Udacity Continuous Control Project

Introduction

This Project is given by Udacity Deep Reinforcement course. The aim is train a double-jointed arm that can move to target location. The aim is hold the arm at a point and move it through special location. A reward 0.1 point will be given for each step that the arm holds its position at target so the aim is hold its position at most. The agent must provide an average score over of +30 over 100 consecutive episodes.

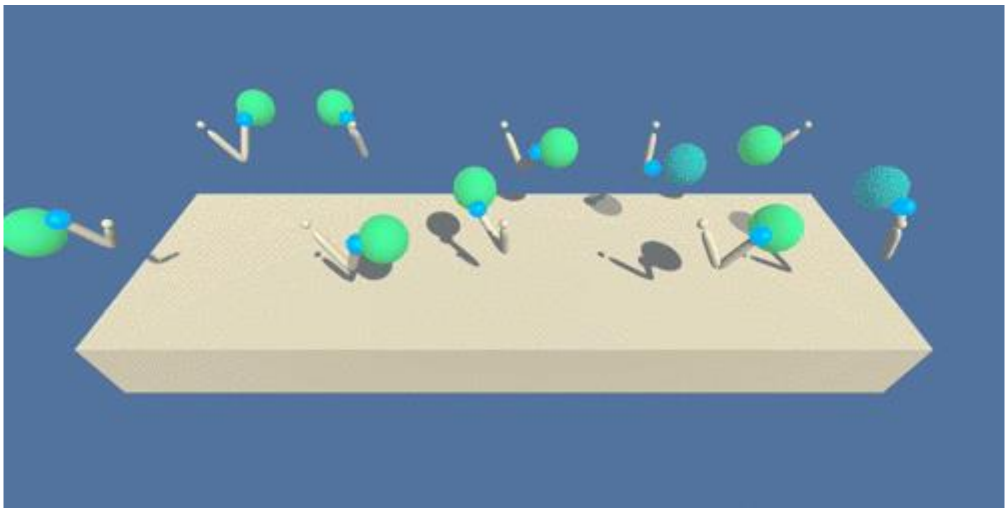

Figure1: An image of environment of continuous control.

The environment has continuous action space between -1 and +1, and 33 states size.

The agent is trained for score of 31 and the result is as following:

Environment solved in 288 episodes!   Average Score: 31.54 in 4876 second

Sensitivity: Public

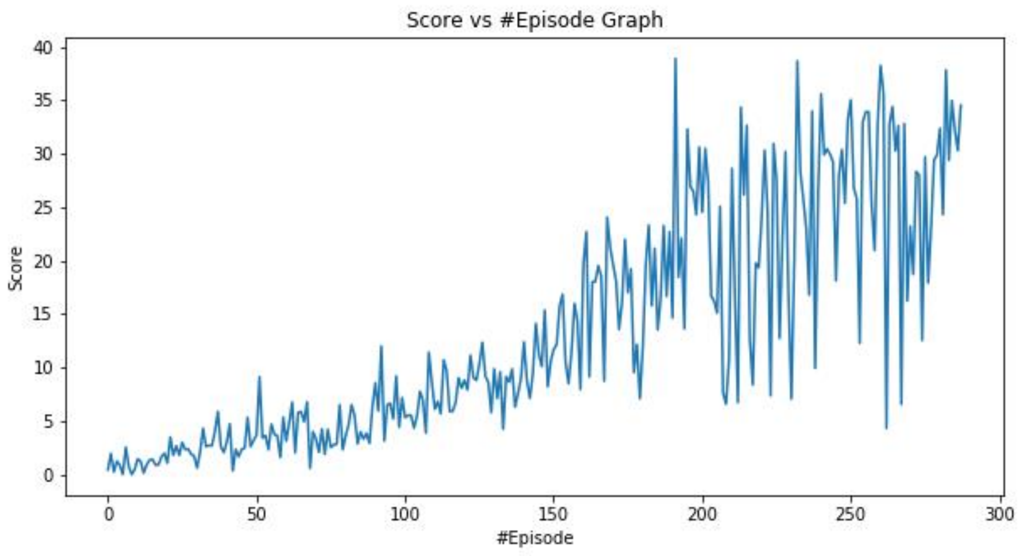

Figure 2: Graph of score 31

Learning Algorithm & Model Architecture:

The DDPG algorithm is used in the project. It has 3 layers. 33 inputs, 128 nodes and 128 nodes hidden layer and 4 action node for continious. The same architecture is used both actor and critic side of algorithm.

Figure 3: Graph of score architecture

The architecture is selected with some experiment. The experiment was run the agent for score 10 and compare the results. Three experiments are done and the same architecture is used both critic and actor side, one for architecture 128*128 nodes, one for 256*128 nodes and one for 128*56 nodes.

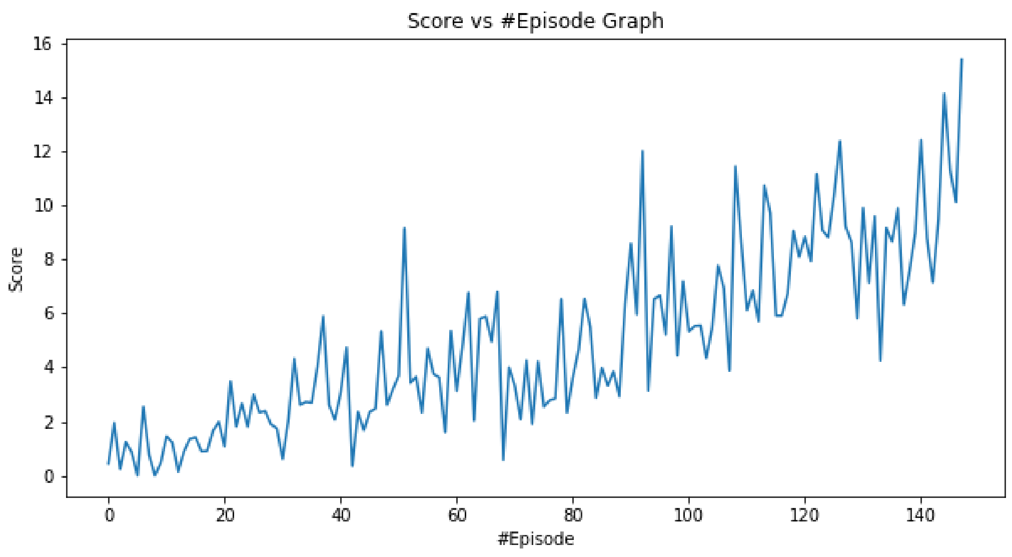

Figure 4: Graph of score 10 for architecture 128*128

When we built the environment as 128*128 the agent solved in 148 episodes in 2328 seconds.

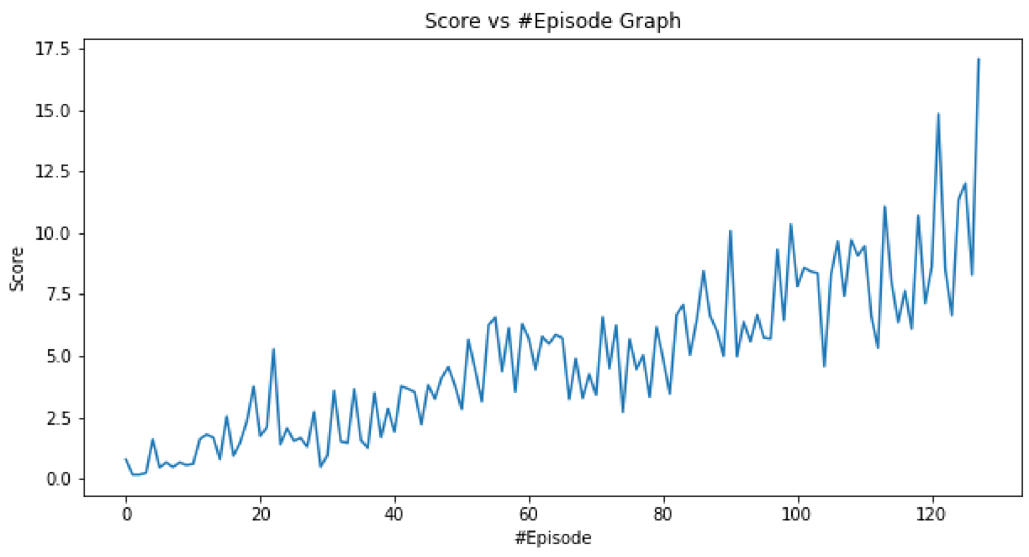

Figure 5: Graph of score 10 for architecture 256*128

When we built the environment as 256*128, the agent solved 128 episodes in 2917 seconds.

Sensitivity: Public

The time is increased in 256*128 architecture but the difference between episodes are become less which means that more stable training appeared.

The architecture of 128*56 cannot solved in logical range so it is stopped and eliminated since it become as average score 8 in 4058 seconds so time was too much.

Therefore, for this project is chosen an architecture 128*128 due to time concern.

These values are directly used from dqn_agent.py

BUFFER_SIZE = int(1e5)	**# replay buffer size**
BATCH_SIZE = 128	**# minibatch size**
GAMMA = 0.99	**# discount factor**
TAU = 1e-3	**# for soft update of target parameters**
LR_ACTOR = 2e-4	**# learning rate of the actor**
LR_CRITIC = 2e-4	**# learning rate of the critic**
WEIGHT_DECAY = 0	**# L2 weight decay**
QUNoise_theta = 0.15	**#QUNoise theta factor**
QUNoise_sigma = 0.15	**#QUNoise sigma factor**

Problems During Project:

During the project, there were two different difficulty. First one was network size problem. The agent was not training takes too long time since at the beginning I took the size from example then I made some experiment as above. The second one is like first one. When I run the code in my local machine it takes too much time so I run the code in Udacity workspace in CPU mode and the speed of workspace is better with respect to my local environment. The experiments did in above. Moreover, during training phase, I run the code for lower scores and after then I run the code for higher scores.

Future Works:

- More architecture experiment and parameter optimization
    - The parameters are taken from exercises so there could be better ones and, I tried 3 different model architectures so there could be better architectures as well.
- Implementing second version of environment and Crawl enrolment.
    - Training agents for these environments could be challenging and gives me inspirations.

Sensitivity: Public