Deep Reinforcement Learning
Project 2
Continuous Control

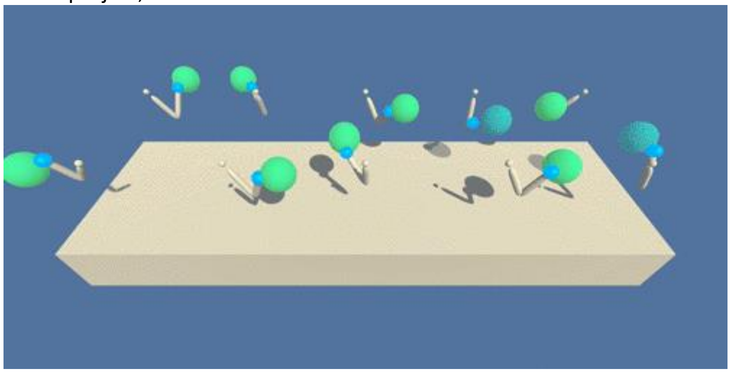
Submitted by: Avinash Kaur
Date: 29th June 2019

Objective

The purpose of this project is to apply RL algorithms to scenarios with continuous action spaces. We intend to use the actor critic methods that utilize policy gradients (actor) as well as value based (critic) to train a smart agent that solves the given environment.

Problem Statement

In this project, we will work with the Reacher Environment.



What we see in the figure are double jointed robotic arms that can move toward a target location. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.

State and Action Spaces

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

Solving the environment

There are 2 versions of the problem that we can work on.

1. Version 1 has a single robotic arm. It is an episodic task and the aim is to get an average score of +30 over 100 consecutive episodes.
2. Version 2 has multiple agents/robotic arms. The aim is to get an average score of +30 0ver 100 consecutive episodes over all agents. Specifically,
   - After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 20 (potentially different) scores. We then take the average of these 20 scores.
   - This yields an **average score** for each episode (where the average is over all 20 agents).

Learning Algorithm

The algorithm chosen for implementation in this project is the Deep Deterministic Policy Gradient (DDPG). Version 2 of the problem (with 20 agents) was chosen for this project.

DDPG (Deep Deterministic Policy Gradient)

DDPG, as the name suggests, is a policy gradient method that directly optimizes parameterized policies instead of estimating a value function and then obtaining an optimal policy. All Policy gradient methods have this as their underlying philosophy. They also provide a robust solution to the intractability caused by continuous action spaces.

In particular, DDPG utilizes two neural networks: an actor and a critic. The Actor tunes the parameter θ of the policy function to decide the best action for a given state, such that:

$$\pi_\theta(s,a) = \mathbb{P}[a \mid s, \theta]$$

Whereas, the critic evaluates the policy function estimated by the actor following the same old Bellman equation such that:
$r_{t+1} + \gamma V^v(S_{t+1}) - V^v(S_t)$
Here the terms carry the same meaning as in the TD setting with nothing new until this point (only in critic).

We utilize the same advances we made in the Q learning algorithm to stabilize training i.e. the usage of a replay buffer and a fixed Q Network. This is done in both actor and critic. So in essence, we have local and target networks for both the actor and the critic. Another change here from the regular Q learning algorithm is the usage of 'soft' updates instead of directly copying the weights of the local network to target after a fixed number of timesteps. It has been observed that this provides more stabilized training than directly updating our target networks. This is expressed by the following equation:
$\theta_{target} = \tau * \theta_{local} + (1 - \tau) * \theta_{target}$
Where 'tau' is the hyperparameter that determines the extent to which we utilize local network to update target network at each time step. It is typically set to 0.001.

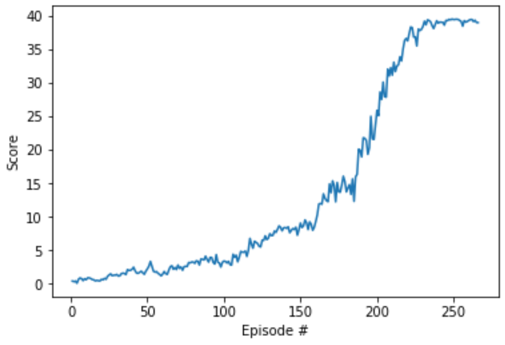
Chosen Hyperparameters

PARAMETER	VALUE
REPLAY BUFFER SIZE	1e6
BATCH SIZE	256
GAMMA	0.99
TAU	0.001
LEARNING RATE ACTOR	2e-4
LEARNING RATE CRITIC	2e-4
WEIGHT DECAY	0.0001
EPSILON	1.0
UPDATE EVERY	20

Neural Network Architecture

ITEM	ACTOR	CRITIC
NUMBER OF FC LAYERS AND UNITS	2 FC layers, 256 and 128 units each	2 FC Layers, 256 and 128 units each
NN ACTIVATION FUNCTION	RELU for both layers	RELU for both layers. The output from the first layer after RELU is concatenated with the 4 actions.
BATCH NORMALIZATION	Yes	Yes.
OUTPUT LAYER	4 nodes with Tanh non linearity to clamp output in the range of -1 to 1.	1 output node with RELU activation.

Plot of rewards per episode



Results

- As can be seen from the above plot, the number of episodes needed to solve the environment by ALL agents: 266 episodes.
- However, the number of episodes required to solve it based on averaged reward collected by all agents together is 217 episodes.

Summary and Future ideas of improvements

- Before trying version 2 of the environment, I tried version 1 with a single agent. However, it took 386 episodes to solve that environment. Clearly, I need to experiment more and perform algorithmic changes to improve that. The same has not been produced here in this report.
- The solution provided solves the environment in 266 episodes. While training it was observed that for the first 100 episodes there's only a marginal improvement in the average reward collected and is almost linear. It's not necessarily bad but probably there could be more work done to accelerate this learning. Therefore, experimenting with NN architecture or tweaking the exploration time (thereby experimenting with epsilon values) might reduce the training time.
- Other experiments like PPO, A2C, A3C, etc. could be implemented and compared to understand the true nature of learning for this experiment.