Background and requirements:

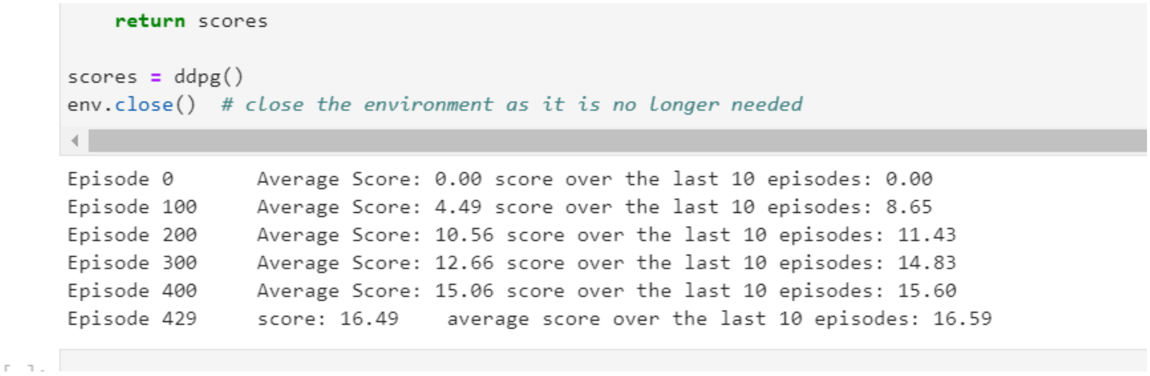
In this project, a reinforcement learning (RL) agent that controls a robotic arm need to be built within Unity's Reacher environment. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of your agent is to maintain its position at the target location 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.

Learning Algorithm

Policy-based, Value-based and Actor-Critic:

RL algorithms using neural networks as function approximators can take approaches with things they are estimating. Firstly, given a state as input they can estimate the expected future rewards for different possible actions. This approach is known as a **value-based method**. However, for tasks that involve continuous actions, a function approximator to estimate the actual policy directly could be applied. This approach is known as a **policy-based method**.

By applying the DDPG (Deep Deterministic Policy Gradient, Continuous Action-space) algorithm as an "Actor-Critic" method is introduced. The implementation of learning algorithm in this project is mainly based on the code in ddpg-bipedal code created by Udacity, but initial simple modified give very slow learning and cannot meet the requested requirements (see result below).



```
    return scores

scores = ddpg()
env.close()  # close the environment as it is no longer needed
```

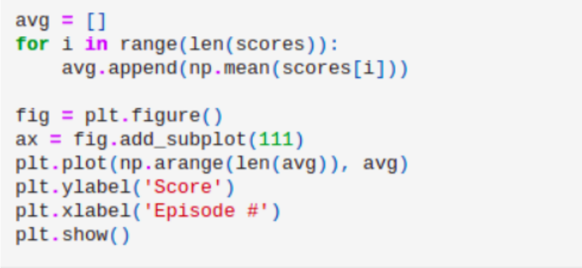
```
Episode 0       Average Score: 0.00 score over the last 10 episodes: 0.00
Episode 100     Average Score: 4.49 score over the last 10 episodes: 8.65
Episode 200     Average Score: 10.56 score over the last 10 episodes: 11.43
Episode 300     Average Score: 12.66 score over the last 10 episodes: 14.83
Episode 400     Average Score: 15.06 score over the last 10 episodes: 15.60
Episode 429     score: 16.49    average score over the last 10 episodes: 16.59
```

To improve the performance, several adjustments has been done on top of the ddpg-bipedal code.

1. On top of the Ornstein-Uhlenbeck process, introduce the epsilon parameter for noise decay
2. Fine-tune the theta (speed of mean reversion) and sigma (volatility) parameters for adding noise
3. Applying gradient clipping and normalization

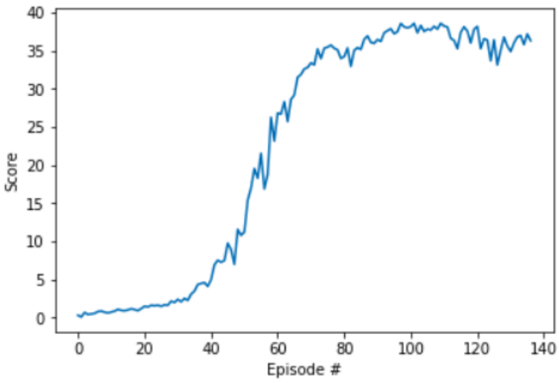
4. Adjusting learning interval, by learn every 20 steps with 10-time multiplication.

After these quick adjustments, the performance improved as shown in the plot, the agent is able to receive an average reward (over 100 episodes, and over all 20 agents) of at least +30.



```
avg = []
for i in range(len(scores)):
    avg.append(np.mean(scores[i]))

fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(avg)), avg)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```



Model architecture

# ACTOR NETWORK

Batch Norm Layer -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) Fully Connected Layer ->Tanh Activation()

# CRITIC NETWORK

Batch Norm Layer -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) -> Fully Connected Layer -> Leaky Relu(leakiness=0.01) Fully Connected Layer

Future ideas

It would be worth experiment with another algorithm for example DP4G, TRPO and PPO etc. Also would be good to have a try with prioritized experience replay, which could be found in this paper: https://cardwing.github.io/files/RL_course_report.pdf