Complex Sequential Modeling

Guojun Wu

PhD Dissertation in Data Science

Worcester Polytechnic Institute, Worcester, MA

Nov 2021

Committee Members:
Dr. Yanhua Li, Assistant Professor, WPI. **Advisor**
Dr. Ziming Zhang, Assistant Professor, WPI. **Co-Advisor**
Dr. Oren Mangoubi, Assistant Professor, WPI.
Dr. Zhi-Li Zhang, Professor, University of Minnesota. **External Member**

Copyright © 2021 by Guojun Wu. This dissertation proposal is an internal Worcester Polytechnic Institute document that contains unpublished material. This document and its content is thus protected by copyright. To make digital or hard copies of all or part of this work, to use in research, educational or commercial programs, to post on servers or to redistribute to lists, requires prior specific permission from the author.

Abstract

Traditional sequential models such as Recurrent neural networks (RNNs) have achieved significant success in learning complex patterns for sequential input data. At each time step t, an RNN stores the previous hidden state vector, $\mathbf{h}_{t-1} \in \mathbb{R}^D$, and upon receiving the current input vector, $\mathbf{x}_t \in \mathbb{R}^d$, linearly transforms the tuple $(\mathbf{h}_{t-1}, \mathbf{x}_t)$ and passes it through a non-linearity to update the state vectors over T time steps. Subsequently, RNNs output the predictions as a function of the hidden states. The model parameters (*i.e.,* state/input/prediction parameters) are learned by minimizing an empirical loss. However, for real world applications involving sequential data like sensor data and human behavior data, plain RNNs usually tends to produce poor results. First, vanishing and exploding gradients often occur in training multi-layer RNNs which is widely used to solve problems with complex patterns. Secondly, recurrent neural networks can not model the problems which current state is related to future expectations, such as problems involving human decision-making process. The main theme of my dissertation is to design, develop and evaluate efficient models to mitigate those issues. I focus on designing RNN models with theoretically guaranteed training stability and applying inverse reinforcement learning model to human decision problems in my dissertation. The dissertation contains following five research themes.

1) Human Decision Modeling with Sequential Human Decision Data. In this paper, I make the first attempt to model passengers' preferences of making various transit choices using Markov Decision Process (MDP). Moreover, we develop a novel inverse preference learning algorithm to infer the passengers' preferences and predict the future human behavior changes, e.g., ridership, of a new urban transit plan before its deployment. I validate our proposed framework using a unique real-world dataset (from Shenzhen, China) with three subway lines opened during the data timespan. With the data collected from both before and after the transit plan deployments, Our evaluation results demonstrated that the proposed framework can predict the ridership with only 19.8% relative error, which is 23%-51% lower than other baseline approaches.

2) Altering Human Decision-Making Process via Reward Advancement. Many real world human behaviors can be characterized as sequential decision making processes, such as urban travelers' choices of transport modes and routes [173]. Differing from choices controlled by machines, which in general follows *perfect rationality* to adopt the policy with highest reward, studies have revealed that human agents make sub-optimal decisions under *bounded rationality* [156]. Such behaviors can be modeled using maximum causal entropy (MCE) principle [206].In this work, I define and investigate a novel reward transformation problem (namely, *reward advancement*): Recovering the range of additional reward functions that transform the agent's policy from π_o to a predefined target policy π_t under MCE principle. I show that given an MDP and a target policy π_t, there are infinite many additional reward functions that can achieve the desired policy transformation. Moreover, I propose an algorithm to further extract the additional rewards with minimum "cost" to implement the policy transformation. I demonstrated the correctness and accuracy of our reward advancement solution using both synthetic data and a large-scale (6 months) passenger-level public transit data from Shenzhen, China.

3) Using Sequential Models in General Human Prediction Problems.In this work, I aim to develop a joint framework of combining inverse reinforcement learning (IRL) with deep learning (DL) regression model, called IRL-DL, to predict drivers' future behavior in ride-hailing platforms. Specifically, I formulate the dynamic evolution of each driver as a sequential decision-making problem and then employ IRL as representation learning to learn the preference vector of each driver. Then, I integrate drivers' preference vector with their static features (e.g., age, gender) and other attributes to build a regression model (e.g., LTSM-neural network) to predict drivers' future behavior. I use an extensive driver data set obtained from a ride-sharing platform to verify the effectiveness and efficiency of our IRL-DL framework, and results show that our IRL-DL framework can achieve consistent and remarkable improvements over models without drivers' preference vectors.

4) Training Stability in Learning Recurrent Models.In this work, I study the training stability in deep recurrent neural networks (RNNs), and propose a novel network, namely, deep incremental RNN (DIRNN). In contrast to the literature, I prove that DIRNN is essentially a Lyapunov

stable dynamical system where there is no vanishing or exploding gradient in training. To demonstrate the applicability in practice, I also propose a novel implementation, namely TinyRNN, that sparsifies the transition matrices in DIRNN using weighted random permutations to reduce the model sizes.

5) Lightweight Convolutional Neural Network via Recurrent Convolution. In this work, we aim to address the problem of learning lightweight networks by proposing a novel CSR-Conv layer that replaces traditional linear convolution with channel-split recurrent convolution. The hidden state transition in the vanilla RNNs leads to deeper networks, given backbones, to compensate for the performance loss while reducing the model sizes. Essentially our CSR-Conv can be viewed as the generalization of linear convolution. We show that the model size of a lightweight network decreases *w.r.t.* the number of the duplicate networks with the rate of $O(\frac{1}{T^2})$.

Acknowledgments

One of the very first words my advisor, Prof. Yanhua Li said to me even befor I started my PhD is, "PhD is more like a journey". Now, almost at the end of this journey, I would like to sincerely thank him for his great help and support. Prof. Li shows me the path to find out *what is research*, *how to do research* and most importantly, *why doing research*. I would thank him for his priceless comments, suggestions on both my work and my life. Especially, I'm grateful for the time and energy he spent when we were in different time zones. This five year definitely would be one of the most memorable, invaluable journey in my life.

I also would like to express my sincere appreciation to my Co-advisor, Prof. Ziming Zhang for his guidance and support along this journey. I would thank him for his wise advice and idea of our work and his tolerance of my occasional weird working schedule during the COVID-19 quarantine.

Many special thanks goes to my committee members, Prof. Oren Mangoubi and Prof. Zhi-Li Zhang for their kindness and thoughtful feedbacks on my works and this very dissertation.

I would like to thank my current and fellow graduate students at WPI, Menghai Pan, Huimin Ren, Xin Zhang, Yingxue Zhang and the whole DSRG group for those discussion we had online and offline. Without those discussions, some of my work might be totally different. And I would never forget the "founding members" of AK123, Matt Weiss, Jianjun Luo, Dr. Chong Zhou, Haitao Liu, Guanxiong Liu and Menghai Pan. Those days and nights at lab are truly invaluable to me.

I also owe my gratitude to all my mentors and colleagues during my internship at Didi. In particular, Prof. Hongtu Zhu and Dr. Shikai Luo help me with both their excellent statistical expertise and amazing personality. I would also like to thank Ge Song, Qichao Wang, Jia Zheng and Shixiang Wan for their brilliant ideas on and off work.

Thanks all my friends in both USA and China, I would never be who I am without you guys. I would like to thank Yuan for those delicious restaurant she took me to. Many special thanks go to Chunxing, and his fiancee, Ruiyang for everything and especially the Michelin-level Canelé they made throughout these years. Also, Liang and Xinyu, I can't even start my PhD without your encouragement.

On of the most special appreciation should go to Mr. Huang, my NOIP coach at high school. He showed what an interesting thing programming and computer science are.

Last but not least, I would like thank my family for their nonstop love and support. I love you from the bottom of my heart.

Contents

List of Figures

List of Tables

List of Publications during my Ph.D. Studies at WPI

1. **G. Wu**, Y. Ding, Y. Li, J. Bao, Y. Zheng, and J. Luo. Mining spatio-temporal reach-able regions over massive trajectory data. In 2017 IEEE 33rd International Confer-ence on Data Engineering (ICDE), pages 1283–1294. IEEE, 2017 [172]

2. **G. Wu**, Y. Ding, Y. Li, J. Luo, F. Zhang, and J. Fu. Data-driven inverse learning of passenger preferences in urban public transits. In 56th IEEE Conference on Decisionand Control, 2017 [173]

3. **G. Wu**, Y. Li, J. Bao, Y. Zheng, J. Ye, and J. Luo. Human-centric urban transit evaluation and planning. In IEEE International Conference on Data Mining, 2018 [174]

4. **G. Wu**, Y. Li, and J. Luo. Transforming policy via reward advancement. In 2019 IEEE 58th Conference on Decision and Control (CDC), pages 4609–4614. IEEE,2019 [176]

5. **G. Wu**, Y. Li, S. Luo, G. Song, Q. Wang, J. He, J. Ye, X. Qie, and H. Zhu. A joint inverse reinforcement learning and deep learning model for drivers' behavioral pre-diction. In Proceedings of the 29th ACM International Conference on Information& Knowledge Management, pages 2805–2812, 2020 [177]

6. J. Urata, Z. Xu, J. Ke, Y. Yin, **G. Wu**, H. Yang, J. Ye. Learning ride-sourcing drivers' customer-searching behavior: A dynamic discrete choice approach. Transportation Research Part C: Emerging Technologies. 2021 Sep 1;130:103293. [160]

7. Z. Zhang*, **G. Wu***, Y. Yue, Y. Li, and X. Zhou. Deep Incremental RNN for Learning Sequential Data: A Lyapunov Stable Dynamical System. In IEEE International Conference on Data Mining, 2021 [195]

8. Z. Zhang, Y. Yue, **G. Wu**, Y. Li and H. Zhang. SBO-RNN: Reformulating Recurrent Neural Networks via Stochastic Bilevel Optimization. Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS), 2021. [196]

Chapter 1

Introduction

Sequential data is a fundamental type of data that represents and ordered value sequence. Sequential data can be generated and consumed by plenty of different application such as audio data, DNA data and financial data. To process sequential data, recurrent models like RNN, GRU and LSTM are often used. However, traditional recurrent model can not handle problems involving human decisions. For example, in the urban transit problems, though transit choices of one passenger could form a sequence, each choice is made based on not only past decisions but also expected future situations such as expected travel time of different transit modes. So, during my PhD study, I first took a look into how to use complex sequential model to build a model reflecting human decision making process and applied it to real world scenarios. These works become the first part of my dissertation. Another problem arising from building complex sequential models for human behavior data other than modeling human decision making process is that the training of sequential models such as RNN or LSTM are extremely unstable. So in the second part, I made some effort to proposed a stacked recurrent neural network with guaranteed training stability. Also, during my PhD study, I focused on how to apply sequential models to a more general context, for example, image processing. In this work, I develop a recurrent-unit

based convolution layer to build a small yet effective convolution layer.

1.1 Human Decision Modeling with Sequential Human Decision Data.

In the first work, we use urban transit system as an example of data involving human decisions to demonstrate how to model those decisions. Urban public transit planning is crucial in reducing traffic congestion and enabling green transportation. However, there is no systematic way to integrate passengers' personal preferences in planning public transit routes and schedules so as to achieve high occupancy rates and efficiency gain of ride-sharing. In this work, we take the first step to exact passengers' preferences in planning from history public transit data. we propose a data-driven method to construct a Markov decision process model that characterizes the process of passengers making sequential public transit choices, in bus routes, subway lines, and transfer stops/stations. Moreover, we develop a novel inverse preference learning algorithm to infer the passengers' preferences and predict the future human behavior changes, e.g., ridership, of a new urban transit plan before its deployment. we validate the proposed framework using a unique real-world dataset (from Shenzhen, China) with three subway lines opened during the data time-span.

1.2 Altering Human Decision Process via Reward Advancement

Different from general reinforcement learning problems, decisions made by human agent can not be easily changed. Instead, human agents complete tasks by evaluating the re-

wards received over states traversed and actions employed. Many real world human behaviors can be characterized as sequential decision making processes, such as urban travelers' choices of transport modes and routes [173]. Differing from choices controlled by machines, which in general follows *perfect rationality* to adopt the policy with highest reward, studies have revealed that human agents make sub-optimal decisions under *bounded rationality* [156]. Such behaviors can be modeled using maximum causal entropy (MCE) principle [206]. In this work, we define and investigate a novel reward transformation problem (namely, *reward advancement*) to recover the range of additional reward functions that transform the agent's policy from π_o to a predefined target policy π_t under MCE principle.

1.3 Use of Sequential Models in General User Prediction Problems

Based on the model developed in previous works, one can easily model human decision process from sequential decision data. Naturally, the next question is, how to use those decision models to provide insights while solving more general problems. In the third work, we focus on fusing human decision modeling with general user behavior prediction problems. Users' behavioral predictions are crucially important for many domains including major e-commerce companies, ride-hailing platforms, social networking, and education. The success of such prediction strongly depends on the development of representation learning that can effectively model the dynamic evolution of user's behavior. This work aims to develop a joint framework of combining inverse reinforcement learning (IRL) with deep learning (DL) regression model, called IRL-DL, to predict drivers' future behavior in ride-hailing platforms. Specifically, we formulate the dynamic evolution of each driver as a sequential decision-making problem and then employ IRL as represen-

tation learning to learn the preference vector of each driver. Then, we integrate drivers' preference vector with their static features (e.g., age, gender) and other attributes to build a regression model to predict drivers' future behavior.

1.4 Training Stability in Learning Recurrent Models

For many real world problems like human behavioral predictions, we need complex recurrent models such as stacked recurrent units. However, there is no guarantee that the training of stacked recurrent neural network is stable. Usually, some tricks like gradient clips are used to mitigate the problem. In this work, we study the training stability in deep recurrent neural networks (RNNs), and propose a novel network, namely, deep incremental RNN (DIRNN). In contrast to the literature, we prove that DIRNN is essentially a Lyapunov stable dynamical system where there is no vanishing or exploding gradient in training. To demonstrate the applicability in practice, we also propose a novel implementation, namely TinyRNN, that sparsifies the transition matrices in DIRNN using weighted random permutations to reduce the model sizes.

1.5 Lightweight Convolutional Neural Network via Recurrent Convolution

Lightweight neural networks refer to deep networks with small numbers of parameters, which are allowed to be implemented in resource-limited hardware such as embedded systems. To learn such lightweight networks effectively and efficiently, in this work we propose a novel convolutional layer, namely *Channel-Split Recurrent Convolution (CSR-Conv)*, where we split the output channels to generate data sequences with length T as the input to the recurrent layers with shared weights. As a consequence, we can construct

lightweight convolutional networks by simply replacing (some) linear convolutional layers with CSR-Conv layers. We prove that under mild conditions the model size decreases with the rate of $O(\frac{1}{T^2})$. Empirically we demonstrate the state-of-the-art performance using VGG-16, ResNet-50, ResNet-56, ResNet-110, DenseNet-40, MobileNet, and EfficientNet as backbone networks on CIFAR-10 and ImageNet.

1.6 Outline of Dissertation

The dissertation is organized as follows:

Chapter 2 elaborates how to model human decision process based on sequential decision data collected from urban transit system and its application on urban transit planning.

Chapter 3 develops a novel reward advancement algorithm which can guide human agent's policy to a predefined desired policy.

Chapter 4 shows potential applications of human decision modeling on general behavioral prediction problems and implements a fused model involving human decision features and historical behaviors.

Chapter 5 introduces the stability issues in training stacked recurrent neural networks and proposes an idea to solve this problem.

Chapter 6 demonstrates a method that can construct lightweight CNN models with recurrent convolutions.

Chapter 2

Human Decision Modeling with Sequential Human Decision Data

2.1 Introduction

With the fast pace of global urbanization, the growth of urban population has already significantly worsen the urban traffic congestion problem [166]. By aggregating the urban trip demands with shared trains and buses, public transits offer affordable ride-sharing services and reduce the road network traffic, which in turn mitigate the traffic congestion problem. Urban public transits, such as city buses and subway passenger trains, are group travel systems, deployed for general public and operated on *established routes* and *fixed schedules*. The goal of developing new public transit plans, e.g., a new subway line or a new bus route/schedule, is to precisely meet the needs from passengers in urban areas, and attract as many passengers as possible, to take the new transit lines, from other transit modes, such as private cars. To achieve such a design goal, urban transit planners primarily conduct surveys to collect trip demand data in urban areas, and develop transit plans that cover the most trip demands. However, survey data are usually sparse

and biased. Moreover, without considering the passengers' preferences (in choosing the transit modes and routes), a transit plan may lead to unexpected (too large or too small) ridership, after deploying it.

Taking Shenzhen, China as an example, there were three subway lines with 63 subway stations opened in 2016. Fig 2.1 shows the dynamic crowd flows of four subway stations along the new subway lines, which count the total number of passengers going into subway stations within every half a hour. XiaSha, Baguoling, and ShangSha stations have clear diurnal patterns, with overall high crowd flow. However, at ShenWanZhan Station of Line #9, the average crowd flow is about 20 passengers over the day. Meanwhile, from the taxi trip data, we observe numerous trips from ShenWanZhan region, before and after Line #9 was opened. This indicates that opening the ShenWanZhan subway station did not successfully attract travelers in that region, since their travel modes did not change. With such a low ridership, it was too costly to open the ShenWanZhan subway station. As a result, it is non-trivial to evaluate the impacts of a transit deployment plan on future passenger behaviors prior to actual deployment, since the travel preferences of passengers along the planned routes may vary.

In this chapter, we make the first attempt to investigate how to characterize the passenger preferences from public transit trajectory data, and develop a novel approach to predict the human behavior changes, e.g., ridership, of an urban transit plan before its deployment. Our contributions are summarized as follows.

- First, we model the travelers' trips using Markov Decision Process (MDP) model, where we consider a traveler, as an "agent", completing a trip from origin to destination by making a sequence of decisions about transport modes and routes. Moreover, from real world data, We extract various decision-making features, that passengers evaluate when making transit choices, such as travel time, cost, level-of-convenience.

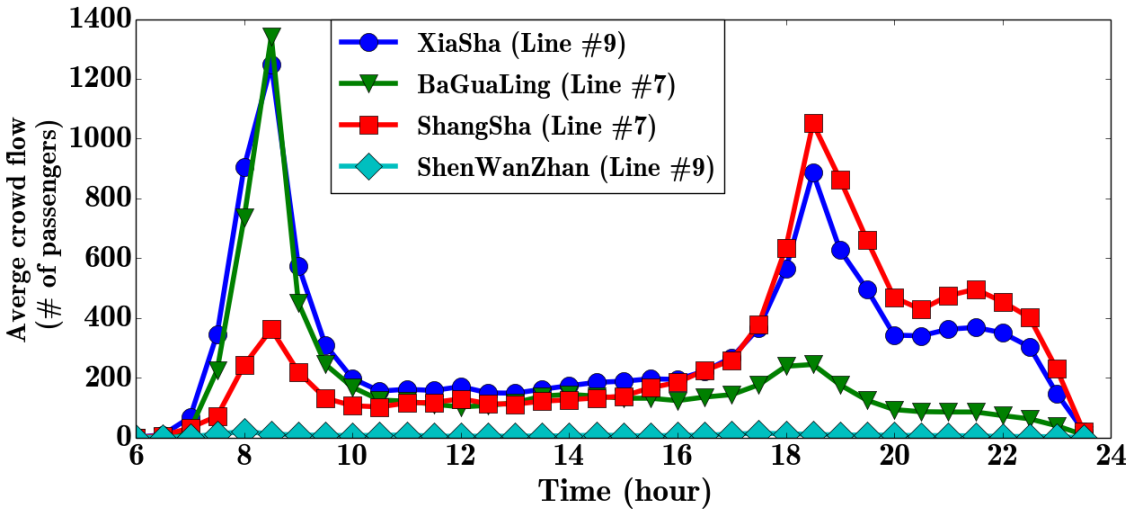


Figure 2.1: Crowd flows of four new subway stations in Shenzhen, China.

- Second, we develop a novel inverse learning algorithm to recover the preference function of passengers from their historical transit trajectories. With the passenger reward functions, we develop transit plan evaluation framework to estimate the future human behaviors, i.e., ridership, crowd flow, after a transit plan is deployed.

- We validate our framework using a unique dataset from Shenzhen, China, with three subway lines opened during the data timespan. This allows us to examine our transit plan evaluation framework, with both data collected before and after the plan is deployed. Our results demonstrated that our proposed framework predicts the ridership with only 19.8% relative error from the ground-truth, which is 23%-51% lower than baseline approaches. *We will make our unique dataset available to contribute to the research community.*

The rest of the chapter is organized as follows. Sec 2.2 defines the problem, describes the data, and outlines our transit evaluation framework. Sec 2.3–2.5 introduce our proposed methodology on data-preprocessing, data-driven modeling, and transit plan evaluation. Sec 2.6 presents evaluation results using a large-scale urban transit trajectory dataset.

2.2 Overview

In this section, we define the transit plan evaluation problem for urban transit system, describe datasets we use, and outline the framework of our methodology.

2.2.1 Problem Definition

In a city, its urban public transit system, including buses and subway lines, naturally forms a directed graph, for which we refer to as *public transit graph* (in short, *transit graph*) defined as follows.

Definition 2.2.1 (Transit Graph) *A transit graph $G = (V, E)$ in a city represents the connections of public transit stops by the transit lines. Vertex set V is a set of transit nodes, i.e., locations of all bus stops and subway stations, and E as the set of transit edges consists of all the bus routes and subway lines between transit stops/stations.*

Passengers in an urban area are completing trip demands over time, e.g., commute trips between home and working place, where we define an urban trip demand as follows.

Each passenger generates a transit trajectory, when taking the public transit system to complete a trip.

Definition 2.2.2 (Trip Demand) *A trip demand td of a passenger indicates the intent of a passenger to travel from a source location src to a destination location dst from a given starting time t, which can be represented as a triple $td = \langle src, dst, t \rangle$.*

Passenger trip demands can be obtained from various data sources. For example, the transaction data from AFC devices in buses and subway systems record passenger trip demands at the level of bus stops and subway stations. Taxi GPS trajectory data with occupation information include the trip demands for taxi trips. To complete a trip demand

by urban public transit, the passenger generates a transit trajectory, when traversing the transit graph.

Definition 2.2.3 (Transit Trajectory) *A transit trajectory tr of a passenger trip demand $\langle src, dst, t_1 \rangle$ is a sequence of spatio-temporal points, that the passenger traverses in the transit graph. Each spatio-temporal point consists of a transit node and transit edge, with a time stamp, i.e., $\ell_i = (v_i, e_i, t_i)$, where $v_i \in V$ and $e_i \in E$.*

Clearly, a spatio-temporal point $\ell = (v, e, t)$ indicates that the passenger takes a transit line $e \in E$, e.g., bus route #3, from a transit stop/station $v \in V$ at time t. With the graph representation of the urban public transit system, a new transit plan, e.g., a new subway line or a new bus route, can be represented as a graph, with a set of new transit nodes, connected by the new transit edges.

Definition 2.2.4 (Transit Plan) *A transit plan $\Delta G = (\Delta V, \Delta E)$ consists of a set of new transit nodes ΔV, i.e., new bus stops and subway stations to be built, and a set of new transit edges ΔE, i.e., new bus routes and subway lines.*

Combining the transit plan graph ΔG and the existing transit graph G yields a new transit graph $G' = (V', E') = (V \cup \Delta V, E \cup \Delta E)$, which represents the urban transit connections available for the passengers, if the transit plan ΔG is deployed.

Problem Definition. Given trip demands $TD = \{td\}$ of a city, transit trajectories $TR = \{tr\}$ of passengers, existing transit graph G, and a transit plan ΔG, our goal is to evaluate/predict the ridership and crowd flow at new transit nodes $v \in \Delta V$ under G', i.e., assuming the transit plan is deployed.

2.2.2 Data Description

In recent years, the fast development of sensing technologies and data collection infrastructure in urban transit systems has led to a large amount of sensing data collected from

Figure 2.2: AFC on bus

Figure 2.3: AFC on subway

Figure 2.4: Three new subway lines opened in 2016

various transit modes in real time, through Automated Fare Collection (AFC) devices on buses, subway trains, and taxis, and GPS sets on vehicles. In this work, three sets of data are available, including (1) public transit trajectory data, (2) taxi trip demand data and (3) transit graph and road map data. Note that for consistency, we choose these datasets aligned within the same time period, i.e., June to December in 2016. Below, we describe these datasets in details, and highlight the unique advantage of our data for this study.

Public transit trajectory data. In Shenzhen, all buses and subway stations are equipped with automatic fare collection (AFC) systems (See Fig 2.2 and Fig 2.3), where passengers swipe their smart cards at AFC devices to get aboard a bus or entering/leaving a subway station. We collected 7 months of passenger transaction data from buses and subway stations. Each transaction record contains five attributes including passenger ID, transaction type, cost, transaction time, transit station/stop name and location. The transaction type field indicates if it is an event to get on a bus, or leaving/entering a subway station. Note that most buses in Shenzhen only record the events of passengers getting on buses, but not getting off buses. We employ the approach in [184] to recover the transit stops where the passengers get off the buses. These transaction data allow us to extract the trip demands and transit trajectories of passengers taking public transits.

Taxi trip demand data. We collected a large-scale taxi trajectory dataset in Shenzhen. These trajectories represent $21,385$ unique taxis in Shenzhen. They are equipped with GPS sets, which periodically (i.e., roughly every 30 seconds) generate $1,797,131,540$ GPS records in total. Each record has five core attributes including unique plate ID, longitude, latitude, time and passenger indicator. The passenger indicator field is a binary value for taxi data, indicating whether a passenger is on board (with value 1) or not. Hence, a sequence of taxi GPS points with passenger indicator as 1 represent a taxi trip, and the first and last GPS points of the sequence are the source and destination locations (i.e., src and dst) of a trip demand. The time stamp of the starting GPS point is the trip starting time t.

Transit Graph and Road Map Data. In this chapter, we retrieve a bounding box of Shenzhen city through the Google Geocoding API [47]. The bounding box was defined by latitude from $22.42°$ to $22.81°$ while longitude from $113.75°$ to $114.68°$. It covers an urban area of about 400 square miles and three million people. Within this bounding box, we obtain Shenzhen transit graph, i.e., all 892 bus routes and 8 subway lines, and road

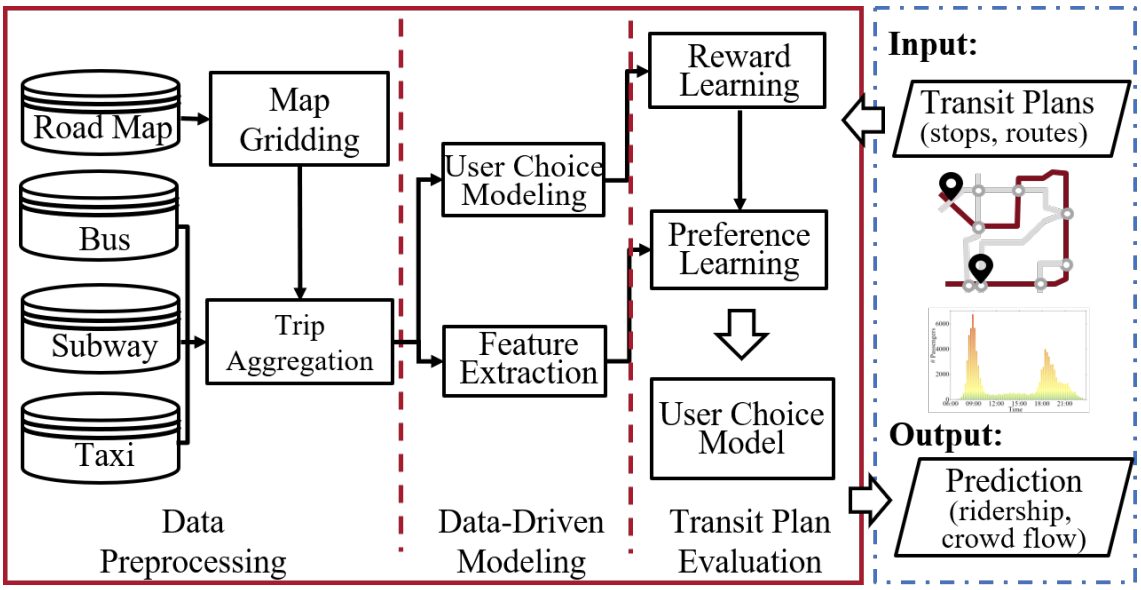


Figure 2.5: Framework of qualifying transit plans system

map data from OpenStreetMap [123].

Unique advantage of our data. Within the six months timespan of our data, three new subway lines were opened (See Fig 2.4). As a result, the transit trajectory data include the passenger behaviors both before and after the subway line deployment. Such data are thus perfect for our study: We can take the three new subway lines as transit plans; predict the human behaviors (i.e., ridership) of these plans using data prior to the deployment; and validate the prediction accuracy using the data after deployment.

2.2.3 Solution Framework

Fig 2.5 provides an overview of our proposed framework, which consists of three main components: Data Preparation, MDP Construction, and Reward Function Evaluation.

- **Stage 1: Data Preprocessing.**

  The urban areas are divided into n equal size grids. As a result, all the transit nodes (bus stops, subway stations), trip demands, and public transit trajectories are

aggregated into the grid level.

- **Stage 2: Data-Driven Modeling.** In this component, we model the travelers' trips using Markov Decision Process (MDP) model, which is reasoned as follows. Each traveler, as an "agent", completes a trip from origin to destination by making a sequence of decisions about transport modes and routes. Inherently, each passenger evaluates various decision-making features associated with the current state and each possible decision, such as travel time, cost, level-of-convenience, by the traveler's reward function. The reward function represents the preference the traveler has over different decision-making features. Each traveler is making their decisions that maximize the total "reward" she obtains out of the trip.

- **Stage 3: Transit Plan Evaluation.** In this component, we first develop a novel inverse learning algorithm to recover the preference function of passengers from their historical transit trajectories. After the transit plan is deployed, the passengers' preferences are invariant. With the extract passenger preferences and the new transit graph $G' = G \cup \Delta G$ (updated by the transit plan ΔG), we implement a policy iteration algorithm to estimate/predict human behaviors, e.g., ridership.

2.3 Stage 1: Data Preprocessing

Map Griding. There are around $5,327$ bus stops and 167 subway stations in Shenzhen, China. Many of them are populated very densely in the urban area, especially in the downtown region. Usually, all transit options within a certain walking distance (e.g., 500 meters) are considered by the passengers. Hence, we partition the urban area into small regions and consider all transit stops in the same small region as a single aggregated transit stop. For the ease of implementation in practice, in this work, we adopt the griding based method, which simply partitions the map into equal side-length grids [100, 99].

Moreover, the griding based method allows us to adjust the side-length of grids, to better examine and understand impacts of the grid size. Hence, in Stage 1, our approach divides the urban area into equal-size grids with a pre-defined side-length s in kilometers. Fig 2.6 shows all grids in the bounding rectangle region of Shenzhen, China, with $s = 1km$. Then, we remove the grids without a road segment, which are usually located in the no-sense areas, such as ocean or mountain. The remaining grid set is denoted as G with $n = |G|$ grids, which can be represented as a graph, with grids as nodes, connected by the urban road network and transit system. Fig 2.6 highlights (in light color) those $n = 1,018$ grids on the road and transit network in Shenzhen, China. Note that we will examine the impacts of different griding side-lengths in Sec 2.6.

Trip Aggregation. We aggregate all transit nodes (i.e., stops and stations) into grids. Moreover, each trip demand $\langle src, dst, t\rangle$ specifies a source location src, and a destination location dst. we aggregate all trip demands to grid pairs, that is, for all trip demands with $src \in g_i$ and $dst \in g_j$, they will be considered in the same group with the source spoke g_i and destination grid g_j. We denote V_{ij} as the total number of trip demands with source grid as g_i and destination grid as g_j. Clearly, $V_{ij} = |\langle src, dst, t\rangle, src \in g_i, dst \in g_j|$. Then, the volume matrix $V = [V_{ij}]$ indicate the number of pairwise trip demands across the grids. Similarly, we can aggregate the transit trajectories on grid level.

2.4 Stage 2: Data-Driven Modeling

Passengers are making a sequence of decisions when completing a trip, such as which bus routes and subway line to take, which stop/station to transfer. Such sequential decision making processes can be naturally modeled as Markov decision processes (MDPs). Below, we will introduce the preliminaries of MDPs, and explain how we model the passenger route choice process as a MDP.

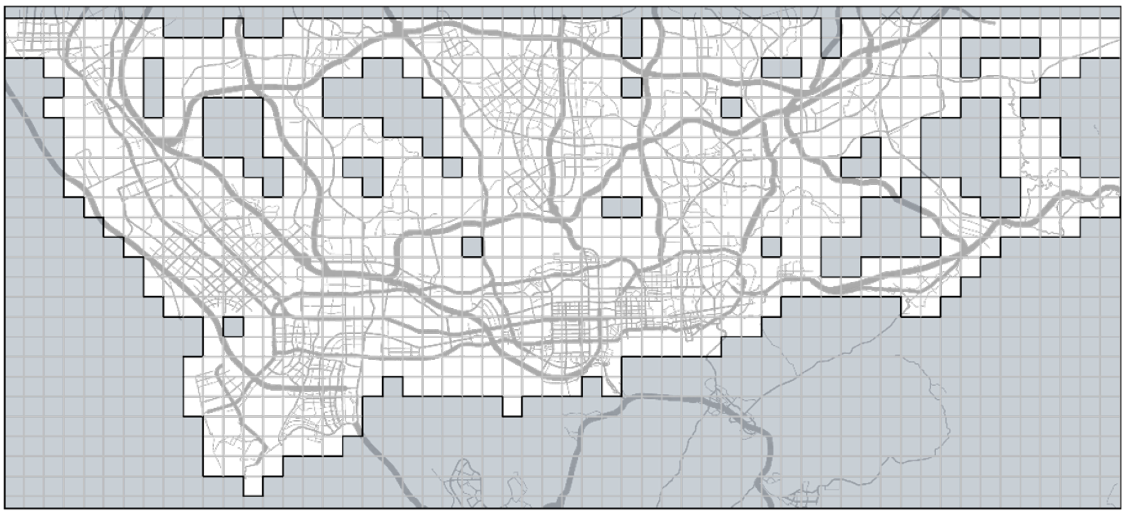


Figure 2.6: Map griding

2.4.1 Markov Decision Process (MDP)

Markov decision processes (MDPs) [149] provides a mathematical framework for modeling decision making processes, where outcomes are partly random and partly under the control of a decision maker, namely, an agent. A MDP is a discrete time stochastic control process. At each time step, the process is in some state s, and the agent may choose any action a that is available at state s, where the agent receives a corresponding reward $R(s, a)$. The process responds at the next time step by randomly moving into a new state s'. The probability that the process moves into its new state s' is influenced by the chosen action. Specifically, it is given by the state transition function $P(s' \mid s, a)$. Hence, an MDP can represented as a 5-tuple $\langle S, A, P, R, \gamma \rangle$, where S is a finite set of states and A is a set of actions. P is the probabilistic transition function with $P(s' \mid s, a)$ as the probability of arriving at state s' by executing action a at state s, $R : S \times A \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards. A randomized, memoryless policy is a function that specifies a probability distribution on the action to be executed in each state, defined as $\pi : S \times A \to [0, 1]$.

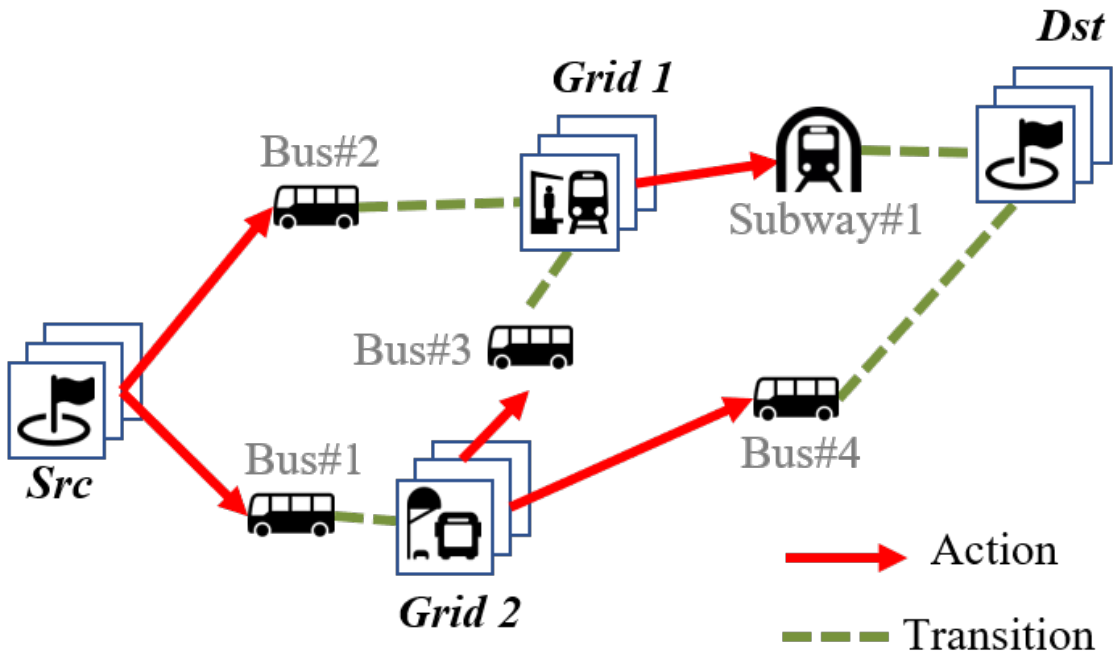


Figure 2.7: Illustration: urban transit choice as an MDP

The planning problem in an MDP aims to find a policy π, such that the expected total reward is maximized, namely,

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}^{\pi}(\sum_{t=0}^{T} \gamma^t R(S_t, A_t) \mid S_0 \sim \mu_0),$$

where S_t and A_t are random variables for the state and action at the time step t, and $T \in \mathbb{R} \cup \{\infty\}$ is the set of time horizons. The initial state S_0 follows the initial distribution $\mu_0 : S \to [0, 1]$. Here, Π is the memoryless policy space.

2.4.2 Modeling Transit Route Choice with MDP

We can consider each traveler in public transit system, as an "agent", who completes a trip from origin to destination by making a sequence of decisions about transit modes and routes. Inherently, each passenger evaluates various decision-making features asso-

ciated with the current state and each possible decision, such as travel time, cost, level-of-convenience, by the traveler's reward function. The reward function represents the preference the traveler has over different decision-making features. Each traveler is making their decisions that maximize the total "reward" she obtains out of the trip. As a result, we model the travelers' trips (i.e., their transit route choices) using Markov Decision Process (MDP) model. Below, we explain how each component in a MDP is extracted from travelers' transit trajectory data.

Agent: Instead of viewing each individual passenger as an agent, we consider an agent as a group of passengers with nearby source and destination locations. Since in reality, people who live in the same residential community and working in the same commercial area tend to have the similar income level and family sizes, that likely lead to the similar preference profile in public transit decision making [86]. Moreover, this allows each agent (as a group of people) to have more trajectory data to learn their preference as a reward function. As we partition the entire urban area into small grids (in Stage 1), we consider all commute passenger trips with the same original and destination grids a single agent.

State set S: Each state $s \in S$ is a spatio-temporal region, denoted as a tuple $(g, \delta t)$, where g represents a grid in the urban area and δt is a discrete time slot with a predefined time interval. The state space S is thus finite, since the map is partitioned into a finite number of grids (e.g., $1,018$ grids in Fig 2.6) and each day is divided into 5-minutes intervals. For an agent, with a starting grid g_{src}, a destination grid g_{dst}, and a start time t_0, the state space only includes a limited number of spatio-temporal grids along the bus and subway lines from g_{src} to g_{dst}. For example, in Fig 2.7, an agent travels from grid g_{src} to g_{dst}, with two possible transfer grids (g_1 and g_2). If there are three time intervals considered in the scenarios, each grid is mapped to three MDP states, when combined with each time interval, as shown as the overlapped squares in Fig 2.6. Thus, there are in total 12 states in the example MDP.

Action set A: An action $a \in A$ is a transit choice decision a passenger can make when completing the trip, e.g., a certain bus route or subway line with transfer stations. For example, in Fig 2.7, the actions the passenger can make at state g_{src} include $Bus\#2 \rightarrow g_1$ and $Bus\#1 \rightarrow g_2$.

Transition probability function $P : S \times A \times S \rightarrow [0, 1]$: Due to the dynamics of urban road traffic and crowd flow conditions, after an agent takes an action a (e.g., bus route) at a state s, the time of reaching the transferring stop may vary, leading to different state s' (of the same spatial grid but different time interval). Such uncertainty is characterized by the transition probability function as $P(s' \mid s, a)$, representing the probability of arriving at the state s' after choosing action a at the state s. The transition probability is obtained from maximum likelihood estimation from real-world urban transit trajectory data as follows. Suppose that we observed m trajectories for an agent in the historical data. Each trajectory ζ is represented as a sequence of discrete states and actions $\zeta = \{s_0, a_0, s_1, a_1, \cdots, s_N\}$ where $s_N = (g_{dst}, \Delta t_N)$ is the destination state and $s_0 = (g_{src}, \Delta t_0)$ is the source state. With this information, the maximum likelihood estimator for the transition $(s, a) \rightarrow s'$ is obtained by $P(s' \mid s, a) = \frac{N(s,a,s')}{\sum_{s' \in S} N(s,a,s')}$, where $N(s, a, s')$ is the count of this transition observed from all historical trajectory data.

Reward R: When passengers make decisions of transit choices, they are considering various decision making features, such as travel time, cost, level-of-convenience. We will detail these decision-making features in Sec 2.4.3. In MDP, $R : S \times A \rightarrow \mathbb{R}$ represents the reward function. It captures the unique personal preferences of an agent, that maps the decision-making features (at a state s when taking an action a) to a reward value. The decision making features include travel time, cost, level-of-convenience, etc. We will discuss these features in the next subsection in more details. Such reward function $R(s, a)$ can be inversely learned from transit trajectory data (See Sec 2.5).

2.4.3 Decision-Making Features

Each state-action pair (s, a) is associated with a list features, represented as a vector $\mathbf{f}_{(\mathbf{s},\mathbf{a})}$, that travelers consider when making their transit decisions. We consider three types of decision-making features, including monetary cost, time cost, level-of-convenience.

2.4.3.1 Monetary Cost

In general, monetary cost indicates how much money the traveler needs to spend if taking an action a at a state s. It includes fare and remaining cost.

Fare(F). At a state-action pair (s, a), the fare feature captures the fare needed when taking an action a at a state s, e.g., the fare for taking a bus route or a subway line.

Remaining Cost(RC) captures the expected additional cost needed (after taking a at s) before reaching the destination. This feature can be viewed as a measurement of how cost-efficient (s, a) is. For example, some action a taken at s, may have a smaller fare feature, but may lead to a state s', that needs at least two or more transfers (i.e., no direct transit option available at s') to reach the destination, thus incurring more remaining cost.

2.4.3.2 Time Cost

Time cost is another crucial factor that travelers consider at a state s when making the next transit choice a, which includes travel time (to the next state s'), remaining time (to meet a deadline to reach destination), and traffic speed (on road network at the current state s).

Travel Time(TT) characterizes the expected travel time from the current state s to the next state after taking an action a. This feature value at a state-action pair (s, a) is estimated from the historical transit trajectory data, and is averaged over all possible next state s' from (s, a).

Remaining Time(RT): We observe that more than 90% of trips are commute trips, namely, repeated round-trips of travelers between homes and working places. The travelers may need to meet certain deadlines to reach destinations, i.e., a traveler needs to arrive the company by 9:00AM in each working day. Such deadlines can be extracted from the historical transit trajectory data, namely, taking the latest arrival time from historical trips as the deadlines. The remaining time feature captures the time difference between the current state time and the deadline, indicating how urgent the current trip is.

Traffic Speed(TS) indicates the average road network traffic speed in the grid and time interval of the current state s. This feature captures the potential influence to the travel time if an above-ground transit mode, e.g., a bus route, is chosen. We observe that in general a traveler has higher chance to choose a nearby subway line if the traffic speed is low.

2.4.3.3 Level of Convenience (LoC)

When travelers make transit choices, they may also consider the comfortableness and convenience of an action a when made at a state s. Such features include number of transfers, number of choices, and transit mode.

Number of Transfers (NoT) represents the expected number of transfers the traveler needs to take after taking action a at state s, before reaching the destination.

Number of Choices (NoC) captures the total number of transit choices (including alternative bus routes, subway lines) the traveler can choose at the current state s. This feature indicates the flexibility the traveler has at a certain state s.

Transit Mode (TM) is a binary indicator feature, indicating if the chosen action is a subway (with $TM = 1$) or not.

2.5 Stage 3: Transit Plan Evaluation

With the MDP model to characterize the decision making process of travelers, we are in a position to investigate how we may evaluate human behaviors, e.g., ridership, of a transit plan ΔG prior to its deployment. The idea is that deploying a transit plan will change the existing transit graph G to $G' = G \cup \Delta G$, thus change the MDP with updated state space $S' = S \cup \Delta S$ and action space $A' = A \cup \Delta A$. However, the unchanged element in MDP is the travelers' preferences (i.e., reward functions), namely, how they evaluate different decision-making features to make transit choices. To evaluate the human behaviors (i.e., ridership) of a transit plan ΔG, we need to answer two questions:

Q1: Given the historical transit trajectory data of an agent (collected prior to a transit plan deployment), how can we learn the preference (reward) function R of the agent, that maximizes the likelihood the collected data being generated?

Q2: With the agents' reward functions (learned from data collected prior to the transit plan deployment), how can we predict the future human behaviors, (e.g., ridership) of a new transit plan, after it is deployed?

To answer $Q1$, we develop a novel preference learning algorithm to extract the reward functions of agents (i.e., travelers) (Sec 2.5.1). For $Q2$, we implement a policy iteration algorithm to infer the travelers' behaviors (as MDP policies) from the updated MDP with new state and action spaces S' and A' and the extracted traveler reward function R Sec 2.5.2.

2.5.1 Preference Learning

User choice modeling and preference learning has been extensively studied in the literature that aims to learn the people's decision-making preferences from the data they have generated [83, 173, 191, 207]. Multinomial logit (MNL) model [83, 191] considers users

making single one-step decision. On the other hand, maximum-entropy inverse reinforcement learning (MEIRL) [207] considers users making a sequence of decisions in MDP, which matches our passenger decision-making process well. However, MEIRL has a strong assumption that reward functions are linear. Below, we first describe the basics of MEIRL, highlight its limitation, and develop a novel preference learning algorithm that can capture the general non-linear reward function of travelers.

2.5.1.1 Maximum Entropy Inverse Reinforcement Learning

The inverse reinforcement learning problem in MDPs aims to find a reward function $R : S \times A \rightarrow \mathbb{R}$ such that the distribution of action and state sequences under a (near-)optimal policy match the demonstrated behaviors. One well-known solution to Maximum Entropy IRL problem [207] proposes to find the policy, which best represents demonstrated behaviors with the highest entropy, subject to the constraint of matching feature expectations to the distribution of demonstrated behaviors.

The reward $R(s, a)$ on a state-action pair (s, a) is given as a linear combination of the feature vector $\mathbf{f}_{(s,a)}$ with weight vector θ, i.e., $R(s, a) = \theta^\top \mathbf{f}_{(s,a)}$. Likewise, given a trajectory ζ with length N, the total reward received along ζ is written as $R(\zeta) = \theta^\top \mathbf{f}_\zeta$, where $\mathbf{f}_\zeta = \sum_{i=0}^{N} \mathbf{f}_{(s_i,a_i)}$. Applying the principle of maximum entropy, the probability of a trajectory ζ being generated follows eq.(2.1) below (See [207]).

$$P(\zeta) = \frac{1}{Z} e^{\theta^\top R(\zeta)}, \tag{2.1}$$

where $Z = \sum_{\zeta \in TR} e^{\theta^\top R(\zeta)}$ and TR is the set of all possible trajectories that can be generated. In non-deterministic MDPs, the user preference θ can be estimated using maximum likelihood estimation, $\theta^* = \mathrm{a}rgmax_\theta L(\theta) = \mathrm{a}rgmax_\theta \sum_{\zeta \in \tilde{TR}} \log P(\zeta \mid \theta)$. Then, a stan-

dard gradient decent method can solve it with

$$\nabla L(\theta) = \tilde{\mathbf{f}} - \sum_{\zeta \in TR} P(\zeta \mid \theta)\mathbf{f}_\zeta = \tilde{\mathbf{f}} - \sum_{s \in S, a \in A} D(s, a, \theta)\mathbf{f}_{(s,a)}, \tag{2.2}$$

where $\tilde{\mathbf{f}} = \frac{1}{|\tilde{TR}|}\sum_{\zeta \in \tilde{TR}} f(\zeta)$ is the expected empirical feature vector, with $\tilde{TR}$ as the demonstrated trajectory set. $D(s, a, \theta) = \sum_{\zeta \in TR, (s,a) \in \zeta} P(\zeta)$ is the expected state-action pair visitation frequency, indicating the chance of (s, a) being visited if a random trajectory ζ is generated.

2.5.1.2 Passenger Preference Learning

Maximum entropy IRL assumes each passenger reward function $R(s, a)$ to be a linear function to the feature vector $\mathbf{f}_{(s,a)}$ at (s, a). Such strong assumption limits its ability to accurately learn passengers' preferences. In Sec 2.6, we show comparison results with real world data between linear vs non-linear reward functions. Now, we will first expand MEIRL to allow non-linear reward functions, and then develop a computational efficient algorithm to inversely learn a non-linear reward function $R(s, a)$ that best match the collected transit trajectory data.

Consider a general non-linear reward function $R(s, a)$. With the principle of maximum entropy, $P(\zeta)$, the probability of a trajectory ζ being generated, can be uniquely obtained by solving the optimization problem below.

$$\textbf{Problem} \quad \textbf{P1}: \qquad \max_{P(\zeta)} : \sum_{\zeta \in TR} P(\zeta)(-\ln P(\zeta)), \tag{2.3}$$

$$s.t. \sum_{\zeta \in TR} P(\zeta) = 1, \tag{2.4}$$

$$\sum_{\zeta \in TR} P(\zeta)R(\zeta) = \tilde{R}. \tag{2.5}$$

$\tilde{R} = \frac{1}{|\tilde{TR}|} \sum_{\zeta \in \tilde{TR}} \tilde{R}(\zeta)$ represents the empirical average reward of trajectories, with $\tilde{TR}$ as the set of collected transit trajectories (i.e., the sample space). The objective function eq.(2.3) is the total entropy of the trajectory distribution. Constraint eq.(2.4) guarantees the total probability of all trajectories equals 1. Constraint eq.(2.5) specifies that the expected trajectory reward matches the empirical average trajectory reward $\tilde{R}$. The close-form solution to this problem is $P(\zeta) = \frac{1}{Z_R} e^{R(\zeta)}$, with $Z_R = \sum_{\zeta \in TR} e^{R(\zeta)}$,where TR is the set of all possible transit trajectories in the MDP (i.e., the population space).

Proof: Denote $P(\zeta)$ as the distribution of the trajectories generated by an agent with its optimal policy. We collected a sampled trajectory set $\tilde{TR} = \{\zeta\}$ from the agent. Problem P1 above is used to estimate the distribution $P(\zeta)$, so it matches the dataset $\tilde{TR}$ well, with a maximum entropy. The Lagrange function of P1 is represented as eq.(2.6).

$$\begin{aligned} L(P(\zeta)) = \sum_{\zeta \in TR} -P(\zeta) ln(P(\zeta) + \lambda_1 \big(\sum_{\zeta \in TR} P(\zeta) - 1 \big) \\ + \lambda_2 \big(\sum_{\zeta \in TR} P(\zeta) R(\zeta) - \tilde{R} \big) \end{aligned} \tag{2.6}$$

Taking the derivative of $L(P(\zeta))$ with respect to $P(\zeta)$, we obtain $\frac{\partial L(P(\zeta))}{\partial P(\zeta)} = 1 - ln(P(\zeta)) + \lambda_1 + \lambda_2 R(\zeta)$. When the derivative $\frac{\partial L(P(\zeta))}{\partial P(\zeta)} = 0$, we have $P(\zeta) = e^{1+\lambda_1} \cdot e^{\lambda_2 R(\zeta)}$. λ_2 is called "temperature" [149] used to quantify the degree to which the agent follows a sub-optimal policy vs a global optimal policy. Without loss of generality, we can set $\lambda_2 = 1$. Moreover, given the second constraint $\sum_{\zeta \in TR} P(\zeta) = 1$ (from eq.(2.5)), we can obtain $e^{1+\lambda_1} = 1/Z_R = 1/\sum_{\zeta \in TR} e^{R(\zeta)}$, which completes the proof. ■

Assume that the reward function follows a non-linear model, e.g. Neural Network (NN), with parameter vector θ, i.e., $R(s, a, \theta)$. The model parameter vector θ can be estimated using maximum likelihood estimation, say, solving the optimization problem

below.

$$\max_{\theta}: \qquad L(\theta) = \sum_{\zeta \in \tilde{TR}} \log P(\zeta \mid \theta) \tag{2.7}$$

Then, a standard gradient decent method can solve it with

$$\nabla L(\theta) = \sum_{s \in S, a \in A} \big(D^*(s,a) - D(s,a,\theta)\big) \frac{\partial R(s,a,\theta)}{\partial \theta}, \tag{2.8}$$

where $D^*(s,a) = \tilde{N}_{(s,a)}/|\tilde{TR}|$ and $\tilde{N}_{(s,a)}$ is the count from all transit trajectories that traverse the station-action pair (s,a). Clearly, at i-th iteration, the state-action pair visitation frequency $D(s,a,\theta_i)$ and the partial derivative $\frac{\partial R(s,a,\theta_i)}{\partial \theta_i}$ are required when updating $\theta_{i+1} = \theta_i + \alpha \cdot \nabla L(\theta)$, where α is the step size for updating θ. As a result, this approach only works for models that allow computing $\frac{\partial R(s,a,\theta)}{\partial \theta}$, such as neural network (with backward propagation), linear regression. Models, such as decision tree, random forest, cannot be applied as reward function models, since no derivatives can be computed. Moreover, mini-batch gradient decent approach cannot be applied, since each iteration requires updating visitation frequencies for all state-action pairs. To tackle these challenges, we employ the following observation (Theorem 2.5.1) to further improve the flexibility of the non-linear preference learning algorithm.

Theorem 2.5.1 *The optimization problem $P1$ is equivalent to the problem $P2$ below. They share the same optimal solution.*

$$\textbf{Problem} \quad \textbf{P2}: \qquad \max_{\theta, R^*}: \sum_{\zeta \in \tilde{TR}} \log \frac{1}{Z_R} e^{R^*(\zeta)}, \tag{2.9}$$

$$s.t. \sum_{\zeta \in \tilde{TR}} \left(R^*(\zeta) - R(\zeta, \theta)\right)^2 \leq \epsilon, \tag{2.10}$$

where $R^*(\zeta)$'s are the reward values received, that best describe the demonstrated trajec-

tory data, and $\epsilon > 0$ is a sufficiently small value.

Proof: Denote Lagrange function of P1 and P2 as $L_1(\theta)$ and $L_2(R^*(\zeta), \theta)$, respectively. $\frac{\partial L_1(\theta)}{\partial \theta}$ is clearly eq.(2.8), which can be rewritten as follows, with $\tilde{P}(\zeta)$ be the empirical probability of trajectory ζ to occur.

$$\mathbf{P1}: \quad \frac{\partial L_1(\theta)}{\partial \theta} = \sum_{\zeta \in \tilde{TR}} \big(\tilde{P}(\zeta) - P(\zeta, \theta)\big) \frac{\partial R(\zeta, \theta)}{\partial \theta} = 0. \tag{2.11}$$

Similarly, the partial derivative of $L_2(R^*(\zeta), \theta)$ is

$$\mathbf{P2}: \quad \frac{\partial L_2(R^*(\zeta), \theta)}{\partial \theta} = \sum_{\zeta \in \tilde{TR}} \big(R^*(\zeta) - R(\zeta, \theta)\big) \frac{\partial R(\zeta, \theta)}{\partial \theta} = 0. \tag{2.12}$$

$P(\zeta, \theta) = \frac{1}{Z_R} e^{R(\zeta, \theta)}$, with $Z_R = \sum_{\zeta \in TR} e^{R(\zeta, \theta)}$, and $\tilde{P}(\zeta) = \frac{1}{\tilde{Z}} e^{R^*(\zeta)}$ with $\tilde{Z} = \sum_{\zeta \in \tilde{TR}} e^{R^*(\zeta)}$. From eq.(2.12) above, P2 is seeking for a $R(\zeta, \theta)$ that is equal to $R^*(\zeta)$, which means that $Z_R = \tilde{Z}$. It is thus equivalent to $P(\zeta, \theta) = \tilde{P}(\zeta)$, which is the solution from problem P1 and completes the proof. ■

Problem P2 can be decomposed into two subproblems as follows.

$$\mathbf{Subproblem} \quad \mathbf{2-1}: \quad \max_{R^*(\zeta)} \sum_{\zeta \in \tilde{TR}} \log \frac{1}{Z_R} e^{R^*(\zeta)}, \tag{2.13}$$

$$\mathbf{Subproblem} \quad \mathbf{2-2}: \quad \min_{\theta} \sum_{\zeta \in \tilde{TR}} (R^*(\zeta) - R(\zeta, \theta))^2. \tag{2.14}$$

Since $R^*(\zeta) = \sum_{(s,a) \in \zeta} R^*(s, a)$, from Subproblem 2-1, it is easy to prove that $R^*(s, a)$ can be learned with gradient decent, with $\nabla L(R^*(s, a)) = D^*(s, a) - D(s, a)$, and $R^*(s, a)$ represents the reward value received at a state-action pair (s, a). Then, $R^*(s, a)$ can be used as input of Subproblem 2-2 to solve an optimal model parameter

Algorithm 1 Preference Learning

1: **INPUT:** MDP M ,trajectories $\tilde{TR}$ and learning rate α, $\mathbf{f}_{(s,a)}$;
2: **OUTPUT:** Preference function $R^*(s,a)$;
3: Initialize $R_0^*(s,a)$; $i=1$;
4: **while** $\|\nabla L(R^*(s,a))\|_2 > \epsilon$ **do**
5: update $D(s,a)$ with $R_{i-1}^*(s,a)$ using Alg 2;
6: update $\nabla L(R^*(s,a)) = D^*(s,a) - D(s,a)$;
7: $R_i^*(s,a) = R_{i-1}^*(s,a) + \alpha * \nabla L(R^*(s,a))$;
8: $i++$;
9: Train a model $R(s,a,\theta)$ with features $\mathbf{f}_{(s,a)}$ and labels $R_{i-1}^*(s,a)$;
10: return $R^*(s,a)$, $R(s,a,\theta)$ and θ;

θ. Note that here the subproblem 2-2 can be viewed and solved using supervised learning approach, which is compatible with any supervised model.

Alg 1 presents the pseudo-code of the passenger preference learning algorithm. Line 4–5 update the state-action pair visitation frequency using Alg 2. Line 6–8 update $R^*(s,a)$ with gradient decent method. Line 9 build supervise machine learning model using vector $\mathbf{f}_{(s,a)}$ as features, and $R^*(s,a)$ as labels. Alg 2 computes the optimal policy $\pi(s,a)$ with policy search, and calculates the state-action pair visitation frequency by solving a group of linear equations as eq (2.15) and eq (2.16).

$$D(s) - \sum_{s' \in S} \sum_{a' \in A(s')} D(s',a') \cdot P(s \mid s',a') = u_0(s), \forall s \in S, \tag{2.15}$$

$$D(s,a) = D(s)\pi(s,a), \forall (s,a) \in (S,A), \tag{2.16}$$

where $D(s) = \sum_{a \in A(s)} D(s,a)$ represents the station visitation frequency, namely, the likelihood of visiting the state s, if a random trajectory is generated. $u_0(s)$ is the initial distribution.

Algorithm 2 Computing State-Action Visitation Frequency

INPUT: $MDP = \langle S, A, P, R(s,a), 1 \rangle$;
OUTPUT: State-action pair visitation frequency $D(s,a)$;
Solve optimal policy $\pi(s,a)$ from MDP with policy iteration [149];
Solve $D(s,a)$ from eq (2.15) and eq (2.16);
return $D(s,a)$;

2.5.2 Transit Plan Evaluation

Given a new transit plan $\Delta G = \langle \Delta V, \Delta E \rangle$, e.g., a new subway line or bus route, the new state and action spaces are $S' = S \cup \Delta S$ and $A' = A \cup \Delta A$, respectively. With the passenger preference function $R(s,a)$, the updated MDP with new transit plan is represented as $\langle S', A', T, R(s,a), \gamma \rangle$, where transition probability matrix T is considered unchanged, and discount factor $\gamma = 1$. With such a new MDP, we can apply Alg 2 to extract the optimal policies $\pi(s,a)$ passengers will employ, $D(s)$ state visitation frequency, $D(s,a)$ state-action pair visitation frequency. These statistics can evaluate many aspects of the new transit plan ΔG, including the ridership of new transit lines, and crowd flow at different regions over time.

Ridership of new transit lines. To evaluate the ridership of a new transit plan at a certain station, we denote (s_e, a_e) as the state-action pair of our interests. State s_e corresponds to the grid with the station to be evaluated. Recall that we consider all trips with the same source-destination grid pair as an agent. Let m denote the total number of agents in the urban area of our interests. For each agent $1 \leq i \leq m$, we denote n_i as the number of passengers in the agent i. we apply Alg 2 to extract the state-action pair visitation frequency $D_i(s_e, a_e)$ for agent i at the state-action pair (s_e, a_e). As a results, $Rider(s_e, a_e) = \sum_{i=1}^{m} D_i(s_e, a_e) \cdot n_i$ represents the total ridership getting on the new transit line a_e from state s_e.

Crowd flow in a grid. Similarly, to predict the total crowd flow at a grid g during certain time interval Δt, let $s_e = [g, \Delta t]$ denote the corresponding state in MDP. We calculate

the state visitation frequency $D(s_e)$ over all m agents. The crowd flow at state s_e can be estimated as $Crowd(s_e) = \sum_{i=1}^{m} D_i(s_e) \cdot n_i$.

2.6 Evaluations

There were three new subway lines opened in 6/01/2016-12/31/2016, i.e., Line #11 on June 28, 2016, and Line #7 and #9 on October 28, 2016. To test the human-centric transit plan evaluation algorithm, we use data prior to the deployment to build models and predict the ridership of those new subway stations, and validate the prediction results with the data collected after their deployment.

2.6.1 Baseline methods

We conduct two sets of experiments: (i) compare reward learning (*Subproblem P2-1* in eq.(2.13) with other inverse learning baselines; (ii) compare ridership prediction (*Subproblem P2-2* in eq.(2.14) with other baseline frameworks.

Baselines for reward learning.

Our Method: Inverse Reinforcement Learning with Sub-optimal Policy(IRL+SP) (Line #1–#8 in Alg 1). It chooses the reward function with the principle of maximum entropy, and assumes that human make decisions with softmax-based suboptimal policies [207].

Baseline 1: Inverse Reinforcement Learning with Optimal Policy (IRL+OP). This baseline chooses the reward function with the principle of maximum entropy, but assumes that human decision-making follows an optimal deterministic policy, which means at each state, there is only one action being taken.

Baseline 2: Apprenticeship Learning (AL) [1]. This baseline employs the principle of maximizing the reward gap between the best and second best policies to choose reward function. It still assumes that passengers take optimal deterministic policy.

Baselines for ridership prediction.

Our Method: Alg 1 combined with multiple machine learning models as Line #9, such as random forest, lasso regression, and linear regression.

Baseline 1: Machine Learning Models (ML). We directly train machine learning models to predict the ridership.

Baseline 2: Multinomial Logit (MNL) Model [62]. This baseline considers each passenger make a single decision of the entire trip trajectory, rather than a sequence of decisions. MNL is used for the reward learning, where the preference learning still employs different machine learning models.

2.6.2 Experiment settings

The evaluations are conducted on a server with 24 Intel X5670 2.93GHz processors, 94GB memory, and Suse Linux Enterprise Server 11. We use stopping criteria as $\epsilon_1 = 10^{-12}$ for *Gradient Decent* unless mentioned otherwise. We employ the following two metrics to evaluate the reward learning and ridership prediction.

Visitation frequency difference. During the reward learning process, we not only extract reward values $R^*(s, a)$ that best match the demonstrated trajectory data, we also obtain the state-action pair visitation frequency $D(s, a)$ for each state-action pair. It can be expressed as a visitation frequency vector $D = [D(s, a)]$. Likewise, we can obtain an empirical state-action pair visitation frequency vector $\tilde{D} = [D^*(s, a)]$ from the transit trajectory data. Each $D^*(s, a)$ represents the percentage of collected trajectories that went through (s, a). The visitation frequency difference is the 2-norm difference of the two vector D and $\tilde{D}$, which characterizes how accurate the learned reward values are.

Ridership Prediction Error. Given a new subway station of a new transit plan (i.e., a subway line), which is within a grid g, denote s as a state over the grid g. The new transit line is considered as an action a. Let $N(s, a)$ be the number of passengers taking the new

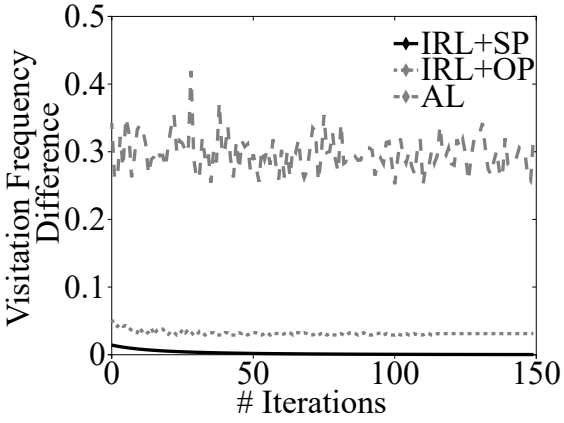


Figure 2.8: Ridership vector difference over iterations.

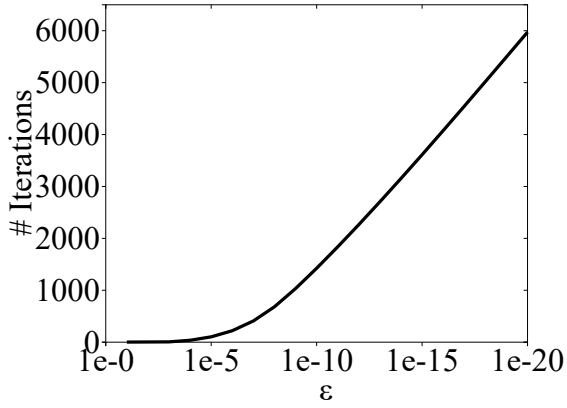


Figure 2.9: Iterations over Ridership vector difference

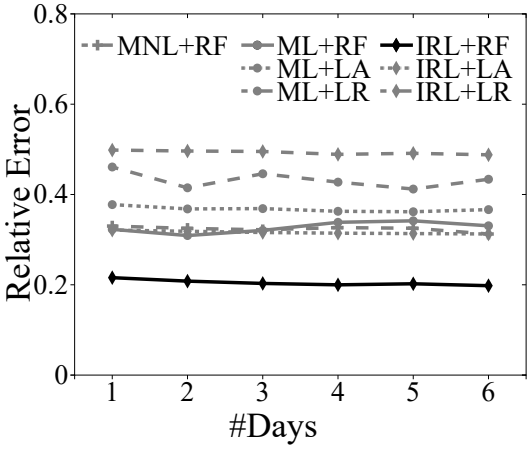


Figure 2.10: Prediction Error over Days of Data Used

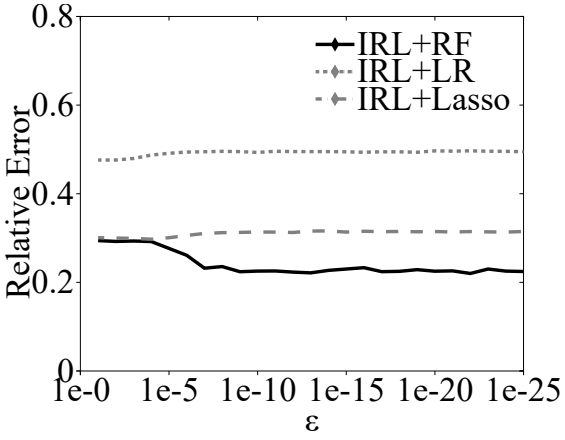


Figure 2.11: Prediction Error over IRL accuracy.

transit line a at state s after the new line is opened, which can be obtained from the data. Denote $N'(s,a)$ as the predicted ridership at state s, taking transit line a. The Ridership prediction error is quantified as $|N(s,a) - N'(s,a)|/N(s,a)$.

2.6.3 Preference Learning

We compare our preference learning method IRL+SP with two baseline methods, including IRL+OP and AL in Section 2.6.1.

The Fig 2.8 clearly indicates that our IRL+SP algorithm outperforms baseline meth-

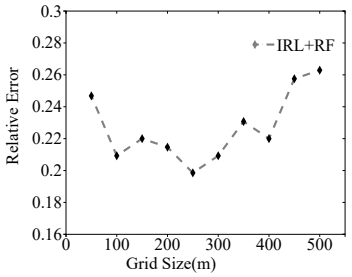


Figure 2.12: Prediction error over grid size.

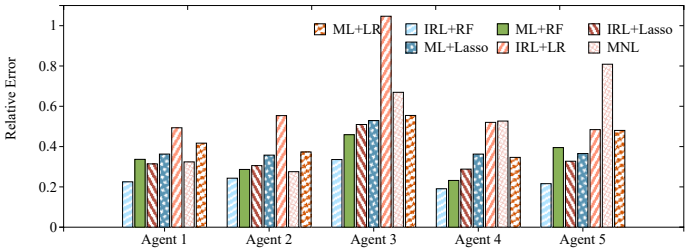


Figure 2.13: Prediction error over agents.

ods on ridership vector difference. After convergence, our IRL+SP method leads to the lowest ridership vector difference, around 3×10^{-12}, where IRL+OP and AL have ridership vector differences as high as 0.023 and 0.3, respectively. Such results indicate that passengers make sub-optimal decisions, rather than optimal decisions. In terms of convergence rate, the results in Fig 2.8 show that both our IRL-SP method and IRL-OP method converge fast, within 30 iterations, while the results with AL fluctuates over iterations.

Fig 2.9 shows how many iterations our IRL-SP algorithm (Alg 1) needs to converge with different stopping criteria (ϵ in Alg 1), i.e., ranging from 10^{-1} to 10^{-20}. The results are promising: less than 1K iterations are needed to achieve a stopping criteria with accuracy as high as 10^{-10}.

2.6.4 Ridership Prediction

Now, we evaluate the accuracy and efficiency of our proposed algorithm in predicting the ridership of a new transit plan.

Prediction accuracy.

Table 2.1 shows the relative errors in ridership prediction for a new transit plan, when comparing with different baseline algorithm. Clearly, our IRL-SP Algorithm (when combined with Random Forest (RF) model at Line #9 in Alg 1) yields the lowest prediction relative error, about 19.8%. IRL-SP based models (right-most column) have the lowest

Table 2.1: Ridership Prediction Error

	ML	MNL	IRL
Linear	0.4337	0.3684	0.4879
Lasso	0.3665	0.3214	0.3126
Random Forest	0.3306	0.3152	0.1982

errors than machine learning models and multinomial logit models. This indicates that passengers are making a sequence of decisions rather than one decision when planning their trip demands. Moreover, the linear models (ML+LR, MNL and IRL+LR) shown in the first row of Table 2.1 have relative errors about 36%–49%, much higher than other non-linear models. This indicates that passengers are evaluating various decision-making features in a non-linear fashion. On the other hand, random forest model when directly used for prediction (ML), or combined with MNL and IRL, outperform linear models and Lasso based models.

Impact of training data size. Fig 2.10 shows how the ridership prediction relative error changes over the amount of training data we use. We change the size of training data from 1 day data to 6 days' data. Clearly, the relative errors of all approaches decrease when more training data are included.

Impact of stopping criteria ϵ. Fig 2.11 shows how the relative errors change with different stopping criteria ϵ of Alg 1, ranging from 10^{-1} to 10^{-25}. The relative error goes down as ϵ decreases for IRL with RF, where it keeps unchanged for IRL with LR and Lasso. Moreover, when ϵ is lower than 10^{-12}, the relative error remains stable. As a result, we choose $\epsilon = 10^{-12}$ in our experiments.

Impact of grid size. Fig 2.12 shows the prediction relative error changes over the grid size. We vary the grid side-length from $50m$ to $500m$, and observe that $250m$ yields the lowest relative error. This can be explained as follows: If the grid size is too small, there will be likely fewer stations in each grid, thus less trajectories aggregated in each grids.

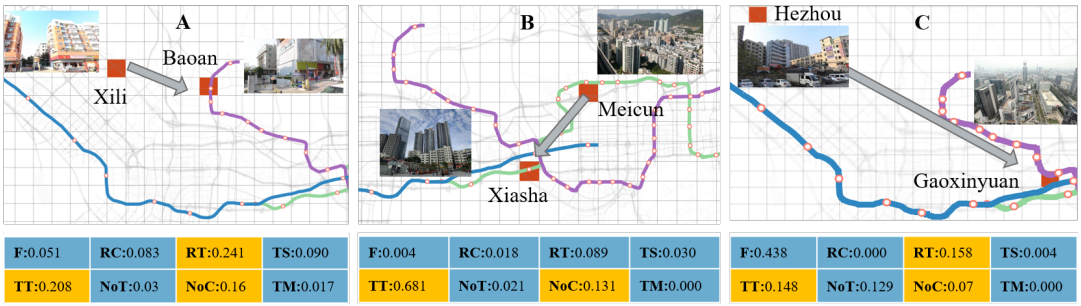


Figure 2.14: Illustration of 3 different agents in Shenzhen[1]

Such sparse data lead to high prediction error. On the other hand, with too large grids, more trajectories and stations will be aggregated into each station, and different decision choices from passengers will be mixed together and considered identical, thus increase the prediction error.

Impact of different agents. We apply all baseline algorithms to different agents, compare their prediction relative errors. Fig 2.13 shows the comparison results on five randomly chosen agents. It is clear that for all five agents, the prediction errors are consistent over different baseline algorithms, i.e., (i) IRL based models out perform ML and MNL based models, (ii) non-linear models performs better than linear models.

2.6.5 Case studies

Now, we show three agents (group of passengers) as examples, which have significantly different reward functions, e.g., weights on decision-making features. Fig 2.14 (A)-(C) show the locations of the source and destination grids, and their preferences to eight different decision-making features.

Fig 2.14(A) represents the traveler group from Baoan to Xili. Baoan is a residential area with low housing prices, where there are many manufacture factories in Xili. So the commuters from Baoan to Xili are likely workers with low income, which explains why their weights to fare and remaining cost are higher. Moreover, factories in Xili

usually require their employees following fixed schedule. As a result, their preference to remaining time is much higher than other agents.

Fig 2.14(B) shows the traveler group from Meicun to Xiasha. Meicun, as a residential area, has high housing prices, where there are many technology companies, with high salary and flexible working schedule. These observations explain the preferences extracted, i.e., they weigh travel time highly when making decisions, while weighing the fare, remaining cost, and remaining time lower.

Fig 2.14(C) presents passenger group from Hezhou to Gaoxinyuan (High-Tech Park). Hezhou has lower housing price, and is in rural area with long distance from Gaoxinyuan. Hence, the passenger group are more likely with low income level. This matches the preference learned from the data, where they turn to weigh more on fare, and weigh less on travel time.

2.7 Related Work

In this section, we summarize the literature works in two related areas to our study: 1) urban computing, and 2) user choice modeling.

Urban Computing. Urban computing integrates urban sensing, data management and data analytic together as a unified process to explore, analyze and solve existing critical problems in urban area such as traffic congestion, energy consumption and pollution[202]. In [106], the authors propose an dynamic urban transit system with shared shuttles using hybrid hub-and-spoke mode. The authors in [7] employ real world trajectory data of sharing bikes to develop bike lane planning strategies. In [161, 187], the authors detect urban events from heterogeneous urban datasets. However, none of those works have explicitly studied the "urban human factors", i.e., how people make decisions. Our work

[1]Fare(F), Remaining Cost(RC), Travel Time(TT), Remaining Time(RT), Traffic Speed(TS), Number of Transfers(NoT), Number of Choices(NoC), Transit Mode(TM)

is the first study investigating how to quantify human preferences, and consider such preferences in urban transit planning.

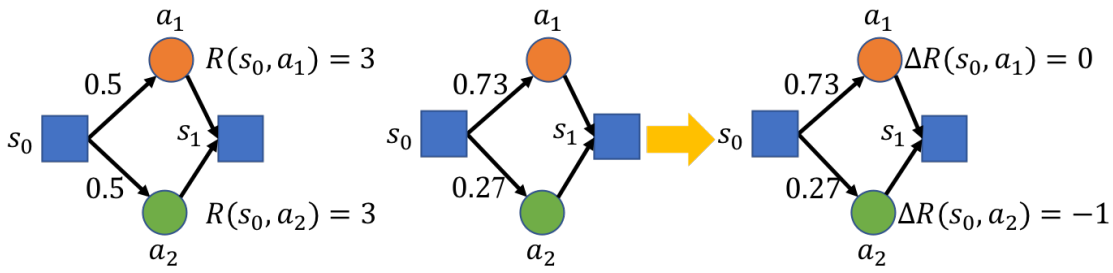
User Choice Modeling. User choice modeling has been extensively studied in the literature with applications, which investigate how users make decisions in various application scenarios. For examples, In [144], they use random utility maximization and random regret minimization to analyze users' choice on park-and-ride lots. In [129], authors estimate a multinomial discrete choice model and a latent variable model of travel mode choice. In [207], where the authors propose a probabilistic approach to discover reward function for which a near-optimal policy closely mimics observed behaviors. However, differing from these works, we employ data-driven approaches to study the unique decision-making process of urban public transit passengers, by proposing a novel learning algorithm to capture the general non-linear preference functions.

Chapter 3

Transforming Policy via Reward Advancement

3.1 Introduction

In sequential decision making problems [206], human agents complete tasks by evaluating the rewards received over states traversed and actions employed. Each human agent may have her own unique reward function, which governs how much reward she may receive over states and actions [170, 191]. For example, urban travelers may evaluate the travel cost vs travel time with different weights, when deciding which transport mode, route, and transfer stations to take [173]. Uber drivers may prefer different urban regions to look for passengers, depending on their familiarity to the regions, and distance to their home locations, etc [175]. To quantify and measure the unique reward function each human agent possesses, maximum causal entropy inverse reinforcement learning (IRL) [207] has been proposed to find the reward function and the corresponding policy, that best represents demonstrated behaviors from the human agent with the highest causal entropy, subject to the constraint of matching feature expectations to the distribution of



(a). Original Reward and Policy (b). Target Policy (c). Additional Reward

Figure 3.1: An Example of Reward Advancement

demonstrated behaviors.

Going beyond the human agent reward learning problem, in this work, we move one step further to investigate how we can influence and change agent's policy (i.e., decisions) to a target policy π_t from the original policy π_o observed from the agent's trajectories, by purposely updating and advancing the rewards received by the human agent. Figure 3.1 illustrates this problem with a concrete example. A small scale Markov Decision Process (MDP) has two states $\{s_0, s_1\}$, and two actions $\{a_1, a_2\}$. Starting from s_0, an agent can reach s_1 by either taking action a_1 or a_2. The original rewards received by the agent from taking action a_1 are a_2 are equal, i.e., $R(s_0, a_1) = R(s_0, a_2) = 3$ (Figure 3.1(a)). As a result, the policy of choosing a_1 vs a_2 are both 50% by the maximum entropy principle [207]. If we want the human agent to switch to a new policy (see Figure 3.1(b)) with $\pi_t(a_1|s_0) = 0.73$ and $\pi_t(a_2|s_0) = 0.27$, respectively, we can introduce additional reward $\Delta R(s_0, a_2) = -1$ to state-action (s_0, a_2), and keep $R(s_0, a_1)$ invariant (Figure 3.1(c)).

This problem of finding additional reward to transform human agent's policy with minimum cost is of great practical importance. For example, urban passengers employ their own unique policies to choose transit modes and transfer stations, which may collectively lead to unbalanced crowd flows, i.e., under- and over-supplied traffic over stations and routes. One way to mitigate such a problem is to motivate or incentivize passengers

to autonomously change and transform their policies, e.g., by providing additional reward to the those passenger agents, in forms of coupons, discounted price, etc [202, 86]. Moreover, it is crucial how to achieve this goal with minimum overall cost. In the literature, reward transformations [118, 79] have been studied extensively, primarily focusing on transforming the reward, with the goal of preserving the same policy (which is formally termed as "reward shaping"). Differing from reward shaping, our design goal is more general, namely, transforming rewards, so the agent behaves as a target policy π_t, which may or may not be the same as the agent's original policy π_o. We refer this problem as a "reward advancement" problem.

In this chapter, we make the first attempt to tackle the reward advancement problem. Given a Markov Decision Process and a target policy π_t, we investigate the range of additional rewards that can transform the agent's policy to the predefined target policy π_t under MCE principle. Our main contributions are summarized as follows.

- We are the first to define and study the reward advancement problem, namely, finding the updating rewards to transform human agent's behaving policy to a predefined target policy. We provide a close-form solution to this problem. The solution indicates that there exist infinite many such additional rewards, that can achieve the desired policy transformation.
- Moreover, we define and investigate min-cost reward advancement problem, which aims to find the additional rewards that can transform the agent's policy to π_t, while minimizing the cost of the policy transformation.
- We also demonstrated the correctness and accuracy of our reward advancement algorithm using both synthetic data and a large-scale (6 months) passenger-level public transit data from Shenzhen, China.

The chapter is organized as follows, Section 3.2 discusses preliminaries and formally

defines the reward advancement problem. Section 3.3 introduces our maximum entropy reward advancement algorithm. Section 3.5 presents evaluation results using both grid world scenario and real world urban passenger data.

3.2 Preliminaries and Problem Definition

In this section, we review the basics of finite Markov Decision Process and Maximum Causal Entropy (MCE) policy.

3.2.1 Markov Decision Process

An MDP is represented as a tuple $\langle S, A, T, \gamma, \mu_0, R\rangle$, where S is a finite set of states and A is a finite set of actions. T is the probabilistic transition function with $T(s'|s,a)$ as the probability of arriving at state s' by executing action a at state s, $\gamma \in (0,1]$ is the discount factor[1], $\mu_0 : S \to [0,1]$ is the initial distribution, and $R : S \times A \to \mathbb{R}$ is the reward function. A randomized, memoryless policy is a function that specifies a probability distribution on the action to be executed in each state, defined as $\pi : S \times A \to [0,1]$. We use $\zeta = [s_0, a_0, s_1, a_1, \ldots, s_L, a_L]$ to denote a trajectory generated by the MDP, which is a sequence of state-action pair. L is the length of trajectory. The planning problem in an MDP aims to find a policy π, such that the expected total reward is maximized.

3.2.2 Policy under Maximum Causal Entropy Principle

One well-known solution to the inverse reinforcement learning problem is Maximum Causal Entropy Inverse Reinforcement Learning [206]. It proposes to find the policy that best represents demonstrated behaviors with highest causal entropy $H(A||S)$, which

[1]Without loss of generality, we assume $\gamma = 1$ in this work, and it is straightforward to generalize our results to $\gamma \neq 1$.

is calculated by $H(A||S) = -\sum_{s\in S}\sum_{a\in A} D(s,a)\ln\pi(a|s)$, where $D(s,a)$ represents the expected visitation frequency of the state-action pair (s,a), when one trajectory is generated under policy $\pi(a|s)$.

The policy under maximum causal entropy principle (i.e., *MCE policy*) best represents the demonstrated behaviors with the highest causal entropy, and is subject to matching the reward expectations of demonstrated behaviors. Denote $Q(s,a) = R(s,a) + \sum_{s'\in S} T(s'|s,a)\sum_{a'\in A}\pi(a'|s')Q(s',a')$ as Q-function on state-action pair (s,a), indicating the expected rewards to be received starting from (s,a), MCE policy is

$$\pi(a|s) = \frac{e^{Q(s,a)}}{\sum_{a'\in A} e^{Q(s,a')}}. \tag{3.1}$$

Eq.(3.1) is the policy conducted by the human agent, that best matches her generated trajectory data. Usually, the $Q(s,a)$ can be represented using a parameterized function $Q(s,a|\theta)$, for instance, a neural network model. If we use $\tilde{TR}$ to represent expert trajectories we collected, the Q-function $Q(s,a|\theta)$'s then can be estimated by solving a maximum likelihood estimation problem,

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, L(\theta) = \operatorname{argmax}\theta \sum_{\zeta\in\tilde{TR}} \ln P(\zeta|\theta). \tag{3.2}$$

3.3 Reward Advancement

Inverse reinforcement learning problem [118, 206, 207, 43, 125, 174, 192] aims to inversely learn agent's reward (or preference) function from their demonstrated trajectories, namely, inferring how agent makes decisions. In this work, we move one step further to investigate how we can influence and transform agent's decision-making policy to a target policy π_t from the original policy π_o observed from the demonstrated trajectories, by purposely updating and advancing the reward functions $R(s,a)$ in the MDP. Reward transfor-

mations [118, 169] have been studied in the literature, primarily focusing on transforming the rewards, with the goal of preserving the same policy (which is formally termed as "reward shaping"). Differing from reward shaping, our design goal is more general, say, transforming rewards, so the agent behaves as a predefined target policy π_t, which may or may not be the agent's current policy π_o. This problem is referred to as a "reward advancement" problem, and we formally define it as follows.

Reward Advancement Problem. Given an MDP $\langle S, A, T, \mu_0, R_o \rangle$, the agent's MCE policy is π_o. we aim to find additional rewards ΔR to be added to the original reward R_o, such that the agent's MCE policy under the updated MDP $\langle S, A, T, \mu_0, R_o + \Delta R \rangle$ follows a predefined target policy π_t. Without loss of generality, we use $\gamma = 1$ as discount factor for simplicity.

For MDP $\langle S, A, T, \mu_0, R_o \rangle$, each policy π running on it leads to a unique Q-function:

$$Q_o^\pi(s,a) = R_o(s,a) + \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi(s',a') Q_o(s',a').$$

From Maximum Causal Entropy Inverse Reinforcement Learning, there exists a unique MCE policy π_o in form of eq.(3.1) that maximizes the likelihood of observing the given demonstration data. However, when appropriate additional rewards $\Delta R(s,a)$ are provided, MDP becomes $\langle S, A, T, \mu_0, R_o + \Delta R \rangle$, and the underlying MCE policy may change to π. This occurs because the additional rewards ΔR transforms and advances the MCE policy from π_o to π. In this case, the Q-function with MCE policy π is characterized as $Q^\pi(s,a) = R(s,a) + \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi(a'|s') Q^\pi(s',a')$, where $R(s,a) = R_o(s,a) + \Delta R(s,a)$, $Q^\pi(s,a) = Q_o^\pi(s,a) + \Delta Q(s,a)$, and $\Delta Q(s,a) = \Delta R(s,a) + \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi(a'|s') \Delta Q(s',a')$.

As a result, transforming from the original MCE policy π_o, the new MCE policy π is a function of addition reward ΔR, or equivalently ΔQ, i.e., $\pi(a|s; \Delta Q)$. Given a predefined

π_t, finding the right ΔQ, such that $\pi(a|s;\Delta Q) = \pi_t(a|s)$ for any $s \in S$ and $a \in A$, solves the reward advancement problem. The following Theorem 3.3.1 introduces the complete solution set to this problem.

Theorem 3.3.1 *Given an MDP $\langle S, A, T, \mu_0, R_o\rangle$, the sufficient and necessary condition to transform its MCE policy to a predefined policy π_t is to provide additional Q-function ΔQ, such that*

$$\Delta Q(s,a) = \ln \frac{\pi_t(a|s)}{e^{Q_o^{\pi_t}(s,a)}} + \beta(s), \tag{3.3}$$

where $\beta : S \to \mathbb{R}$ is any real number function defined on states. Such additional Q-function is called "advancement function".

Proof:

(Sufficiency.) If $\Delta Q(s,a)$ follows eq.(3.3), the new Q-function over π_t becomes:

$$Q^{\pi_t}(s,a) = Q_o^{\pi_t}(s,a) + \Delta Q(s,a) = \ln \pi_t(a|s) + \beta(s).$$

As a result, for any (s,a), the ratio between $e^{Q^{\pi_t}(s,a)}$ and $\sum_{a'} e^{Q^{\pi_t}(s,a')}$ is exactly $\pi_t(a|s)$, thus π_t is the MCE policy of the new MDP.

(Necessity.) Given a certain additional Q-function ΔQ, if it transforms the MCE policy to π_t, that infers

$$\pi_t(a|s) = \frac{e^{Q_o^{\pi_t}(s,a)+\Delta Q(s,a)}}{\sum_{a'\in A} e^{Q_o^{\pi_t}(s,a')+\Delta Q(s,a')}}, \tag{3.4}$$

$$e^{\Delta Q(s,a)} = \frac{\pi_t(a|s)}{e^{Q_o^{\pi_t}(s,a)}} \sum_{a'\in A} e^{Q_o^{\pi_t}(s,a')+\Delta Q(s,a')}. \tag{3.5}$$

Define $\beta(s) = \ln \sum_{a'\in A} e^{Q_o^{\pi_t}(s,a')+\Delta Q(s,a')}$, we have $\Delta Q(s,a) = \ln \frac{\pi_t(a|s)}{e^{Q(s,a)}} + \beta(s)$. It completes the proof. ■

The advancement function introduced in Theorem 3.3.1 is defined on additional Q-function, which can be easily "translated" into additional reward function ΔR by the following mapping function.

$$\Delta R(s,a) = \Delta Q(s,a) - \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi_t(s',a') \Delta Q(s',a'). \tag{3.6}$$

Theorem 3.3.1 indicates that there are infinite many advancement functions that can transform an original MCE policy π_o to a given π_t. However, different advancement functions may lead to different costs in reality to apply the additional rewards. For example, in ride-hailing service, additional rewards provided to Uber drivers could be in the form of monetary values; in urban public transportation systems, the additional rewards to passengers could be in forms of ride discount. More additional rewards applied lead to more cost to the system. Without lower bound on $\beta(s)$, the advancement function ΔQ can be as low as $-\infty$. In turn, the addition rewards ΔR inferred from eq.(3.6) can be arbitrarily small as well. It is equivalent to increase the ride rate to be extremely large for public transits, which is not feasible in real world scenario. Next, we will introduce and provide solution to the reward advancement problem with minimum cost as the objective.

3.4 Min-Cost Reward Advancement

Now, we investigate how to identify additional rewards that transform the agent to an MCE policy π_t, while guaranteeing minimum "implementation cost", namely, a *min-cost reward advancement problem*. Without loss of generality, we consider that the total cost of transforming the agent's policy is equal to the expected additional rewards offered to

the agent, i.e.,

$$C(\Delta R) = \sum_{s\in S}\sum_{a\in A} D_t(s,a)\Delta R(s,a)$$
$$= C(\Delta Q) = \sum_{s\in S}\sum_{a\in A} \mu_0(s)\pi_t(a|s)\Delta Q(s,a),$$

where $D_t(s,a)$ is the state-action pair visitation frequency under target policy π_t, and $\mu_0(s)$ is the initial state distribution. As a result, the general form of min-cost reward advancement problem can be formulated as follows.

Problem 1: Min-Cost Reward Advancement:

$$\min_{\Delta Q} \quad C(\Delta Q) = \sum_{s\in S}\sum_{a\in A} \mu_0(s)\pi_t(a|s)\Delta Q(s,a), \tag{3.7}$$

$$s.t. \quad \pi(a|s;\Delta Q) = \pi_t(a|s), \quad \forall s \in S, a \in A, \tag{3.8}$$

$$\Delta Q(s,a) \geq \phi(s,a), \quad \forall s \in S, a \in A. \tag{3.9}$$

Constraint eq.(3.8) guarantees to transform the agent's MCE policy to π_t, representing the infinite many feasible solutions given in Theorem 3.3.1. Constraint eq.(3.9) specifies the minimum additional expected reward we can offer to the agent, namely, $\phi : S \times A \to \mathbb{R}$ are system constants. Constraint eq.(3.9) makes sense in reality, which infers that the expected reward received by the agent cannot be lower than a certain minimum value. without this constraint, $\Delta Q(s,a) = -\infty$ becomes a trivial solution to Problem 1.

Theorem 3.4.1 *The solution to the min-cost reward advancement problem in eq.(3.7)-*

(3.9) is

$$\begin{cases} \Delta Q(s,a) = \max\limits_{a' \in A}(\ln \frac{e^{Q_o^{\pi_t}(s,a')}}{e^{Q_o^{\pi_t}(s,a)}} \frac{\pi_t(a|s)}{\pi_t(a'|s)} + \phi(s,a')), \mu_0(s) > 0, \\ \Delta Q(s,a) \geq \max\limits_{a' \in A}(\ln \frac{e^{Q_o^{\pi_t}(s,a')}}{e^{Q_o^{\pi_t}(s,a)}} \frac{\pi_t(a|s)}{\pi_t(a'|s)} + \phi(s,a')), \mu_0(s) = 0. \end{cases}$$

Proof: Theorem 3.3.1 indicates the solution set of constraint eq.(3.8). As a result, we can safely remove constraint eq.(3.8) and replace $\Delta Q(s,a)$ with eq.(3.3). Then, Problem 1 is transferred to the following format, with variable $\beta(s)$, instead.

Problem 2: Min-Cost Reward Advancement in β

$$\min_{\beta} \quad \sum_{s \in S} \sum_{a \in A} \mu_0(s) \pi_t(a|s) (\beta(s) + \ln \frac{\pi_t(a|s)}{e^{Q_o^{\pi_t}(s,a)}}), \tag{3.10}$$

$$s.t \quad \beta(s) \geq \max_{a \in A}(\ln \frac{e^{Q_o^{\pi_t}(s,a)}}{\pi_t(a|s)} + \phi(s,a)), \forall s \in S. \tag{3.11}$$

Eq.(3.10) is clearly a linear function of $\beta(s)$. As a result, the minimum objective function eq.(3.10) is achieved, when each $\beta(s)$ is minimum, that is, when the equality is attained in constraint eq.(3.11):

$$\beta(s) = \max_{a \in A}(\ln \frac{e^{Q_o^{\pi_t}(s,a)}}{\pi_t(a|s)} + \phi(s,a)), \quad \mu_0(s) \geq 0. \tag{3.12}$$

Moreover, eq.(3.10) indicates that the value of objective function only hinges on $\beta(s)$, with $\mu_0(s) > 0$. For other states with $\mu_0(s) = 0$, $\beta(s)$ only needs to fulfill the constraint eq.(3.11), and has no impact on the value of the objective function. The complete set of solutions to Problem 2 is as follows.

$$\begin{cases} \beta(s) = \max_{a \in A}(\ln \frac{e^{Q_o^{\pi_t}(s,a)}}{\pi_t(a|s)} + \phi(s,a)), & \mu_0(s) > 0, \\ \beta(s) \geq \max_{a \in A}(\ln \frac{e^{Q_o^{\pi_t}(s,a)}}{\pi_t(a|s)} + \phi(s,a)), & \mu_0(s) = 0. \end{cases}$$

Plugging the above solution set to eq.(3.3) yields the solution to ΔQ, and completes the proof. ■

Again, the solutions to advancement function ΔQ can be mapped to additional rewards ΔR, by applying

$$\Delta R(s,a) = \Delta Q(s,a) - \sum_{s' \in S} T(s'|s,a) \sum_{a' \in A} \pi_t(a'|s') \Delta Q(s',a').$$

Moreover, Theorem 3.4.1 indicates that the optimal solutions of $\Delta Q(s,a)$ on states with $\mu_0(s) > 0$ are unique (with equalities), while solutions on states with $\mu_0(s) = 0$ are infinite many, following inequalities.

Practical challenges in algorithm design. Theorem 3.4.1 provides a nice close-form solution set to the min-cost reward advancement problem. However, it requires calculating $Q_o^{\pi_t}(s,a)$ based on original reward $R_o(s,a)$ and target policy $\pi_t(a|s)$. A natural way to calculate it is value iteration method [207, 149], which employs an iterative framework, and solves a dynamic programming sub-problem within each iteration. As a result, such approach would be time-consuming, when the state space is large. Moreover, the value iteration will not work, when the transition matrix T is unknown.

To address the problems of scalability and unknown dynamics, we propose to apply Monte Carlo policy evaluation method to estimate Q-function of original rewards under target policy $Q_o^{\pi_t}(s,a)$. Here, we adopt the first-visit Monte Carlo method [149]. From first-visit Monte Carlo method, we have $Q_o(s,a)$ as the average total reward after first

visit to state-action pair (s, a) on each trajectory, denoted as

$$Q_o(s,a) = \frac{1}{|\tilde{TR}_{s,a}|} \sum_{\zeta \in \tilde{TR}_{s,a}} \sum_{t \geq t_{(s,a)|\zeta}} R_o(s_t, a_t), \tag{3.13}$$

where ζ is one trajectory, (s_t, a_t) is a state-action pair on the trajectory ζ, $t_{(s,a)|\zeta}$ is the first occurrence of (s, a) in ζ and $|\tilde{TR}_{s,a}|$ is number of trajectories traversing (s, a). Since trajectory set $\tilde{TR}_{s,a}$ were collected with original policy π_o, we adopt importance sampling method to estimate Q-function of original reward under target policy $Q_o^{\pi_t}(s, a)$ by

$$Q_o^{\pi_t}(s,a) = \frac{\sum_{\zeta \in \tilde{TR}_{s,a}} \sum_{t \geq t_{(s,a)|\zeta}} \frac{\pi_t(a|s)}{\pi_o(a|s)} R_o(s_t, a_t)}{|\tilde{TR}_{s,a}|}, \tag{3.14}$$

where the $\frac{\pi_t(a|s)}{\pi_o(a|s)}$ is the importance ratio. Similarly, we can estimate $\Delta R(s, a)$ through importance sampling as

$$\begin{aligned} \Delta R(s,a) &= \Delta Q(s,a) \\ &- \frac{1}{|\tilde{TR}_{s,a,s',a'}|} \sum_{\zeta \in \tilde{TR}_{s,a,s',a'}} \frac{\pi_t(a'|s')}{\pi_o(a'|s')} \Delta Q(s',a'), \end{aligned} \tag{3.15}$$

where $\tilde{TR}_{s,a,s',a'}$ are trajectories containing (s, a, s', a') and $|\tilde{TR}_{s,a,s',a'}|$ is the number of those trajectories. Moreover, if the original policy π_o is unknown, we can estimate it from the observed trajectories.

The algorithm for reward advancement via Monte Carlo Policy Evaluation is summarized in Algorithm 3. Specifically, Line 3 calculates $Q_o^{\pi_t}(s, a)$ using Monte Carlo policy evaluation.

Algorithm 3 Min-Cost Reward Advancement via Monte Carlo Policy Evaluation

1: **INPUT:** States S, Actions A, Original Rewards R_o and Original Trajectory Set $\tilde{TR}$;
2: **OUTPUT:** Additional reward on each state-action pair $\Delta R(s,a)$ (One from many solutions in Theorem 3.4.1);
3: For each state-action pair (s,a), calculate $Q_o^{\pi_t}(s,a) = \frac{\sum_{\zeta \in \tilde{TR}_{s,a}} \sum_{t \geq t_{(s,a)|\zeta}} \frac{\pi_t(a|s)}{\pi_o(a|s)} R_o(s_t,a_t)}{|\tilde{TR}_{s,a}|}$;
4: Calculate $\beta(s) = \max_{a \in A}(\ln \frac{e^{Q_o^{\pi_t}(s,a)}}{\pi_t(a|s)} + \phi(s,a))$ for each state s;
5: Calculate $\Delta Q(s,a) = \ln \frac{\pi_t(a|s)}{e^{Q_o^{\pi_t}(s,a)}} + \beta(s)$ for each stat-action pair (s,a);
6: For each (s,a), calculate $\Delta R(s,a) = \Delta Q(s,a) - \frac{1}{|\tilde{TR}_{s,a,s',a'}|} \sum_{\zeta \in \tilde{TR}_{s,a,s',a'}} \frac{\pi_t(a'|s')}{\pi_o(a'|s')} \Delta Q(s',a')$;
7: Return $\Delta R(s,a)$;

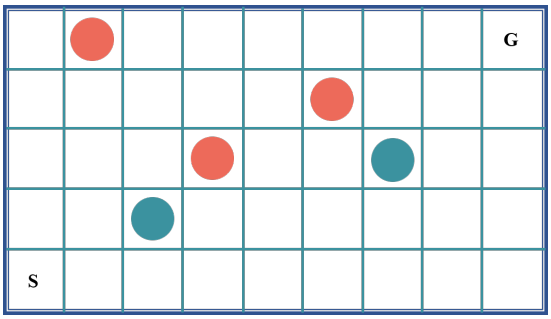


Figure 3.2: A 5×9 Object World with 2 different colors.

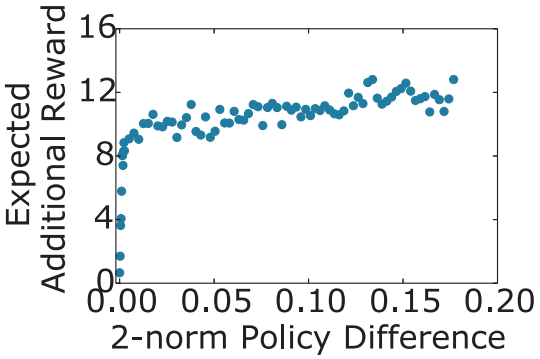


Figure 3.3: Expected additional reward over policy difference.

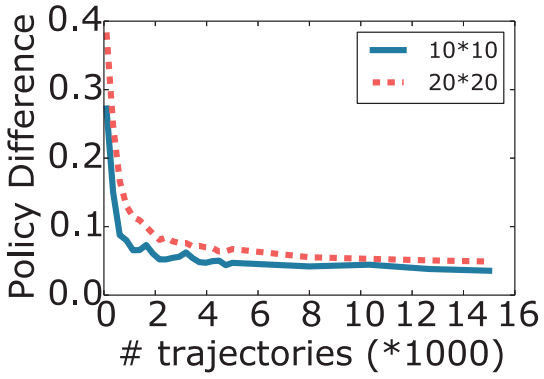


Figure 3.4: Policy difference over number of trajectories used.

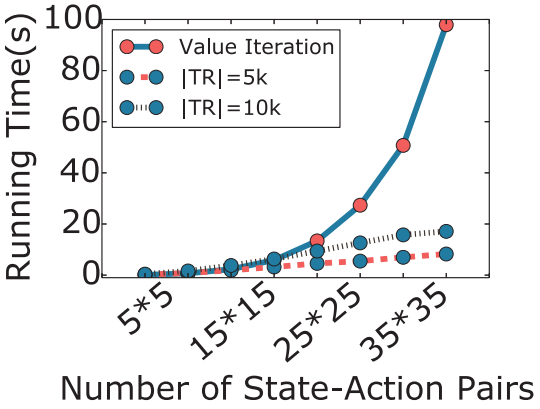


Figure 3.5: Running time over the size of state space.

3.5 Evaluation

In this section, we first evaluate the correctness and accuracy of our (min-cost) reward advancement algorithm, with synthetic object world scenario. Then, by modeling passengers' travel decisions in public transit system as a Markov Decision Process, we conduct empirical case studies using a large-scale (6 months) passenger-level public transit data collected in Shenzhen, China, from 07/01/2016 to 12/30/2016.

3.5.1 Evaluation on object world

First, we use an object world [91] scenario to evaluate our reward advancement algorithm. A Object World is a Grid World with random placed colored objects. Running into grids with objects in different colors will lead to different rewards. We call it "collect the object". The agent will also get a large reward by arriving the destination. So, the ideal policy should be going to the destination, while collecting as many objects with higher rewards as possible. Figure 3.2 shows an example of object world. There are 5×9 grids. We randomly placed 2 green objects and 3 red objects in the scenario. Each object has an color. An agent will gain a positive reward ($[5, 8]$ in our setting), when it reaches a grid with a red object, and a negative reward (ranging within $[-5, -3]$) when visiting a green object. A grid with no object leads to a negative reward of -1. Moreover, when the agent reaches the destination, it gets a large reward, within $[15, 30]$. At each grid, an agent can take 5 different actions, including "stay" and "move" towards one of four directions. With certain given transition probability, the agent would go to a random neighboring grid along the direction it has chosen. We choose discount factor γ to be 1 for all experiments. To evaluate our algorithm, we first randomly generate an Object World with pre-defined parameters, including the number of colors and objects. Then, we randomly place all objects in grids. We run value iteration with entropy-enhanced reward [142] to calculate a

randomized original policy for the agent in the generated Object World. The target policy is also randomly generated as the objectives of reward advancement.

Impact of difference between π_o and π_t. First, we examine the impact of the difference between original policy and target policy to the expected additional reward. We use normalized 2-norm difference of two policy vectors to indicates the policy difference, which is calculated by $\|\pi_o, \pi_t\|_2 = \frac{\sum_{s\in S}\sum_{a\in A}(\pi_o(a|s)-\pi_t(a|s))^2}{|(s,a)|}$, where $|(s,a)|$ is the number of state-action pairs. We evaluate the expected additional reward, which is calculated as $\sum_{s\in S}\sum_{a\in A} D_t(s,a)\Delta R(s,a)$ indicating the total amount of additional reward we would provide. It's hard to design target policy with specific difference from original policy, so we group target policy with similar difference together and calculate the average ΔR. The result is shown in Figure 3.3. The figure indicates that with increase of policy difference, the expected additional reward would increase. When larger than a certain threshold of policy difference, the expected reward increases linearly, while the policy difference increases linearly as well. This shows that even with target policy far from original difference, we can successfully transfer the policy at a reasonable cost.

Impact of the number of trajectories used in Monte Carlo policy evaluation. Monte Carlo algorithm is employed to reduce algorithm running time. However, lower running time lowers down the inference accuracy. The smaller the sample size is, the less accurate the policy transformation is. Denote π_t' as the policy transformed to after executing Algorithm 3. Figure 3.4 shows how Monte Carlo sample size impacts the difference between π_t and π_t'. It is clear that more sampled trajectories lead to more accurate results. Roughly, we need around $5,000$ trajectories to achieve a relatively good accuracy. Moreover, with the increase of Object World size, more trajectories are needed to enable accurate estimation of additional rewards, for policy transformation.

Impact of state space size on algorithm running time. Though we have obtained the close-form solution of $\Delta Q(s,a)$ and $\Delta R(s,a)$, computing $Q_o^{\pi_t}(s,a)$ is still the bottleneck

component, in running time. Here we evaluate the efficiency of the Monte Carlo based algorithm we proposed. Figure 3.5 shows the running time of 3 different algorithms. The blue solid line with red dot indicates running time of Value Iteration, the brown and red dash line with blue dot means running time of Monte Carlo method with $5,000$ and $10,000$ sampled trajectories, respectively. Obviously, the Monte Carlo method takes much less time than standard Value Iteration. Moreover, the running time of Monte Carlo method increases linearly, as the number of state increases quadratically. The running time increases with more trajectories used. The result shows that the Monte Carlo method is more computationally efficient.

3.5.2 Case studies

In this section, we use a real-world dataset to demonstrate the effectiveness of proposed reward advancement algorithm. To validate the algorithm, we need to verify that agent's behaviors still follow MCE principle after reward advancement. To validate this assumption, we use a large public transit dataset. We collected 6 months passenger-level public transit data from Shenzhen, China, which allows us to evaluate the potential of redistributing passengers by transforming their decision policies, in trip starting time, station and transport mode selection.

Passengers are making sequences of decisions when completing trips, such as which bus routes and subway lines to take, which stops/stations to transfer at. Such sequential decision making processes can be naturally modeled as Markov decision processes (MDPs). Since nearby stops/stations usually are similar to passengers, we will split the whole city into grid cells and aggregate stops/stations within each grid cell. The states are regional grids during different time intervals. Actions are available bus routes and subway lines passengers can take. Our model and formulation follow the work [175]. We inversely learn the reward functions of passengers using Maximum Causal Entropy

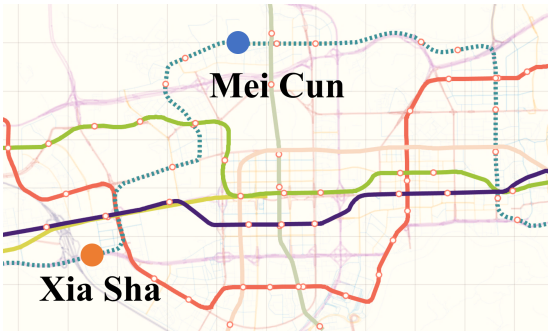


Figure 3.6: Map with source and destination of one agent and the newly established subways.

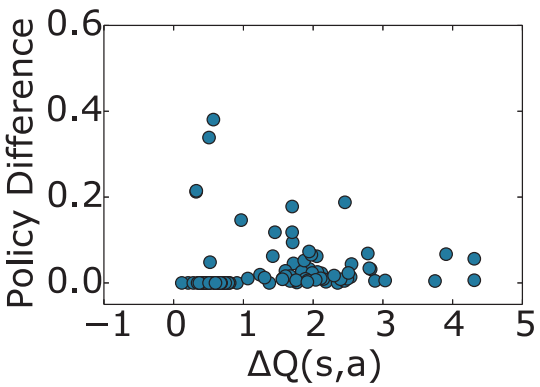


Figure 3.7: Policy difference vs $\Delta Q(s, a)$.

Inverse Reinforcement Learning [206] and the features we use include monetary cost, travel time, waiting time, and etc.

Additional reward can be provided using different methods, for example, deploying one new subway line can surely provide additional rewards to passengers in many different ways, for example, more transit choices, and lower average travel time. The Figure 3.6 illustrates the deployment of a new subway line in Shenzhen, which is the light blue dash line. To validate our assumption that providing additional rewards can transform passengers' behaviors to a target policy, we first learn the reward function from passengers' trajectories before deployment of the subway line. Then, we use features changed by deploying the new subway to calculate how much additional reward were provided. Lastly, we calculate a new policy, π_t, after reward advancement and compare this policy with ground-truth policy, π_{true}, of passengers after the deployment of new subway line.

Figure 3.7 shows that the policy differences of state-action pairs between π_t and π_{true} are small, with the X-axis as the Q-function difference of each state-action pair before and after subway deployment and the Y-axis as the relative error of π_t and π_{true}. The small difference between π_t and π_{true} validates that our reward advancement theory in Theorem 3.3.1.

Chapter 4

Using Sequential Models in General Human Prediction Problems

4.1 Introduction

Accurately predicting users' behavior based on their history and short-term features plays a critical role in many domains, such as major e-commerce companies, ride-hailing platform, social networking, and education. As an illustration, we consider an important problem of predicting drivers' behavior, such as working hours or income per day, in some ride-hailing platforms, such as Uber and Lyft. We call such a problem as **Drivers' Behavioral Prediction**. Solving such driver prediction problem usually requires the integration of drivers' history with some 'static' features including personal characteristics (e.g., age or gender), non-platform environment variables (e.g., government policy), and platform policies (e.g., dispatching policy) to predict drivers' future behaviors. Improving the accuracy of such prediction may allow the ride-hailing platform to achieve a healthy equilibrium between dynamic supply and demand systems.

However, most existing predictive models focus on the use of static features and some

summary statistics of users' history, such as Recency, Frequency, and Monetary (RFM) value [10]. For instance, RFM analysis as a marketing technique has been widely used to analyze customer's behavior including how recently a customer has purchased (recency), how often the customer purchases (frequency), and how much the customer spends (monetary). It is beneficial to improve customer segmentation by dividing customers into various groups for future personalized services and for identifying customers who are more likely to respond to promotions [77]. However, those summary statistics may have minimal predictive power in some problems, such as driver's behavioral prediction. For instance, as shown in Table 4.2, the inclusion of RFM gains little improvement in prediction accuracy.

A significant challenge associated with moving beyond the simple summary statistics of users' history is how to use representation learning to learn an effective embedding vector of the evolution of users' properties. In many cases, such evolution can be very complicated due to its considerable heterogeneity across users and/or time intervals. For instance, in the driver's behavioral prediction problem, we consider a working cycle model such that each driver makes a sequence of decisions ordered by time, such as being idle, taking orders, and logging off every day. Even within each driver, such a sequence pattern may vary dramatically across days. Moreover, since the working cycle model is a sequential decision problem, it may be solved by using reinforcement learning (RL) [201]. Different from normal sequential problem, which can be solved using RNN or LSTM, the original idea of RL is to find an optimal policy mapping from states to actions to maximize user's long-term rewards, rather than learning his/her embedding vector. Instead, we consider Inverse Reinforcement Learning (IRL) algorithm for learning the user's embedding vector based on his/her preference vector.

The IRL problems arise naturally when one is interested in predicting future behavior of agents based on the observations of his/her past behavior. The key idea of IRL

is to find a reward function such that the distribution of state and action sequences under a (near-)optimal policy with respect to the reward function matches the demonstrated trajectories from an agent [119, 207]. Various estimation methods for IRL including apprenticeship learning [1], maximum entropy IRL [207], Bayesian IRL [133], relative entropy IRL [12], and maximum causal entropy IRL [11], have been developed in the literature. For instance, a broadly used solution to IRL problem [12] proposes a model-free method to find the policy, while neural-network-based reward function [43] can represent more complex expert behaviors. Moreover, IRL can also be viewed as a special case of Generative Adversarial Networks (GANs) [42, 60].

Motivated by solving the drivers' behavioral prediction problem, we develop a joint IRL-DL framework of integrating drivers' history and static features to predict drivers' future behavior. We use RL to formulate each driver's working cycle in a day sorted by time as a sequential decision-making process. The working cycle of each driver consists of different decisions when the driver maximizes his/her total inherent rewards weighted by income and other aspects. Then, we use IRL to learn each driver's decision-making preference vector. Finally, we combine each driver's preference vector with other attributes to build a deep learning regression model (e.g., LTSM-neural network) for prediction. Our contributions can be summarized as follows.

- We are the first to propose a general prediction framework of combining drivers' static features and decision-making preference vector together for prediction.
- By modeling a driver's daily working cycle process as a sequential decision-making problem, we use IRL to learn the driver's preference vector.
- We use a large-scale real-world data set obtained from a ride-sharing platform to show that the use of a preference vector can achieve up to 13% improvement than baseline models in terms of the prediction accuracy on different tasks.

The rest of this chapter is organized as follows. We introduce the driver's behavioral prediction problem and provide a brief overview of data and the architecture of IRL-DL in Section 4.2. The data-preprocessing part is elaborated in Section 4.3. Section 4.4 is to model the driver's working cycle as a Markov Decision Problem, and in Section 4.5, we show how to learn a driver's preference vector. Details of experiments and evaluations are given in Section 4.6, followed by a brief discussion of related works in Section 4.7.

4.2 Overview

In this section, we introduce the drivers' behavioral prediction problem, provide a description of the data set we use, and outline the IRL-DL framework.

4.2.1 Drivers' Behavioral Prediction Problem

Definition 4.2.1 (Working Cycle) *Each driver in a ride-hailing platform can decide when to start to work and when to stop working every day. We define a complete working cycle of every driver at a given day as the time interval from the starting time, denoted as* $Time_{in}$*, to the ending time, denoted as* $Time_{off}$*.*

We introduce an *inter-cycle trajectory* as a working static sequence of multiple working cycles. The inter-cycle trajectory is used to model the long-term behaviors of a driver.

Definition 4.2.2 (Inter-cycle trajectory) *The inter-cycle trajectory of a driver is an ordered sequence, denoted as* $Tr_{inter} = (W_1, \ldots, W_N)$*, where* N *is the total number of working cycles and* $W_i = \{t_{start,i}, t_{end,i}, f_{W_i}\}$ *represents the* $i-$*th working cycle, in which* $t_{start,i}$ *is the start time of* W_i*,* $t_{end,i}$ *is the end time of* W_i*, and* f_{W_i} *is the feature vector of* W_i *for* $i = 1, \ldots, N$*.*

Furthermore, we introduce an *intra-cycle trajectory* consisting of a sequence of all orders that each driver finishes within the same working cycle. We use intra-cycle trajectories to characterize the short-term behaviors of a driver.

Definition 4.2.3 (Intra-cycle trajectory) *The intra-cycle trajectory of a driver's $i-th$ working cycle is an ordered sequence of*

$$Tr_{i,intra} = (O_{i,1}, O_{i,2}, \ldots, O_{i,N_i}),$$

where N_i is the total number of orders in the $i-th$ working cycle, and $O_{i,k} = \{t_{k;start,i}, t_{k;end,i}, f_{O_{i,k}}\}$ represents an order with $t_{k;start,i}$ being the start time of order $O_{i,k}$, $t_{k;end,i}$ being the end time of $O_{i,k}$, and $f_{O_{i,k}}$ being the feature vector of $O_{i,k}$ for $k = 1, \ldots, N_i$.

We use X to denote the vector of all features other than intra-cycle and inter-cycle trajectories. We now formally define the drivers' behavioral prediction problem.

Drivers' Behavioral Prediction Problem Definition Given a driver's other features in X, intra-cycle trajectory TR_{intra}, and inter-cycle trajectory TR_{inter}, we want to predict his/her future behavior (e.g., total online time, income, or finished orders), denoted as G, in the next period T.

4.2.2 Data Description

We use four data types in the data set obtained from a ride-hailing platform including (i) order data; (ii) finance data; (iii) customer service worksheet data; and (iv) driver log data. For consistency, all these data types are aligned with the same time period starting from March 2018 to July 2018.

Order Data. Figure 4.1 shows a typical order in a ride-hailing platform consisting of three key stages including pick-up stage(driving to the origin), waiting stage(waiting for the passenger), and serving stage(sending the passenger to the destination).

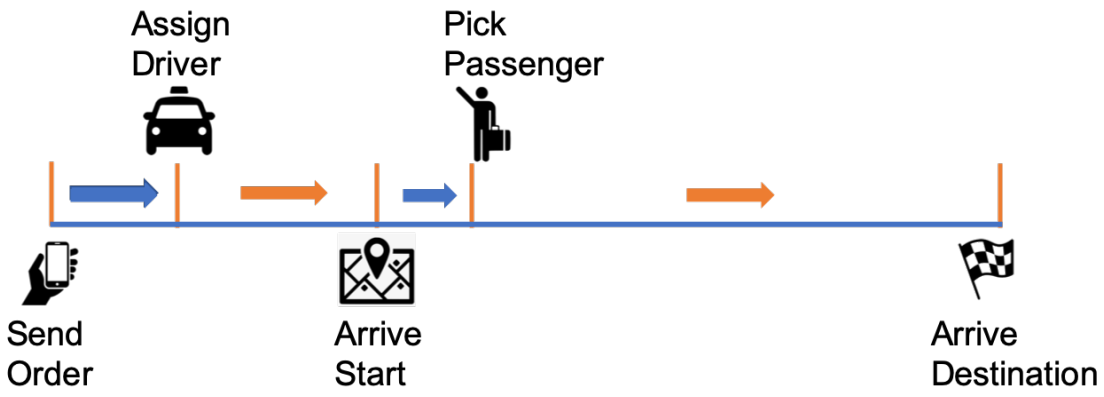


Figure 4.1: Illustration of an order being processed.

Finance Data. We consider various charging variables of each order including customer's payment, driver's cut, driver's incentives, time rate, distance rate, and sometimes long-distance rate.

Customer Service Worksheet Data. We extract some variables from customer service worksheet, such as the number of complaints of each driver received from passengers since drivers with more complaints are more likely to leave the platform.

Driver Log Data. We extract the inter- and intra-cycle trajectories of each driver.

4.2.3 IRL-DL Framework

Figure 4.2 provides an overview of our proposed IRL-DL framework consisting of three main components including data preprocessing, preference learning, and driver prediction.

- **Data Preprocessing.** We first extract drivers' working cycles from their log data. Then, we align order data, finance data, and customer service data together at the order level. Subsequently, we generate the intra-cycle trajectories, inter-cycle trajectories, and additional features of all drivers.

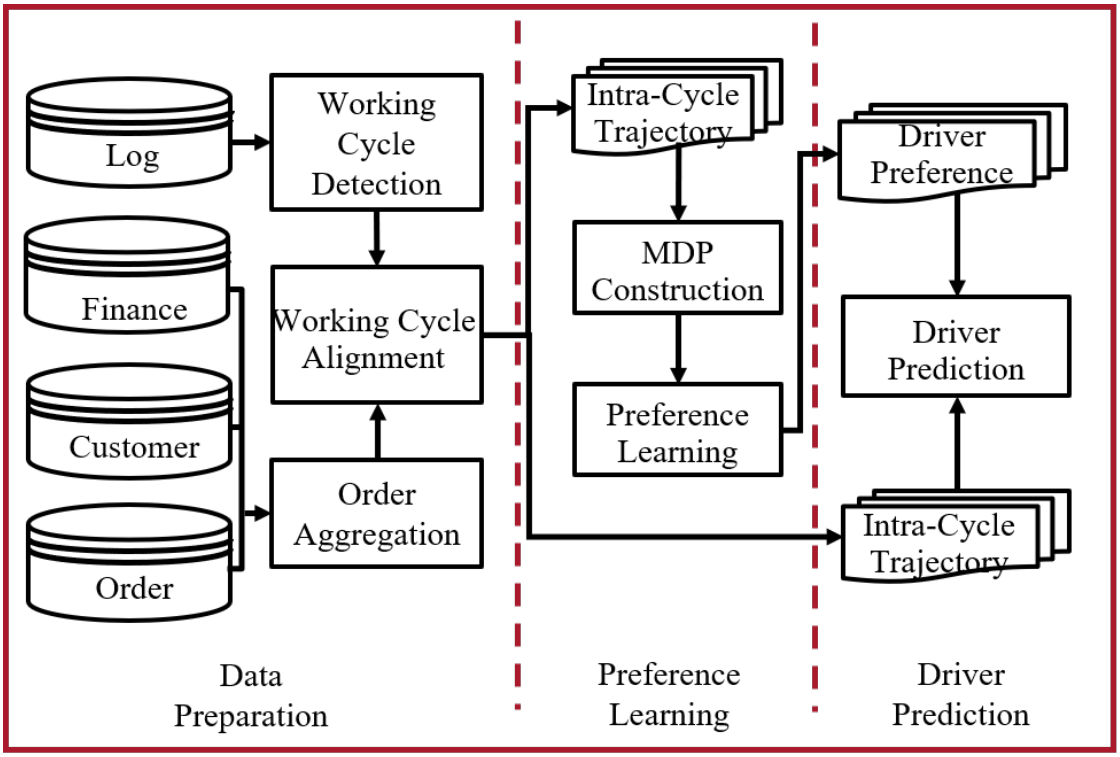


Figure 4.2: The IRL-DL framework for the driver prediction problem

- **Preference Learning.** For each driver, we model the whole working cycle as a Markov Decision Process (MDP) and use IRL to learn a reward function that a driver uses to make various decisions based on intra-cycle trajectories. The reward function represents the preference function that each driver would decide to keep working or log off based on current and future expected rewards.

- **Driver Prediction.** We integrate the preference vector of each driver with all other features to build a predictive model to predict a driver's status.

4.3 Data Preprocessing

In this section, we introduce three key preprocessing steps to generate various features and trajectories.

Table 4.1: Example of Driver Log

Driver	Time	Action
0001	2018-03-20 08:00:35	Log in
0001	2018-03-20 08:03:04	Assigned Order
0001	2018-03-20 08:45:13	Finish Order
...	...	...
0001	2018-03-20 18:54:29	Log off

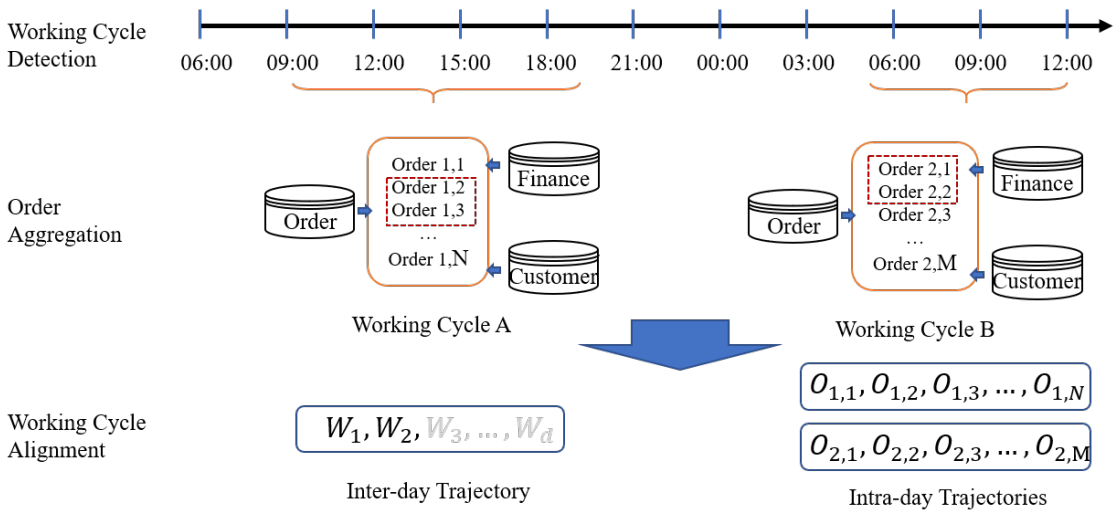


Figure 4.3: An illustration of data preparation process. The process contains 3 parts. The upper time axis indicates two consecutive days. There are two valid log-in/log-off pairs. The first one is from 9am to 19pm in the first day and the second pair is from 5am to 12am in the second day. Then, we generate features for each order and aggregate them by working cycle. The outputs of this process contain both inter-day and intra-day trajectories. For intra-day trajectories, we have 2 trajectories since we have two working cycles.

4.3.1 Working Cycle Detection

Actions of each driver can be viewed as a sequence of actions ordered by time. Table 4.1 illustrates an example of one driver's action log.

The working cycle should start with a log-in action and end with the following log-off action. We show an example in Figure 4.3. As we have stated, we split those working cycles based on the driver's log-in/log-off actions.

Figure 4.4: Illustration of an trip with 3 ExpressPool orders.

4.3.2 Order Aggregation

In the order aggregation process, we mainly have two tasks. The first task is to aggregate features of the same order sitting in different data sources. The second task is to aggregate ExpressPool orders based on which trip they belong to. ExpressPool is a carpool product, whose details will be given later.

Features of one order are in different data sources. Features, such as pick-up distance, driving distance, and other trip-related features are in the ordered dataset. Financial features also include subsidies of drivers in the financial dataset, and one order may have multiple records for different subsidies, such as passenger's coupon and driver's incentive. Other features including customer service features like rating and complaints are in a separate data source, and we align them using order IDs. For some complaints without a particular order ID, we use driver ID and passenger ID to identify, which order those complaints belong to.

The second task is to process ExpressPool orders. We use Figure 4.4 to demonstrate how ExpressPool works. In Figure 4.4, For example, if there is a trip with 3 different orders, where a trip is defined as from assignment of the first ExpressPool order to the finish of the last ExpressPool order. Each passenger would pay fixed fees based on the ExpressPool pricing model. As for the driver, he/she would still get his/her share based on the distance and time he/she drove. In this case, the income of the driver was calculated

based on the travel distance and travel time from picking up the first passenger to dropping off the last passenger. This rule is to guarantee that the driver's income would not decrease even if he/she only gets one order the whole trip (since ExpressPool orders always have a discount). We then should aggregate ExpressPool orders to their corresponding trips. Some features, such as subsidy and income can be added together. However, some features of ExpressPool order should be aggregated based on the pricing strategy. For example, the pick-up distance is the distance to pick up the very first passenger other than the sum of pick-up distances of all 3 orders. After aggregating ExpressPool orders, we also need to aggregate their features from other data sources as mentioned above. For example, in Working Cycle A of Figure 4.3, orders 2 and 3 are ExpressPool Orders, so we have to aggregate their features together.

4.3.3 Working Cycle Alignment

Finally, we can then use working cycles and aggregated orders to generate both intra-cycle and inter-cycle trajectories. For intra-cycle trajectories, we fit aggregated order data into different working cycles of each driver and then sort all orders by using its assignment time. For inter-cycle trajectories, we can calculate features for each working cycle. The features that we used in intra-cycle and inter-cycle trajectories include *Working Status* such as **Total Finished Orders** and **Last Order Feature** like the total income of last order. Just like the example in Figure 4.3, for the two working cycles, we can generate one intra-day trajectory for each working cycle and one inter-day trajectory for all cycles.

4.4 MDP for Working Cycle Modeling

We explain how to use MDP to model the decision-making processes of a driver's working cycle.

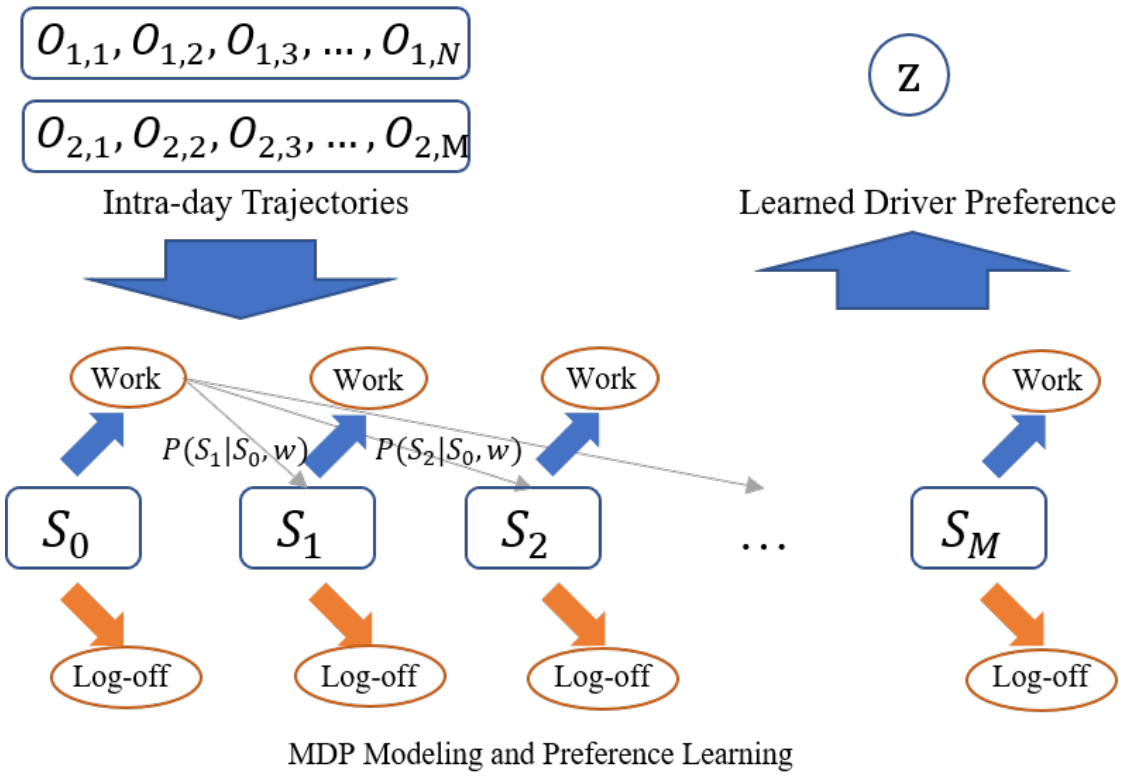


Figure 4.5: An illustration of MDP model for driver working problems: $s_0, \ldots, s_M$ are states and each state has two actions including work and log-off. We use intra-day trajectories to build MDP and learn the preference of each driver using IRL. The output of this process is a preference vector Z for each driver.

We regard each driver as an independent "agent" since each driver has his/her own preference. Each driver uses individual preference to evaluate various decision-making features associated with the current state and each possible action, such as income, working hours, and personal experience [86]. The reward function indicates the driver's preference for different features, and each driver makes his/her own decision to maximize a total inherent reward during the whole working cycle other than only maximizing his/her real income. For example, if a driver could have an unpleasant day, then he/she might decide to stop working. Thus, for long term reward maximization problem, it is natural to use MDP to model driver's decision-making process. Below, we explain how to build an MDP model based on the preprocessed data in Section 4.3.

Agent: We use each driver as a single agent and then collect a certain amount of intra-cycle trajectories, e.g., 3-month data. However, for some part-time drivers, we may not have enough trajectories to learn a robust preference function, so we collect more trajectories until there are at least N intra-cycle trajectories. We set N to be 30, and thus we have at least 30 trajectories for each driver. However, for simplicity, we only draw two trajectories in Figure 4.5.

Action set A: Let T_m be a predefined time interval associated with S_m for $m = 0, 1, \ldots, M$. As illustrated in Figure 4.5, a driver always has two actions during the idle stage including continuing to work and logging off, forming an action set A in our problem. For logging off, we omit temporal log-offs. Naturally, one intra-cycle trajectory would have one log-off action in the end and multiple working actions.

State set S: As shown in Figure 4.5, we divide each day into fixed-length time slots, for example, 10 minutes per time slot. Thus, we have $1,440$ states in total and $M = 1440$. However, since drivers usually have their own pattern and they can work at most 12 hours per working cycle, the state space is not very large.

Transition probability function $P : S \times A \times S \to [0, 1]$. There are two actions in A for each driver. If a driver would decide to log off, then the driver could finish the trajectory. If the driver would continue to work, then it could lead to different result states. Specifically, if the platform does not assign any order to the driver, then it would result in an idle state. Otherwise, the driver transfers to the state represented by the end time of an order. Thus, the transition probability contains two parts including the probability to be assigned an order, denoted as $P_o(s)$, and the distribution of driving time, denoted as $P_d(s'|s, a)$. By combining those two probabilities together, we have $P(s'|s, a) = \{1 - P_o(s)\} + P_o(s)P_d(s'|s, a)$. For example, in Figure 4.5, if the driver logs in at state S_0 and does not get any order assigned to him or her, then the driver will transit to state S_1. Otherwise, the driver will transit to S_2 or other states based on how it

takes the driver to finish the order. However, for some complicated dispatching strategy, $P(s'|s,a)$ may be different across subjects, which makes it very difficult to estimate $P_o(s)$ and $P_d(s'|s,a)$. Therefore, we employ a model-free method to learn the reward function elaborated in Section 4.5.

Reward R: When drivers make decisions on whether to log off, they consider various decision-making features, such as working hours, income, and experience. We will give these decision-making features in the following subsection for more details.

In MDP, $R : S \times A \to \mathbb{R}$ captures the unique personal preference of an agent, which maps decision-making features (at a state s while taking action a) to reward value. These decision-making features include working hours, income, and experience, among others. Such reward function $R(s,a)$ can be inversely learned from intra-cycle trajectory data. In our problem, we assume $R(s, a_{off}) = 0$, indicating that the immediate reward is 0 when a driver decides to log off.

4.5 IRL-DL

In this section, we introduce IRL, describe how to learn driver's preference based on the MDP of driver's working cycle, and integrate driver's preference with other attributes to predict driver's future behavior.

4.5.1 Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL) has been widely used to learn a reward function $R(s,a)$ of an MDP in the past decade. The IRL is to find a reward function $R(s,a)$ such that the distribution of action and state sequences under a (near-)optimal policy with respect to $R(s,a)$ matches with the demonstrated trajectories observed from an agent [119, 207]. A broadly used solution to IRL problem [12] proposes a model-free

method to find the policy, which best represents demonstrated behaviors with the highest entropy, subject to the constraint of matching feature expectations to the distribution of demonstrated trajectories. The reward function $R(s,a)$ is assumed to be a linear function of observed feature $f(s,a)$ at state-action pair (s,a), that is $R(s,a) = \theta^T \cdot f(s,a)$, where θ is a preference vector. Thus, the reward of a trajectory is given by $R(\zeta) = \theta^T f(\zeta)$, where ζ represents a trajectory from MDP and $f(\zeta) = \sum_{(s,a)\in\zeta} f(s,a)$. Then, if some trajectories are collected, we have $\sum_{\zeta\in TR} P(\zeta) f(\zeta) = \widetilde{f}$, where TR is the set of all possible trajectories, $P(\zeta)$ is the probability of ζ being generated by the MDP and $\widetilde{f}$ is an empirical feature expectation based on collected trajectories. We can then calculate the preference vector θ by solving a maximum entropy problem and have $P(\zeta) = e^{\theta^T f(\zeta)} / \sum_{\zeta\in TR} e^{\theta^T f(\zeta)}$. We may calculate the maximum likelihood estimate of θ by using standard gradient decent method with the gradient given by $\widetilde{f} - \sum_{\zeta\in\widetilde{TR}} P(\zeta) f(\zeta)$, where $\widetilde{TR}$ is the set of observed drivers' trajectories.

4.5.2 Driver's Preference Learning

In our problem, $\widetilde{f}$ and $f(\zeta)$ can be easily calculated from observed data. However, it remains an open question how to calculate the probability of trajectory being generated by the MDP constructed by using method mentioned in Section 4.4, namely $P(\zeta)$. From the definition of $P(\zeta)$, we know that,$P(\zeta) = P_0(s_0) \sum_{t=1}^{T} \pi(s_t, a_t) P(s_{t+1}|s_t, a_t)$, where $P_0(s_0)$ is the initial start distribution, $\pi(s_t, a_t)$ is the probability from policy, and $P(s_{t+1}|s_t, a_t)$ is the transition probability, in which s_{t+1} is the next state in ζ. It follows that the transition probability contains two parts including $P_o(s_t)$ and $P_d(s_{t+1}|s_t, a_t)$.

However, it is difficult to estimate the two parts of the transition probability by using observed data. The travel time distribution $P_d(s_{t+1}|s_t, a_t)$ is long-tailed and quite wide due to the presence of occasionally long orders, such as orders to airport or even sometimes to another city. It brings difficulties that wide distribution would increase computa-

tional complexity and the long-tailed part of $P_d(s_{t+1}|s_t, a_t)$ would not be robust to noise. Moreover, the probability of getting an order $P_o(s_t)$ is more difficult to estimate due to dispatching strategy and environment. Thus, model-based methods based on transition probability do not work well for our problem.

We employ Relative Entropy Inverse Reinforcement Learning (REIRL) to learn the preference vector. The REIRL is a model-free method and uses importance sampling to estimate $F = \sum_{\zeta \in \widetilde{TR}} P(\zeta) f(\zeta)$ as follows:

$$F = \sum_{\zeta \in \widetilde{TR}} \frac{U(\zeta)\phi(\zeta)^{-1} e^{\theta^T f(\zeta)}}{\sum_{\zeta' \in \widetilde{TR}} U(\zeta')\phi(\zeta')^{-1} e^{\theta^T f(\zeta')}} f(\zeta), \tag{4.1}$$

where $U(\zeta)$ is a uniform distribution and $\phi(\zeta)$ is the trajectory distribution from a sample policy ϕ, where $\phi(\zeta) = \Pi_{(s,a)\in\zeta}\phi(s, a)$. The brief derivation and the proof of (4.1) can be found in [12].

The gradient $\widetilde{f} - \sum_{\zeta \in \widetilde{TR}} P(\zeta) f(\zeta)$ can be estimated by

$$\begin{aligned} \nabla_\theta L(\theta) =& \widetilde{f} - \sum_{\zeta \in \widetilde{TR}} P(\zeta) f(\zeta) \\ =& \widetilde{f} - \sum_{\zeta \in \widetilde{TR}} \frac{U(\zeta)\phi(\zeta)^{-1} e^{\theta^T f(\zeta)}}{\sum_{\zeta' \in \widetilde{TR}} U(\zeta')\phi(\zeta')^{-1} e^{\theta^T f(\zeta')}} f(\zeta). \end{aligned} \tag{4.2}$$

Applying the IRL algorithm, we can learn the preference vector θ.

To make preferences across models of different drivers comparable, we use *z-score* [89] normalization. Higher/lower z-score on the k-th feature, denoted as z_{ik}, means that driver i has strong preference on the k-th feature. The use of z-score in our problem has two major advantages. First, we can compare different drivers' preferences via their z-scores. Second, the elements of z_i would most likely lie between $[-3, 3]$, but those of θ_i could be any real value. This shows the advantage of training neural networks. The

z-score can be calculated as $z_i = \text{diag}(H_i^{-\frac{1}{2}}) \odot \theta_i$, where i indicates the i-th driver, H_i^{-1} is the inverse of Hessian Matrix of θ_i, diag$(\cdot)$ means diagonal elements of a matrix, and we use $\odot$ to denote element-wise multiplication. Furthermore, we have

$$\begin{aligned} H_i =& \frac{\partial^2 L(\theta_i)}{\partial^2 \theta_i} \\ =& \left[\sum_{\zeta} P(\zeta) f_i(\zeta) \left(f_j(\zeta) - \sum_{\zeta'} P(\zeta') f_j(\zeta') \right) \right], \end{aligned} \tag{4.3}$$

where $f_i(\zeta)$ and $f_j(\zeta)$ means the i-th and j-th elements of $f(\zeta)$, respectively. And $P(\zeta)$ can be estimated using importance sampling. After acquiring the z-score z_i of each driver, we can then build up a model to predict drivers' future behavior.

4.5.3 Regression Models for Drivers' Behavioral Prediction

We elaborate more on regression models and features that we use to predict drivers' behavior, such as total online time, income and finished orders. Figure 4.6 illustrates the general LatentCross [9] framework of the neural network for doing driver's prediction. It is the Driver Prediction part in Figure 4.2. The first component is the trajectory embedding model, which in our case we use PhasedLSTM [116] as an example, but any model dealing with sequential data can be plugged in the sequential layer. As shown in Figures 4.2 and 4.6, this part takes Inter-day trajectories, which are the output of data preparation process demonstrated in Figure 4.3, as input and produces an trajectory embedding. The right component of Figure 4.2 is to take the driver preference that we learn in the preference learning phase as input. Then, we apply several dense layers to add non-linearity. As stated above, the input is z_i rather than the raw preference vector θ_i. Then, we use the element-wise product layer to combine trajectory embedding and driver's preference, which is the same as LatentCross, in order to capture two-way relationship between the hidden state and each context feature [9]. Finally, a dense layer is added to produce final

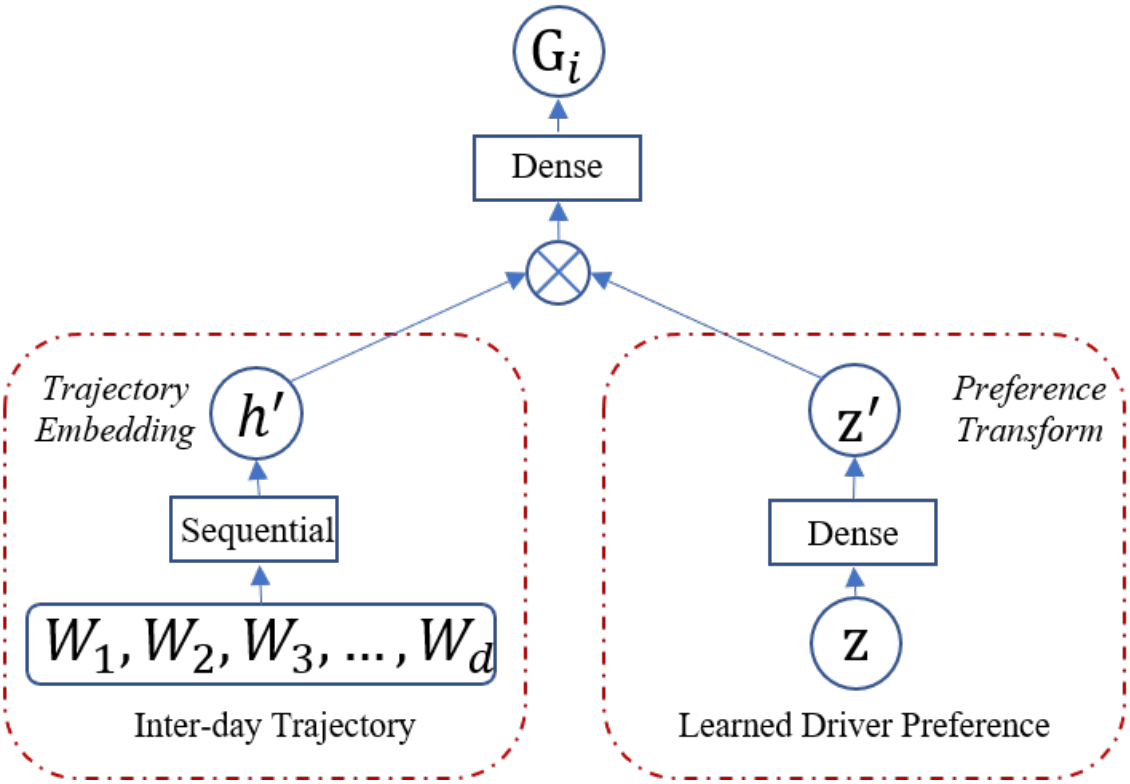


Figure 4.6: Illustration of the network structure of IRL-DL. The left component takes inter-day trajectory as input to feed into a sequential layer and the output is r h'. The right part takes driver preference Z as input and the output is denoted as z'.

prediction.

Algorithm 4 shows how to predict a driver's behavior for a trained prediction model. First, we collect intra-cycle trajectories from multiple data sources that we described in Section 4.2. Then, we construct a working-cycle MDP based on the observed intra-cycle trajectories and we can use MDP and IRL to learn driver's preference vector. Finally, we combine the inter-cycle trajectory, driver's preference vector, and other attributes to predict driver's behavior. Furthermore, for a new driver, without historical behavior data, we return the mean prediction of all drivers as predictive value.

Algorithm 4 Driver Prediction

1: **INPUT:** Driver ID D_i;
2: **OUTPUT:** Some predicted status of D_i, denoted as G_i;
3: Collect intra-cycle trajectory set TR_{intra} of D_i;
4: Build Markov Decision Process using TR_{intra};
5: Learn the preference vector θ_i via Relative Entropy Inverse Reinforcement Learning;
6: Extract inter-cycle trajectory set TR_{inter} for driver D_i's historical behaviors;
7: Use trained PhasedLSTM model to predict G_i based on the input of θ_i and TR_{inter} using the LatentCross Framework in Figure 4.6;
8: Return G_i

4.6 Evaluation

In this section, we use extensive experiments to verify the effectiveness and efficiency of our proposed IRL-DL for drivers' income.

4.6.1 Experiments Setting

We extracted data from $58,160$ drivers starting from March, 2018 to July, 2018 in a city. Data from March to May were used to train, and data in June and July were used as test data. First, we collected drivers' two-month intra-cycle trajectories to learn drivers' preference and then used 30-day long inter-cycle trajectories and drivers' preference to build prediction models, in which the target value is driver's total income in the next 30 days. To predict income, we use driver's total income in the 30 days of July as the ground truth. Then, intra-cycle trajectories generated in June were used to learn driver's individual preference and the prediction was made based on driver's preference and 30-day long inter-cycle trajectories. Since the accurate prediction of when a driver would log off indicates that the learned preference vector may indicate how drivers make his/her log-off decision, we used AUC score to evaluate the accuracy of log-off action prediction accuracy for preference learning.

As for driver's income prediction, Mean Absolute Error (MAE) is used as the evalua-

tion metric. For information safety, we normalize the actual response by dividing by their average. We compare our model with three sets of baseline models.

- **Shallow Models.** We compare our method with two types of models. The first one is the BG-NBD model, in which we denote drivers' working days as "purchase" and their earnings as "monetary value" and build a RFM model to do drivers' behavioral prediction. Moreover, instead of using LSTM and neural network, we generate statistical features in various time intervals and feed those features into a shallow model, such as XGBoost, to predict each driver's income.

- **Models Without Driver Preference.** In those models, we omit driver's preference and only use driver's historical behaviors to predict driver's income. We also use RFM features extracted from the BG-NBD model to train those models.

- **Trajectory Embedding Component.** We also compare multiple sequential models including RNN, LSTM, and Phased-LSTM. Moreover, any other sequential model can be added into our IRL-DL framework since the goal of this chapter is that the inclusion of additional decision-making preference can be combined with any trajectory embedding models in order to achieve better prediction accuracy.

4.6.2 Preference Learning

We evaluate the effectiveness of our preference learning model. Figure 4.7 shows the distribution of AUC scores with the y-axis being the number of driver. It indicates that for most drivers, preference vector learned can accurately predict whether a driver would log-off. However, for a few number of drivers, we do not have accurate prediction due to the lack of sufficient records. See Figure 4.8 for the relationship between AUC score and the size of records. Each point in Figure 4.8 indicates one individual driver. We also verify that the preference vector we learn is correlated with driver's future behavior. Specifically,

Table 4.2: Online time prediction error (MAE)

	w/o RFM w/o Pref.	w RFM w/o Pref.	w/o RFM w Pref.	w RFM w Pref.
BG-NBD	N/A	0.5953	N/A	N/A
XGB	0.4532	0.4166	0.3898	0.3890
RNN-30	0.3883	0.3703	0.3258	0.3257
RNN-60	0.3681	0.3642	0.3146	0.3140
LSTM-30	0.3574	0.3510	0.3105	0.3103
LSTM-60	0.3430	0.3413	0.2957	0.2956
PLSTM-30	0.3502	0.3419	0.3004	0.2993
PLSTM-60	0.3483	0.3371	0.2933	0.2930

we use the K-Means algorithm to split drivers into five different clusters based on their preferences. Figure 4.9 shows the distribution of future online time in the next 30 days for each cluster, revealing that different clusters correspond to different online time distributions. It may further indicate that drivers' preferences define their working patterns, contributing to their future performance. In summary, driver's preference is accurate and of great potential value for the drivers' behavioral prediction problem.

4.6.3 Drivers' Behavioral Prediction

We conduct experiments to show that our proposed framework can increase the prediction performance in a variety of tasks.

Algorithm Efficiency. We show the computational effectiveness of our proposed framework. Table 4.2 shows 29 different models that we used. The first one is the BG-NBD model, which has the poorest performance. The second four models are based on XGBoost. We also consider RNN, LSTM and PhasedLSTM models with different numbers of output size in the trajectory embedding layer as well as the output dimension of preference layer.

We have the following observations. First, the use of driver's preference can reduce prediction error for more than 10%. Second, deep models, such as RNN and LSTM,

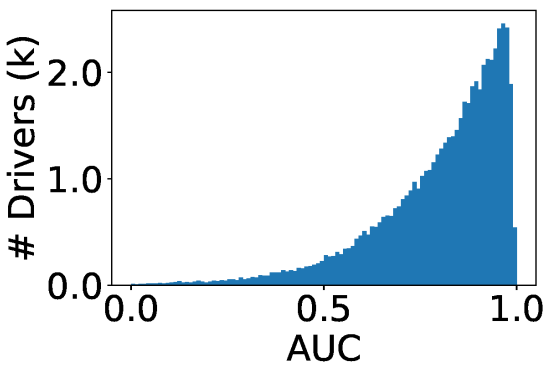


Figure 4.7: Number of drivers vs AUC score.

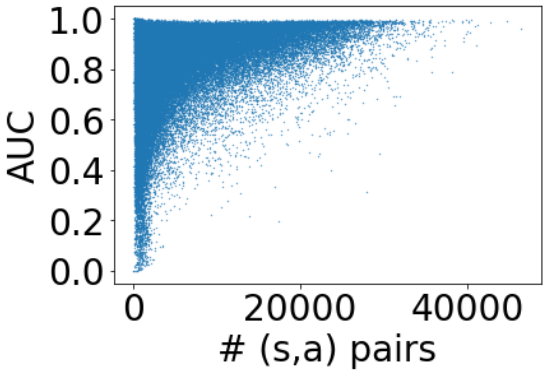


Figure 4.8: AUC score vs number of (s, a) pairs collected

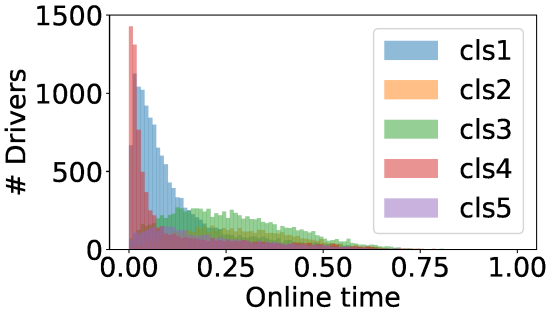


Figure 4.9: Online time distribution of different group of driver

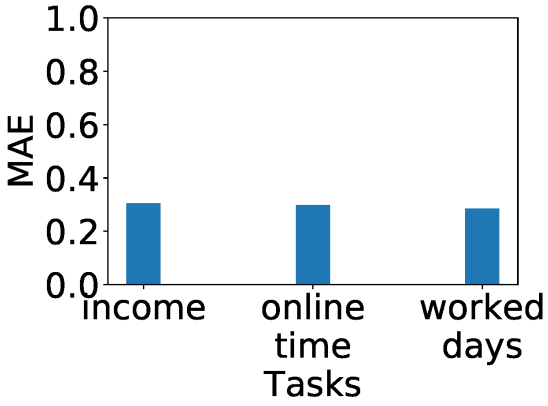


Figure 4.10: MAE on different tasks.

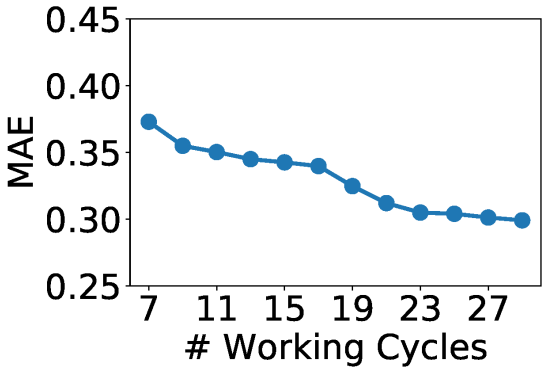


Figure 4.11: MAE vs amount of data.

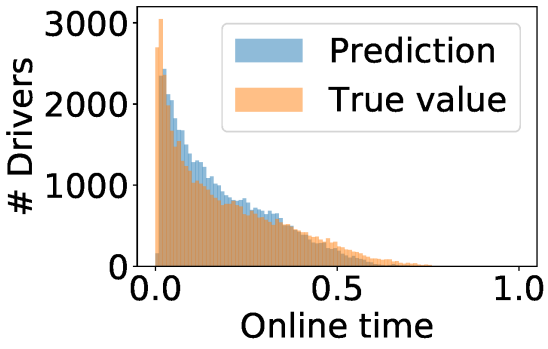


Figure 4.12: Prediction distribution of Task.

outperform the XGBoost model. Third, the benefit of increasing network complexity from 30 to 60 is not significant enough. Therefore, without loss generality, we fix output

size at 30. Fourth, the RNN model without driver preference can outperform XGBoost even with the use of driver preference. Fifth, among all models, PhasedLSTM with the use of driver preference can achieve the best performance since PhasedLSTM can capture the long-term effect of temporal sequences. Therefore, PhasedLSTM would be used as a default option in the following evaluation. Also, for shallow models, RFM features can slightly improve prediction accuracy solely. Also, for deep models like LSTM and PhasedLSTM, the contribution of using recency, frequency and monetary values is pretty marginal, since the RFM related pattern can be learned by sequential models.

Different Tasks. We consider additional tasks including predicting total online time, driver's average income per day, and driver's worked days in the next month. The results in Figure 4.10 indicate that our model performs well in all tasks. Therefore, preference vector may represent driver's behavior in different aspects.

Amount of Data. We investigate the amount of data that we need. In this experiment, we choose drivers' total online time in the following 30 days as target response. The result in Figure 4.11 shows the prediction error (MAE) versus the number of working cycles that we used to train the model. The results indicate that MAE is essentially the same even with more than 23 working cycles. We can also reduce the randomness of data by including more observations. This randomness may come from two resources. The first one is the environment randomness and the second one is that drivers may exhibit certain randomness.

Prediction Illustration. We choose one task to illustrate the prediction result obtained from the proposed framework. Figure 4.12 presents the actual distribution of driver's online time and the predictive distribution. We have the following observations. First, the two distributions are very similar to each other. Specifically, the KL-divergence of those two distributions is equal to 0.1503. However, if we aggregate all drivers together, then their behaviors would be more predictable and the prediction performance is

better. Second, the true distribution of A is more long-tailed compared with the prediction distribution of A. It indicates that our prediction model may make a few conservative predictions since the long-tailed part is often caused by rare cases.

4.7 Related Work

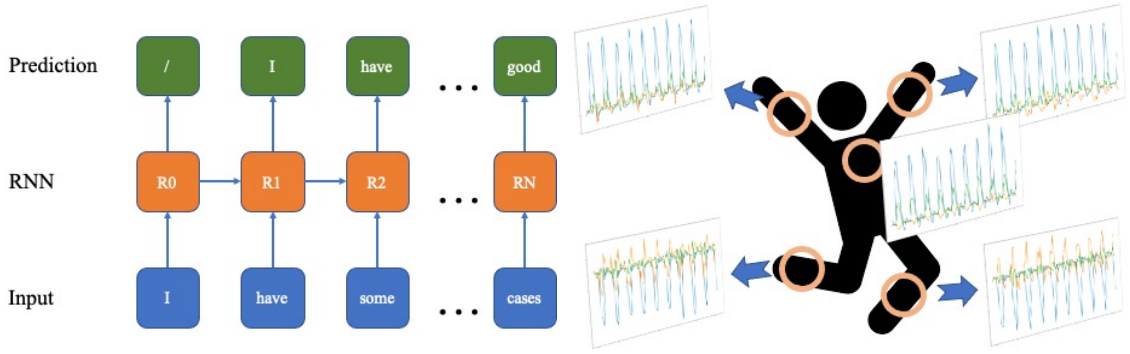
In this section, we summarize the literature works in two related areas to our study: 1) inverse reinforcement learning, and 2) user choice modeling. Learning reward function $R(s, a)$ of an MDP is a problem which has been broadly studied in the past decade, which is called Inverse Reinforcement Learning. The inverse reinforcement learning problem (IRL) is to find a reward function $R(s, a)$, such that the distribution of action and state sequences under a (near-)optimal policy with respect to $R(s, a)$ matches the demonstrated trajectories from an agent [119, 207]. A broadly used solution to IRL problem [12] proposes a model-free method to find the policy. Besides, neural-network-based reward function [43] is considered, which can represents more complex expert behaviors. And the Inverse Reinforcement Learning can be viewed as a special application of GAN [42, 60]. Also, IRL is used to model human's sequential decision-making process and is used to predict human behaviors in urban transit system [164]. Inspiring by these work, we also use Inverse Reinforcement Learning algorithm to extract driver's preference vector. User choice modeling has been extensively studied in the literature with applications, which investigate how users make decisions in various application scenarios. For examples, in [144], they use random utility maximization and random regret minimization to analyze users' choice on park-and-ride lots. In [207], the authors propose a probabilistic approach to discover reward function for which a near-optimal policy closely mimics observed behaviors. However, differing from these works, we employ data-driven approaches to study the unique decision-making process of urban public transit passengers.

Chapter 5

Training Stability in Learning Recurrent Models

5.1 Introduction

Thanks to the fast development of mobile sensing technologies, large amounts of sequential data are being collected rapidly, such as vehicle GPS records, stock prices, sensor data from air quality detectors, etc. In these sequential datasets, the points in a sequence are dependent on the other points in the dataset. In practice, there are many applications in mining these sequential data. For example, *Natural Language Processing (NLP)* [34] aims to predict or guess the next word for us (Fig 5.1(a)), and *human activity recognition (HAR)* [84, 2] predicts or classifies human activities from sequential data collected from IoT wearable devices, such as accelerometers and gyroscopes (Fig 5.1(b)). Recurrent neural networks (RNNs) have achieved significant success in tackling these applications, i.e., learning complex patterns for sequential input data. At each time step t, an RNN stores the previous hidden state vector, $\mathbf{h}_{t-1} \in \mathbb{R}^D$, and upon receiving the current input vector, $\mathbf{x}_t \in \mathbb{R}^d$, linearly transforms the tuple $(\mathbf{h}_{t-1}, \mathbf{x}_t)$ and passes it through a non-linearity



(a) Language modeling. (b) Human activity prediction.

Figure 5.1: Illustration of various sequential tasks. For (a)language modeling task, we use sentence as input of recurrent models to predict the next word. As for (b)human activity prediction, the figure shows sensor data from IoT wearable devices on body, arms and legs respectively when the user jumps.

to update the state vectors over T time steps. Subsequently, RNNs output the predictions as a function of the hidden states. The model parameters (*i.e.,* state/input/prediction parameters) are learned by minimizing an empirical loss.

In the literature, there are significant amount of works on developing RNNs such as, just to name a few, long short-term memory (LSTM) [61], gated recurrent unit (GRU) [23], UGRNN [26], FastGRNN [84], unitary RNNs [4, 72, 188, 113, 130], deep RNNs [127, 208, 114], linear RNNs [13, 90, 6], residual/skip RNNs [70, 8, 18, 15, 84], ordinary differential equation (ODE) based RNNs [152, 121, 16, 84, 19, 138, 73].

Continuous-Time RNN (CTRNN). [135] introduced continuous-time RNN to mimic activation propagation in neural circuitry, which is modeled as follows:

$$\beta \dot{\mathbf{g}}_t = -\alpha \mathbf{g}_t + \phi(\mathbf{U}\mathbf{g}_t + \mathbf{W}\mathbf{x}_t + \mathbf{b}), \tag{5.1}$$

where at the t-th time step, $\mathbf{x}_t \in \mathbb{R}^d$ denotes the input signal, $\mathbf{g}_t \in \mathbb{R}^D$ denotes the hidden state vector, $\dot{\mathbf{g}}_t$ denotes the change rate of the vector $\mathbf{g}_t$, ϕ denotes the activation function parametrized by $\mathbf{U} \in \mathbb{R}^{D\times D}$, $\mathbf{W} \in \mathbb{R}^{D\times d}$, $\mathbf{b} \in \mathbb{R}^D$, and $\alpha, \beta \in \mathbb{R}^+$ denote some

predefined constants.

Incremental RNN (iRNN). Inspired by CTRNN, [73] introduced an incremental RNN whose hidden-state transition function is defined as follows:

$$\begin{aligned} &\mathbf{z}_t = \mathbf{g}_t + \mathbf{h}_{t-1}, \\ &\beta \dot{\mathbf{g}}_t = -\alpha \mathbf{z}_t + \phi(\mathbf{U}\mathbf{z}_t + \mathbf{W}\mathbf{x}_t + \mathbf{b}), \mathbf{g}_t(0) = \mathbf{0}, \mathbf{h}_t = \mathbf{g}_t^*, \end{aligned} \tag{5.2}$$

where $\mathbf{g}_t(0)$ denotes the initial value for $\mathbf{g}_t$, $\mathbf{g}_t^*$ denotes an equilibrium point of the ODE, if any, and $\alpha, \beta \in \mathbb{R}^+$ are learnable through iRNN training. Then by using Euler's method to discretize the ODE in Eq. 5.2, we can further rewrite the equation as

$$\begin{aligned} &\mathbf{z}_t(k) = \mathbf{g}_t(k) + \mathbf{h}_{t-1}, \\ &\mathbf{g}_t(k+1) = \mathbf{g}_t(k) + \eta_t^k(\phi(\mathbf{U}\mathbf{z}_t(k) + \mathbf{W}\mathbf{x}_t + \mathbf{b}) - \alpha \mathbf{z}_t(k)), \\ &\mathbf{g}_t(0) = \mathbf{0}, \mathbf{h}_t = \mathbf{g}_t(K), k \in [K-1], \end{aligned} \tag{5.3}$$

where $\alpha, \eta_t^k \in \mathbb{R}^+$ are some learnable parameters. [73] proved that under mild conditions $\mathbf{g}_t(k)$ converges linearly and $\lim_{K\to\infty} \mathbf{g}_t(K) = \mathbf{g}_t^*$, if any equilibrium exists. Note that Eq. 5.3 is used for implementation.

Training Stability Challenge of RNNs. Vanishing and exploding gradients often occur in training RNNs, due to the repeatability of network weights in the chain rule when computing gradients, leading to instability in training with their magnitude either too small or too large. It has been proven that theoretically there is no vanishing/exploding gradient in the training of iRNN, leading to fast convergence empirically. In the literature, some other RNNs have analyzed and demonstrated their training stability as well, such as AntisymmetricRNN [16], LipschitzRNN [40], MomentumRNN [120], nnRNN [74], expRNN [92]. In particular, the stability analysis often comes with the eigenvalues of the Jacobian

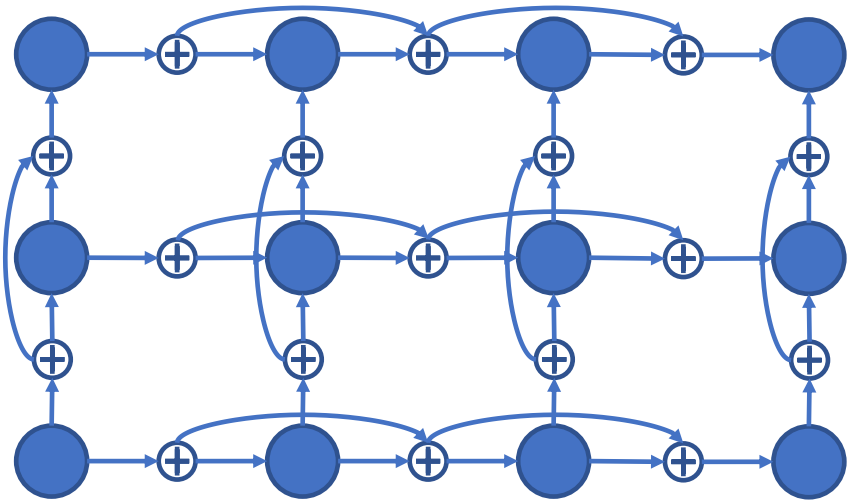


Figure 5.2: Illustration of the network architecture in our DIRNN with $L = 3$ and $T = 4$. Here each blue node represents a neuron in the network that a hidden state vector is generated by an ODE.

of the hidden state dynamics in order to study the problem of vanishing/exploding gradients. Recently, [134] proposed analyzing RNN training using attractors and smoothness as alternatives.

Lyapunov Stability in Dynamical Systems. RNNs are often considered as dynamical systems, and several derivatives are developed from this perspective, such as iRNN and AntisymmetricRNN. However, there are few works that borrow the theory of dynamical systems to analyze the RNN training stability (without strong assumptions on the Jacobian). Recently, [39] studied the Lyapunov spectra of chaotic recurrent neural networks. [162] proposed using Lyapunov exponents to understand the information propagation in RNNs, but unfortunately there is no discussion on how to introduce such nice Lyapunov stability into the development of RNNs. [36] proposed viewing neural networks from a dynamical systems perspective as pointwise affine maps. However, the theoretical results are adapted from dynamical system analysis and the assumptions for deep neural networks are too strong to be met in practice. [159] proposed an ODE based network implementation to guarantee stability as well as incorporating prior knowledge.

Deep (Stacked) RNNs. By stacking the conventional shallow RNNs such as LSTM, GRU, or iRNN, we can generate deep RNNs whose hidden states depend on the previous states along not only time steps but also network layers. In particular, in this work we consider the deep RNN as illustrated in Fig. 5.2. It is well-known that deep RNNs require a considerable amount of work (such as learning rate and clipping) to ensure proper convergence. In other words, the training stability of deep RNNs can be achieved empirically with careful initialization, but its theoretical property still remains elusive.

Our Contributions. In this chapter, we study the training stability of deep RNNs from the perspective of Lyapunov stability, and propose a novel *deep incremental RNN* (DIRNN) that is a generalization of iRNN into the deep regime.

- *Theoretical contributions:* We prove that our DIRNN is essentially a Lyapunov stable dynamical system with a set of equilibrium points, each for a hidden state. We then prove the training stability of DIRNN where there exists no vanishing/exploding gradient. To the best of our knowledge, we are the *first* to provide such theoretical results on the training stability for deep RNNs.
- *Empirical contributions:* The model sizes of deep RNNs grow linearly with the increase of the network depth, which may limit the applications of DIRNN in reality. To address this problem, we propose a sparsified DIRNN, namely *TinyRNN*, that significantly reduces the number of parameters with marginal accuracy loss. We evaluate our approach on seven benchmark datasets and demonstrate state-of-the-art performance.

5.2 Related Work

RNN Architectures. LSTM [61] applies gate-controlled memory cells to mitigate the vanishing/exploding gradient issue in sequence-based tasks. Another widely-used variant

of RNNs is GRU [23]. Both LSTM and GRU are developed through sophisticated recurrent units [25]. In particular, FastGRNN [84] feed-forwards state vectors to induce skip or residual connections, to serve as a middle ground between feed-forward and recurrent models, and to mitigate gradient decay. ShaRNN [32] was proposed to induce long-term dependencies and yet admit parallelization, where the first layer splits the input sequence and runs several independent RNNs, and the second layer consumes the output of the first layer using a second RNN thus capturing long term dependencies. iRNN [73] was proposed based on a novel ODE based formulation to facilitate the training of RNNs by achieving identity gradient between hidden layers and low-rank matrix decomposition. Unitary and orthogonal RNNs aim to preserve the norm of hidden features by controlling the eigenvalues, explicitly or implicitly, that has been studied extensively in recent years [4, 72, 188, 113, 130, 40, 74, 92, 56, 111]. For instance, [92] proposed expRNN by performing the first-order optimization with orthogonal and unitary constraints based on a parametrization stemming from Lie group theory through the exponential map.

Deep RNNs. RNNs are inherently deep in time. Inspired by this property, researchers are seeking to develop new networks to investigate the benefits of depth in space of RNN architectures. For instance, [49] combined multiple recurrent levels on the basis of bi-directional LSTM [50, 143] to improve RNN performance in speech recognition task. Another study in [58], with a deeper analysis of the different emergent time scales, also proposed a similar stacking architecture. [95] proposed IndRNN where neurons in the same layer are independent of each other and they are connected across layers. Other deep RNNs have been proposed in the literature as well [20, 38, 41, 141, 48, 69, 127, 131].

Lightweight RNNs. Recently, lightweight RNNs, *i.e.,* RNNs with small model sizes, have been attracting more and more attention due to the great potentials in both academic research and industrial products [73, 32, 107, 80, 115, 137, 84]. Such RNNs can be loaded on mid-end and more powerful high-end edge devices such as smartphones, and have been

introduced into some real-world applications such as machinery fault diagnosis [107]. In particular, FastGRNN [85] employed gated techniques and fixed-point quantization to further compress the models learned by FastRNN. [157, 158] proposed hybrid matrix decomposition (HMD) and Kronecker product (KP) approaches, respectively, for RNN compression on edge devices. Both approaches belong to low-rank tensor decomposition. DirNet [189] was proposed based on an optimized fast dictionary learning algorithm to compress RNNs on mobile devices.

5.3 DIRNN: Deep Incremental RNN

Motivation. iRNN [73] introduced an interesting notion of "identity" gradient in back-propagation. That is, given two arbitrary hidden state vectors $\mathbf{h}_t, \mathbf{h}_{t-1}, (t \in [T])$, it holds that $\partial \mathbf{h}_t + \partial \mathbf{h}_{t-1} = \mathbf{0}$ where ∂ denotes the partial derivative operator. Letting τ denote the continuous time for measuring the change rate, we then can achieve the following ODE:

$$\dot{\mathbf{h}}_t(\tau) + \dot{\mathbf{h}}_{t-1}(\tau) \equiv \frac{\partial \mathbf{h}_t(\tau)}{\partial \tau} + \frac{\partial \mathbf{h}_{t-1}(\tau)}{\partial \tau} = \mathbf{0}, \forall t \in [T]. \tag{5.4}$$

Now if we take $\mathbf{h}_t, \mathbf{h}_{t-1}$ as variables, then such linear dynamical systems are Lyapunov stable (see Def. 5.3.1). This novel perspective motivates us to study the training stability for deep RNNs.

Formulation. In order to prove the Lyapunov stability, we intentionally propose the following ODE based hidden-state transition function for our DIRNN with L hidden layers:

$$\begin{aligned}
&\mathbf{z}_{l,t} = \mathbf{g}_{l,t} + \sum_{m=1}^{l-1} \gamma_{m,t} \mathbf{h}_{m,t} + \sum_{n=1}^{t-1} \rho_{l,n} \mathbf{h}_{l,n}, \qquad (5.5)\\
&\beta_l \dot{\mathbf{g}}_{l,t} = -\alpha_l \mathbf{z}_{l,t} + \phi(\mathbf{U}_l \mathbf{z}_{l,t} + \mathbf{W}_l \mathbf{x}_t + \mathbf{b}_l),\\
&\mathbf{g}_{l,t}(0) = \mathbf{0}, \mathbf{h}_{l,t} = \mathbf{g}^*_{l,t}, \forall l \in [L], \forall t \in [T],
\end{aligned}$$

where $\mathbf{g}_{l,t}(0)$ denotes the initial value for $\mathbf{g}_{l,t}$, $\mathbf{g}^*_{l,t}$ denotes an equilibrium point of the ODE, if any, and $\alpha_l, \beta_l \in \mathbb{R}^+, \gamma_{m,t}, \rho_{l,n} \in \mathbb{R}$ are learnable through DIRNN training. Compared with Eq. 5.2, we can see that the key difference between DIRNN and iRNN lies in the auxiliary variable $\mathbf{z}_{l,t}$ where DIRNN involves additional stacked hidden state vectors with suitable weights. This formulation corresponds to the network architecture in Fig. 5.2.

5.3.1 Stability Analysis

Definition 5.3.1 (Lyapunov Stability [140]) *Consider a nonlinear dynamical system $\dot{x} = f(x(\tau))$, $x(0) = x_0$ with the system state vector $x(\tau) \in \mathcal{D} \subseteq \mathbb{R}^n$ and a continuous function $f : \mathcal{D} \to \mathbb{R}^n$. Suppose that f has an equilibrium at x^* so that $f(x^*) = 0$, then this equilibrium is said to be* Lyapunov stable*, if for every $\epsilon > 0$, there exists a $\delta > 0$ such that, if $\|x(0) - x^*\| < \delta$, then for every $\tau \geq 0$ we have $\|x(\tau) - x^*\| < \epsilon$.*

In other words, if the solutions that start near an equilibrium point x^* stay near x^* forever, then x^* is Lyapunov stable.

Proposition 5.3.2 ([16]) *The solution of an ODE is (Lyapunov) stable if $\max_{i=1,\cdots,n} Re(\lambda_i(\mathbf{J}(t))) \leq 0, \forall t \geq 0$, where $\mathbf{J}(t) \in \mathbb{R}^{n\times n}$ denotes the Jacobian matrix of function f, $\lambda_i(\cdot)$ denotes the i-th eigenvalue, and $Re(\cdot)$ denotes the real part of a complex number.*

Lemma 5.3.3 *Consider the change rates of hidden states $\mathbf{h}$ over continuous time τ. Then DIRNN satisfies*

$$\dot{\mathbf{z}}_{l,t} = \dot{\mathbf{h}}_{l,t} + \sum_{m=1}^{l-1} \gamma_{m,t} \dot{\mathbf{h}}_{m,t} + \sum_{n=1}^{t-1} \rho_{l,n} \dot{\mathbf{h}}_{l,n} = \mathbf{0}, \forall l, \forall t. \tag{5.6}$$

Proof: By plugging the equilibrium point into the ODE, we have $-\alpha_l \mathbf{z} + \phi(\mathbf{U}_l \mathbf{z} + \mathbf{W}_l \mathbf{x}_t + \mathbf{b}_l) = \mathbf{0}$. Now by taking the derivative *w.r.t.* τ, we have $\alpha_l \dot{\mathbf{z}}_{l,t} = \left[\nabla \phi \cdot \mathbf{U}_l^T\right] \dot{\mathbf{z}}_{l,t}$

that holds for all possible $\mathbf{x}_t$, leading to $\dot{\mathbf{z}}_{l,t} = \mathbf{0}$. Here ∇ denotes the gradient operator and $(\cdot)^T$ denotes the matrix transpose operator. We now complete our proof. ■

Theorem 5.3.4 (Stability of Hidden States) *We define* $\mathbf{s} = \sum_{l,t} \mathbf{z}_{l,t} = \sum_{l,t} \theta_{l,t}\mathbf{h}_{l,t}$ *where* $\theta_{l,t} \in \mathbb{R}, \forall l, \forall t$ *denotes the combination weights. Then it holds in DIRNN that*

$$\dot{\mathbf{s}}(\tau) = \mathbf{0}, \forall \tau \geq 0, \tag{5.7}$$

leading to a Lyapunov stable dynamical system.

Proof: We can achieve Eq. 5.7 based on Eq. 5.6 in Lemma 5.3.3. This is equivalent to $\dot{\mathbf{s}} = \mathbf{0} \cdot \mathbf{s}$ whose Jacobian is $\mathbf{0}$ where all the eigenvalues are zeros. Then based on Prop. 5.3.2, we can complete our proof. ■

Theorem 5.3.5 (Training Stability) *Assuming* $\theta_{l,t} \neq 0, \forall l, \forall t$ *in Thm. 5.3.4, then it holds in DIRNN that* $\left\| \frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}} \right\|_F$ *is constant in time, leading to no vanishing or exploding gradient. Here* $\| \cdot \|_F$ *denotes the Frobenius norm of a matrix, respectively.*

Proof: Base on Thm. 5.3.4, we have $\frac{\partial}{\partial \tau} \left(\frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}} \right) = \frac{\partial \dot{\mathbf{s}}(\tau)}{\partial \mathbf{h}_{l,t}} = \mathbf{0}, \forall l, \forall t$. Then based on Def. 5.3.1, the magnitude of $\frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}}$ along any direction is constant in time, and thus $\left\| \frac{\partial \mathbf{s}(\tau)}{\partial \mathbf{h}_{l,t}} \right\|_F$ is constant in time. If there were vanishing gradients over all the hidden states (*i.e.,* slow update), the F-norm would be arbitrarily small. If there were exploding gradients over at least one hidden state (*i.e.,* fluctuation), the F-norm would be arbitrarily big. We now complete our proof by the contradictions of both cases to the constant F-norm.

■

5.3.2 Implementation

Similar to iRNN in Eq. 5.3, we apply Euler's method to compute the equilibrium point for each hidden state, *i.e.,*

$$\begin{aligned}
&\mathbf{z}_{l,t}(k) = \mathbf{g}_{l,t}(k) + \sum_{m=1}^{l-1} \gamma_{m,t}\mathbf{h}_{m,t} + \sum_{n=1}^{t-1} \rho_{l,n}\mathbf{h}_{l,n}, \qquad (5.8)\\
&\mathbf{g}_{l,t}(k+1)\\
&= \mathbf{g}_{l,t}(k) + \eta_{l,t}^k(\phi(\mathbf{U}_l\mathbf{z}_{l,t}(k) + \mathbf{W}_l\mathbf{x}_t + \mathbf{b}_l) - \alpha_l\mathbf{z}_{l,t}(k)),\\
&\mathbf{g}_{l,t}(0) = \mathbf{0}, \mathbf{h}_{l,t} = \mathbf{g}_{l,t}(K), k \in [K-1], \forall l \in [L], \forall t \in [T].
\end{aligned}$$

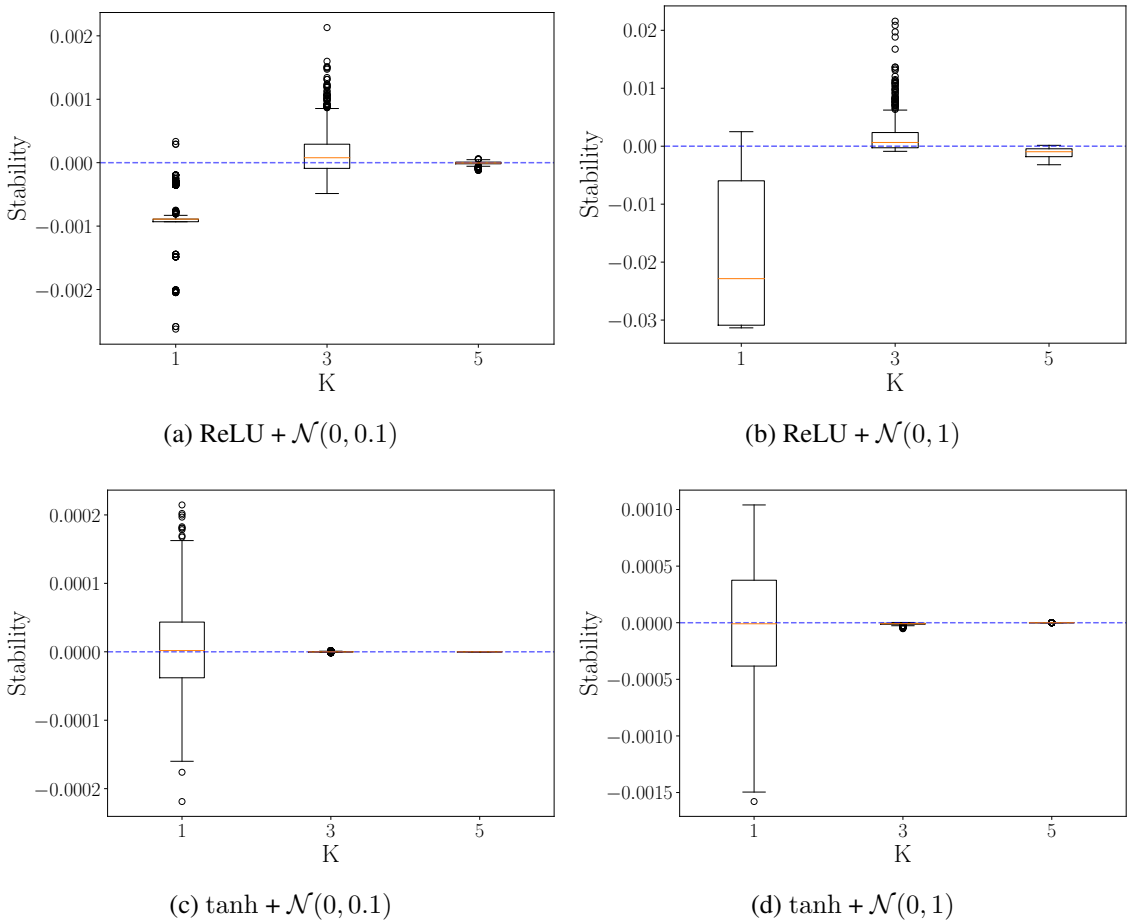
Theorem 5.3.6 *Suppose that $\phi(\cdot)$ is a 1-Lipshitz function in the norm induced by $\|\cdot\|$ (*i.e., *the ℓ_2 norm), $\|\mathbf{U}_l\| < \alpha_l$, and $\eta_{l,t}^k \leq \frac{1}{\alpha_l - \|\mathbf{U}_l\|}, \forall k$, then it follows that DIRNN in Eq. 5.8 converges to the equilibrium point asymptotically,* i.e., *$\mathbf{h}_{l,t} = \lim_{K\to\infty} \mathbf{g}_{l,t}(K) = \mathbf{g}_{l,t}^*$. Moreover, if $\eta_{l,t}^k = \eta_{l,t}, \forall k$ holds, then the convergence is linear with the rate of $1 - \eta_{l,t}(\alpha_l - \|\mathbf{U}_l\|)$.*

Proof: Letting $T(\mathbf{g}) = \mathbf{g} + \eta_{l,t}^k(\phi(\mathbf{U}_l\mathbf{z}_{l,t}(\mathbf{g}) + \mathbf{W}_l\mathbf{x}_t + \mathbf{b}_l) - \alpha_l\mathbf{z}_{l,t}(\mathbf{g}))$, similar to iRNN, it follows that $T(\cdot)$ is a contraction: $\|T(\mathbf{g}) - T(\mathbf{g}')\| \leq [1 - \eta_{l,t}^k(\alpha_l - \|\mathbf{U}_l\|)]\|\mathbf{g} - \mathbf{g}'\| < \|\mathbf{g} - \mathbf{g}'\|$. Therefore, the Euler's method in Eq. 5.8 leads to asymptotic convergence, with linear rate if $\eta_{l,t}^k$ are the same over k. We then complete the proof. ■

Learning Objective. Given training data $(x, y) \sim \mathcal{X} \times \mathcal{Y}$ where $x = \{\mathbf{x}_1, \cdots, \mathbf{x}_T\}$ is a data sample and y is its label, we aim to minimize the following loss:

$$\min_{\alpha_l, \gamma_{l,t}, \rho_{l,t}, \eta_{l,t}^k, \omega, \mathbf{U}_l, \mathbf{W}_l, \mathbf{b}_l} \sum_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} \ell\Big(y, \mathbf{z}_{L,T}; \omega\Big), \qquad (5.9)$$

where $\ell : \mathcal{Y} \times \mathbb{R}^D \times \Omega \to \mathbb{R}$ denotes the loss function parameterized by ω such as cross-entropy.



(a) ReLU + $\mathcal{N}(0, 0.1)$

(b) ReLU + $\mathcal{N}(0, 1)$

(c) tanh + $\mathcal{N}(0, 0.1)$

(d) tanh + $\mathcal{N}(0, 1)$

Figure 5.3: Training stability validation for Thm 5.3.5.

5.3.3 Discussion

We validate our analysis on the *Adding task* that is widely used for RNN evaluation. We strictly follow [4] to generate the dataset using the public code[1]. There are two sequences with length $T = 50$. The first sequence is sampled uniformly at random $\mathcal{U}[0, 1]$. The second sequence is filled with 0 except for two entries of 1. The two entries of 1 are located uniformly at random position i_1, i_2 in the first half and second half of the sequence. The prediction value is the sum of the first sequence between $[i_1, i_2]$. The ground-truth mean squared error is 0.167 when a model simply guesses 1 as the output regardless of

[1] https://github.com/rand0musername/urnn

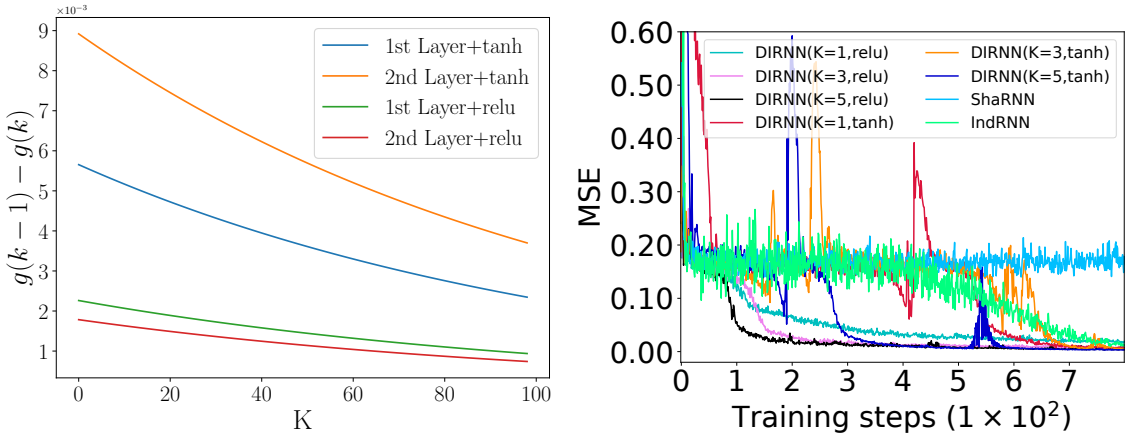
the input sequence. We use the value 0.167 as the baseline.

Stability Validation in Thm. 5.3.5 with Random Gaussian Weights for DIRNN. Note that $\mathtt{s}$ is a variable over continuous time, which is difficult to measure directly. Instead, we utilize the sampling approach to simulate it. Specifically, first we randomly select a sample from the dataset (denoted as $\tau = 0$) and add Gaussian noise (sampled from either $\mathcal{N}(0, 0.1)$ or $\mathcal{N}(0, 1)$) to each time step of the sample to generate a new sample at the time τ, leading to 5,000 new samples in total. Then we feed these samples together with the original one to DIRNN with random Gaussian weights for $L = 3$ as demonstration. Such weights satisfy the conditions in Thm. 5.3.6. The activation functions ϕ are either ReLU or $\tanh$, and the hidden state vectors are well distributed. Finally we compute $\left[\|\frac{\partial \dot{\mathbf{s}}(\tau)}{\partial \mathbf{h}_{l,t}}\|/\|\frac{\partial \dot{\mathbf{s}}(0)}{\partial \mathbf{h}_{l,t}}\| - 1\right]_{l=3,t=1}$ to measure the stability, and illustrate the results in Fig. 5.3. Though the errors in computing the equilibrium points are propagated through the network, we can still observe that:

- A larger K leads to better stability, regardless of the noise, indicating that our analysis should hold for the equilibrium points, *i.e.,* $K \to +\infty$;
- With small amounts of noise, all K's can perform similarly, on average, to preserve the stability;
- With large amounts of noise, $K = 1$ seems to fail due to the poor estimation of the equilibrium points.

Such observations can be made across different network depth L, sequence length T, network weights, and activation functions that satisfy the conditions in Thm. 5.3.6. Please refer to our supplementary file for more results.

Validation of Convergence to Equilibrium in Thm. 5.3.6. In general, it will be challenging to learn a deep model that can meet all the conditions in Thm. 5.3.6, except for *homogeneous* functions over $\mathbf{U}_l, \mathbf{W}_l, \mathbf{b}_l, \gamma_{l,t}, \rho_{l,t}$. To see this, we can rewrite the conditions as $\max_k\{\alpha_l - 1/\eta_{l,t}^k\} \leq \|\mathbf{U}_l\| < \alpha_l$. Since empirically it often holds that $0 \leq \alpha_l \leq 1$



(a) Equilibrium

(b) Training loss

Figure 5.4: Illustration of convergence on the adding task.

and $0 \leq \eta_{l,t}^k \leq 1$, the lower bound here is often less than 0, while $\|\mathbf{U}_l\|$ is often bigger than 0. Therefore, very often the lower bound holds. To guarantee the upper bound, we can simply divide $\mathbf{U}_l, \mathbf{W}_l, \mathbf{b}_l, \gamma_{l,t}, \rho_{l,t}$ by a sufficiently large constant. Such rescaling will not affect the lower bound.

Note that the conditions in Thm. 5.3.6 are sufficient. Therefore, in practice even the function in Eq. 5.8 is not homogeneous, the update for ODE may still converge. We illustrate such convergence behavior in Fig. 5.4(a) with either ReLU (leading to a homogeneous function in Eq. 5.8) or $\tanh$ as our activation functions. The network architecture is the same as the one for stability validation. Then we use Eq. 5.8 to compute the distances between the k and $k-1$ iterations in ODE. The monotonically decreasing curves over different layers demonstrate the convergence for both activation functions.

Training Convergence on Adding Task. To demonstrate the effect of our stability analysis on training networks, we illustrate the convergence of several RNNs on the adding task in Fig. 5.4(b), where ShaRNN and indRNN are another two deep RNNs. As we see, our DIRNN can converge with either ReLU or $\tanh$ activations, and ReLU leads to faster convergence. The setting of ReLU plus $K = 5$ converges the fastest in terms of the training steps. Note that neither ShaRNN nor indRNN has the convergence guarantee

theoretically, and thus some tricks such as gradient clipping are often used in training. In contrast, there is no need at all to use such training tricks in DIRNN to avoid vanishing/exploding gradients.

5.4 TinyRNN: A Sparsified DIRNN

A critical problem in DIRNN is that the number of parameters grows linearly with the number of layers L, leading to larger model sizes and longer running time *w.r.t.* other RNNs that may limit its applicability. To address this issue, we further propose a novel implementation of DIRNN, namely TinyRNN, where all the transition matrices are sparsified. Note that our stability analysis holds for any transition matrix which guarantees that TinyRNN can be trained well.

Inspired by ShuffleNet [193] and ShufflingRNN [136], we propose reparametrizing the transition matrices in Eq. 5.8 using weighted permutation matrices. That is,

$$\mathbf{U}_l = \mathbf{A}_l\mathbf{B}_l, \mathbf{W}_l = \mathbf{C}_l\mathbf{D}_l, \forall l \in [L], \tag{5.10}$$

where $\mathbf{A}_l, \mathbf{C}_l \in \mathbb{R}^{D\times D}$ denote two diagonal weighting matrices and $\mathbf{B}_l \in \mathbb{R}^{D\times D}, \mathbf{D}_l \in \mathbb{R}^{D\times d}$ denote two permutation matrices, respectively.

Our proposed weighted permutation matrices essentially favor the learning of orthogonality in the transition matrices for training RNNs [92]. To see this, let us take the calculation of $\mathbf{Uz}$ (no subscripts for simplicity) for example, $\mathbf{Uz} = \mathbf{ABz} = [a_i\mathbf{B}(i)\mathbf{z}]_{i\in[D]}$, where a_i denotes the i-th entry along the diagonal, $\mathbf{B}(i)$ denotes the i-th row in the matrix, and $[\cdot]$ denotes the vector concatenation operator. In our permutation matrices $\mathbf{B}_l, \mathbf{D}_l, \forall l$, the rows of each matrix are predefined as orthogonally with each other as possible. Such permutation matrices define new coordinate systems where the diagonal

matrices are learned for proper scaling factors.

Predefined Random Permutation Matrices. Note that learning the permutation matrices along with the TinyRNN training is very challenging, because the permutation imposes a strong constraint in the matrix structure that is difficult to be satisfied during learning. To address this problem, we propose initializing both permutation matrices randomly ($\mathbf{B}_l$ without repetition and $\mathbf{D}_l$ with repetition if $d < D$, otherwise without repetition) and fixing them during training. We only learn the weighting matrices accordingly.

Number of Model Parameters. To further reduce the model size, we simply set $\mathbf{b}_l = \mathbf{0}, \forall l$, because we only observe marginal accuracy improvement with such parameters. Therefore, the learnable parameters in our TinyRNN are $\mathbf{A}_l, \mathbf{C}_l, \alpha_l, \gamma_{l,t} = \gamma_l, \rho_{l,t} = \rho_l, \eta^k_{l,t} = \eta_l$, leading to the total number of parameters as $(2D + 4)L$. We also learn a linear classifier ω on top of hidden state $\mathbf{z}_{L,T}$ consisting of $D|\mathcal{Y}|$ parameters, where $|\mathcal{Y}|$ denotes the cardinality of the label set. Together, we have $(2D+4)L + D|\mathcal{Y}|$ parameters.

Fix-Point Quantization. To further reduce the model size, the fix-point quantization techniques can be integrated into TinyRNN as well, either during or after the training. For simplicity to demonstrate this capability, in our experiments we apply the quantizer in [85] as a post-processing step to compress our models.

5.5 Experiments

Datasets. We test our approach on different tasks that are widely used in the literature of RNN evaluation. All the statistics of the benchmark datasets are listed in Table 5.1.

- *Benchmark vision task:* We conduct the experiments on two datasets, *i.e.,* Pixel-MNIST and Perm-MNIST. Specifically, Pixel-MNIST refers to pixel-by-pixel sequences of images in MNIST where each 28 $\times$ 28 image is flattened into a 784 time sequence vector, while a random permutation to the Pixel-MNIST is applied to generate a harder time

Table 5.1: Dataset statistics and default hyperparameters in DIRNN and TinyRNN. Please refer to our ablation study for more details.

Dataset	Statistics					DIRNN			TinyRNN		
	#Train	#Test	#Time	#Feat.	#Cls.	D	L	K	D	L	K
Pixel-MNIST	60k	10k	784	1	10	128	3	3	128	3	2
Perm-MNIST	60k	10k	784	1	10	128	5	3	128	3	2
Noisy-MNIST	60k	10k	1000	28	10	128	4	3	128	10	1
Noisy-CIFAR	50k	10k	1000	96	10	256	4	3	256	10	1
HAR-2	7.3k	2.9k	128	9	2	128	5	3	128	5	3
DSA-19	4.6k	4.6k	125	45	19	128	4	3	128	6	3
PTB	929k	82k	300	300	10k	300	5	3	300	5	1

sequence dataset as Perm-MNIST.

- *Noise padded vision task:* For this task, we utilize Noisy-MNIST and Noisy-CIFAR. For Noisy-MNIST, each row of an MNIST image with dimension 28 is fed into the model at every time step. The first 28 time steps of input contain the original 28 rows of MNIST. The remaining 972 time steps are filled with random noise, leading to $T = 1,000$ time steps in total. Noisy-CIFAR is created in the same way with input dimension $32 * 3 = 96$ due to the RGB channels.
- *Human activity recognition task:* We test our method on HAR-2 [84] and DSA-19 [2]. HAR-2 was collected from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone. DSA-19 was collected from a resource constrained IoT wearable device with 5 Xsens MTx sensors having accelerometers, gyroscopes and magnetometers on the torso and four limbs.
- *Language modeling task:* We test our method on Penn Treebank (PTB) dataset [112]. 300 length word sequences were used for word level language modeling task using Penn Treebank (PTB) corpus. The vocabulary consisted of $10,000$ words and the size of trainable word embeddings was kept the same as hidden units. This is the setup used in [85]. Note that, for the language modeling task, just the model size of the various RNN architectures has been reported.

Baseline Algorithms.. We compare our results with baselines including vanilla RNN,

LSTM [61], AntisymmetricRNN [16] [2], FastRNN [85][3], FastGRNN [85][4], IndRNN [95][5], ShaRNN [32][6], iRNN [73][7], LipschitzRNN [40][8] and MomentumRNN [120][9]. As a demonstration for comparison on model size, we employ the pruning algorithm from [52][10] as a post-processing step for TinyRNN to further compress the learned models. Note that using the public code we have verified the results of each competitor on the datasets that were reported in the references. For simplicity and consistency we cite the numbers from the references, if exist, otherwise, we report our reproduced results with the tuned best hyperparameters.

Training & Testing Protocols. In our implementation, we utilize ReLU as our activation functions, as we observed that this activation works better than others such as $\tanh$ in terms of both accuracy and convergence. This observation is consistent with the state-of-the-art methods such as iRNN and FastRNN. In both DIRNN and TinyRNN, we set $\gamma_{l,t} = \gamma_l, \rho_{l,t} = \rho_l, \eta^k_{l,t} = \eta_l$ over different time steps and iterations in the ODE solver. Following FastRNN and iRNN for a fair comparison, we replicate the same benchmark training/testing split with 20% of training data for

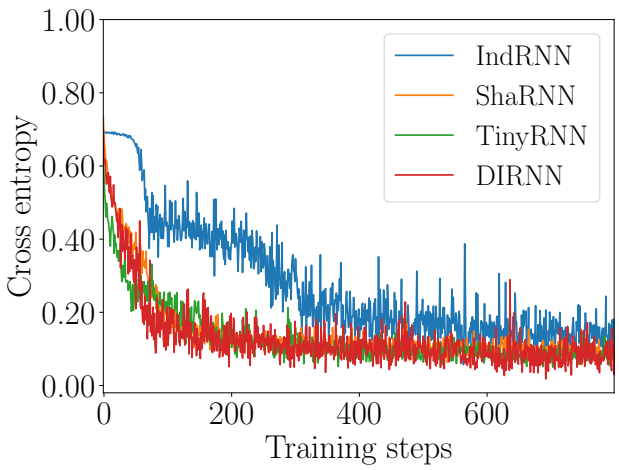


Figure 5.5: Training curve comparison on HAR-2.

[2] https://github.com/KurochkinAlexey/AntisymmetricRNN
[3] https://github.com/microsoft/EdgeML
[4] https://github.com/microsoft/EdgeML
[5] https://github.com/Sunnydreamrain/IndRNN_pytorch
[6] https://github.com/microsoft/EdgeML
[7] https://github.com/dtake1336/ERNN-for-speech-enhancement
[8] https://github.com/erichson/LipschitzRNN
[9] https://github.com/minhtannguyen/MomentumRNN
[10] https://github.com/mightydeveloper/Deep-Compression-PyTorch

validation to tune hyperparameters. Then we retrain the models using best tuned hyperparameters using the full training set and test them on the test set. We report our results over three trials with randomization wherever needed. All the experiments were run on an Nvidia Tesla P40 GPU with CUDA10 on a machine with Intel Xeon@2.60GHz CPU with 20 cores.

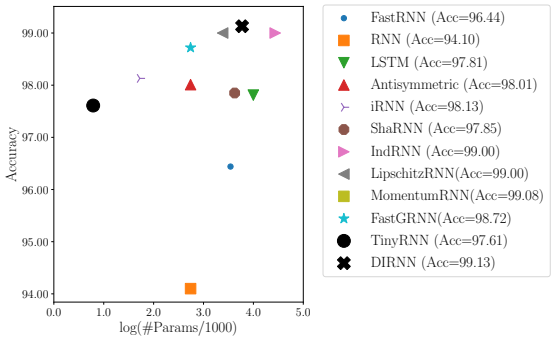
Hyperparameters. We use the grid search to fine-tune the hyperparameters of each baseline as well as ours on the validation datasets whenever is necessary. Table 5.1 lists all the hyperparameters for DIRNN and TinyRNN architectures on different datasets, where we follow [85, 73, 16] to set the hidden feature dimension D, and fine-tune the numbers of layers in depth L, and update times in iRNN K, respectively. The batch size of 128 seems to work well across all the datasets for all the methods. Adam [78] is used as the optimizer for all the methods. The learning rate is always initialized to 10^{-3} with linear scheduling of weight decay.

5.5.1 Ablation Study on HAR-2

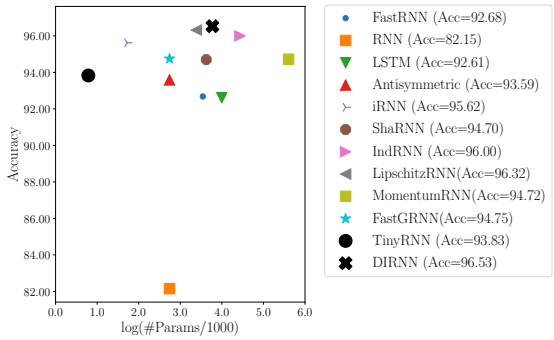
We first show the training convergence in Fig. 5.5 that has similar behavior to Fig. 5.4(b) for the adding task. This demonstrates that the training stability can be generalized on different datasets. Then we illustrate the performance of our method under different (D, L, K)-hyperparameter settings in Fig. 5.7, where we can see that:

- The deep architectures contribute the most to the performance of both architectures;
- $D = 128$ is already sufficient for good accuracy with a small model size;
- Among all settings, we can observe a performance drop from $L = 5$ to $L = 6$;
- It seems that small K's can already produce good accuracy that leads to good computational efficiency as well.

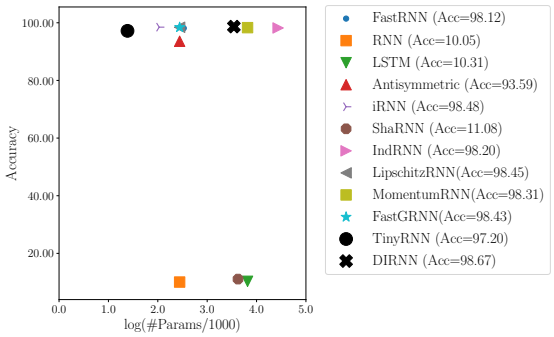
5.5.2 State-of-the-art Performance Comparison



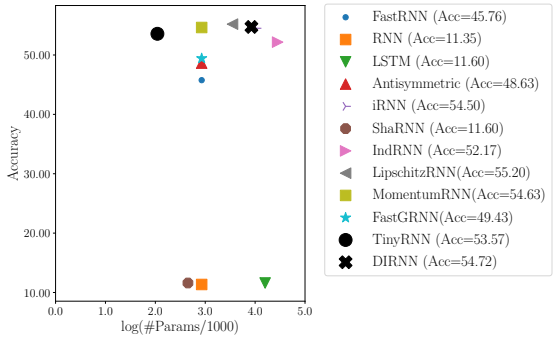
(a) Pixel-MNIST



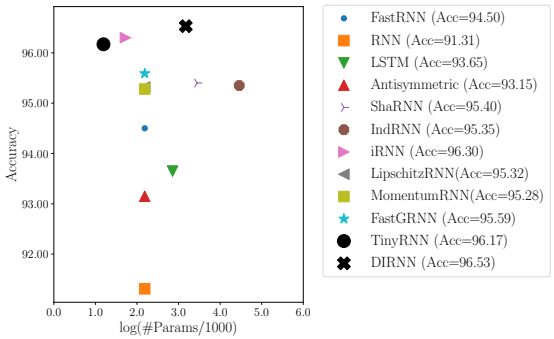
(b) Perm-MNIST



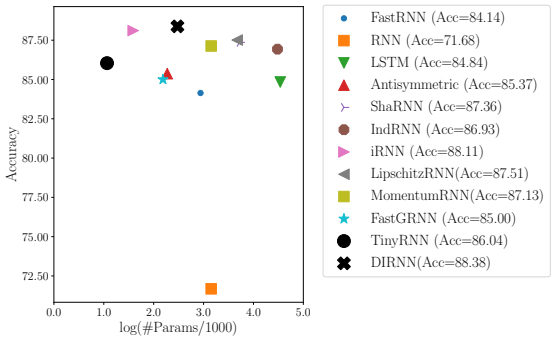
(c) Noisy-MNIST



(d) Noisy-CIFAR



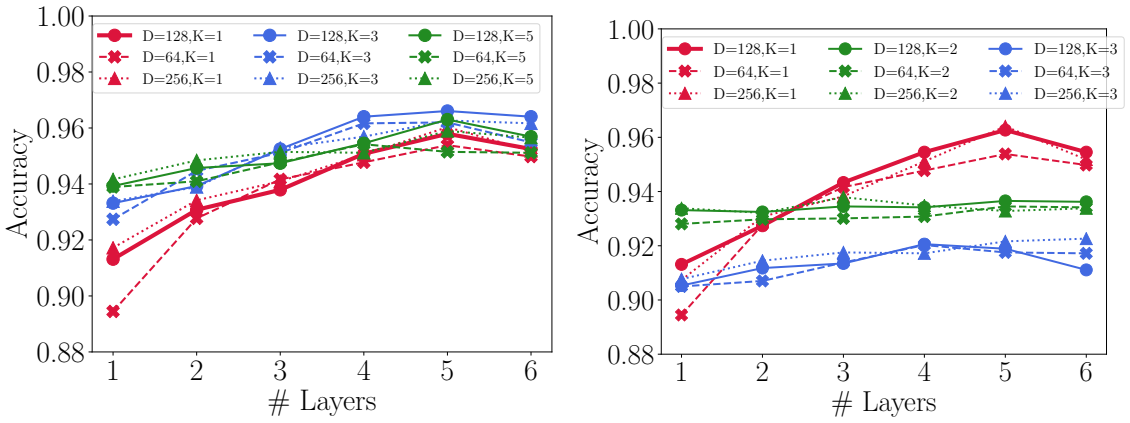
(e) HAR-2



(f) DSA-19

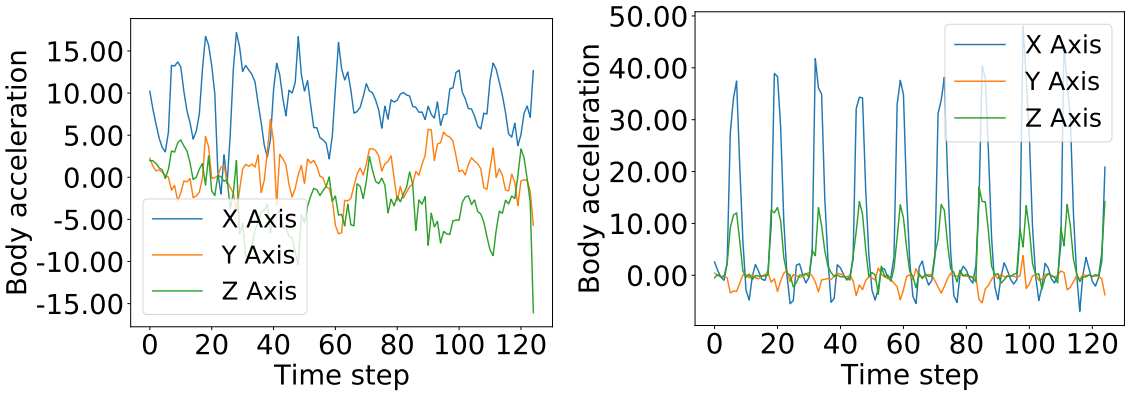
Figure 5.6: Test accuracy comparison. To better view the differences between different RNN architectures, by following some recent papers such as FastGRNN and iRNN, here the numbers of parameters exclude those for linear classifiers that are identical for all the competitors.

Benchmark Vision Task. Fig. 5.6(a-b) illustrate our performance comparison. The y-axis is the test accuracy of different models and the x-axis is the number of parameters in the log scale. As we see, DIRNN achieves the best on both datasets, indicating the



(a) DIRNN (b) TinyRNN

Figure 5.7: Ablation study on the HAR-2 dataset.



(a) Playing basketball (b) Jumping

Figure 5.8: Two kinds of activities in DSA-19 dataset.

advantages of deep RNN structures. Compared with it, the results of TinyRNN are lower by $1\% \sim 2\%$ with a tiny fraction of parameters. This compact design in TinyRNN can provide us the capability of processing larger data orrunning multiple models in parallel in real-world applications.

Noise Padded Vision Task. This task is specifically designed for evaluating the capability of RNNs to capture the long term dependency (LTD) among the data. Fig. 5.6(c-d) illustrate our comparison results. On Noisy-MNIST, DIRNN achieves the best accuracy, while on Noisy-CIFAR LipschitzRNN achieves the best and DIRNN is the second. TinyRNN

Table 5.2: Result comparison of quantization. Our results are over three trials with networks being trained from the scratch.

Algorithms	HAR-2		DSA-19		PTB	
	Acc.(%)	Model (KB)	Acc.(%)	Model (KB)	Perplexity	Model (KB)
RNN	91.31	29	71.68	20	144.71	129
LSTM	93.65	74	84.84	526	117.41	2052
FastRNN	94.50	29	84.14	97	127.76	513
FastGRNN	95.59	3	83.73	3.25	116.11	39
Antisymmetric	93.15	29	85.37	32	116.87	45
iRNN	96.30	18	88.11	19	115.71	288
ShaRNN	95.40	29	87.36	40	116.14	130
IndRNN	95.35	139	86.93	93	116.02	653
LipschitzRNN	95.32	35	87.51	260	115.36	578
MomentumRNN	95.28	29	87.13	20	115.87	130
DIRNN	**96.53**$\pm$0.18	174	**88.38**$\pm$0.16	124	**115.35**$\pm$0.35	775
TinyRNN	96.17$\pm$0.24	8	86.04$\pm$0.17	12	115.81$\pm$0.45	45
TinyRNN+Q	95.73$\pm$0.32	**0.99**	85.65$\pm$0.18	**1.4**	116.09$\pm$0.43	**20**

achieves only $\sim$1% accuracy loss with $\sim$3x smaller architecture compared with LipschitzRNN. Surprisingly, ShaRNN works as poorly as vanilla RNNs on both datasets. We hypothesize that this comes from the network design choice where the two-layer architecture with data splitting actually breaks the LTD among the data, although this may not occur on other datasets.

Human Activity Recognition Task. We illustrate our comparison results in Fig. 5.6(e-f). Unsurprisingly, our DIRNN achieves the best on HAR-2 with an accuracy of 96.53%. And on DSA-19, the DIRNN architecture also achieves the best accuracy. Besides, our TinyRNN achieves the second best on HAR-2 with the smallest number of parameters. This is probably because the data is relatively simple with low-dimensional features and fewer classes where models with small complexity can handle.

Model Size Comparison. Table 5.2 lists the test accuracy and model sizes (*i.e.,* the storage on the hard drive, including the linear classifiers) of different competitors. *TinyRNN+Q* indicates the method where the quantization method in FastGRNN is applied to further reduce the sizes of learned models by TinyRNN. As we see, DIRNN works the best in

terms of accuracy, and TinyRNN+Q performs slightly worse but with smaller models by two orders of magnitude. This indicates that the proposed tiny architecture can achieve a good balance between model size and performance.

Training & Inference Time. Such numbers are highly dependent on the network architectures of DIRNN as well as the data. For instance, on HAR-2 and DSA-19, the training time is 103s and 100s per epoch and the inference time is 0.46ms and 0.35ms per data sequence on an Nvidia Tesla P40 GPU.

5.5.3 Case Study

We use real world cases from DSA-19 dataset to demonstrate the advantage of the proposed DiRNN model. The DSA-19 dataset contains 19 different human activities and some activities have similar patterns which sometimes makes it hard to distinguish those similar activities. For example, both jumping and playing basketball are categories of those 19 activities. However, jumping is a common activity when playing basketball. In Fig. 5.8, we show the acceleration sensor data of users' body movement when they perform playing basketball and jumping, respectively. From the Fig. 5.8 (b), we can observe periodic fluctuations of the acceleration in X Axis, which indicates the body ascending and descending in the air. However, when a user plays basketball, the jumping sequence becomes noisy, since the player usually completes more complex movements with the basketball such like dribbling or shooting. In our experiments, we notice that the traditional RNN model usually cannot tell the difference between jumping and playing basketball, we believe it is because that the vanilla RNN model only detects the jumping patterns in the basketball sequence and treat other irregular patterns as noise. However, with our proposed DiRNN model, we can successfully classify the jumping and playing basketball activities. This indicates that the proposed DiRNN model can better learn the sequential patterns in the series data.

Chapter 6

Lightweight Convolutional Neural Network via Recurrent Convolution

6.1 Introduction

Convolutional neural networks (CNNs) [82] have revolutionized computer vision by achieving state-of-the-art performance on many applications [54, 45, 71]. The impressive improvement usually comes with a substantial increase in the number of parameters (*i.e.,* model size) which is undesirable for the model applicability in many real-world applications [46, 51], such as embedded systems where the computing resources (*e.g.,* processor and memory) in the hardware are limited. Therefore, how to design/learn lightweight neural networks, *i.e.,* reducing storage requirement in terms of parameters while achieving good performance, is becoming increasingly demanded [139, 33, 105, 204, 93, 147].

Generally speaking, there are two families of approaches for learning lightweight networks in the literature: (1) network architecture design/search, and (2) network compression. Typical works in the former family include SqueezeNet [67], MobileNet [139], ShuffleNet [193], EfficientNet [154], MnasNet [153], and ProxylessNAS [14]. Such ap-

proaches focus on developing network architectures (*e.g.,* using small convolutional filters) to satisfy certain requirements such as model size while achieving good performance for the applications. The latter family includes the approaches such as compression with learning [168, 94, 97, 75, 37, 200, 132, 63, 193] or after learning [52, 85], whose basic ideas are to remove the network redundancy by imposing some structural assumptions on the convolutional filters. Nice surveys on this topic can be found in [22, 186, 24, 30, 117].

Motivation. Intuitively, reducing the number of parameters in each convolutional layer can significantly compact a given network. However, this may lead to poor generalization of the network, as wider networks have been shown to effectively improve the performance [185, 154]. In order to compensate for the performance loss due to model size reduction (*i.e.,* lightweight networks), we are mainly motivated by the following works:

- *Deeper Networks:* In complement to the universal approximation theorem [28], recent works such as [110] have shown that with the increase of network depth, the number of hidden neurons can be dramatically reduced to approximate a function with similar expressive power. This motivates us to consider constructing a deeper network using narrow networks.
- *Visual Transformer (ViT):* Recently [35] demonstrated excellent performance on image recognition using ViT that are designed to handle sequential input data, similar to recurrent neural networks (RNNs). In their work, each image is divided into 16×16 patches in the spatial domain, and then fed into ViT as an input data sequence. This motivates us to consider exploring RNNs (not ViT due to its large model size) in different ways to learn lightweight networks.

Our proposed approach and contributions. Based on consideration above, we propose a novel convolutional layer, namely *Channel-Split Recurrent Convolution (CSR-Conv)*, as illustrated in Fig. 6.1(c) where the 256 input channels are equally split into 4 groups, fed into a recurrent convolutional layer (implemented using a vanilla RNN in the figure

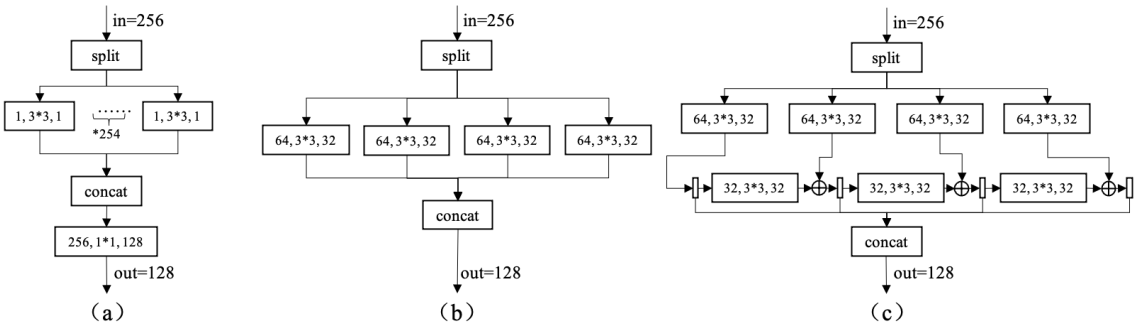


Figure 6.1: Comparison with 256 input channels and 128 output channels among **(a)** depth-wise separable convolution, **(b)** group convolution, and **(c)** our channel-split recurrent convolution (CSR-Conv) using vanilla RNNs. The linear convolutional operation is denoted as (#input channels, filter size, #output channels) and vertical small rectangles in **(c)** denote ReLU activation functions.

as demonstration) as input, and the hidden states in the RNN are concatenated to generate the 128 output channels. Compared with depth-wise separable convolution (used in MobileNet [139]) and group convolution (used in ShuffleNet [193] and ResNeXt [178]) in Fig. 6.1(a-b), respectively, we can see clearly that our key difference is to replace each linear convolution with a recurrent convolution. As a result, if imaging each figure as a graph where all the linear convolutions are denoted by nodes, then the depths (*i.e.,* the longest paths) between node "split" and node "concat" in the figures are different: in Fig. 6.1(a-b) the depths are both 2, while in Fig. 6.1(c) the depth is 5. In other words, recurrent convolution can lead to deeper network architectures, which could be beneficial for learning lightweight convolutional networks.

We are aware that the integration of RNNs with convolution for deep learning has been explored in the literature such as [163, 101, 76, 151, 145, 122, 148]. However, to the best of our knowledge, *we are the first to explore the applicability of recurrent deep models (*e.g., *RNNs, GRUs, LSTM) as general recurrent convolutional layers to learn lightweight CNNs.* Given a backbone network such as VGGNet [146] or ResNet [54], we can easily replace its linear convolutional layers using our CSR-Conv to reduce the model

size, achieving a deeper network as well as preserving its performance[1]. We analyze the relationship between model size and CSR-Conv to show that the model compression rate is fairly controllable. We also demonstrate the state-of-the-art performance of our approach based on seven existing network architectures on CIFAR-10 [81] and ImageNet [29] datasets.

6.2 Related Works

Recurrent neural networks. RNNs have achieved significant success in learning complex patterns for sequential input data, and have been widely used in computer vision [197, 203, 198, 190, 27, 128]. At each time step, an RNN updates the state vector based on the current state and input data. Subsequently, RNNs output the predictions as a function of the hidden states. The model parameters are learned by minimizing an empirical loss. In the literature, there are significant amount of works on developing RNNs such as, just to name a few, long short-term memory (LSTM) [61], gated recurrent unit (GRU) [23], UGRNN [26], FastGRNN [84], unitary RNNs [4], antisymmetric RNN [17], incremental RNN [73], exponential RNN [92], Lipschitz RNN [40].

Recurrent convolutional neural networks (RCNN). [101] proposed incorporating the recurrent connections in each convolutional layer to generate features with different resolutions. [163] added a gate to the recurrent connections in RCNN to control context modulation and balance the feed-forward information and the recurrent information. [76] imposed very deep recursive layers to improve performance without introducing new parameters for additional convolutions. [151] developed a recursive convolutional neural network with the residual connection. [145] replaced vanilla RNN architecture with an

[1]Certainly we can design new networks using our standalone CSR-Conv layers, but this is beyond the scope of this chapter. In this work, we only focus on learning lightweight networks given certain backbone networks.

LSTM structure in RCNN. [122] used dilated convolution in the RCNN to reduce computational complexity.

Variants of convolutional operator for compression. [31] proposed using a linear combination of basis functions to predict parameters for compression. [5] proposed encoding convolutions by few lookups to a dictionary trained to cover the space of weights in CNNs. [171] presented a parameter-free, FLOP-free "shift" operation as an alternative to spatial convolutions. [44] proposed channel-wise convolutions, which replace dense connections among feature maps with sparse ones in CNNs. [168] proposed a Structured Sparsity Learning (SSL) method to regularize the structures such as filters, channels, filter shapes, and layer depth of CNNs. [102] proposed an efficient CircConv operator based on the presumed circulant structures of convolutional filters where Fast Fourier Transform (FFT) can be used to compute the filter responses in feed-forward and inverse FFT can be applied in back-propagation.

Network compression. Weight pruning [52, 53, 98] aims at reducing non-significant weights to reduce computation and memory usage of the model. Other than weight pruning, filter level pruning which leads to the removal of the corresponding feature maps is also studied intensively [55, 96]. Regularization constraints are also introduced in pruning [3, 108, 66]. Low-rank factorization [88, 150, 68, 194, 183] aims to decompose the large weight matrices in the convolutional layers into smaller matrices with fewer parameters. Knowledge distillation [59, 87, 126] aims to force a smaller student network to fit specific features from a larger teacher network for knowledge transfer.

6.3 Our Approach

6.3.1 Problem Definition

In this chapter, we only focus on learning lightweight convolutional networks by replacing some linear convolutions with CSR-Conv in a given backbone network such as VGGNet or ResNet, so that the model size can satisfy certain requirements. We will not design or propose new network architectures.

Specifically, given a backbone network with L convolutional layers, a desirable model compression rate ρ_M (this constraint is optional depending on the applications/users), and a training dataset $\{\mathbf{x}, y\} \subseteq \mathcal{X} \times \mathcal{Y}$ with sample $\mathbf{x} \in \mathcal{X}$ and label $y \in \mathcal{Y}$, we propose the following optimization problem as our objective for learning lightweight networks:

$$\min_{\omega, \mathcal{T} \in \mathbb{Z}^L} \mathbb{E}_{(\mathbf{x},y)} \ell\Big(f(\mathbf{x}; \omega, \mathcal{T}), y\Big), \text{ s.t. } \frac{M_C}{M_B} \approx 1 - \rho_M, \tag{6.1}$$

where f denotes the modified network with CSR-Conv parametrized by ω, $\mathcal{T}$ denotes a set of the sequence lengths as input to the recurrent convolutional layers in CSR-Conv (if the length is equal to 1, there will be no change to the linear convolution), ℓ denotes the loss function, $\mathbb{E}$ denotes the expectation operation, and M_C, M_B denote the numbers of parameters in the modified and backbone networks, respectively. In case that achieving the exact compression rate ρ_M may be impossible, we instead try to search for the best network architectures with similar compression rates.

Grid-search solver with CSR-Conv. In contrast to network architecture search (NAS) that is optimized in the network architecture space, in this work we simply use grid-search to determine $\mathcal{T}$, same as EfficientNet [154], because our search space is much smaller than NAS given the compression rate and backbone network. To accelerate our training, in our implementation we further reduce our search space to $\mathcal{T} \in \{1, T\}^L$, that

Table 6.1: Illustration of our CSR-Conv-4 architecture in Table 6.2 for VGG-16 with $T = 5$, where the parameters in the 6th-13th convolutional layers are converted to the parameters $\mathbf{U}, \mathbf{V}$ in CSR-Conv with the same spatial sizes.

Layer	VGG-16	Ours	#Param (ρ_M)
Conv1	[3×64]	[3×64]	1,728(0.0%)
Conv2	[64×64]	[64×64]	36,864(0.0%)
Conv3	[64×128]	[64×128]	73,728(0.0%)
Conv4	[128×128]	[128×128]	147,456(0.0%)
Conv5	[128×256]	[128×**260**]	299,520(-1.6%)
Conv6	[256×256]	**[52×52]** **[52×52]**	48,672(91.9%)
Conv7	[256×256]	**[52×52]** **[52×52]**	48,672(91.9%)
Conv8	[256×512]	**[52×103]** **[103×103]**	134,685(87.8%)
Conv9	[512×512]	**[103×103]** **[103×103]**	190,962(91.9%)
Conv10	[512×512]	**[103×103]** **[103×103]**	190,962(91.9%)
Conv11	[512×512]	**[103×103]** **[103×103]**	190,962(91.9%)
Conv12	[512×512]	**[103×103]** **[103×103]**	190,962(91.9%)
Conv13	[512×512]	**[103×103]** **[103×103]**	190,962(91.9%)
FC	/	/	267,264(0.0%)
Total			2,022,489(86.5%)

is, a linear convolutional layer is either unchanged or split into T groups of channels. We then determine $T > 1$ using grid-search as well as learning ω. We list an exemplar of our network implementation in Table 6.1 where the bold parts are the filter sizes in CSR-Conv. We restrict our grid search so that the number of channels in the backbone network is approximately preserved by the RNNs.

6.3.2 CSR-Conv: Channel-Split Recurrent Convolution

We illustrate the general architecture of CSR-Conv in Fig. 6.2, where "Split" and "Concat" denote the channel split and concatenation operations, respectively. The in-between recurrent convolutional layer takes the split data sequence as input and outputs the hidden states over time. It can be implemented using an RNN, GRU, LSTM, *etc*. Recall that Fig. 6.1 illustrates our customized implementation based on a vanilla RNN, where the input and output are 3D features and the network weights are 4D. For simplicity, we represent all the input and output data as vectors, and network weights as matrices. Specifically, we denote $\mathbf{x}_l \in \mathbb{R}^{d_l}, \forall l \in [L]$ as a d_l-dim input for the l-th convolutional/recurrent layer ($\mathbf{x}_l = \mathbf{x}$, *i.e.,* the input data to the

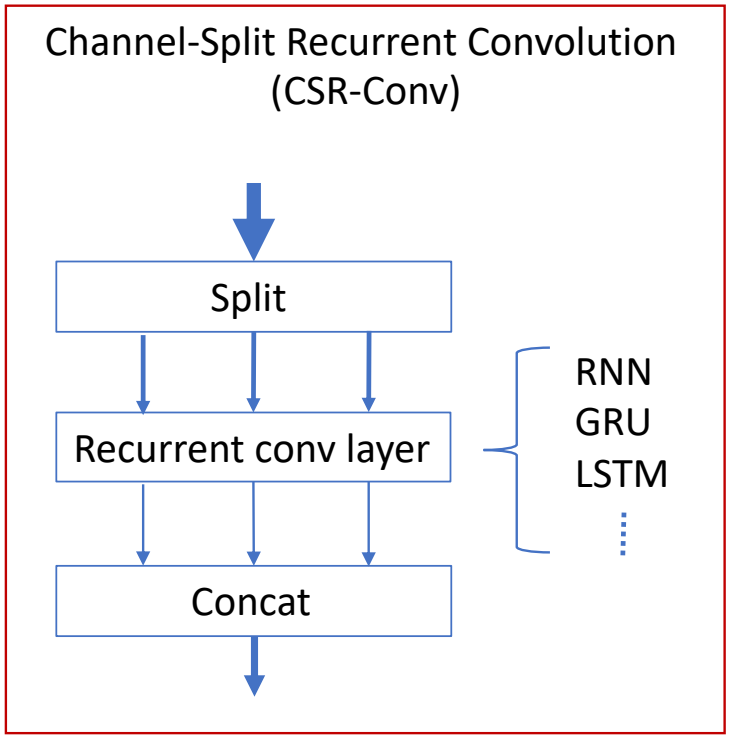


Figure 6.2: General architecture.

network, when $l = 0$). We will explain the architecture based on a vanilla RNN as well.

Channel split. The goal of this step is to generate data sequence based on the input channels for further process in the recurrent layer. Imagining that we need a sequence with length T at the l-th convolutional layer, then we reshape $\mathbf{x}_l$ to a matrix $\mathbf{X}_l = [\mathbf{x}_{l,t}]_{t\in[T]} \in \mathbb{R}^{\lceil \frac{d_l}{T} \rceil \times T}$ where $\lceil \cdot \rceil$ denotes the ceiling operator and $[\cdot]_{t\in[T]}$ denotes the vector concatenation operator. This new matrix will be fed into the recurrent layer column-by-column sequentially.

Vanilla RNN based recurrent convolution. We follow the simplest RNN formulation (*i.e.,* vanilla RNN) as below to implement the recurrent layer:

$$\mathbf{h}_{l,t} = \sigma\Big(\mathbf{U}_l^T \mathbf{h}_{l,t-1} + \mathbf{V}_l^T \mathbf{x}_{l,t}\Big), \mathbf{h}_{l,0} = \mathbf{0}, \forall t \in [T], \tag{6.2}$$

where at the layer l and time step t, $\mathbf{h}_{l,t} \in \mathbb{R}^{D_l}$ denotes the hidden state vector, $\mathbf{U}_l \in \mathbb{R}^{D_l \times D_l}$, $\mathbf{V}_l \in \mathbb{R}^{\lceil \frac{d_l}{T} \rceil \times D_l}$ denote the shared state and data transition matrices in the RNN, σ denotes the activation function such as ReLU, and $(\cdot)^T$ denotes the matrix transpose operator. Here we do not take the bias term into account, because in practice we do not observe any significant improvement with the bias term but introducing more parameters. Note that the recurrent layer defined in Eq. 6.2 can be viewed as the generalization of the traditional linear convolution, because both will be equivalent when $T = 1$. For other implementations, one can replace the formula in Eq. 6.2 with the corresponding formula to construct the recurrent layer.

Channel concatenation. Once we have the collection of the hidden state vectors, we simply concatenate them into a $(D_l \times T)$-dim vector $\mathbf{h}_l = [\mathbf{h}_{l,t}^T]^T$ that will be used in further process.

Table 6.2: Summary of our results on **(2nd block)** CIFAR-10 and **(3rd block)** ImageNet, where "#C-C" denotes the number of CSR-Conv modules used in the networks for learning compact networks, and "ρ_M" denotes the model size compression rate.

Network	Top-1 Err.(%)	$\rho_M(\downarrow)$	#Param.	#C-C	T
VGG-16	6.04±0.05	0.0%	14.98M	0	1
CSR-Conv-1	**5.89±0.06**	39.3%	9.09M	5	2
CSR-Conv-2	6.01±0.10	49.0%	7.64M	4	3
CSR-Conv-3	6.16±0.10	67.2%	4.91M	6	3
CSR-Conv-4	6.35±0.08	86.5%	2.02M	9	5
CSR-Conv-5	7.08±0.12	95.0%	0.75M	12	9
ResNet-56	6.74±0.14	0.0%	0.85M	0	1
CSR-Conv-1	**6.12±0.11**	21.8%	0.66M	4	2
CSR-Conv-2	6.69±0.12	61.0%	0.33M	11	3
CSR-Conv-3	6.83±0.10	70.3%	0.25M	17	3
CSR-Conv-4	7.93±0.19	78.8%	0.18M	15	4
CSR-Conv-5	9.15±0.13	88.9%	0.09M	22	5
ResNet-110	6.50±0.05	0.0%	1.74M	0	1
CSR-Conv-1	5.72±0.07	17.2%	1.44M	7	2
CSR-Conv-2	**5.55±0.05**	36.4%	1.11M	14	2
CSR-Conv-3	6.12±0.11	61.3%	0.67M	22	3
CSR-Conv-4	7.06±0.15	79.6%	0.35M	31	4
CSR-Conv-5	8.57±0.18	87.1%	0.22M	33	5
DenseNet-40	5.19±0.04	0.0%	1.06M	0	1
CSR-Conv-1	5.19±0.12	15.9%	0.89M	19	2
CSR-Conv-2	5.13±0.09	35.2%	0.69M	11	3
CSR-Conv-3	**5.09±0.14**	50.3%	0.53M	22	3
CSR-Conv-4	6.01±0.13	63.7%	0.38M	23	4
CSR-Conv-5	8.30±0.15	82.4%	0.19M	34	5
MobileNet-V2	5.53±0.15	0.0%	2.24M	0	1
CSR-Conv-1	5.21±0.13	26.4%	1.65M	4	3
CSR-Conv-2	**5.08±0.14**	34.3%	1.47M	7	3
CSR-Conv-3	5.37±0.09	44.1%	1.25M	17	3
CSR-Conv-4	5.84±0.12	51.4%	1.09M	18	3
CSR-Conv-5	6.10±0.21	57.3%	0.95M	18	4
ResNet-50	23.85±0.23	0.0%	25.56M	0	1
CSR-Conv-1	**23.51±0.27**	35.7%	16.43M	10	3
CSR-Conv-2	24.61±0.24	70.3%	7.59M	15	4
EfficientNet-B0	22.90±0.23	0.0%	5.28M	0	1
CSR-Conv-1	**22.34±0.31**	18.9%	4.28M	3	3
CSR-Conv-2	27.59±0.31	26.3%	3.89M	4	5
MobileNet-V2	27.80±0.29	0.0%	3.50M	0	1
CSR-Conv-1	**27.65±0.32**	14.0%	3.01M	2	4
CSR-Conv-2	29.45±0.32	29.5%	2.47M	12	4

6.3.3 Analysis

Proposition 6.3.1 (Model Size) *Suppose that the numbers of input and output channels in each convolutional layer of the backbone network are equal to those from CSR-Conv with sequence length $T(T > 1)$. Then we can compute the model size ratio, λ_M, between CSR-Conv in Eq. 6.2 and the corresponding linear convolution as follows:*

$$\begin{aligned}\lambda_M &= \frac{k^2 D(D+d)}{k^2 DdT^2} = \left(1 + \frac{d}{D}\right)\frac{1}{T^2} \\ &= O\left(\frac{1}{T^2}\right).\end{aligned} \tag{6.3}$$

Often empirically $d \leq D \Leftrightarrow 0 < \frac{d}{D} \leq 1$ holds. Meanwhile, given the fact that the number of parameters in unchanged sub-networks is trivial, the compression rate will be heavily dominated by the number of the duplicate networks T.

Proposition 6.3.2 (FLOPs) *Suppose that (1) the computational complexity of add, multiplication, and σ is a unit operation with one FLOP, and (2) the input and output dimension for the backbone network can be represented as dT and DT, respectively. Then we can compute the FLOP ratio, λ_F, between CSR-Conv in Eq. 6.2 and the corresponding linear convolution as follows:*

$$\begin{aligned}\lambda_F &= \frac{k^2 WHDT(1 + 2D + 2d)}{k^2 WHDT(1 + 2dT)} \\ &= \frac{1 + 2D + 2d}{1 + 2dT} \leq \left(1 + \frac{D}{d}\right)\frac{1}{T},\end{aligned} \tag{6.4}$$

where the equation holds if and only if $T = 1 + \frac{D}{d}$ that leads to $\lambda_F = 1$.

The upper-bound in Eq. 6.4 indicates that the FLOPs of CSR-Conv tends to decrease *w.r.t.* T approximately. For instance, empirically our CSR-Conv-1 for ResNet-56 in Table 6.2 has the same FLOPs as ResNet-56, even with better performance and smaller model size,

because we set $T = 2$ and $d = D$ in CSR-Conv in our implementation. Differently, CSR-Conv-5 can achieve 42.0% of FLOP compression rate, compared with ResNet-56.

6.4 Experiments

Datasets. Following the literature, we evaluate our approach comprehensively on CIFAR-10 [81] and ImageNet [29] for the image classification task. CIFAR-10 consists of 50k training images and 10k testing images from 10 classes. ImageNet is a large dataset, which contains over 1m training images and 50k testing images from 1000 categories.

Backbone networks & baseline approaches. We conduct experiments based on five main stream CNNs, *i.e.,* VGGNet [146][2], ResNet [54][3], DenseNet [64][4], MobileNet [139][5], and EfficientNet [154][6]. To better demonstrate the effectiveness of our approach in learning lightweight networks, we mainly compare it with state-of-the-art (1) lightweight networks and (2) network compression methods, including L1 [93], SSS [66], Variational Pruning [199], HRank [103],NISP [182], GAL [104], Hinge [97], CNN-FCF [96], Group Lasso [124], L2PF [65], EGL [124] and DEGL [124], DCP-A [205], Slimming [108] and GBN [181].

Implementation. We use PyTorch to implement our network architecture. Following the literature as well as the original code for each network, in our experiments we use the SGD optimizers with the cross-entropy loss and set the initial learning rate, momentum, and decay as 0.05, 0.9, and 0.0005, respectively. The learning rate is divided by 2 every 30 epochs on CIFAR-10 and by 10 every 10 epochs on ImageNet. We use Top-1 error as our performance measure for both datasets. We report our results based on three random

[2]https://pytorch.org/docs/stable/torchvision/models.html
[3]https://pytorch.org/docs/stable/torchvision/models.html
[4]https://pytorch.org/docs/stable/torchvision/models.html
[5]https://github.com/tonylins/pytorch-mobilenet-v2
[6]https://github.com/lukemelas/EfficientNet-PyTorch

trials in terms of mean and standard deviation.

6.4.1 Results Summary

We summarize our results in Table 6.2 based on seven classic network architectures. In general, we use grid search to determine which convolutional layers in the backbone network should be replaced by CSR-Conv layer. Overall, CSR-Conv can be used to learn smaller but better lightweight networks based on different backbones. Specifically,

- CSR-Conv can effectively learn lightweight networks using less than half of the model sizes of the backbone networks with no, or only $< 1\%$ performance loss. On CIFAR-10, CSR-Conv can even achieve $\rho_M > 80\%$ with $1\% \sim 3\%$ performance loss.
- CSR-Conv seems to be able to improve the performance by $0.1\% \sim 1\%$ when $\rho_M < 50\%$.
- CSR-Conv performs stably, as the standard deviation ranging from 0.04% to 0.31%.
- Often more CSR-Conv layers are needed to learn more lightweight networks. Meanwhile, deeper RNNs are desirable for better performance. This validates our motivation.

6.4.2 Comparison with Lightweight Networks

We also compare our CSR-Conv based networks with the state-of-the-art lightweight networks. The comparison results are listed in Table 6.3, where we show 3 CSR-Conv based networks with EfficientNet and MobileNet as our backbones. It is clear that CSR-Conv with EfficientNet has the lowest error among all the networks. The "lighter" models with the MobileNet backbone also have similar or better performance comparing to the networks with similar model sizes such as MUXNet-s and DY-MobileNetV2 x0.35. Note that the DY-MobileNetV2 x0.35 model also uses MobileNetV2 as the backbone network, and our model can achieve significantly better performance with even less parameters.

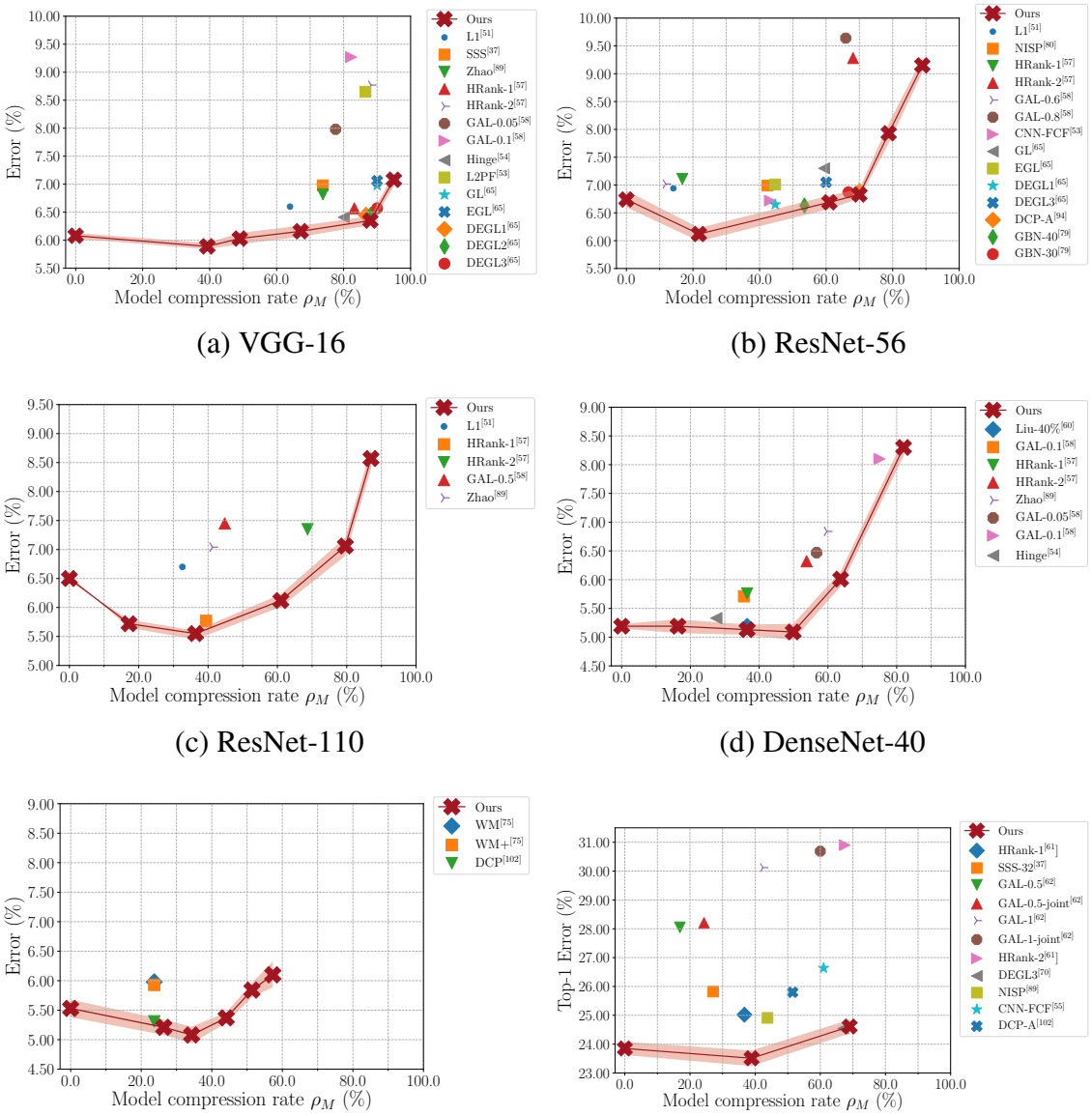


Figure 6.3: Comparison of error *vs.* compression rate on **(a-e)** CIFAR-10 and **(f)** ImageNet.

Table 6.3: Lightweight network comparison on ImageNet in terms of the number of parameters and top-1 error. All the networks with model sizes smaller than 5M are included.

Networks	#Param.	Err. (%)
MUXNet-xs [109]	1.8M	33.3
MUXNet-s [109]	2.4M	28.4
Ours-1 (MobileNet-V2)	**2.5M**	**29.5**
DY-MobileNetV2 x0.35 [21]	2.8M	35.1
Ours-2 (MobileNet-V2)	**3.0M**	**27.7**
ECA-Net [165]	3.3M	27.4
PVTv2-B0 [167]	3.4M	29.5
MUXNet-m [109]	3.4M	24.7
MnasNet-A1 [153]	3.9M	24.7
DY-MobileNetV2 x0.5 [21]	4.0M	30.6
Proxyless [14]	4.0M	25.4
MUXNet-l [109]	4.0M	23.4
MixNet-S [155]	4.1M	24.2
Ours-3 (Efficient-B0)	**4.3M**	**22.3**
GreedyNAS-C [180]	4.7M	23.8
DY-MobileNetV3-Small [21]	4.8M	30.3
MnasNet-A2 [153]	4.8M	24.4
ViTAE-T-Stage [179]	4.8M	23.2
PiT-Ti [57]	4.9M	25.4

This also validates the effectiveness of our CSR-Conv layer. Since these competitors are based on standard linear convolutions, we strongly believe that our CSR-Conv layer can further reduce the model sizes of such networks while preserving (even improving) their performance. Also, post-processing such as pruning can be applied to our networks to achieve smaller networks. See Table 6.6 later for example.

6.4.3 Comparison with Network Compression

Fig. 6.3 illustrates our comparison with the state-of-the-art on both CIFAR-10 and ImageNet, where methods towards the bottom right corner are preferred. We can see the performance trends as discussed above for Table 6.2. Surprisingly, our approach forms "lower-bound" curves in each subfigure, indicating that given similar model compression

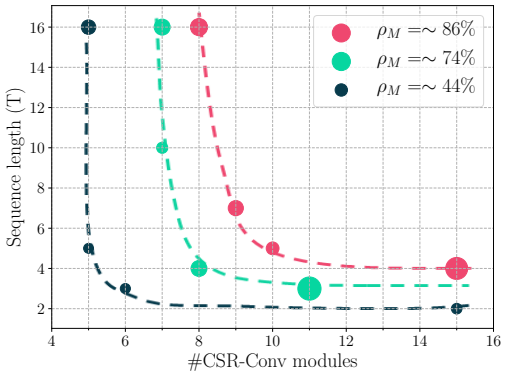


Figure 6.4: VGG-16 error (dot size) on CIFAR-10. Each curve indicates similar ρ_M values.

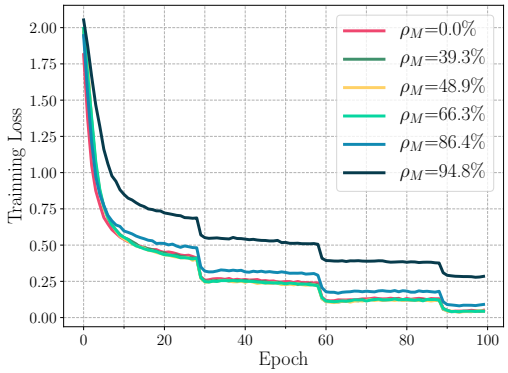


Figure 6.5: Training loss comparison using the VGG-16 backbone on CIFAR-10.

rates CSR-Conv often works best. This is because of the overparameterization in the neural networks so that we have a sufficiently large parameter space to identify a better yet lightweight architecture. Thanks to our design, CSR-Conv has the flexibility of exploring the performance with a specific compression rate. In summary, CSR-Conv can manage to learn lightweight networks effectively, consistently and robustly using different backbone networks on large-scale complicated datasets.

6.4.4 Ablation Study

Impacts of the hidden state transition in vanilla RNNs. The hidden state transition helps construct deeper networks, compared with the backbones, to compensate for the performance loss when learning lightweight networks. To verify this usage, we compare our model with a baseline model with shared weights in group convolutions (denoted as "s-GroupConv"), as illustrated in Fig. 6.1(b), to replace our CSR-RNN layers. We then tune such networks so that the model size compression ratios are approximately the same as ours. We list some results in Table 6.4, where we can see that in all the cases our results are consistently better than this baseline, demonstrating the need of the hidden

state transition.

Impacts of the number of CSR-Conv layers and input sequence length. Recall that we use grid search to seek a lightweight network architecture to meet a certain model size compression rate, if required. We take VGG-16 for example to demonstrate their impacts on the performance, as illustrated in Fig. 6.4. Note that we select convolutional layers in the VGG-16 architecture in descending order. We can see that:

Table 6.4: Top-1 error (%) comparison on CIFAR-10 using VGG-16.

Networks	$\rho_M(\downarrow)$	Ours	s-GroupConv
CSR-Conv-1	39.4%	5.89	6.23
CSR-Conv-2	49.0%	6.01	6.56
CSR-Conv-3	67.2%	6.16	7.03
CSR-Conv-4	86.5%	6.35	7.27
CSR-Conv-5	95.0%	7.08	7.82

- The model compression rates towards the bottom left corner are lower and lower.
- Given the same model size compression rate, the networks form nice "U" shaped contours where more CSR-Conv layers need short sequence length.
- Lightweight networks with small errors, given compression rates, are distributed along the valley.

These observations are very useful as guidance for our approach on how to search for a lightweight network architecture effectively.

RNN variants, GRU, and LSTM as the recurrent layer. Overall, we do not observe any significant performance improvement over the vanilla RNN implementation. For instance, to learn lightweight networks based on VGG-16 with a model compression rate of $\sim$87% on CIFAR-10, vanilla RNN, incremental RNN, and FastRNN achieve 6.35%, 6.45%, and 7.87% in terms of classification error, respectively. Using the same amount of parameters Lipschitz RNN achieves 6.55% error with a model compression rate of $\sim$78%. Similarly, we replace vanilla RNNs with GRUs and LSTMs to learn lightweight networks based on ResNet-56 that achieve (10.9%, 7.53%) and (-18.8%, 7.80%) in terms

of (ρ_M, error) on CIFAR-10, respectively. These results are worse than vanilla RNNs as well, probably due to the short sequence length. Therefore, by default we utilize vanilla RNNs as the recurrent layer in our CSR-Conv.

FLOP reduction. As demonstration, we verify the FLOP reduction of our approach using ResNet-56 in practice and list our results in Table 6.5. Recall that our main focus of this chapter is to learn lightweight networks, and FLOPs tend to decrease as well with the increase of sequence length, in general. For CSR-Conv-1, the input and output dimensions are the same so that $T = 1 + \frac{D}{d}$ holds, and thus no drop in FLOPs exists. Such results in Table 6.5 also verify Prop. 6.3.2 properly.

Running time. Recall that our CSR-Conv layer leads to deeper networks that need to be optimized/inferred sequentially. Therefore, our running time is heavily dependent on the numbers of CSR-Conv layers in the networks, as well as the bottleneck computation in the backbone networks. For instance, the training time is 0.3ms per batch on a Quadro RTX 6000 GPU when we run ResNet-56 on CIFAR-10 dataset. Under the same setting, CSR-Conv-1 (CSR-Conv-5) involves 4 (22) CSR-Conv layers and runs for 0.56ms (1.31ms), with the increase of compression rate from 21.8% to 88.9%. Differently, on ImagetNet the MobileNet-V2 architecture takes 1.068s to train per batch and CSR-Conv-1 (CSR-Conv-2) takes 1.071s (1.096s) that involves 2 (7) CSR-Conv layers.

Table 6.5: Comparison on CIFAR-10.

Networks	Err.(%)	FLOPs(↓)	Param(↓)
ResNet-56	6.74	0.0%	0.0%
CSR-Conv-1	6.12	0.0%	21.8%
CSR-Conv-2	6.69	12.8%	61.0%
CSR-Conv-3	6.83	17.2%	70.3%
CSR-Conv-4	7.93	29.6%	78.8%
CSR-Conv-5	9.15	43.5%	88.9%

Training curves. It is critical to make sure that our lightweight networks are easy to train even with a small portion of parameters and RNNs that share parameters. To demonstrate this, we illustrate our training curves of VGG-16 in Fig. 6.5 where $\rho_M = 0$ denotes the

backbone network and the rest are the variants of our approach. For simplicity, we only plot the training curves of the first 100 epochs. As expected, the networks with a higher compression rate are more difficult to train, leading to larger training losses and test errors. Note that the trends of loss are very similar to each other, indicating that our lightweight yet deeper networks can be trained as easily as backbone networks.

Further compression with existing methods. Note that the learned filters in our CSR-Conv layers are still dense, and thus we can apply network compression methods as post-processing to further reduce the model size. We list some results in Table 6.6 using the classic compression algorithm in [52] to prune our learned lightweight networks. It is apparent that the pruning algorithm can further reduce $\sim 5\%$ of model sizes with marginal $\sim 0.06\%$ error increase. These results show that our CSR-Conv layer can be considered as being orthogonal to the literature of network compression.

Table 6.6: Pruning results on CIFAR-10 based on our learned CSR-Conv networks. Here, VGG-16 and ResNet-56 are two backbone networks.

Network	Err.(%)	$\rho_M(\downarrow)$
CSR-Conv + VGG-16	6.35	86.5%
CSR-Conv + VGG-16 + [52]	6.40	91.9%
CSR-Conv + ResNet-56	6.83	70.3%
CSR-Conv + ResNet-56 + [52]	6.90	75.3%

Chapter 7

Conclusion and Future Work

Complex recurrent models can benefit people in many aspects. In this dissertation, we illustrate this in the following five domains.

1) Human Decision Modeling with Sequential Human Decision Data. In this work, we introduce a novel transit plan evaluation framework that can evaluate the future human behaviors, e.g., ridership and crowd flow of a new transit plan, before its deployment. In this framework, we develop a preference learning algorithm to inversely learn the passengers' preference functions when making transit decisions, and predict future human behaviors of a new transit plan. Our extensive evaluation results using real-world data demonstrated that our framework can predict the ridership with only 19.8% relative error, which is 23%-51% lower than other baseline approaches.

2) Altering Human Decision-Making Process via Reward Advancement. We define and study a novel reward advancement problem, namely, finding the updating rewards to transform human agent's behavior to a predefined target policy π_t. We provide a close-form solution to this problem. The solution we found indicates that there exist infinite many such additional rewards, that can achieve the desired policy transformation. Moreover, we define and investigate a min-cost reward advancement problem, which aims to

find the additional rewards that can transform the agent's policy to π_t, while minimizing the cost of the policy transformation. We solve this problem by developing an efficient algorithm. We demonstrated the correctness and accuracy of our reward advancement solution using both synthetic data and a large-scale (6 months) passenger-level public transit data from Shenzhen, China.

3) Using Sequential Models in General Human Prediction Problems.In this work, we introduce a novel drivers' behavioral prediction framework that can make accurate prediction of driver's future behavior. In this framework, we use driver's historical intra-cycle behaviors to learn driver's preference on his/her decision-making process and then combine the learned preference with inter-cycle features to predict driver's behavior in the future. Our extensive evaluation results based real-world data sets demonstrate that our framework can achieve the prediction accuracy of $MAE = 0.29$, which is on average 13% lower than existing state-of-the-art approaches without using driver's preference. We believe that the idea can benefit domains where both micro (i.e., intra-cycle) and macro (i.e., inter-cycle) decision making process are involved. For example, e-commerce platforms, like Amazon, Alibaba and etc, may apply this methodology to improve the prediction of customers' life time value.

4) Training Stability in Learning Recurrent Models.In this work, we study the problem of training stability in deep RNNs. We propose a novel deep incremental RNN (DIRNN) that has skip connections along both dimensions of time steps as well as network depth. Inspired by recent works such as iRNN, we propose a novel ODE based formulation for DIRNN that can be solved efficiently using Euler's method. We prove that DIRNN is a Lyapunov stable dynamical system where there is no vanishing/exploding gradient in training, and thus leading to good training stability. To the best of our knowledge, we are the first in the literature to provide such theoretical results on the training stability for deep RNNs. To address the model complexity issue in DIRNN, we also propose

a novel implementation, namely TinyRNN, where the transition matrices are sparsified using weighted random permutation matrices to reduce the number of parameters in the network. The learned models can be further compressed using other techniques such as network pruning. We evaluate both RNN models on five different tasks that involve seven benchmark datasets and ten baseline algorithms. Our DIRNN can achieve state-of-the-art accuracy and TinyRNN (with pruning) can achieve the best trade-off between accuracy and model size. Note that our theoretical results can be valuable to analyze the training stability of other networks such as ResNet [54].

5) Lightweight Convolutional Neural Network via Recurrent Convolution. In this work, we aim to address the problem of learning lightweight networks by proposing a novel CSR-Conv layer that replaces traditional linear convolution with channel-split recurrent convolution. The hidden state transition in the vanilla RNNs leads to deeper networks, given backbones, to compensate for the performance loss while reducing the model sizes. Essentially our CSR-Conv can be viewed as the generalization of linear convolution. We show that the model size of a lightweight network decreases *w.r.t.* the number of the duplicate networks with the rate of $O(\frac{1}{T^2})$. We then conduct comprehensive experiments to evaluate our CSR-Conv on CIFAR-10 and ImageNet. We demonstrate state-of-the-art performance on learning lightweight networks in terms of accuracy *vs.* model size.

7.1 Future Work

7.1.1 Causal Inference with Human Decision-making Process

In previous research, I studied the preference of agents when they make different decisions. Further more, we use a real-world case to demonstrate the potential of altering agent's decision by imposing additional rewards. However, the problem of behavior re-

shaping can be viewed as a causal inference problem. The plain preference learning model is a naive single learner model, where we directly model the treatment effect of imposing additional reward on change of agents' future behaviors. Obviously, single learner models suffer from unstable training and poor performance when the ratio of signal and noise is low. In the future work, I intend to combine reward advancement models with causal inference methods such as Double Machine Learning or DragonNet to improve the accuracy and efficiency of reward advancement learning model.

7.1.2 Meta Reward Preference Learning

As shown in previous chapters, we do reward advancement by learning preference of human agents first. In the previous chapter, the preference of one agent is learned using its own trajectories. But agents may share some common preferences. For example, though taxi drivers have their own preference, they tend to go to areas with more potential orders. If those preferences are learned independently, we may ignore those similarities. Also, for some agents, such as new drivers or part-time drivers, we may not have enough trajectories to learn a solid preference. If we can apply meta learning technique, we may be able to learn one agent's preference based on some common preference vectors and only a few trajectories we collected from those particular agents. This definitely can improve the quality and efficiency of learning multiple agents' decision making preference.

Bibliography

[1] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, page 1, 2004.

[2] K. Altun, B. Barshan, and O. Tunçel. Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recognition*, 43(10):3605–3620, 2010.

[3] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[4] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[5] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7120–7129, 2017.

[6] D. Balduzzi and M. Ghifary. Strongly-typed recurrent neural networks. *arXiv preprint arXiv:1602.02218*, 2016.

[7] J. Bao, T. He, S. Ruan, Y. Li, and Y. Zheng. Planning bike lanes based on sharing-bikes' trajectories. In *KDD, 2017*, 2017.

[8] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628, 2013.

[9] A. Beutel, P. Covington, S. Jain, C. Xu, L. Jia, V. Gatto, and E. H. Chi. Latent cross: Making use of context in recurrent recommender systems. In *Eleventh Acm International Conference*, 2018.

[10] D. Birant. Data mining using rfm analysis. In *Knowledge-oriented applications in data mining*. IntechOpen, 2011.

[11] M. Bloem and N. Bambos. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *53rd IEEE Conference on Decision and Control*, pages 4911–4916. IEEE, 2014.

[12] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[13] J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. *CoRR*, abs/1611.01576, 2016.

[14] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[15] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*, 2017.

[16] B. Chang, M. Chen, E. Haber, and E. H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019.

[17] B. Chang, M. Chen, E. Haber, and E. H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019.

[18] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.

[19] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[20] W.-Y. Chen, Y.-F. Liao, and S.-H. Chen. Speech recognition with hierarchical recurrent neural networks. *Pattern Recognition*, 28(6):795–805, 1995.

[21] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

[22] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[23] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[24] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, pages 1–43, 2020.

[25] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[26] J. Collins, J. Sohl-Dickstein, and D. Sussillo. Capacity and Trainability in Recurrent Neural Networks. *arXiv e-prints*, page arXiv:1611.09913, Nov. 2016.

[27] E. Corona, A. Pumarola, G. Alenya, and F. Moreno-Noguer. Context-aware human motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[28] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[30] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[31] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

[32] D. Dennis, D. A. E. Acar, V. Mandikal, V. S. Sadasivan, V. Saligrama, H. V. Simhadri, and P. Jain. Shallow rnn: accurate time-series classification on resource constrained devices. In *Advances in Neural Information Processing Systems*, pages 12896–12906, 2019.

[33] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[35] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[36] J. Drgona, E. Skomski, S. Vasisht, A. Tuor, and D. Vrabie. Spectral analysis and stability of deep neural dynamics. *arXiv preprint arXiv:2011.13492*, 2020.

[37] E. Eban, Y. Movshovitz-Attias, H. Wu, M. Sandler, A. Poon, Y. Idelbayev, and M. A. Carreira-Perpinán. Structured multi-hashing for model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11903–11912, 2020.

[38] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499, 1996.

[39] R. Engelken, F. Wolf, and L. Abbott. Lyapunov spectra of chaotic recurrent neural networks. *arXiv preprint arXiv:2006.02427*, 2020.

[40] N. B. Erichson, O. Azencot, A. Queiruga, and M. W. Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.

[41] S. Fernández, A. Graves, and J. Schmidhuber. Sequence labelling in structured domains with hierarchical recurrent neural networks. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007*, 2007.

[42] C. Finn, P. Christiano, P. Abbeel, and S. Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

[43] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[44] H. Gao, Z. Wang, and S. Ji. Channelnets: Compact and efficient convolutional neural networks via channel-wise convolutions. In *Advances in Neural Information Processing Systems*, pages 5197–5205, 2018.

[45] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[46] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[47] Google GeoCoding. Road map data. `https://developers.google.com/maps/documentation/geocoding/`, 2016.

[48] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[49] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[50] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[51] S. Gui, H. N. Wang, H. Yang, C. Yu, Z. Wang, and J. Liu. Model compression with adversarial robustness: A unified optimization framework. In *Advances in Neural Information Processing Systems*, pages 1285–1296, 2019.

[52] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[53] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[54] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[55] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[56] K. Helfrich, D. Willmott, and Q. Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pages 1969–1978, 2018.

[57] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh. Rethinking spatial dimensions of vision transformers. *arXiv preprint arXiv:2103.16302*, 2021.

[58] M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks. In *Advances in neural information processing systems*, pages 190–198, 2013.

[59] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[60] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[61] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[62] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant. *Applied logistic regression*. 2013.

[63] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[64] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[65] Q. Huang, K. Zhou, S. You, and U. Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018.

[66] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[67] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[68] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[69] H. Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. Technical report, Jacobs University Bremen, 2007.

[70] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks : the official journal of the International Neural Network Society*, 20:335–52, 05 2007.

[71] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[72] L. Jing, Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, Y. LeCun, M. Tegmark, and M. Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning*, pages 1733–1741, 2017.

[73] A. Kag, Z. Zhang, and V. Saligrama. Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients? In *International Conference on Learning Representations*, 2020.

[74] G. Kerg, K. Goyette, M. P. Touzel, G. Gidel, E. Vorontsov, Y. Bengio, and G. Lajoie. Non-normal recurrent neural network (nnrnn): learning long time dependencies while improving expressivity with transient dynamics. In *Advances in Neural Information Processing Systems*, pages 13613–13623, 2019.

[75] H. Kim, M. U. K. Khan, and C.-M. Kyung. Efficient neural network compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12569–12577, 2019.

[76] J. Kim, J. K. Lee, and K. M. Lee. Deeply-recursive convolutional network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1637–1645, 2016.

[77] S.-Y. Kim, T.-S. Jung, E.-H. Suh, and H.-S. Hwang. Customer segmentation and strategy development based on customer lifetime value: A case study. *Expert systems with applications*, 31(1):101–107, 2006.

[78] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[79] G. Konidaris, I. Scheidwasser, and A. Barto. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(May):1333–1371, 2012.

[80] I. Korvigo, M. Holmatov, A. Zaikovskii, and M. Skoblov. Putting hands to rest: efficient deep cnn-rnn architecture for chemical named entity recognition with no hand-crafted rules. *Journal of cheminformatics*, 10(1):1–10, 2018.

[81] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[82] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[83] A. A. Kumar, J. E. Kang, C. Kwon, and A. Nikolaev. Inferring origin-destination pairs and utility-based travel preferences of shared mobility system users in a multimodal environment. *Transportation Research Part B: Methodological*, 2016.

[84] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, 2018.

[85] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pages 9017–9028, 2018.

[86] U. Lachapelle, L. Frank, B. E. Saelens, J. F. Sallis, and T. L. Conway. Commuting by public transit and physical activity: where you live, where you work, and how you get there. *Journal of Physical Activity and Health*, 2011.

[87] X. Lan, X. Zhu, and S. Gong. Knowledge distillation by on-the-fly native ensemble. In *Advances in neural information processing systems*, pages 7517–7527, 2018.

[88] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[89] E. L. Lehmann and G. Casella. *Theory of point estimation*. Springer Science & Business Media, 2006.

[90] T. Lei, Y. Zhang, S. I. Wang, H. Dai, and Y. Artzi. Simple recurrent units for highly parallelizable recurrence. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[91] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *NIPS*, 2011.

[92] M. Lezcano-Casado and D. Martı**********nez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pages 3794–3803, 2019.

[93] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[94] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun. Oicsr: Out-in-channel sparsity regularization for compact deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7046–7055, 2019.

[95] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5457–5466, 2018.

[96] T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3977–3986, 2019.

[97] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[98] Y. Li, S. Lin, J. Liu, Q. Ye, M. Wang, F. Chao, F. Yang, J. Ma, Q. Tian, and R. Ji. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6438–6447, June 2021.

[99] Y. Li, J. Luo, C.-Y. Chow, K.-L. Chan, Y. Ding, and F. Zhang. Growing the charging station network for electric vehicles with trajectory data analytics. In *ICDE*, 2015.

[100] Y. Li, M. Steiner, J. Bao, L. Wang, and T. Zhu. Region sampling and estimation of geosocial data with dynamic range calibration. In *ICDE*, 2014.

[101] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3367–3375, 2015.

[102] S. Liao and B. Yuan. Circconv: A structured convolution with low complexity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4287–4294, 2019.

[103] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538, 2020.

[104] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019.

[105] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 806–814, 2015.

[106] G. Liu, Y. Li, Z.-L. Zhang, J. Luo, and F. Zhang. Citylines: Hybrid hub-and-spoke urban transit system. In *SIGSPATIAL*, 2017.

[107] W. Liu, P. Guo, and L. Ye. A low-delay lightweight recurrent neural network (llrnn) for rotating machinery fault diagnosis. *Sensors*, 19(14):3109, 2019.

[108] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[109] Z. Lu, K. Deb, and V. N. Boddeti. Muxconv: Information multiplexing in convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12044–12053, 2020.

[110] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, volume 30, pages 6231–6239, 2017.

[111] K. D. Maduranga, K. E. Helfrich, and Q. Ye. Complex unitary recurrent neural networks using scaled cayley transform. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4528–4535, 2019.

[112] G. Melis, C. Dyer, and P. Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[113] Z. Mhammedi, A. D. Hellicar, A. Rahman, and J. Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. *CoRR*, abs/1612.00188, 2016.

[114] A. Mujika, F. Meier, and A. Steger. Fast-slow recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 5915–5924, 2017.

[115] D. Neil, J. H. Lee, T. Delbruck, and S.-C. Liu. Delta networks for optimized recurrent network computation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2584–2593. JMLR. org, 2017.

[116] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in neural information processing systems*, pages 3882–3890, 2016.

[117] J. O. Neill. An overview of neural network compression. *arXiv preprint arXiv:2006.03669*, 2020.

[118] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning (ICML)*, volume 99, pages 278–287, 1999.

[119] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[120] T. M. Nguyen, R. G. Baraniuk, A. L. Bertozzi, S. J. Osher, and B. Wang. Momentumrnn: Integrating momentum into recurrent neural networks. *arXiv preprint arXiv:2006.06919*, 2020.

[121] M. Y. Niu, L. Horesh, and I. Chuang. Recurrent neural networks in the eye of differential equations. *arXiv preprint arXiv:1904.12933*, 2019.

[122] P. Ondruska, J. Dequaire, D. Z. Wang, and I. Posner. End-to-end tracking and semantic segmentation using recurrent neural networks. *arXiv preprint arXiv:1604.05091*, 2016.

[123] OpenStreetMap. Road map data. http://www.openstreetmap.org/, 2016.

[124] O. Oyedotun, D. Aouada, and B. Ottersten. Structured compression of deep neural networks with debiased elastic group lasso. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2277–2286, 2020.

[125] M. Pan, Y. Li, X. Zhou, Z. Liu, R. Song, H. Lu, and J. Luo. Dissecting the learning curve of taxi drivers: A data-driven approach. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 783–791. SIAM, 2019.

[126] W. Park, D. Kim, Y. Lu, and M. Cho. Relational knowledge distillation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3967–3976, 2019.

[127] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[128] L. V. Pato, R. Negrinho, and P. M. Q. Aguiar. Seeing without looking: Contextual rescoring of object detections for ap maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[129] M. Paulssen, D. Temme, A. Vij, and J. L. Walker. Values, attitudes and travel behavior: a hierarchical latent variable mixed logit model of travel mode choice. *Transportation*, 2014.

[130] J. Pennington, S. Schoenholz, and S. Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4785–4795. 2017.

[131] P. H. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *31st International Conference on Machine Learning (ICML)*, 2014.

[132] A. Prabhu, A. Farhadi, M. Rastegari, et al. Butterfly transform: An efficient fft based neural architecture design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12024–12033, 2020.

[133] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *29th International Joint Conferences on Artificial Intelligence*, volume 7, pages 2586–2591, 2007.

[134] A. H. Ribeiro, K. Tiels, L. A. Aguirre, and T. Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness. In *International Conference on Artificial Intelligence and Statistics*, pages 2370–2380. PMLR, 2020.

[135] F. Rosenblatt. *Principles of neurodynamics.* Spartan Books, Washington, D.C., 1962.

[136] M. Rotman and L. Wolf. Shuffling recurrent neural networks. *arXiv preprint arXiv:2007.07324*, 2020.

[137] D. Roy, S. Srivastava, P. Jain, A. Kusupati, M. Varma, and A. Arora. Lightweight deep rnns for radar classification. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pages 360–361, 2019.

[138] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. Latent odes for irregularly-sampled time series. *CoRR*, abs/1907.03907, 2019.

[139] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[140] S. Sastry. Lyapunov stability theory. In *Nonlinear Systems*, pages 182–234. Springer, 1999.

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

[142] J. Schulman, X. Chen, and P. Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

[143] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[144] B. Sharma, M. Hickman, and N. Nassir. Park-and-ride lot choice model using random utility maximization and random regret minimization. *Transportation*, 2017.

[145] X. Shi, Z. Chen, H. Wang, D. Y. Yeung, W. K. Wong, and W. C. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 2015:802–810, 2015.

[146] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[147] P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri. Play and prune: Adaptive filter pruning for deep model compression. *arXiv preprint arXiv:1905.04446*, 2019.

[148] C. J. Spoerer, P. McClure, and N. Kriegeskorte. Recurrent convolutional neural networks: a better model of biological object recognition. *Frontiers in psychology*, 8:1551, 2017.

[149] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[150] C. Tai, T. Xiao, Y. Zhang, X. Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[151] Y. Tai, J. Yang, and X. Liu. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3147–3155, 2017.

[152] S. S. Talathi and A. Vartak. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771*, 2015.

[153] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[154] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[155] M. Tan and Q. V. Le. Mixconv: Mixed depthwise convolutional kernels. *arXiv preprint arXiv:1907.09595*, 2019.

[156] S. Tao, D. Rohde, and J. Corcoran. Examining the spatial–temporal dynamics of bus passenger travel behaviour using smart card data and the flow-comap. *Journal of Transport Geography*, 41:21–36, 2014.

[157] U. Thakker, J. Beu, D. Gope, G. Dasika, and M. Mattina. Run-time efficient rnn compression for inference on edge devices. *arXiv preprint arXiv:1906.04886*, 2019.

[158] U. Thakker, J. Beu, D. Gope, C. Zhou, I. Fedorov, G. Dasika, and M. Mattina. Compressing rnns for iot devices by 15-38x using kronecker products. *arXiv preprint arXiv:1906.02876*, 2019.

[159] A. Tuor, J. Drgona, and D. Vrabie. Constrained neural ordinary differential equations with stability guarantees. *arXiv preprint arXiv:2004.10883*, 2020.

[160] J. Urata, Z. Xu, J. Ke, Y. Yin, G. Wu, H. Yang, and J. Ye. Learning ride-sourcing drivers' customer-searching behavior: A dynamic discrete choice approach. *Transportation Research Part C: Emerging Technologies*, 130:103293, 2021.

[161] A. Vahedian, X. Zhou, L. Tong, Y. Li, and J. Luo. Forecasting gathering events through continuous destination prediction on big trajectory data. In *SIGSPATIAL*, 2017.

[162] R. Vogt, M. P. Touzel, E. Shlizerman, and G. Lajoie. On lyapunov exponents for rnns: Understanding information propagation using dynamical systems tools. *arXiv preprint arXiv:2006.14123*, 2020.

[163] J. Wang and X. Hu. Gated recurrent convolution neural network for ocr. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 334–343, 2017.

[164] P. Wang, Y. Fu, G. Liu, W. Hu, and C. Aggarwal. Human mobility synchronization and trip purpose detection with mixture of hawkes processes. In *KDD*, 2017.

[165] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu. Eca-net: Efficient channel attention for deep convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[166] S. Wang, L. He, L. Stenneth, P. S. Yu, and Z. Li. Citywide traffic congestion estimation with social media. In *SIGSPATIAL, 2015*, 2015.

[167] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao. Pvtv2: Improved baselines with pyramid vision transformer. *arXiv preprint arXiv:2106.13797*, 2021.

[168] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[169] E. Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.

[170] R. Wong, W. Szeto, and S. Wong. A two-stage approach to modeling vacant taxi movements. *Transportation Research Part C: Emerging Technologies*, 59:147–163, 2015.

[171] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9127–9135, 2018.

[172] G. Wu, Y. Ding, Y. Li, J. Bao, Y. Zheng, and J. Luo. Mining spatio-temporal reachable regions over massive trajectory data. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 1283–1294. IEEE, 2017.

[173] G. Wu, Y. Ding, Y. Li, J. Luo, F. Zhang, and J. Fu. Data-driven inverse learning of passenger preferences in urban public transits. In *56th IEEE Conference on Decision and Control*, 2017.

[174] G. Wu, Y. Li, J. Bao, Y. Zheng, J. Ye, and J. Luo. Human-centric urban transit evaluation and planning. In *IEEE International Conference on Data Mining (ICDM)*, pages 547–556. IEEE, 2018.

[175] G. Wu, Y. Li, J. Bao, Y. Zheng, J. Ye, and J. Luo. Human-centric urban transit evaluation and planning. In *IEEE International Conference on Data Mining, 2018*, 2018.

[176] G. Wu, Y. Li, and J. Luo. Transforming policy via reward advancement. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4609–4614. IEEE, 2019.

[177] G. Wu, Y. Li, S. Luo, G. Song, Q. Wang, J. He, J. Ye, X. Qie, and H. Zhu. A joint inverse reinforcement learning and deep learning model for drivers' behavioral prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2805–2812, 2020.

[178] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[179] Y. Xu, Q. Zhang, J. Zhang, and D. Tao. Vitae: Vision transformer advanced by exploring intrinsic inductive bias. *arXiv preprint arXiv:2106.03348*, 2021.

[180] S. You, T. Huang, M. Yang, F. Wang, C. Qian, and C. Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008, 2020.

[181] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2133–2144, 2019.

[182] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[183] X. Yu, T. Liu, X. Wang, and D. Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[184] N. J. Yuan, Y. Wang, F. Zhang, X. Xie, and G. Sun. Reconstructing individual mobility from smart card transactions: A space alignment approach. In *ICDM, 2013*, 2013.

[185] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[186] C. Zhang, P. Patras, and H. Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(3):2224–2287, 2019.

[187] C. Zhang, G. Zhou, Q. Yuan, H. Zhuang, Y. Zheng, L. Kaplan, S. Wang, and J. Han. Geoburst: Real-time local event detection in geo-tagged tweet streams. In *SIGIR*, 2016.

[188] J. Zhang, Q. Lei, and I. S. Dhillon. Stabilizing gradients for deep neural networks via efficient svd parameterization. In *ICML*, 2018.

[189] J. Zhang, X. Wang, D. Li, and Y. Wang. Dynamically hierarchy revolution: dirnet for compressing recurrent neural network on mobile devices. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3089–3096, 2018.

[190] J. Zhang and H. Zuo. A deep rnn for ct image reconstruction. In *Medical Imaging 2020: Physics of Medical Imaging*, volume 11312, page 113124N. International Society for Optics and Photonics, 2020.

[191] L. Zhang. Agent-based behavioral model of spatial learning and route choice. Technical report, 2006.

[192] X. Zhang, Y. Li, X. Zhou, and J. Luo. Unveiling taxi drivers' strategies via cgail – conditional generative adversarial imitation learning. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1–6. IEEE, 2019.

[193] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[194] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015.

[195] Z. Zhang, G. Wu, Y. Yue, Y. Li, and X. Zhou. Deep incremental rnn for learning sequential data: A lyapunov stable dynamical system. In *IEEE International Conference on Data Mining (ICDM)*. IEEE, 2021.

[196] Z. Zhang, Y. Yue, G. Wu, Y. Li, and H. Zhang. Sbo-rnn: Reformulating recurrent neural networks via stochastic bilevel optimization. In *Thirty-fifth Conference on Neural Information Processing Systems(NeurIPS)*, 2021.

[197] B. Zhao, X. Li, and X. Lu. Cam-rnn: Co-attention model based rnn for video captioning. *IEEE Transactions on Image Processing*, 28(11):5552–5565, 2019.

[198] B. Zhao, X. Li, and X. Lu. Tth-rnn: Tensor-train hierarchical recurrent neural network for video summarization. *IEEE Transactions on Industrial Electronics*, 2020.

[199] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2780–2789, 2019.

[200] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, and Z. Zhang. Building efficient deep neural networks with unitary group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11303–11312, 2019.

[201] J. Zheng and L. M. Ni. Modeling heterogeneous routing decisions in trajectories for driving experience learning. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 951–961. ACM, 2014.

[202] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2014.

[203] Z. Zhong, Y. Gao, Y. Zheng, and B. Zheng. Efficient spatio-temporal recurrent neural network for video deblurring.

[204] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[205] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018.

[206] B. D. Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy (CMU PhD dissertation). 2010.

[207] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[208] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber. Recurrent highway networks. In *ICML*, pages 4189–4198. JMLR. org, 2017.