Learning-based Lane Following and Changing Behaviors for Autonomous Vehicle

Yilun Chen

CMU-RI-TR-18-26

May 17, 2018

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
John Dolan (co-chair)
Katharina Muelling (co-chair)
David Held
Wenhao Luo

Submitted in partial fulfillment of the requirements for the degree of Masters of Science in Robotics.

For my family

Abstract

This thesis explores learning-based methods in generating human-like lane following and changing behaviors in on-road autonomous driving. We summarize our main contributions: 1) derive an efficient vision-based end-to-end learning system for on-road driving; 2) propose a novel attention-based learning architecture with hierarchical action space to obtain lane changing behavior using a deep reinforcement learning algorithm; 3) use LSTM to make vehicle's trajectory prediction with the demonstration of human driving trajectories. We first propose an end-to-end imitation learning algorithm to teach the car how to drive on-road with visual input. The elementary principle is to construct a neural network that maps from image input to steering angle and acceleration. To improve the maneuver's stability and boost the training efficiency, we apply transfer learning from other tasks, use LSTM to add temporal information, supplement with segmentation results and add sensor fusion. We evaluate our model in the Udacity simulator and obtain smooth driving performance on unseen curvy maps. Later, we extend learning lane following task to the lane changing task by using deep reinforcement learning. This approach avoids direct human supervision in a model-free fashion, easing the effort of extensive annotated training data. The contribution here is that we formulate the lane change behavior as a hierarchical action and we propose a model to solve deep reinforcement learning in this high-dimensional, structured space. In the meantime, we explore the attention mechanism in deep reinforcement learning, and the observed behavior is improved after applying spatial and temporal attention. The overall algorithm is tested and evaluated in the TORCS platform. Finally, we fulfill the task of trajectory prediction in on-road driving. The aim is to discover when and how people would make the decision of lane changing. We divide the task into predicting the driver's discrete intention and forecasting the subsequent continuous trajectory. We solve this sequential prediction task with LSTM and further extend the model by capturing the surrounding environment information. We compare and evaluate our prediction results with real human driving trajectories in the NGSIM dataset.

Acknowledgments

First and foremost, I would like to express my profound gratitude for my advisors, Dr. John Dolan and Dr. Katharina Muelling. I'm truly thankful for their patience, continuous support and valuable guidance during my two-year master study at Carnegie Mellon. It is a great honor to join the GM-CMU autonomous driving collaborative research project, which introduces me to the amazing field of autonomous driving.

I would like to thank my collaborators at General Motors, Dr. Priyantha Mudalige and Praveen Palanisamy. During our bi-week meeting, we have had lots of meaningful discussions, which inspired lots of interesting ideas.

I would like to thank my committee members, Dr. David Held and Wenhao Luo, who have given me lots of precious advice on my research and thesis writing.

I am also extremely thankful to my amazing fellow labmates for an enthusiastic and cooperative lab environment. I would give special thanks to Chiyu Dong, who have helped me grow both inside and outside research, Zhiqian Qiao, whom I have gone through all the up and downs for the GM project with. I am also grateful for Shuang Su and Shivam Gautam, whose work I built upon. It is a privilege to meet you guys and I have learned so much from every one of you. I enjoyed and treasured all the fun and precious time we have together.

Finally, I would like to thank my parents for their endless support and encouragement. I would not have accomplished any of these without your unconditional love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Autonomous Driving

Fully autonomous driving vehicles have been widely researched in academia for decades. One of the first successful demonstrations of an autonomous driving vehicle goes back to 1986 [9]. At that time, simulated road images were used to train a three-layer neural network and produce as output the direction the car should follow on-road. Since then, a wide variety of aspects of autonomous driving have been investigated and applied including perception, localization, planning, and control. People gradually began to focus on more realistic urban settings with uncertain traffic environment, which is currently still an important goal. One of the most influential and important milestones that have been reached was the 2007 DARPA Urban Challenge [10]. In this challenge, teams from across the world designed both hardware and software for autonomous vehicles that could handle dynamic obstacles, merging scenarios and intersections. Six robots out of 50 completed the race, after driving on a closed route for an entire day.

The rapid global urbanization in the recent past has led to severe road congestion, a rise in pollution levels and an increase in road accidents. Private automobiles are always the first to blame because of its unsustainable feature on personal urban mobility. Fortunately, great strides have been made in the development of autonomous driving technologies. As the autonomous agent takes over the driving task, the human driver becomes a passenger during the autonomous journey, can take his/her hands off the steering wheel and pedals and is free to pursue other activities. The reason that autonomous driving is widely and continuously researched and investigated is that it has the potential to reduce accidents and be economically beneficial and environmentally friendly. While providing an opportunity to develop safe and sustainable solutions to personal mobility [11], autonomous vehicles also may allow transportation sharing as Autonomous Vehicles-on-Demand [12].

Thanks to recent advances in artificial intelligence technology, Automotive Driver Assistance Systems (ADAS) have been making rapid progress over the past few years. More and more vehicles are equipped with increasingly sophisticated ADAS systems. Currently, numerous research groups and companies are working hard on bringing this technology to market. Some of them are shown in Figure 1.1. Google's driverless car project, now called Waymo, is one of the leaders, with a history of developing self-driving technology since 2009. With a total test driving distance

of over 4 million miles on public roads, Waymo has already set up an early public riders program in Phoenix. At the same time, Uber has done extensive road tests in Pittsburgh and plans to equip 30,000 more autonomous cars on the road. Tesla launched its Autopilot in 2014 and is working towards a full self-driving demonstration by the end of 2019. Other major car companies like GM, Ford, Toyota, and Honda have also set up their autonomous driving departments and are making continuous progress on that.

Figure 1.1: Some well-known autonomous driving vehicles from academia and industry, from left to right and top to bottom: CMU BOSS self-driving car for 2007 Darpa Challenge [1], Uber self-driving car [2], Waymo self-driving car [3], Cruise self-driving car [4].

A typical autonomous driving vehicle is a complex system consisting of several subsystems [13]. The perception subsystem fuses data collected from multiple sensors like camera, Lidar, Radar, GPS and IMU to obtain a semantic understanding of the world. The perception subsystem should be able to provide locations of obstacles/vehicles on the road as well as ego vehicle's position relative to the road. The prediction subsystem makes predictions of the surrounding objects in a short period of time. This inference helps our ego vehicle to better understand the future environment we may encounter. Finally, the decision-making subsytem judges all the information and inference given by perception and prediction subsytem, and decide on the exact trajectory the car should next follow. The overall software system should be able to ensure coordination of different subsystems and be robust for real-time processing.

1.2 Decision Making for Autonomous Driving

In this section, we describe the decision making architecture often used in an autonomous car. The decision-making system of a self-driving car can be hierarchically decomposed into four

components [14] as shown in Figure 1.2: The highest level is a **route/mission planner** that navigates and plans the route to the destination through the road network. The next level is the **behavior planner**, which decides on a local driving task while obeying the traffic rules and progressing towards the final destination. The **motion planner** then follows the strategy given by the behavior planner and generates a continuous path/trajectory. Finally, the **control planner** will use the trajectory to generate the execution commands like acceleration, throttle, and steering angle.

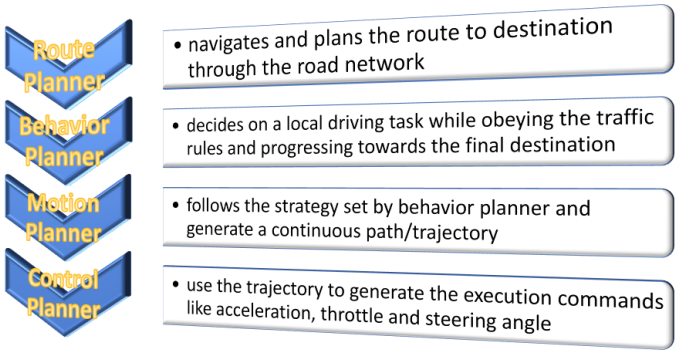

Figure 1.2: Current decision making hiearchy for autonomous driving.

1. **Route planner**: As the highest level of the decision-making system, the route planner selects the best route from the starting position to the destination. Since the road network can be seen as a connected graph, selecting the optimal route is equivalent to finding the lowest-cost path under certain restrictions given by traffic rules. People have done intensive investigations in finding proper routes in large-scale maps: see [15] for details.
2. **Behavior planner**: As an intermediate level between route and trajectory, behavior-level planning needs to interact with surrounding traffic and environment according to driving rules on the road. The selected behavior must be a safe yet efficient driving behavior with consideration of all other traffic participants. For example, if a car wants to exit the highway, it needs to navigate from its lane to the rightmost lane without interfering with the normal driving of all other vehicles on the road. The behavior can be high-level instructions like cruise-in-lane, change-lane, turn-left.

   Previous methods mostly focused on designing the behaviors as a finite set and coupled different driving scenarios with different heuristic behaviors. Then a finite state machine is used to transition between each behavior, like most teams in the DARPA Urban Challenge [10]. However, real-world driving is full of uncertainty. To solve this problem, machine learning-based techniques and probabilistic planning formalisms, like Markov Decision Processes (MDP) and its generalization have recently been used to deal with this ambiguity [16].
3. **Motion planner**: The motion planning module follows the strategy given by the behavior planner and derives a continuous trajectory that will be executed by the low-level controller. The motion planner needs to generate a trajectory physically feasible for the vehicle, safe for the traffic and comfortable for the passenger. The trajectory should avoid any

possible collisions with any obstacle or other traffic participants on the road.

The problem of motion planning is to search for a feasible solution in the vast configuration space, which is computationally intractable in most cases. Thus, most popular motion planning methods use a numerical approximation. The two most common approaches are graph-based planning and sample-based planning. The graph-based planner [17] discretizes the vehicle's drivable path into a connected graph and searches for the shortest path using graph search. The sample-based planner [18] searches for a collision-free path by sampling points on the feasible path until a viable trajectory is found.

4. **Control planner**: The control planner is used to select appropriate actuation to execute the planned trajectory and correct tracking errors. The inaccuracy of the tracking is accumulated during the execution due to the inaccuracy of the actual vehicle model. The control module should be able to correct this deviation and act as we command. A good control planner ensures robustness and stability in the closed-loop decision-making. Popular control planner includes pure pursuit method [19] and MPC (Model Predictive Control) methods [20].

In this thesis, we will focus mostly on the behavior planning module and motion planning module. We will also break the boundary of the traditional division of the decision-making hierarchy and use a more end-to-end approach for the autonomous car decision making.

1.3 Approaches

In this thesis, we first propose a vision-based end-to-end approach to learn a maneuver controller for autonomous vehicles. The basic idea is to learn a complicated mapping that can go directly from sensor input, images, to the control output, steering angle. This is done by using a deep neural network and leveraging the availability of a significant amount of human driving data as supervision. Based on the architecture described in [7], we propose several novel ideas to improve the data efficiency and stability of the end-to-end learning approach. We first use transfer learning to incorporate prior knowledge into the driving task. We add temporal information with LSTM to make predictions based on current and past states. Image segmentation information is also used as an auxiliary task for learning the driving behavior. Finally, we also apply sensor fusion with vehicle kinematics to better represent vehicle state. The resulting model can learn the driving policy with much fewer data and higher accuracy in imitating the driving supervision data. Finally, we apply this model to several unseen environments and see it can quickly adapt to new unseen environments.

Secondly, we extend the imitation learning-based method to a deep reinforcement learning-based method applied to the task of learning to drive, which can avoid a significant amount of human driving data that are both hard to collect and expensive in reality. Previously, there is already literature using DRL for driving [21]. In this thesis, we make two major improvements. First, most previous work focuses on the lane-keeping task in autonomous driving, whereas here we focus on lane changing behavior. To accomplish this, we divide the behavior of driving into a hierarchy. We first choose high-level decisions of the left lane change, right lane change or lane following and then execute low-level direct control parameters. The resulting system can be

seen as having subpolicies instead of an overall policy. Second, we try to explore the attention mechanism of driving. As human drivers pay attention to only part of the road configuration when driving, we want to extract the portion from features that are most valuable for making the decision. This is done by designing a new layer in a neural network that can automatically select the parts that have a more substantial impact on training. Our results show better lane changing behavior and faster and smoother overall training.

Finally, we investigate the task of predicting lane change behavior for on-road driving. For our ego autonomous car, it is essential to understand and forecast other on-road vehicles' behavior to obey traffic rules and drive safely alongside. Then this problem can be formulated as follows: given the past environment states in a time window, can we predict the future actions of other vehicles? Encouraged by the promising result of recurrent networks for integrating past information, we construct an LSTM (long-short-term-memory)-based method for lane change prediction. We further subdivide the problem into a discrete prediction for the lane change category and then a continuous prediction for the whole trajectory. To make the prediction more accurate, besides its own past states, we also consider the past states of its surrounding vehicles to know their influence on the car's behavior. The experiment is done on the NGSIM dataset, from which we can obtain hundreds of real lane change scenarios with the vehicle's trajectory. The method's prediction results are compared with the true human driving trajectories to see how well it can recover humans' behavior.

The thesis is organized as follows: In Chapter 2 we briefly go through the preliminary background, including the formal definition of deep reinforcement learning, End-to-end learning and the basics of Long Short Term Memory. In Chapter 3, we introduce our End-to-end learning of driving policy, the novel methods we used and the experiment results. In Chapter 4, we present the deep reinforcement learning perspective of learning driving behavior. In Chapter 5, we investigate the trajectory prediction problem, show our LSTM-based approach, and repeat our experimental results with a comparison of human driving statistics in the NGSIM dataset. Finally, we conclude our work in Chapter 6 and address some of the interesting directions of future work.

Chapter 2

Background

2.1 Deep Reinforcement Learning

Basics of Reinforcement Learning

Consider a typical Reinforcement Learning setup, where our agent acts in an environment E. At every discrete time step t, the agent observes a state$s_t \in S$, picks an action $a_t \in \mathcal{A}$ and receives a scalar reward $r(s_t, a_t) \in \mathbb{R}$. This is formally known as a Markov Decision Process (MDP) defined by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$ of set of states, actions, transitional probabilities, rewards and a discount factor to keep the expectations finite in the case of an MDP without terminal states. The policy is defined as a distribution of actions given the state s

$$\pi(a|s) = P(A_t = a|S_t = s).$$

The total return is defined as total discounted reward at time step t with discount factor γ

$$G_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i).$$

The goal of RL is to find a policy that maximizes this discounted expected return. So following a policy π the objective function can then be formulated as

$$J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[G_1].$$

The value function V^π and action-value function Q^π describe the expected sum of future rewards following the policy π and can be expressed recursively with the Bellman equation.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(S_{t+1})|S_t = s] \\ Q^\pi(s, a) &= \mathbb{E}_\pi[r_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \end{aligned}$$

Deep reinforcement learning uses deep neural networks to represent value function, policy or the model. It optimizes the objective function by stochastic gradient descent.

Deep Q-Network

Deep Q-Network[22] (DQN) is a recent reinforcement learning algorithm based on the popular Q-learning algorithm. Q-learning is an off-policy model-free algorithm, where an agent learns an approximated Q function and follows a greedy policy at each time step. The Q-learning algorithm is based on the Bellman equation:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha[r + \gamma \arg\max_{a'} Q_t(s',a') - Q_t(s,a)]$$

. The Q-learning algorithm starts from an initial state and explores the state space until the episode ends. In every time step, the agent tries to optimize the current loss function L_t and updates the Q function parameterized by θ_t.

$$L_t(\theta_t) = \mathbb{E}_{s,a,r,s'}(y_t - Q_{\theta_t}(s_t,a_t))^2$$

The algorithm continues until the convergence of the Q function.

Deep Q-Network extends the Q-Learning algorithm by estimating the Q function with a deep neural network. The algorithm has been shown to be able to match human performance in the game of Atari by using just image pixels as input state [23]. In order to overcome the problem that using nonlinear function approximators for the Q function would lead to unstable learning in practice, DQN applies two novel modifications called *replay memory* and *target network*. Replay memory will break the correlations in the training data and make sampling much more efficient. The target network freezes updates to a certain frequency, which can stablize training in practice.

Deep Deterministic Policy Gradient

The DQN algorithm can only solve applications that have a discrete action space. Later on, the Deep Deterministic Policy Gradient (DDPG) algorithm was proposed to operate on continuous actions. DDPG is an off-policy actor-critic method that originates from Deterministic Policy Gradient [24].

In the previous method, we approximate the action-value function using parameter θ. Here we directly parametrize the policy: $\pi_\theta(s,a) = P[a|s,\theta]$. Policy gradient methods perform gradient ascent on the policy objective function J with respect to the parameters θ of policy π:

$$\nabla_\theta J(\theta) = \mathcal{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_t \theta}(s,a)].$$

The Actor-Critic architecture uses two parts to optimize the objective function J. We define θ^μ, θ^Q as the parameters for approximating the Actor model function μ and the Critic model function Q respectively. The Actor and Critic work together and are trained together in the algorithm.

The Critic estimates the value of the current policy given the current state and action. It tries to optimize the following loss function in a single update

$$L(\theta^Q) = (r(s_t,a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) - Q(s_t,a_t|\theta^Q))^2.$$

The Actor defines the current policy and learns a mapping from state space to actions. The Actor updates the policy in a direction that improves Q:

$$\nabla_{\theta^\mu} J = \nabla_a Q(s, a|\theta^Q) \nabla_{\theta^\mu} \mu(s|\theta^\mu).$$

The Deterministic Policy Gradient methods uses policy gradients on the actor-critic architecture. The weight updates for actor and critic are as follows:

$$\theta^\mu_{t+1} = \theta^\mu_t + \alpha^\mu \nabla_{\theta^\mu} \mu_(s_t) \nabla_a Q(s_t, a^t)|_{a=\mu(s)}$$
$$\theta^Q_{t+1} = \theta^Q_t + \alpha^Q \delta_t \nabla_{\theta^Q} Q(s_t, a_t).$$

2.2 Long Short Term Memory

Traditional neural network only considers state space in one time step. In order to incorporate temporal information, the notion of recurrent neural network is introduced. The expectation is to let the network have a memory module so that it can store historical information. RNN is designed to connect and combine history information with current state, so that the prediction can be based on both.

However, one of the main issue of vanilla recurrent neural network is that RNN suffers from gradient vanishing when we want to look far back into the history. Experiments shows the memory capacity is only limited to remember a few steps before. To resolve the problem, the Long Short Term Memory(LSTM) network is proposed by [25]. LSTM is a specially designed series of networks based on recurrent module that interacts in a way to control the information flow passed by the network. Note that there are many modified version of LSTM, but in this section, we only introduce the standard one. An unfolded decomposition of the LSTM unit is shown in Figure 2.1.

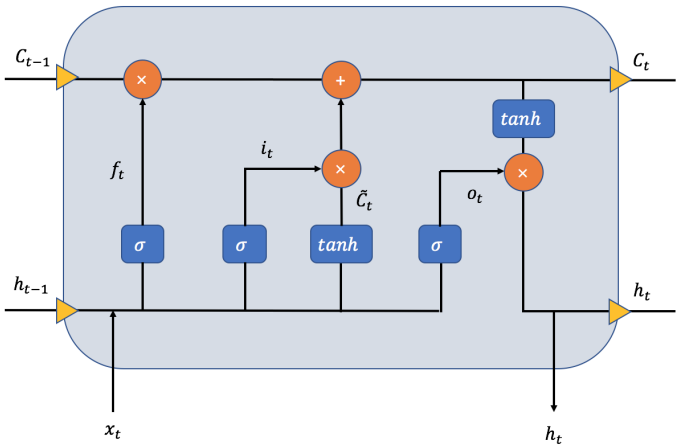

Figure 2.1: A decomposition illustration of LSTM unit.

The LSTM consists of two variables that are passed by through time, the hidden state h and the cell state C. Several interaction functions are commonly regarded as gates.

The *forget gate* combines the information passed by hidden states at previous time step h_{t-1} and the current input x_t. It decides which part of the information should be thrown away.

$$f_t = \sigma(w_f[h_{t-1}, x_t]^T + b_f)$$

The *input gate* adds new information to the cell. It contains two streams i_t and $\tilde{C}_t$. The first sigmoid layer i_t decides how much each element should contribute to the new cell state. The second tanh layer proposes new candidates values for the cell state.

$$\begin{aligned} i_t &= \sigma(w_i[h_{t-1,x_t}]^T + b_i) \\ \tilde{C}_t &= \tanh(w_c[h_{t-1}, x_t]^T + b_c) \end{aligned}$$

The cell state C_t is updated by combining the old cell and the new information scaled by the forget gate and input gate.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{2.1}$$

The hidden state h_t is updated by a point-wise multiply of the tanh clipped C_t and output vector o_t. The output vector o_t is obtained through a sigmoid function.

$$\begin{aligned} o_t &= \sigma(w_o[h_{t-1}, x_t]^T + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned}$$

It should be mentioned that all the parameters in the LSTM unit are derivable and can be updated through training. Thus the LSTM automatically fits in the neural network training with backpropogation.

2.3 End-to-end Learning

With the rapid growth of deep learning technology, end-to-end learning machines have appeared as a common solution to solve complex practical robotic systems. In robotics literature, end-to-end process refers to a robot or an agent consisting of only one network without modularization from sensors to motors.

The definition of end-to-end learning is in comparison with traditional methods that divide a task into several modules. Traditional methods would break the system into several fundamental building blocks, solve each one separately and then optimize the pipeline jointly. However, end-to-end learning machines enable a direct mapping from raw sensor input to the desired output only using one network. In an end-to-end system, all parameters are trained at one time jointly, with a comparison of step-by-step in traditional methods.

End-to-end learning has shown great success in many fields. Mnih [22] has surpassed human-level performance in Atari game by training an end-to-end deep reinforcement learning agent. Sergey [26] trained an end-to-end policy for a PR2 robot to learning grasp from the pure visual input. Deep Speech [27] replaces the entire speech recognition pipeline using hand-designed features with neural networks and results in a more robust model in environments with different noise, accents, and languages.

End-to-end learning has also emerged as a new approach for self-driving cars [7]. In order to learn driving on the road, a traditional approach would involve object and lane detection in an image, path planning to calculate trajectory and PID control for final control output. However, an end-to-end approach would learn a single network to map from sensor perception to steering angles. Despite the less effort on engineering hand-crafted features, the end-to-end approach relies on large amounts of data but gets better performance than traditional methods.

2.4 Lane Change Behavior

Learning the maneuver of a lane change behavior can be seen as a subproblem of the overall decision making for an autonomous vehicle. The reason to consider a lane change can be either to reach the desired speed or to move to the correct lane for the planned destination. For lane change behavior, the autonomous car has to choose the best position and time to perform the action of lane change while ensuring the passenger's safety and comfort. A typical lane change behavior can be seen as three steps: have the intention of lane change, check the feasibility of lane change and finally execute lane change. A critical issue is to consider the surrounding traffic participants' intentions when making our own decision. The decision making of lane change should be seen as an interactive process of negotiation with other vehicles on the road.

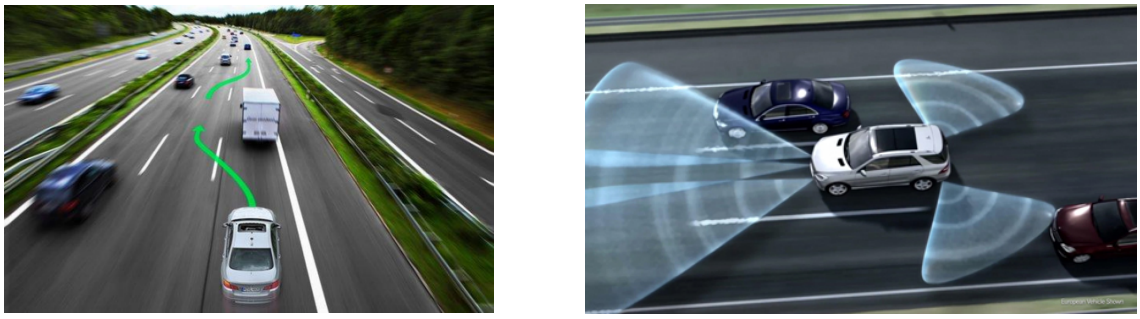

Figure 2.2: Demonstration of (a) a lane change need for higher speed in highway driving [5], and (b) Visualization of perception for considering a lane change behavior [6].

Currently, it does not exist a universal framework to represent and tackle lane changing problem. Several popular categories of approaches and methodologies are summarized as follow:

- **The rule-based methods** check empty slots on the road according to kinematic information. Dolan [28] developed CMU Boss's self-driving vehicle using this slot-based method, which won the first place in 2007 Darpa Urban Challenge. These methods follow certain deterministic predefined rules and are straightforward and simple to implement. However, they do not consider the uncertainty of the environment and the vehicles on the road.
- **The MPC-based methods** formulate the lane changing problem as an optimization problem and solve by minimizing some predefined cost function. These methods mainly use Model Predictive Control(MPC) to tackle lane change behavior [29, 30]. However, these methods lacks global awareness and can get stuck in local minimum. It has no deterministic termination time guarantee.

- **The MDP-based methods** use MDP or POMDP to represent the lane changing behavior on the road. Sadigh [31] uses POMDP to model the interaction between ego vehicle and target vehicles on the road and then solve the POMDP using reinforcement learning algorithms. The theory framework of these works is promising, but currently, it is limited to simulation and cannot scale up to real, complex driving scenarios.
- **The sample-based methods** predefine a pool of possible lane change maneuvers and select behavior by matching the most similar one with the current scenario. Lee [32] uses Gaussian processes to generate a series of candidate trajectories and applies inverse reinforcement learning to select the trajectory with the highest reward. Yoshida [33] predefines several maneuver templates and does matching and selection at the testing time to select the right behavior pattern. This approach is limited by the number of predefined driving styles and can be computationally too expensive.
- **The learning-based methods** learn driving behavior from real human driving data. Learning through the demonstration of other vehicles' trajectories on the NGSIM dataset, Jianqiang [34] uses an SVM with hand-designed features, and Tobias [35] uses a graphical model, to predict the discrete intentions of lane change behavior. This approach can generate human-like behavior but may suffer from overfitting. Also, a significant amount of labeled data are required, which may not always be possible.

Chapter 3

Learning Lane Following Behavior with End-to-end Behavior Cloning

3.1 System Overview

In this chapter, we present an end-to-end learning approach to produce steering angles from raw image input and make the car remain in the targeted lane. Instead of dividing the problem into two separate problems, a perception and a steering angle prediction problem, we follow the principle of end-to-end learning and learn directly to control the car from the image input. More specifically, we are interested in training a deep neural network and using a supervised learning method to teach a vehicle how to follow the lane of the road.

This approach has been suggested and successfully demonstrated by [7]. However, the approach in its original form still suffers from a few drawbacks: (1) the actions chosen are not completely consistent in subsequent frames, which makes the car sometimes have small oscillation on road. (2) the method needs a large amount of training data and the whole training time is long. Here, we address these shortcomings and propose and investigate alternative architectures to greatly boost both the performance and stability of the algorithm.

Our proposed network structure is shown in Figure 3.1. We get raw RGB images from the camera mounted in the front of the autonomous vehicle. First, we do some data preprocessing to increase the diversity of data. Then the preprocessed data are fed into a pre-trained segmentation network. From the segmentation network, we get the segmentation map of the particular image. Then a traditional convolutional network is used to extract necessary features for learning the proper control parameters. After the convolutional layer, we have an LSTM network to incorporate temporal dynamics of the system. The LSTM network here is designed to remember past states of the road configuration. Then finally, we add additional vehicle information, to concatenate the learned features with necessary vehicle kinematics. This concatenated vector is then fed into a fully connected layer to finally learn the continuous steering angle.

Our system differs from the original network structure in four aspects: (1) the overall system take advantage of additional information by first obtaining a segmentation map instead of directly use the implicit raw image; (2) our learning system transfers knowledge from other task and hence speeds up training; (3) we consider the temporal information by adding recurrent

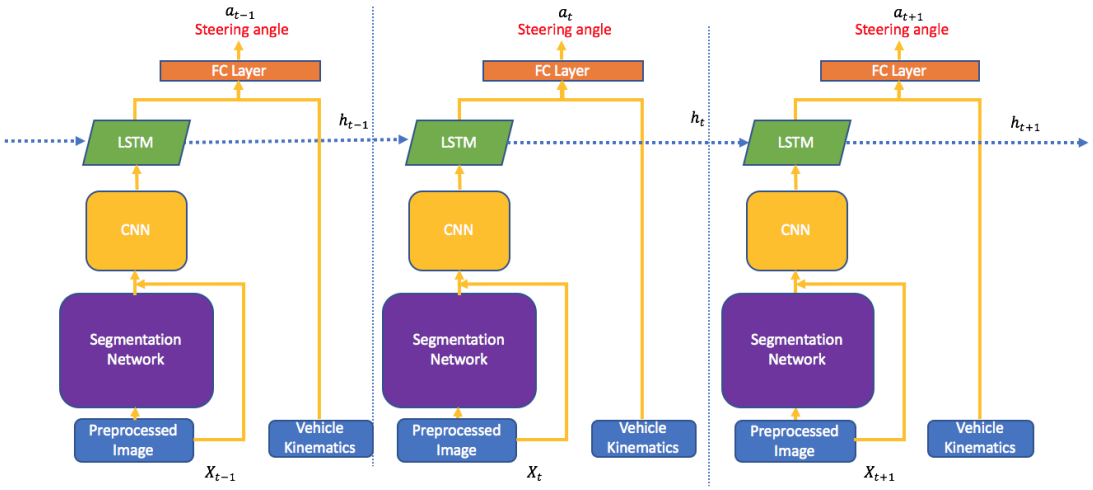

Figure 3.1: The overall network structure of our proposed end-to-end learning architecture. Compared to the original network structure proposed by [7], our network has additional modules of segmentation network, LSTM and vehicle kinematic input.

module into the network; (4) we use vehicle kinematics information additional to image input. The finalized network improves the baseline pure convolutional network by incorporating more information, both from human knowledge and history states.

3.2 Proposed Methods

In this section, we will introduce our proposed methods upon the original network in detail. The methods we discuss here includes using auxiliary segmentation, transferring from existing tasks, utilizing temporal information and incorporating vehicle information.

3.2.1 Auxiliary Segmentation

Image segmentation has been widely researched for decades. Image segmentation is the process of partitioning an image into multiple sets of pixels to simplify its representation and derive something more meaningful and easier to analyze. Image segmentation is often used to recognize and locate certain categories of objects by assigning a label to each pixel.

In autonomous driving, image segmentation is often performed to understand the surrounding environment of the ego vehicle, for example, to recognize surrounding vehicles, pedestrians, road boundaries, buildings, etc. This information is crucial for determining the next actions. However, the result of the segmentation process is often used ambiguously and hard to apply directly to drive the car.

Here, we incorporate image segmentation directly into the task of learning to control a car in an end-to-end fashion. We believe the learned segmentation map contains auxiliary information for controlling the car's behavior. So we add a segmentation map as an extra input to the system.

The auxiliary information will explicitly tell the network, for example, where the road boundary is and where the opponent vehicles on the road are. This will decrease the difficulty of learning everything implicitly from the original raw image.

We integrate the image segmentation into our architecture by using the segmentation network proposed by [36].

3.2.2 Transferring from Existing Tasks

CNN has been applied recently to a significant number of practical and essential tasks. The training results have shown that this approach is very successful in jobs like object recognition. This motivates us to leverage the power of a pre-trained network and apply the concept of transfer learning [37]. Currently, numerous famous network structures in the literature have been proven to be powerful. The most popular task is to learn object recognition on the Imagenet dataset that contains 1.2 million images of approximately 1000 different classes. The resulting trained model can generalize a generic set of features, and recognize a large variety of objects with high accuracy. The intermediate features learned are found to have universal expressiveness across multiple domains. We hence want to utilize this point and transfer the pre-trained network from a vast field to the specific task of learning the driving policy.

In this thesis, we compare three models: Resnet, the Vgg16 network and our baseline CNN for the CNN module in our overall network. The Resnet and Vgg16 network are pre-trained on Imagenet. For feature extraction purposes, we only use the convolutional layers. The detailed configuration of our CNN network is illustrated in Section 3.3. The three networks differ in depth and number of total parameters. To make the comparison fair and even, we froze some of the parameters in Resnet and Vgg16 net, so that only parts of the Resnet and Vgg16 net are tunable. This makes the total number of adjustable parameters approximately the same for each model tested.

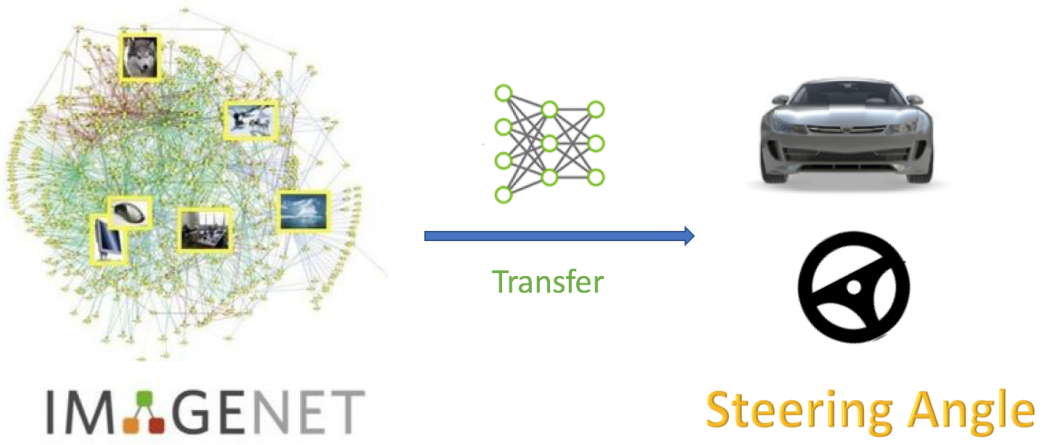

Figure 3.2: Illustration of transfer learning from object recognition in Imagenet to learning steering angle for the self-driving car.

3.2.3 Temporal Information

Decision making for an autonomous vehicle is not an independent choice at every time step. A human would consider past environment information and previous actions taken and then make a consistent decision of driving behavior. This requires our system not only to be based on the current state but also incorporate past states. So, apart from the convolutional neural network to capture the spatial information, we introduce recurrent modules into our network architecture.

After getting the spatial features from the CNN layer, we added an LSTM layer to pass in previous information of the environment. The LSTM processes a feature vector v from the CNN in a sliding window of size w. This means the steering action prediction result is dependent on w past input observations $X_{t-w+1} - X_t$. By changing the parameter of w, we can alter how long the system considers to make the decision. Small w leads to shorter-term memory, so it has faster reaction time but is prone to sudden sensor failure. Larger w, on the contrary, will lead to a much smoother and stable behavior. The problem of larger w is that it requires longer training and test time for choosing actions.

With the visual states at each time step, the LSTM fuses all past states and current state into a single state. So the state here is complete, and the autonomous car is theoretically given all the historical information to make the necessary action choice.

3.2.4 Additional Vehicle Information

We further hypothesis that visual input alone is not good enough to make a good steering angle decision. The vehicle's behavior is better estimated by adding the vehicle's kinematic information. The kinematic information ensures that the car does not follow some driving behavior that is against any physical rules.

It can be speculated that making a U-turn at 10mph and 30 mph is different regarding turning angle and the strategy used. However, the visual observations given are almost the same. Although we can infer the speed of the vehicle by the scene change speed, it remains ambiguous and is not easy to learn from images. That's the reason why we need vehicle kinematic information like vehicle speed.

Limited to the simulation environment and real-world dataset, we select the following kinematic parameters as an extra input to the fully connected layer:

- vehicle acceleration rate
- vehicle speed
- vehicle heading
- vehicle lateral distance to road boundary
- vehicle previous steering angle
- vehicle steering torque

3.3 Experiment

In this section, we discuss our experimental environment selection both in simulation and real dataset. We show how we do data preparation and explain the details of our experimental design and implementation.

Environment Selection

Lately more and more public datasets and simulation platforms for on-road driving have become available. These datasets contain diverse driving scenarios including cities, highways, towns and rural areas in the US and across the world. Here, we give a comprehensive overview and comparison of available datasets.

Datasets collected in the real world:

- The **Kitti dataset** contains sensor information captured with a car driving around the city of Karlsruhe, Germany in rural areas and on highways. Image data were collected using two high-resolution color and grayscale video cameras. The ground truth is provided by a Velodyne laser scanner and a GPS localization system. The Kitti dataset is suitable for investigating the task of stereo, optical flow, visual odometry, 3D object detection and 3D tracking.
- The **CityScape dataset** is a diverse set of stereo video sequences recorded in street scenes from 50 different cities across the world. It has high-quality pixel-level annotations of 5000 frames in addition to a larger set of 20,000 weakly annotated frames. It can be best used for semantic urban scene understanding.
- The **Comma.ai dataset** contains 7.5 hours of highway driving. The sensor input is recorded at 20 Hz with a camera mounted on the windshield of an Acura ILX 2016. Together with the images, the dataset contains information such as the car's speed, acceleration, steering angle, GPS coordinates and gyroscope angles.
- The **Oxford Robocar dataset** contains 100 repetitions of consistent routes through Oxford, UK, captured over a period of over a year. The dataset captures a combination of weather, illumination, dynamic objects, traffic, and pedestrians information, along with seasonal changes, construction, and roadworks.

Simulation environments:

- The **Torcs simulator** is an open-source racing car simulator written in C++. It is a popular video game as well as a common research platform for studying AI agent. In Torcs, many trials with various environment settings and car models with different behaviors are available to users' usage.
- The **Udacity simulator** is an open-source simulator developed based on Unity. The simulation gives a realistic 3D visualization of the vehicle driving on three given tracks in the desert, mountain and forest.

A comparison of the different datasets regarding settings, type, time length, scenario and weather diversity is given in Table 3.1.

For the experiment parts, we used simulation environment Udacity simulator and the Comma.ai

dataset as evaluation platforms for our algorithms. The selected datasets have access to driver's control actions together with sensor perception information, which could fit the need of our task.

datasets	settings	type	time length	weather diversity	day/night driving
Kitti	city, rural, highway	real world	less than 2 hours	clear weather	day time
CityScape	city	real world	less than 100 hours	multiple weather condition	day time
Comma.ai	highway	real world	7.5 hours	clear weather	day time and night time
Oxford Robocar	city	real world	214 hours	multiple weather condition	day time and night time
Torcs	highway	simulation	-	clear weather	day time
Udacity	rural	simulation	-	clear weather	day time

Table 3.1: Camparison between different benchmark datasets on autonomous driving.

Data Preparation

In the Udacity simulation environment, we use three tracks. The three tracks respectively depict a highway in desert, suburb, and mountain. Example screenshots of the different trials are shown in Figure 3.3. The desert track is used for training purposes, and the suburb and mountain tracks are used for testing.

We collected image data for training by driving the car in the simulation environment. To introduce various driving styles from multiple people, we collected data from six people each driving the desert track twice. We recorded the steering angle, speed, acceleration rate and braking rate paired with the corresponding images while driving with keyboard input. The system operates at a 10-hertz frequency. Altogether we collected 6245 images which are about 1 hour of driving data. We sampled images at 2 Hz to prevent redundant pictures. The images captured are simulated the front view from the vehicle via a camera mounted on top of the car.

Figure 3.3: **Sample screenshots** of the environment in the Udacity autonomous driving simulator. The left one shows the training track in the desert, while the two on the right show the test track in suburb and mountain. The test sets are different from the training set regarding **lighting conditions, curvature, inclination, road geometry and off-road scenery** and thus are considered much more difficult.

The images obtained are not directly used for training purposes. Before training, we preprocess and augment the data similar to the techniques described in [7]. Data augmentation is used here to increase the size of the training set and also the diversity of training samples. The following operations were performed.

- **Cropping**: The images are cropped to remove extraneous elements. We removed the top of an image which includes a large area of sky or mountain tops and the bottom of the image which contains the car hood.
- **Upsampling**: In the original training set, most scenarios are going straight along the road. Images with a steering angle larger than 15 degrees are scarce compared to a significant number of training samples with a steering angle less than 5 degrees, which means the steering angle distribution is strongly biased towards zero. To overcome the problem of imbalanced data, we manually upsample images with steering angle greater than 10 degrees by ten times and steering angle greater than 5 degrees by five times. The data are then randomly shuffled before training.
- **Brightness changes**: Each frame is first converted to HSV space and the value channel is then multiplied by a random value from 0 to 10. This changes the lighting situation and makes the model adaptive to all kinds of weather.
- **Flipping**: We flip all frames to obtain their mirror in the horizontal direction. This helps to make sure we have exactly the same amount of left and right turning samples. The algorithm won't suffer from any bias in the left or right direction.

For the real data from the Comma.ai driving dataset, there is no need for cropping and brightness preprocessing. We use the same upsampling and flipping techniques to deal with the data balance problem. The dataset includes 11 video clips of 7.5 hours of highway driving at 20 Hz. Here we only want to consider stable highway driving at normal speed in daylight. We further exclude the driving videos at night or in traffic jams with speed under 10mph. The finally selected footage has a length of about 2 hours of driving. We split it by using 130K frames for training and 20K frames for testing.

Implementation Detail

As a baseline to compare our algorithms against we use a variation of the CNN network structure proposed in [7]. The difference here is that we add batch normalization and dropout layers after each convolutional layer for better convergence and performance.

The CNN network consists of 5 convolutional layers. The first three layers have a kernel size of 5×5, and the last two layers have a kernel size of 3×3. The depth of each feature map is 24, 36, 48, 64, 64. The activation function we use here is ReLu.

Our model has an additional LSTM layer apart from convolutional layers, as shown in Figure 3.1. The LSTM has 128 hidden units. The output of LSTM and the vehicle kinematic dynamics are concatenated before fed into the FCN module. The FCN module consists of 3 fully connected layers with 256, 128 and 64 units followed by a $Tanh$ activation function.

We use Adam optimization to train all networks. The learning rate is fixed to 0.001 with a momentum decay of 0.9. The batch normalization and dropout layers are used to prevent overfitting.

3.4 Evaluation and Results

We report our experimental results on the Udacity simulation and the Comma.ai dataset. For the steering angle prediction task, we use Root Mean Square Error (RMSE) as the evaluation metric. RMSE can express average system prediction error on the dataset. The RMSE metric is defined as

$$RMSE = \sqrt{\frac{1}{|D|}\sum_{i=1}^{|D|}(\hat{a}_i - a_i)^2},$$

where $\hat{a}_i$ and a_i are the ground-truth and predicted steering angle for frame i and $|D|$ is the total number of frames in the test set. The angles are measured in angular degrees.

The RMSE can estimate the precision of the system but can neglect the stability of the system. So we also define a stability metric based on the deviation of our prediction. The intuition behind it is that we want our predictions to change smoothly without any sudden bump in the steering angle. We call this metric Mean Continuity Error (MCE).

$$MCE = \sqrt{\frac{1}{|D|-1}\sum_{i=1}^{|D|-1}(a_{i+1} - a_i)^2}$$

We evaluate the influence of the different methods suggested with the baseline method (see Table 3.2). We first compare the baseline CNN structure with two popular networks Vgg [38] and Resnet [39] for the image recognition tasks. To maintain roughly the same number of weights for training in different models, we only train the last five layers of Vggnet and Resnet. Here transfer learning shows that the pre-knowledge from the image recognition task is beneficial for the job of predicting steering angles. It should be noted that the performance boost in the Comma.ai dataset is much more substantial that Udacity simulation. This is most likely due to the fact that the Comma.ai dataset contains real imagery which has a higher resemblance with Imagenet dataset than the simulation environment.

Proposed Network Structure	Udacity Simulation		Comma.ai Dataset	
	RMSE/degrees	MCE/degrees	RMSE/degrees	MCE/degrees
baseline CNN	7.68	2.32	19.84	7.26
Vgg CNN	7.45	2.12	15.86	5.73
Resnet CNN	7.34	2.09	15.23	5.35
Resnet CNN + SegMap	5.23	1.57	12.33	4.21
Resnet CNN + SegMap + LSTM	4.50	1.33	10.72	3.78
Resnet CNN + SegMap + LSTM + vehicle kinematics	**4.23**	**1.32**	**10.23**	**3.66**

Table 3.2: Comparison between different network structures for vision-based end-to-end learning of steering angle. Our proposed method has the lowest RMSE and MCE both in the Udacity simulation and on the Comma.ai dataset compared to the baseline method.

To evaluate the effect of the segmentation map augmentation, we compare the result of adding the segmentation map as an extra feature map for input to the convolutional layers. We use the segmentation map output categories of the sky, road marking, road, tree, pavement, and

vehicle. We construct a binary map for each group and stack them together with the original three channels of the image. As can be seen in the fourth row of Table 3.2, the precision and stability boost is massive compared to the raw image input. Example segmentation map outputs are shown in Figure 3.4.

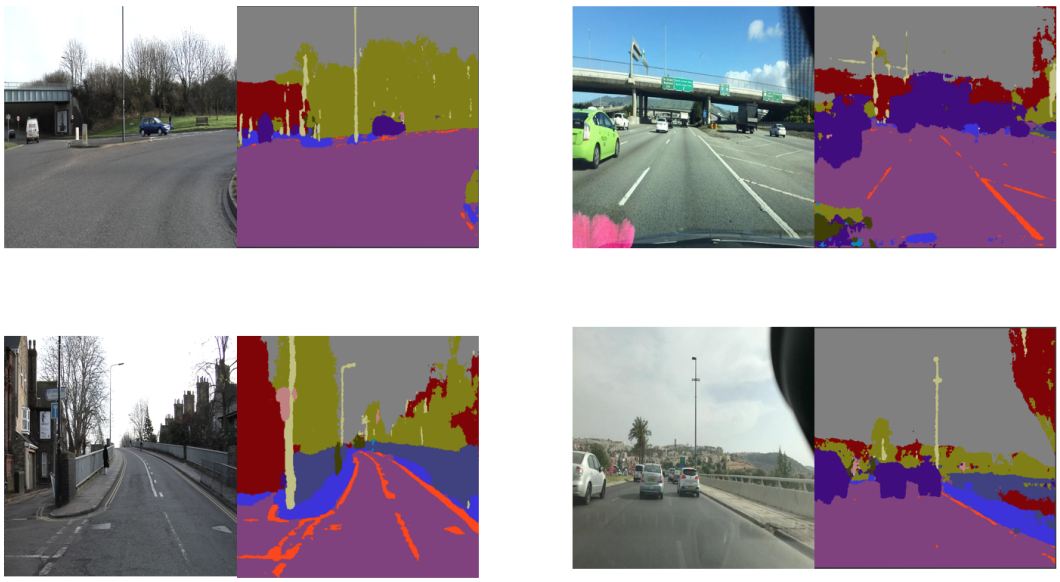

Figure 3.4: Example intermediate segmentation outputs obtained in the end-to-end learning procedure.

Next, we evaluated the effect of adding temporal information by using the layer of LSTM. The results are shown in the firth and sixth rows of Table 3.2. The performance is increased both in the Comma.ai dataset and in the Udacity simulation. We conducted grid-search for the optimal window size w for the LSTM and found that $w = 3$ generates the best result, which means we look back for 1.5 seconds. We compare the main difference of the prediction with the baseline model and discover that after using temporal information, absurd outlier predictions are significantly reduced. The cases where two sequential frames make dramatically different steering angle predictions almost disappear. This dramatically improves the stability of the algorithm and also improves the prediction accuracy.

Adding the vehicle kinematics also slightly improves the performance, by about 3%. The ablation test shows the vehicle previous steering angle is the most useful, followed by vehicle speed and vehicle heading. We observe no performance boost in using the vehicle acceleration rate, vehicle lateral distance to road boundary and vehicle steering angle.

Our overall proposed architecture has reduced prediction RMSE error by 44.92% in the Udacity simulation and 48.44% in the Comma.ai dataset. The prediction MCE error was reduced by 41.81% in the Udacity simulation and 49.58% in the Comma.ai dataset.

We also do an empirical test on the Udacity simulation environment to see if the car can successfully drive in a new scenario. The result shows good driving performance on an unseen map with various sceneries in suburb and mountain. A video clip of our trained model is available at `https://youtu.be/reqAHtXtnrI`.

3.5 Conclusion

We suggested a network architecture that improves the baseline vision-based end-to-end learning of steering angle. The suggested network architecture uses four distinct methods: adding auxiliary segmentation, transferring from existing tasks, utilizing temporal information and incorporating vehicle information. We found that using transfer learning from the Imagenet recognition task can be helpful in learning the task of steering for on-road driving. Using the pre-trained segmentation mask to categorize the image at the pixel level can empower the network with more information and thus result in better prediction accuracy. The incorporation of temporal information of history states indeed helps to make better current decisions, which again proves the concept that driving policy is a long-term decision-making process. Finally, the proper addition of some vehicle kinematics makes the state representation more concrete.

Chapter 4

Learning Lane Change Behavior with Deep Reinforcement Learning

In the previous chapter, we investigated the use of an end-to-end neural network to mimic human behaviors in both simulation and real-world datasets directly. However, this approach is limited in the following respects. First, it requires a significant amount of human driving data, which is not available in many cases. The system cannot adapt to entirely different and unseen environments. Second, it is limited to the driving behavior of human drivers. If the human driver shows terrible driving habits, it will reproduce the good as well as the bad behavior as it is not able to distinguish between them. Finally and most importantly, the driving data are all from successful driving scenarios, so when the error accumulates in the system, and the car is facing danger, the system cannot formulate by itself what driving policy to use under this circumstance. To overcome these problems, we apply deep reinforcement learning.

4.1 Deep Reinforcement Learning for Self-driving

Deep reinforcement learning allows the autonomous agent to surpass the limitations of human supervision by learning from trial and error. Previous work mostly focused on applying DQN or DDPG to learn to drive the car [39, 40, 41]. They were able to successfully demonstrate that a car can learn to drive itself in the simulation without leaving the road boundary. However, most of these works focus on the simple task of following the lane but fail to consider the interaction with the surrounding vehicles and more complex behaviors such as overtaking or lane changing.

In this work, we propose to use a deep reinforcement learning-based method that can learn sub-policies for lane changing behavior. Lane change is a fundamental behavior for on-road driving that is commonly executed for overtaking or navigation purposes. It requires high-level reasoning about other surrounding vehicles' intentions and behavior as well as forming an effective strategy for our driving purpose under the safety requirement. At the same time, it requires low-level reasoning to plan what exact path to follow, generally known as the path planning problem. Each of these parts has been researched separately in previous literature. However, the internal connections between them makes it hard to treat them separately for the problem at hand. Many integrated systems suffer from the problem of continuous unnecessary replanning

because of the separation of these two modules [42]. In our work, we combine these two levels of reasoning into a hierarchical structure, while maintaining it in one system that can be trained together. Our network structure is capable of learning when and how to do lane change using a unified network.

At the same time, we also explore the concept of attention used in autonomous driving. We notice that among human drivers, people do not usually pay equal attention to the information given. People select the information that is helpful for determining the current maneuver and ignore the unrelated information. Here we pick up and incorporate the concept of attention from recent advances in deep neural networks [43]. We show that during the training of deep reinforcement learning, our attention mechanism will automatically focus on the end of the road or related on-road vehicles that may influence our driving behavior. The resulting network can better use the information and thus results in shorter convergence time and better performance.

A structural overview of our method is shown in Figure 4.1.

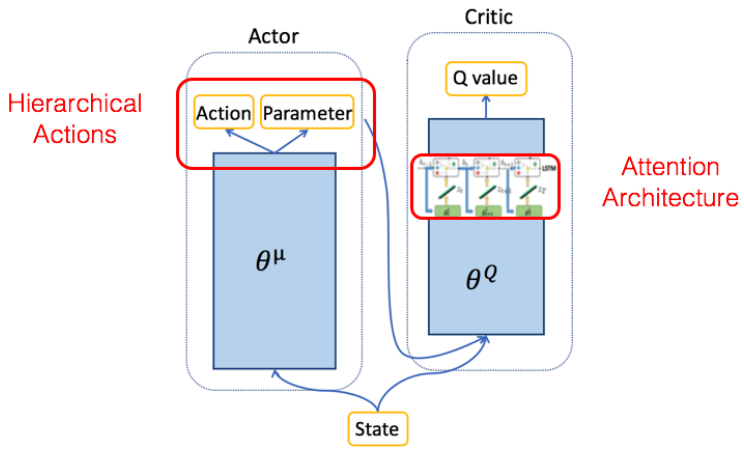

Figure 4.1: Illustration of the structure of our algorithm. Our algorithm is based on deep reinforcement learning with actor and critic. We propose hierarchical actions and attntion mechanism to tackle lane change behavior.

4.2 Methodology

In this section, we discuss the details of our method and introduce hierachical action and attention mechanism approaches for deep reinforcement learning in learning lane change behaviors of autonomous driving.

4.2.1 Hierarchical Action Space for Lane Change Behavior

Referring to the parameterized action space introduced by Hausknecht [44], we create a hierarchical action space for autonomous driving as follows. There are three mutually exclusive discrete high-level actions: Left Lane Change, Lane Following, Right Lane Change. At each time step, the agent must choose one of the three high-level actions to execute. Each action requires

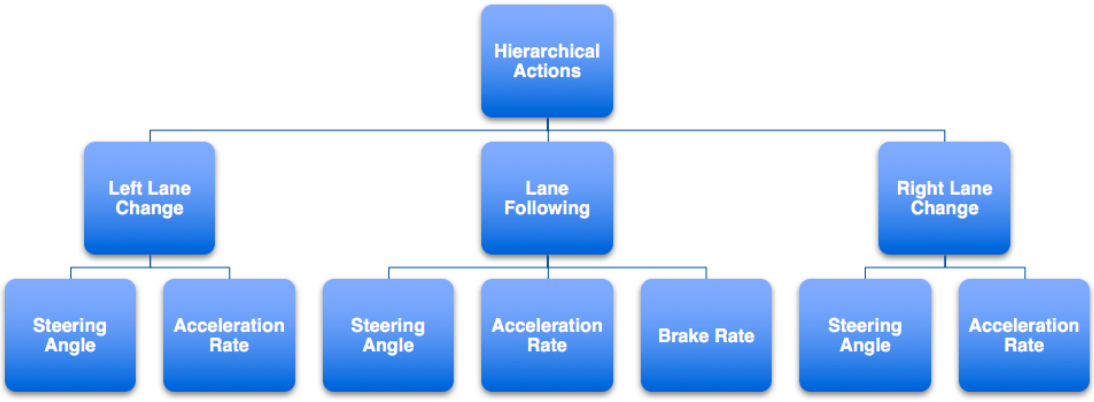

Figure 4.2: Illustration of the hierarchical action space for lane change behavior.

2-3 continuous-valued parameters that must be specified. The choice of hiearchical actions is shown in Figure 4.2 and details of each action are listed as follows:

- **Left Lane Change (steering angle, acceleration)**: do left lane change in the direction of the steering angle with acceleration applied for the control module.
- **Lane Following (steering angle, acceleration, brake)**: go straight in the direction of the steering angle with acceleration and brake applied for the control module.
- **Left Lane Change (steering angle, acceleration)**: do right lane change in the direction of the steering angle with acceleration applied for the control module.

The steering angle is continuous limited to the range of $[-60, 60]$ degrees. Large steering angles are prevented for safety reasons. Acceleration and brake are double variables in the range of $[0, 100]$. Formally, the high-level discrete actions are defined as $A_d = \{a^{straight}, a^{left}, a^{right}\}$. Each discrete action $a \in A_d$ contains a set of continuous parameters $P_a = \{p_1^a, ..., p_n^a\}$. The overall hierarchical action space we utilize here can be defined as

$$A = (straight, p_{angle}^{straight}, p_{accelerate}^{straight}, p_{brake}^{straight}) \cup (left, p_{angle}^{left}, p_{accelerate}^{left}) \cup (right, p_{angle}^{right}, p_{accelerate}^{right})$$

Here, we assume that when people perform lane changes, they do not brake to stop. So the lane change behavior does not have the parameter of braking. The three actions represent three types of driving policy respectively. The system should be able to learn how each policy would behave and when to apply which policy after training.

4.2.2 Actor-critic Based DRL Architecture

We develop our algorithm based on the popular deep reinforcement learning algorithm DDPG (Deep Deterministic Policy Gradient) [8]. To incorporate hierarchical actions, we use here the actor-critic architecture suggested by [8]. This architecture decouples the action evaluation and the action selection process into two separate deep neural networks: actor-network and critic-network, as shown in Figure 4.3. The actor-network μ, parameterized by θ^μ, takes as input state s and outputs action a along with its parameter p_a. The critic-network Q, parameterized by

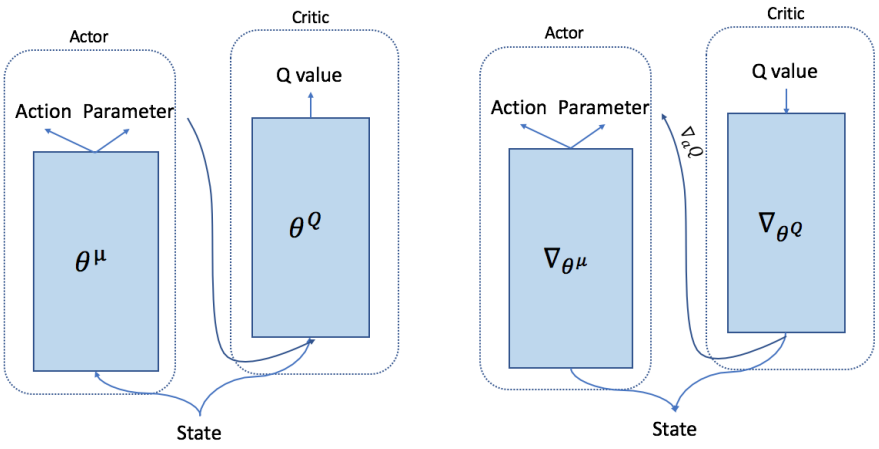

Figure 4.3: The Actor-Critic architecture used in our algorithm, first introduced by [8]. On the left is the data flow in the forward pass and on the right is the gradient flow in the back-propagation.

θ^Q, takes as input state s and action a along with its parameter p_a and outputs a scalar Q-Value $Q(s, a)$.

The action a is represented as the probability of action choice and the parameters p_a are coupled to each discrete action. The discrete action is chosen to be the output with maximum value among the choices of actions. Then it is coupled with the corresponding parameters from the parameter outputs. Though the parameters of all actions are output, only the parameters of the chosen action are used. In this way, the actor-network simultaneously outputs which discrete action to execute and how to choose parameters for that action. The critic-network receives as input all the values in the output layer in the actor. We do not indicate which exact action is applied for execution or which parameters are associated with which action. In the back-propagation stage, the critic-network only provides gradients for the selected action and the corresponding parameters. This assures we update the policy only in the direction where we explore.

For stability reasons, we use the standard target network for the critic-network and the actor-network, which updates at a slower rate for a more stable iterative step in the Bellman equation. We also use replay memory to store experiences, which can significantly break the dependency between experiences. The exploration strategy now has to deal with both discrete action choice and continuous parameters. We use $\epsilon - greedy$ exploration to randomly explore among the given set of discrete actions. Then we sample uniformly for the continuous parameters. That means we choose a random discrete action $a \in A_d$ with probability ϵ and associate it with continuous parameters $\{p_1^a, p_2^a, ...\}$, each sampled uniformly over the range of its possible values.

4.2.3 Attention Mechanism for Deep Reinforcement Learning

The attention model we developed for the deep reinforcement learning algorithm originates from the DRQN algorithm [44]. In this section, we first briefly introduce how to add recurrence to the DDPG algorithm, which we call Deep Recurrent Deterministic Policy Gradient (DRDPG). Next,

we illustrate our novel architecture of bringing attention to DRDPG. This includes two kinds of models: one considers spatial attention and another considers temporal attention. Note that the architecture we proposed applies to other sequential prediction problems using a neural network, not just to learning for driving behaviors of self-driving cars. It can be used as an auxiliary task to enhance the performance of the original algorithm since the new information needed is limited in our method.

Deep Recurrent Deterministic Policy Gradient (DRDPG)

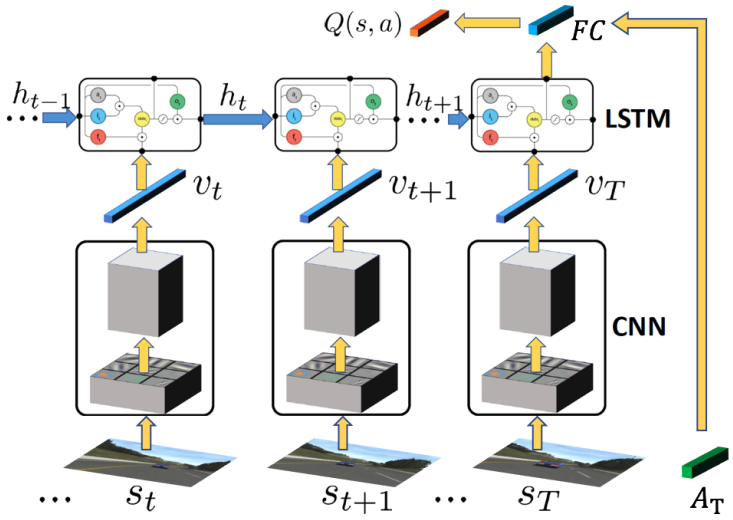

Figure 4.4: **The architecture of the critic network in DRDPG.** DRDPG convolves over an image of the input screen. The resulting feature maps are processed through time by an LSTM layer. The final Q value is obtained by concatenating the action vector with a fully connected layer. During training, the LSTM is trained with an unrolling of T=8 frames.

The concept of deep recurrent Q-network (DRQN) was recently introduced by Hausknecht et al. [44] as a combination of the recurrent neural network (RNN) and the deep Q-network. The main contribution is to use an RNN module to process the temporal inputs of observations instead of just stacking historical observations as input. With a recurrent module in the original DQN architecture, a longer sequence of history information can be incorporated, which helps with more complex strategies. We refer to the structure of DRQN and introduce the recurrent module using LSTM in the critic network of the DDPG algorithm, as shown in Figure 4.4. We call this new methodology Deep Recurrent Deterministic Policy Gradient (DRDPG).

As compared with traditional DDPG, DRDPG offers several advantages, including the ability to handle longer input sequences and exploration of temporal dependencies, as well as better performance in case of partially observable experiences.

Attention Mechanism

We are also interested in exploring the effectiveness of an attention mechanism for DRDPG. Attention models [45] have recently been applied in the areas of image caption generation [43] and object tracking [46]. They show effectiveness in compressing the input information to enable training speedups, and also offering an interpretable visualization about "where" and "what" the agent chooses to focus on.

In this work, attention models are explored in two streams: a temporal attention model and a spatial one. We combined both streams in DRDPG with some adaptation. The spatial attention will detect the most important and relevant regions in the image for driving; temporal attention will weigh the last few frames to decide the current driving policy.

DRDPG with Temporal Attention

Temporal attention has recently been shown to boost performance in sequence-to-sequence learning for classification tasks [47]. However, there is no current literature showing whether it can learn more accurate q-values in the context of reinforcement learning.

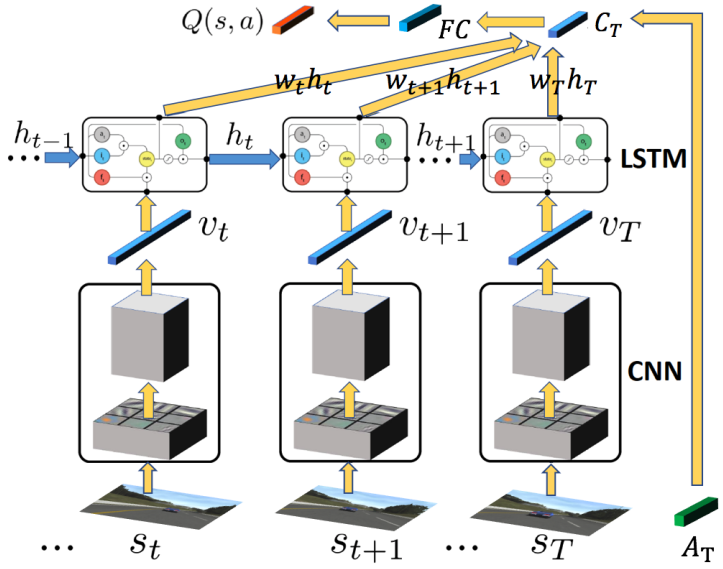

Figure 4.5: **Architecture of the critic network of the Temporal Attention DRDPG.** An additional context vector C_T for computing Q values is derived as a linear combination of T LSTM outputs concatenated with weighted action vector A_T, with comparison of the last frame LSTM output used in DRDPG. The weights can be optimized through backpropagation during training.

Inspired by [48], we apply temporal attention over the output of the LSTM layer in DRDPG model, as shown in Figure 4.5. The temporal attention mechanism learns scalar weights for LSTM outputs in different time steps. The weight of each LSTM output w_i is defined as an inner product of the feature vector v_i and LSTM hidden vector h_i, followed by a softmax function to normalize the sum of weights to 1. By this definition, each learned weight is dependent on the previous timestep's information and current state information.

$$w_{T+1-i} = Softmax(v_{T+1-i} \cdot h_{T+1-i}) \quad i = 1, ..., T \tag{4.1}$$

Then we compute the combined context vector C_T. The context vector C_T is a weighted sum of LSTM outputs through T timesteps, concatenated with weighted action vector output A_T from the actor-network.

$$C_T = \sum_{i=1}^{T} (w_{T+1-i} h_{T+1-i}) + A_t w_A \tag{4.2}$$

The derived context vector C_T is passed by a fully connected layer FC before obtaining the final Q value of the critic network. The learned weights $\{w_{T+1-i}\}_{i=1}^{T}$ here can be interpreted as the importance of the LSTM output at a given frame. Therefore, the optimizing process can be seen as learning to choose which observations are relatively more important for learning the Q values.

Temporal attention DRDPG is superior to DRDPG in the sense that it explicitly considers the past T frames LSTM output features for computing the Q value, while this information is only passed implicitly through LSTM in the original DRDPG. By increasing the value of T, the model can consider a longer sequence of history frames and thus can make a better action choice.

DRDPG with Spatial Attention

Spatial attention models [43] learn weights for different areas in an image, and the context feature used for decision making is a combination of spatial features according to the learned weights. According to how the "combination" is modeled, spatial attention models are divided into "hard" attention and "soft" attention [43]. Similar to the ideas presented in [49], we use a "soft" version attention for DRDPG, which means learning a deterministic weighted context in the system.

The Spatial attention DRDPG architecture, as shown in Figure 4.6, contains three types of network: convolution, attention and recurrent. At time step t, suppose the convolutional layers produce a set of d feature maps with size $m \times n$. These feature maps can also be seen as a set of region vectors with length d : $\{v_t^i\}_{i=1}^{L}, v_t^i \in \Re^D, L = m \times n$. Each region vector corresponds to the features extracted by the CNN at a different image region. In soft attention mechanism, we assume the context vector z_t is represented by a weighted sum of all-region vectors $\{v_t^i\}_{i=1}^{L}$.

$$z_t = \sum_{i=1}^{L} g_t^i \cdot v_t^i \tag{4.3}$$

The weights in this sum are chosen in proportion to the importance of this vector (aka the extracted feature in this image region), which is learned by the attention network g. The attention network g_t^i has region vector v_t^i and hidden state h_{t-1} produced by LSTM layer as input and outputs the corresponding importance weight for the region vector v_t^i. The attention network g_t^i here is represented as a fully connected layer followed by a softmax function.

$$g_t^i = Softmax(w_v \cdot v_t^i + w_h \cdot h_{t-1}) \quad i = 1, ..., T \tag{4.4}$$

The context vector z_t is fed into the LSTM layer. The output of the LSTM layer is concatenated with action vector A_T and then used to compute the Q values.

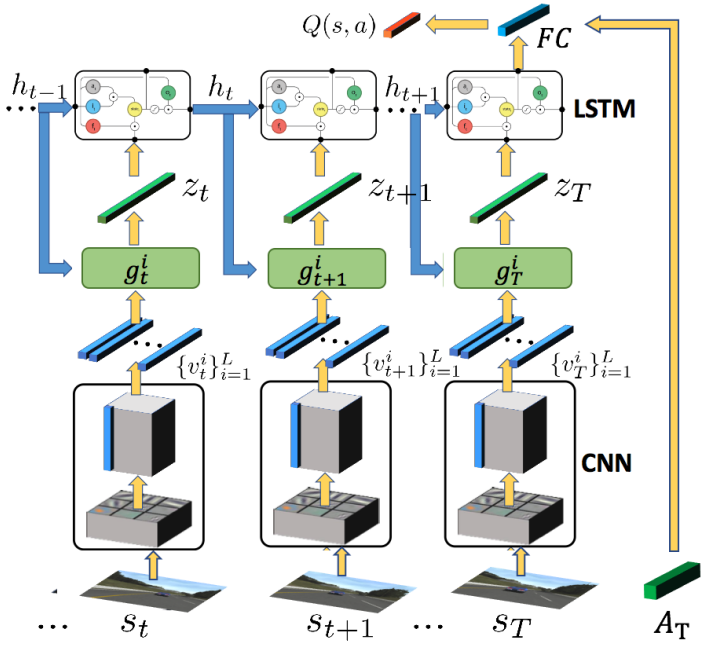

Figure 4.6: **Architecture of critic network of Spatial Attention DRDPG.** Feature maps extracted by CNN are interpreted as region vectors. An attention network will learn the importance weight of each region vector and derive a weighted sum of region vectors before feeding it to LSTM layer.

The attention network can be interpreted as a mask over the CNN feature maps, where it reweights the region features to get the most informative features for computing the Q-values. Thus the Spatial attention DRDPG acquires the ability to select and focus on the essential regions when making the action choice. This also helps to reduce the total number of parameters in the network for more efficient training and testing.

4.2.4 Reward Signal Design

In reinforcement learning algorithms, an important part is to design a good task-specific reward to guide the learning. A simple and straightforward reward in self-driving can be the distance that the car can go before crashing. However, such a reward signal is too uninformative for the learning agent to collect any information. So we define reward functions to encourage the car to stay in the lane and also to perform a lane change when the situation allows. We introduce a hand-crafted reward signal that assures both driving comfortability and efficiency. The final reward contains five components as follows, illustrated in Figure 4.7.

- We want the speed of the vehicle to align with the direction of the road. We reward the speed in the road direction and punish the speed deviating from the road direction. Here $\theta > 0$ represents the angle of deviation from the road direction.

$$r_1 = \cos\theta - sin\theta$$

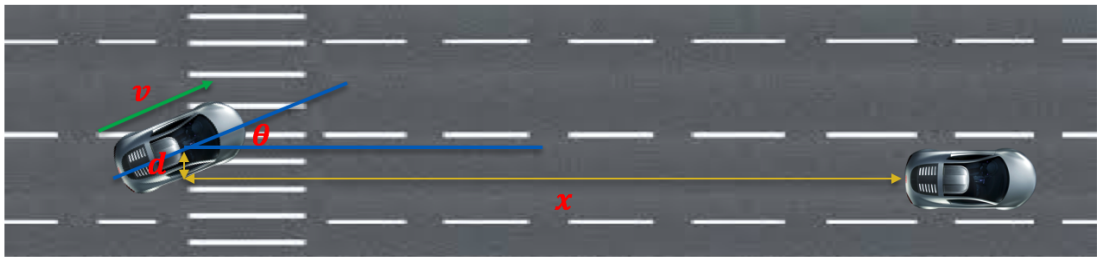

Figure 4.7: Illustration of the notation used in reward functions.

- We wish the car to remain in the center of the lane while driving, so we punish any lateral distance from the center of the lane. d represents the current distance to the center of the lane.

$$r_2 = -|d|$$

- We give a large negative penalty when the car goes outside the road boundary. The game also ends and restarts if a car enters this state.

$$r_3 = -\mathbb{1}\{OutofBoundary\}$$

- We encourage the car to have larger speed, but not to exceed 35m/s.

$$r_4 = \begin{cases} v & v \leq 35m/s \\ 70 - v & v > 35m/s \end{cases}$$

- For lane change purposes, we encourage the car to overtake if the front vehicle is within a distance of 100 meters. Here x means the distance to the front vehicle in the same lane. If no vehicle is found, then x has a default value of 100.

$$r_5 = -max(0, 100 - x)$$

The overall reward function is a linear combination of terms above with assigned weights w: $R = \sum_{i=1}^{5} w_i r_i$. Here we first normalize the rewards to the range $(0, 1)$ and then search for a weight vector w that generates a good result. A more powerful model would test on the different weighting coefficients to find the best combination.

A possible future direction for improvement is to use inverse reinforcement learning to extract the parameters for the reward function automatically.

4.3 Evaluation and Analysis

In this section, we will show how we setup the experimental environment and implement the algorithms. The quantitive evaluation of different methodologies applied will also be given and compared. Finally, we give illustrative visualizations to help understand the insights of attention mechanism.

4.3.1 Experiment Setup

We test our algorithm in the open source racing car simulation environment TORCS. To generate a universal model that is not limited to a particular road configuration or car type, we select five tracks in Torcs and ten types of cars to generate our training set. Each track is customized to have two to three lanes with various brightness levels of the sky. The selected tracks have a different surrounding environment and route shape so that our training agent can confront all circumstances of driving, which ensures the difficulty of the driving scenarios. For example, in the Corkscrew scenario, the road has many curves and few straight sections, and there is a tricky high-speed corner. To successfully traverse the selected tracks, the car has to learn various skills like U-turn, hill climbing, overtaking and throttling before a large-angle turn. All these challenges are posed to the training agent with no human supervision.

We add 20 traffic cars in different locations on every track in each trial of the training. The traffic cars' behavior is controlled by the internal AI of the TORCS environment. We add diversity to the traffic cars' behavior by changing the internal parameters in the car AI simulation. This will change the car's driving style, e.g., being more aggressive when turning. In this way, we would like to mimic the real traffic patterns on the road. We set the speed limit of traffic cars to 30m/s and set the speed limit of our ego vehicle to 35m/s. We do this to encourage the overtaking and lane changing behavior of our vehicle.

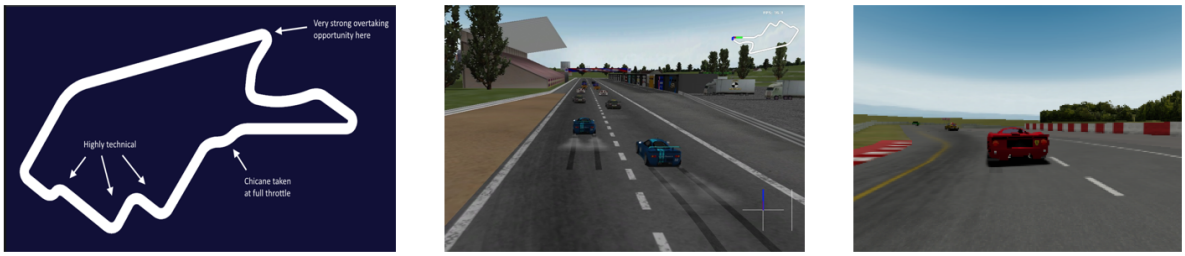

Figure 4.8: Track *Street-1* used for training. From left to right: the map of the *Street-1* course, image top view when starting a new episode, a screenshot of the front view camera during training.

For the sensor input, our method is based on image input. The images are down-sampled to 320×240 for faster training. For stable and safe lane changing behavior, the front view camera alone is not enough. We also tried to add some additional LIDAR-based information and the ego car's physical information apart from pure image input to the network to facilitate the training. This is applied as a comparison with the architecture having image input alone which we described before. A vector of 30 range finder sensor spanning a 360° field-of-view is appended to the input of the LSTM layer. Each sensor returns the distance between the road edge and the ego car with a maximum of 150 meters. This sensor input explicitly includes distance to other traffic particpants on the road, which can help the system to learn to avoid collision.

The training starts by automatically selecting one of the five tracks. Every time the car goes outside of the road boundary, the episode will end, and the program will restart to generate a new episode. The CNN network we use is the standard AlexNet architecture. There are five

convolutional layers followed by an LSTM layer with 256 hidden units. The final hidden layer is a fully connected layer with output dimension of 128. For training comparison, we restrict each training model to 200,000 iterations before stopping the process, which takes about $18-30$ hours on an Nvidia Titan X GPU.

4.3.2 Results

We first evaluate our final model on the five maps in TORCS, as shown in Figure 4.9. We test the model by letting the agent drive through the map and see if it can successfully finish a loop. The final model with hierarchical actions, spatial and temporal attention, and range finder sensor shows overall excellent performance in all five trials with a success rate between 60 and 80 percent to complete a runway circle on the map. The failure cases mainly come from collision or out-of-bounds cases, about half of each. For example, some sharp turns need rapid deceleratino for safe traversal. That is the most common circumstance where the car goes out of bounds. For the collision issue, the reasons are mixed. Some collisions happen when trying to pass a vehicle while turning. Some happen when the front car slows down and our ego vehicle tries to pass it.

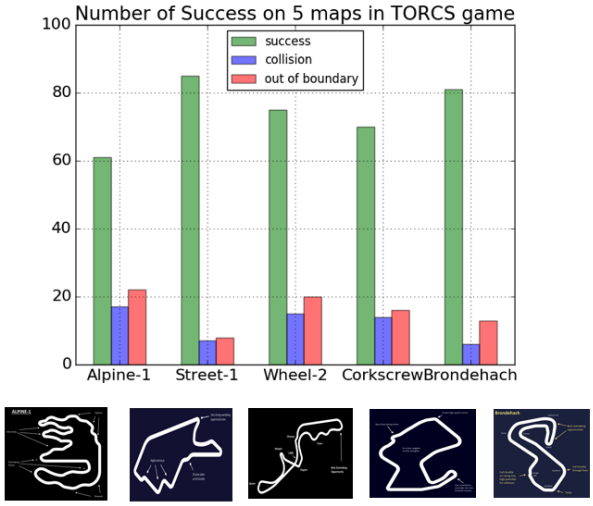

Figure 4.9: Final DRDPG model with hierarchical actions, Spatial and Temporal attention tested on different trials in TORCS game. We obtain the result of each map by running 100 episodes.

We compare our model with the original baseline models. Typically we compare five models: Deep Deterministic Policy Gradient (DDPG), DDPG with Hierarchical Actions (H-DDPG), H-DDPG with Recurrent Module (H-DRDPG), H-DRDPG with Spatial Attention (SpatialAtt) and H-DRDPG with Temporal Attention (TemporalAtt). We train each model to 200,000 iterations before a stop. We test each model with 20 games on each trial and calculate the average speed, the number of lane changes, the total reward received during each episode and the percentage of successful episodes, as shown in Figure 4.10.

We observe a boost in average speed and number of lane change after we apply our hierarchical action space. The results show the original DDPG algorithm tends to drive more conservatively and stays behind other vehicles more often without trying to overtake. In comparison, our

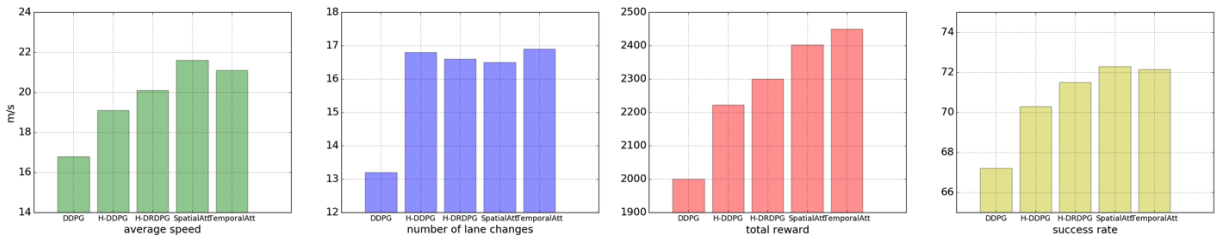

Figure 4.10: Comparison of different models on average speed, number of lane changes, total reward during an episode and percentage of successful episodes.

model is designed to learn separate policies for left lane change and right lane change, so it tends to act more aggressively for lane changes and finish one episode with less time. The resulting higher rewards show our method of hierarchical actions, temporal and spatial attention contribute to better training efficiency and stability. It should be clarified that although our method encourages overtake behavior, it still guarantees safety at the same time. Since overtake behavior could avoid getting too close to the front vehicle and hitting the rear-end, our method actually has fewer collisions with other vehicles, thus resulting in higher success rate for a complete episode than the original DDPG algorithm.

To better understand how the attention mechanism helps in terms of learning the driving behavior, we give the visualization of a turning and lane changing situation. In the scenario given by Figure 4.11, increasing weights are assigned to the input sequence when the car is making a sharp turn at lane corner. In this case, a single last frame does less help than a sequence of frames in that. By looking back into the past (by looking at the weighted sum of features of several frames before), the agent can determine the best trajectory for turning at the corner.

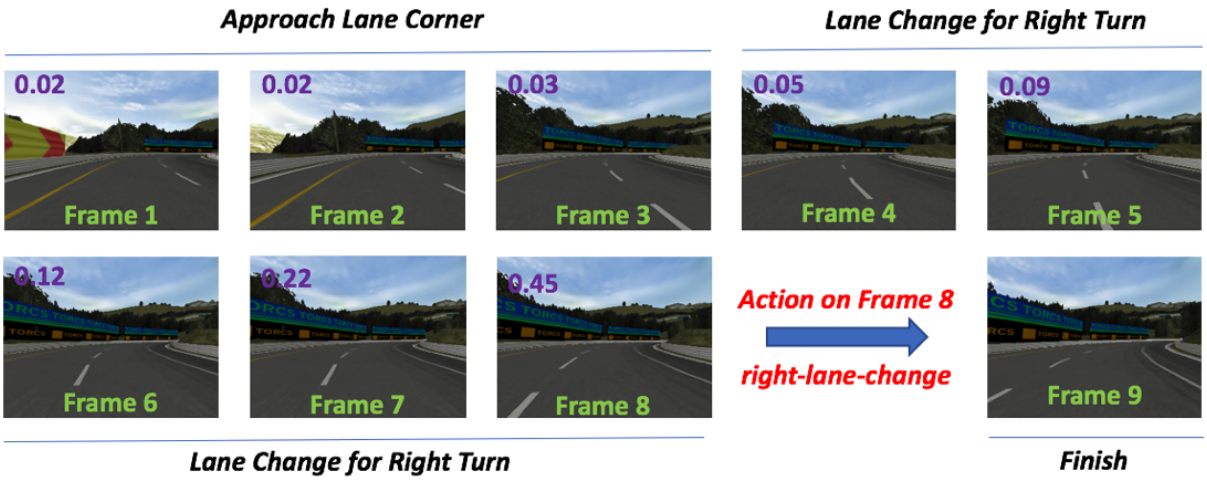

Figure 4.11: **Qualitative results of Temporal Attention DRDPG** The number in the upper-left corner of each image is the weight assigned to that frame (higher weight indicates more importance).

In the scenario given by Figure 4.12, the regions of lane end and front vehicle are targeted by the spatial attention model. For this overtaking behavior, we can see that the attention model focuses on the front vehicle and figures out when and how to launch the proper lane change behavior. By learning a mask over the input domain (more specifically, the CNN features with correspondence to the input image), the agent can grab the context relevant to the task and learns the proper behavior more efficiently.

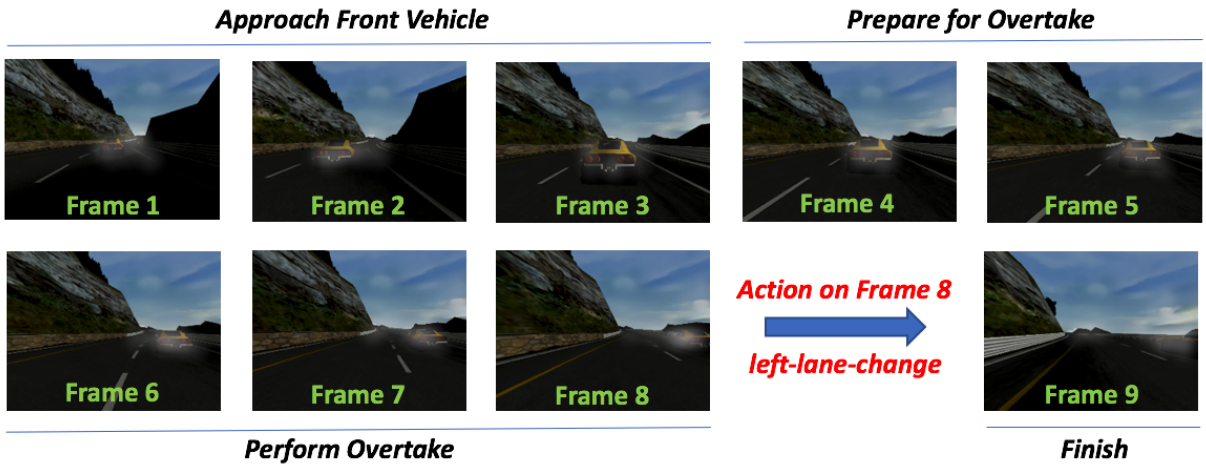

Figure 4.12: **Qualitative results of Spatial Attention DRDPG** A mask over the input domain is learned by the attention mechanism. Brighter colors indicate higher weights. The weights are smoothed with a Gaussian kernel in the visualization.

4.4 Discussion

The overall performance boost compared to the original DDPG-based method validates our methodology of introducing a hierarchical action space and shows that the attention mechanism is meaningful for learning proper driving behaviors.

The proposed hierarchical action space for learning driving behaviors establishes sub-policies for lane change behaviors. This subdivided policy ensures our model can have specialized policies for different driving styles on the road, while still maintaining an end-to-end learning procedure without manually designing submodules for a lane change. The final model, as shown in the results section, can encourage lane change behavior, which results in more stable and efficient driving with higher average speed. This hierarchical action space for DDPG gives us another way to think about how to incorporate a hierarchy of behavior in learning an end-to-end policy. Although during the whole process, we do not give direct instructions on when and how a lane change should be performed, the system still learns to separate different patterns in driving, which results in a performance boost of the overall driving in the end.

We also introduce an attention mechanism which is prevalent in traditional computer vision tasks (e.g., video/image captioning) into recurrent DDPG in an attempt to speed up training by learning from filtered features and prioritizing history states. We demonstrate our attention model

indeed learns meaningful content in the task of driving. The output of the learned attention layers gives us a chance to explain what is learned in the black box of the neural network.

The drawbacks and probable future work directions are: (1) we introduced more weights into the network which might potentially slow down the training speed per iteration; (2) the LSTM is often unstable in training: a possible solution is to stabilize LSTM and expand its capacity by introducing bidirectional LSTM or using layers of LSTM instead of just one layer; (3) we have not used any predefined physical knowledge to help reduce the exploration and search space in the DDPG algorithm. For example, we could leverage the fact that when we are in the leftmost lane, there is no chance we can do a left lane change.

Chapter 5

LSTM-based Lane Change Behavior Prediction

5.1 Problem Overview

In the previous two chapters, we investigated using supervised learning and deep reinforcement learning to directly learn a driving model for autonomously driving on-road both in simulation and in a real-world dataset. In this chapter, we focus on a more specific aspect of on-road driving: predicting lane change behavior. It is essential to understand other drivers' lane-change behavior so that we can better plan our driving trajectories on the road. Predicting other drivers' behavior before their real action can be observed will leave us more time to plan how we should react.

We intend to discover the common characteristics of lane change behavior performed by human drivers. By using real human driving data, we intend to construct a system that can answer the following question: when will a car near us do a lane change and what trajectory will the car follow when it performs the lane change action?

Instead of assuming that surrounding vehicles follow some unified pre-defined lane change maneuver, we adopt an approach that directly derives the model from real human driving data. The advantage of a learning-based model lies in the fact that it does not make assumptions about human behavior and can directly mimic it . Here, we use an LSTM as the learning structure. LSTM has shown its potential to learn objectives evolving through time in tasks like machine translation, video recognition and image captioning. We intend to incorporate temporal environment state information through the LSTM model to make better decisions in the current state.

We separate the lane change behavior into two high-level stages: decision and execution. The decision stage can also be viewed as the intention of a human driver. In this stage, the driver observes the environment and decides according to the traffic flow whether it is a proper time to do a lane change. This decision is a discrete choice: the agent can make the choice between a left lane change, a lane following and a right lane change. After this high-level decision, we intend to learn a trajectory generator that can directly perform the lane change action. This action depends on its current ego state and can vary according to factors distance to surrounding vehicles.

Here, we present an LSTM-based method that can predict other cars' intentions as well as trajectories based on observation from the environment. A systematic overview of our lane

change prediction is shown in Figure 5.1.

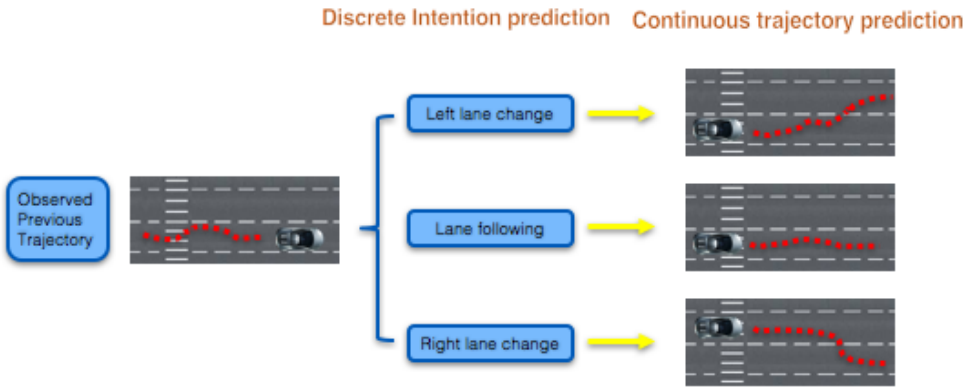

Figure 5.1: A systematic overview of the lane change behavior prediction algorithm.

5.2 Methodology

5.2.1 Problem Formulation

We formulate our lane change prediction problem as two stages. The discrete intention prediction is defined as : at given time t, our goal is to determine a particular vehicle's intention given our sensor observation of the vehicle's state as well as the surrounding environment information in the past T_p timesteps. T_p can vary according to how long we would like to consider the history. The intention can be one of the following three discrete decisions: left lane change, lane following, and right lane change. It can be formally represented as:

$$I_t = \{left\_turn, right\_turn, go\_straight\}$$

. The continuous trajectory prediction is defined as: we also predict the future position coordinates of the target vehicle up to T_f timesteps in the future according to the predicted intention. T_f varies according to how long we would like to predict the future. The model we plan to derive is deterministic. The future trajectory is represented by a discrete sequence of lane coordinates. The trajectory prediction is based on the classification result of intention prediction and we have a separate model to predict the trajectory for each category of intention. The predicted trajectory can be represented as:

$$O_t = \{(x_{t+i}, y_{t+i})\}_{i=1}^{T_p}$$

We make predictions based on the target vehicle's information and observation of the target vehicle's surrounding environment, which we summarize as the state space:

$$s^t = [s_t^t, s_s^t]$$

The target state s_t^t contains basic vehicle information including global coordinates of the agent and its absolute speeds in the horizontal and vertical directions:

$$s_t^t = [p_{t_x}^t, p_{t_y}^t, v_{t_x}^t, v_{t_y}^t]$$

Here y corresponds to the direction of the lane and x corresponds to the direction perpendicular to the lane. The origin of the coordinates starts from the leftmost lane mark, as shown in Figure 5.2.

The surrounding state s_s^t contains the relative vehicle information of 6 surrounding vehicles and two binary indicator variables:

$$s_s^t = [s_{v1}^t, s_{v2}^t, s_{v3}^t, s_{v4}^t, s_{v5}^t, s_{v6}^t, b_l, b_r]$$

The binary variables b_l and b_r indicate whether the target vehicle's current lane has a lane to its left or right. $v1 - v6$ correspond to the six vehicles we take into consideration. They are vehicles in front of and behind the target vehicle in left lane, target lane and right lane, as shown in Figure 5.2. The relative vehicle information also contains location and speed information but is relative to the target vehicle.

$$s_v^t = [p_{v_x}^t - p_{t_x}^t, p_{v_y}^t - p_{t_y}^t, v_{v_x}^t - v_{t_x}^t, v_{v_y}^t - v_{t_y}^t]$$

If any of these vehicles don't exist or the target vehicle doesn't have a left or right lane, we set a default value for the vehicle information. By default, we assume the vehicle of interest have the same speed of the target vehicle at infinity position to the direction of the lane.

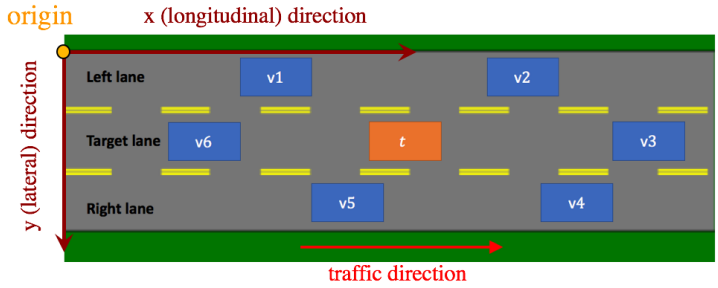

Figure 5.2: Notation for vehicles in consideration on road. Orange vehicle is the target vehicle; the blue vehicles are the 6 vehicle that are incorporated into the surrounding information.

In summary, our model makes discrete intention prediction I_t and future continuous trajectory prediction O_t of up to T_f seconds in the future at timestep t, based on our observation of the past states $(s^{t-T_p}, ..., s^{t-1})$ in T_p timesteps.

5.2.2 Predicting Discrete Driver Intention

Our intention prediction algorithm is based on a deep neural network structure that use LSTM units. The overall network architecture is shown in Figure 5.3. The state at each timestep is first passed through a fully connected layer of 256 hidden units. Then the LSTM receives as input the concatenation of the vehicle states in the past T_p timesteps. The choice of T_p is set to 5 seconds in our experiment for best performance after comparison with $T_p = 1s$ and $T_p = 3s$. It has been shown that deep LSTMs significantly outperform shallow LSTMs [50], so we use a stack of 4 LSTM layers, each with 128 hidden units. Note that we only unroll LSTM to T_p timesteps for training convenience and speed. Each LSTM unit has a recurrent context vector c_t as output and

a hidden embedding h_t that is passed through time. Here the initial state h_0 is set to zero during training. After unrolling, the recurrent context vector output of the last LSTM stack is passed to another fully connected layer to decide the current discrete maneuver choice. A softmax layer is used to calculate the possibilities of a left lane change, lane following, and right lane change. The overall model is trained end-to-end as a multi-class classification problem using cross-entropy as the loss function.

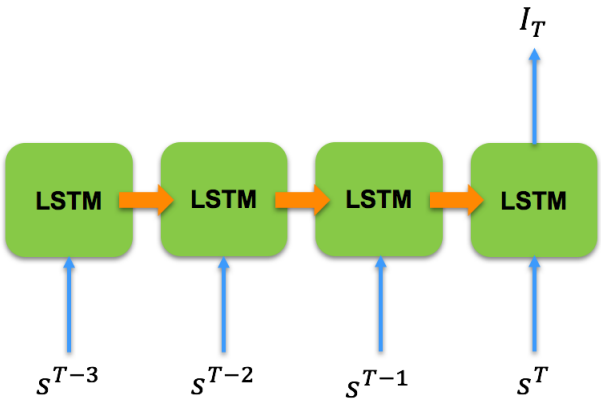

Figure 5.3: Network architecture for predicting discrete driver's intention.

5.2.3 Predicting Continuous Vehicle Trajectory

Our continuous vehicle trajectory prediction is based on preliminary result of discrete intention prediction. We choose the trajectory generator with the maximum likelihood and make conditional predictions. Models of a left lane change, lane following, and a right lane change are derived separately from driving scenarios falling into these categories.

The network architecture for this sequence-to-sequence trajectory prediction task is shown in Figure 5.4. The goal of using LSTM here is to estimate the conditional probability of the trajectory $p((x_{T+1}, y_{T+1}), ..., (x_{T+T_f}, y_{T+T_f})|s_{T-T_p}, ..., s_{T-1})$. Our model can be seen as first extracting a context representation v of input states $s_{T-T_p}, ..., s_{T-1}$, which is given by the hidden state of LSTM h_T at timestep T. This part can be regarded as processing past information into v. Later the predictions x_t and y_t are made sequentially using previous state s_{t-1} and previous LSTM hidden state h_{t-1}, which starts with v. Note, we have two separate networks with the same structure but different parameters for predicting x_t and y_t. We find out that x_t and y_t are not strongly correlated and it harms the convergence if we train them together. The LSTM prediction formulation can be represented as

$$p((x_{T+1}, y_{T+1}), ..., (x_{T+T_f}, y_{T+T_f})|s_{T-T_p}, ..., s_{T-1}) = \prod_{t=T+1}^{T+T_f} p(x_t, y_t|v, s_{T+1}, ..., s_{t-1})$$

The structure of the FC layers and LSTM cells is the same as in the network for predicting discrete driver's intention. The loss function we adopt here is a simple regression loss over T_f

timesteps into the future

$$L_x(\theta) = \sum_{t=T+1}^{T+T_f} (x_t - \hat{x}_t)^2 + +\lambda \sum_{i=1}^{k} \theta_i^2$$

$$L_y(\beta) = \sum_{t=T+1}^{T+T_f} (y_t - \hat{y}_t)^2 + \lambda \sum_{i=1}^{k} \beta_i^2$$

where x_t, y_t are ground truth for the vehicle's coordinates and $\hat{x}_t, \hat{y}_t$ are our predictions. A L2 regularization term is added to prevent overfitting.

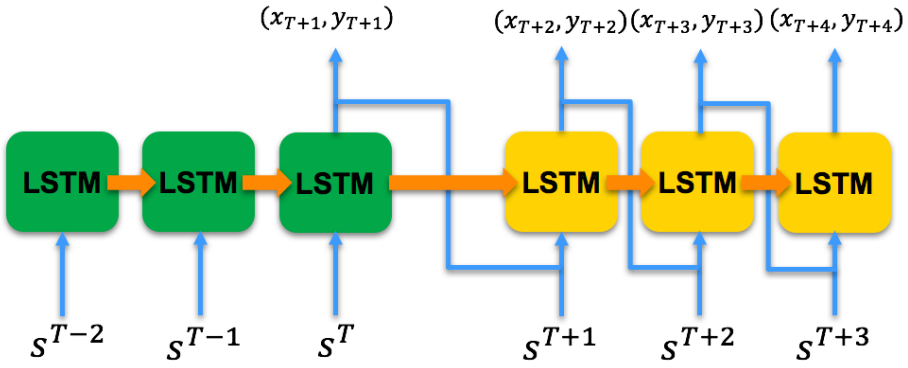

Figure 5.4: Network architecture for predicting continuous vehicle trajectory.

5.3 Experiment and Analysis

5.3.1 Data Preparation

We use the NGSIM dataset as our test benchmark. The NGSIM dataset [51] was collected by US Department of Transportation in Los Angeles, the USA in 2007. We use US-101 and I-80 data from NGSIM to study detailed vehicle trajectories on US highways. The two datasets contain vehicle trajectories in an area of 640 meters and 503 meters of a six-lane highway on US-101 and I-80 respectively. The dataset contains six 15-minute periods of vehicle trajectories on a busy highway. The dataset provides the location and speed of each vehicle in the study area with a frequency of 10 Hz.

We extract trajectories of all vehicles in the dataset. To define the trajectory sections involving lane change, we consider the frame where the vehicle's lane ID changes and take it as the timestep that a lane change behavior happens. This way, we detect 490 right lane change and 978 left lane change scenarios. As can be seen, the dataset is unbalanced towards the left lane changes. One possible reason may be that people prefer to go to the leftmost lane for higher speed. We also collect lane following scenarios with the criteria of not containing any lane change frames in the time span. To prevent biasing through an unbalanced dataset, we select 490 scenarios for each of the three actions for training and testing.

For each maneuver scenario, we extract a time period of 10 seconds centered at the lane change frame and use the first 5 seconds as history data and the second 5 seconds as future data we would like to predict. To predict lane change behavior before the actual lane change point, we augment the data with a time period of 10 seconds centered at 0.5, 1, 1,5, 2, 2.5 and 3.0 seconds before the lane change point. An illustration of how lane change frames is selected is shown in Figure 5.5. For the trajectory prediction task, we mix all these samples for training.

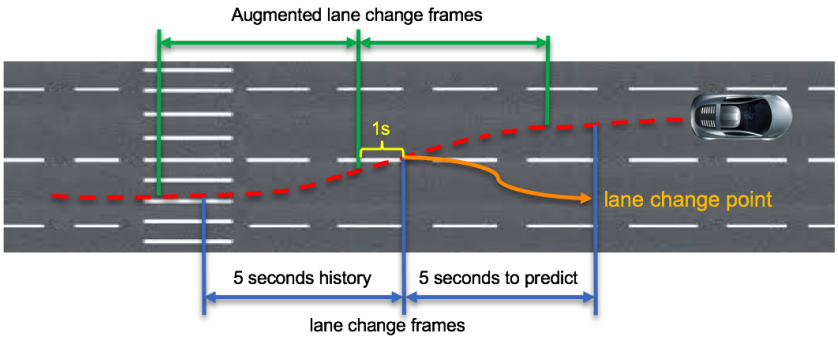

Figure 5.5: Illustrations of how lane change frames are selected. The augmented lane change frames can be 0.5, 1, 1,5, 2, 2.5 and 3.0 seconds before the lane change point, but we only show 1 second beforehand as an example.

5.3.2 Experimental Results

For the discrete driver intention prediction, we compare our results with logistic regression. For the logistic regression implementation, we use all the states from T_p past frames as input and directly map to maneuver types. We compare the prediction precision and recall at different times before the lane change point, as shown in Table 5.1.

Time before Lane Change Point	Recall							Precision
	3s	2.5s	2s	1.5s	1s	0.5s	0s	
Left Lane Change	50.3%	56.5%	62.3%	73.3%	82.2%	89.4%	91.2%	77.3%
Lane Following				82.6%				81.3%
Right Lane Change	51.1%	57.2%	64.2%	74.4%	83.6%	91.0%	92.2%	75.6%

Table 5.1: Recall and precision for discrete driver's intention prediction for different times before the lane change point.

Here recall means the percentage of recognizing the behavior in all observation scenarios. Precision means the percentage that our prediction is correct. Our model can achieve above 90% recall of driver's intention when the car of interest approaches the lane change point, while the recall decays to nearly 50% about 3 seconds before the lane change point. The results show that our model can detect lane change behaviors with at least 80% accuracy when a car is approaching lane mark 1 second before and at least 60% 2 seconds before. The precision here is counted on all scenarios, and all three categories achieve nearly 80% accuracy, which means our model is quite reliable in its prediction.

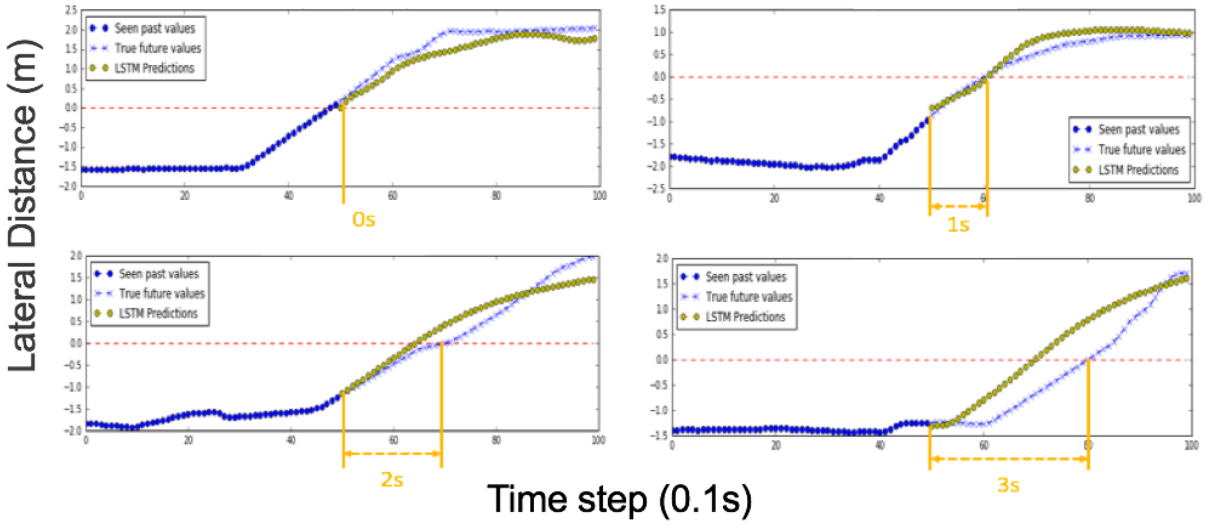

Figure 5.6: Illustration of lateral distance prediction result with different prediction times. The red line corresponds to the lane marking. The four figures show the result of our algorithm predicting at 0 or 1, 2 and 3 seconds before the lane change point.

After categorizing the driver's intention, the appropriate model among the 3 types of maneuver to predict the continuous trajectory is applied. We train our model using RMSProp optimization with learning rate 0.0005, decay 0.9 and momentum 0.0001. We set the limit to 15,000 iterations in total, which takes 12 hours in training. Figure 5.6 shows examples of the quality of our model in predicting 0 or 1, 2 and 3 seconds before approaching the lane change point. Regardless of the trajectory's previous shape, our model can generate robust, human-like future trajectories. For example, about 3 seconds before the lane change point, the previous trajectory contains little information, but our model can still successfully forecast the future lane change behavior.

We calculate the prediction error against the length of the prediction, as shown in Figure 5.7. The error is defined as the average absolute difference between prediction and ground truth at a given prediction time. We observe that the error accumulates as uncertainty increases in the future for both the lateral and longitudinal direction. However, the lateral prediction error drops for both left lane change and right lane change after 4.5 seconds. We believe this phenomenon occurs because the model captures the behavior that vehicles will tend to follow the center of the lane after their lane change behavior.

We make a comparison of our model with curve-fitting based methods for extrapolation as suggested by [52, 53]. Curve fitting methods minimize the total deviation between a predefined shaped curve with the existing data by looking for the optimal parameters of the curve. The resulting curve can have the best fit to a series of seen data points and can predict future trajectory beyond the original observation range, though subject to greater uncertainty. The three curves

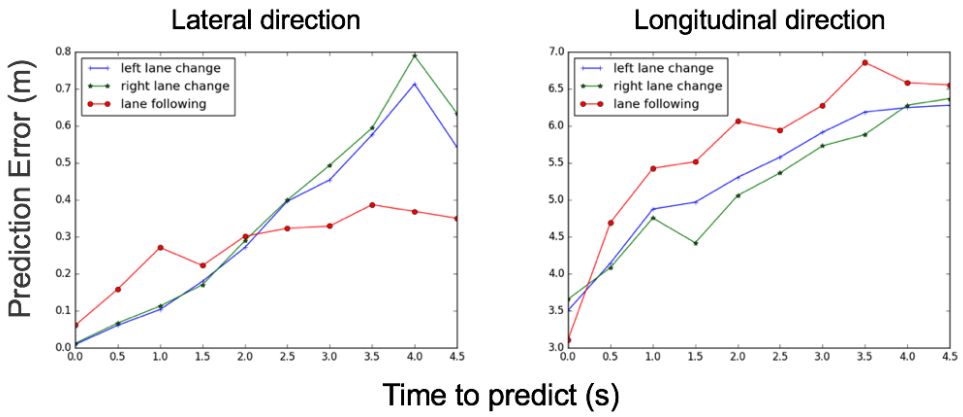

Figure 5.7: Prediction error v.s length of prediction. Left: lateral distance prediction. Right: Longitudinal distance prediction.

we define are linear, quadratic and sigmoid fit with the following representations.

$$
\begin{aligned}
f_{linear} &= ax + b \\
f_{quadratic} &= ax^2 + bx + c \\
f_{sigmoid} &= a \pm 4/(1 + e^{-bx+c})
\end{aligned}
$$

Linear and quadratic are widely used polynomial curves for fitting, while sigmoid is less commonly used. We select sigmoid here mainly for the reason that the shape of a lane change strongly resembles a sigmoid function. We further set the height of the sigmoid function to 4 meters, which is the typical width of a lane. We solve the curve fitting problem using least square loss with gradient descent.

The quantitative results comparing different methods are shown in Table 5.2. We use 80% of the trajectories in each category for training and 20% of the trajectories for testing. We calculate the average total square error for a prediction horizon of 5 seconds for all test trajectories. The error metric is defined as

$$
Error = \frac{1}{T}\sum_{t=1}^{T}\sqrt{\frac{\sum_{i=1}^{D}[(\hat{y}_t^i - y_t^i)^2 + (\hat{x}_t^i - x_t^i)^2]}{D}}
$$

Here, T stands for the number of timesteps in the prediction, D is the the number of test trajectory samples and x_t^i, y_t^i represent the x, y in sample i at timestep t in the prediction. We can observe that our LSTM-based method has the lowest error for predictions compared to the linear, quadratic and sigmoid curve fitting methods. We also compare the prediction result with and without the surrounding environmental information. The result show that the surrounding vehicles indeed positively help to increase the prediction precision. By incorporating the surrounding states, the social interaction between the vehicles is implicitly considered on the road.

A typical sample result of the different methods applied to two different lane change scenarios is shown in Figure 5.8. We can see that our LSTM model works best in predicting the future trajectory. The linear and quadratic polynomial curves work fine over short distances but tend to

	Linear	Quadratic	Sigmoid	LSTM w/	LSTM w/o
Left Lane Change	6.01	7.13	4.84	**4.00**	4.55
Lane Following	3.46	4.91	-	**3.21**	3.41
Right Lane Change	5.78	6.96	4.89	**4.15**	4.49

Table 5.2: Comparison of mean square error (meters) for 50 timesteps of trajectory between curve fitting methods and our LSTM-based method with/without surrounding state.

diverge significantly after about 50 meters. The sigmoid fit successfully recovers the shape of a typical lane change but fails to adjust flexibly to lane changes case by case. Our model however, has both stability and accuracy for predictions as far as 5 seconds in the future.

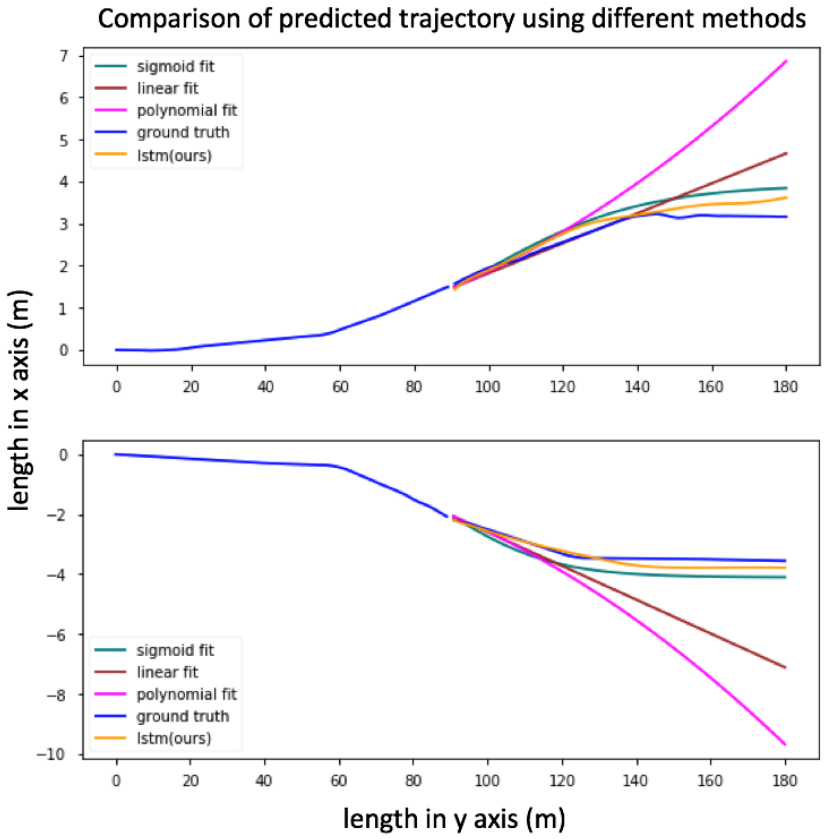

Figure 5.8: Comparison of trajectory prediction results using linear fit, polynomial fit, sigmoid fit and LSTM (ours). The left half of the trajectory represents seen observations and the right half represents predictions. The two plots show a typical left and right lane change scenario respectively.

5.4 Conclusion

In this chapter, we have demonstrated we can learn lane change behavior using LSTMs. We have shown preliminary results for predicting both discrete driver intentions and continuous vehicle trajectories. Our proposed models have shown improved results over logistic regression for intention prediction as well as curve fitting methods in trajectory prediction. By combining the two prediction subroutines, our system can predict possible positions of observed vehicles surrounding the target vehicle in a short period of future time in highway settings.

Since we divide the task into first predicting intention and then the trajectory, our model is interpretable and easy to train and fine-tune. While we only consider lane change maneuvers in highway settings in this study, we can further extend our model to other maneuver types in more complex settings. Thanks to the transparency of our composite model, we only need to define a new maneuver type and then let the model learn to classify it and predict its trajectory shape. The whole process will be smooth and won't harm our learned maneuver prediction. A possible next step may be to predict turning and lane crossing in intersection scenarios, which is common in city driving.

Chapter 6

Conclusion and Future Work

This work investigates the use of neural networks for behavior learning and prediction in autonomous on-road driving, particularly for lane following and lane changing tasks. First, we study learning a vision-based end-to-end maneuver controller for autonomous vehicles. The task is to construct a deep neural network to map from sensor input to control output. We improve the architecture described in [7] in four respects to enhance the performance. We use transfer learning to incorporate prior knowledge, add temporal information with LSTM, include image segmentation information as an auxiliary task and apply sensor fusion with vehicle kinematics. We test our new model both in simulation and on a real-world dataset and compare with the previous method. We observe a significant performance boost and see that the model can quickly adapt to new unseen environments.

In the second work, for the reason of data efficiency, we switch to a deep reinforcement learning (DRL) method for the same task with a focus on lane changing behavior. We propose a learning structure with a hierarchy for the purpose of learning explicit sub-policies for lane change behavior. We explore the attention mechanism when making decisions for on-road driving. We design a new layer that doesn't need extra information but can learn the importance of inputs through space and time. The test results in a racing car game TORCS demonstrate successful lane change behavior and more stable driving experience with reduced training time compared to the DDPG baseline.

Finally, we investigate the task of predicting lane change behavior for on-road driving. We formulate and subdivide this problem into predicting discrete driver's intention and predicting continuous vehicle trajectories given the past environment states in a time window. Each problem is solved separately, and the results are combined for overall evaluation. We construct the models using LSTM to learn the temporal information through history states. The experiment is done on the NGSIM dataset, which contains hundreds of real-world lane changing trajectories. The prediction is compared with curve fitting methods, and the result shows our method can best replicate human lane changing behavior.

For future work, some interesting directions can serve as a follow-up:

1. Use Inverse Reinforcement Learning (IRL) to recover the reward function for on-road behavior learning instead of manually designing reward functions. The difficulty lies in keeping the algorithm efficient and convergent when the state space is high-dimensional, and the search space is vast.

2. Use social LSTM [54] to consider the interaction of vehicles directly in the model construction for prediction of vehicle's intention and trajectory.
3. Currently, our algorithms are all trained and tuned to one or several specific simulations or datasets. It would be meaningful to consider the problem of domain adaptation and try to derive a universal model that can easily be used or fine-tuned in a new environment.

Bibliography

[1] Cmu boss autonomous car for the 2007 darpa challenge. URL https://www.nrec.ri.cmu.edu/solutions/defense/other-projects/urban-challenge.html. (document), 1.1

[2] Uber self-driving car. URL https://www.uber.com/cities/pittsburgh/self-driving-ubers/. (document), 1.1

[3] Waymo self-driving car. URL https://waymo.com/. (document), 1.1

[4] Cruise self-driving car. URL https://getcruise.com/. (document), 1.1

[5] lane change need in highway driving. URL https://www.theguardian.com/technology/2015/sep/13/self-driving-cars-bmw-google-2020-driving. (document), 2.2

[6] Visualization of perception in a legitimate lane change behavior. URL https://thetechreader.com/tech/self-driving-cars-the-future-is-on-the-horizon-autonomous-cars-are-com (document), 2.2

[7] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. (document), 1.3, 2.3, 3.1, 3.1, 3.3, 3.3, 6

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016. (document), 4.2.2, 4.3

[9] Takeo Kanade, Chuck Thorpe, and William Whittaker. Autonomous land vehicle project at cmu. In *Proceedings of the 1986 ACM fourteenth annual conference on Computer science*, pages 71–80. ACM, 1986. 1.1

[10] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009. 1.1, 2

[11] Walther Wachenfeld, Hermann Winner, J Chris Gerdes, Barbara Lenz, Markus Maurer, Sven Beiker, Eva Fraedrich, and Thomas Winkle. Use cases for autonomous driving. In *Autonomous Driving*, pages 9–37. Springer, 2016. 1.1

[12] Tomasz Janasz and Uwe Schneidewind. The future of automobility. In *Shaping the Digital*

Enterprise, pages 253–285. Springer, 2017. 1.1

[13] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008. 1.1

[14] Michal Paden, Brian, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016. 1.2

[15] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016. 1

[16] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. Probabilistic mdp-behavior planning for cars. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1537–1542. IEEE, 2011. 2

[17] Xingxing Du, Xiaohui Li, Daxue Liu, and Bin Dai. Path planning for autonomous vehicles in complicated environments. In *Vehicular Electronics and Safety (ICVES), 2016 IEEE International Conference on*, pages 1–7. IEEE, 2016. 3

[18] Liang Ma, Jianru Xue, Kuniaki Kawabata, Jihua Zhu, Chao Ma, and Nanning Zheng. Efficient sampling-based motion planning for on-road autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1961–1976, 2015. 3

[19] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992. 4

[20] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013. 4

[21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016. 1.3

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS'13 Workshop on Deep Learning*, 2013. 2.1, 2.3

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 2.1

[24] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 2.1

[25] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999. 2.2

[26] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373,

2016. 2.3

[27] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016. 2.3

[28] Christopher R Baker and John M Dolan. Traffic interaction in the urban challenge: Putting boss on its best behavior. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1752–1758. IEEE, 2008. 2.4

[29] Hidehisa Yoshida, Shuntaro Shinohara, and Masao Nagai. Lane change steering manoeuvre using model predictive control theory. *Vehicle System Dynamics*, 46(S1):669–681, 2008. 2.4

[30] Chao Huang, Fazel Naghdy, and Haiping Du. Model predictive control-based lane change control system for an autonomous vehicle. In *Region 10 Conference (TENCON), 2016 IEEE*, pages 3349–3354. IEEE, 2016. 2.4

[31] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*, 2016. 2.4

[32] Sang-Hyun Lee and Seung-Woo Seo. A learning-based framework for handling dilemmas in urban automated driving. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1436–1442. IEEE, 2017. 2.4

[33] Stefanie Manzinger, Marion Leibold, and Matthias Althoff. Driving strategy selection for cooperative vehicles using maneuver templates. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 647–654. IEEE, 2017. 2.4

[34] Jianqiang Nie, Jian Zhang, Xia Wan, Wanting Ding, and Bin Ran. Modeling of decision-making behavior for discretionary lane-changing execution. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pages 707–712. IEEE, 2016. 2.4

[35] Tobias Rehder, Wolfgang Muenst, Lawrence Louis, and Dieter Schramm. Learning lane change intentions through lane contentedness estimation from demonstrated driving. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pages 893–898. IEEE, 2016. 2.4

[36] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 3.2.1

[37] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010. 3.2.2

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3.4

[39] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. End-to-end deep reinforcement learning for lane keeping assist. *arXiv preprint arXiv:1612.04340*,

2016. 3.4, 4.1

[40] Peter Wolf, Christian Hubschneider, Michael Weber, André Bauer, Jonathan Härtl, Fabian Dürr, and J Marius Zöllner. Learning how to drive in a real world simulation with deep q-networks. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 244–250. IEEE, 2017. 4.1

[41] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based torcs. 2013. 4.1

[42] Ron Alterovitz, Sven Koenig, and Maxim Likhachev. Robot planning in the real world: research challenges and opportunities. *AI Magazine*, 37(2):76–84, 2016. 4.1

[43] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015. 4.1, 4.2.3, 4.2.3

[44] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 2015. 4.2.1, 4.2.3, 4.2.3

[45] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014. 4.2.3

[46] Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184, 2012. 4.2.3

[47] Wenjie Pei, Tadas Baltrušaitis, David MJ Tax, and Louis-Philippe Morency. Temporal attention-gated model for robust sequence classification. *arXiv preprint arXiv:1612.00385*, 2016. 4.2.3

[48] Clare Chen, Vincent Ying, and Dillon Laird. Deep q-learning with recurrent neural networks. *stanford cs229 course report*, 2016. 4.2.3

[49] Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. Deep attention recurrent q-network. *arXiv preprint arXiv:1512.01693*, 2015. 4.2.3

[50] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 5.2.2

[51] Next generation simulation (ngsim) vehicle trajectories and supporting data. data available at `https://catalog.data.gov/dataset/next-generation-simulation-ngsim-vehicle-trajectories`. 5.3.1

[52] Israa Hadi and Mustafa Sabah. Behavior formula extraction for object trajectory using curve fitting method. *International Journal of Computer Applications*, 104(2), 2014. 5.3.2

[53] Yu Bing Dong, Ming Jing Li, and Hai Yan Wang. Trajectory prediction algorithm based on iterative fitting. In *Applied Mechanics and Materials*, volume 687, pages 3988–3991. Trans Tech Publ, 2014. 5.3.2

[54] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei,

and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971, 2016. 2

[55] William J Mitchell, Chris E Borroni-Bird, and Lawrence D Burns. *Reinventing the automobile: Personal urban mobility for the 21st century*. 2010.

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

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.