My Approach for CS492 Driving Competition based on Simulator SMARTS

Zhaoyu Li[*]
Department of Computer Science
Shanghai Jiao Tong University
`apollo19990425@sjtu.edu.cn`

Abstract

Self-driving cars have been a hot topic in recent years, and is a thriving and booming research area holding the promise of a new generation in transportation development. Intending to further research in autonomous driving and in identifying rising stars, our CS492 Reinforcement Learning lesson is hosting a competition under the support of HUAWEI Autonomous Driving Platform and their modern Simulator SMARTS. For my approach, I use discrete action and PPO algorithm to train the model. I also do some comparative experiments to achieve quite good performance on both public dataset (Rank 4) and competition (Rank 3).

1 Introduction

Self-driving cars (also known as autonomous cars and driverless cars) have been studied and developed by many universities, research centers, car companies, and companies of other industries around the world since the middle 1980s [Badue et al., 2019]. The architecture of the autonomy system of self-driving cars is typically organized into two main parts: the perception system, and the decision-making system. The perception system is generally divided into many subsystems responsible for tasks such as autonomous car localization, static obstacles mapping, road mapping, moving obstacles detection and tracking, traffic signalization detection and recognition, among others. The decision-making system is commonly partitioned as well into many subsystems responsible for tasks such as route planning, path planning, behavior selection, motion planning, obstacle avoidance and control, though this partitioning is somewhat blurred and there are several different variations in the literature.

For decision making part, it's nature to use reinforcement learning to learn how to make a good decision. However, as RL relies on try and error strategies, an end-to-end driving prototype still seems too dangerous for real-life learning, so many researchers develop some simulators to train models in a virtual environment to avoid these problems.

SMARTS is one of the simulators that can model realistic dynamical behaviour and offer these scenarios that emulates real-world behaviours at different granularity levels to bridge the gap between research and application. In this course work, we build our model based on SMARTS to train a self-driving car which can not only drive safely but also drive quite fast that it is able to overtake other social vehicles by changing lanes.

The remainder of this report is structured as follows. In Section 2, I briefly introduce SMARTS and its backbone lib – RLlib. In section 3, I present the method I use to train the self-driving car. In Section 4, I show the results of my model, some ablation studies and comparative experiments. Finally, the conclusion will be drawn in Section 5.

[*]This is a course report for *Reinforcement Learning*, CS492, SJTU

2 Background

2.1 SMARTS and RLlib

Scalable Multi-Agent RL Training School (SMARTS) is an autonomous driving platform for RL. It supports fast and flexible construction of RL environments and flexible scripting of training scenarios in Python. SMARTS provides Scenario Studio, a flexible tool to generate different traffics in a scenario and create our own maps. For agent, we can create and use adapters to wrap interaction between an agent model and a SMARTS environment. For policy training, we can use the RLib framework to train an agent. RLlib is an open-source library for reinforcement learning that offers both high scalability and a unified API for a variety of applications [Liang et al., 2018]. RLlib natively supports TensorFlow, TensorFlow Eager, and PyTorch, but most of its internals are framework agnostic.

3 My Approach

For my approach, I try several methods, including different observations, different action spaces (discrete action space and continuous action space) and different reward functions. In this section, I will introduce them in order.

3.1 Observation

The default observation has 6 attributes:

- Distance from center, which calculates the distance from the car to the center of the lane.
- Angle error, which calculates the angle between the lane and the front of the car.
- Speed, which records the ego car's speed.
- Steering, which records the angle of front wheels in degrees.
- Ego lane dist, which calculates the distance between the ego car and the closest social vehicle along each path.
- Ego ttc, which calculates the time to collide with the closest social vehicle along each path.

The first four features are quite straight forward, they describe the state of the ego vehicle. From a higher perspective, the last two features describe the global information of the ego car. Since the state of the ego car is very important and necessary, so I firstly try to modify the last two information in observation. The default implementation of ego lane dist and ego ttc are calculated based on at most three lanes: the current lane and the two lanes nearby. However, there are more than three lanes in the competition and I think it's better for the ego car to have more lane information. Thanks to the demo code from our TA Jiayu Miu, he also implements the expansion of ego lane dist and ego ttc, so I use his 5-lane ego lane dist and ego ttc.

For continuous action, I find a problem during training that the ego car always drives off the road when there is a sharp bend since its speed is too fast (always faster than 45km/h) to make a turn. To address this issue, I consult from the code of discrete action controller in SMARTS which forces the ego car to slow down when the curve of the front road is sharp. So I add road curveness in the observation, which is derived by the interpolation of the coordinates of the waypoints in front of the ego car.

For discrete action, note that the car drive along the lane perfectly and always drives on the center of the road, it's not necessary to use distance from center, angle error and steering. So I discard them in the observation to reduce their influences, which makes a slight improvements. I also add road curveness in the observation as continuous action.

3.2 Action

There are three action types. The first two types lie in continuous action space while the last one's domain is discrete:

- Continuous: continuous action space with throttle, brake, absolute steering angle. It is a tuple of throttle [0, 1], brake [0, 1], and steering [-1, 1].
- ActuatorDynamic: continuous action space with throttle, brake, steering rate. Steering rate means the amount of steering angle change per second (either positive or negative) to be applied to the current steering angle. It is also a tuple of throttle [0, 1], brake [0, 1], and steering_rate, where steering rate is in number of degrees per second.
- Lane: discrete lane action space of strings including "keep_lane", "slow_down", "change_lane_left", "change_lane_right" and can also be a tuple of an integer for "lane_change" and a float for "target_speed".

For continuous action, I try both and conduct the comparative experiments with the same parameters. However, it seems like the first one performs better. I guess that it is because it's easy for the ego car to learn the absolute steering angle in the map than the relative steering since the latter needs additive ego car's steering information.

For discrete action, I use "lane_change" and "target_speed" instead of descriptive action since this type describes the speed and lane more accuracy and it's easier for model to learn. What's more, "keep_lane", "slow_down", "change_lane_left" and "change_lane_right" makes the model have 50 percent probability not to change the lane, which makes the ego car harder to change lane. So I finally choose "target_speed" = 20, 40, 60 for "change_lane" = -1, 0, 1 as my discrete action space, which makes the probability of the lanes for the ego car to choose equal.

3.3 Reward

The default reward function is env_reward, which calculates the distance the ego car drive in one step. However, there is a problem. When I print the reward on the screen, it seems like the env_reward is calculated based on the way points, that is, the env_reward is close to the number of way points the ego car travels and is not very accuracy. To address this issue, I use the tangent velocity of the ego vehicle as reward function, that is,

$$r = v \times cos(angle\_error \times \pi/180). \tag{1}$$

In order to help the model avoid accidents, I set a large penalty when the ego car off the road or collide with other vehicles. For continuous action space, I also set a small regularization on the distance from center to encourage the ego vehicle drive on the center of the road. The final reward function is the summation of above terms.

3.4 Training

Based on the framework of RLlib, all the observations are stacked together and fed into a simple two-layer full connected neural network. For policy, RLlib provides several existing algorithm to use. I intend to use the SOTA algorithm Soft Actor-Critic(SAC) [Haarnoja et al., 2018], however, it seems like SMARTS does not support it yet. So I choose Proximal Policy Optimization(PPO) algorithm to train the model [Schulman et al., 2017]. PPO algorithm has both the stability and reliability of trust-region methods but are much simpler to implement, which is a very popular algorithm to use in the study of reinforcement learning.

4 Experiments and Results

I use the empirical parameters to train the model, that is, learning rate = 1e-4, num_sgd_iter = 10, λ = 0.95, clip_param = 0.2, sgd_minibatch_size = 1024 and train_batch_size = 30720.

Firstly, I conduct a comparative experiments on continuous action and discrete action. All the parameters are the same except max_episode for continuous action is 3000 while max_episode for discrete action is 2000. However, although max_episode for discrete action space is much smaller, it's much better than the continuous action. We can see from the Figure 1, using continuous action makes the ego vehicle drive fast but less safe and stable that its average traveled distance is much shorter than the discrete one. So I finally use discrete action.

For observation, I conduct a experiment compare using 3-lane ego ttc dist and ego ttc with 5-lane ego ttc dist and ego ttc. It turns out that the more lanes makes the ego car see more lane to drive and to

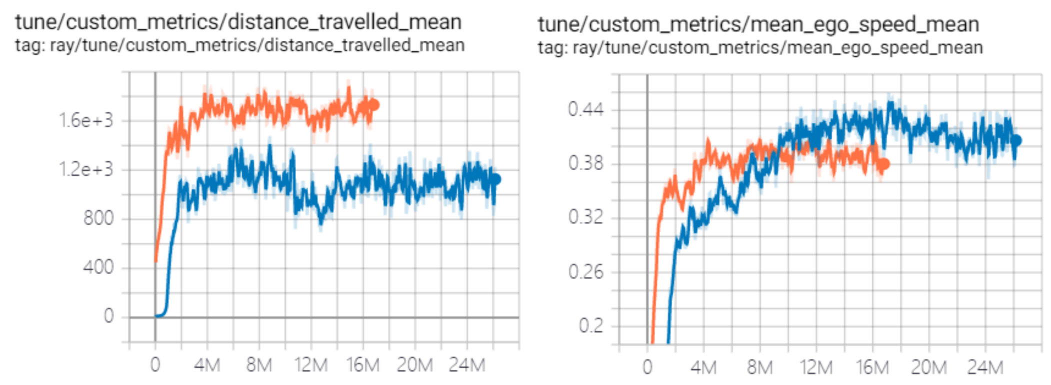

Figure 1: The training curve of continuous action and discrete action. The blue curve is for continuous action and the orange curve is for discrete action.

overtake other social vehicles. It makes it possible for the ego car change lane continuously which leads to a better performance. I also make an ablation study on the observation that discards distance from center, angle error and steering. It turns out that by discarding them in observation, the training is more stable, the ego car drive a little bit faster and its traveled distance can be more than 1900 while the max_episode is 2000.

For reward function, I change the penalty of accidents, but its influence is quite small that there is no distinct difference of the performance.

So I finally use discrete action with the observation of speed, 5-lane ego ttc dist ego ttc and road curveness and train the model for 24 hours, which leads to rank 4 in public dataset and rank 3 in competition. From the training curve, I also think the performance can be improved further by training for longer time.

5 Conclusion

In this course project, I train an self-driving car based on the simulator SMARTS. I carefully select the observation, action and reward function and use PPO algorithm to train the model, which leads to a quite good performance. This course work helps me get a deeper understanding of reinforcement learning as well as self-driving.

Acknowledgement

Thank my teacher Weinan Zhang, TA Ming Zhou and Jiayu Miu for their design and help of this project. Especially, thank Jiayu for solving the memory leak problem of envision which troubles me for a long time. Also thank Ziyu Wan and Yuxuan Chen who discuss with me about their approaches.

References

Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Paixao, Filipe Mutz, et al. Self-driving cars: A survey. *arXiv preprint arXiv:1901.04407*, 2019.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.