DDPG for Unity Reacher Environment (Single Arm)

I. Environment

In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. The goal of the agent is to maintain its position at the target location for as many time steps as possible.

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.

In this implementation a single agent version of the Reacher environment is solved.

The task is episodic, and to solve the environment, the agent must get an average score of +30 score over 100 consecutive episodes.

II. Results

The DDPG network was an MLP with 2 hidden layers. The input layer has 33 neurons as there were 33 observations from the environment and the output layer has 4 neurons. The Hidden layers each had 256 neurons. Please refer to architecture for details. The network achieved an average of 30+ over 100 consecutive episodes within 400 iterations of training. To converge gradient clipping was needed. This is in the learn function before critic_optimizer.step().

Parameters :

Mini batch size :256.

TAU for soft updates : 1e-3

Hidden layers : 2 layers of 256 neurons

Discount rate gamma = 0.99

Noise parameters:

Theta: 0.10 and Sigma = 0.10 for Ornstein-Uhlenbeck process.

The training started converging only after many tweaks. The training report screen shots are below.

1) Results with learning update once as the score reaches 5.0 fig 1.

Starting learning rate (actor and critic) : 2e-4

Ending learning rate : (actor & critic) : 1e-4

The actor converges fast in this case after the learning rate is updated. :1

```
(drlnd) Jojimons-iMac:net2 jojimonvarghese$ !py
python Reacher.py
=======================The Brain=========================================
Mono path[0] = '/Users/jojimonvarghese/Desktop/Udacity/deep-reinforcement-learning/Reacher/net2/Reacher.app/C
Mono config path = '/Users/jojimonvarghese/Desktop/Udacity/deep-reinforcement-learning/Reacher/net2/Reacher.a
INFO:unityagents:
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :
                goal_speed -> 1.0
                goal_size -> 5.0
Unity brain name: ReacherBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 33
        Number of stacked Vector Observation: 1
        Vector Action space type: continuous
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
=======================Show env==========================================
Number of agents: 1
Number of actions: 4
States look like: [ 0.00000000e+00 -4.00000000e+00  0.00000000e+00  1.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -4.37113883e-08  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -1.00000000e+01  0.00000000e+00
  1.00000000e+00 -0.00000000e+00 -0.00000000e+00 -4.37113883e-08
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  5.75471878e+00 -1.00000000e+00
  5.55726671e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00
 -1.68164849e-01]
States length: 33
================================================================================
================================================================================
=======================Network Training=========================================
Episode 100     Average Score: 1.28
Episode 200     Average Score: 4.45
Episode 213     Average Score: 4.97
Updating LR..
Episode 300     Average Score: 8.51
Episode 400     Average Score: 23.44
Episode 448     Average Score: 30.06
Environment solved in 448 episodes!     Average Score: 30.06
Episode 500     Average Score: 31.63
Episode 600     Average Score: 31.43
Episode 700     Average Score: 30.68
```

Fi. 1 : Outputs

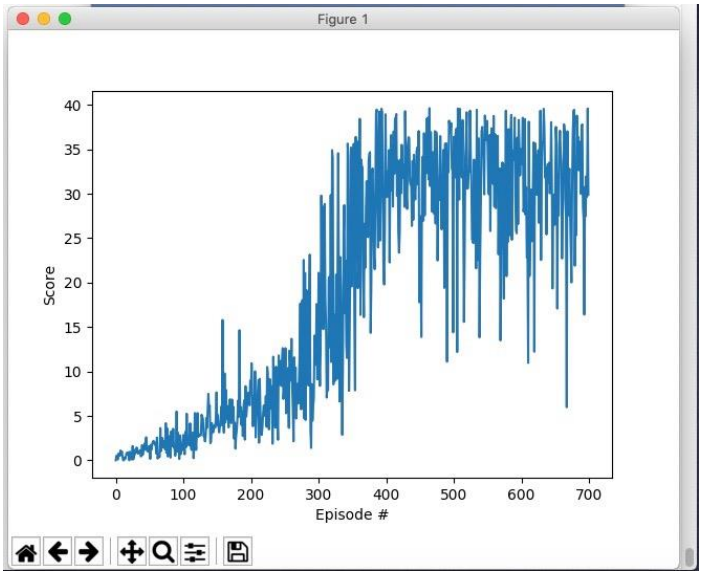


Fig. 2 Score plot

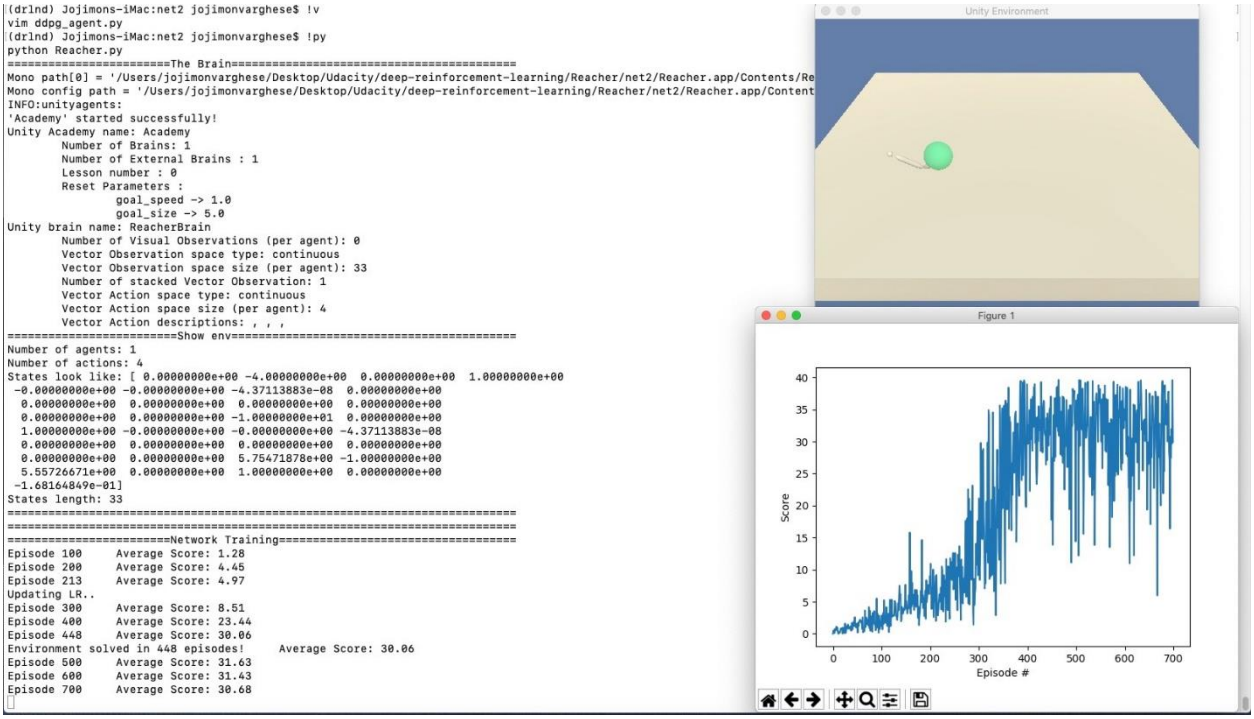


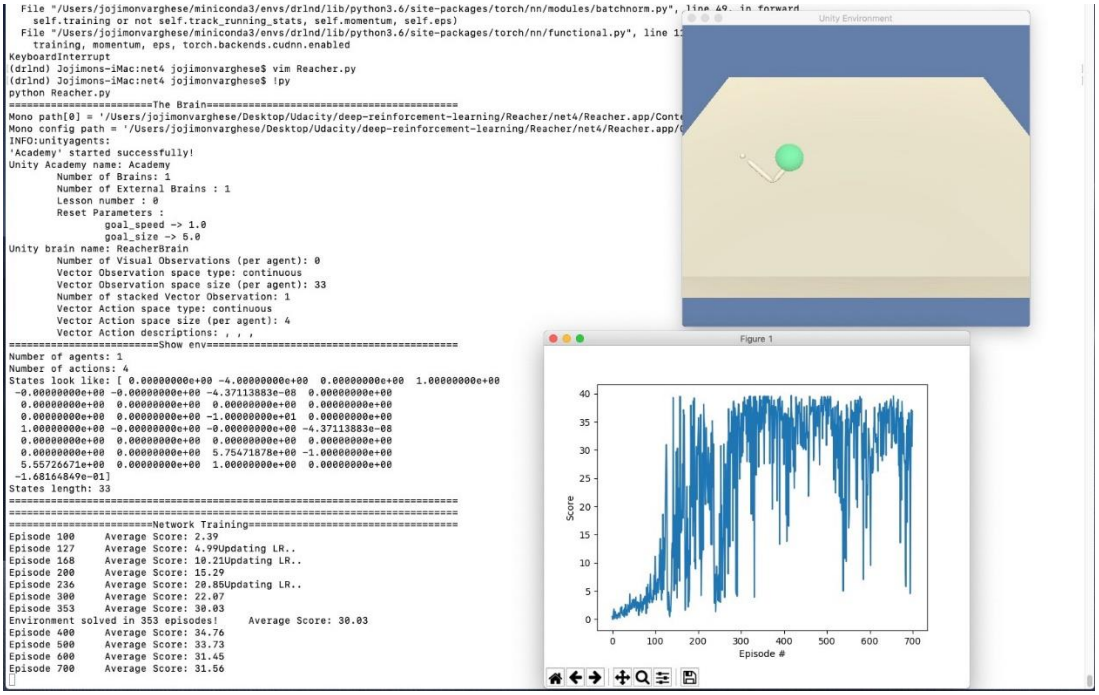
Fig. 3.

2) Results with learning rate updates and replay buffer refresh as training converges. Fig2.

Starting learning rate (actor and critic) : 2e-4

Ending learning rate : (actor & critic) : 5e-5

Convergence is slower after second LR update when score is 20+.



III. Future Improvements

1. Adding priority to experience replay

Based on the many experiments I ran to get DDPG converge it seem to be a good option to use prioritized experience replay with DDPG for faster and stable convergence. To use prioritization based on TD-Error or Stochastic prioritization is better for DDPG is something than need to be explored.

1.1 Prioritizing with TD-error

The central component of prioritized replay is the criterion by which the importance of each transition is measured. One idealized criterion would be the amount the RL agent can learn from a transition in its current state (expected learning progress). While this measure is not directly accessible, a reasonable proxy is the magnitude of a transition's TD error δ, which indicates how 'surprising' or unexpected the transition is: specifically, how far the value is from its next-step bootstrap estimate. The TD-error can be a poor estimate in some circumstances as well, e.g. when rewards are noisy.

1.2 Stochastic prioritization

In Stochastic prioritization, a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling is used. It ensures that the probability of being sampled is monotonic in a transition's priority, while guaranteeing a non-zero probability even for the lowest-priority transition. Concretely, the probability of sampling transition i is defined as

$$P(i) = \frac{p_i^{\propto}}{\sum_k P_k^{\propto}}$$

2.Asynchronous Methods for Deep Reinforcement Learning.

I had been training DDPG on a single CPU machine on single thread. This takes extremely long time (many hours) to experiment with different parameters. An additional improvement would be to use A3C to optimize the training and debug times. For this the 20-agent version of Reacher environment would be best. As well.