Assignment 5

Yuxiang Lu 518021911194

1 Introduction

Asynchronous Advantage Actor-Critic (A3C) algorithm is an improved version of Actor-Critic method. In this assignment, A3C algorithm is implemented to solve the Pendulum problem, which is a classical RL continuous control scenario.

2 Pendulum

The inverted pendulum swingup problem is a classical control problem in RL, which is provided in `gym` as `Pendulum-v0`, shown in Figure 1. The goal is to swing it up so it stays upright.

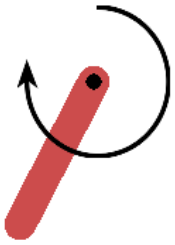

Figure 1: Pendulum Environment

Observation The state of the pendulum has three parameters, representing the angle and the angular velocity, shown in Table 1. The starting state is a random angle from $-\pi$ to π, and random velocity between -1 and 1. There is no specified goal state.

Num	Observation	Range
0	cos(theta)	[-1.0, 1.0]
1	sin(theta)	[-1.0, 1.0]
2	theta dot	[-8.0, 8.0]

Table 1: State of Pendulum

Action As shown in Table 2, the action of the pendulum is a continuous variable with a limit on minimum and maximum value, which represents the control torque of the motor on pendulum.

Num	Observation	Range
0	Joint effort	[-2.0, 2.0]

Table 2: Action of Pendulum

Reward The equation for reward is

$$-(theta^2 + 0.1 * theta_{dt}^2 + 0.001 * action^2)$$

where *theta* is normalized between $-\pi$ and π. Therefore, the lowest reward is $-(\pi^2 + 0.1 * 8^2 + 0.001 * 2^2) = -16.27$, and the highest reward is 0. In essence, the goal is to remain at zero angle (vertical), with the least rotational velocity, and the least effort.

3 Asynchronous Advantage Actor-Critic (A3C)

In Monte-Carlo Policy Gradient, the policy gradient is calculated by

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n)$$

where return $R(\tau^n)$ is used as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$. With tips like adding a baseline and assigning suitable critic, there is

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (\sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n - b) \nabla \log p_\theta(a_t^n | s_t^n)$$

Since it is obtained via interaction, it may be very unstable. Therefore, in Actor-Critic algorithm, action value function $Q^\pi(s_t, a_t)$, named Critic, is used to estimate $Q^{\pi_\theta}(s_t, a_t)$, while state value function $V^\pi(s_t)$ is used to estimate the baseline b.

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (Q^\pi(s_t^n, a_t^n) - V^\pi(s_t^n)) \nabla \log p_\theta(a_t^n | s_t^n)$$

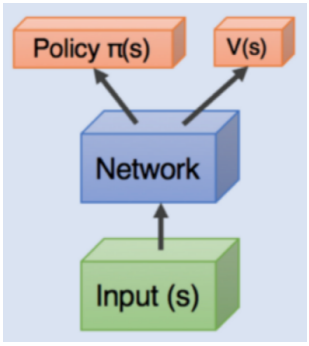

Figure 2: Network used in A2C

Here we need to learn two networks, Q^π and V^π. Advantage Actor-Critic (A2C) algorithm proposes that we can only estimate one network, by changing

the advantage function $Q^\pi(s_t^n, a_t^n) - V^\pi(s_t^n)$ to $r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n)$. Another trick introduced by A2C is that the parameters of actor $\pi(s)$ and critic $V^\pi(s)$ can be shared, therefore the network can be simplified, as shown in Figure 2.

In A3C algorithm, several local agents and a global agent are maintained. The multiple local agents can run in parallel asynchronously, and the global agent can update the global model by the gradients computed by the local agents, shown in Figure 3.

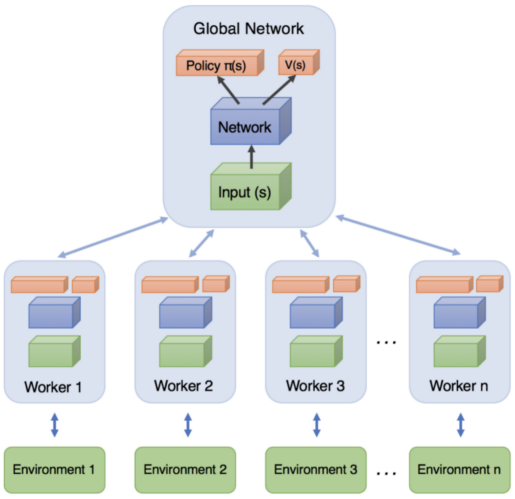

Figure 3: A3C Algorithm

With the help of multiprocessing, the training time is reduced, and no large computing resource like GPU is required. The detailed pseudo code of A3C algorithm is shown in Figure 4.

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_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 θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_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 θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Figure 4: Pseudo Code of A3C Algorithm

4 Experiment and Discussion

Network Structure

According to Figure 2, the network parameters of actor $\pi(s)$ and critic $V(s)$ can be shared, thus getting the network structure shown in Figure 5.

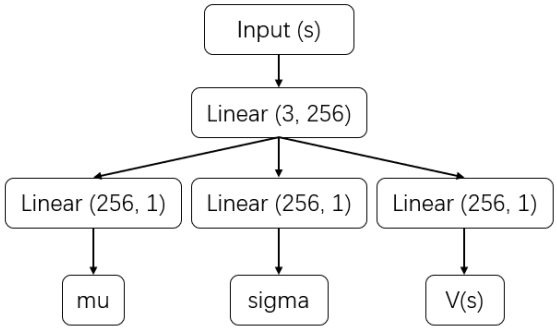

Figure 5: Network Structure

The policy is selected from the Gaussian distribution $G(a|\mu, \sigma)$, where the mean μ and the variance σ are derived from the network. The input is the state s, then the first fully connected layer is shared, and the second layer consists three separated fully connected layers to compute μ, σ and state value $V(s)$.

Settings

There are some hyper-parameters to set in A3C algorithm. Since there is no terminate state in Pendulum environment, the maximum step of each episode is limited to 200. The reward is normalized to $[-1, 1]$ for easier training. The network is trained every five steps in an episode. The discount value γ is set as 0.9. For the training part, the optimizer is a modified shared Adam optimizer, which can use the gradients of local networks to update the global network, and its learning rate is 0.0001. The number of local agents has the upper limit equals to CPU threads, and is set as 8 in my implementation.

Results

The training result of A3C is shown in Figure 6. We can see that the reward sum grows from -1500 to around -300, and converges after 2000 episodes. Then the trained global model is tested, and it turns out the learned policy can successfully swing the pendulum up and keep it upright.

We can also notice that the rewards still have some fluctuations from -400 to -200. This may result from the policy selection generated by a normal distribution. Although the mean and variance are learned by the network, the policy is randomly sampled, instead of a deterministic choice. The randomness causes the algorithm unstable, even after it converges.

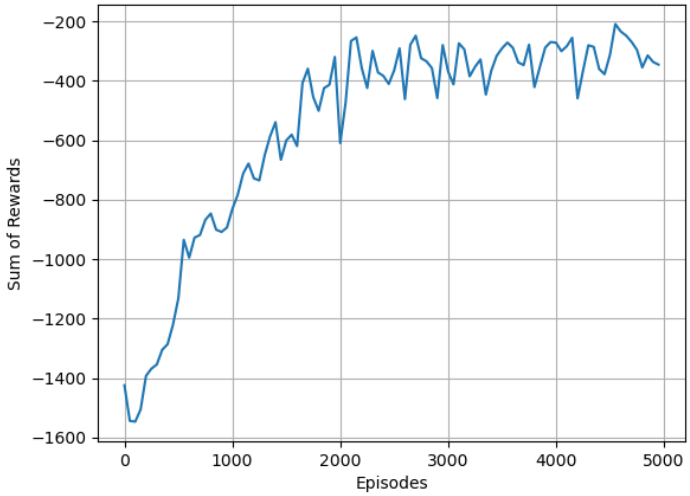

Figure 6: Training Result

Then we change the hyper-parameter that the network is trained every 20 steps in an episode, and the result is shown in Figure 7. We can see that the algorithm obviously learns slower than the previous one, for it converges after over 3000 episodes. A larger training interval let the agent have more experience during each training, and reduce the number of training in the same episodes, therefore the learning process is more efficient. Though it converges slower, the overall computation time may decrease.

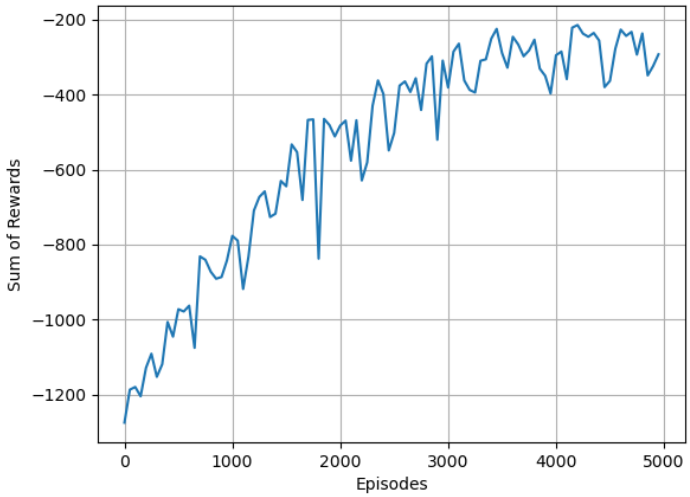

Figure 7: Training Result