Car Racing on OpenAI Gym with Reinforcement Learning Based on DDQN

Qi Jiachen
2021533125
qijch@

Shi Junyan
2021533154
shijy@

Xu Chenhao
2021533119
xuchh@

Zhao Zeyu
2021533136
zhaozy2@

Zhu Yuhang
2021533173
zhuyh3@

Abstract

Car racing games have a very important practical significance and pose a series of challenging technical problems. This final project tries to build a model based on reinforcement learning techniques, Double Q-learning (DDQN) and the OpenAI Gym environment. We model a reward system and experiment with image processing and model achitectures. After 1,400 iterations, this agent can achieve the same level of performance as humans.
Github repo: `https://github.com/Sleepless9/CS181-2023fall`

1 Introduction

Car racing games can be set up as a real-world simulation, you can define different constraints and goals to be achieved. Generally, OpenAI Gym's CarRacing problem needs the car to traverse all track tiles to finish the game, instead of running a full lap. While, in this project, we set the end flag to have no points for a while (in frames), but only points deducted, for this end condition is intended to encourage the agent to keep moving forward on the road, and end quickly when yawing, which maybe useful in the field of autonomous driving.

2 Problem Analysis

2.1 Environment

We choose to use OpenAI Gym's *CarRacing-v0* version environment to train our car agent. The track is randomly generated each time the simulation begins, and is referred to as an "episode". Each state is a 96×96 RGB pixel frame which is the only thing that agent could observe. The frame is a top-down view of an area including and surrounding the car's current position. Between two frames, the car could make a decision to take an action or not. The action space is a combination of steering direction, gas and break. Each of them ranges in [-1, 1]. The whole action space is shown below:

Table 1: Action Space

(-1, 1, 0.5)	(0, 1, 0.5)	(1, 1, 0.5)
(-1, 1, 0)	(0, 1, 0)	(1, 1, 0)
(-1, 0, 0.5)	(0, 0, 0.5)	(1, 0, 0.5)
(-1, 0, 0)	(0, 0, 0)	(1, 0, 0)

Preprint. Under review.

2.2 Uncertainty

Each state is a frame with its pixels' information.

!!!!!!!

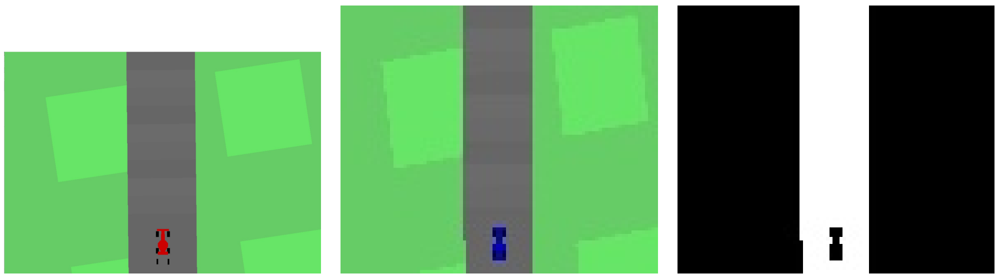
3 Related Work

3.1 Grayscale Convertion

How to perform environment recognition to assist the agent in understanding the environment? This problem can lead to many solutions and be applied to the field of autonomous driving.

We choose to do **grayscale convertion** to help the agent learn about the environment from observation. Such a solution is enough to help us complete the recognition in the box2d environment, and will not produce too much calculation to drag down the training of the model.

Figure 1: Grayscale Convertion Process



(a) Observed Image (b) RGB Image (c) Grayscale Image

3.2 Algorithm

4 Approach

4.1 Grayscale Convertion

Get a 96 $\times$ 96 RGB three-color image from the game. Multiply the y-axis by 0.85 to remove the following scores and the status bar of the vehicle to get a complete picture, i.e. reduce the number of pixel blocks to be processed. Use cv2 to use color segmentation to convert roads into white, lawns and vehicles into black.

4.2 Training

We set ***episode_num*** times to loop, and run the game once in each loop. If the game is over or the road cannot be recognized in the image or the total reward is less than the minimum value or no positive reward is obtained for 100 consecutive frames (that is, no blocks are eaten, it is purely a time deduction), the loop ends. Each time, the obtained state is converted from the color state to a grayscale image, and ***road_visibility*** is obtained from it to ensure that the road can be recognized in the image and avoid meaningless training sets. From the obtained grayscale image, the agent finds the optimal action corresponding to the image in the model, and uses the action to run the game to obtain the next state. The ***skip_frames*** method is used to introduce time continuity and provide the agent with more information for decision-making. At the same time, we process the reward obtained and set an upper limit for it. Because we don't want it to run too fast, so that it will be easily thrown away when turning. Therefore, no matter how fast the car runs, it can only get so many rewards at most, and going too fast will result in a low score, which will drive the car to drive rationally. Then convert the resulting new state from the color state to a grayscale image, and update ***road_visibility***. After that, the old and new status, action, reward, and whether it is completed are stored in the agent, and experience playback is performed. Finally, reward is added to the total.

The process would update the agent's action_value network every 5 cycles.

4.3 Test

This part is to load the model from model to agent. Same as training part, We set ***test_num*** times to loop, and run the game once in each loop. If the game ends or the total reward is less than the minimum value or no positive reward is obtained for 300 consecutive frames (that is, no tiless have been traversed, it is purely a time deduction), the loop ends. Then convert the obtained state from color state to grayscale image each time. From the obtained grayscale image, the agent finds the optimal action corresponding to the image in the model, and uses the action to run the game to obtain the next state. The ***skip_frames*** method is used to introduce time continuity and provide the agent with more information for decision-making. After that, the old and new status, action, reward, and whether it is completed are stored in the agent, and experience playback is performed. Later, reward is added to the total.

Finally, calculate the maximum, minimum, average, and variance of the reward obtained, and store the data in the ".csv" file.

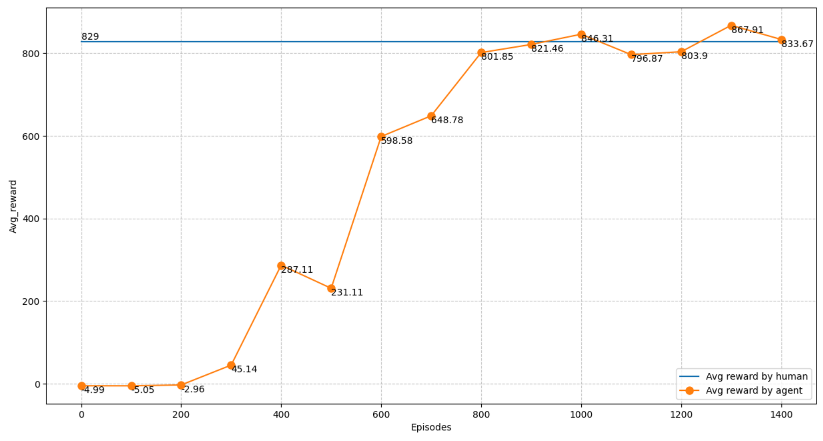


Figure 2: Average Rewards

References

[1] andywu0913[2020]OpenAI-GYM-CarRacing-DQN, `https://github.com/andywu0913/OpenAI-GYM-CarRacing-DQN`

[2] Car Racing - Gym Document, `https://www.gymlibrary.dev/environments/box2d/car_racing/`