Project 2- Continuous Control

Introduction

As part of Continuous Control project, I have leveraged deep reinforcement learning to help train Unity-ML agent to move a double-jointed arm to a target location. I have leveraged Deep Deterministic Policy Gradient (DDPG) as described in the paper
https://arxiv.org/pdf/1509.02971.pdf

Deep Deterministic Policy Gradient

As part of the Deep Deterministic Policy Gradient approach we have built for this project, the space size is 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. It leverages 2 hidden layers. The ***action space*** is continuous and each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector is a number between -1 and 1 to build final solution for this project.
The ***final output layer*** of the actor was a tanh layer, to bind the actions.
A ***reward*** from +0.01 to +0.04 is provided for each step that the agent's hand is in the goal location. A higher reward is provided if the agent's hand is closer to the center of the target.
The agents must get an average score of +30 (over 100 consecutive episodes, and over all agents).
The environment is considered solved, when the average (over 100 episodes) of those average scores are at least +30.
Thus, our final policy has the following parameters:

1. Input Layer - 33 input variables
2. Hidden Layer 1 - 400 units
3. Hidden Layer 2 - 300 units
4. Output Layer - Tanh layer

Both the Hidden layer uses Actor-critic approach

Under the Hood

As part of this implementation, complete code / python notebook can be found at location:
https://github.com/divya-bhanot/Project-2_Continous_control

Hyper parameters used

Hyper Parameters are parameters which are given by the users to train a model. Similarly, for DDPG we defined several hyper-parameter to ensure we get optimized results.
Though it took me some iterations to come up with right set of values, following are the final values which helped me get optimum threshold results and their significance:

Buffer Size -> This hyper-parameter define how many values of state, action, etc. we could store in memory. For our model, we have defined this to be 1e5 (100,000)

Batch Size -> Number of examples to be picked up when the agent updates the learning curve i.e. for learning updates. I have taken that as 128 following the DDPG paper.

Gamma -> The discount factor which we have used is multiplied by future rewards in order to dampen these rewards' effect on the agent's choice of action. We have used discount value of 0.99.

LR_Actor -> This rate denotes how the learning curve for Actor optimizes the results in each run and minimize the loss function. The value for current implementation is taken as 2e-4.

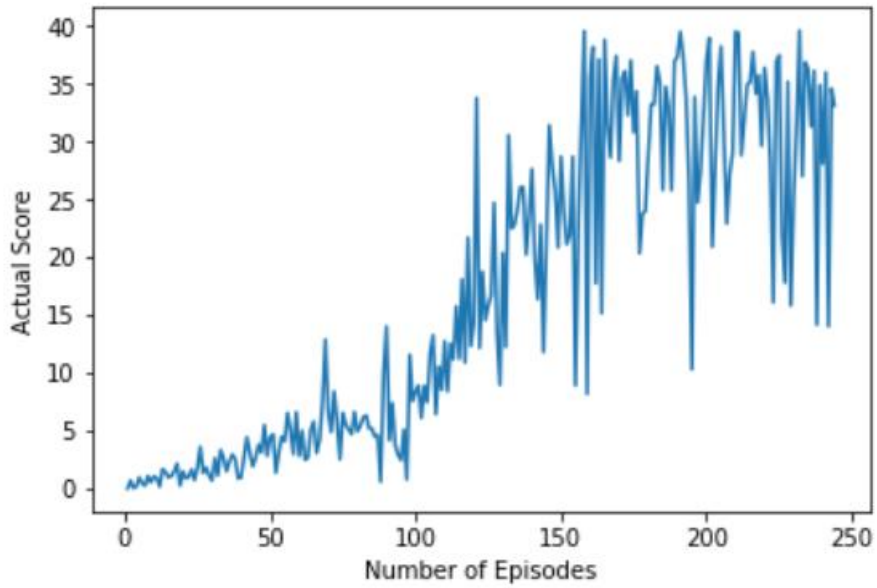
LR_CRTIC -> This rate denotes how the learning curve for Critic optimizes the results in each run and minimize the loss function. The value for current implementation is taken as 2e-4.

Tau -> Tanh activation factor, which steers the action values corresponding to different actions like position, rotation, velocity, and angular velocities of the arm of the agent.

WEIGHT_DECAY -> Multiplicative factor, which helps while training the agent.

Output

Below is the plot of the rewards per episode experience by the agent during final solution training.



Future Changes

Here are list of changes which we could do:

1. Use prioritized experience replay. The current implementation of PER is not efficient enough to use with large buffer sizes, it needs to be reworked, but could be another way to solve the environment in fewer episodes.

2. A thorough hyper parameter grid search. Time and money constraints prevented an exhaustive hyper parameter search.