Assignment 4

Yuxiang Lu 518021911194

1 Introduction

Deep Q Network combines the advantage of Q-learning and neural network. In this assignment, the DQN algorithm and two improved algorithms, Double DQN and Dueling DQN, are implemented to solve the MountainCar problem, a classical RL control scenario.

2 MountainCar

MountainCar is a classic control problem in RL, which is provided in OPENAI gym as `MountainCar-v0`. The goal is to get an under powered car to the top of a hill, shown in Figure 1.

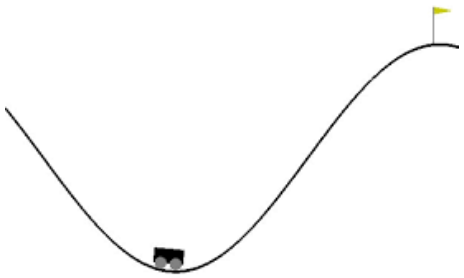


Figure 1: MountainCar Environment

Observation The state of the car has two parameters representing position and velocity, and their details are shown in Table 1. The starting state is a random position from -0.6 to -0.4 with no velocity. The goal state is reached when position equals 0.5.

Num	Meaning	Range
0	position	[-1.2, 0.6]
1	velocity	[-0.07, 0.07]

Table 1: State of MountainCar

Action As shown in Table 2, MountainCar has three actions, going left or right, or no push.

Reward The reward is -1 for each time step, until the goal position is reached.

Num	Action
0	push left
1	no push
2	push right

Table 2: Action of MountainCar

Termination The episode ends when 0.5 position is reached, or if 200 iterations are reached.

3 Deep Q Network (DQN)

In Q-learning, we use a lookup table to represent value function. However, for large MDPs, there are too many states and actions to store in memory, which makes it hard to learn the value of each state and action. The solution is to estimate value function with function approximation. DQN introduces deep learning to Q-learning as well as two tricks, fixed target network and experience replay. Existing experiments prove that DQN performs well in many RL problems. The algorithm is elaborated in Figure 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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Figure 2: DQN Algorithm

4 Double DQN

In DQN method, Q value is usually overestimated, as the future maximum approximated value is evaluated using the same Q function, which may slow the learning. Double DQN is proposed to solve this issue by using another function Q' to select the next action. Specifically, the target in DQN is

$$r_t + \gamma \max_a Q(s_{t+1}, a)$$

while the target in Double DQN is

$$r_t + \gamma Q'(S_{t+1}, arg\max_a Q(s_{t+1}, a))$$

Therefore, Q' would give the right value to the overestimated action a.

5 Dueling DQN

Dueling DQN splits the Q values into two parts, the value function $V(s)$ and the advantage function $A(s, a)$. The value function $V(s)$ tells us how much reward we will collect from state s. And the advantage function $A(s, a)$ tells us how much better one action is compared to the other actions. A clear comparison of DQN and Dueling DQN is shown is Figure 3.

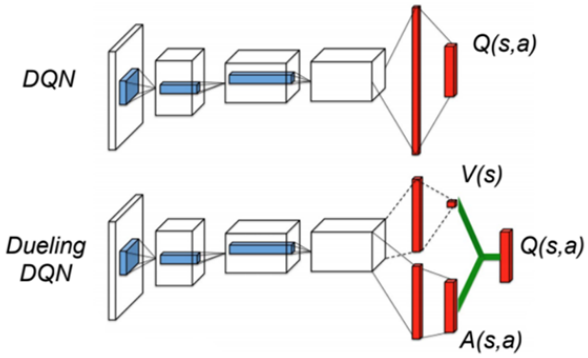


Figure 3: DQN vs Dueling DQN

Dueling DQN splits the last layer of neural network into two parts, estimating $V(s)$ and $A(s, a)$ respectively, and at the end it combines both parts into a single output, which estimates the Q values with the equation follows,

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|A|}\sum_{a'} A(s, a')$$

6 Experiment and Discussion

Hyper-parameters

Since there are many hyper-parameters in DQN algorithm, the first step of the experiment is to set these hyper-parameters. Considering the experience replay part, the size of the replay buffer is 5000, and at the beginning of training, it will sample until there are more than 1000 tuples in the buffer.

Then we move to the neural network part, which is implemented by Pytorch. Generally speaking, when you input a state s to the network, it will output the evaluation of $Q(s, a)$ for every action a. For DQN and Double DQN, it is a simple network, which contains two fully connected layers. The first layer is 2×64, where 2 is the dimension of the input variable, state. The second layer is 64×3, where 3 is the number of possible choices in action. For Dueling DQN, the second layer

is separated to a 64×1 one and a 64×3 one, and then the output is calculated by two parts. The target network uses the same structure as the model, while it is updated every 10 episodes.

For the training part, the learning rate is 0.001, the optimizer is Adam, and the batch size is 64.

Since the agent uses ϵ-greedy policy in action selection, we let ϵ start from 1 and decrease by a rate of 0.995 until reaching 0.01. This enables agent to explore more situations in earlier stage, and later makes the learning stable. The discount rate γ is set as 0.9 to make learning converge faster.

Results

The three algorithms are trained with 2000 episodes, and the sum of rewards during each episode are shown in Figure 4, note that the data are averaged every 50 episodes to reduce randomness.

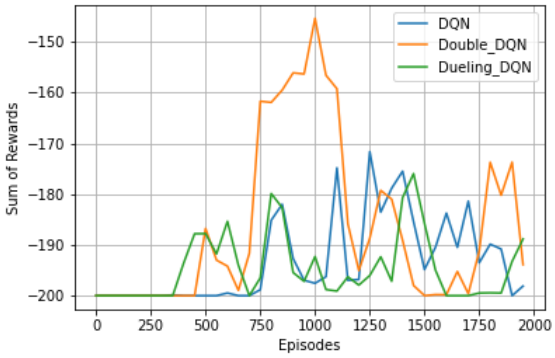


Figure 4: Training Results

We can see that according to the rewards, Double DQN is higher than DQN and Dueling DQN. However, all these methods fluctuate, and sometimes drop to the lower limit. It shows that the learning methods are not stable and do not converge to an optimal value. We can also find that the rewards of DQN start to rise after about 750 episodes, while Double DQN and Dueling DQN go up after 300 to 400 episodes, which means two improved algorithm learn faster than DQN.

Then the trained networks are tested by playing episodes with greedy action selection (just let $\epsilon = 0$). From Figure 5, we can see that only Dueling DQN can reach the goal with the rewards sum around -160, while the other two are terminated when reaching 200 episodes.

Modified Reward

We can conclude that the neural network is not trained well in the experiment above, but how to solve this problem? A possible method is to modify the reward got in each step. With reference to some materials[1], I change the reward as follows,

$$r = 100 * ((0.0025 \sin(3p') + 0.5v'^2) - (0.0025 \sin(3p) + 0.5v^2))$$

[1]https://towardsdatascience.com/open-ai-gym-classic-control-problems-rl-dqn-reward-functions-16a1bc2b007

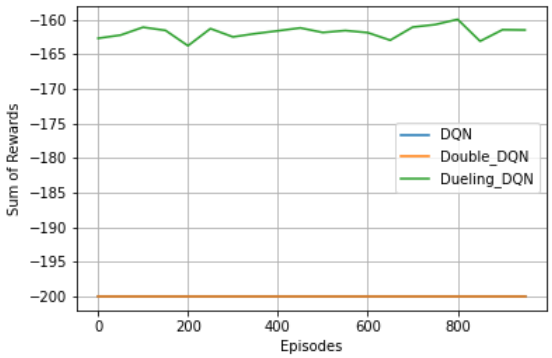
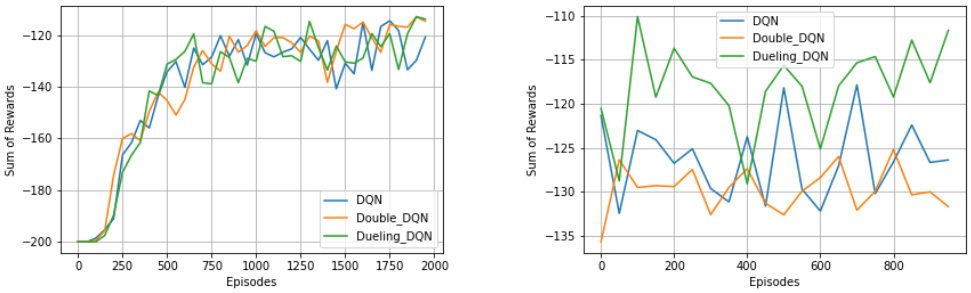


Figure 5: Test Results

Where p, v represents the position and velocity in the current state and p', v' stands for the observation in next state.

Then the three algorithms are trained with the modified reward function, and the results are shown in Figure 6. With the modified reward, the training process of three methods are improved a lot, as they can converge to an optimal value around -120 to -140. The training is also accelerated, for the rewards go up before 200 episodes and almost converge at 1000 episodes.

For the test part, Dueling DQN outperforms DQN and Double DQN again, while the latter two get similar results around -125 to -130. Most rewards of Dueling DQN are above -120 and the highest value can reach -110.



(a) Training Reward

(b) Test Rewards

Figure 6: Results with Modified Reward

Conclusion

From the experiments on three DQN algorithms, we can conclude that Dueling DQN is a better method than DQN and Double DQN. When using the original reward, only Dueling DQN can push the car to mountain top in the tests. And when using the modified reward, Dueling DQN can get more rewards than others in tests, which means it is more efficient. We can also learn that reward function is very important in DQN and RL, a better reward function can make learning process converge faster and conduct more accurate values.