**Final Project Report**

**519021911325 刘畅**

1. **Introduction**

In this project, we are required to implement two kinds of model-free reinforcement learning methods, which are value-based method and policy-based method, to solve environment of Atari and MuJuCo.

Value-based methods try to get the most suitable action-value function or state-value function to better value each action or state. And thus use those functions to choose the next action each time. Among them, Deep Q-learning (DQN) is representative for introducing deep neural network to replace Q-table in traditional Q-learning and achieves great performance.

In comparison, policy-based methods try to learn the policy directly, and updating policy function using policy gradient. Among them, Actor-Critic method is representative. And to solve the drawback of sampling inefficiency, off-policy methods like DDPG are introduced.

Considering the weak computing power of my laptop, even a easy model needs to be trained for a long time. So I choose simpler environments for training to obtain relatively ideal results. In this project, DQN is implemented on PongNoFrameSkip-v4, and PPO on Ant-v2.

1. **Methods**
   1. **DQN**

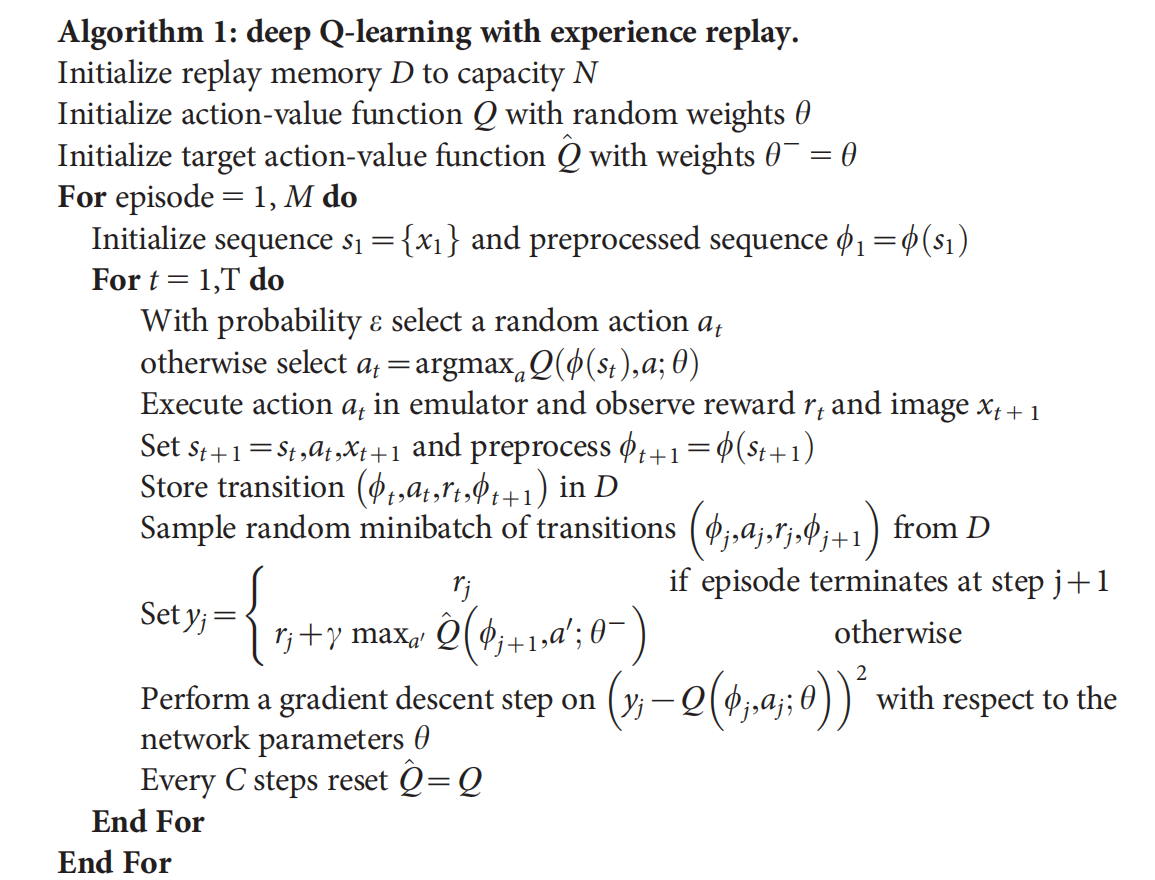
DQN is the improved version of Q-Learning, which maintains a Q-table to record the value of state-value pair. But it has a severe problem. When facing complicated environment like Atari games, which have too many actions or states, the Q-table has to be huge, which is not sensible. Thus, DQN is introduced. In comparison with Q-learning, DQN makes two main improvements.

First, DQN uses experience replay to solve the correlation and non-static distribution problem. Experience replay uses a random sample of prior actions instead of the most recent action. Each step the agent generates a sample like (state, action, next\_action, reward), and it will be stored and randomly chosen to train the network. Experience replay helps to accelerate the backup of rewards and remove the correlation of samples from the environment.

Second, DQN uses a Q-network to fit the value function and a target network to decide the action. And method of fixing target network is applied. Target network with older network parameters is used when estimating the Q-value for the next state in an experience. The target network updates every N steps. Such a target network fixed the policy when Q-network updates which leads to stability.

The detail of DQN algorithm is shown below in Figure. 1.

For the game of PongNoFrameSkip-v4, as each state is in form of a picture, thus convolutional neural network(CNN) needs to be applied, which contains convolutional layers, batch normalization layers and fully connected layers. It takes a transformed picture as input, and output the value of each action.



**Figure 1. DQN Algorithm**

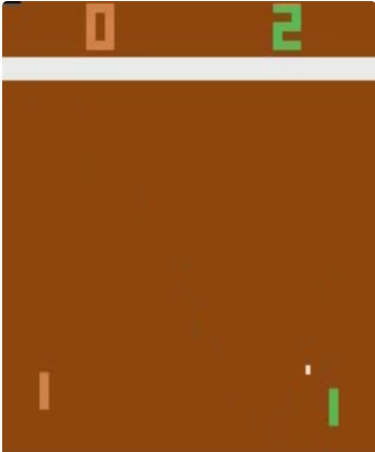
* 1. PPO
  2. DDPG

1. **Experiments**
   1. **Atari: PongNoFrameSkip-v4**
      1. **Environment**

Atari contains many classic games. I choose the game of PongNoFrameSkip-v4 to apply DQN.

The observation of Atari environment is the screen of the game of size 210\*160\*3. And some preprocessing need to be done before training. First, as RGB value is in range [0, 255], I first convert it into [0, 1] by dividing 255. Then, to better match the input form of network, the input would be transformed to 3\*210\*160, making input dimension 3.

Some improvements are applied during training. If the environment is generated directly by **env=gym.make(‘PongNoFrameSkip-v4’)**, I noticed that the training would be extremely slow (about 160 episodes in 8 hours) and achieve poor performance. So methods of NoopResetEnv, MaxAndSkipEnv and TimeLimit are added. NoopResetEnv skip some of the initial frames, making the initial state more random. MaxAndSkipEnv uses the same action in neighboring states, and returning once in every few frames to accelerate training. TimeLimit would limit the maximum actions made, returning done=True when achieving the limit. By applying these methods, not only the performance is much better, the training time is greatly reduced ( 134 episodes in 1 hour).



**Figure 4. PongNoFrameSkip-v4**

* + 1. **Implementation**

The convolutional network contains eight layers, three convolutional layers, three batch normalization layers and two fully-connected layers. Detailed setting is shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Layer** | **Input dim** | **Output dim** | **Kernel Size** | **Stride** | **Activation** |
| Conv2d | in\_channels | 32 | 8\*8 | 4\*4 | relu |
| BatchNorm2d | 32 | 32 | / | / | / |
| Conv2d | 32 | 64 | 4\*4 | 2\*2 | relu |
| BatchNorm2d | 64 | 64 | / | / | / |
| Conv2d | 64 | 64 | 3\*3 | 1\*1 | relu |
| BatchNorm2d | 64 | 64 | / | / | / |
| Linear | 14\*11\*64 | 512 | / | / | relu |
| Linear | 512 | n\_actions | / | / | none |

**Table 1. QNet Structure**

For ε-greedy action selection, ε would decay with steps. At first ε=1, and in each action selection, ε is multiplied by a decay rate 0.995. In this way, the first steps has more exploration chance, and latter steps is keeps doing best steps.

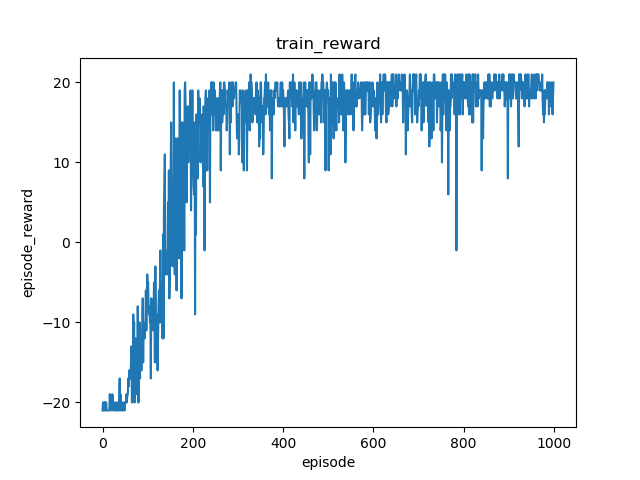
Other hyper-parameters are set as follows:

|  |  |
| --- | --- |
| **Hyper-parameters** | **Value** |
| batch size | 32 |
| γ | 0.99 |
| ε\_start | 1 |
| ε\_end | 0.02 |
| target update stride | 1000 |
| policy update stride | 2 |
| learning rate | 0.001 |
| n\_episode | 1000 |
| memory size | 100000 |

**Table 2. DQN hyper parameters**

* + 1. **Result**

I trained DQN on PongNoFrameSkip-v4 provided in gym for 1000 episodes. The result is shown in Figure.5. It could be seen that training is quite fast, after about 200 episodes, the reward significantly improved. It means the DQN algorithm quickly learns a good policy. And it also shows that the improved environment is much better. In comparison, after 200 episodes in the original environment, the reward is still around -20.



**Figure 5. DQN result on Pong**

For evaluation of the trained model, I use the model to play a rendered game to see what’s the outcome. For the first few points, my agent would likely to lose, but after about 3 points, it seems to find a trick, and keeps winning using the same strategy. The average score is 17.42, which is a bit lower than my expectation.

* 1. Mujoco: Ant-v2

1. **Conclusion**

In this project, I implement both value-based and policy-based methods on Atari and MuJoCo environments. In specific, DQN on PongNoFrameSkip-v4, PPO and DDPG on Ant-v2.

For value-based RL algorithm, DQN does perform well in Atari game, but the training time is quite costly, 1000 episodes would take hours. It also suffers from sample unpredictability. If the agent couldn’t get a positive reward, which means hitting the ball and win a point in the game, it wouldn’t achieve a good performance.

For policy-based RL algorithms, PPO is faster but not having good performance, while DDPG is slower but having great performance.

In general, the environment and parameters could greatly influence the performance of algorithms, it may not be fair to judge which algorithm is better merely by its performance in a specific environment. So, the selection of algorithm is a key factor.