REINFORCEMENT LEARNING
(CS3316)

Assignment Report

Assignment 4

Mountain Car
Deep Q Network

Name: Kezhi Li ID: 520021911013
Date: 13 April 2023

Contents

1 Introduction

The mission of this assignment is solving the mountain car problem in the Gymnasium environment developed by OpenAI. A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.

The state of the car is decided by the position of the car along the horizontal axis and the velocity. At the beginning of the episode, the car starts from the bottom of the hills (valley). The episode ends when the car reaches the goal or after 200 steps. The action space consists of 3 discrete actions: {push left, do nothing, push right}. A negative reward of -1 is applied at each timestep. The objective of the car is to reach the top of the hill on the right as early as possible, because at each timestep it will be rewarded negatively.

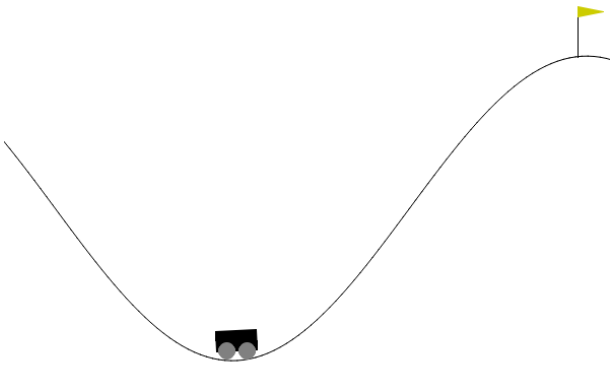

Figure 1: MountainCar

2 Experiment

2.1 DQN and DDQN

Since the state of this problem is continuous, the traditional reinforcement learning algorithms like Sarsa or Q-learning appear to be inappropriate for this question. In classical Q-learning methods, the action value function Q is intractable with the increased size of the state space and action space. On the other side, the deep Q network (DQN) leverage the benefits of Q-learning and neural network. It uses a deep network to estimate the state value with the state as input. Specifically, the output of the deep network should be compared to the output by TD method of the target deep network, and the loss can be calculated to update the network. Moreover, the experience replay is another trick for DQN to decrease the strong correlation between data, and efficiently learn from each transition pair. The detailed pseudo code of DQN is shown Fig. 2.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Figure 2: Deap Q-learning with experience replay

To solve the problem that the DQN always overestimates the state value, the Double Deep Q Network (DDQN), which uses two networks to calculate the target state value, is devised. Specifically, the DDQN uses the action network to choose the action of the next state, and calculate the state value through the target network. In this assignment, I will compare the difference of DQN and DDQN in solving the mountain car problem.

2.2 Experimental Setup

Parameters modifying is always a tough work in deep learning. To reasonably compare the difference of the two methods, I use the same experimental setups, except the difference of the algorithms that calculate the target state value. I summarized all the influential parameters in Tab. 1.

One more important setup is the frame of the deep Q network. Since the state is only a two dimensional vector, there is no need to use the convolutional layer to extract the features. Thus, I only connect the input and output with two fully connected layers. Figure 2.2 shows the detailed framework. Moreover, I choose the smooth L1 loss function during training, instead of the L2 loss shown in the pseudo codes.

Table 1: Parameters

PARAMETERS	VALUES
Learning Rate α	0.0005
Discount Factor γ	0.98
Exploration Rate ϵ	$1 \rightarrow 0.05$, decay exponentially
Memory Length	2000
Sample Batch Size	64
Updating Interval	10
Total Episodes	500

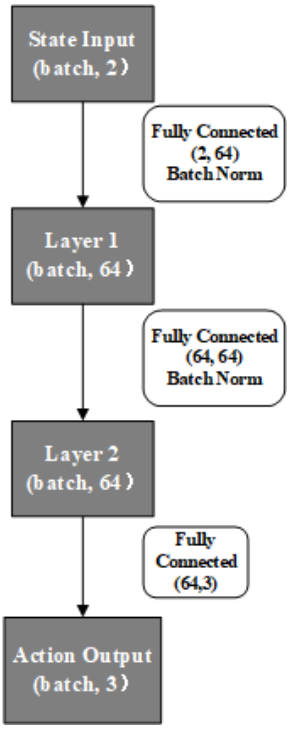

Figure 3: Deep Q network framwork

2.3 Results

Among all the 500 episodes, I choose the models with the highest performance of DQN and DDQN. The comparason is shown in Tab. 2. The average score is calculated by running the model in the environment for 50 times. It can be shown that DDQN used less episodes to find a generally good model. Besides, the DQN find a better model surprisingly. However, due to the fortuity of the model, I think it doesn't mean the DQN is better than DQN. The two different models are attached in the compressed file. If you are interested in playing them, you can run the "test_visual()" function

with the name of the file.

Table 2: Comparason between DQN and DDQN best models

Model	**Episode**	**AvgScore**
DQN	460	-128.9
DDQN	230	-147.5

To have a better insight into the deep Q network of the two models, I drew the heatmap of the decisions of action under different states. Figure 4 and Figure 5 are the heatmaps of DQN model and DDQN model respectively. It should be specified that the value '0' with red color indicates that the car should accelerate left. '2' with blue color indicates the car should accelerate right, and '1' with white color indicates the car should stop. The horizontal axis of the heatmap is the position of the car, which is from -1.2 to 0.6. The perpendicular axis of the heatmat is the velocity of the car, which is from 0.07 to -0.07. The distributions of these two models are similar. When the velocity is high enough, the car is more likely to keey accelerating to reach the goal place. When the car does not have enough velocity, the car tends to accelerate left to accumulate momentum. The difference is that when the position is very close to the goal but the velocity is small, the DDQN is more "racial" that it keeps accelerating. However, the DQN appears to be "conservative" that it turns to accelerate left to obtain more momentum for the next try.

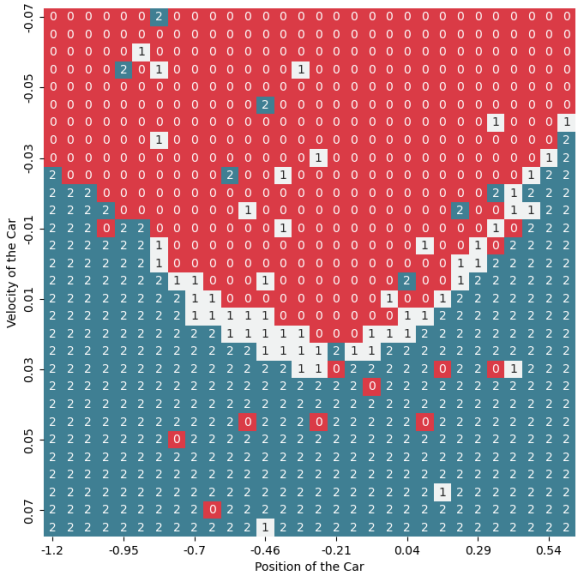

Figure 4: Action heatmap of DQN model

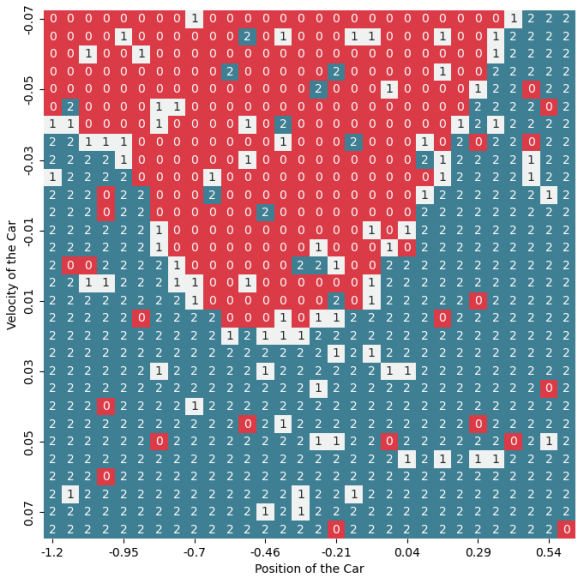

Figure 5: Action heatmap of DDQN model

3 Conclusion

Both DQN and DDQN can well solved the mountain car problem. With a high exploration rate at the beginning, the model can explore multiple choices of actions, and finally find a way to the mountain. However, this problem is still simple, since the action space only contains three discrete values, and the state space is only a two dimensional vector. I cannot imagine what will be cost to train a model like Alphago with such large action space and state space.