Actor-Critic

Sivashanmugam Saravanan Omprakash

Department of Mechanical & Aerospace Engineering

University at Buffalo (SUNY)

Buffalo, NY 14260

saravan3@buffalo.edu

Abstract

The aim of this assignment is to understand the policy gradient concepts, to implement an actor-critic algorithm to solve OpenAI's Gym environment.

1. Advantage Actor Critic (A2C) Algorithm

The Advantage Actor Critic (A2C) algorithm is an on-policy policy-gradient algorithm. This algorithm is the synchronous version of Asynchronous Advantage Actor Critic (A3C) algorithm. It is deterministic and waits for other agents to finish their work before updating the global weights. This makes effective use of the computing resources while also being as effective or sometimes better than the asynchronous algorithm. The critic loss is basically the Root Mean Squared (RMS) error between the TD error & the current state value, while the actor loss is the sample from a categorical distribution then calculate the loss like usually do for probabilities scaled by the advantage, that is using the negative log likelihood.

2. Actor-Critic vs Value-based algorithms

The value-based algorithms, as the name suggests, estimates or learns only the value function. They also suffer from poor convergence. Some examples of value-based algorithms are Q-learning, SARSA, etc. Meanwhile, actor-critic algorithms learn both the policy & value function simultaneously. On the other hand, the Policy based methods have smoother learning curves and performance improves with every update. The drawback with this method is that it tends to converge to local maximas and suffer from high variance, sample inefficiency. Actor critic methods combine the best of both worlds. Employing an actor & a critic results in high sample efficiency.

3. Environments

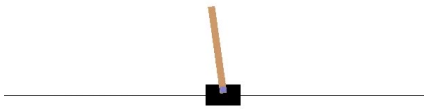
3.1 Cart Pole

A pole attached to a cart moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. A reward of +1 is provided for every time-step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

Actions : [-1, +1]

Rewards : 1

States : Box[4,]



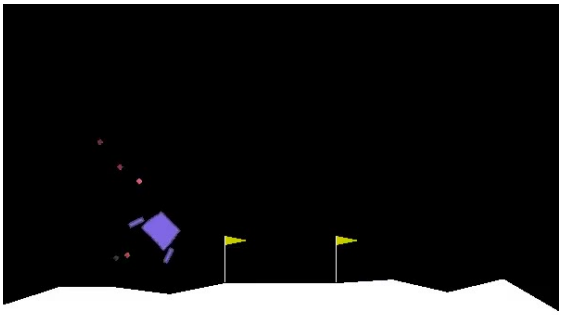
3.2 Lunar Lander

Landing pad is at (0,0) position of the frame. The first two numbers in state vector are the coordinates. Reward for moving from the top of the screen to landing pad and zero speed is about 100-140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Four discrete actions are possible - do nothing, fire left orientation engine, fire main engine, fire right orientation engine.

Actions : Nothing, fire left, fire right, fire main

Rewards : [-100, +100]

States : Box[2,]



3.3 Simple Grid

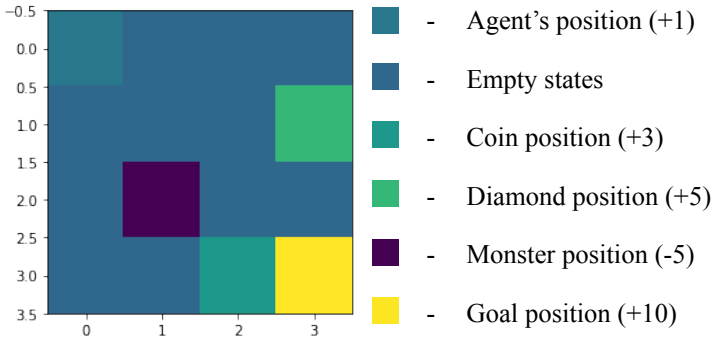


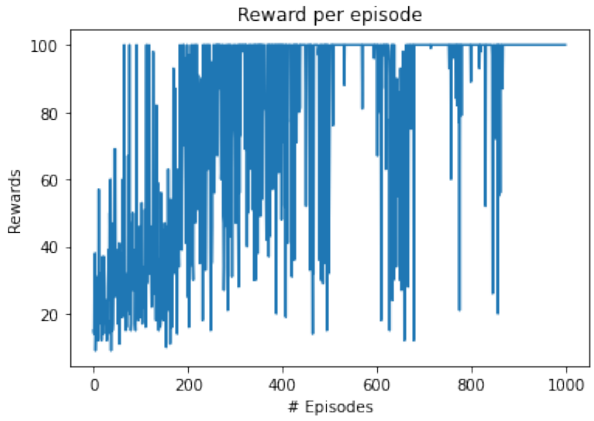
Figure 3: Default environment layout

	Environment	
	Number	**Set**
Action	4	{Left, Up, Right, Down}
States	16	{(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 1), (2, 2), (2, 3), (3, 0), (3, 1), (3, 2), (3, 3)}
Rewards	5	{-5, 1, 3, 5, 10}

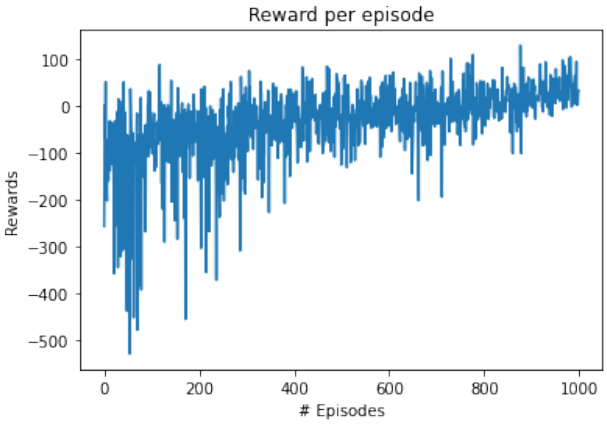
The grid environment has 16 states & 4 discrete actions possible. The agent always starts at (0, 0), the top-left state. Based on what element the agent encounters when exploring will determine the value of the reward obtained. Empty states have a reward of zero.

4. Result

The A2C agent was implemented on the above said environments and the following are the rewards per episode performance in each case. In all the cases, the maximum time-step allowed per episode is 100 and the default number of episodes for which the agent acts on the environment is 1000. The gamma value is set to 0.99 by default and the learning rate of the agent's optimizer is 0.001.



The above picture shows the performance of the A2C agent in the CartPole environment. As we can see, the agent has performed pretty well. In this algorithm, the critic network will estimate the value while the actor network estimates the probability distribution of the actions for the given state. By tuning the hyper-parameters, such as the discount rate, learning rate, etc., the performance can be further improved.



The above chart shows the performance of the A2C agent in the Lunar Lander environment. As we can see the agent, over the episodes, learns the environment and shows a general upward trend of the total rewards earned per episode. Some of the drops are due to the randomly inferring negative rewards over multiple time steps. Again, by tuning the hyper parameters, the performance can be further increased. Changing the network dimensions can also attribute to better Q prediction for a given state.