A4

Requirements

Programming language: python3

You should compare the performance of DQN and one kind of improved DQN and test them in a classical RL control environment–MountainCar. OPENAI gym provides this environment, which is implemented with python.

Introduction

In former assignments, we use state-action pair to store the values to the list. However, as the task becomes more and more complex, using deep neural network is a better choice to fit values better in high-dimensional state spaces. Such as this assignment, we need to work on a Montain-Car problem, which has complicated states and actions.

Simply using DQN has its drawback. For example, Q-value is usually over-estimated. Therefore, we need advanced-DQN to optimize our algorithm. In this lab, I use Double DQN and Dueling DQN for comparison.

My Implementation

env

First of all, we need to introduce the environment.

```
import gym
env = gym.make('MountainCar-v0')
env.reset()
for i in range(1000):
    env.render()
    ...
    env.step(env.action_space.sample()) # take a random action
env.close()
```

I use pytorch to implement the neural network.

DQN

For DQN, first to build the network

```
import torch
import torch.nn as nn

class Net(nn.Module):
    def __init__(self, ):
        super(Net, self).__init__()
        self.fc = nn.Sequential(
```

```
            nn.Linear(n_states, 24),
            nn.ReLU(),
            nn.Linear(24, 24),
            nn.ReLU(),
            nn.Linear(24, n_actions)
        )
        # optimizer and loss_func
        self.opt = torch.optim.Adam(self.parameters(), lr=LR)
        self.mls = nn.MSELoss()

    # Given state, return action value
    def forward(self, s):
        return self.fc(s)
```

In DQN, first build two networks, first a eval_net, then a target_net. And then initialize a memory set to store states, actions and rewards.

I have the function to choose action by the given state and storing the parameters to the memory set and the function to learn. In the learning function, I use a delayed update to update the target network.

The code can be viewed in my project.

DDQN

As for me, DDQN and DQN is almost the same. I use the same Net class. The only difference is in the learing process. DDQN use two functions Q and Q' while updating Q

Dueling DQN

Dueling DQN changes the network structure. Now, `Q(s,a)=V(s)+A(s,a)`

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.features = nn.Sequential(
            nn.Linear(n_states, 16),
            nn.ReLU(),
            nn.Linear(16, 16),
            nn.ReLU(),
        )
        self.adv = nn.Linear(16, n_actions, bias=True)
        self.val = nn.Linear(16, 1, bias=True)
        # optimizer and loss_func
        self.opt = torch.optim.Adam(self.parameters(), lr=LR)
        self.mls = nn.MSELoss()

    def forward(self, x):
        x = self.features(x)
        adv = self.adv(x)
        val = self.val(x)
```

```
        x = val + adv - adv.mean()
        return x
```

Reward Function

For a better result and a better convergence, I customize the reward function.

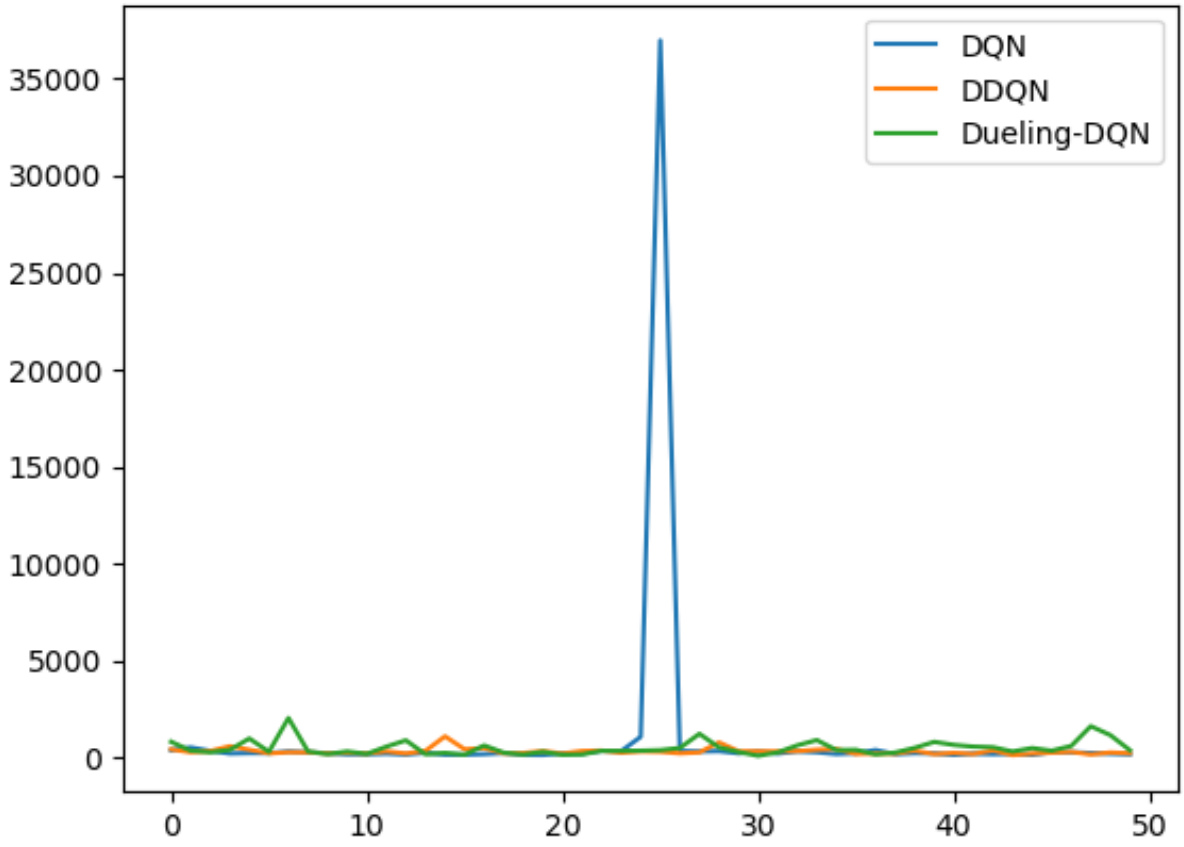
```
""" Rewrite the reward function, start from -0.5, reach the end
    point at 0.5 . Because we hope the car can goes to the right side,
    if goes lefter than start point, gives a negative reward.
    The much closer it gets to the end point, it will receive a larger reward.
"""
if -0.4 < s_[0] < 0.5:
  r = 10 * (s_[0] + 0.4) ** 3
elif s_[0] >= 0.5:
  r = 100
elif s_[0] <= -0.4:
  r = -0.1
```

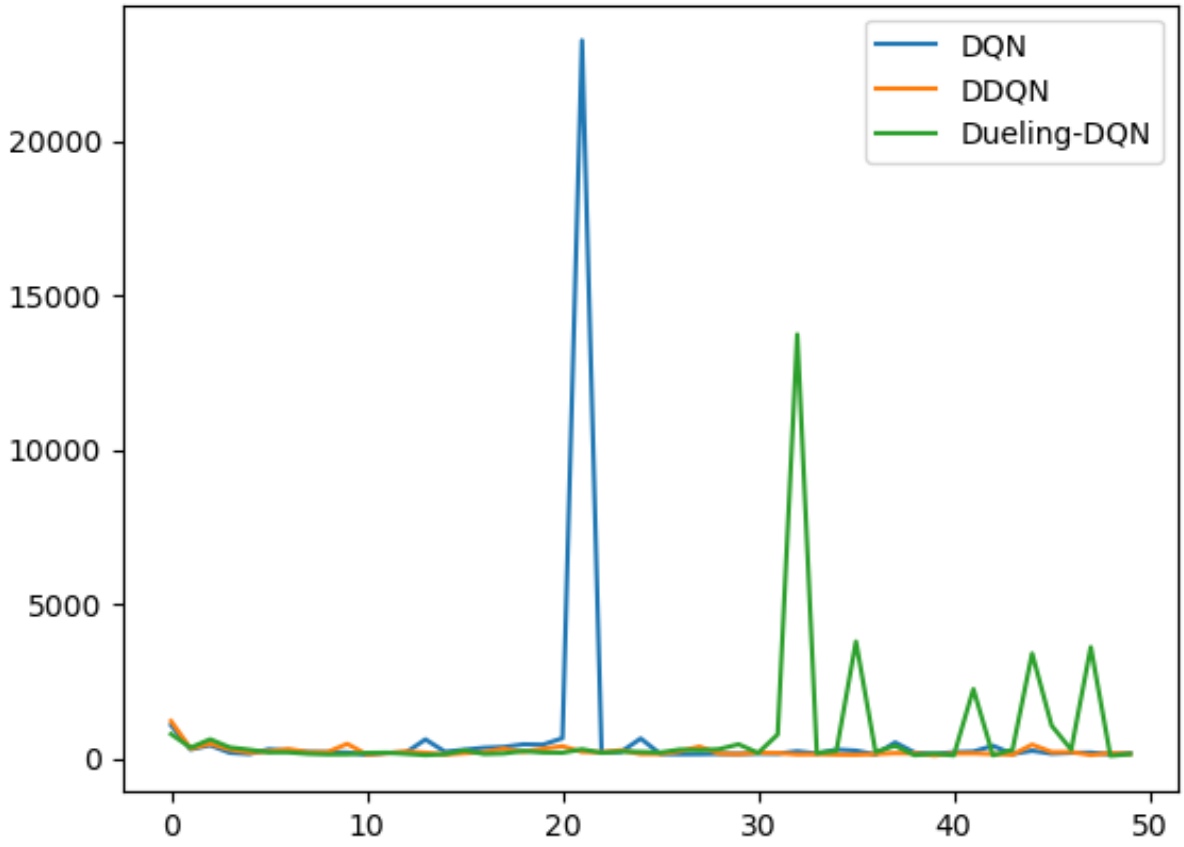
Improvements

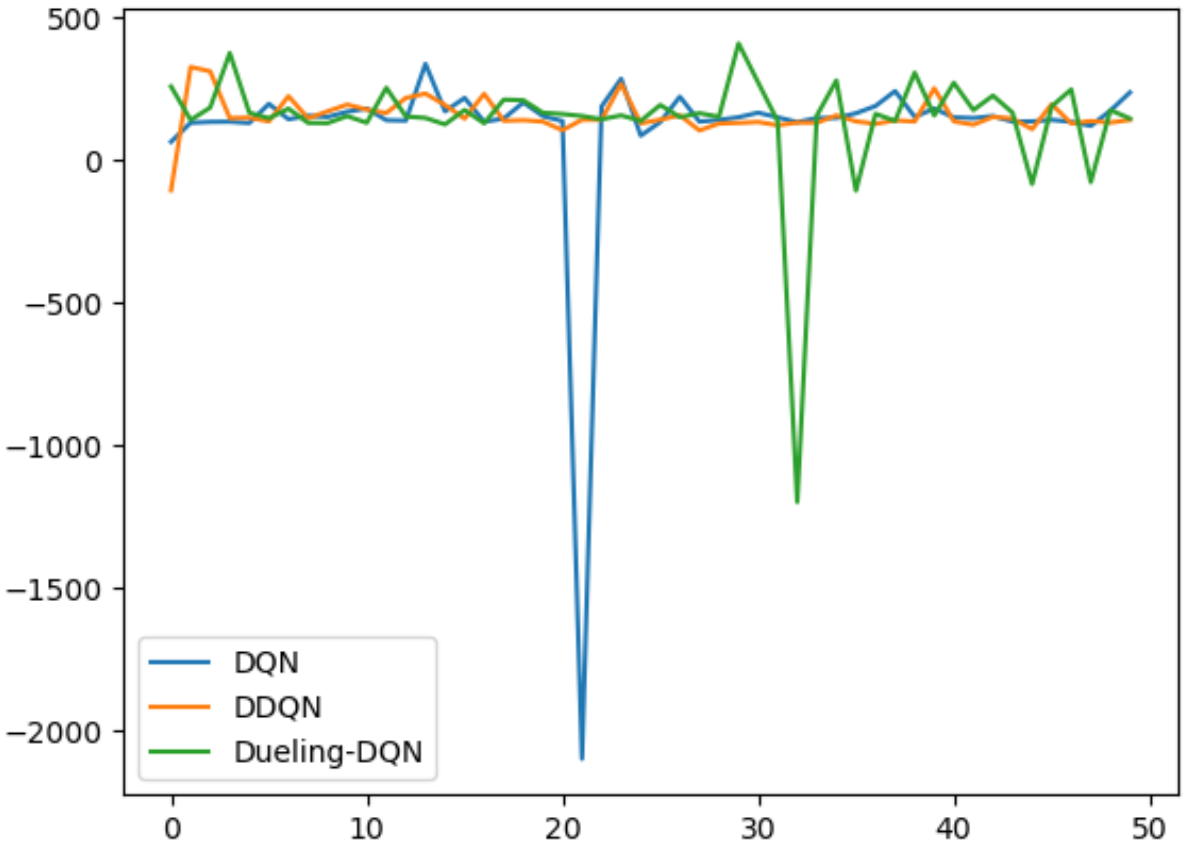
After each task, I store the current memory set to a local file. As I start the next task, the memory set will first read the information other than creating an empty set. This makes the results better, it can quickly reaches to the end point even at the first several episodes.

Result

First two pictures are **episode_num-steps** maps, and the third one is **episode_num-rewards** map.


DQN
DDQN
Dueling-DQN
35000
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50


DQN
DDQN
Dueling-DQN
20000
15000
10000
5000
0
0
10
20
30
40
50



We can see, at most times the car takes few steps(200 times for average) to reach the terminal point. The result of Double DQN is obviously more stable than DQN. Dueling-DQN also performs better than DQN.

Summary&Thinking

This lab uses pytorch and gym which I haven't use in the previous labs, which makes me a little confused when implementing it after learning the relevant knowledges. The process of learning new knowledges is also rewarding. As I do the lab, the result of DQN lets me doubt whether I have the right result. After drawing the maps, I find that I may get the right solution. Because of the time, I don't implement other advanced-DQN such as NoisyNet DQN. I will try it in the future and I wish that in the later labs I can also learn a lot.