics. When the historical data is 4 hours, the proposed model can achieve 3.89 in RMSE and 2.27 in MAE. The results at least 9.9% better in MAE and 11.3% better in RMSE than the best one among all baseline methods and about 34.6% better in MAE and 37.8% better in RMSE than the worst among all baselines. Ridge linear regression performs worst because it only considers the linear relation among features. Note that the models on 2 hour data and 4 hour data are trained to predict future demands, respectively. When more historical data is utilized to predict, the performance improves.

Table 3.1 Comparisons with Different Baselines

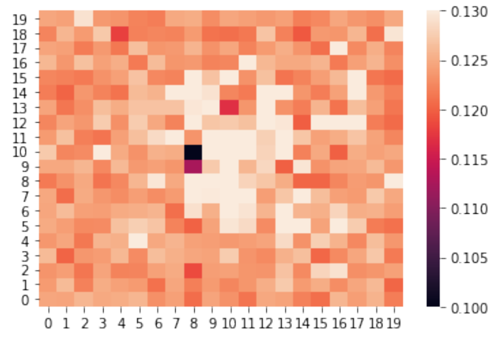
Model name	2 hours		4 hours	
	MAE	RMSE	MAE	RMSE
ARIMA	3.61	6.42	2.85	4.91
Ridge linear	3.50	6.32	3.47	6.25
XGBoost	3.48	6.18	3.29	5.87
ST-ResNet	2.90	5.15	2.86	5.02
AttConLSTM	2.63	4.58	2.60	4.55
FCL-Net	2.58	4.46	2.52	4.39
STEF-Net	2.31	4.05	**2.27**	**3.89**



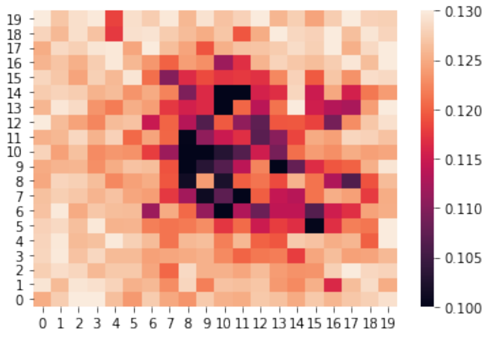
(a) The second last time step in a sunny day



(b) The last time step in a sunny day



(c) The second last time step in a rainy day



(d) The last time step in a rainy day

Figure 3.5 The attention probabilities of the historical time steps that influence the current passenger demands under different weather conditions.

Qualitative Results This section illustrates how the passenger demands are influenced by the historical data. Heatmaps are used to plot the attention weights under different weather conditions (sunny or rainy) at 8pm on two different days in Figure 3.5. In each day, the attention weights of the last time step and the second to the last time step were plotted, which represent the intermediate history and next intermediate history. Note that there are 20×20 grids, and thus in the heatmaps, every small square represent one geographic grid. Center grids represent the downtown area of the city and border grids represent suburban areas. Comparing Figure 6.3(a) with Figure 6.3(b), and comparing Figure 6.3(g) with Figure 6.3(g), it is shown that the current passenger demands in the city center are more influenced by the second to the last time step than by the last one while those in the border area are more influenced by the last time step regardless of the weather. Comparing figures in the sunny day and rainy day in Figure 3.5, it is shown that the weather greatly influences the passenger demands. When the outside is rainy, the passenger demands are less influenced by the last time step in the center of the city (downtown). The rainy weather may incur passengers in the downtown area to change their travel plan by pre-scheduling their activities, e.g., reducing the outdoor activities, while the users in the suburbs are not influenced much.

Ablation Studies: Comparisons with Variants of STEF-Net In the evaluation, STEF-Net with its variants are also compared to explore how different components influence the prediction performance. The following variants are explored,

- **ConvLSTM $\rightarrow$ CNN&LSTM :** 3 convolutional layers with the 64 windows of size 3×3 and 3 LSTM layers with 64 units in each layer are stacked to replace ConvLSTMs for the demand information processing.
- **ConvLSTM $\rightarrow$ LSTM:** 3 LSTM layers with 64 units in each layer are stacked to replace ConvLSTMs for the demand information processing.
- **Fuzzy $\rightarrow$ LSTM:** 3 LSTM layers with 64 units in each layer are stacked to replace the fuzzy neural network for the external information processing.
- **No attention layers:** no attention layers are used after the outputs from the ConvLSTM and the fuzzy neural network are fused.
- **No external information:** no external information is used in the model. Features from the ConvLSTM are directly fed to the attention layer.
- **Weighted addition:** weighted addition is used to replace the convolutional operation in data fusion.

The results are shown in Table 3.2. It is shown that the proposed model, STEF-Net, has the best performance among all variants. Specifically, the proposed model can achieve the smallest RMSE and MAE compared to its variants. Comparing to these variants, it is shown that the ConvLSTM can effectively capture the spatio-temporal information, which performs better than both CNN&LSTM and LSTM, and the fuzzy neural network can outperform the LSTM in processing the external

Table 3.2 Results in Ablation Studies

Model name	MAE	RMSE
ConvLSTM $\rightarrow$ CNN&LSTM	2.32	4.01
ConvLSTM $\rightarrow$ LSTM	2.36	4.09
Fuzzy $\rightarrow$ LSTM	2.58	4.51
No attention layers	2.59	4.35
No external data	2.40	4.24
Weighted addition	2.38	4.09
STEF-Net	**2.27**	**3.89**

information. In addition, it is shown that attention layers can further capture the temporal relevance and improve the performance. Among all the components, fuzzy neural network can significantly improve the performance comparing to the LSTM. The feature fusion method can further capture the temporal feature interaction and the weighted addition performs worse than the fusion with convolution. Comparing the results between the proposed model with the model without external information, it is shown that the external information is important in predicting the passenger demands, which matches the intuition of fusing it into the proposed model.

3.5 Chapter Summary

In this chapter, a deep Spatio-TEmporal Fuzzy neural Network (STEF-Net) is proposed to accurately predict passenger demands in the near future. The proposed model can effectively capture complex input dependencies, including spatial, temporal and external factors, which may influence future passenger demands. In the proposed approach, deep learning is combined with a fuzzy neural network to model spatio-temporal and external information, respectively. A new feature fusion method with convolution followed by an attention layer is employed to fuse two neural networks into one and keep temporal relations for further temporal relevance modeling. Extensive experiments on real-world dataset show that, the proposed model outperforms the state-of-the-art approaches with over 10% improvement in RMSE.

CHAPTER 4

A DEEP CONVOLUTIONAL NEURAL NETWORK FOR TRANSPORTATION MODE DETECTION

4.1 Introduction

The increasing sensing and computing capabilities of smartphones offer a promising new approach to monitoring human activities [58, 4], including means to detect travelers' transportation modes, which is particularly important for developing many transportational applications. For example, (1) knowledge of people's transportation modes is useful for improving urban transportation planning [39, 46]. The new method would transform the way how transportation demand information is gathered for supplementing the traditional information acquisition practice based on telephone interviews and questionnaires, which is expensive and time consuming to conduct [110]. (2) The knowledge can also improve the performance of location and positioning systems. With the awareness of transportation modes, localizing systems can more precisely narrow down users' location [39]. (3) The information facilitates targeted and customized advertisements and services [110]. (4) The acquired information can also help improve smartphone users' physical habits for environment protection purpose. For example, the CO_2 footprint as well as the calories burned by individuals can be better monitored with the information [75, 46]. In this way, the data can help users build green transportation habits to protect the environment.

Most previous studies use data from the Global Positioning System (GPS) to detect people's transportation modes [110, 128, 75]. However, GPS-based methods suffer from the following drawbacks [39, 46]: (1) GPS signals are not available everywhere. GPS requires an unobstructed view to satellites, limiting its applicability in metropolitan areas with highrises or in shielded areas; (2) a GPS sensor consumes a significant amount of energy and may rapidly deplete the battery of a mobile device. To address these issues, some existing work uses alternative sensors to detect transportation modes. For example, Jahangiri *et al.* [46] propose leveraging an accelerometer coupled with a gyroscope and a rotation vector sensor to detect five transportation modes. Fang *et al.* [23] use a deep neural network to classify five transportation modes based on the data from the accelerometer, magnetometer and gyroscope. Hemminki *et al.* [39] propose using an accelerometer sensor to detect six transportation modes. However, only a limited number of transportation modes are detected in the above work, such as walk, bicycle, run and car and the detection accuracy of the above work, less than 90%, still needs improvement when only using the accelerometer sensor. In contrast, the proposed approach detects all common transportation modes, including stationary, walk, bicycle, car, bus, subway and train, using data only acquired from accelerometer sensors in smartphones. The key design objectives of the new work include low energy consumption, applicability in all situations and detection accuracy.

To obtain these objectives, a system utilizing smartphone accelerometers is proposed to detect users' transportation modes efficiently. In the system, the acceleration data are collected from accelerometers in smartphones. The data are processed by removing gravity and smoothing. To minimize the influence from different axes and rotation, the magnitude instead of the value in every axis is used to develop the new models. In this way, the detection accuracy will not be affected by the position of a smartphone since a user can hold her phone in any orientation. The data are divided into small windows. Data in every window are fed into a proposed CNN to detect the window's corresponding transportation mode. Traditional classification models in machine learning are trained with features from both time and frequency domains as benchmarks in this work. Simulation results show that the proposed system can achieve a higher detection accuracy than all peer methods. To summarize, this work makes the following contributions. (1) In this chapter, a transportation mode detection system is proposed by applying a deep learning framework onto data acquired by smartphones. (2) Different from existing systems, the proposed transportation mode detection system is robust to the position and orientation of smartphones since the magnitude instead of the vector in three axes is considered. (3) The proposed system uses only accelerometer sensors, which are available in all modern smartphones and it is more energy-efficient compared to systems using multiple types of sensors [33].

The remainder of this chapter is organized as follows. Section 4.2 presents new system model. Section 4.3 discusses the details of the deep learning employed system. Section 4.4 introduces the models using traditional machine learning methods as a benchmark. Section 4.5 shows the experiments. Section 4.6 compares the system with existing work. This chapter concludes in Section 4.7.

4.2 System Model

In this chapter, the main goal is to detect people's transportation modes using a smartphone. To achieve this goal, a system built upon deep learning techniques using only data acquired through smartphone accelerometers, like Google Nexus 5X and Nexus 6, is proposed to balance the energy consumption and accuracy. The energy consumption in this work mainly comes from the data-collecting sensors, i.e. accelerometers. The accelerometer is a low energy-consumption sensor, which consumes 10 times less than other motion sensors [33]. The high detection accuracy mainly relies on the proposed model using the state-of-the-art deep learning techniques. Deep learning methods are rapidly developing in computer vision and other machine learning fields in recent years [60]. These approaches can represent highly complex functions by stacking multiple layers in a deep artificial neural network. Every layer in the network is composed of a simple but non-linear module,

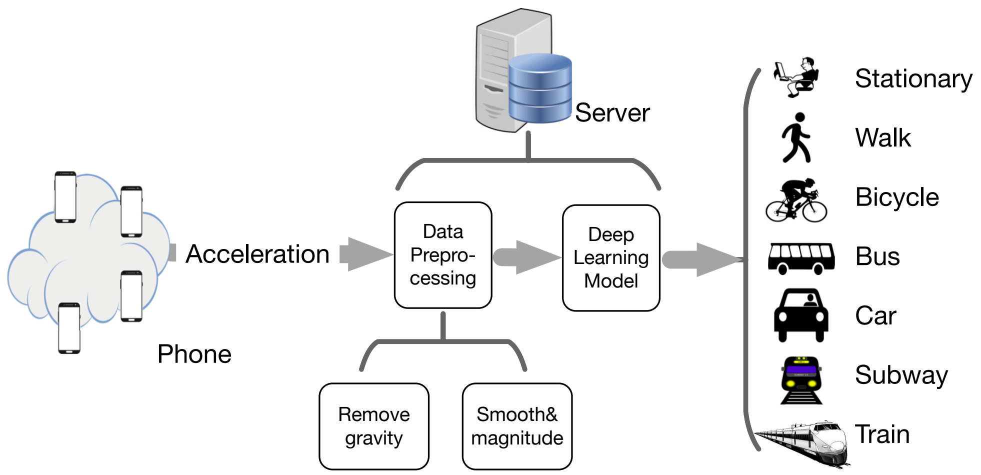


Figure 4.1 Model architecture in the proposed transportation mode detection system.

which receives its lower-layers' information. By utilizing the deep learning method, the accelerometer's data can be fully exploited to detect transportation modes.

In this chapter, a system model is designed including smartphones and a server to detect transportation modes. The architecture is shown in Figure 4.1. In the system, input data are gathered from different mobile devices and are uploaded to a server. The server pre-processes the data and trains a deep learning model accordingly to infer the corresponding transportation modes. There are seven transportation modes supported in the proposed system, including stationary, walk, bicycle, bus, car, subway and train.

In the proposed work, an Android application is developed. Smartphones, installed with the application, are carried by four users all the time when they travel. Every user is also asked to manually input the current transportation mode as the ground truth whenever the mode changes. In every transportation mode, two-hour data are totally recorded. The details of the data will be presented in Section 4.5. The acceleration data are automatically sampled by the application and uploaded to the server along with the corresponding ground truth data regarding transportation modes. The server gathers all data and removes noisy information in the data, including gravity and unrelated fluctuation. On the server end, a

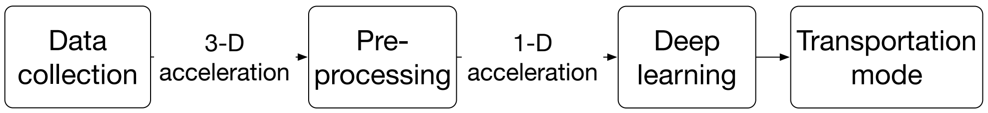


Figure 4.2 Flow chart of the proposed transportation mode detection system.

deep learning model is trained using all labeled acceleration data gathered from users and the model is applied for the online detection. The deep learning model is a convolutional neural network in the system implementation, which focuses on detecting the aforementioned seven most popular transportation modes. The proposed method can also be potentially applied to detect other activities, such as climbing and running.

4.3 Algorithm

The objective of this work is to design a system to efficiently detect transportation modes in real time through smartphones. There are two tasks in implementing such a system: the first one is to determine what data should be collected and how to collect the data efficiently from smartphones; the second one is to process the data in an effective way to detect the transportation modes. For the first task, an Android application is developed to collect the accelerometer data. The application is implemented in a way to balance the energy consumption and the movement information collected by its sensor. For the second task, a series of processes are proposed to clean the data, followed by a convolutional neural network to detect people's transportation modes. Specifically, the proposed system is presented in the following three steps, as shown in Figure 4.2, including (1) Data collection, which is performed by the Android application, (2) Data preprocessing: the data are processed to remove the impact of the gravity, to be smoothed, and to be transferred into one-dimension segments. (3) Deep learning model: a convolutional neural network on one-dimension data is built to determine people's transportation modes.

4.3.1 Data Collection

The system is built utilizing the accelerometer's data to detect transportation modes. The design choice is made due to accelerometer's energy-efficiency, consuming 10 times less power than other motion sensors [33], like gyroscope. To collect accelerometer data, an Android application is developed. The phone was carried under different transportation modes and the collected data were manually labeled with the corresponding transportation mode by the corresponding travellers. When collecting the data, the phone was placed as usual and was not required to stay in a specific position or orientation. In previous studies, the acceleration sampling frequency ranges from 25Hz [75] to 100Hz [7, 46]. In this chapter, a middle sampling frequency, 50Hz, is chosen, which is the same as the one adopted in the prior study [87]. This frequency can balance the information precision and the energy consumption.

The original data acquired from an accelerometer are organized as three dimensional vectors, where each vector component corresponds to the value in one axis in the mobile phone's coordinate system. The coordinate system is shown in Figure 4.3 [34]. One piece of sample data is a vector with one value per axis in the coordinate system in the unit of m/s^2.

4.3.2 Preprocessing

The collected acceleration data contain a gravity component, which is pervasively sampled as it is generated by the earth and everyone on the earth is affected. Thus, the first step of preprocessing is to remove the gravity component so that the remaining part only carries characteristics of different transportation modes. The data are then smoothed to remove large fluctuation, which may be caused by sudden movements. For example, a walking person may suddenly stop to check his beeping phone and

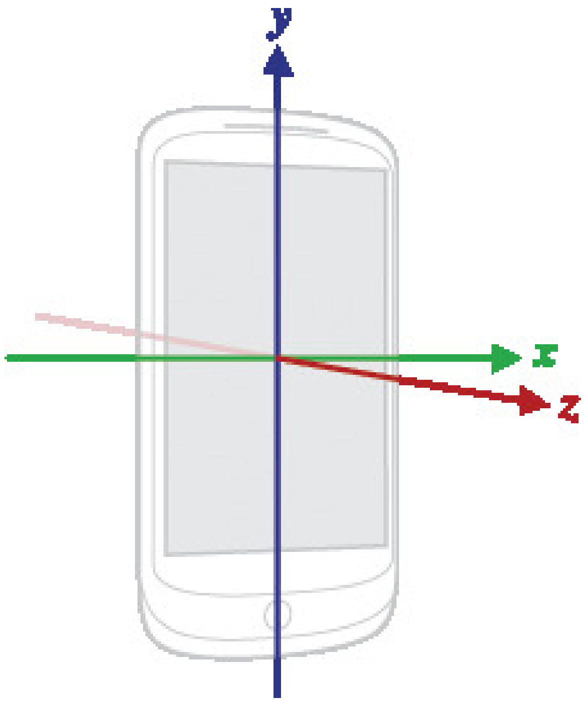


Figure 4.3 Coordinate system in a mobile phone.

then continue walking. Finally, the acceleration magnitude is obtained to build the classification model.

Removing gravity Removing gravity is not straightforward because the gravity part is reflected in all three axes. When a phone is put on a stationary desk with the screen up, the acceleration value should be $(0, 0, -g)$, where g is the gravity constant. However, a phone can be placed in any direction, so even when the phone is stationary, the first two dimension of the data may not be zero and the last one may not be $-g$. Generally, all the three axes will contain a component of the gravity except in the above case. Removing the gravity results in a new movement record, which is used for actual transportation mode detection. The acceleration data obtained after removing the gravity part are called linear acceleration data.

Since the gravity is much more stable than the acceleration generated by movement during a relative long period of time [48, 33], a low-pass filter is applied to remove gravity [49]. Let $\mathbf{A}_k$ denote the vector collected from the accelerometer and $\mathbf{G}_k$ denote the gravity vector at the k^{th} time point in the mobile phone's coordinate system. Let $\mathbf{L}_k$ denote the linear acceleration data after removing the gravity component. The gravity is then estimated by the following equation[48, 33],

$$\mathbf{G}_k = \alpha \cdot \mathbf{G}_{k-1} + (1-\alpha)\mathbf{A}_k, \tag{4.1}$$

where α is the exponential decay weight between old estimated gravity and the new one. An empirical value, 0.8, is taken in the system [34]. $\mathbf{L}_k$ is calculated as follows,

$$\mathbf{L}_k = \mathbf{A}_k - \mathbf{G}_k. \tag{4.2}$$

The linear acceleration data are the collected acceleration data minus the estimated gravity in three axes.

Smoothing Data smoothing is necessary since a phone's movement is not always consistent with its user's movement and the inconsistent part needs to be removed as cleanly as possible. There are two causes of the inconsistency. One is the sudden movement of a phone irrelevant to its user's movement. For example, a user may suddenly pick up the phone while driving or in other transportation modes. In a sudden movement like this, the acceleration changes abruptly. More importantly, the data do not reflect its user's movement, and thus affect detecting the true transportation modes. Therefore, the data are smoothed to reduce the influence by similar sudden movements.

The data are smoothed by the central moving average algorithm, which is a special Savitzky-Golay filter[99]. It is calculated by averaging an odd number of nearest neighbors, m, in the time series. m is a predefined constant value, which is set to 5 in this dissertation. For the original acceleration at the k^{th} time point, $\mathbf{L}_k$, let $\hat{\mathbf{L}}_k$ denote the corresponding estimated linear acceleration value after smoothing.

K is the total number of vectors in the data. $\hat{\mathbf{L}}_k$ is estimated as follows,

$$\hat{\mathbf{L}}_k = \begin{cases} \frac{\sum_{i=1}^{2k-1} \mathbf{L}_i}{2k-1} & k \in [1, \lfloor \frac{m}{2} \rfloor], \\ \frac{\sum_{i=k-\lfloor m/2 \rfloor}^{k+\lfloor m/2 \rfloor} \mathbf{L}_i}{m} & k \in (\lfloor \frac{m}{2} \rfloor, K - \lfloor \frac{m}{2} \rfloor), \\ \frac{\sum_{i=2k-K}^{K} \mathbf{L}_i}{2(K-k)+1} & k \in [K - \lfloor \frac{m}{2} \rfloor, K]. \end{cases} \tag{4.3}$$

If the number of neighbors in one side is less than K, then the average algorithm takes the same maximum possible number of neighbors from both sides.

Magnitude Inconsistency aforementioned may also come from the fact that a phone's orientation is likely to be different from that of its user. The phone may be put in any position and orientation subject to changes at times. Thus, classifiers are trained using the magnitude of the acceleration data. Even though there may be some information loss, the magnitude is more robust to a phone's changing and unpredictable orientation [96]. For $\hat{\mathbf{L}}_k$=$(\hat{L}_{k_x}, \hat{L}_{k_y}, \hat{L}_{k_z})$, $|\hat{\mathbf{L}}_k|$ denotes its magnitude, which is calculated as follows,

$$|\hat{\mathbf{L}}_k| = \sqrt{\sum_{i=x,y,z} \hat{L}_{k_i}^2}. \tag{4.4}$$

To conduct real-time detection, the time series data are divided into small windows. The system detects every separate window's transportation mode. Specifically, the data are grouped into a window of samples with a window size of S seconds and a sliding size of s seconds. By default, S is set to 10.24 seconds (512 samples) and s is set to 1.28 seconds (64 samples). The reason for choosing the two values is that the frequency transformation in traditional machine learning methods requires the number of samples to be the exponentiation of 2. With the selected values, a transportation mode is detected almost every second based on the 10-second historical information. The values of S and s are changed in the experiments to examine their impact on performance.

4.3.3 Deep Learning Model

The deep learning model adopted in this system is a CNN. CNNs, as one special type of deep learning models, are commonly used to recognize objects in image processing. In this chapter, a CNN is built on the one-dimension acceleration data to determine the transportation mode in every time window. One of the key elements in the CNN's architecture is the various types of operation in constructing the layers. Therefore the operations, convolution, max-pooling and full-connection of the CNN used in the proposed approach are first introduced followed by its architecture. The next subsections introduce the nonlinear function in every layer, the loss in evaluating the performance of a classification model and the optimization method used to minimize the loss function. The last element is the normalization on the data to improve performance.

Convolution, max-pooling, and full-connection The convolutional operation is a basic module between two neural layers in a CNN. Units in a convolutional layer are organized by feature maps, which aggregates local patches in the feature map of the previous layer through a set of weights. The set of weights is called a filter bank. All units in a feature map share the same weights in a filter bank. In a feature map, the filter bank shifts a fixed length of step defined by the stride, to generate one unit. Different feature maps have different filter banks. The convolutional operation makes the presence of a pattern more important than the pattern's position.

Figure 4.4 shows an example on how the calculation takes place in a convolutional layer. In the figure, the feature map from the previous layer is denoted by X, which is assumed to be a one-dimensional vector. The collection of filter banks is denoted by W, where the index j indicates the j^{th} filter bank. Every filter bank generates one feature map of the next layer by shifting a window on the previous layer's feature map. The shifting stride is pre-defined, which splits X into multiple

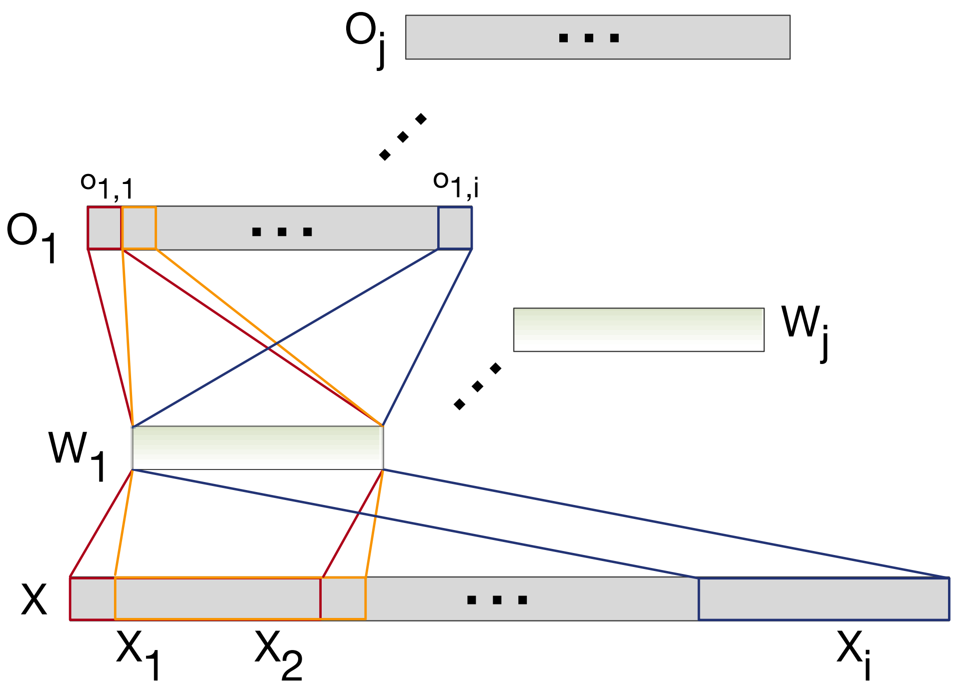


Figure 4.4 Illustration of the calculation in a convolutional layer.

segments, X_1, X_2, ..., X_i, with the same size as the filter bank. The feature map of the next layer from the j^{th} filter bank is denoted by O_j, which contains the sum of multiplication between all segments in X and the filter bank. The value o_i of the i^{th} unit in the feature map is the sum of values multiplied by the weights in the filter,

$$o_{j,i} = \sum_k w_{j,k} x_{i,k} + b_j. \tag{4.5}$$

In the equation, $x_{i,k}$ and $w_{j,k}$ are elements in X_i and W_j, respectively. b_j is the bias for this filter bank. The value in a unit is the sum of all weighted values in a corresponding segment. The generated feature maps are used to generate the next layer's feature map. Thus, in a convolutional layer, three things need to be defined, the number of filter banks, the shape of a filter bank, and the shape of the stride.

The max pooling layer is used to pick out the salient values from a local patch of units. It can reduce the dimensionality by removing less important information in the feature map. Specifically, the max pooling operation selects the max value in a local patch of units and then the local patch shifts a step with the stride size. Thus, it is required to define the size of local patches and the stride's size in a max pooling layer.

The fully-connected layer is to connect all units in the previous layer to all units in the next layer. In a fully-connected layer, the number of units in the next layer is required to be set as a hyperparameter.

Overall architecture The CNN is designed for one-dimensional data with the architecture shown in Fig. 4.5. The architecture takes the one-dimensional

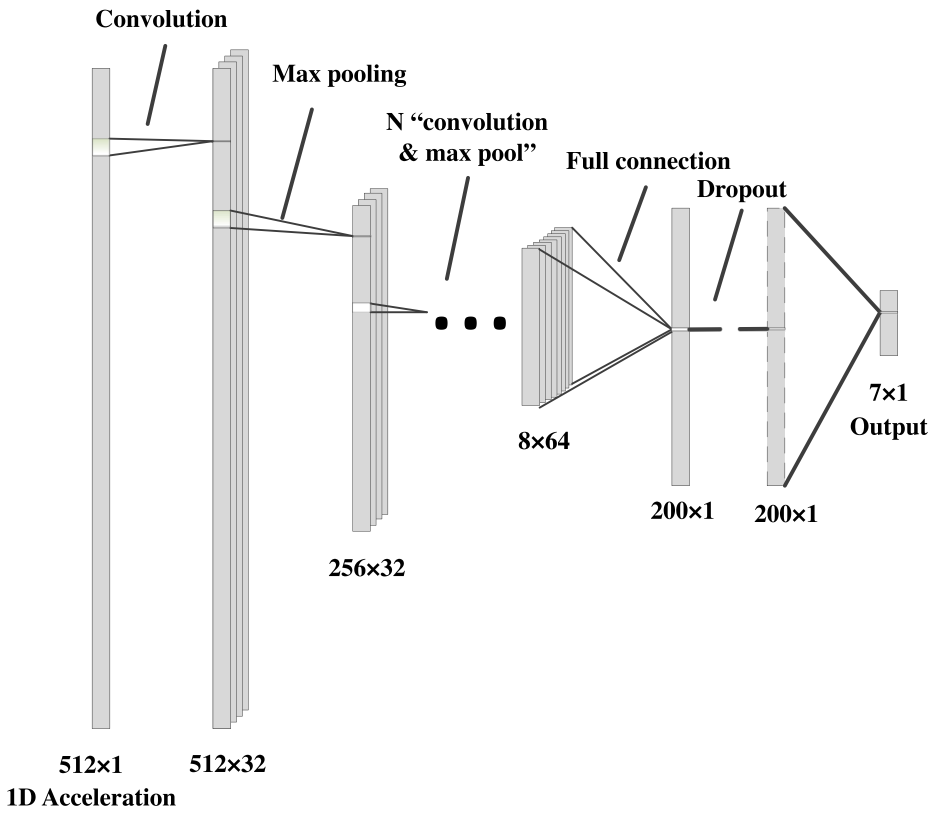


Figure 4.5 The proposed CNN architecture in this transportation mode detection system.

acceleration data in a window as input and outputs the probability of being every transportation mode for the window. The architecture consists of a succession

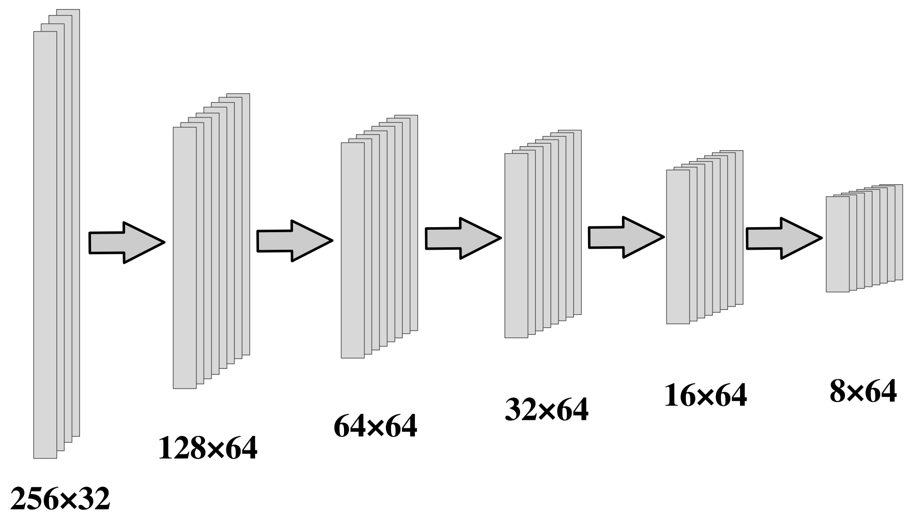


Figure 4.6 The hidden N layers in Figure 4.5.

of convolutional, max pooling and fully-connected layers, whose specification is as follows.

The input feature is a 512 $\times$ 1 vector regarding magnitude of the acceleration data. The first dimension is the temporal space in the window and the second dimension is the size of values at every temporal point. The first convolutional layer has 32 filter banks with the 15$\times$1 shape and the filter bank moves with a stride of 1 feature at a time. The following max pooling layer is set to a 4-feature window and a stride of two features. The convolutional layer and max pooling layer are repeated N times. In the proposed network, N is 6. All the max pooling layers are configured using the same parameters. The second and third convolutional layers have 64 filter banks on a 10-feature window with a stride size of 1 feature. The other four convolutional layers filter the data with 64 filter banks on a 5-feature window with a stride of 1 feature. Conducting the convolution and max pooling processes six times, shown in Figure 4.6, the data become 8 $\times$ 64. The data are flattened and fed into a fully-connected layer to become 200 $\times$ 1. A dropout layer is not employed in the proposed system, but it is added after the fully-connected layer only for comparison, which is used to evaluate whether the dropout layer can help improve the performance. Finally, the data are transferred into a 7×1 output vector with full connection. The value in the output is the probability for one transportation mode.

Nonlinear function In a neural network, the common ways to model a neuron's output f as a function of its input x are with tanh, sigmoid or Rectified Linear Unit (ReLU). ReLU outperforms the other two with its simple form ($f(x) = max(0, x)$) in the fast convergence of stochastic gradient descent [57], but it may generate 'dead' neurons, which are never activated. In the proposed model, a leaky ReLU[38] is employed, which is defined as follows:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \beta x, & \text{if } x \leq 0 \end{cases}. \tag{4.6}$$

β is a small constant to avoid zero gradient in the negative side. In the experiments, it is shown that the leaky ReLU can converge faster than the traditional ReLU. The nonlinear activation function is attached to every convolutional layer and the fully-connected layer.

Loss function A loss function is used to evaluate how labels derived by the current classification model deviate from the corresponding true labels. The loss function is usually composed of two parts, the variance between the estimated labels and true labels, and the regularization. The loss L is shown in the following equation,

$$L = V + \lambda \cdot R, \tag{4.7}$$

where V denotes the variance and R is the regularization value. λ is the weight decay determining how much the regularization affects the final loss.

The variance is expressed by the softmax cross entropy, as follows:

$$V = -\sum_{id} \sum_{c} t_{id,c} \cdot log(y_{id,c}) \tag{4.8}$$

where id denotes the index of a sample in the mini-batch and $y_{id,c}$ is the value of sample id at class c in the output layer. If the sample belongs to class c, the value of $t_{id,c}$ is 1; otherwise, it is 0. If an estimated label is the same as the true one, the contribution to the final V from the sample is 0; if the estimated label and the true one are different, the contribution becomes very large.

The regularization is to avoid overfitting during model training [125]. The proposed neural network contains over 100,000 parameters. To overcome overfitting, there are two possible ways, regularization [125] and dropout [57]. Regularization is to decrease the scale of a neural network by making weights as close to zero as possible. Dropout is to randomly make some weights as zero to increase the network's robustness. In this network, the L2 regularization is employed in the loss function. In L2 regularization, the value of R is the squared sum of all weights in the CNN.

Optimization Gradient descent is one of the most popular algorithms to optimize the loss function in neural networks. Among all variants of gradient descent algorithms, Adaptive Moment Estimation (Adam) [52] is favorably reviewed due to its capability for attaining satisfactory overall performance with a fast convergence and adaptive learning rate [97]. The Adam optimization method adaptively updates the learning rate considering both first-order and second-order moments using the stochastic gradient descent procedure. Specifically, let $\boldsymbol{\theta}$ denote the parameters in the CNN and $L(\boldsymbol{\theta})$ denote the loss function. Adam first calculates the gradients of the parameters,

$$\mathbf{g} = \nabla_{\theta} L(\boldsymbol{\theta}). \tag{4.9}$$

It then respectively updates the first-order and second-order biased moments, $\mathbf{s}$ and $\mathbf{r}$, by the exponential moving average,

$$\begin{aligned}\mathbf{s} &= \rho_s \mathbf{s} + (1 - \rho_s)\mathbf{g}, \\ \mathbf{r} &= \rho_r \mathbf{r} + (1 - \rho_r)\mathbf{g},\end{aligned} \tag{4.10}$$

where ρ_s and ρ_r are the exponential decay rates for the first-order and second-order moments, respectively. The first-order and second-order biased moments are corrected using the time step t through the following equations,

$$\begin{aligned}\hat{\mathbf{s}} &= \frac{\mathbf{s}}{1 - \rho_s^t}, \\ \hat{\mathbf{r}} &= \frac{\mathbf{r}}{1 - \rho_r^t}.\end{aligned} \tag{4.11}$$

Finally the parameters are updated as follows,

$$\begin{aligned}\boldsymbol{\theta} =& \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \\ =& \boldsymbol{\theta} + (-\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}}),\end{aligned} \tag{4.12}$$

where ϵ is the initial learning rate and δ is a small positive constant to attain numerical stability.

Normalization The classification results can be improved through normalization. Assume the dataset is not too large to store in the memory. The whole training dataset is used to normalize the whole dataset. The mean μ and variance σ of the training dataset are derived first. The normalized data x' from the original x is calculated through [66],

$$x' = \frac{x - \mu}{\sigma}. \tag{4.13}$$

The normalization process is done before the data is used to train the CNN.

4.4 Traditional Machine Learning Methods

The traditional machine learning methods, including Bayes Classifiers, C4.5 Decision Tree, K-Nearest Neighbors, Random Forest, Adaptive Boosting, Neural Network and Support Vector Machine, are taken as the benchmarks for comparison [68]. In traditional machine learning methods, common features adopted in previous work [68, 39, 46] are selected from the training data to develop the models. The trained models are subsequently used to classify new data. In the following, the features and models in the traditional machine learning methods are explained briefly.

4.4.1 Features

In every window, the features are selected in both time and frequency domains, which are listed in Table 4.1, which are usually widely employed as features in peer work. When obtaining the features in the frequency domain, a window function is applied to the data.

Table 4.1 Features in Traditional Methods

Domain	*Features*
Time	mean, standard deviation, median, root mean square, min, max, range, kurtosis, and skewness
Frequency	log-scale power spectral density at 1-10Hz, and power spectral centroid

The data in a window are scaled by a window function before they are transferred into the frequency domain. Window functions are shown effective in reducing the lobeside effect [89]. A commonly-used window function, Hamming, is employed in the system with the following formula,

$$w_n = \beta - \gamma cos(\frac{2\pi n}{N}), \tag{4.14}$$

where w_n is the weight value in the window, β and γ are two constants and n is an integer from 1 to N. N is the number of samples in a window. By default, β is set to 0.54 and γ is set to 0.46, which are used to balance the information loss and reduce lobeside effect.

Till now, features in the traditional methods are obtained, but different features have different scales [66]. The normalization is applied on every feature in the data, which is the same process as shown in Section 4.3.3.

4.4.2 Traditional Classification Models

Several common traditional machine learning models are adopted to classify the data. Brief introduction to every model is given in the following paragraphs. Every model's parameters are finely tuned by 10-folder cross validation.

Bayes classifiers Bayes classifiers are statistical classifiers [8]. They predict an instance's class by calculating the probability that the instance belongs to each particular class via the similarity of feature values. The simplest one in Bayes classifiers is the Naive Bayes (NB), which assumes that all features are uncorrelated [8]. It calculates the probability of one instance X in one specific class C based on the Bayes' Theorem,

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}. \tag{4.15}$$

In the equation, $P(X)$ and $P(C)$ are known priors. In the assumption, the features are mutually independent, so $P(X|C)$ is the product of the probability of all features in one specific class. The instance is classified into the class with the maximum probability. A more complicated Bayesian classifier is Bayesian Networks (BNs) [93]. A Bayesian Network is a directed acyclic graph, which is to build a graph by the estimated correlation between features.

Decision tree The Decision Tree (DT) is to build a classification tree. The tree structure is presented as a leaf indicating a class and each node specifying some test on a single feature value with the branch and subtree for the possible outcome. To classify a case, it starts at the root and moves through the tree until a leaf is encountered [8]. The tree is split by information gain [45] and Gini index [62]. Only one feature is used to split the tree at every node. One of the most popular decision trees is C4.5 [8], which is used in this chapter for comparison.

K-nearest neighbors The K-Nearest Neighbors (K-NN) classifier is to classify an instance based on the closest k nearest neighbors in the training data [19]. It is called the lazy-learning algorithm, since its computation/overhead is much lighter during learning than testing time. The closeness between instances is defined as the distance, which is usually Euclidean or Manhattan distance. Among the k nearest neighbors, the instance is classified into the most common class, which class that the most neighbors belong to [93, 8]. The performance of K-NN may be affected by the choice of k.

Random forest The Random Forest (RF) method uses ensembles of unpruned decision trees [41, 114]. A common decision tree is usually pruned to avoid overfitting, but in random forest, the decision trees are unpruned. It draws bootstrap samples from the training data. It randomly chooses a subset of features in the samples to build a complete decision tree according to the samples. Multiple decision trees are built with different samples in the same way. The classification result is predicted by aggregating the classification results from all trees.

Adaptive boosting The Adaptive Boosting (AB) is to train multiple weak classifiers from subsets with the same size. The final classification result is obtained by aggregating the classifiers with weights. The weights are adaptive. If one instance outside the subset in the training data is classified correctly, then the weight is reduced; otherwise it increases [25].

Neural network The Neural Network (NN) is a set of connected input/output units in which a weight is associated with each connection. A Neural network is usually composed of an input layer, one or more hidden layers and an output layer. The data is received in the input layer and processed in the hidden layers. The output layer produces the classification results [8]. The network is built by updating weights via backpropagation.

Support vector machine The Support Vector Machine (SVM) builds a hyperplane to separate two data classes by maximizing the margin between two classes and the hyperplane based on a cost function. The SVM is at first outlined for linearly separable cases. A kernel function is defined to transfer nonlinear features into linear ones with high dimensions [18, 8]. The SVM classifies multiple classes via training several SVMs on every two classes, or every one class and another class including all the data in the other classes.

4.5 Experiments

Key experimental results are presented in this section with performance attained by the method. The implementation details are first presented, and then the data are analyzed. Finally, the detection performance of the system and the comparison with other methods are presented in detail.

4.5.1 Implementation

In the system, an Android application is developed and installed on a Google Nexus 5X and a Google Nexus 6, which are shared by four different users at different time. During their transportation, users can freely hold the phone in any orientation to their preference. The orientation and placement of mobile phones are not restricted in the system. The Android application records the accelerometer data at 50Hz and the current transportation mode is manually input by the user. Matlab is used to preprocess the data and the CNN is built using Tensorflow [1] on one NVIDIA GTX 1050 Ti 4GB GPU. The traditional models are built and tested in Weka [129].

The acceleration data are collected in the following seven transportation modes, stationary, walk, bicycle, bus, car, subway and train. The data in every state are collected for about 2 hours. Specifically, the stationary state is sampled on campus; the walk state is acquired on campus and in nearby parks; the bus state is taken when users go to school and back home; the car state is recorded in local roads and on highways; the subway is taken in New York City; and the train is acquired from New Jersey to Washington, D.C..

By default, a 512-sample window moves 64 samples every time to generate a new window data (512 and 64 are chosen for generating frequency values in traditional machine learning models, which are nor required in the CNN). It means one output is generated every 1.28 seconds based on about 10-second historical values. In the experiments, the data are primarily split into 80% as training and 20% as testing sets. The parameters in the CNN are shown in Table 4.2.

Table 4.2 Parameters Used in the Convolutional Neural Network

Parameter	*Value*
Minibatch size	100
L2 regularization λ	0.001
Leaky ReLU β	0.01
1st-order moment weight ρ_s	0.9
2nd-order moment weight ρ_r	0.999
Learning rate ϵ	0.0001
Constant δ	10^{-8}

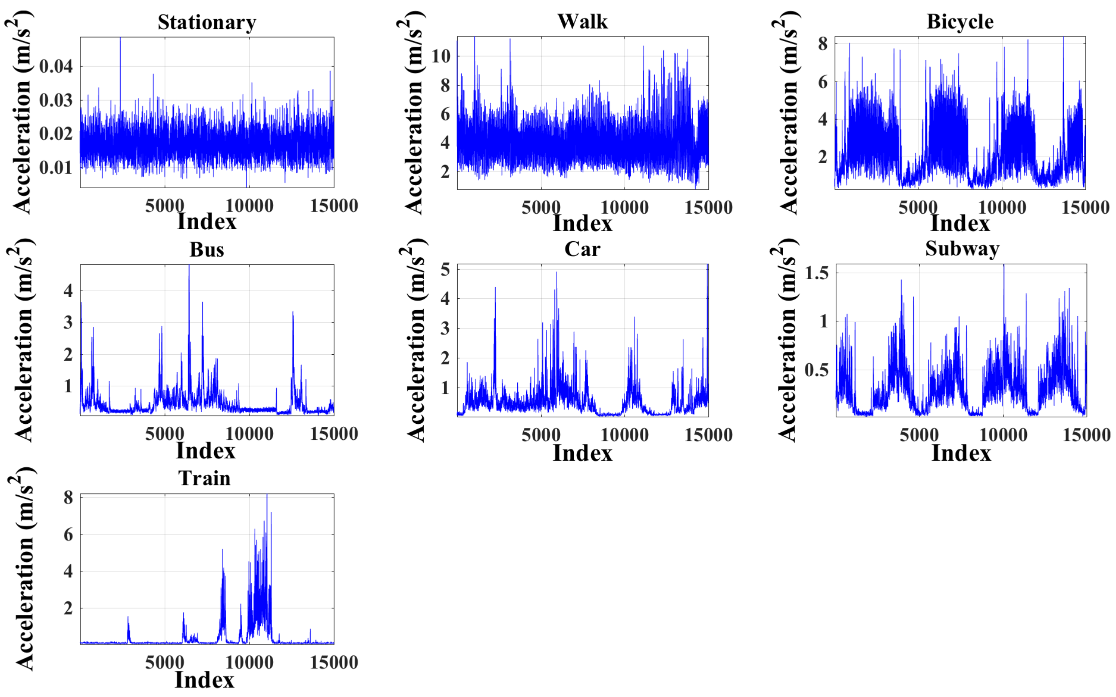


Figure 4.7 The acceleration during a period in seven modes.

4.5.2 Data Analysis

The acceleration data in different transportation modes are analyzed. Figure 4.7 shows a period in all modes after smoothing. In the figure, the x-axis is data index and the y-axis is the acceleration magnitude. From the figure, it is shown that the acceleration data in different modes show different patterns. For example, the acceleration data in bicycle shows obvious periodicity and the data in walk have the largest acceleration values compared with others. The acceleration value in the stationary mode is the smallest. The figure also shows that the data in the bus and car are similar in the shape as both modes keep low acceleration at most time and have a few rapid changes.

In the frequency domain, a time-frequency figure is drawn in Figure 4.8. The data are sampled every 128 values with 64 values overlapping with neighbors. It means that the window is 2.56 seconds and the overlap is 1.28 seconds. In Figure 4.8, the x-axis is the frequency and y-axis is the time. The color shows the magnitude,

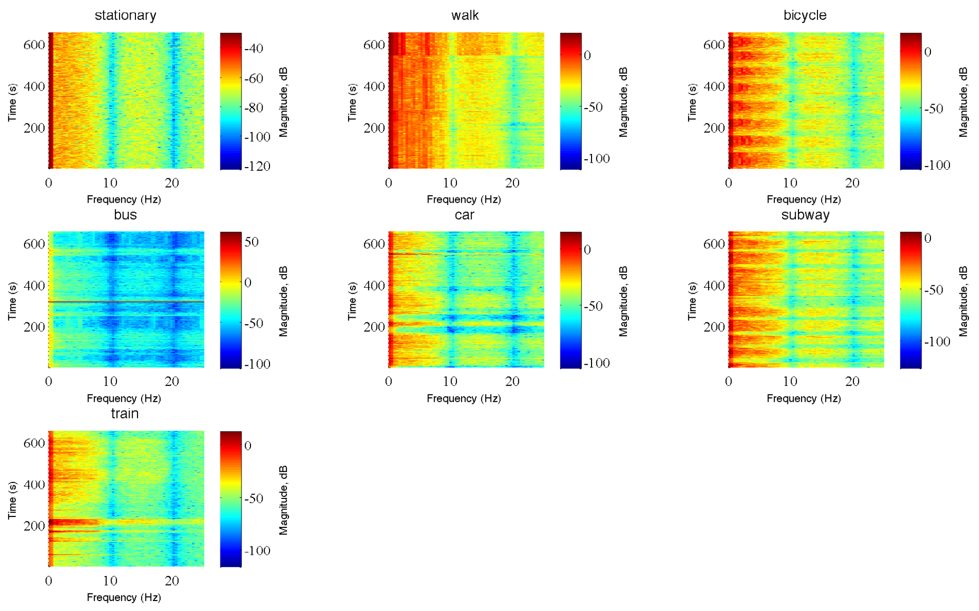


Figure 4.8 Time-frequency figures in different transportation modes.

which is the same as the log-scale power spectral density. From Figure 4.8, it is shown that the most power is gathered in the frequencies less than 10Hz. It is reasonable for us to select the power density in the frequency from 1Hz to 10Hz as features.

4.5.3 Classification Results

The classification results attained by CNNs of different architectures are first compared with the proposed CNN, and then the testing accuracy changes during the training process are graphed. Finally, the detection accuracy attained by traditional machine learning models is presented.

CNNs' results The proposed CNN method's result matrix is shown in Table 4.3. In the table, every row means one transportation mode in the ground truth. Every transportation mode has 600 samples in the testing data. The columns show the predicted transportation mode by the proposed CNN. It is shown from the table that the proposed CNN can achieve the accuracy of 94.48%. The result also shows that it is hard to classify the motorised transportation modes, like car, bus and subway.

Table 4.3 Classification Matrix of the Proposed System

	Stationary	*Walk*	*Bicycle*	*Bus*	*Car*	*Subway*	*Train*	*Accuracy (%)*
Stationary	*598*	*2*	*0*	*0*	*0*	*0*	*0*	*99.7*
Walk	*3*	*558*	*0*	*7*	*4*	*12*	*14*	*93.0*
Bicycle	*0*	*0*	*592*	*2*	*6*	*0*	*0*	*98.7*
Bus	*0*	*6*	*0*	*554*	*28*	*12*	*0*	*92.3*
Car	*0*	*3*	*6*	*30*	*533*	*26*	*2*	*88.9*
Subway	*1*	*19*	*0*	*12*	*33*	*535*	*0*	*89.2*
Train	*0*	*0*	*0*	*0*	*3*	*0*	*597*	*99.5*
Average accuracy								*94.48*

The comparison among multiple CNNs is presented in Table 4.4. The first row shows the results from the employed architecture of CNN in this system. The other rows are the compared CNNs. In the compared CNNs, one part of the proposed CNN is modified to get a new CNN model. All CNNs are trained 1.5 million batch

Table 4.4 The Proposed System's Classification Results

Activation function	*Regularization*	*Dropout*	*Accuracy (%)*
Leaky ReLU	*L2*	*No*	***94.48***
Leaky ReLU	*No*	*No*	*92.63*
Leaky ReLU	*L2*	*0.9*	*91.90*
ReLU	*L2*	*No*	*91.87*

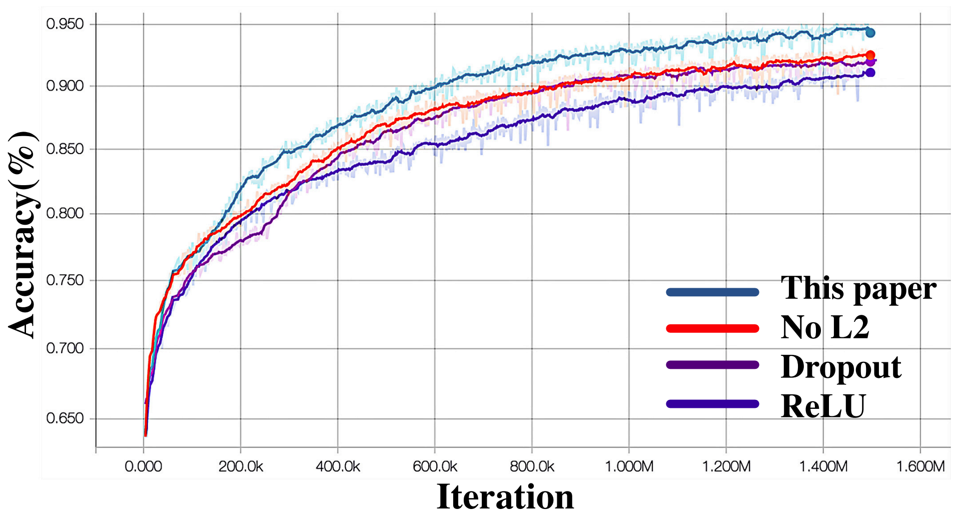


Figure 4.9 Testing accuracy changes during the training period.

iterations with 100 windows in one batch. The results show that the proposed CNN outperforms the other three. Specifically, the CNN achieves an accuracy of 94.48%. The accuracy changes in the four CNNs during the training are shown in Figure 4.9. In the figure, the x-axis shows the iteration index and the y-axis indicates the testing accuracy. The data in the figure are smoothed with a parameter 0.5. Shown in the figure, the proposed CNN outperforms the others and converges faster than the others.

Comparison with traditional methods In this part, The results are compared between the proposed CNN and other traditional machine learning methods under different window sizes. The window size changes from 128 to 512 with 64 distinguishing values between two adjacent windows. It means the time length in a window is from 2.56 to 10.24 seconds. The classification results under different classification models with different window sizes are shown in Table 4.5.

The table shows that the proposed CNN outperforms the other methods under all window sizes. Specifically, among all traditional methods, random forest performs best in accuracy by assembling multiple decision trees and different features. Even so, the proposed CNN outperforms random forest by 94.48% to 90.11% when the

Table 4.5 Classification Results Comparing with Traditional Models under Different Window Sizes

Algorithm	Window size		
	128	256	512
Naive Bayes	*58.34%*	*59.87%*	*60.79%*
Bayes Network	*62.47%*	*65.30%*	*68.77%*
Decision Tree	*67.34%*	*72.96%*	*81.96%*
K Nearest Neighbor	*66.48%*	*69.07%*	*76.03%*
Random Forest	*74.09%*	*79.68%*	*90.11%*
Adaptive Boosting	*65.63%*	*71.65%*	*80.73%*
Neural Network	*69.14%*	*70.43%*	*76.30%*
Supporting Vector	*73.26%*	*75.76%*	*84.80%*
CNN	*75.48%*	*82.42%*	***94.48%***

window size is 512. In addition, considering any particular classification algorithm, the larger the window size is, the higher the accuracy can be reached. It means when the window size is too small, much information cannot be covered in the window. For example, a sudden stop while driving may be classified as stationary when the window size just catches the stop period.

4.6 Comparison

In this section, the performance is compared with other studies using accelerometers in detecting transportation modes.

There are several studies using accelerometers only to detect transportation modes [87, 75, 39, 130]. Hemminki *et al.* [39] collect accelerometer data at the frequency from 60 to 100Hz and divide the data into 1.2-second windows with 50% overlap. They extract 27 features in every window and train an adaptive boosting to classify the data into six modes, stationary, walk, bus, train, metro and tram. They achieve an accuracy of 80.1%. Manzoni *et al.* [75] collect accelerometer data at the frequency of 25Hz and divide it into windows of 10.24 seconds length with 50% time overlap. They also extract features from the FFT coefficients in every window and train a decision tree to classify the data into eight modes, walk, bicycle, bus, car, metro, train, still, and motorcycle. They achieve an accuracy of 82.14%. Yang [130] collects accelerometer data at the frequency of 36Hz and divide the data into 10-second windows with 50% overlap. Features are extracted from time and frequency domains and a decision tree is used to classify six transportation modes, sitting, standing, walk, run, bicycle, and car. The accuracy is 90.6%. Table 4.6 shows a summary of the three studies using only the acceleration data. Compared with the existing studies using acceleration data, the proposed system can provide higher accuracy. Existing studies usually use the traditional machine learning methods to detect transportation methods. In this work, it is shown that CNNs outperform traditional machine learning methods in detecting transportation modes. In addition, among the traditional methods, random forest performs best in the accuracy metric instead of other ones used in the existing work.

Table 4.6 Summary of Past Work Using Acceleration to Detect Transportation Modes

Study	*Classes*	*Data*	*Window*	*Method*	*Accuracy*
Hemminki et al. [39]	*1.stationary 2.walk 3.bus 4.train 5.metro 6.tram*	*Accelerometer 60-100Hz pockets, bags*	*1.2 seconds*	*Adaptive boosting*	*80.1%*
Manzoni et al. [75]	*1.walk 2.bicycle 3.bus 4.car 5.metro 6.train 7.still 8.motorcycle*	*Accelerometer 25Hz*	*10.24 seconds*	*Decision tree*	*82.14%*
Yang [130]	*1.stand 2.sit 3.walk 4.run 5.bicycle 6.car*	*Accelerometer 36Hz*	*10 seconds*	*Decision tree*	*90.6%*
This chapter	*1.stationary 2.walk 3.bicycle 4.bus 5.car 6.subway 7.train*	*Accelerometer 50Hz*	*10.24 seconds*	*Convolutional neural network*	*94.48%*

4.7 Chapter Summary

In this chapter, a robust system on Android smartphones is proposed to accurately detect users' transportation modes by employing the smartphone's accelerometer. This is the first system that utilizes convolutional neural networks to detect transportation modes with the accelerometer only. In this system, the collected data are processed by removing gravity and smoothing. The acceleration magnitude is used to build a convolutional neural network to recognize the corresponding transportation mode. Extensive experiments verify that the proposed system outperforms the CNNs of other architectures and traditional machine learning models. The proposed system can achieve as high as 94.48% in detection accuracy, which outperforms the existing studies.

CHAPTER 5

DEEP REINFORCEMENT LEARNING FOR TRAFFIC LIGHT CONTROL

5.1 Introduction

The intersection management of busy or major roads is primarily done through traffic lights, whose inefficient control causes numerous problems, such as long delay of travelers and huge waste of energy. Even worse, it may also incur vehicular accidents [86, 71]. Existing traffic light control either deploys fixed programs without considering real-time traffic or considering the traffic to a very limited degree [10]. The fixed programs set the traffic signals equal time duration in every cycle, or different time duration based on historical information. Some control programs take inputs from sensors such as underground inductive loop detectors to detect the existence of vehicles in front of traffic lights. However, the inputs are processed in a very coarse way to determine the duration of green/red lights.

In some cases, existing traffic light control systems work, though at a low efficiency. However, in many other cases, such as a football event or a more common high traffic hour scenario, the traffic light control systems become paralyzed. Instead, it is often to witness an experienced policeman directly manages the intersection by waving signals. In high traffic scenarios, a human operator observes the real time traffic condition in the intersecting roads and smartly determines the duration of the allowed passing time for each direction using his/her long-term experience and understanding about the intersection, which is very effective. This observation motivates us to propose a smart intersection traffic light management system which can take real-time traffic condition as input and learn how to manage the intersection just like the human operator. To implement such a system, 'eyes' are needed to watch the real-time road condition and 'a brain' to process it. For the former, recent advances in sensor and networking technology enables taking real-time traffic information as input, such as the number of vehicles, the locations of vehicles, and their waiting time [22]. For the 'brain' part, reinforcement learning, as a type of machine learning techniques, is a promising way to solve the problem. A reinforcement learning system's goal is to make an action agent learn the optimal policy through interacting with the environment to maximize the reward, e.g., the minimum waiting time in this intersection control scenario. It usually contains three components: states of the environment, action space of the agent, and reward from every action [112]. A well-known application of reinforcement learning is AlphaGo [107], followed by AlphaGo Zero [108]. AlphaGo, acting as the action agent in a Go game (environment), first observes the current image of the chessboard (state), and takes the image as the input of a reinforcement learning model to determine where to place the optimal next playing piece 'stone' (action). Its final reward is to win the game or to lose. Thus, the reward may not be obvious during the playing process but becomes clear when the game is over. When applying reinforcement learning to the traffic light control problem, the key point is to define the three components at an intersection and quantify them to be computable.

Some previous work proposes to dynamically control the traffic lights using reinforcement learning. Some define the states by the number of waiting vehicles or the waiting queue length [22, 2]. But real traffic situation cannot be accurately captured by only the number of waiting vehicles or queue length [29]. With the popularization of vehicular networks and sensor networks, more accurate on-road traffic information can be extracted, such as vehicles' speed and waiting time [37]. However, rich information causes the number of states to increase dramatically. When the number of states increases, the complexity in a traditional reinforcement learning system grows exponentially. With the rapid development of deep learning [68], deep neural networks have been employed to deal with the large number of states, which constitutes a deep reinforcement learning model [84]. A few recent studies have proposed to apply deep reinforcement learning in the traffic light control problem [63, 121]. But there are two main limitations in existing studies: (1) the traffic signals are usually split into fixed-time intervals, and the duration of green/red lights can only be a multiple of this fixed-length interval, which is not efficient in many situations; (2) the traffic signals are designed to change in a random sequence, which is not a safe or comfortable way for drivers. In this chapter, the problem on how to control the traffic light signal duration in a cycle is studied based on the extracted information from vehicular networks or sensor networks.

The general idea is to mimic an experienced operator to control the signal duration in every cycle based on the information gathered from vehicular networks. To implement such an idea, the operation of the experienced operator is modeled as an Markov Decision Process (MDP). The MDP is a high-dimension model, which contains the time duration of every phase. The system learns the control strategy based on the MDP by trial and error in a deep reinforcement learning model. To fit a deep reinforcement learning model, the whole intersection is divided into grids to build a matrix, each element of which is the vehicles' information in the corresponding grid collected by vehicular networks or extracted from cameras via image processing. The matrix is defined as the states and the reward is the cumulative waiting time difference between two cycles. In the proposed model, a convolutional neural network is employed to match the states and expected future rewards. Note that, every traffic light's action produced from this model affects the environment. When the traffic flow changes dynamically, the environment becomes unpredictable. To solve this problem, a series of state-of-the-art techniques in the proposed model is employed to improve the performance, including dueling network [127], target network [84], double Q-learning network [122], and prioritized experience replay [100].

The contribution of the chapter includes 1) This work is the first one to combine dueling network, target network, double Q network and prioritized experience replay into one framework to solve the traffic light control problem, which can be easily applied into other problems. 2) The proposed control system decides the phases' time duration in a whole cycle instead of dividing the time into segments. 3) Extensive experiments on a traffic micro-simulator, Simulation of Urban MObility (SUMO) [54], show the effectiveness and high-efficiency of the proposed model.

The reminder of this chapter is organized as follows. The model and problem statement are introduced in Section 5.3. The background on reinforcement learning is introduced in Section 5.2. Section 5.4 details the proposed reinforcement learning model in the traffic light control system. Section 5.5 extends the reinforcement learning model into a deep learning model to handle the complex states in the this system. The model is evaluated in Section 5.6. Finally, the chapter is concluded in Section 5.7.

5.2 Background on Deep Reinforcement Learning

Reinforcement Learning (RL) is a type of algorithms in machine learning. It interacts with the environment to learn better actions to maximize the objective reward function in the long run through trial and error. In reinforcement learning, an agent, the action executor, takes an action and the environment returns a numerical reward based on the action and the current state. A four-tuple $\langle S, A, R, T\rangle$ can be used to define the reinforcement learning model:

- S : the possible state space. s is a specific state ($s \in S$);
- A : the possible action space. a is an action ($a \in A$);
- R : the reward space. $r_{s,a}$ denotes the reward in taking action a at state s;

- T : the transition function space among all states, which represents the probability of the transition from one state to another.

In a deterministic model, T is usually omitted.

A policy is made up of a series of consequent actions. The goal in reinforcement learning is to learn an optimal policy to maximize the cumulative expected rewards starting from the initial state. Generally speaking, the agent at one specific state s takes an action a to reach state s' and gets a reward r, which is denoted by $\langle s, a, r, s' \rangle$. Let t denote the t^{th} step in the policy π. The cumulative reward in the future by taking an action a at state s is defined by $Q(s, a)$ in the following equation,

$$\begin{aligned} Q^{\pi}(s,a) &= E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi\right] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi\right]. \end{aligned} \tag{5.1}$$

In the equation, γ is the discount factor, which is usually in $[0, 1)$. It means the nearest rewards are worthier than the rewards in the future.

The optimal action policy π^* can be obtained recursively. If the agent knows the optimal Q values of the succeeding states, the optimal policy simply chooses the action that achieves the highest cumulative reward. Thus, the optimal $Q(s, a)$ is calculated based on the optimal Q values of the succeeding states. It can be expressed by the Bellman optimality equation to calculate $Q^{\pi^*}(s, a)$,

$$Q^{\pi^*}(s,a) = E_{s'}\left[r_t + \gamma \max_{a'} Q^{\pi^*}(s', a') | s, a\right]. \tag{5.2}$$

The intuition is that the cumulative reward is equal to the sum of the immediate reward and optimal future reward thereafter. If the estimated optimal future reward can be obtained, the cumulative reward since now can be calculated. This equation can be solved by dynamic programming, but it requires that the number of states is finite to make the computing complexity manageable. When the number of states becomes large, a function θ is needed to approximate the Q value.

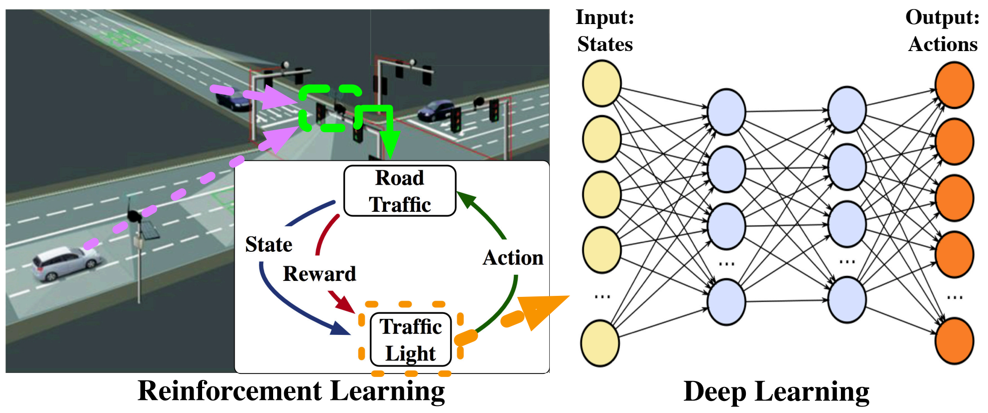


Figure 5.1 The traffic light control model in the proposed system.

5.3 Problem Statement

This chapter targets on controlling the traffic lights at road intersections. A traffic light at an intersection has three signals: green, yellow and red. When there are vehicles from multiple directions at an intersection, one traffic light may not be enough to manage all the vehicles and multiple traffic lights need to cooperate at a multi-direction intersection. A status is defined as one of all the legal combinations of all traffic lights' red and green signals omitting the yellow signals. At an intersection, the traffic signal guides vehicles from non-conflicting directions at one time by changing the traffic lights' statuses. The time duration staying at one status is called one phase. The number of phases is decided by the number of legal statuses at an intersection. All the phases cyclically change in a fixed sequence to guide vehicles to pass the intersection. It is called a cycle when the phases repeat once. The sequence of phases in a cycle is fixed, but the duration of every phase is adaptive based on the current traffic condition. If one phase needs to be skipped, its duration can be set 0 second. In this problem, the duration in every phase is dynamically adjusted to deal with different traffic situations at an intersection to minimize the delay.

The problem in this chapter is to optimize the efficiency of the intersection usage by dynamically changing every phase's duration of a traffic light via learning from historical experiences. The duration of a phase should be extended strategically if there are more vehicles in that direction. In this chapter, a deep Q learning network is built to learn the timing strategy of every phase to optimize the traffic management. The proposed network self-updates by continuously receiving states and rewards from the environment. The model is shown in Figure 5.1. The left side shows the structure in a traffic light. The traffic light first gathers road traffic information via a vehicular network [37] or other tools, which is presented by the dashed purple lines in the figure. The traffic light processes the data to obtain the road traffic's state and reward, which has been assumed in many previous studies [29, 121, 28]. The traffic light chooses an action based on the current state and reward using a deep neural network shown in the right side. The left side is the reinforcement learning part and the right side is the deep learning part.

5.4 Reinforcement Learning Model

In this section, the three elements of the proposed RL model are defined: states, actions and rewards.

5.4.1 States

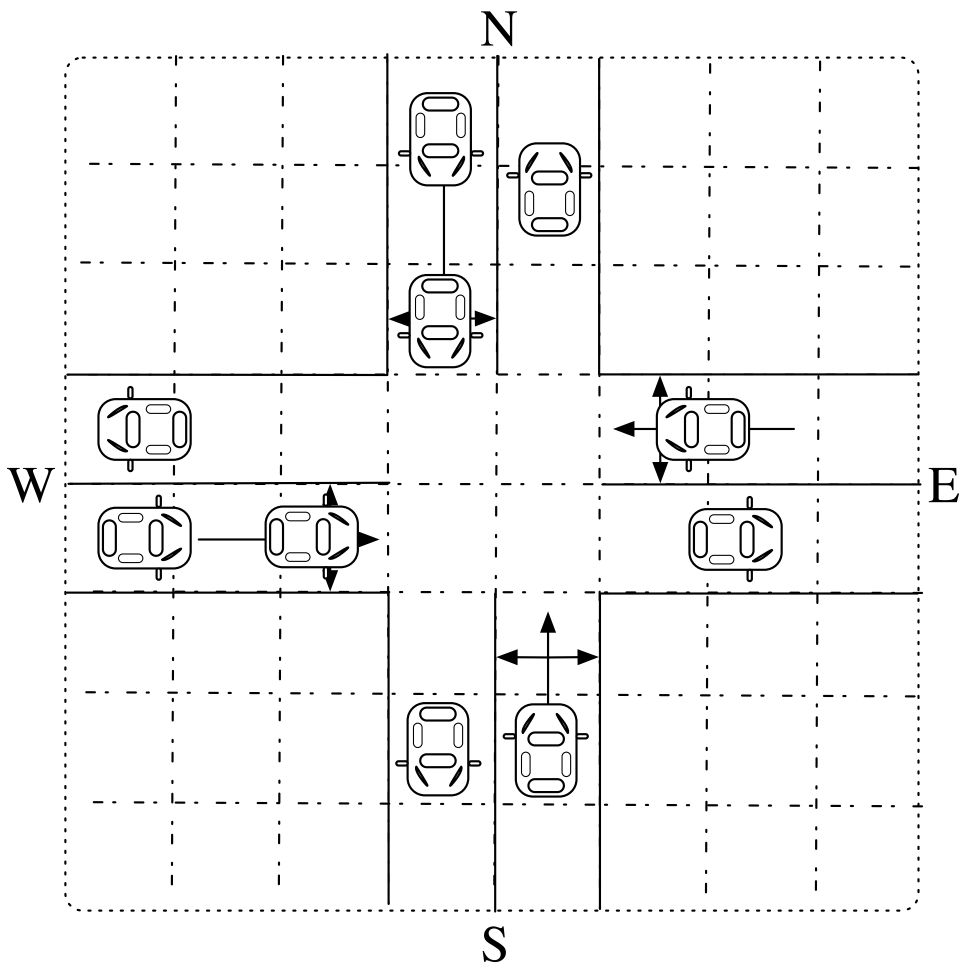
The states are defined based on the position and speed of vehicles at an intersection. Through a vehicular network or other tools, vehicles' position and speed can be obtained [37]. The traffic light can extract a virtual snapshot image of the current intersection. The whole intersection is divided into same-size small square-shape grids. The length of grids, c, should guarantee that no two vehicles can be held in the same grid and one entire vehicle can be put into a grid to reduce computation. In every grid, the state value is a two-value vector $< position, speed >$ of the inside vehicle. The position dimension is a binary value, which denotes whether there is a vehicle in the grid. If there is a vehicle in a grid, the value in the grid is 1; otherwise, it is 0. The value in the speed dimension is an integer, denoting the vehicle's current speed in m/s.

Fig. 5.2 is an example to show how to set up the state values. Figure 5.2(a) shows a snapshot of the traffic status at a simple one-lane four-way intersection, which is divided into square-shape grids. The position matrix has the same size of the grids, which is shown in Figure 5.2(b). In the matrix, one cell corresponds to one grid in Figure 5.2(a). The blank cells mean no vehicle in the corresponding grid, which are 0. The other cells with vehicles inside are set 1.0. The value in the speed dimension is built in a similar way. If there is a vehicle in the grid, the corresponding value is the vehicle's speed; otherwise, it is 0.

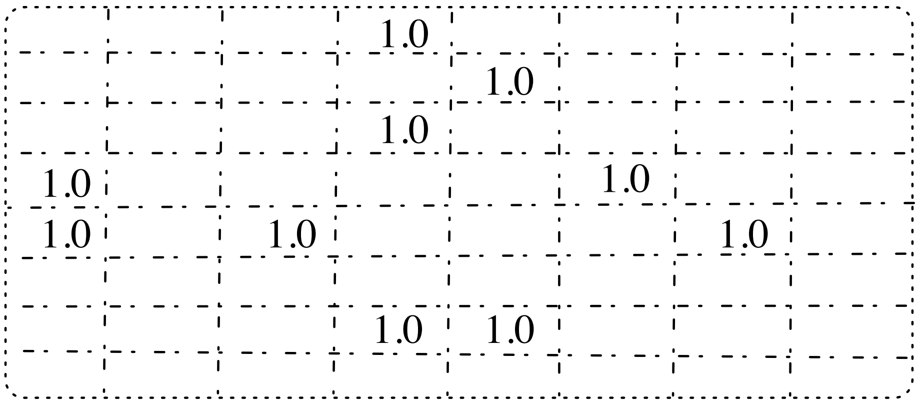
5.4.2 Actions

In the proposed model, the actions' space is defined by how to update the duration of every phase in the next cycle. Considering the system may become unstable if the duration change between two cycles is too large, a change step is specified. In this chapter, it is set to be 5 seconds. The duration changes of two phases between two neighboring cycles are modeled as a high-dimension MDP. In the model, the traffic light changes only one phase's duration by 5 seconds if there is any change.

The intersection in Figure 5.2(a) is taken as an example. At the intersection, there are four phases, north-south green, north-east&south-west green, east-west green, and east-south&west-north green. The other unmentioned directions are red by default. The yellow signals are omitted here and will be presented later. Let a four-tuple $< t_1, t_2, t_3, t_4 >$ denote the duration of the four phases in current cycle. The legal actions in the next cycle is shown in Figure 5.3. In the figure, one circle means the durations of the four phases in one cycle. Note that the duration change from the current cycle to the succeeding cycle is 5 seconds. The duration of one and only one phase in the next cycle is the current duration added or subtracted by 5 seconds. After choosing the phases' duration in the next cycle, the current duration becomes the chosen one. The traffic light can select an action in a similar way as the

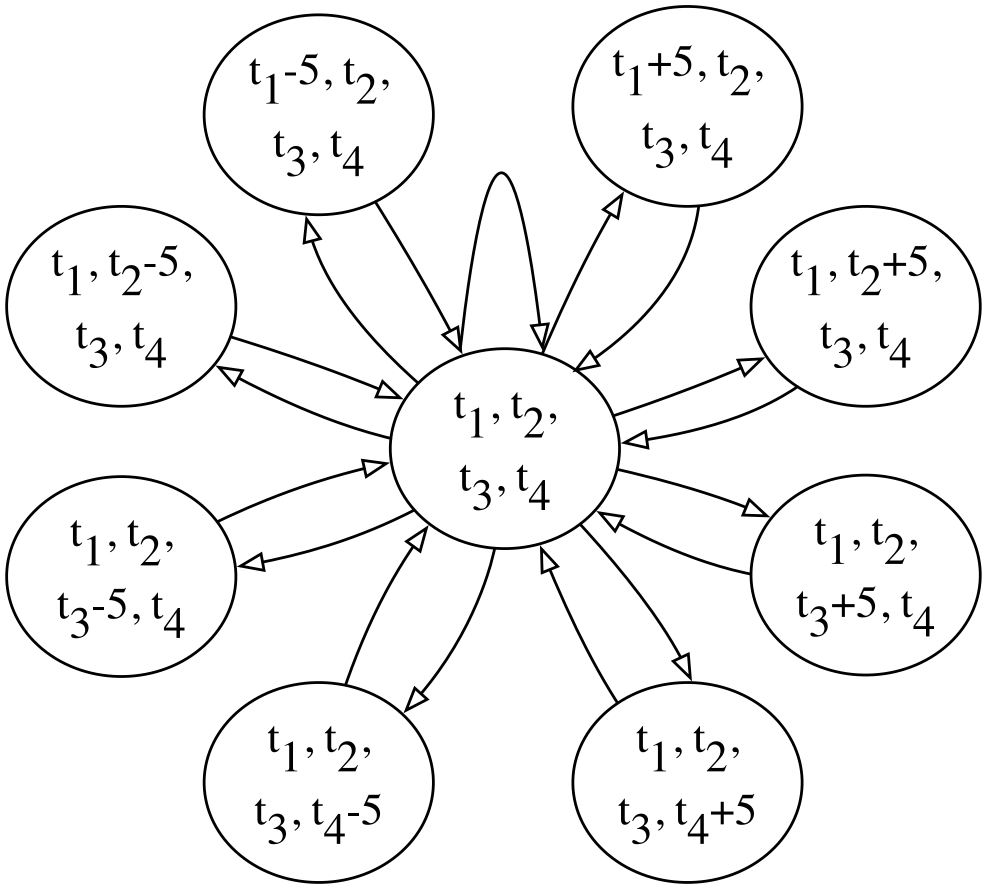


(a) The snapshot of traffic on a road at one moment



(b) The corresponding position matrix on this road

Figure 5.2 Process to build the state matrix.



t_1-5, t_2, t_3, t_4
t_1+5, t_2, t_3, t_4
t_1, t_2-5, t_3, t_4
t_1, t_2+5, t_3, t_4
t_1, t_2, t_3, t_4
t_1, t_2, t_3-5, t_4
t_1, t_2, t_3+5, t_4
t_1, t_2, t_3, t_4-5
t_1, t_2, t_3, t_4+5

Figure 5.3 Part of the Markov decision process in a multiple traffic lights scenario.

previous procedure. In addition, the max duration of a phase is set 60 seconds and the minimal is 0 second.

The MDP is a flexible model. It can be applied into a more complex intersection with more traffic lights, such as an irregular intersection with five or six ways, which needs more phases. When there are more phases at an intersection, they can be added in the MDP model as a higher-dimension value. The dimension of the circle in the MDP is equal to the number of phases at the intersection.

The phases in a traffic light cyclically change in sequence. Yellow signal is required between two neighboring phases to guarantee safety, which allows running vehicles to stop before signals become red. The yellow signal duration T_{yellow} is defined by the maximum speed v_{max} on that road divided by the most commonly-seen decelerating acceleration a_{dec}.

$$T_{yellow} = \frac{v_{max}}{a_{dec}}. \tag{5.3}$$

It means the running vehicle needs such a length of time to firmly stop in front of the intersection.

5.4.3 Rewards

The role of rewards is to provide feedback to a reinforcement learning model about the performance of the previous actions. It is important to define the reward appropriately so to correctly guide the learning process, which accordingly helps take the best action policy.

In the proposed system, the main goal is to increase the efficiency of an intersection and reduce the waiting time of vehicles. Thus, the rewards are defined as the change of the cumulative waiting time between two neighboring cycles. Let i_t denote the i^{th} observed vehicle from the starting time to the starting time point of the t^{th} cycle and N_t denote the corresponding total number of vehicles till the t^{th} cycle. The waiting time of vehicle i till the t^{th} cycle is denoted by $w_{i_t,t}, (1 \le i_t \le N_t)$.

The reward in the t^{th} cycle is defined by the following equation,

$$r_t = W_t - W_{t+1}, \tag{5.4}$$

where

$$W_t = \sum_{i_t=1}^{N_t} w_{i_t,t}. \tag{5.5}$$

It means the reward is the increment in cumulative waiting time between before taking the action and after the action. If the reward in the current cycle becomes larger than before, the waiting time increases less than before. Considering the delay is non-decreasing with time, the overall reward is always negative. The proposed model aims to maximize the reward so to reduce the waiting time.

5.5 Double Dueling Deep Q Network

In the traffic light control system in vehicular networks, the number of states are very large, and thus it is challenging to directly solve equation (5.2). In this chapter, a CNN is proposed [67] to approximate the Q value. Combining with the state-of-the-art techniques, the proposed whole network is called Double Dueling Deep Q Network (3DQN).

5.5.1 Convolutional Neural Network

The architecture of the proposed CNN is shown in Figure 5.4. It is composed of three convolutional layers and several fully-connected layers. In the proposed system, the input is the small grids including the vehicles' position and speed information. The number of grids at an intersection is 60×60. The input data become $60 \times 60 \times 2$ with both position and speed information. The data are first put through three convolutional layers. Each convolutional layer includes three parts, convolution, pooling and activation. The convolutional layer includes multiple filters. Every filter contains a set of weights, which aggregates local patches in the previous layer and

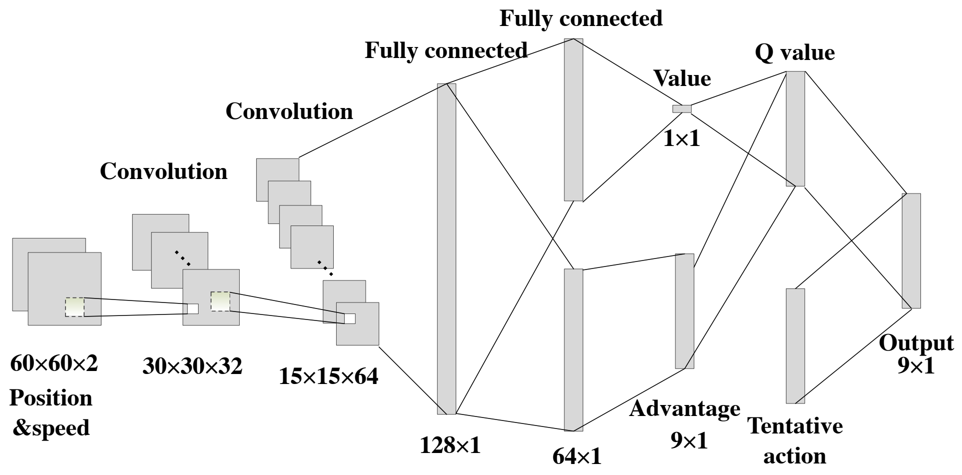


Figure 5.4 The architecture of the deep convolutional neural network to approximate the Q value.

shifts a fixed length of step defined by the stride each time. Different filters have different weights to generate different features in the next layer. The convolutional operation makes the presence of a pattern more important than the pattern's position. The pooling layer selects the salient values from a local patch of units to replace the whole patch. The pooling process removes less important information and reduces the dimensionality. The activation function is to decide how a unit is activated. The most common way is to apply a non-linear function on the output. In this chapter, the leaky ReLU [38] is employed as the activation function with the following form (let x denote the output from a unit),

$$f(x) = \begin{cases} x, & \text{if } x > 0, \\ \\ \beta x, & \text{if } x \leq 0. \end{cases} \tag{5.6}$$

β is a small constant to avoid zero gradient in the negative side. The leaky ReLU can converge faster than other activation functions, such as tanh and sigmoid, and prevent the generation of "dead" neurons from regular ReLU.

In the architecture, three convolutional layers and full connection layers are constructed as follows. The first convolutional layer contains 32 filters. Each filter's size is 4×4 and it moves 2×2 stride every time through the full depth of the input data. The second convolutional layer has 64 filters. Each filter's size is 2×2 and it moves 2×2 stride every time. The size of the output after two convolutional layers is $15 \times 15 \times 64$. The third convolutional layer has 128 filters with the size of 2×2 and the stride's size is 1×1. The third convolutional layer's output is a $15 \times 15 \times 128$ tensor. A fully-connected layer transfers the tensor into a 128×1 matrix. After the fully-connected layer, the data are split into two parts with the same size 64×1. The first part is then used to calculate the value and the second part is for the advantage. The advantage of an action means how well it can achieve by taking an action over all the other actions. Because the number of possible actions in this system is 9 as shown in Figure 5.3, the size of the advantage is 9×1. They are combined again to get the Q value, which is the architecture of the dueling Deep Q Network (DQN).

With the Q value corresponding to every action, illegal actions must be highly penalized, which may cause accidents or reach the max/min signal duration. The output combines the Q value and tentative actions to force the traffic light to take a legal action. Finally the Q values of every action are obtained in the output with penalized values. The parameters in the CNN is denoted by θ. $Q(s, a)$ now becomes $Q(s, a; \theta)$, which is estimated under the CNN θ. The details in the architecture are presented in the next subsections.

5.5.2 Dueling DQN

As mentioned before, the proposed network contains a dueling DQN[127]. In the network, the Q value is estimated by the value at the current state and each action's advantage compared to other actions. The value of a state $V(s; \theta)$ denotes the overall expected rewards by taking probabilistic actions in the future steps. The advantage corresponds to every action, which is defined as $A(s, a; \theta)$. The Q value is the sum of the value V and the advantage function A, which is calculated by the following equation,

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \frac{1}{|A|} \sum_{a'} A(s,a';\theta) \right). \tag{5.7}$$

$A(s,a;\theta)$ shows how important an action is to the value function among all actions. If the A value of an action is positive, it means the action shows a better performance in numerical rewards compared to the average performance of all possible actions; otherwise, if the value of an action is negative, it means the action's potential reward is less than the average. It has been shown that the subtraction from the mean of all advantage values can improve the stability of optimization compared to using the advantage value directly. The dueling architecture is shown to effectively improve the performance in reinforcement learning.

5.5.3 Target Network

To update the parameters in the neural network, a target value is defined to help guide the update process. Let $Q_{target}(s,a)$ denote the target Q value at the state s when taking action a. The neural network is updated by the Mean Square Error (MSE) in the following equation,

$$J = \sum_{s} P(s)[Q_{target}(s,a) - Q(s,a;\theta)]^2, \tag{5.8}$$

where $P(s)$ denotes the probability of state s in the training mini-batch. The MSE can be considered as a loss function to guide the updating process of the primary network. To provide stable update in each iteration, a separate target network θ^-, the same architecture as the primary neural network but different parameters, is usually employed to generate the target value. The calculation of the target Q value is presented in the double DQN part.

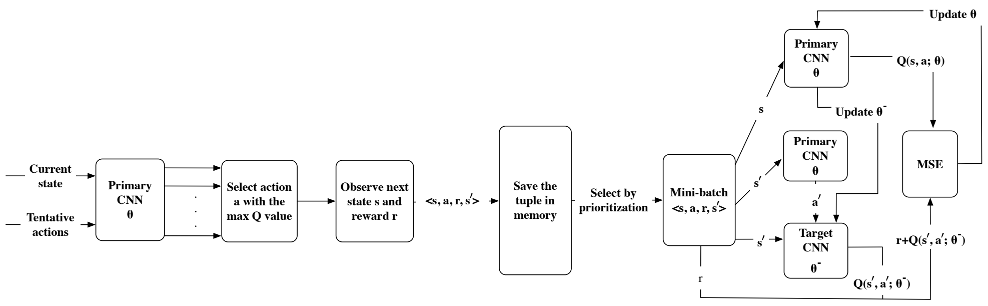


Figure 5.5 The architecture of the reinforcement learning model in this system.

The parameters θ in the primary neural network are updated by back propagation with Equation (5.8). θ^- is updated based on the θ in the following equation,

$$\theta^- = \alpha\theta^- + (1 - \alpha)\theta. \tag{5.9}$$

α is the update rate, which presents how much the newest parameters affect the components in the target network. A target network can help mitigate the over optimistic value estimation problem.

5.5.4 Double DQN

The target Q value is generated by the double Q-learning algorithm [122]. In the double DQN, the target network is to generate the target Q value and the action is generated from the primary network. The target Q value can be expressed in the following equation,

$$Q_{target}(s, a) = r + \gamma Q(s', \arg\max_{a'}(Q(s', a'; \theta)), \theta^-). \tag{5.10}$$

It is shown that the double DQN effectively mitigates the overestimation and improves the performance [122].

In addition, the ϵ-greedy algorithm is employed to balance the exploration and exploitation in choosing actions. With the increasing steps of training process, the value of ϵ decreases gradually. The starting and ending values of ϵ are set. The number of steps is set to reach the ending value from the starting value. The value of ϵ linearly decreases to the ending value. When ϵ reaches the ending value, it keeps the value in the following procedures.

5.5.5 Prioritized Experience Replay

During the updating process, the gradients are updated through the experience replay strategy. A prioritized experience replay strategy chooses samples from the memory based on priorities, which can lead to faster learning and to better final policy[100]. The key idea is to increase the replay probability of the samples that have a high temporal difference error. There are two possible methods estimating the probability of an experience in a replay, proportional and rank-based. Rank-based prioritized experience replay can provide a more stable performance since it is not affected by some extreme large errors. In this system, the rank-based method is taken to calculate the priority of an experience sample. The temporal difference error δ of an experience sample i is defined in the following equation,

$$\delta_i = |Q(s, a; \theta)_i - Q_{target}(s, a)_i|. \tag{5.11}$$

The experiences are ranked by the errors and then the priority p_i of experience i is the reciprocal of its rank. Finally, the probability of sampling the experience i is calculated in the following equation,

$$P_i = \frac{p_i^\tau}{\sum_k p_k^\tau}. \tag{5.12}$$

τ presents how much prioritization is used. When τ is 0, it is random sampling.

5.5.6 Optimization

In this chapter, the neural networks are optimized by the ADAptive Moment estimation (Adam) [52]. The Adam is evaluated and compared with other back propagation optimization algorithms in [97], which concludes that the Adam attains satisfactory overall performance with a fast convergence and adaptive learning rate. The Adam optimization method adaptively updates the learning rate considering both first-order and second-order moments using the stochastic gradient descent procedure. Specifically, let $\boldsymbol{\theta}$ denote the parameters in the CNN and $J(\boldsymbol{\theta})$ denote the loss function. Adam first calculates the gradients of the parameters,

$$\mathbf{g} = \nabla_{\theta} J(\boldsymbol{\theta}). \tag{5.13}$$

It then respectively updates the first-order and second-order biased moments, $\mathbf{s}$ and $\mathbf{r}$, by the exponential moving average,

$$\begin{aligned} \mathbf{s} &= \rho_s \mathbf{s} + (1 - \rho_s)\mathbf{g}, \\ \mathbf{r} &= \rho_r \mathbf{r} + (1 - \rho_r)\mathbf{g}, \end{aligned} \tag{5.14}$$

where ρ_s and ρ_r are the exponential decay rates for the first-order and second-order moments, respectively. The first-order and second-order biased moments are corrected using the time step t through the following equations,

$$\begin{aligned} \hat{\mathbf{s}} &= \frac{\mathbf{s}}{1 - \rho_s^t}, \\ \hat{\mathbf{r}} &= \frac{\mathbf{r}}{1 - \rho_r^t}. \end{aligned} \tag{5.15}$$

Finally the parameters are updated as follows,

$$\begin{aligned} \boldsymbol{\theta} =& \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \\ =& \boldsymbol{\theta} + \left(-\epsilon_r \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}} \right), \end{aligned} \tag{5.16}$$

where ϵ_r is the initial learning rate and δ is a small positive constant to attain numerical stability.

5.5.7 Overall Architecture

The proposed deep learning architecture is illustrated in Figure 5.5. The current state and the tentative actions are fed to the primary convolutional neural network to choose the most rewarding action. The current state and action along with the next state and received reward are stored into the memory as a four-tuple $\langle s, a, r, s' \rangle$. The data in the memory are selected by the prioritized experience replay to generate mini-batches and they are used to update the primary neural network's parameters. The target network θ^- is a separate neural network to increase stability during the learning. Double DQN [122] and dueling DQN [127] are used to reduce the possible overestimation and improve performance. Through this way, the approximating function can be trained and the Q value at every state to every action can be calculated. The optimal policy can then be obtained by choosing the action with the max Q value.

The pseudocode of the proposed 3DQN with prioritized experience replay is shown in Algorithm 2. Its goal is to train a mature adaptive traffic light, which can change its phases' duration based on different traffic scenarios. The agent first chooses actions randomly till the number of steps is over the pre-train steps and the memory has enough samples for at least one mini-batch. Before the training, every samples' priorities are the same. Thus, they are randomly selected into a mini-batch to train. After training once, the samples' priorities change and they are selected by different probabilities. The parameters in the neural network is updated by the Adam back propagation [97]. The agent chooses actions based on the ϵ and the action that has the max Q value. The agent finally learns to get a high reward by reacting on different traffic scenarios.

5.6 Evaluation

5.6.1 Evaluation Methodology and Parameters

Evaluation Metrics The proposed model's objective is to maximize the defined reward, which is to reduce the cumulative delay of all vehicles. Thus, the performance of the proposed model is evaluated using the following two metrics: cumulative reward and average waiting time. The cumulative reward is measured by adding up the

Algorithm 2: 3DQN with Prioritized Experience Replay Algorithm

```
Input   : replay memory size $M$, minibatch size $B$, greedy $\epsilon$, pre-train steps
          $tp$, target network update rate $\alpha$, discount factor $\gamma$
Output: A well learnt 3DQN model
Notation:
$\theta$: the parameters in the primary neural network.
$\theta^-$: the parameters in the target neural network.
$m$: the replay memory.
$i$: step number.
1  Initialize parameters $\theta$, $\theta^-$ with random values.
2  Initialize $m$ to be empty and $i$ to be zero.
3  Initialize $s$ with the starting scenario at the intersection.
   while there exists a state $s$ do
4      Choose an action $a$ according to the $\epsilon$ greedy.
5      Take an action $a$ and observe reward $r$ and new state $s'$.
       if the size of memory $m > M$ then
6          Remove the oldest experiences in the memory.
       end
7      Add the four-tuple $\langle s, a, r, s' \rangle$ into $M$.
8      Assign $s'$ to $s$: $s \leftarrow s'$.
9      $i \leftarrow i + 1$.
       if $|M| > B$ and $i > tp$ then
10         Select $B$ samples from $m$ based on the sampling priorities.
11         Calculate the loss $J$:
12           $J = \sum_s \frac{1}{B}[r + \gamma Q(s', \arg\max_{a'}(Q(s', a'; \theta)), \theta^-) -$
                 $Q(s, a; \theta)]^2$.
13         Update $\theta$ with $\nabla J$ using Adam back propagation.
14         Update $\theta^-$ with $\theta$:
15           $\theta^- = \alpha\theta^- + (1 - \alpha)\theta$.
16         Update every experience's sampling priority based on $\delta$.
17         Update the value of $\epsilon$.
       end
   end
```

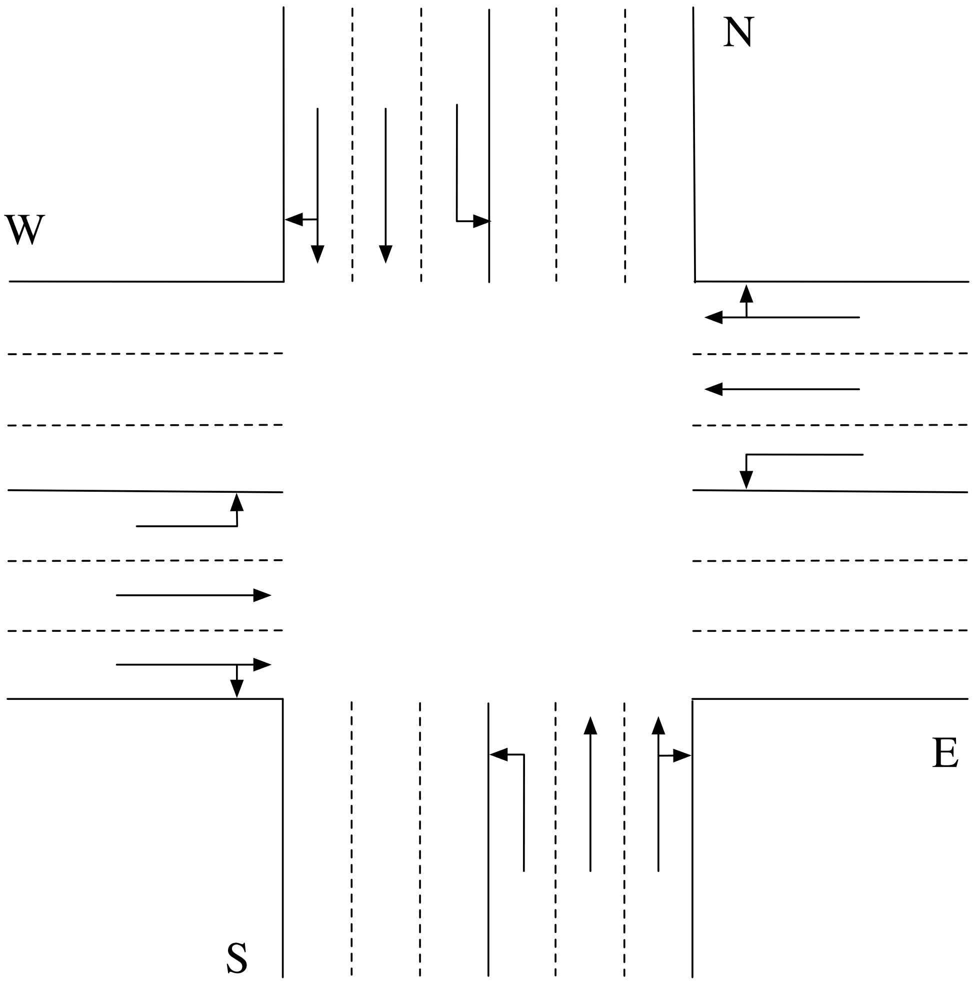


Figure 5.6 The intersection scenario tested in the evaluation.

rewards of all cycles in every episode within one hour period. The average waiting time is measured by dividing the total waiting time by the number of vehicles in an episode.

Traffic Parameters The evaluation is conducted in SUMO [54], which provides real-time traffic simulation. Python APIs provided by SUMO are used to obtain the intersection's information and to send orders to change the traffic light's timing. The intersection is composed of four perpendicular roads, as shown in Figure 5.6. Each road has three lanes. The right-most lane allows right-turn and through traffic, the middle lane only allows through traffic, and the left inner lane allows only left-turn traffic. The simulated intersection is a $300m \times 300m$ area. The grid length c is 5 meters, which means the total number of grids is 60×60. The lane length is 150 meters. Vehicles are 5 meters long and the minimal gap between two vehicles is 2 meters. Vehicles arrive at the intersection following a random process. The average vehicle arrival rate in every lane is 1/10 per second (i.e., on average, there is one vehicle arriving every 10 seconds). Two lanes allow for through traffic, so the flow rate of all through traffic (west-to-east, east-to-west, north-to-south, south-to-north) is 2/10 per second (i.e., on average, there are two vehicles every 10 seconds). Turning traffic (east-to-south, west-to-north, south-to-west, north-to-east) is 1/10 per second. Krauss following model [55] is used for vehicles on the road, which guarantees safe driving. The max speed of a vehicle is 13.9 m/s (50 km/h). The max accelerating acceleration is 1.0 m/s^2 and the decelerating acceleration is 4.5 m/s^2. The duration of yellow signals T_{yellow} is set to be 4 seconds.

Model Parameters The model is trained in iterations. One iteration is an episode in an hour. The reward is accumulated in an episode. The simulation results are the average of 50 iterations. The development environment is built on the top of Tensorflow [1]. The parameters in the deep learning network are shown in Table 5.1.

Comparison Study The performance of the proposed system is compared with three strategies: The first one is the simplest setting: the traffic light's duration is fixed. It is set 30 seconds and 40 seconds for every phase. The second one is a conventional method called Adaptive Traffic Signal Control (ATSC) [90], which set the traffic lights with fixed-time signals. The third one is a state-of-the-art method, Deep Q Network (DQN) [63]. In ATSC, the authors propose a Webster's method to estimate the optimal light time duration based on the most recent cycles' saturation. In DQN, the authors propose to use reinforcement learning with an auto-encoder. They use the queue length as the state to control traffic lights. The authors show that one single Q network can learn good control strategy in a two-phase intersection. Regarding the proposed framework, the ablation studies are also conducted with different reinforcement learning architectures and different parameters to present the proposed model's good performance.

Table 5.1 Parameters in the Reinforcement Learning Network

Parameter	*Value*
Replay memory size M	20000
Starting ϵ	1
Ending ϵ	0.01
Steps from starting ϵ to ending ϵ	10000
Pre-training steps tp	2000
Target network update rate α	0.001
Discount factor γ	0.99
Learning rate ϵ_r	0.0001
Leaky ReLU β	0.01

5.6.2 Experimental Results

Cumulative reward The cumulative reward in every episode is first evaluated under the same traffic flow rate from every lane. All strategies have the same rewards as the proposed work. Note that the aim is to maximize the the rewards, which is to minimize the cumulative waiting time, represented as a negative number. The simulation results are shown in Figure 5.7. From this figure, it is seen that the proposed 3DQN outperforms the other strategies. Specifically, the cumulative reward in 3DQN is greater than -50000 (note that the reward is negative since the vehicles' delay is positive) while that in the two fixed-time strategies is less than -6000. The fixed-time traffic signals always obtains a low reward even after more iterations while the proposed model can learn to achieve a higher reward with more iterations. This is because the fixed-time traffic signals do not change the signals' time under different traffic scenarios. DQN's performance is very unstable, which cannot accurately capture the whole information in a complex intersection by a deep neural network with the queue length only. Because the normal traffic scenario is much more complex than a simple two-phase intersection, the traffic information represented by queue length is inaccurate, which makes DQN choose false actions when two traffic scenarios are different but the queue length is the same. In addition, one network in DQN is easy to overfit the training data. ATSC only chooses the phases' time duration based on several previous cycles, which is inaccurate to predict the future traffic scenarios. In 3DQN, the signals' time changes to achieve the best expected rewards, which learns a more general strategy to handle different traffic states. When the training process iterates over 1000 times, the cumulative rewards become more stable than previous iterations. It means 3DQN has learnt how to handle different traffic scenarios to get the most rewards after 1000 iterations.

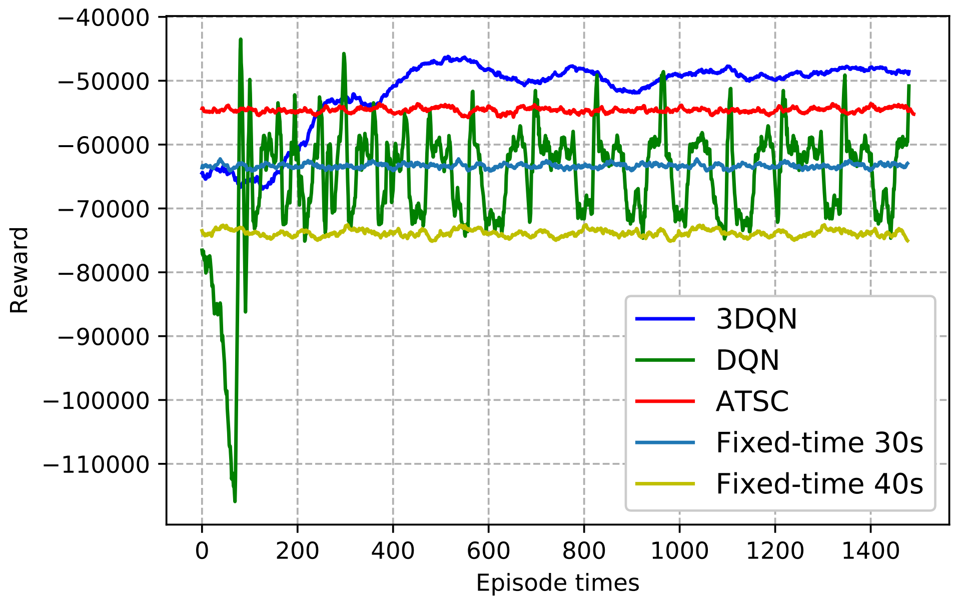


Figure 5.7 Cumulative reward during all the training episodes.

Average waiting time The average waiting time of vehicles in every episode is calculated, which is shown in Figure 5.8. From this figure, it is shown that 3DQN outperforms the other four strategies. Specifically, the average waiting time in the fixed-time signals is always over 35 seconds. The proposed model can learn to reduce the waiting time to about 26 seconds after 1200 iterations from over 35 seconds, which is at least 25.7% less than the fixed-time strategies. ATSC can get better performance than the fixed-time strategy, but it only uses several most recent cycles' information, which cannot well represent future traffic. DQN's performance is very unstable, which means one neural network with the queue length cannot accurately capture the real traffic information. The results show that the proposed model can obtain the most stable and best performance in vehicles' average waiting time among all the methods.

Ablation studies In this part, the proposed model is evaluated by comparing to others with different parameters and different architectures. In the proposed model, a series of techniques are employed to improve the performance of deep Q networks. For comparison, one of these techniques is removed each time to see how every technique

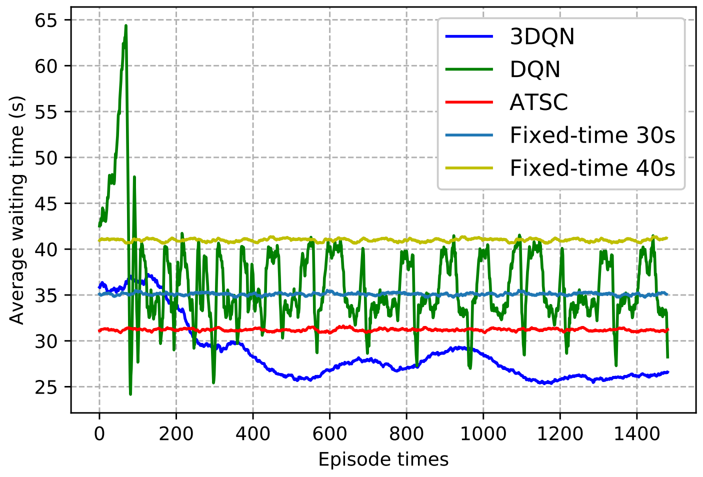


Figure 5.8 Average waiting time during all the training episodes.

influences the performance. The techniques include double network, dueling network and prioritized experience replay. The reward changes in all methods are shown in Figure 5.9. It is shown that the proposed model can learn fastest among the four models. It means the proposed model reaches the best policy faster than others. Specifically, even there is some fluctuation in the first 400 iterations, the proposed model still outperforms the other three after 500 iterations. The proposed model can achieve greater than -47000 rewards while the others have less than -50000 rewards.

Average waiting time under rush hours In this part, the proposed model is evaluated by comparing the performance under the rush hours. The rush hour means the traffic flows from all lanes are not the same, which is usually seen in the real world. During the rush hours, the traffic flow rate from one direction doubles, and the traffic flow rates in the other lanes keep the same as normal hours. Specifically, in the experiments, the arrival rate of vehicles on the lanes from the west to east becomes 2/10 per second and the arrival rates of vehicles on the other lanes are still 1/10 per second. The experimental results are shown in Figure 5.10. From the figure, it is shown that the best policy becomes harder to be learnt than the previous

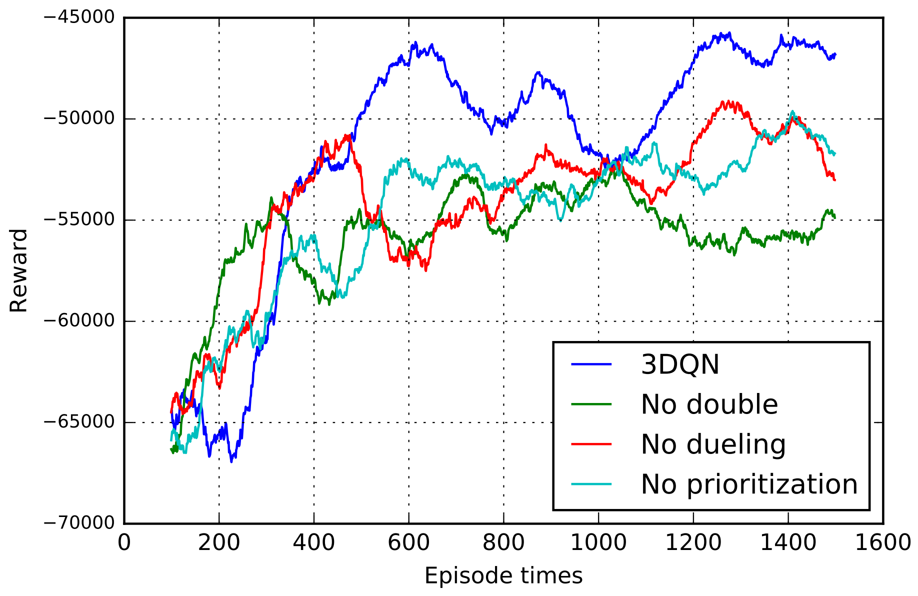


Figure 5.9 Cumulative reward during all the training episodes in different network architectures.

scenario. This is because the traffic scenario becomes more complex, which contains more uncertain factors. But after trial and error, the proposed model can still learn a good policy to reduce the average waiting time. Specifically, the average waiting time in 3DQN is about 33 seconds after 1000 iterations while the average waiting time in the other two fixed-time methods is over 45 seconds. The proposed model reduces about 26.7% of the average waiting than the fixed-time methods. ATSC can achieve better results than one fixed-time method and worse than the other because the optimal phases' time duration in the most recent cycles does not work in the future traffic considering the traffic scenario becomes very complex. DQN's performance becomes more unstable than that in the previous scenario. In summary, 3DQN can achieve the best performance under the rush hours.

5.7 Chapter Summary

In this chapter, the traffic light control problem is proposed to be solved by using the deep reinforcement learning model. The traffic information is gathered from

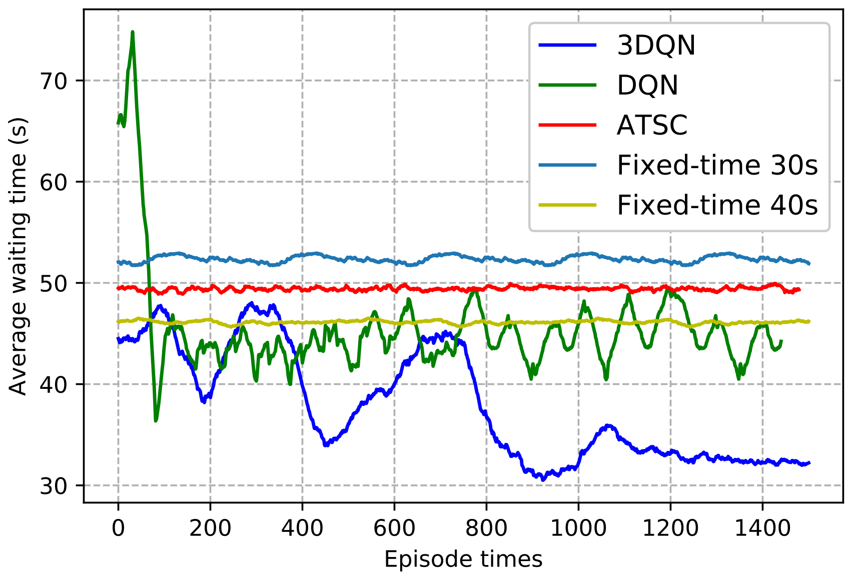


Figure 5.10 Average waiting time in all the training episodes during the rush hours with unbalanced traffic from all lanes.

vehicular networks. The states are three-dimension values with the vehicles' position and speed information. The actions are modeled as a Markov decision process and the rewards are the cumulative waiting time difference between two cycles. To handle the complex traffic scenario in this problem, a Double Dueling Deep Q network (3DQN) with prioritized experience replay is proposed. The model can learn a good policy under both the rush hours and normal traffic flow rates. It can reduce over 20% of the average waiting timing from the starting training. The proposed model also outperforms others in learning speed, which is shown in extensive simulation in SUMO and TensorFlow.

CHAPTER 6

LEARNING K-WAY D-DIMENSIONAL DISCRETE EMBEDDING FOR HIERARCHICAL DATA VISUALIZATION AND RETRIEVAL

6.1 Introduction

Data embedding methods have been successfully deployed in many applications, including unsupervised and supervised data visualization [73, 83, 82], natural language understanding [79, 92, 103], computer vision [26], information retrieval [16], bioinformatics analysis [21], and many others.

These embedding strategies, however, fail to sufficiently reveal essential semantic structures of the data in the embedded space. Typically, these methods associate a real-valued embedding vector with each symbol or data point, which is equivalent to applying a linear transformation to "one-hot" encoding of discrete symbols or data points. Despite their simplicity, these methods are incapable of encoding the internal semantic structure of data, failing to effectively preserve the interplay of the symbols/data points in the embedded space, such as the hierarchical relationship of the symbols or data samples. Hierarchical clusters of data will allow one to know how the symbols/data points are grouped and how lower layer groups form upper layer clusters. Such structural information is, therefore, critical for data understanding and fast information retrieval.

To cope with the aforementioned challenge, this dissertation proposes a regularized autoencoder framework for data embedding. The proposed approach is capable of capturing essential semantic structures of the data, thus leading to both hierarchical data visualization and exploration, and efficient nearest neighbor retrieval. This method builds on the success of the recent K-way D-dimensional discrete encoding [13, 106]. These discrete encoding algorithms encode, through deep neural networks, data points with discrete codes, thus being able to significantly reduce the storage space when compared to real-valued embedding. The goal of this part of the disseration is at enforcing the discrete codes to have structural information: different bits of a code are used to identify their relationships with other data points. In detail, a regularized autoencoder is leveraged to learn compact hierarchical K-way D-dimensional discrete embedding of symbols or data points. An autoencoder framework is employed with a discrete embedding layer regularized by a stochastic exemplar-centered neighborhood preserving loss, in which different dimensions of a discrete code vector are combined using exponentially decaying weights to achieve Hierarchical K-way D-dimensional embedding (HKD). Consequently, the HKD embedding codes have a tree structure, where similar symbols tend to have the same codes in front bits while the back codes are different from each other to separate them. In addition, the autoencoder is regularized to preserve exemplar-centered neighborhoods, resulting in embeddings with similar codes tightly close to each other.

Experimental results on synthetic and real-world datasets show that, the proposed HKD embedding can, in addition to storage efficiency, reveal the semantic structure of data via hierarchical data visualization and greatly reduce search space of nearest neighbor retrieval while preserving high accuracy.

This model is the first one to propose a method to learn hierarchical discrete embedding, thus enabling hierarchical data visualization and fast nearest neighbor retrieval in addition to embedding storage efficiency. These salient features make the proposed embedding strategy particularly attractive in practice, where neither the computation power nor the storage resources are abundant.

6.2 Hierarchical K-way D-dimensional Discrete Embedding

The proposed Hierarchical K-way D-dimensional Discrete Embedding method (denoted as HKD encoding) leverages an autoencoder framework, where data features/embeddings are first encoded into the HKD embeddings which are then required to be able to decode (reconstruct) the original given data features/embeddings. Two novel components are devised to attain the goals of the HKD encoding, as follows. First, the HKD embedding codes have a tree structure, where similar symbols tend to have the same codes in front bits while the back codes are different from each other to separate them. Second, the autoencoder is regularized to preserve exemplar-centered neighborhoods, resulting in embeddings with similar codes tightly close to each other. The two novel components will be discussed in detail as follows.

6.2.1 Learning Hierarchical Discrete Codes with an Autoencoder

The aim of the proposed HKD encoding method is to associate every symbol (data point) with a K-way D-dimensional discrete code. The whole process from embedding to discrete codes and verse vice is illustrated in Figures 6.1, and will be discussed in detail next.

Suppose, the discrete code for the i^{th} symbol (data point) is denoted by $\boldsymbol{c}_i = (\boldsymbol{c}_{i,1}, \boldsymbol{c}_{i,2}, ..., \boldsymbol{c}_{i,D})$, where $\boldsymbol{c}_{i,d}$ is a set of code bits with cardinality K. Consider $\boldsymbol{c}_{i,d}$ is a one-hot vector. With this setting, given a symbol/data point i's embedding $\boldsymbol{e}_i$, the HKD first uses an encoder to learn its discrete codes $\boldsymbol{c}_i$. Next, a decoder in the HKD framework is then deployed to reconstruct the embedding $\hat{\boldsymbol{e}}_i$ to approach the real embedding $\boldsymbol{e}_i$ as much as possible. The encoding and decoding processes are formally formulated as follows.

HKD Encoding As illustrated in Figure 6.1, given the embedding $\boldsymbol{e}_i$, the hidden layers of the neural network first transfer the embedding into the K*D dimensional values $\boldsymbol{h}_i$:

$$\boldsymbol{h}_i = \mathbf{f}(\boldsymbol{M}\boldsymbol{e}_i), \tag{6.1}$$

where $\boldsymbol{M}$ denotes the weights of the hidden layer and $\mathbf{f}$ is a nonlinear activation function with multiple hidden layers.

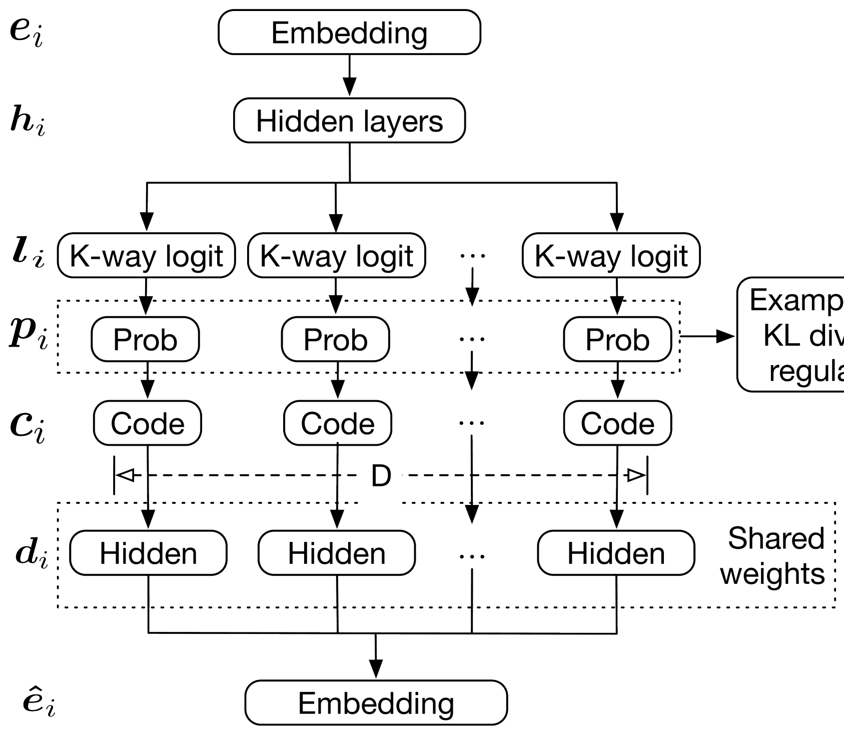


Figure 6.1 Illustration of the framework in the HKD method.

Subsequently, the hidden layer' outputs are equally split into D partitions, where each partition has K values and each corresponds to exactly one dimension in the final discrete codes. Let $\boldsymbol{l}_{i,d}(d = 1, 2, ..., D)$ denote the d^{th} partition and $l_{j,i,d}$ denote the exact j^{th} value in the d^{th} partition, the code probabilities $\boldsymbol{p}_{i,d}$ are then calculated via a Softmax function on every partition, as follows:

$$\begin{aligned} \boldsymbol{p}_{i,d} &= \text{softmax}(\boldsymbol{l}_{i,d}) \\ &= \frac{\exp(\boldsymbol{l}_{i,d})}{\sum_{j=1}^{K} \exp(l_{j,i,d})}. \end{aligned} \tag{6.2}$$

The computed code probabilities $\boldsymbol{p}_{i,d}$ are used to form the discrete encodes by first passing through an argmax function and then representing by a one-hot vector:

$$\boldsymbol{c}_{i,d} = \text{one\_hot}\left(\arg\max_{j} \{p_{j,i,d}\}\right), j = 1, 2, ..., K. \tag{6.3}$$

To cope with the possible gap between the discrete codes and continuous variables, a temperature τ is used to approximate the discrete codes during training as in [13],

$$\boldsymbol{c}_{i,d} \approx \text{softmax}\left(\frac{\boldsymbol{h}_{i,d}}{\tau}\right). \tag{6.4}$$

Similar techniques have been introduced in a Gumbel-Softmax trick [47, 74].

HKD Decoding After the encoding phase, a decoder is applied on the discrete codes generated by the encoder to reconstruct the original embeddings, as follows.

$$\hat{\boldsymbol{e}}_i = \sum_d w_d \, \mathbf{g}(Proj\,(\boldsymbol{A}_{i,d}\boldsymbol{c}_{i,d})), \tag{6.5}$$

where $\mathbf{g}(\cdot)$ is a sub-neural network with one or more hidden layers shared by all code dimensions as in Figure 6.1, $\boldsymbol{A}_{\boldsymbol{i},\boldsymbol{d}}$ is the transformation weights for the i^{th} symbol in the d^{th} dimension, and w_d is the decayed weight for dimension d, which will be presented in the next subsection. $Proj(\boldsymbol{x})$ is a projection function, which is shown as follows,

$$Proj(\boldsymbol{x}) = \begin{cases} \frac{\boldsymbol{x}}{||\boldsymbol{x}||+\epsilon} & \text{if } ||\boldsymbol{x}|| \geq 1 \\ \boldsymbol{x} & \text{otherwise} \end{cases}. \tag{6.6}$$

The loss of the autoencoder here is to minimize the reconstruction error, which is defined by the mean square error,

$$E = \frac{1}{n}\sum_i^n ||\hat{\boldsymbol{e}}_{\boldsymbol{i}} - \boldsymbol{e}_{\boldsymbol{i}}||^2, \tag{6.7}$$

where n denotes the number of data points in the dataset.

To further capture hierarchical semantic structures of the given data in the embedding space, a regularizer is leveraged to force the model to incorporate data neighborhood information during the encoding process, which is discussed next.

6.2.2 Regularized Autoencoder Preserving Neighborhoods

The adopted regularization method aims at enabling the generated discrete KD codes to capture the semantic information in the data. To this end, the parametric t-distributed stochastic exemplar-centered embedding (pt-SEE) strategy [81] is leveraged, by extending pt-SEE to weight different dimensions of the KD codes, to model the neighborhood information of the data points. Pt-SEE is an extension of t-SNE [73], which is an effective method to preserve the neighboring information when learning low-dimensional embeddings. pt-SEE significantly reduces the computational complexity of t-SNE. In specific, unlike t-SNE, pt-SEE does not compute pairwise neighboring probabilities. Instead, it chooses an enough number z exemplars to represent the distribution of raw data ($z \ll n$). The z exemplars can be formed in two ways, one is chosen by running some iterations of k-means on the raw data features/embeddings and the other is randomly chosen from the dataset. Promisingly, it has at most linear computational complexity with respect to the size of the whole dataset.

Formally, let $\boldsymbol{e}_j$ denote the raw embedding of the j^{th} exemplar chosen by k-means or random sampling, where $j \in [1, z]$. Same as before, $\boldsymbol{e}_i$ denotes the raw embedding/feature vector of the i^{th} data point. The neighboring probability in the raw data feature/embedding space is estimated by a Gaussian distribution.

$$\begin{aligned} p_{j|i} &= \frac{\exp(-d(\boldsymbol{e}_i, \boldsymbol{e}_j)/2\sigma_i^2)}{\sum_{k=1}^{z} \exp(-d(\boldsymbol{e}_i, \boldsymbol{e}_k)/2\sigma_i^2)}, \\ p_{j|i} &= \frac{p_{j|i}}{n}. \end{aligned} \tag{6.8}$$

Here, $d(\cdot)$ is a problem-specific distance function, for e.g., squared Euclidean distance or Poincaré distance, $i \in [1, n]$, and $j \in [1, z]$. Variance of the Gaussian distribution σ_i is set such that the perplexity of the conditional distribution $p_{j|i}$ equals to a user-specified perplexity u that can be interpreted as the expected number of nearest exemplars of data point i.

In the proposed HKD encoding approach, because discrete codes cannot be directly used to calculate the neighboring probabilities, the code probabilities are used instead. In detail, to compute the neighboring probabilities in the code space, a t-distribution is used:

$$\begin{aligned} q_{j|i} &= \frac{(1+d_{ij})^{-1}}{\sum_{i=1}^{n}\sum_{j=1}^{z}(1+d_{ij})^{-1}}, \\ d_{ij} &= ||\boldsymbol{p}_i - \boldsymbol{p}_{e_j}||^2. \end{aligned} \tag{6.9}$$

where $\boldsymbol{p}_{e_j}$ denotes the code probabilities of the j^{th} exemplar.

In this way, the neighboring probabilities in the discrete code space are obtained. However, doing so, the KL divergence strategy simply treats every KD code equally. To attain a hierarchical coding, a weighted version of distance calculation is considered, which makes the front codes more important than the back codes, resulting in the following distance calculation formula,

$$d_{ij} = ||\boldsymbol{w} \circ (\boldsymbol{p}_i - \boldsymbol{p}_{e_j})||^2, \tag{6.10}$$

where $\circ$ denotes element-wise multiplication, and $\boldsymbol{w}$ is a weight vector with the same size as $\boldsymbol{p}_i$, in which all the weights for the d^{th} dimension of $\boldsymbol{p}_i$ have the same values w_d calculated by a decay function,

$$w_d = w_0 \exp(-\lambda d), \tag{6.11}$$

where w_0 is the initial starting weight. And the exemplar-based KL divergence is computed as follows,

$$KL = \sum_{i=1}^{n}\sum_{j=1}^{z} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}. \tag{6.12}$$

The final cost function of the proposed HKD approach consists of two parts: the reconstruction error plus the exemplar-based KL divergence.

$$J = \alpha E + \beta KL, \tag{6.13}$$

where β is the penalty coefficient for the KL divergence regularization term.

To train the proposed model, the pseudo-code of the whole training procedure in Algorithm 3. In this algorithm, the goal is to train a network to generate hierarchical K-way D-dimensional discrete codes.

```
Algorithm 3: The training of the proposed HKD algorithm
    Input  : Training dataset $E = \{e_i, i \in [1, n]\}$
    Output: A code generation network including all weights in Fig. 1.
    Notation:
    $z$ : number of examplars
    $b$ : mini-batch size
  1 Select examplars $e_j, (j \in [1, z])$ by running T (e.g., 100) iterations of
      K-means and/or random data sampling.
    while not done do
  2     Randomly sample from $E$ for a mini-batch of size $b$, ß $= \{e_i, i \in [1, b]\}$.
  3     Feed the batch ß and all examplars $e_j$ into the network to calculate the
          loss according to Eq. (6.13).
  4     Update all weights through back-propagation.
    end
```

6.3 Experiments

6.3.1 Settings and Baselines

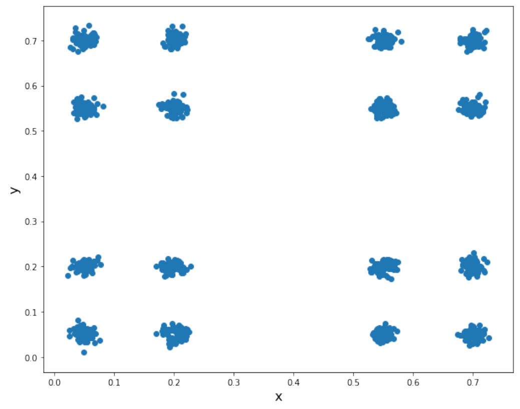
The proposed method is evaluated in terms of its capability to hierarchically organize codes for speeding up nearest neighbor search and visualizing the semantic structure of the given data. To evaluate the method, the following two metrics are adopted. The first one is the percentage of nearest neighbor entities that share the same code in the first N (out of D) codes and the reduced percentage of entities that are not neighbors and have different codes. The second one is the visualization on how the embedding codes correspond to the clusters of the given data.

The proposed approach is compared against the state-of-the-art KD code learning method [106] on three datasets: a synthetic dataset, Poincaré embedding [88] on WordNet [80] and embedding on the CIFAR100 dataset [56]. The first one is a synthetic dataset, aiming at better understanding the behavior of the proposed HKD encoding schema. For the synthetic data, the data are generated using two-dimensional independent Gaussian distributions. There are in total 16 clusters, which is shown in Figure 6.2(a). Second, the proposed encoding schema is evaluated using the Poincaré embedding, with the aim of investigating how the proposed method can keep the hierarchy in the code space. The last experiment uses the widely-used CIFAR100 dataset.

In the evaluation, the examplars are made by two parts, one is the centers generated by k-means, and the other is centers combined with 10 nearest neighbors of every point. It means different points' examplars are different. The number of examplars are 10∼20% of the number of training data points. The network is trained using RMSprop [117] with learning rate of 0.0001 and mini-batch of size 128. The whole model is built using PyTorch [91] and is trained using a GTX 1080 Ti GPU. The hyperparameters are chosen based on the validation data by comparing the magnitude of different loss terms. They are different in different models, which will be presented in the their experimental results.

6.3.2 Results on the Synthetic Dataset

The synthetic dataset is split into two parts, training and test. The code performance on the training set is shown in Table 6.1 and test set is shown in Tables 6.2. In this dataset, the number of hidden units is set 20 considering there are only 2 dimensions in the raw data. The value of D is set 16 and K is 16. The value of α is 0.1. The two KL divergences' weights are both 1 and the perplexities in the two KL divergences are 5 and 11 respectively. The hierarchical property is evaluated by the accuracy of


(a) An illustration to the synthetic dataset

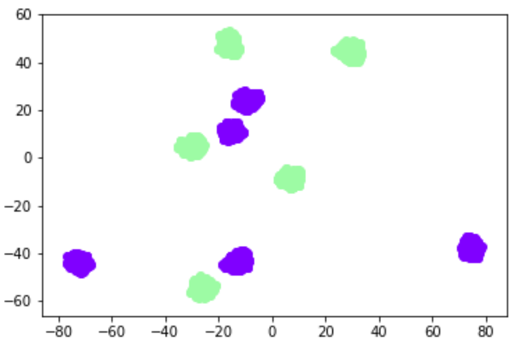

(b) An illustration to the two superclasses in the CIFAR100 dataset

Figure 6.2 Illustration of the two datasets.

the same first N codes in every entity's nearest neighbors. The accuracy is defined by the percentage of same codes in the first N codes in the nearest neighbors. The experiments also evaluate how smaller the search space can be reduced in finding nearest neighbors, which is defined by the average percentage of the number of entities that have the same first N codes to the total number of entities. This metric indicates that the research space from the number of the whole entities can be reduced.

The data visualization is also explored using generated HKD codes. If two codes are exactly the same in the first N dimensions, they have the same color. The results are shown in Figure 6.3. The figures in Figure 6.3 clearly show that the HKD codes can form hierarchical clusters which are consistent with the known clusters of the synthetic data. For example, with the first layer code (sub-figure (a)), the embeddings are clustered into 2 clusters, which are consistent with the known super clusters as shown in Figure 6.2(a). When moving down the hierarchical structure of codes formed, more and more sub-clusters are formed by the HKD codes. As an example, on the second layer of the codes (sub-figure (b)), the two clusters from sub-figure (a) are perfectly divided into four clusters. These four clusters are further divided into 8 clusters when moving down one more layer of the HKD code hierarchy,

Table 6.1 Nearest Neighbor Preserving Percentage by KD Codes on the Training Set of the Synthetic Data

First N layer (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.53	0.32	0.28	0.25	0.21	0.20	0.20	0.2
	HKD	1.0	1.0	0.999	0.999	0.999	0.999	0.999	0.985
KNN=10	Shu *et al.*	0.46	0.23	0.19	0.15	0.10	0.10	0.10	0.10
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.976
KNN=15	Shu *et al.*	0.44	0.21	0.16	0.12	0.08	0.07	0.06	0.06
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.968
KNN=20	Shu *et al.*	0.43	0.19	0.15	0.10	0.06	0.05	0.05	0.05
	HKD	1.0	1.0	0.996	0.996	0.996	0.996	0.996	0.960
Search complexity reduction (%)		63.1	77.3	85.4	89.4	90.9	91.7	92.5	93.7

Table 6.2 Nearest Neighbor Preserving Percentage by KD Codes on the Test Set of the Synthetic Data

First N code (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.57	0.39	0.31	0.24	0.23	0.22	0.20	0.2
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.968
KNN=10	Shu *et al.*	0.52	0.31	0.23	0.14	0.13	0.12	0.11	0.10
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.962
KNN=15	Shu *et al.*	0.49	0.28	0.19	0.10	0.09	0.07	0.06	0.06
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.958
KNN=20	Shu *et al.*	0.43	0.19	0.15	0.10	0.06	0.05	0.05	0.05
	HKD	1.0	1.0	0.996	0.996	0.996	0.996	0.996	0.953
Search complexity reduction (%)		64.3	77.1	85.3	89.1	90.7	91.4	92.3	93.6

as shown in sub-figure (c). When moving down to the either layer of the HKD codes (sub-figure (h)), 16 sub-clusters are formed.

Ablation studies on the synthetic data are also conducted by the following variants: no reconstruction error, no KL divergence, and no decay weights. The results in Table 6.3 clearly indicate that the HKD model can accurately retrieval nearest neighbors via searching codes step by step along the hierarchical structure formed, significantly outperforming the variants. For example, the proposed method with the first two layers of codes can cover 100% of all the data points, which are meaningfully better than about 50% achieved by the best variant. It is shown that the reconstruction, examplar-based KL divergence and decay weights are important in the proposed model.

These results show that the HKD encoding schema can capture the semantic structure of the given data when generating discrete embedding codes. Next, the proposed HKD method is evaluated against real-world datasets.

6.3.3 Results on the Poincaré Embedding of WordNet

Hierarchical embedding can be achieved by the Poincaré embedding method [88]. The embedding generated by the Poincaré embedding method is chosen to train the proposed hierarchical codes to explore whether the codes can maintain the hierarchical property.

In this task, the mammal subtree in the WordNet dataset is selected. In the dataset, there are 1182 entities and 7724 semantic relations among them. The Poincaré embedding is first trained with 10 dimensions per entity for 30 epochs. The trained Poincaré embedding is used to encode the hierarchical KD codes. The code size is 16×16. There are 200 centroids from K-means and 100 samples that are randomly selected as the examplars. The value of α is 1. One KL divergence is used, and its weight β is 1 and perplexity is 10.

Table 6.3 Ablation Study Results of HKD Codes Preserving Nearest Neighbors on the Test Set of the Synthetic Data

First N code (s)		1	2	3	4	5	6	7	8
KNN=5	No reconstruction	0.51	0.19	0.16	0.12	0.08	0.07	0.06	0.06
	No KL divergence	0.17	0.12	0.07	0.06	0.05	0.05	0.05	0.05
	No decay weights	0.45	0.28	0.17	0.11	0.11	0.08	0.07	0.07
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.968
KNN=10	No reconstruction	0.55	0.23	0.19	0.13	0.09	0.08	0.08	0.08
	No KL divergence	0.20	0.14	0.08	0.06	0.06	0.06	0.05	0.05
	No decay weights	0.50	0.31	0.19	0.14	0.14	0.10	0.09	0.08
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.962
KNN=15	No reconstruction	0.55	0.23	0.19	0.13	0.10	0.08	0.08	0.08
	No KL divergence	0.20	0.14	0.09	0.06	0.06	0.06	0.05	0.05
	No decay weights	0.51	0.31	0.19	0.14	0.11	0.10	0.10	0.09
	HKD	1.0	1.0	0.997	0.997	0.997	0.997	0.997	0.958
KNN=20	No reconstruction	0.53	0.25	0.20	0.12	0.10	0.08	0.08	0.08
	No KL divergence	0.21	0.14	0.10	0.07	0.06	0.06	0.05	0.05
	No decay weights	0.52	0.31	0.20	0.15	0.15	0.11	0.10	0.08
	HKD	1.0	1.0	0.996	0.996	0.996	0.996	0.996	0.953

The results on the training data and test data of Poincaré embedding are shown in Tables 6.4 and 6.5, respectively. The first 8 codes are chosen and the number of nearest neighbors between 5 and 20 are selected. It is shown that the proposed method can have about 100% of the five nearest neighbors having the same first codes and the percentage maintains over 90% after searching the first eight codes. Meanwhile, the search space shrinks to only 5.6% of the whole dataset after looking for the first six codes.

To have better insights into the encoding codes, a case study is also conducted using the 'dog' category in the WordNet dataset. Results are presented in Table 6.7. In this table, the entity 'dog.n.01' is chosen as the base and other entities' distances to it are calculated. Comparing the distance and the codes in this table, it is shown

Table 6.4 Nearest Neighbor Preserving Percentage by KD Codes on the Training Set of Poincaré Embedding Dataset

First N layer (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.62	0.36	0.32	0.26	0.23	0.22	0.21	0.19
	HKD	1.0	0.99	0.98	0.97	0.95	0.95	0.94	0.94
KNN=10	Shu *et al.*	0.56	0.28	0.23	0.17	0.14	0.13	0.12	0.08
	HKD	0.99	0.97	0.96	0.94	0.92	0.91	0.90	0.90
KNN=15	Shu *et al.*	0.54	0.24	0.19	0.13	0.10	0.09	0.08	0.06
	HKD	0.98	0.95	0.94	0.93	0.91	0.90	0.90	0.89
KNN=20	Shu *et al.*	0.54	0.23	0.18	0.11	0.09	0.08	0.07	0.05
	HKD	0.97	0.94	0.93	0.92	0.90	0.88	0.87	0.86
Search complexity reduction (%)		77.4	88.1	89.9	92.1	93.6	94.4	95.3	95.7

that, in the proposed method, the nearer the entity is to the 'dog.n.01' entity, the more similar the codes are to those of the 'dog.n.01' entity. More specifically, when the distance is closer, the more codes at the first places are the same as those of the 'dog.n.01' entity. When the distance becomes further, the different codes may become more front. For example, the first code that is different between 'hunting.dog.n.01' and 'dog.n.01' is at the 7^{th} dimension while the first code that is different between 'whitetail prairie dog.n.01' and 'dog.n.01' is at the 2^{nd} dimension because 'whitetail prairie dog.n.01' is further to 'dog.n.01'. Meanwhile, 'flying_fox.n.01' is chosen to show entities at different categories have totally different codes. Comparing to the codes from Shu *et al.* shown in Table 6.6, it is shown that the codes are generated randomly in all dimensions.

6.3.4 Results on the CIFAR100 Dataset

In the CIFAR100 dataset, there are 20 superclasses, and each superclass has 5 classes. This dataset has 50000 training images and 10000 test images in total. The wide ResNet [135] is used to pre-train the dataset based on class information to get every

Table 6.5 Nearest Neighbor Preserving Percentage by KD Codes on the Test Set of Poincaré Embedding Dataset

First N code (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.53	0.20	0.16	0.09	0.04	0.03	0.02	0.02
	HKD	0.99	0.99	0.98	0.97	0.97	0.96	0.95	0.93
KNN=10	Shu *et al.*	0.52	0.20	0.15	0.09	0.04	0.02	0.2	0.01
	HKD	0.99	0.98	0.97	0.96	0.95	0.94	0.92	0.90
KNN=15	Shu *et al.*	0.52	0.20	0.15	0.09	0.04	0.02	0.2	0.01
	HKD	0.99	0.96	0.95	0.94	0.94	0.92	0.92	0.89
KNN=20	Shu *et al.*	0.52	0.20	0.15	0.09	0.04	0.02	0.1	0.01
	HKD	0.98	0.96	0.93	0.92	0.90	0.89	0.87	0.86
Search complexity reduction (%)		76.8	88.0	88.8	91.6	93.2	94.0	94.8	95.4

image's embedding. In this dataset, the number of hidden units is set 100. The value of D is set 16 and K is 16. The value of α is 0.01. In this one, the KL divergence on the centers and nearest neighbors is used, whose weight is 1 and perplexity is 15. Two superclasses with five classes in each superclass are chosen, which are shown in Figure 6.2(b) after t-SNE. Two kinds of colors denote two superclasses. HKD codes in two superclasses are explored via visualization as shown in Figure 6.4. In these figures, dots in one color denote these points share the same codes in the first N dimensions. Through the first code, one class from the two superclasses can be split,

Table 6.6 Code Case Studies on Poincaré Embedding Dataset Using the Method From [Shu and Nakayama, 2018]

Entity	Code	Distance to 'dog.n.01'
dog.n.01	[12 0 8 11 7 14 14 3 4 15 12 11 10 12 15 12]	0.00
hunting_dog.n.01	[12 0 1 14 14 4 3 3 10 13 12 12 10 12 15 14]	0.05
coondog.n.01	[1 7 12 14 7 14 11 3 13 13 5 2 14 9 15 12]	0.29
hearing_dog.n.01	[1 0 5 1 2 4 3 3 13 13 12 1 10 12 2 14]	0.43
crab-eating_dog.n.01	[12 0 6 14 7 12 3 3 0 15 12 11 14 12 15 2]	1.15
whitetail_prairie_dog.n.01	[12 0 6 11 14 5 12 5 3 13 12 4 5 3 8 12]	2.35
flying_fox.n.01	[12 0 6 4 8 5 12 7 10 14 12 1 5 9 8 10]	1.92

which is shown in Figure 6.4(a). When moving down the hierarchical structure of codes formed, more and more classes can be extracted, which are shown from Figure 6.4(b) to Figure 6.4(h). More specifically, the proposed model can find smaller classes, which are grouped by only a few nearest neighbors, which is shown in Figure 6.4(h). The nearest neighbor retrieval results are available in the supplementary material.

6.4 Chapter Summary

In this chapter, a regularized autoencoder framework is proposed to generate hierarchical K-way D-dimensional codes from symbol/data point embeddings. The generated codes can significantly speed up the retrieval process by effectively reducing the search space. Such reduction is attained by making neighbor embeddings hold

Table 6.7 Code Case Studies on Poincaré Embedding Dataset Using the Proposed Method

Entity	Code	Distance to 'dog.n.01'
dog.n.01	[11 13 7 0 6 9 0 15 6 11 13 4 13 3 7 1]	0.00
hunting_dog.n.01	[11 13 7 0 6 9 11 15 6 4 13 4 13 3 7 1]	0.05
coondog.n.01	[11 13 7 0 6 9 13 5 2 11 13 2 13 3 7 1]	0.29
hearing_dog.n.01	[11 13 7 0 6 4 12 13 5 11 14 5 13 3 7 11]	0.43
crab-eating_dog.n.01	[11 13 7 0 6 15 5 11 6 10 13 0 13 3 8 9]	1.15
whitetail_prairie_dog.n.01	[11 12 7 0 13 11 4 1 5 10 13 5 9 12 2 14]	2.35
flying_fox.n.01	[7 13 7 8 13 1 1 1 6 10 14 4 6 3 0 10]	1.92

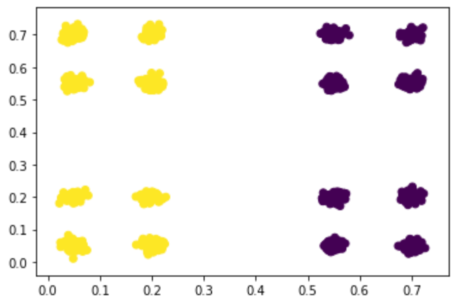
the same codes in the front dimensions, through leveraging code combinations with exponentially decaying weights and embracing an examplar-based KL divergence loss. Experimental results on synthetic and real-world datasets show that the proposed method can successfully build a hierarchical structure in the discrete KD codes, with over 90% nearest neighbors sharing the same codes in the first several dimensions. The empirical studies also indicate that the proposed approach can reveal the semantic structure of data via hierarchical data visualization.

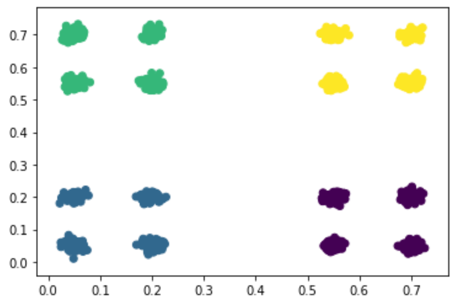
Table 6.8 Nearest Neighbor Preserving Percentage by KD Codes on the Training Set of the CIFAR100 Data

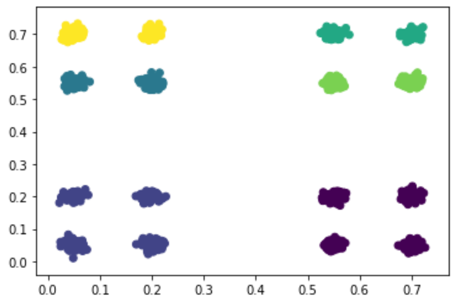
First N layer (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.52	0.33	0.25	0.23	0.21	0.20	0.20	0.20
	HKD	0.98	0.96	0.93	0.92	0.90	0.87	0.86	0.83
KNN=10	Shu *et al.*	0.45	0.24	0.16	0.13	0.11	0.10	0.10	0.10
	HKD	0.98	0.95	0.92	0.91	0.88	0.87	0.83	0.82
KNN=15	Shu *et al.*	0.43	0.21	0.13	0.09	0.08	0.07	0.07	0.07
	HKD	0.97	0.94	0.91	0.90	0.88	0.86	0.81	0.80
KNN=20	Shu *et al.*	0.41	0.20	0.11	0.08	0.06	0.05	0.05	0.05
	HKD	0.97	0.93	0.91	0.90	0.88	0.84	0.80	0.78
Search complexity reduction (%)		49.6	91.0	97.6	98.9	99.1	99.2	99.3	99.3

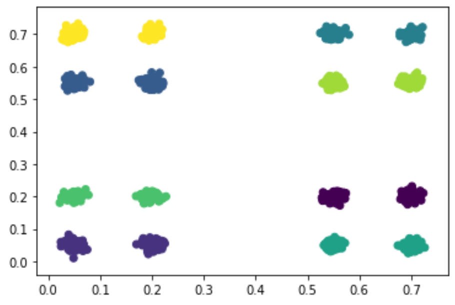
Table 6.9 Nearest Neighbor Preserving Percentage by KD Codes on the Test Set of the CIFAR100 Data

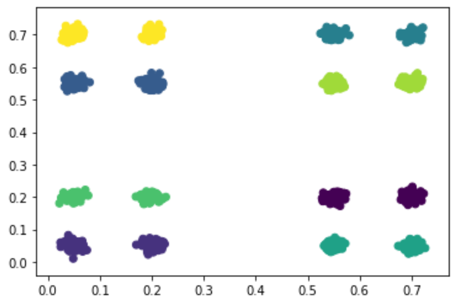
First N layer (s)		1	2	3	4	5	6	7	8
KNN=5	Shu *et al.*	0.72	0.38	0.17	0.11	0.07	0.03	0.02	0.01
	HKD	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99
KNN=10	Shu *et al.*	0.72	0.37	0.16	0.11	0.06	0.03	0.02	0.01
	HKD	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.98
KNN=15	Shu *et al.*	0.72	0.37	0.16	0.11	0.06	0.03	0.02	0.01
	HKD	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.97
KNN=20	Shu *et al.*	0.71	0.37	0.16	0.11	0.06	0.03	0.02	0.01
	HKD	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.96
Search complexity reduction (%)		49.6	91.0	97.6	98.9	99.1	99.2	99.3	99.4

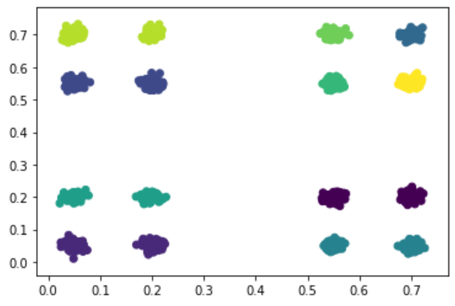

(a) The first code

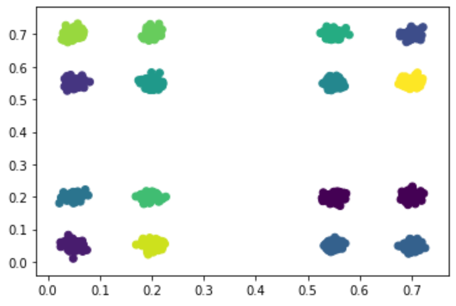

(b) The first two codes


(c) The first three codes


(d) The first four codes


(e) The first five codes


(f) The first six codes


(g) The first seven codes

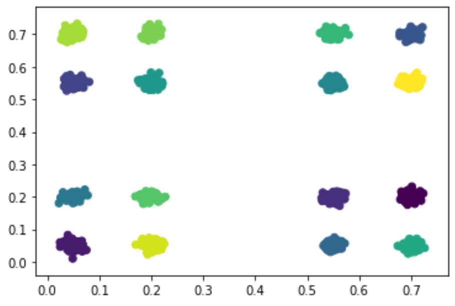
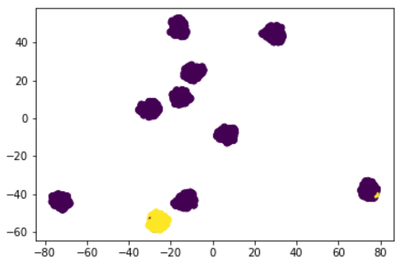
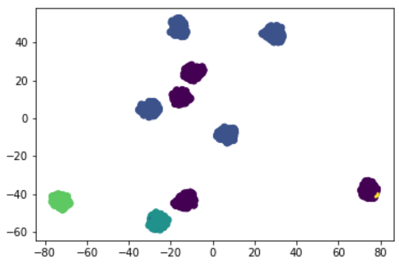

(h) The first eight codes

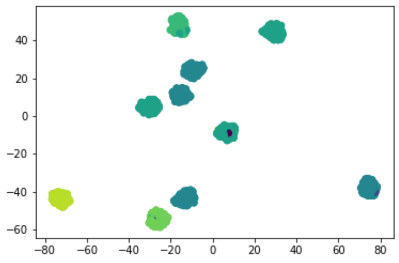
Figure 6.3 The figures to illustrate that the proposed method can hierarchically split the data into clusters in the synthetic dataset.



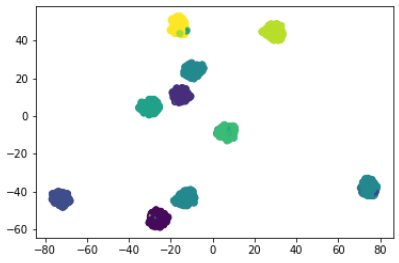
(a) The first code



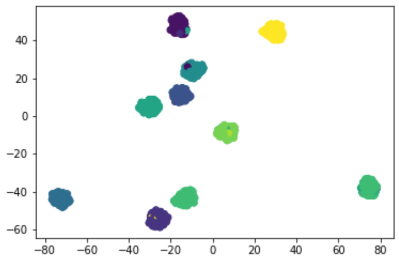
(b) The first two codes



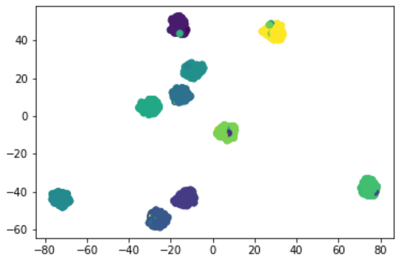
(c) The first three codes



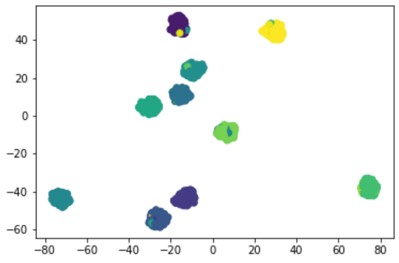
(d) The first four codes



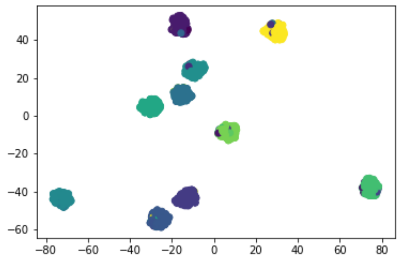
(e) The first five codes



(f) The first six codes



(g) The first seven codes



(h) The first eight codes

Figure 6.4 The figures to illustrate that the proposed method can hierarchically split the data into clusters in the CIFAR100 dataset.

CHAPTER 7

SUMMARY OF THIS DISSERTATION

This dissertation applies deep learning techniques in transportation and has successfully addressed multiple transportation problems with decent performance, including passenger demand prediction, transportation mode detection and traffic light control. Specifically, a deep spatio-temporal fuzzy neural network is proposed to predict the passenger demands considering all the spatial, temporal and external correlations [69]. The proposed model outperforms the state-of-the-art approaches more than 10% in RMSE. A lightweight system based on a convolutional neural network is proposed to detect smartphone users' transportation modes employing the energy-efficient sensor, accelerometer [67, 70]. The proposed system can achieve over 94% accuracy in detecting seven transportation modes. A realtime control system based on deep reinforcement learning is proposed to control traffic lights to efficiently manage vehicles such that less waiting time can be achieved [64]. The proposed system can reduce over 20% waiting time comparing to the starting point and it converges faster than its variants.

The dissertation further extends general embedding systems by designing hierarchical k-way d-dimensional codes to replace one-hot codes to explore the data visualization and retrieval. The proposed codes can reduce the searching space over 90% when searching nearest neighbors [65].

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, March 2016.

[2] Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. Holonic multi-agent system for traffic signals control. *Engineering Applications of Artificial Intelligence*, 26(5):1575–1587, May-June 2013.

[3] Felicity R Allen, Eliathamby Ambikairajah, Nigel H Lovell, and Branko G Celler. An adapted gaussian mixture model approach to accelerometry-based movement classification using time-domain features. In *Proceedings of the 28th Annual International Conference of Engineering in Medicine and Biology Society*, pages 3600–3603, New York, NY, August 2006.

[4] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Ambient Assisted Living and Home Care*, pages 216–223, December 2012.

[5] Itamar Arel, Cong Liu, T Urbanik, and AG Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, June 2010.

[6] PG Balaji, X German, and D Srinivasan. Urban traffic signal control using reinforcement learning agents. *IET Intelligent Transport Systems*, 4(3):177–188, September 2010.

[7] Ling Bao and Stephen S Intille. Activity recognition from user-annotated acceleration data. In *International conference on Pervasive Computing*, pages 1–17, Vienna, Austria, April 2004.

[8] Hetal Bhavsar and Amit Ganatra. A comparative study of training algorithms for supervised machine learning. *International Journal of Soft Computing and Engineering*, 2(4):2231–2307, September 2012.

[9] Chen Cai, Chi Kwong Wong, and Benjamin G Heydecker. Adaptive traffic signal control using approximate dynamic programming. *Transportation Research Part C: Emerging Technologies*, 17(5):456–474, October 2009.

[10] Noe Casas. Deep deterministic policy gradient for urban traffic light control. *arXiv preprint arXiv:1703.09035*, March 2017.

[11] Kit Yan Chan, Tharam S Dillon, Jaipal Singh, and Elizabeth Chang. Neural-network-based models for short-term traffic flow forecasting using a hybrid exponential smoothing and levenberg–marquardt algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):644–654, June 2012.

[12] Tianqi Chen and Carlos Guestrin. Xgboost: a scalable tree boosting system. In *Proceedings of the 22nd sigkdd International Conference on Knowledge Discovery and Data Mining*, pages 785–794, August 2016.

[13] Ting Chen, Martin Renqiang Min, and Yizhou Sun. Learning k-way d-dimensional discrete codes for compact embedding representations. In *Proceedings of the 35th International Conference on Machine Learning*, pages 854–863, Stockholm Sweden, July 2018.

[14] Stephen Chiu and Sujeet Chand. Adaptive traffic signal control using fuzzy logic. In *The 1st Regional Conference on Aerospace Control Systems*, pages 1371–1376, Westlake Village, CA, April 1993.

[15] François Chollet et al. Keras. https://github.com/fchollet/keras, April 2015. Accessed April 30th, 2018.

[16] Stéphane Clinchant and Florent Perronnin. Aggregating continuous word embeddings for information retrieval. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 100–109, Sofia, Bulgaria, August 2013.

[17] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, Helsinki, Finland, July 2008.

[18] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, September 1995.

[19] Thomas M Cover and Peter E Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.

[20] Yue Deng, Zhiquan Ren, Youyong Kong, Feng Bao, and Qionghai Dai. A hierarchical fused fuzzy deep neural network for data classification. *IEEE Transactions on Fuzzy Systems*, 25(4):1006–1012, August 2017.

[21] Jingcheng Du, Peilin Jia, Yulin Dai, Cui Tao, Zhongming Zhao, and Degui Zhi. Gene2vec: distributed representation of genes based on co-expression. *BMC genomics*, 20(1):82, February 2019.

[22] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. Design of reinforcement learning parameters for seamless application of adaptive traffic signal control. *Journal of Intelligent Transportation Systems*, 18(3):227–245, July 2014.

[23] Shih-Hau Fang, Yu-Xaing Fei, Zhezhuang Xu, and Yu Tsao. Learning transportation modes from smartphone sensors based on deep neural network. *IEEE Sensors Journal*, 17(18):6111–6118, September 2017.

[24] Tao Feng and Harry JP Timmermans. Transportation mode recognition using gps and accelerometer data. *Transportation Research Part C: Emerging Technologies*, 37:118–130, December 2013.

[25] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[26] Andrea Frome, Gregory S. Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. Devise: a deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, pages 2121–2129, Lake Tahoe, NV, December 2013.

[27] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, April 1980.

[28] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito, and Norio Shiratori. Adaptive traffic signal control: deep reinforcement learning algorithm with experience replay and target network. *arXiv preprint arXiv:1705.02755*, May 2017.

[29] Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, November 2016.

[30] Andrej Gisbrecht, Alexander Schulz, and Barbara Hammer. Parametric nonlinear dimensionality reduction using kernel t-sne. *Neurocomputing*, 147:71–82, January 2015.

[31] Yoav Goldberg and Omer Levy. Word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, Feburary 2014.

[32] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, November 2016.

[33] Google. Motion sensors. https://developer.android.com/guide/topics/sensors/sensors_motion.html, August 2017. Accessed August 10th, 2017.

[34] Google. Sensor event. https://developer.android.com/reference/android/hardware/SensorEvent.html, August 2017. Accessed August 10th, 2017.

[35] Victor Gradinescu, Cristian Gorgorin, Raluca Diaconescu, Valentin Cristea, and Liviu Iftode. Adaptive traffic lights using car-to-car communication. In *IEEE Vehicular Technology Conference Spring*, April 2007.

[36] David Harris and Sarah Harris. *Digital design and computer architecture*. Morgan Kaufmann, July 2010.

[37] Hannes Hartenstein and LP Laberteaux. A tutorial survey on vehicular ad hoc networks. *Communications Magazine*, 46(6):164–171, June 2008.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the International Conference on Computer Vision*, pages 1026–1034, Las Condes, Chile, December 2015.

[39] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. Accelerometer-based transportation mode detection on smartphones. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, pages 13–27, New York, NY, November 2013.

[40] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in Neural Information Processing Systems*, pages 857–864, British Columbia, Canada, December 2003.

[41] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, pages 278–282, Montreal, Canada, August 1995.

[42] Sepp Hochreiter, Yoshua Bengio, and Paolo Frasconi. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In J. Kolen and S. Kremer, editors, *Field Guide to Dynamical Recurrent Networks*. January 2001.

[43] Arthur E Hoerl and Robert W Kennard. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, Feburary 1970.

[44] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[45] Richard A Hunt. On l (p, q) spaces. *Enseign. Math*, 12(2):249–276, June 1966.

[46] Arash Jahangiri and Hesham A Rakha. Applying machine learning techniques to transportation mode recognition using mobile phone sensor data. *Transactions on Intelligent Transportation Systems*, 16(5):2406–2417, October 2015.

[47] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, November 2016.

[48] Kaleb. Android acceleration sensor: low-pass filter. http://kircherelectronics.com/index.php/2017/12/28/android-acceleration-sensors-low-pass-filter/, December 2017. Accessed April 30th, 2018.

[49] Jim Karki. Active low-pass filter design. *Texas Instruments Application Report*, October 2000.

[50] Jintao Ke, Hongyu Zheng, Hai Yang, and Xiqun Michael Chen. Short-term forecasting of passenger demand under on-demand ride services: a spatio-temporal deep learning approach. *Transportation Research Part C: Emerging Technologies*, 85:591–608, December 2017.

[51] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, Phoenix, AZ, March 2016.

[52] Diederik Kingma and Jimmy Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, December 2014.

[53] Martin Kodyš, Pau Oliver, Joaquim Bellmunt, and Mounir Mokhtari. Human urban mobility classification in aal deployments using mobile devices. In *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, pages 311–319, Salzburg, Austria, November 2017. ACM.

[54] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of sumo-simulation of urban mobility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.

[55] Stefan Krauß. Towards a unified view of microscopic traffic flow theories. *Transportation Systems*, 30(8):901–905, June 1997.

[56] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, April 2009.

[57] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, Lake Tahoe, CA, December 2012.

[58] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, March 2011.

[59] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, January 1997.

[60] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[61] Young-Seol Lee and Sung-Bae Cho. Activity recognition using hierarchical hidden markov models on a smartphone with 3d accelerometer. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 460–467, Berlin, Germany, May 2011.

[62] Robert I Lerman and Shlomo Yitzhaki. A note on the calculation and interpretation of the gini index. *Economics Letters*, 15(3):363–368, January 1984.

[63] Li Li, Yisheng Lv, and Fei-Yue Wang. Traffic signal timing via deep reinforcement learning. *Journal of Automatica Sinica*, 3(3):247–254, July 2016.

[64] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, January 2019.

[65] Xiaoyuan Liang, Martin Renqiang Min, Hongyu Guo, and Guiling Wang. Learning k-way d-dimensional discrete embedding for hierarchical data visualization and retrieval. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, Macau, China, August 2019.

[66] Xiaoyuan Liang, Jie Tian, Xiaoning Ding, and Guiling Wang. Aa risk and similarity aware application recommender system. *CIT. Journal of Computing and Information Technology*, 23(4):303–315, 2015.

[67] Xiaoyuan Liang and Guiling Wang. A convolutional neural network for transportation mode detection based on smartphone platform. In *Proceedings of the IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 338–342, Orlando, FL, October 2017.

[68] Xiaoyuan Liang and Guiling Wang. A light-weight and accurate transit mode detection system based on smartphone platform. Technical report, Transportation Research Board 96th Annual Meeting, January 2017.

[69] Xiaoyuan Liang, Guiling Wang, Martin Renqiang Min, Yi Qi, and Zhu Han. A deep spatio-temporal fuzzy neural network for passenger demand prediction. In *Proceedings of the SIAM International Conference on Data Mining*, pages 100–108, Calgary, Canada, May 2019.

[70] Xiaoyuan Liang, Guiling Wang, and Songhua Xu. A deep learning model for transportation mode detection based on smartphone sensing data. *IEEE Transactions on Intelligent Transportation Systems*. Submitted.

[71] Xiaoyuan Liang, Tan Yan, Joyoung Lee, and Guiling Wang. A distributed intersection management protocol for safety, efficiency, and driver's comfort. *Internet of Things Journal*, 5(3):1924–1935, March 2018.

[72] Weirong Liu, Gaorong Qin, Yun He, and Fei Jiang. Distributed cooperative reinforcement learning-based traffic signal control that integrates v2x networks' dynamic clustering. *IEEE Transactions on Vehicular Technology*, 66(10):8667–8681, October 2017.

[73] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, November 2008.

[74] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: a continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, November 2016.

[75] Vincenzo Manzoni, Diego Maniloff, Kristian Kloeckl, and Carlo Ratti. Transportation mode identification and real-time co2 emission estimation using smartphones. *SENSEable City Lab, Massachusetts Institute of Technology*, November 2010.

[76] Nitin Maslekar, Mounir Boussedjra, Joseph Mouzna, and Houda Labiod. Vanet based adaptive traffic signal control. In *Proceedings of the 73rd Vehicular Technology Conference Spring*, pages 1–5, Budapest, Hungary, May 2011.

[77] Uwe Maurer, Asim Smailagic, Daniel P Siewiorek, and Michael Deisher. Activity recognition and monitoring using multiple sensors on different body positions. In *International Workshop on Wearable and Implantable Body Sensor Networks*, pages 4–pp, Cambridge, MA, April 2006.

[78] Mukesh Kumar Mehlawat and Pankaj Gupta. Fuzzy chance-constrained multi-objective portfolio selection model. *IEEE Transactions on Fuzzy Systems*, 22(3):653–671, June 2014.

[79] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, January 2013.

[80] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, November 1995.

[81] Martin Renqiang Min, Hongyu Guo, and Dinghan Shen. Parametric t-distributed stochastic exemplar-centered embedding. In *Proceedings of the 2018 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 477–493, Dublin, Ireland, September 2018.

[82] Martin Renqiang Min, Hongyu Guo, and Dongjin Song. Exemplar-centered supervised shallow parametric data embedding. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2479–2485, Melbourne, Australia.

[83] Martin Renqiang Min, Laurens van der Maaten, Zineng Yuan, Anthony Bonner, and Zhaolei Zhang. Deep supervised t-distributed embedding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 791–798, Haifa, Israel, June 2010.

[84] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[85] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402, September 2013.

[86] Seyed Sajad Mousavi, Michael Schukat, Peter Corcoran, and Enda Howley. Traffic light control using deep policy-gradient and value-function based reinforcement learning. *arXiv preprint arXiv:1704.08883*, April 2017.

[87] Ben Nham, Kanya Siangliulue, and Serena Yeung. Predicting mode of transport from iphone accelerometer data. *Machine Learning Final Projects*, December 2008.

[88] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pages 6338–6347, Long Beach, CA, January 2017.

[89] Albert H Nuttall. Some windows with very good sidelobe behavior. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 29(1):84–91, Feburary 1981.

[90] Kartik Pandit, Dipak Ghosal, H Michael Zhang, and Chen-Nee Chuah. Adaptive traffic signal control with vehicular ad hoc networks. *IEEE Transactions on Vehicular Technology*, 62(4):1459–1471, May 2013.

[91] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, Long Beach, CA, December 2017.

[92] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar, October 2014.

[93] Thair Nu Phyu. Survey of classification techniques in data mining. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 18–20, March 2009.

[94] LA Prashanth and Shalabh Bhatnagar. Threshold tuning using stochastic optimization for graded signal control. *IEEE Transactions on Vehicular Technology*, 61(9):3865–3880, November 2012.

[95] Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 233–242, January 2002.

[96] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks*, 6(2):13, Feburary 2010.

[97] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, September 2016.

[98] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, October 1988.

[99] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, July 1964.

[100] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, November 2015.

[101] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, August 2015.

[102] Shashank Shekhar and Billy Williams. Adaptive seasonal time series models for forecasting short-term traffic flow. *Transportation Research Record: Journal of the Transportation Research Board*, 2024(1):116–125, December 2008.

[103] Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. Baseline needs more love: on simple word-embedding-based models and associated pooling mechanisms. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 440–450, Melbourne, Australia, July 2018.

[104] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: a machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, Montral, Canada, December 2015.

[105] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul JM Havinga. Fusion of smartphone motion sensors for physical activity recognition. *Sensors*, 14(6):10146–10176, June 2014.

[106] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. In *Proceedings of the 6th International Conference on Learning Representations*, Vancouver, Canada, April 2018.

[107] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda

Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[108] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, October 2017.

[109] Xuan Song, Hiroshi Kanasugi, and Ryosuke Shibasaki. Deeptransport: prediction and simulation of human mobility and transportation mode at a citywide level. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 2618–2624, July 2016.

[110] Leon Stenneth, Ouri Wolfson, Philip S Yu, and Bo Xu. Transportation mode detection using mobile phones and gis information. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63, November 2011.

[111] Xing Su, Hernan Caceres, Hanghang Tong, and Qing He. Online travel mode identification using smartphones with battery saving considerations. *Transactions on Intelligent Transportation Systems*, 17(10):2921–2934, October 2016.

[112] Richard S Sutton and Andrew G Barto. *Reinforcement learning: an introduction*, volume 1. MIT press Cambridge, March 1998.

[113] Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. Hash embeddings for efficient word representations. In *Advances in Neural Information Processing Systems*, pages 4928–4936, Long Beach, CA, December 2017.

[114] Vladimir Svetnik, Andy Liaw, Christopher Tong, J Christopher Culberson, Robert P Sheridan, and Bradley P Feuston. Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of Chemical Information and Computer Sciences*, 43(6):1947–1958, November 2003.

[115] Fengxiao Tang, Bomin Mao, Zubair Md Fadlullah, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control. *IEEE Wireless Communications*, 25(1):154–160, February 2018.

[116] Jie Tian, Xiaoyuan Liang, and Guiling Wang. Deployment and reallocation in mobile survivability-heterogeneous wireless sensor networks for barrier coverage. *Ad Hoc Networks*, 36:321–331, January 2016.

[117] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, October 2012.

[118] Weather Underground. Historical weather. https://www.wunderground.com/history/, April 2018. Accessed April 30th, 2018.

[119] Laurens Van Der Maaten. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342*, January 2013.

[120] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, January 2014.

[121] Elise van der Pol. Deep reinforcement learning for coordination in traffic light control. Master's thesis, University of Amsterdam, August 2016.

[122] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100, Phoenix, AZ, February 2016.

[123] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems*, pages 5998–6008, Long Beach, CA, December 2017.

[124] Toan H Vu, Le Dung, and Jia-Ching Wang. Transportation mode detection on mobile devices using recurrent nets. In *Proceedings of the ACM on Multimedia Conference*, pages 392–396, October 2016.

[125] Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, pages 351–359, Lake Tahoe, NV, December 2013.

[126] Hao Wang, GaoJun Liu, Jianyong Duan, and Lei Zhang. Detecting transportation modes using deep neural network. *IEICE Transactions on Information and Systems*, 100(5):1132–1135, May 2017.

[127] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, November 2015.

[128] Peter Widhalm, Philippe Nitsche, and Norbert Brändie. Transport mode detection with realistic smartphone sensor data. In *21st International Conference on Pattern Recognition (ICPR)*, pages 573–576, Tsukuba, Japan, November 2012.

[129] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: practical machine learning tools and techniques*. Morgan Kaufmann, October 2016.

[130] Jun Yang. Toward physical activity diary: motion recognition using simple acceleration features with mobile phones. In *Proceedings of the 1st International Workshop on Interactive Multimedia for Consumer Electronics*, pages 1–10, Beijing, China, October 2009.

[131] Zhirong Yang, Jaakko Peltonen, and Samuel Kaski. Scalable optimization of neighbor embedding for visualization. In *International Conference on Machine Learning*, pages 127–135, Atlanta, GA, June 2013.

[132] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, and Jieping Ye. Deep multi-view spatial-temporal network for taxi demand prediction. *arXiv preprint arXiv:1802.08714*, February 2018.

[133] Rose Yu, Yaguang Li, Cyrus Shahabi, Ugur Demiryurek, and Yan Liu. Deep learning: a generic approach for extreme condition traffic forecasting. In *Proceedings of the SIAM International Conference on Data Mining*, pages 777–785, Houston, TX, June 2017.

[134] Lotfi A Zadeh. Fuzzy sets. In *Fuzzy Sets, Fuzzy Logic, And Fuzzy Systems: Selected Papers by Lotfi A Zadeh*, pages 394–432. World Scientific, May 1996.

[135] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, May 2016.

[136] Junbo Zhang, Yu Zheng, and Dekang Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *31st AAAI Conference on Artificial Intelligence*, pages 1655–1661, San Francisco, CA, Feburary 2017.

[137] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, Montral, Canada, December 2015.

[138] Zheng Zhao, Weihai Chen, Xingming Wu, Peter CY Chen, and Jingmeng Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, January 2017.

[139] Yu-Jun Zheng, Wei-Guo Sheng, Xing-Ming Sun, and Sheng-Yong Chen. Airline passenger profiling based on fuzzy deep machine learning. *IEEE Transactions on Neural Networks and Learning Systems*, 28(12):2911–2923, December 2017.

[140] Binbin Zhou, Jiannong Cao, Xiaoqin Zeng, and Hejun Wu. Adaptive traffic light control in wireless sensor network-based intelligent transportation system. In *IEEE Vehicular Technology Conference Fall*, September 2010.

[141] Xian Zhou, Yanyan Shen, Yanmin Zhu, and Linpeng Huang. Predicting multi-step citywide passenger demands using attention-based neural networks. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, pages 736–744, Marina Del Rey, CA, February 2018.

[142] Li Zhu, Ying He, F Richard Yu, Bin Ning, Tao Tang, and Nan Zhao. Communication-based train control system performance optimization using deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 66(12):10705–10717, 2017.