**Assignment 5**

519021911325 刘畅

51902 杨雨欣

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.

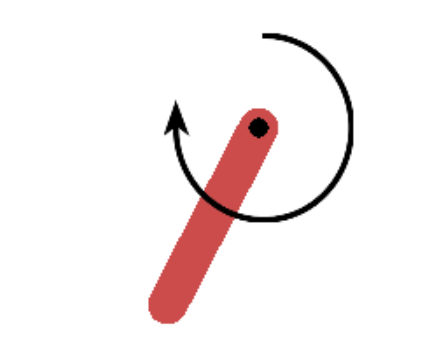
In this report, I will apply two improved method mentioned above, Asynchronous Advantage Actor Critic (A3C) and Deep Deterministic Policy Gradient (DDPG) in the game of Pendulum.

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Above all, one thing to notice is that Pendulum-v0 has been out of date, using Pendulum-v0 in gym would cause errors, so use Pendulum-v1 instead, which is basically the same.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

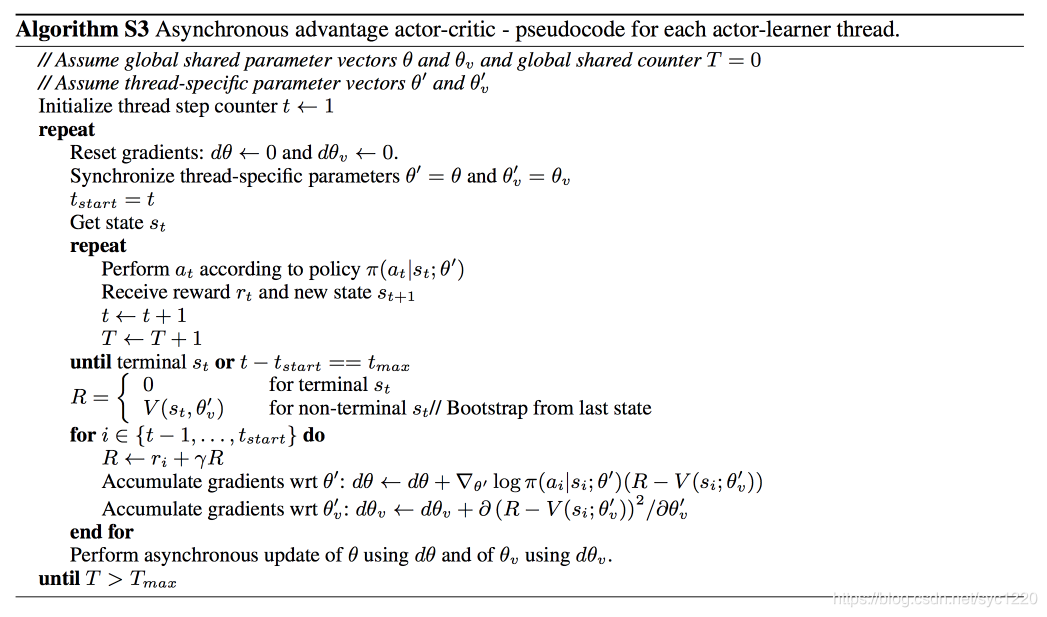


**Figure 1. Pendulum-v1**

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.

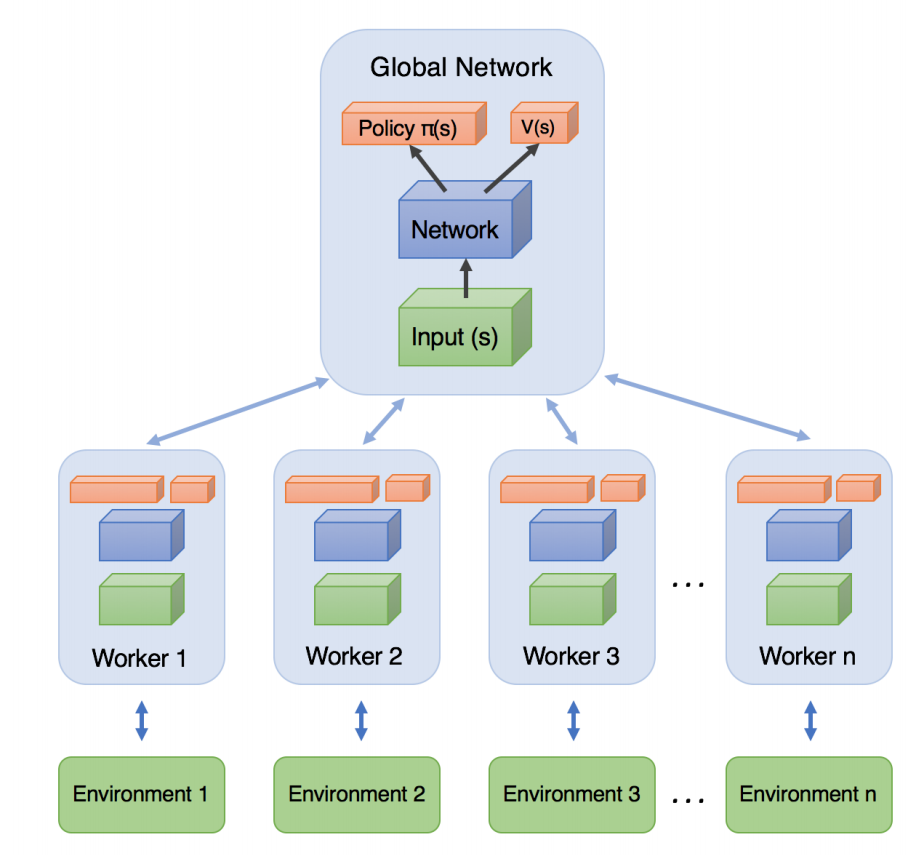


**Figure 2. A3C Algorithm**

**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.

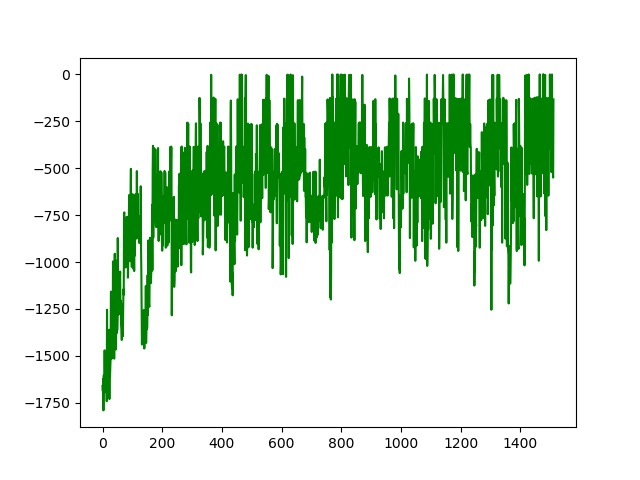
One thing to notice is that the action space is continuous one, thus we use normal distribution , with , .



**Figure 3. Structure of A3C**

**3.3 Result**

As we can see from fig.4, 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 4. Record of total reward using A3C**

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

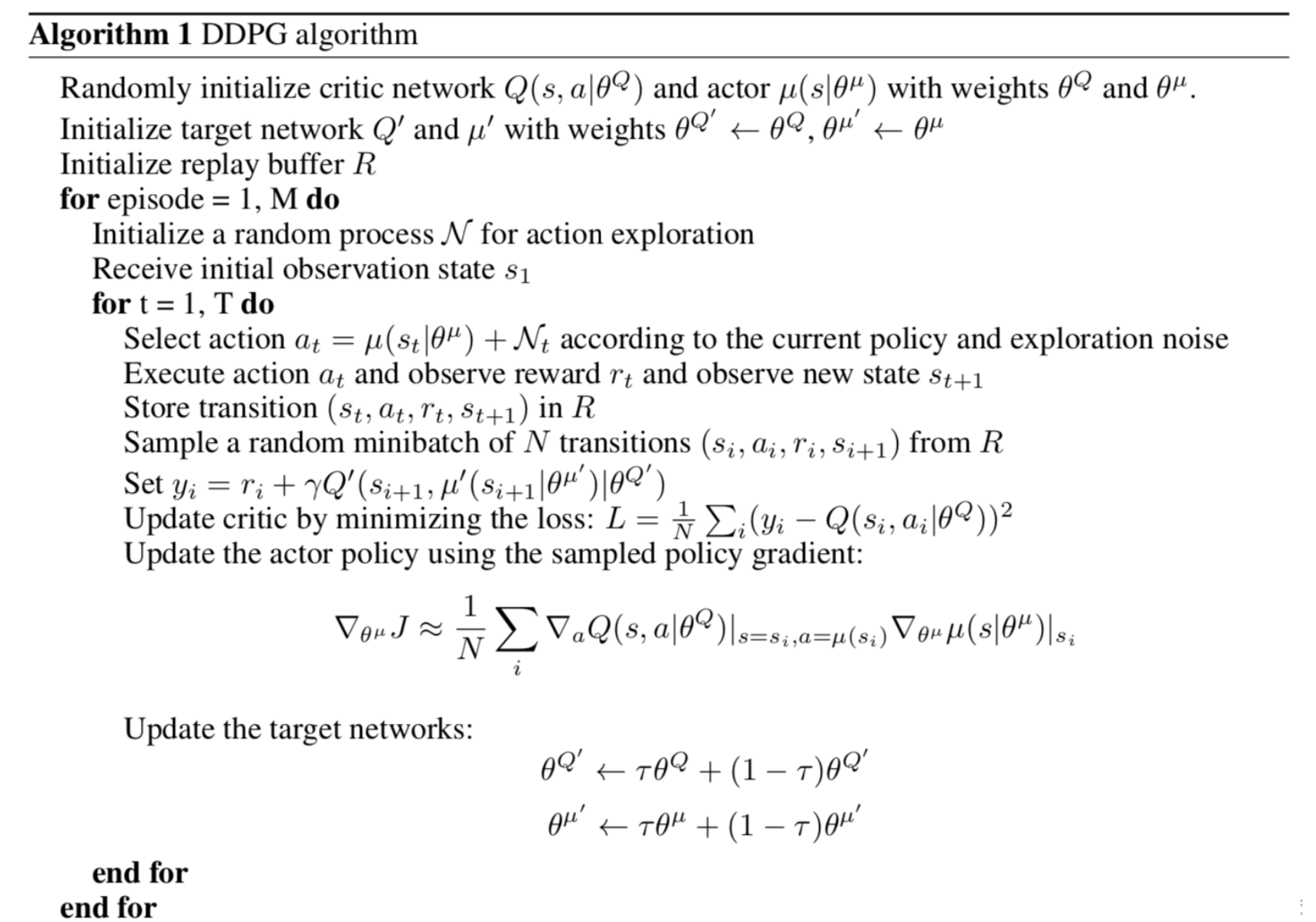
**4.1 Introduction**

To solve the continuous action space problem, DDPG[3] use policy gradient method. It maintains an actor function µ(s|θµ), which decides the action to choose according to the state, and an critic function Q(s, a|θQ) to record the value of each state-action pair.

Due to the divergence problem of neural networks, DDPG also set target networks similar to the target network in [4], but modified for actor-critic and using soft target updates. The target network for actor network is set as µ0 (s|θµ0 ) while that for critic network is set as Q0 (s, a|θQ).

What’s more, unlike the c step in DQN, DDPG have the weights of those target networks slowly track the learned network: θ0 ← τθ + (1 − τ )θ0 with τ ≤ 1, which can solve the problem of divergence.

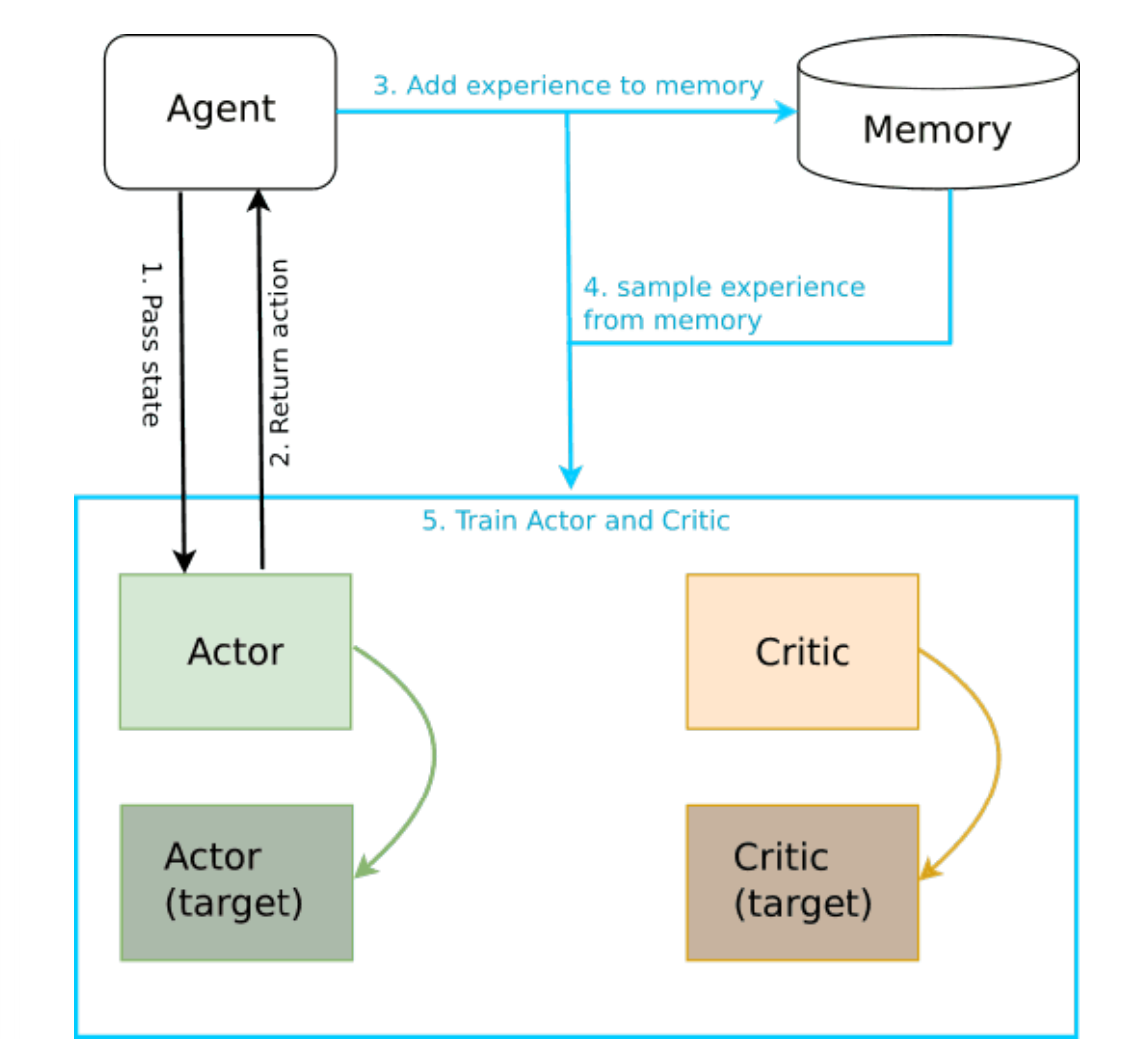
There is also a challenge that the neural network needs independently and identically distributed samples. Therefore, a replay buffer is used in DDPG to store the new samples generated and the actor and critic function will be updated by sampling a minibatch uniformly from the buffer. As DDPG is an off-policy algorithm, the buffer can be quite large for samples.



**Figure 5. DDPG Algorithm**

**4.2 Implementation**

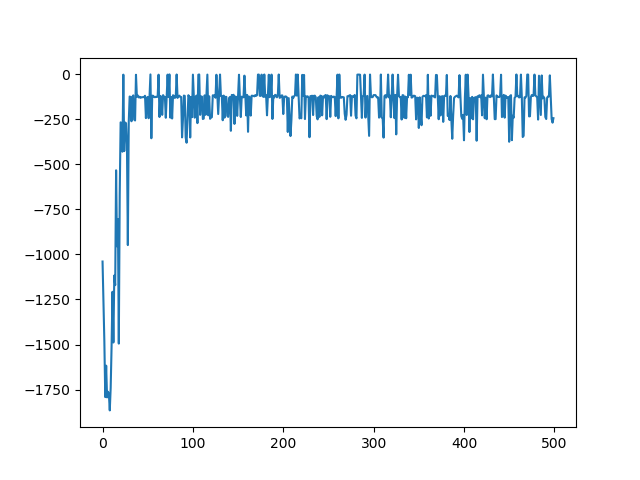
We train 500 episodes to observe the performance of DDPG and for each episode the steps is no larger than 200. Then we set the learning rate of actor to 0.0001 while the learning rate of critic to 0.001. Other settings are γ = 0.99, τ = 0.01, the size of buffer is 1000000 and the size of each batch is 100. As for the noise part, we also use normal distribution, which takes output of actor as , and a descending variation as . After each round of training, the variation would descend according to a certain rate of 0.9995.



**Figure 6. Structure of DDPG**

**4.3 Result**

Here is the record of total rewards during training. Compared to A3C, the converge process is much better, and it takes less than 100 episodes to reach and keep a stable state.



**Figure 7. Record of total reward using DDPG**