Udacity Deep Reinforcement Learning Nanodegree
Project 2
Continuous Control

In this project, we control 20 agents with double jointed arms to reach and maintain a moving target. We use the Deep Deterministic Policy Gradient Algorithm to train the agents.

Reacher Environment

In Reacher 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 an 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 project we solved a multi-agent version of Reacher environment. This version has 20 identical agents and arms, each with its own copy of the environment.

To solve the environment, an algorithm must get an average score of +30 over 100 consecutive episodes, and 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).
- The environment is considered solved, when the average (over 100 episodes) of those average scores is at least +30.

Reacher environment is provided by Unity Machine Learning Agents Toolkit.

DDPG Algorithm

We use the multi agent version of the Deep Deterministic Policy Gradient Algorithm to train the 20 agents. DDPG is an actor critic method for reinforcement learning over continuous action spaces.

The actor component takes in a state vector S and outputs a vector of continuous actions A. The actor is a feedforward neural network with two copies, one being used as a target for the other one.
The critic component takes in a state vector S and vector A of continuous actions as input and outputs the Q value for the given state and action pair, i.e Q(s,a). It is implemented as a feedforward neural network with two copies, one being used as a target for the other one.
During training, in order to encourage exploration, we add Ornstein-Uhlenbeck noise to the agent at every step. In order to encourage exploitation in the later stages of training, we reduce the amount of noise added by introducing a scaling factor epsilon which we decreases after every training loop.

Network Architecture

Actor Network – input : 33, Hidden Layer 1 : 400, Hidden Layer 2 : 300
Critic Network – input : 33, Hidden Layer 1 : 400, Hidden Layer 2 : 300

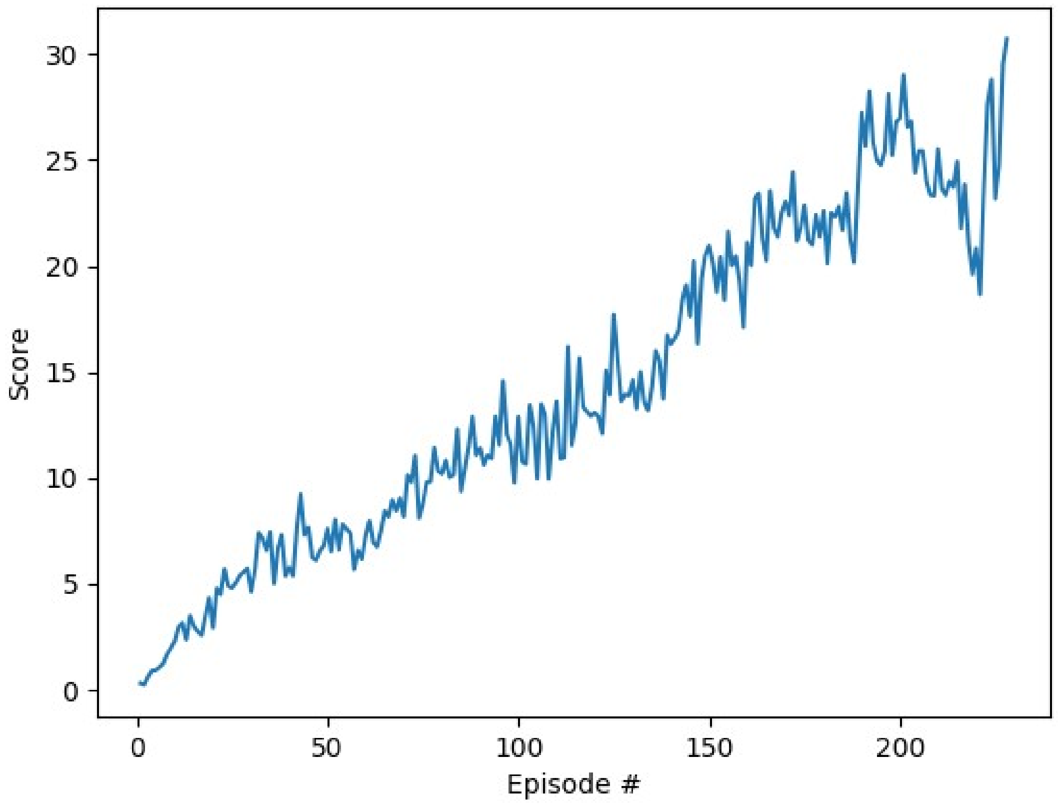
We initialized the weights of all layers using Glorot Initialization scheme.

We use the following hyperparameters for training the agents -

```
BATCH_SIZE = 64 # 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-4 # learning rate of the critic
WEIGHT_DECAY = 0 # L2 weight decay
Ornstein-Uhlenbeck process
mu=0., theta=0.15, sigma=0.2
#no of updates per step
self.n_updates = 20
#updating after how many steps
self.update_rate = 10
#scaling noise to increase exploitation at later stages
self.noise_scale = 1
#decay rate of noise scale
self.noise_decay = 0.995
#minimum noise scale
self.min_noise_scale = 0.01
```

Results

DDPG solves the environment in around 128 episodes. The graph belows shows the average score plotted as a function of number of episodes.



Ideas for future work

We wish to explore the role of Batchnorm layers in convergence of DDPG algorithm. Another idea for future work is benchmarking DDPG against TRPO, PPO, D4PG.