**Assignment 5**

1. **Introduction**

In the previous assignment, we have applied Deep Q Network (DQN) to solve mountain car problem, which has continuous state space and discrete action space. But with continuous action space, DQN couldn’t do anything about it because it needs to find the exact action that maximize the action-value function.

One solution is Actor-Critic method, which is introduced to solve this problem with continuous action space. Compared with DQN that learns the action-value function, AC learns the gradient of the policy, thus could solve high dimensional and continuous action space.

Another solution is based on both DQN and AC, using experience replay and Actor-Critic structure. Each time, the agent directly output action of highest possibility instead of possibility of each action, thus solving continuous action problem.

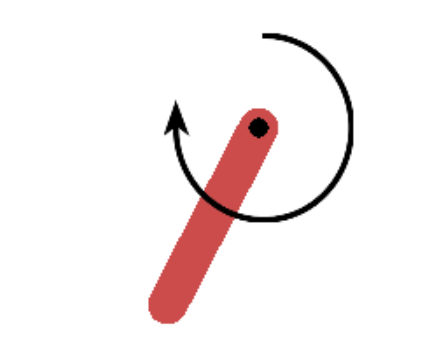
In this report, I will apply two improved method mentioned above, Asynchronous Advantage Actor Critic (A3C) and Deep Deterministic Policy Gradient (DDPG) in the game of Pendulum.

1. **Task: Pendulum**

Above all, one thing to notice is that Pendulum-v0 has been out of date, using Pendulum-v0 in gym would cause errors, so use Pendulum-v1 instead, which is basically the same.

As shown in fig. 1, the goal of the pendulum-v1 problem is to swing a pendulum upright to makes it stay upwards. The pendulum starts in a random position every time, and after 200 steps it would end. There are three observation inputs for this environment, sin and cos to represent the angle of the pendulum and its angular velocity. The initial state is .

The action is joint effort, which ranges in [−2, 2]. With state and action, the precision equation of reward is . In general, the more straight upward and stable it is, the better reward it would get.



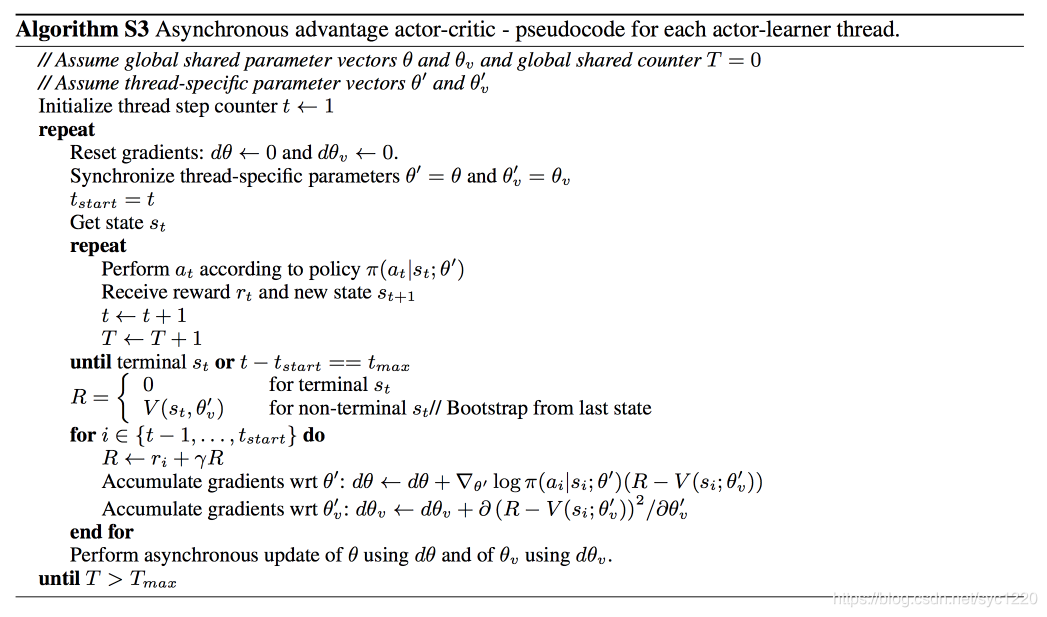
**Figure 1. Pendulum-v1**

In this experiment we use gym to provides this environment, just input current action, gym would output next state, reward and whether it’s done.

1. **Asynchronous advantage actor-critic (A3C)**

**3.1 Introduction**

As the sampling process of A2C is quite costly, so A3C is proposed to save time in an asynchronous way. It will create multiple parallel environment, and multiple worker agents which has the copy of main agent could update parameters in the main structure at the same time. The agents in parallel do not interfere with each other, while the main structure parameters update discontinuity interference when copy structure updates, therefore the correlation of update is reduced then the convergence is improved. The algorithm details of A3C is shown below.



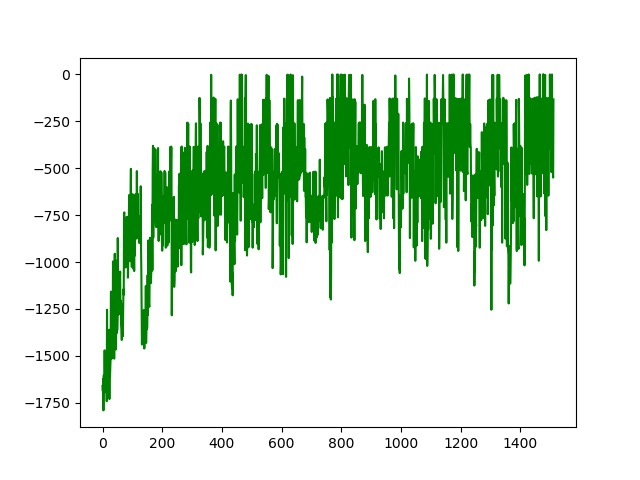
**Figure 2. A3C Algorithm**

**3.2 Implementation**

We train 1500 episodes to observe the performance of A3C and for each episode the steps is no larger than 200. The number of workers is 12. Then we set the learning rate of actor to 0.0005 while the learning rate for critic to 0.001. Other settings are discount factor γ=0.99. And every 5 episodes in each worker agent, the worker would update the parameters in the main agent, and copy the main agent.

**3.3 Result**

As we can see from fig.3, A3C successfully converged, and the moving reward will increase over training at the beginning. After about 800 episodes, the moving reward starts to be stable.



**Figure 3. Record of total reward using A3C**

1. **Deep Deterministic Policy Gradient (DDPG)**

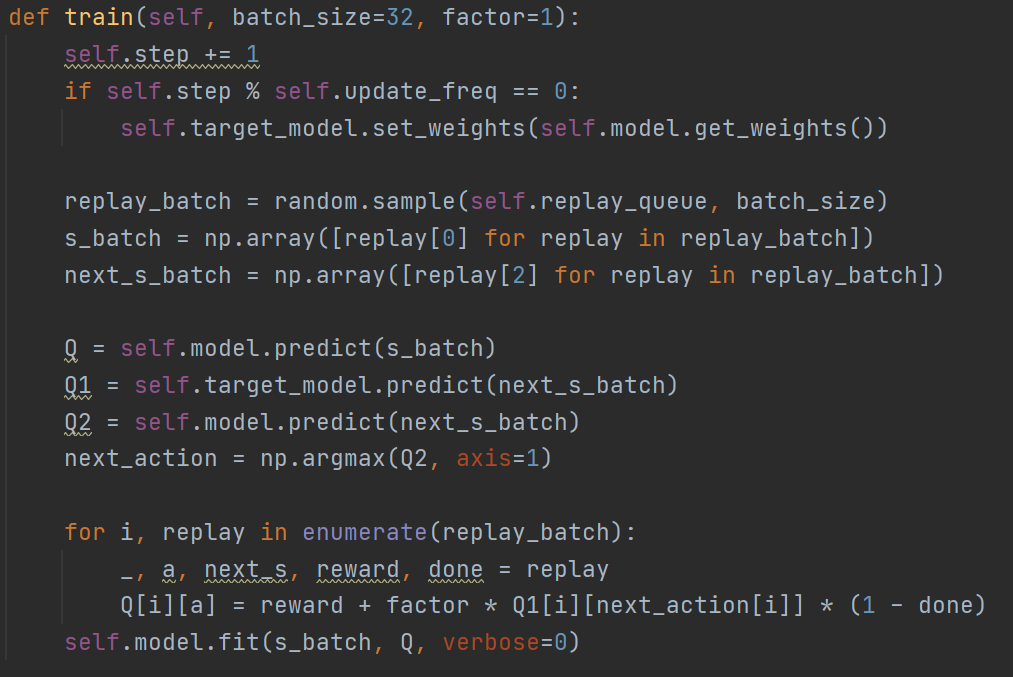
**4.1 Introduction**

Although DQN achieve quite a good outcome, there is a problem. Because it always choose the highest Q-value to update neutral network, it tends to overestimate Q-Values. To solve this, Double DQN was invented. Double DQN introduces another network, and it reduces Q-Value overestimation by splinting max operator of DQN into action selection using original network and action evaluation using another network.

**4.2 Implementation**

As the natural DQN algorithm also uses an additional network for fixing Q-value for a while, so no additional networks are actually added compared with DQN. Weights of the second network are replaced with the weights of the target network for the evaluation of the policy. Target Network is still updated every N episodes, by copying parameters from Q-Network.

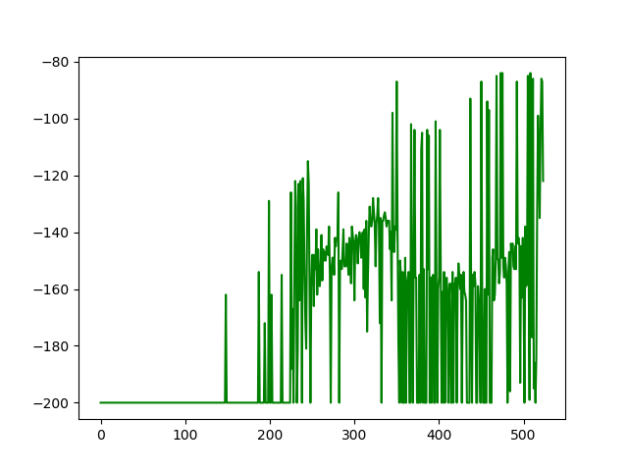
And to better converge, when calculating target Q value, if the next state is the terminal state(done = True), then the Q value would be considerably higher, leading the car this way.



**Figure 6. Training of DDQN**

**4.3 Result**

Here is the record of total rewards during training. Compared to natural DQN, it could reach terminal state more frequently, and after about 500 episodes, it reaches a rather steady state.



**Figure 7. Record of total reward in Double DQN**

In my testing, it takes 117 steps to the top, and I have record a video(Double DQN.mp4) of it.