Assignment 5

519021911325 刘　畅
519021911179 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.

2. 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Above all, one thing to notice is that Pendulum-v0 has been out of date, using Pendulum-v0 in gym would cause errors, so use Pendulum-v1 instead, which is basically the same.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 $\theta \in [-\pi, \pi], \dot{\theta} \in [-1,1]$.

$$\cos(\theta) \in [-1,1]$$
$$\sin(\theta) \in [-1,1]$$
$$\dot{\theta} \in [-8,8]$$

The action is joint effort, which ranges in [−2, 2]. With state and action, the precision equation of reward is $R = -\theta^2 - 0.1\dot{\theta}^2 - 0.001\text{action}^2$. In general, the more straight upward and stable it is, the better reward it would get.

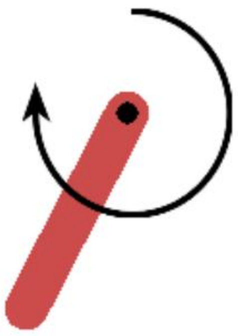
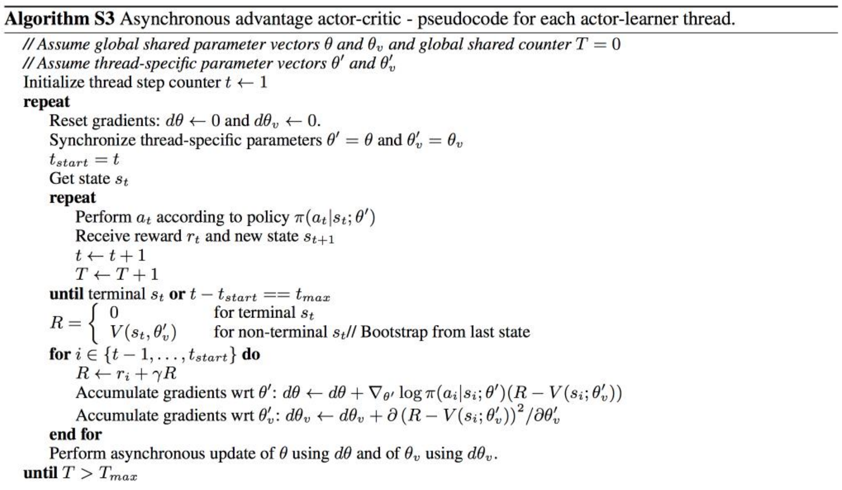


Figure 1. Pendulum-v1

3. 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.



```
Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.
// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$
// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform $a_t$ according to policy $\pi(a_t|s_t;\theta')$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal $s_t$ or $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ do
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
until $T > T_{max}$
```

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 $A \sim N(\mu, \sigma^2)$, with $p(a) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, $\log(p(a)) = -\frac{(x-\mu)^2}{2\sigma^2} - \frac{\log(2\pi\sigma^2)}{2}$.

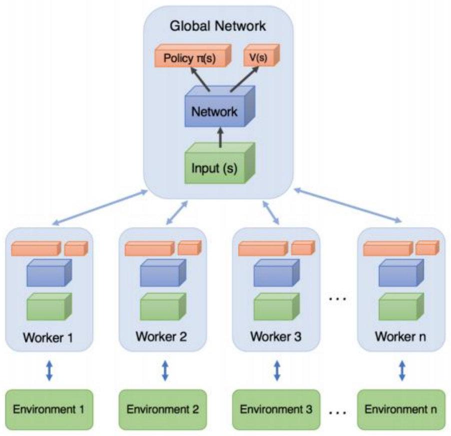


Figure 3. Structure of A3C

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.

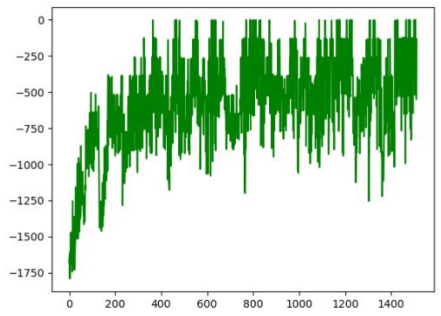


Figure 4. Record of total reward using A3C

4. Deep Deterministic Policy Gradient (DDPG)

4.1 Introduction

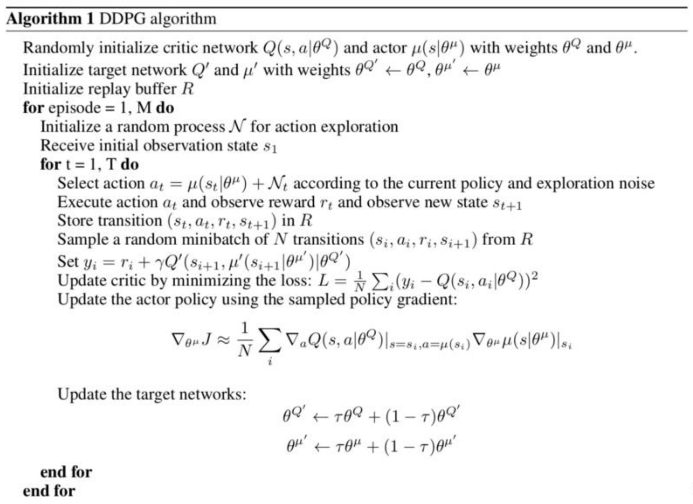
DDPG is the abbreviation of deep deterministic policy gradient. DDPG borrows the success from DQN by introducing experience replay and target Q network into deterministic policy gradient method (DPG). DPG deterministically maps states to specific action instead of outputting a stochastic policy over all actions.

DDPG is similar to DQN, which can be considered as a continuous action version of DQN. The only difference is the action space in DQN is the exact discrete actors, while the action space in DDPG is successive actor network.

$$\textbf{DQN} : a^* = \arg\max_a Q^*(s,a)$$

$$\textbf{DDPG} : a^* = \arg\max_a Q^*(s,a) \approx Q_\phi(s, \mu_\theta(s))$$

Here is the pseudo code of DDPG. For each episode’ every step, DDPG uses action network μ to select a determined action and store transitions to build a memory set with limited capacity. Then it samples a minibatch to update the critic network Q by MSE, and uses the critic network Q to update the actor policy μ by policy gradient. Finally, it uses μ and Q to update the two target network(μ’, Q’).



```
Algorithm 1 DDPG algorithm
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  for t = 1, T do
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

        $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$

    Update the target networks:
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$

  end for
end for
```

Figure 5. DDPG Algorithm

4.2 Implementation

Here is the framework of DDPG. DDPG has two target networks(actor μ’ and critic Q’) and two initial networks(μ, Q).

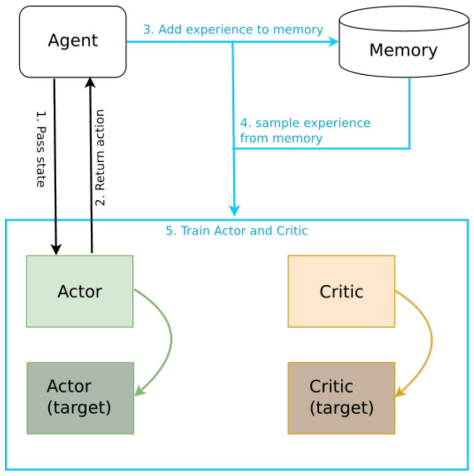

Agent
3. Add experience to memory
Memory
1. Pass state
2. Return action
4. sample experience from memory
5. Train Actor and Critic
Actor
Critic
Actor (target)
Critic (target)

Figure 6. Structure of DDPG

In the program, we set the memory size as 1000000, minibatch size as 100, τ=0.01, the action choice noise is added by a normal distribution, whose variance is decreased. In the gradient decrease part, we set the critic network’s learning rate is 0.001 and the action network’s learning rate is 0.0001. Following the previously described algorithm, we get the reward diagram as below.

4.3 Result

Here is the record of total rewards during training. Compared to A3C, we can see DDPG performs well, it approximately converges to -200 within 50 steps.

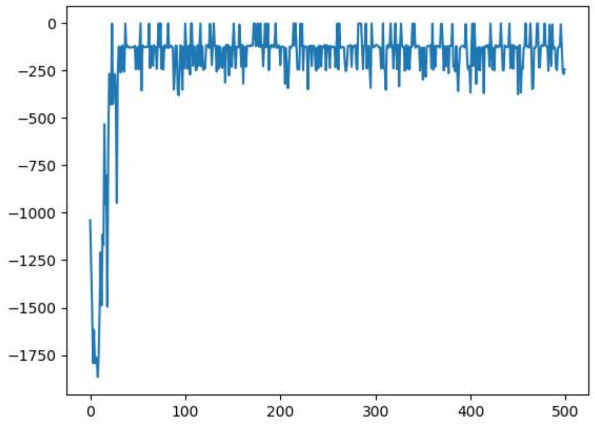


Figure 7. Record of total reward using DDPG