Solving OpenAIGym's CarRacing Game using Deep Reinforcement Learning

Supriyo Sadhya
Utah State University
Logan
a02345100@usu.edu

***Abstract*—Automated cars and vehicles pose a pressing and challenging technical problem. In this work a car navigation system, in a randomly generated racetrack, was developed using deep reinforcement learning techniques such as Deep Q-learning (DQN) and the OpenAI Gym environment. A reward system and a custom CNN model as a DQN was developed and experimented with a lot of hyper-parameter tuning, model architectures, and input image processing. Car navigation is integral to many current automated systems. In this project focus is on the problem of teaching an agent – an "intelligent" car – how to navigate independently around a simulated race track. The car has to take actions (turn, accelerate, and brake certain magnitudes) to follow the grey-marked path and complete the randomly generated track, all while doing so as quickly as possible. Car navigation has a multitude of practical applications and is a challenging and pressing problem that many autonomous vehicle companies are trying to tackle. The current system though running needs further tuning in order to enhance the performance to match that of a state of the art model.**

Keywords—Automated cars, navigation system, Deep Q-learning, custom CNN model, hyper-parameter tuning, image processing, randomly generated tracks.

Introduction

Since the 1980s, developments in the field of autonomous vehicles have been widely publicized and recognized. We have also used positive reinforcement to incentivize the vehicle to stay on the desired path which is similar to what is being developed for autonomous vehicles. The objective of this study is to explore a real-world application of machine learning that is related to self-driving vehicles. We have used OpenAI's CarRacing-v2 2D autonomous vehicle environment and a Deep Q-Network in order to train our agent.

CarRacing-v2 environment state consists of 96x96 pixels, starting off with a classical RGB environment. The reward is equal to -0.1 for each frame and +1000/N for every track tile visited, where N is represented by the total number of tiles throughout the entirety of the track. To be considered a successful run, the agent must achieve a reward of 900 consistently, thus meaning that the maximum time the agent has to be on the track is 1000 frames. An episode finishes when the car visits all the tiles.

The action space if continuous has 3 actions: steering (-1 is full left, +1 is full right), gas, and breaking. If discrete has 5 actions: do nothing, steer left, steer right, gas, brake.

I. METHOD

We have used Deep Q Network to train our network. The DQN uses a CNN model to compute the Q values for each action given a state.

Reinforcement learning is more unstable when neural networks are used to represent the action-values. Training such a network requires a lot of data, but even then, it is not guaranteed to converge on the optimal value function. In fact, there are situations where the network weights can oscillate or diverge, due to the high correlation between actions and states.

In order to solve this, we have used two techniques used by the Deep Q-Network:

- **Experience Replay** - This technique is called replay buffer or experience buffer. The replay buffer contains a collection of experience tuples (S, A, R, S'). The tuples are gradually added to the buffer as we are interacting with the Environment.
- **Target Network** - To make training more stable, there is a trick, called target network, by which we keep a copy of our neural network and use it for the Q(s', a') value in the Bellman equation.

The DQN follows the Bellman equation for Q value estimation. Figure 1 below shows the loss function computation by finding the estimated Q value.

$$Q(s,\ a) = r(s,\ a) + \gamma\ \max_a\ Q(s',\ a)$$

Fig 1. Q estimation for loss computation.

The above estimated Q value in Figure 1 along with the observed Q value is used to compute the MSE loss.

The state is an RGB image which is converted to grayscale and reshaped to 84 x 84. 3 State frames are stacked to train our DQN network. We have discretized the continuous action space into 12 discrete 3d vectors.

II. RESULTS

The main purpose of these experiments is to train the agent to drive a car automatically around the generated track. In order to do so we train the network for 1500 epochs using Adam optimizer and a learning rate of 0.001. We get average scores of above 350 in the end while training and a score of around 636 score during testing our agent. Figure 2 below shows the learning curve and Figure 3 shows the score

obtained by the DQN agent. The scores in the learning curve are moving average scores.

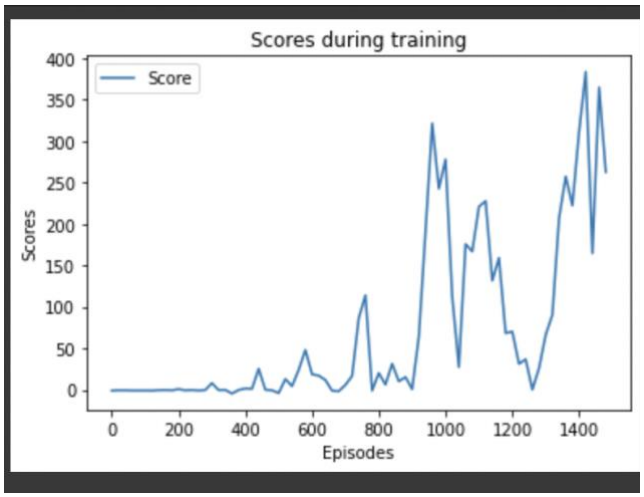

Fig 2. Training Curve of the DQN Agent.

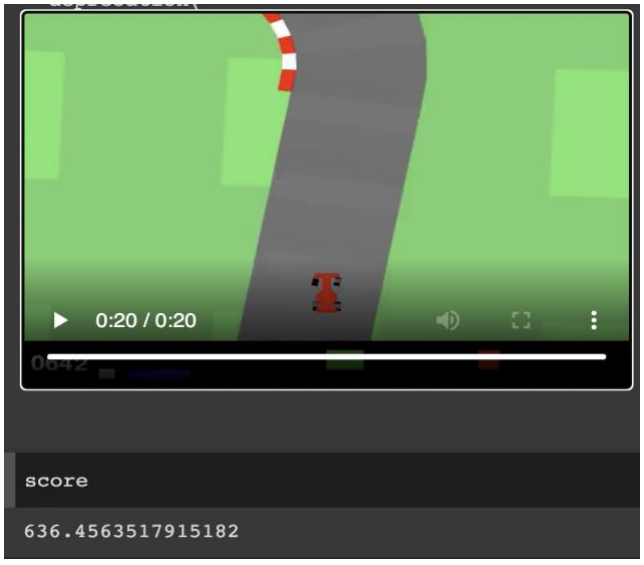

Fig 3. Score of the DQN Agent.

III. Comparison

For comparing our method, we tried to solve the problem using a Double DQN. The Double DQN algorithm solves the over-estimation problem imposed by the Bellman equation in DQN. Overestimation happens due to the max term in Figure 1. However, in our case the Double DQN perform poorly as it takes much more time start learning. Figure 4 shows the learning curve and the Figure 5 shows the score obtained by the Double DQN agent. The scores in the learning curve are moving average scores.

As it can be seen in the following figures the Double DQN does not learn that fast and may need a larger number of epochs to start learning effectively.

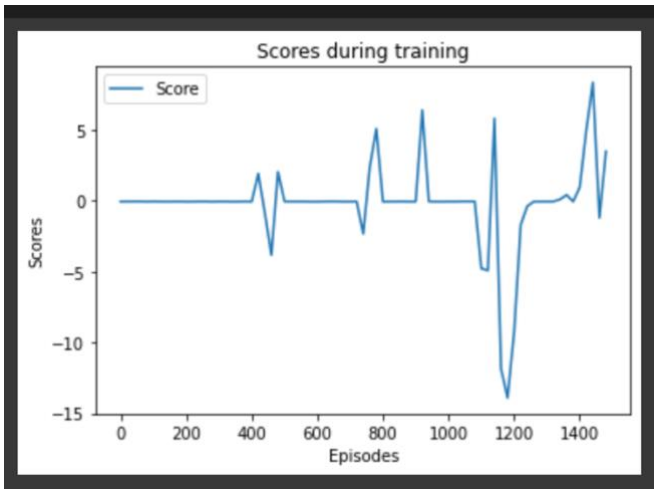

Fig 4. Training Curve of the Double DQN Agent.

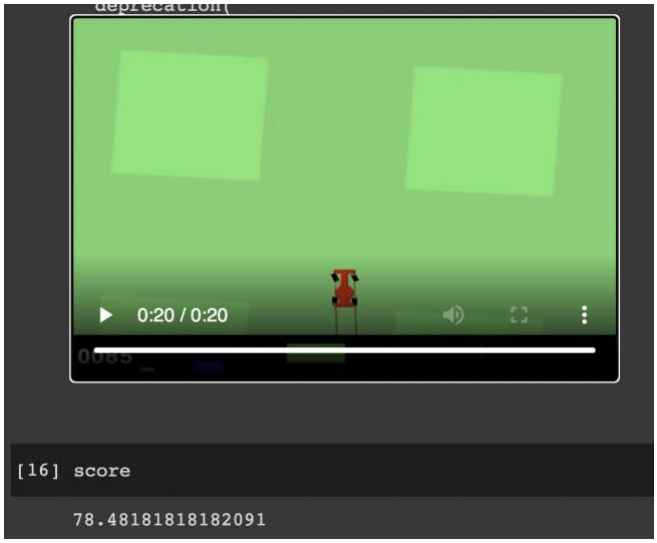

Fig 5. Score of the Double DQN Agent.

IV. Summary

Car navigation is a challenging problem which many autonomous vehicle companies are trying to tackle. There working on a similar setting is a great learning experience. The DQN I have used is very simple and straightforward as I was testing out the various techniques that I have learnt to solve a real world problem. Obviously better techniques can be used to solve this problem for better results.

V. Conclusion

We can conclude by pointing out some things we learnt in the process. Based on our observations there are certain tricks for training RL models. Hyperparameters like replay memory size, batch size, number of episodes, target network update duration, action space play key role in the training of RL models(DQN). By varying these parameters, we can adjust the performance and learning of our agent. Also, the agent learning is sensitive to the amount of exploration and exploration decay set by us.

The DQN learns the optimal policy much faster in our case and proves to be the better model. For the Double DQN a lot more training is required for the model to start performing effectively.

References

[1] Aldape, P., Sowell, S.: Reinforcement learning for a simple racing game

[2] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529 (2015)

[4] Raffin, A., Sokolkov, R.: Learning to drive smoothly in minutes. https://github.com/araffin/ learning-to-drive-in-5-minutes/ (2019)