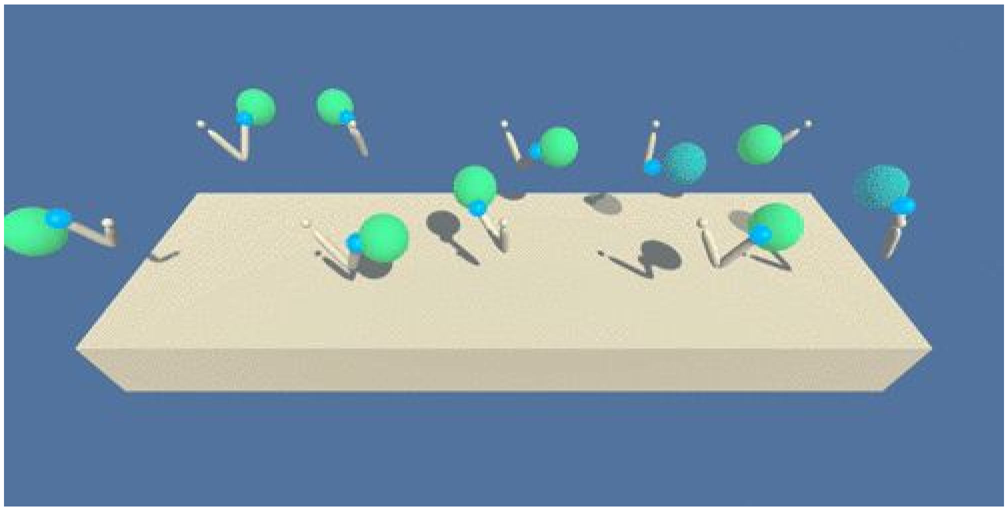
Continuous Control using DDPG



Matthew Hayes

09.09.2019
Udacity Deep Reinforcement Learning Project 2

INTRODUCTION

For this project, you will work with the Reacher 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. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.

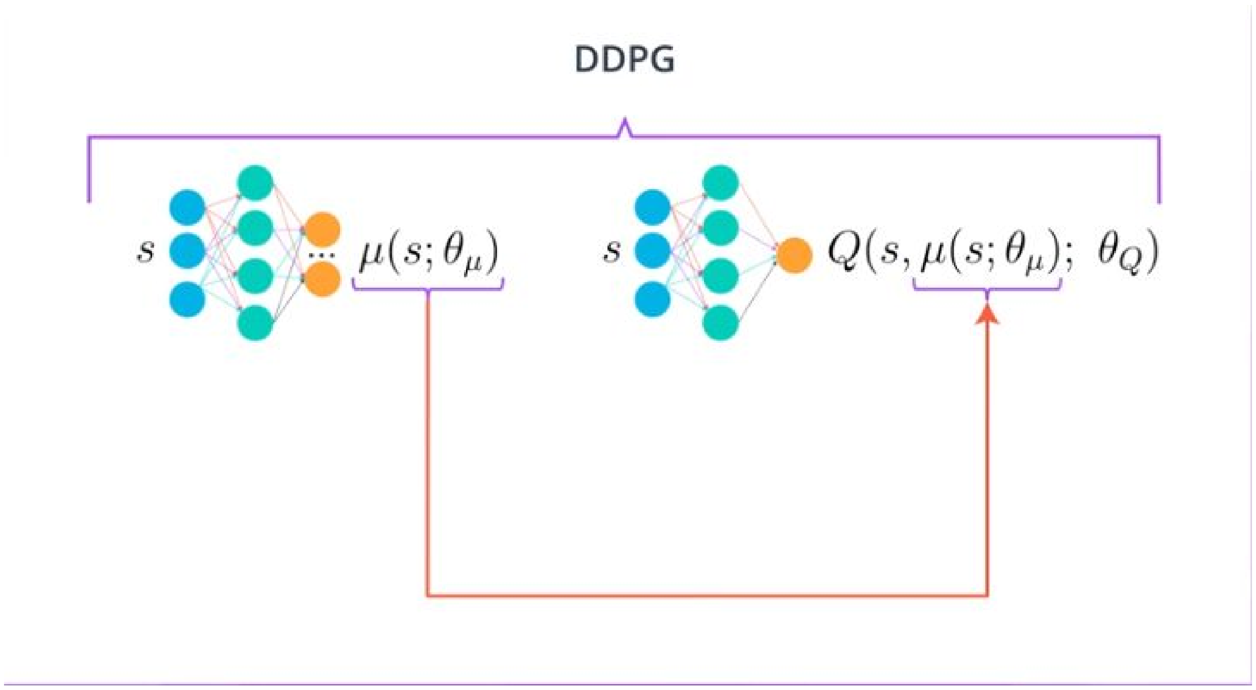
The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

DDPG Overview

The Deep DPG (DQN) was a deep reinforcement learning method proposed by DeepMind, The published paper illustrated the techniques DDPG paper.

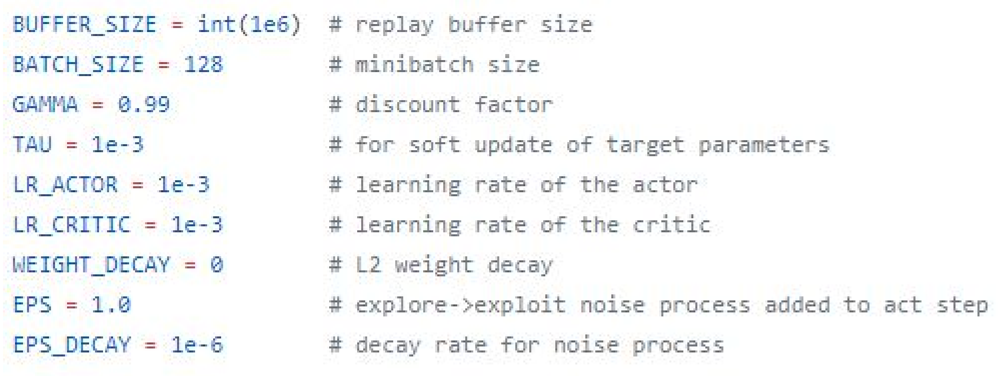
DDPG has these important features:

- An **actor-critic**, **model-free** algorithm based on the deterministic policy gradient that can operate over **continuous action spaces**.
- **Experience Replay**: experience replay that randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. Because DDPG is an **off-policy algorithm**, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.
- **Target Network**: In TD error calculation, target function is changed frequently with DNN. Unstable target function makes training difficult. So Target Network technique fixes parameters of target function and replaces them with the latest network every X steps.
- DDPG is similar to the target network used in (Mnih et al., 2013) but modified for actor-critic and using **“soft” target updates**, rather than directly copying the weights.
- When learning from **low dimensional feature vector observations**, the different components of the observation may have **different physical units** (for example, positions versus velocities) and the ranges may vary across environments. This can make it difficult for the network to learn effectively and may make it difficult to find hyper-parameters which generalise across environments with different scales of state values. One approach to this problem is to manually scale the features so they are in similar ranges across environments and units. We address this issue by adapting a recent technique from **deep learning called batch normalization (Ioffe & Szegedy, 2015)**. This technique normalizes each dimension across the samples in a minibatch to have unit mean and variance.



*source: Udacity Deep Reinforcement Learning

DDPG Optimization Params



```
BUFFER_SIZE = int(1e6)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-3         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
EPS = 1.0               # explore->exploit noise process added to act step
EPS_DECAY = 1e-6        # decay rate for noise process
```

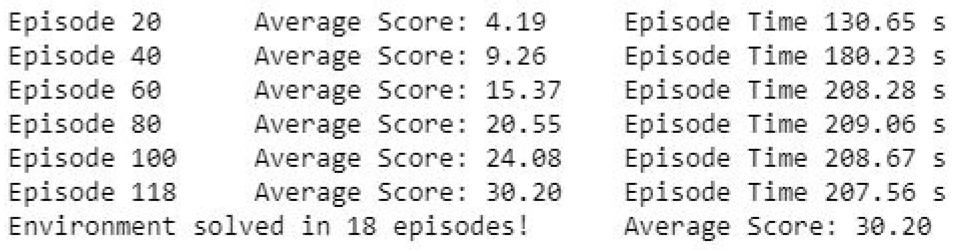
Code Overview

1. Continuous_Control.ipynb: iPython Notebook driving the DDPG training:
    a. Imports packages
    b. Examine environment
    c. Demonstrate how to take random actions in env.
    d. Train DDPG based on 20 agents.
    e. Plot results.
2. ddpg_agent.py: Implements DDPG.
    a. Initialize local and target copies of actor critic networks, noise generator and replay mem.
    b. Step: Save experience in replay memory. Every 20 agents of data call learn.
    c. Act: Returns actions for given state as per current policy. Noise is introduced for exploration but noise reduced as solution coverges
    d. Learn: Update actor critic networks.
    e. Update: Soft update model parameters.
3. Model.py:
    a. Actor network architecture.
    b. Critic Network architecture.

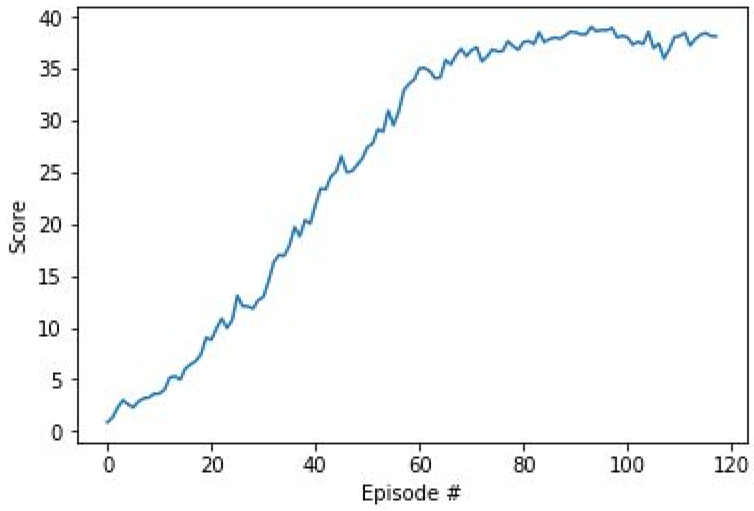
*https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum was used as a code baseline for implementation.

Results

Project's requirements satisfied as the agent is able to receive an average reward over 100 episodes of at least +30 in 188 episodes trained over 20 agents.



```
Episode 20      Average Score: 4.19     Episode Time 130.65 s
Episode 40      Average Score: 9.26     Episode Time 180.23 s
Episode 60      Average Score: 15.37    Episode Time 208.28 s
Episode 80      Average Score: 20.55    Episode Time 209.06 s
Episode 100     Average Score: 24.08    Episode Time 208.67 s
Episode 118     Average Score: 30.20    Episode Time 207.56 s
Environment solved in 18 episodes!      Average Score: 30.20
```



IDEAS FOR FUTURE

1. The learning could be optimized further through hyper-parameter optimization:
   a. RMS prop type momentum
   b. Learning rate adaptation through episodes
   c. Batch size adjustements through episodes
2. Using other algos
   a. Dueling DQN

REFERENCES

1. CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING, ICLR 2016
2. https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893