Reinforcement Learning Assignment 5

Hao Sun 516030910362
Xueyan Li 516030910373

June 2019

1 Introduction

To solve the complex tasks from unprocessed, high-dimensional, sensory input, previous algorithms, such as Deep Q Network (DQN)[1], which applied deep network to reinforcement learning, have achived great performance. However, DQN can only solve problems with discrete and low-dimensional action spaces because it needs to find discrete actions that maximize the action-value function. Therefore, Actor-Critic method is used to solve the problem with continuous and high-dimension action space, due to the advantage of policy gradient. In this report, we will introduce two improved AC method, Asynchronous Advantage Actor Critic (A3C)[2] and Deep Deterministic Policy Gradient (DDPG)[3], and implement them on the gym environment "Pendulum-v0".

2 Environment: Pendulum-v0

As shown in fig. 1, the goal of the pendulum-v0 problem is to swing a frictionless pendulum upright so it stays vertical, pointed upwards. The pendulum starts in a random position every time, and, since pendulum-v0 is an unsolved environment, it does not have a specified reward threshold at which it is considered solved nor at which the episode will terminate. There are three observation inputs for this environment, representing the angle of the pendulum (0 and 1) and its angular velocity. The inputs for environments are shown as table 1.

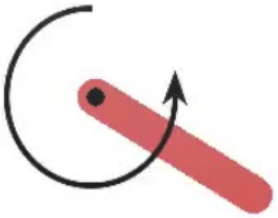


Figure 1: Pendulun-v0

Table 1: Inputs for Environments

Observation	Min	Max
$\sin(\theta)$	-1.0	+1.0
$\cos(\theta)$	-1.0	+1.0
θ_{dt}	-8.0	+8.0

The action *joint effort* ranges in $[-2, 2]$. And the precision equation of reward is

$$R = -\left(\theta^2 + 0.1 * \theta_{dt}^2 + 0.001 * \text{action}^2\right) \tag{1}$$

where $\theta \in [-\pi, \pi]$ and $\theta_d t$ represents the velocity of angle.

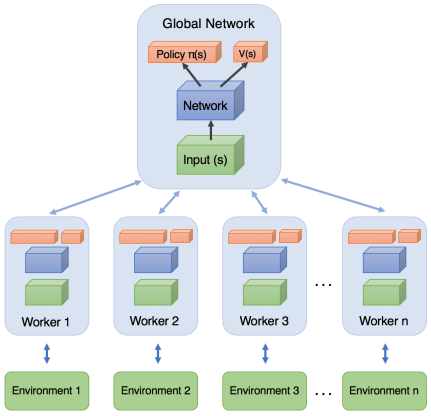
The initial state is random angle $\theta \in [-\pi, \pi]$ and random velocity $\theta_{dt} \in [-1, 1]$.

3 Algorithm

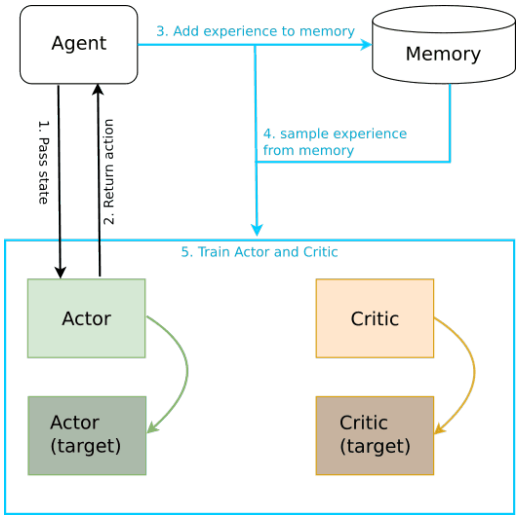
3.1 Asynchronous Advantage Actor Critic (A3C)

A3C was proposed by DeepMind to solve the problem of the Actor Critic convergence. It will create multiple parallel environment, let more agents who has the copy structure on the parallel environment at the same time to update the parameters in the main structure. 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 as algorithm 1. And the network structure of A3C is shown as fig. 2(a).



(a) The network structure of A3C



(b) The network structure of DDPG

3.2 Deep Deterministic Policy Gradient (DDPG)

To solve the continuous action space problem, DDPG[3] use policy gradient method. It maintains an actor function $\mu(s|\theta^{\mu})$, which decides the action to choose according to the state, and an critic function $Q(s, a|\theta^{Q})$ 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 $\mu'(s|\theta^{\mu'})$ while that for critic network is set as $Q'(s, a|\theta^{Q})$. What's more, unlike the c step in DQN, DDPG have the weights of those target networks slowly track the learned network: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \leq 1$, 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.
With all of the above modification, the pseudo code of DDPG is shown in algorithm 2 and the network construction is shown in fig. 2(b)

4 Setting

4.1 A3C

We train 2000 episodes to observe the performance of A3C and for each episode the steps is no larger than 200. The number of workers (CPU) is 4. Then we set the learning rate of actor to 0.0001 while the learning rate for critic to 0.001. Other settings are discount factor γ=0.9.

4.2 DDPG

We train 200 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 0.001 while the learning rate of critic to 0.002. Other settings are $\gamma = 0.9$, $\tau = 0.01$, the size of buffer is 10000 and the size of minibatch is 32.

Algorithm 1: Asynchronous advantage actor-critic for each actor-learner thread

Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$;
Assume thread-specific parameter vector θ' and θ'_v;
Initialize thread step counter $t \leftarrow 1$;
while *true* **do**
    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;
    while *true* **do**
        Perform a_t according to policy $\pi\left(a_t|s_t;\theta'\right)$;
        Receive reward r_t and new state s_{t+1};
        $t \leftarrow t + 1$;
        $T \leftarrow T + 1$;
        if *terminal* s_t *or* t-$t_{start} == t_{max}$ **then**
            Break;
        end
    end
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V\left(s_t, \theta'_v\right) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$;
    for $i \in \{t-1, \cdots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$;
        Accumulate gradients wrt θ':$d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi\left(a_i|s_i;\theta'\right)\left(R - V\left(s_i;\theta'_v\right)\right)$;
        Accumulate gradients wrt θ'_v:$d\theta_v \leftarrow d\theta_v + \partial\left(R - V\left(s_i;\theta'_v\right)\right)^2 / \partial\theta'_v$;
    end
    Perform asynchronous update of θ *using* $d\theta$ *and of* θ_v *using* $d\theta_v$.;
    if $T > T_{max}$ **then**
        Break;
    end
end

Algorithm 2: Deep Deterministic Policy Gradient Algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^Q)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' 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 gradient:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{2}$$

    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
end

5 Result

We made videos of DDPG and A3C to show the training progress of pendulum, which has been put in the same directory. And the code attached with detailed comments is provided.

5.1 A3C

The method successfully converged. The moving reward of each episode is shown as fig. 2. From the figure, we can see that the moving reward will increase over training at the beginning. After about 750 episodes, the moving reward starts to be stable.

5.2 DDPG

The result of DDPG is shown in fig. 3. From the figure, we can find that the moving reward will increase with the training, which proves that DDPG can converge. And after the model converge, the average reward of each episode is about -100, which indicates that DDPG is able to learn good policies. And we find that the convergence of DDPG is very fast. The diagram shows that at about episode 75, the slope of curve is pretty high.

6 Conclusion

We introduce A3C and DDPG, as well as implementing them in our work. From the experiment we conducted with A3C and DDPG, we can find that both algorithm can successfully converge and learn a good policy in continuous action space. Therefore, combining other algorithms like DQN with actor-critic can solve the problem of slow convergence while retaining the advantages of single-step updates.

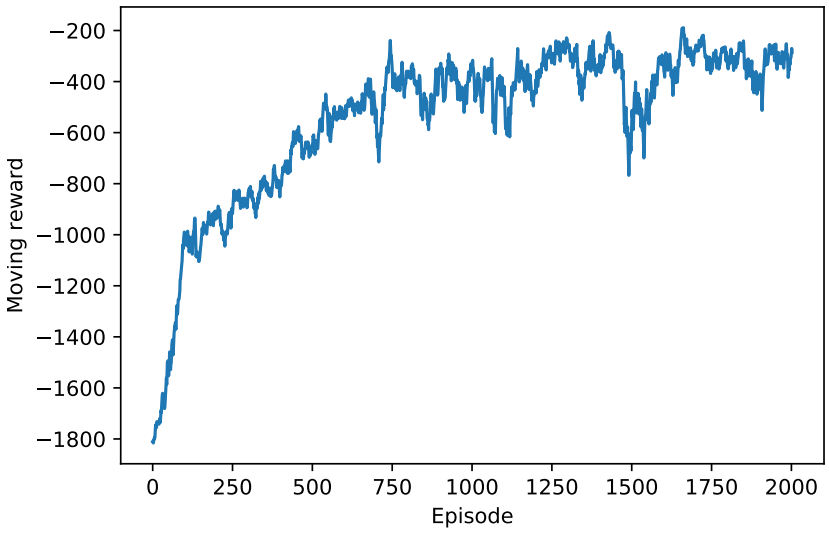


Figure 2: The curve of moving reward of each episode. We can see that the curve converges as last, which shows that the agents have gotten a good policy to keep the pendulum upright.

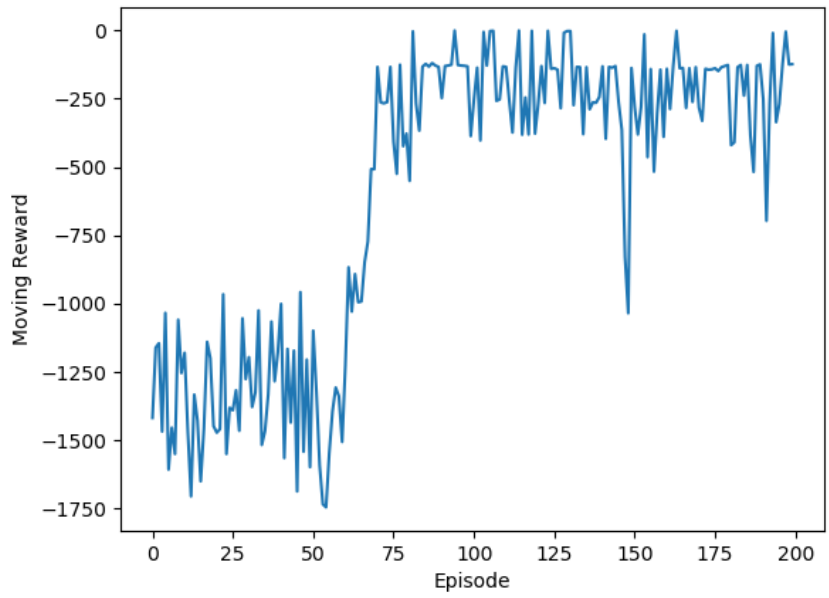


Figure 3: The moving reward with episode in DDPG, which shows that the reward will increase with the training and finally fluctuate aroud about -100. Thererfore, DDPG and converge and reach a good policy.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[2] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, 2014, pp. 387–395. [Online]. Available: http://proceedings.mlr.press/v32/silver14.html

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602