Solving Car Racing with Deep Q-Networks

Ethan Callanan, Anshul Pattoo, John Doe, John Doe

Queen's University

Fall 2022

1 Introduction and Problem Formulation

We aim to design deep Q-Networks (DQN) that are able to solve the reinforcement learning problem posed by the Car Racing environment, a 2D autonomous vehicle which navigates a race track, from OpenAI Gym. OpenAI Gym is a notable open-source repository of reinforcement learning (RL) environments and development tools, assembled for developers to primarily apply RL agents that can solve problems posed by environments. This repository has enabled the development of a range of user-generated solutions to problems.

The objective of Car Racing is to "pilot a car through a randomly generated two-dimensional world of racetrack, grass, and boundaries, reaching the end of the track in as little time as possible" [1]. The agent is a car in acting on a race track environment in which it must navigate from its starting state and travel over all track tiles, i.e., complete a lap around the track.

The state space consists of the sequence of frames for the in-game screen, each of which is represented by an RGB (Red, Green, Blue) grid of dimensions 96 x 96 x 3, i.e., three matrices of size 96 x 96, each matrix pertaining to red, green, and blue. The action space is a discretized version of the set of tuples $A = (s, a, d) | s \in [-1, 1], a \in [0, 1], d \in [0, 1]$. The reward of a given execution of an agent through the environment is computed as follows, where t is the number of in-game frames involved in the car travelling from the starting point to the end of the track (denoted as a function $reward(t)$):

$$reward(t) = 1000 - \frac{t}{10}$$

2 State/Observation Space

The state space is a sequence of 96 x 96 x 3 matrices, each of which represents an in-game frame. Each matrix is 96 x 96 pixels, with each entry being a floating point value from 0 to 255, making the state space continuous. If we assumed that each entry in a given matrix is an integer value however, we are able to arrive at a size for our state space, $256^{3 \cdot 96^2}$. For each of the 256 possibilites of a particular entry in a red or R matrix, there are 256 additional possibilities for the corresponding entry of the green or G matrix, and 256 further possible integers for the matching pixel of the blue or B matrix. Such set of pixels needs to be considered 96 x 96 times.

3 Action Space

The action space is a discretized version of the action space defined in section 1, with elements s, a, and d, represented by degree of steering, acceleration, and breaking, respectively: this action space is discrete. The size of the actions space is 12. For an agent or object `RacingAgent`, `RacingAgent.actions` is set as follows:

```
class RacingAgent:
        def __init__(
                self,
                actions=[
                (-1, 1, 0.2), (0, 1, 0.2), (1, 1, 0.2), (-1, 1, 0),
                (0, 1, 0), (1, 1, 0), (-1, 0, 0.2), (0, 0, 0.2),
                (1, 0, 0.2), (-1, 0, 0), (0, 0, 0), (1, 0, 0)
                ]),
# ...
```

4 Reward Scheme

The reward is -0.1 every frame and $+1000/t$ for every track tile visited, where t is the total number of tiles visited in the track. For example, if you have finished in 503 frames, your reward would be 1000 - 0.1*503 = 949.7. The episode finishes once all of the track tiles are visited. The car can also travel outside of the playfield — that is, far off the track, in which case it will receive -100 reward and die.

5 Methodology and Algorithm Description

5.1 Description of the Algorithm

5.2 Ethan's Methdologies / Experiments

5.3 Anshul's Methdologies / Experiments

5.4 Max's Methdologies / Experiments

The description of the algorithm, network architecture and your implementation. What kind of methods you explored to solve the problem? If you had to for example, use any type of discretization method to discretize your observation space you can describe it here.

The selection of hyperparameters can also be discussed here. This process is important in determining how the agent will function.

6 Results

6.1 Ethan's Results

6.2 Anshul's Results

6.3 Max's Results

For the results section, you should discuss why you obtained the results you did. What if you tried a different hyperparameter, how would it have affected your results? If you are adding figures or table to visualize your results make sure to add informative captions to describe the figures and tables.

7 Discussion

In this section you can provide some information about what were the challenges you had during the implementation and how you overcame them. Finally, you can discuss what future work and improvement you may consider later. Here you can also add each group member contribution in this project.

References

[1] S. S. Pablo Aldape, "Reinforcement learning for a simple racing game."