Project 2: Continuous Control

Problem Statement

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.

Environment Analysis

- 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 is a continuous number between -1 and 1.
- The goal is for each agent to move hand to the location.
- The environment has 20 agents, all linked to the same brain
- The benchmark to qualify the solution as successful is to achieve the reward for mean score of at least: +30

Implementation Details

The approach followed was:

- Understand the state and action space
- Implementation of Deep Deterministic Policy Gradient Model (DDPG)
- Train the agent with the above model over different epochs till the target was achieved
- Plot the results of learning across episodes

Algorithm and Implementation

Since the environment has two critical complexities over previous project – Multiple Agents and Continuous action spaces, a policy-based method is suitable over value based method to achieve the end result.

For this project, the agents were trained on DDPG that uses Actor Critic method to approximate policy function required for our output.

The current implementation is over the top of resource mentioned in one of the Udacity practice exercise: https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum

The algorithm DDPG (https://arxiv.org/pdf/1509.02971.pdf) builds on top of DQN by adding an Actor which estimates optimal policy with max reward using gradient ascent where as a Critic will estimate value using value-based approach – thus complementing learning of actor and enhancing speed of policy estimation.

Another improvement over previous project is use Ornstein Uhlenbeck Noise parameters with decay over the previously traditional epsilon greedy approach considering the continuous movement of the action space.

The gradient is also clipped in my Critic learner to prevent the issue of exploding gradients, a traditional problem in gradient based machine learning.

Finally, the first connected layer of NN was batch normalized to speed up the computation (as I saw that the training times were huge and learning was slow).

Architecture:

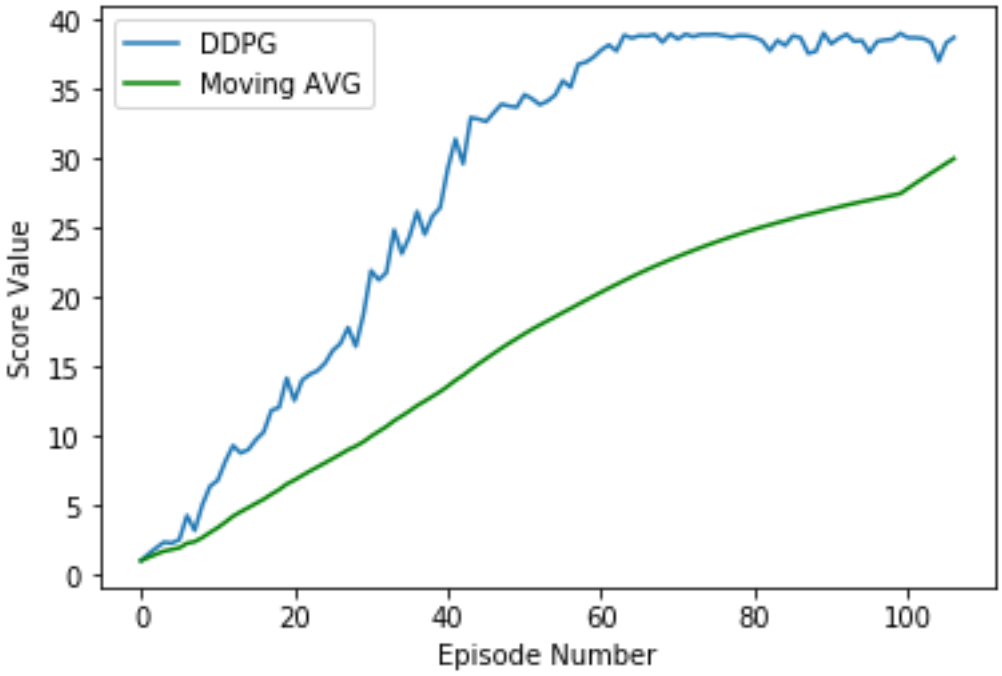
- The Neural Network used for both Actor and Critic are fully connected Neural Network with 3 layers:
  First layer: 400 neurons, Second layer: 300 Neurons, Third layer: Size of Action Space.
- All input connected layers were relu active, with output for first connect layer batch normalized
- The DDPG used Ornstein Uhlenbeck noise process
- Experience Replay was used to allow agent to learn from previous experiences (like how we did in DQN impmenetation). The replay used a buffer and fetched experiences at random.

Hyperparameters Used for learning:

Replay buffer size = 1e5
Mini Batch Size = 64
Discount factor (Gamma) = 0.99
Soft update for Target Parameters (Tau) = 1e-3
Learning Rate for actor (For Adam Optimizer) = 1e-3
Learning Rate for critic (For Adam Optimizer) = 1e-3
L2 weight decay = 0
Update Interval = 20
Learning passes = 10
Epsilon(for eps greedy action selection) = 1.0
Epsilon Decay = 1e-6
Ornstein Uhlenbeck Noise Sigma = 0.2
Ornstein Uhlenbeck Noise Theta = 0.15

Results

The algorithm was able to achieve the goal in 107 Episodes (Over 100 episodes and at least 30+ average reward, as specified in the project rubric: https://review.udacity.com/#!/rubrics/1890/view) for 20 agents.



Output plot for scores vs episode number for mean and moving average using DDQN algorithm

Enhancements for the future

- Add more layers to neural network to improve the efficiency of the training
- Tweak hyperparameters or use hyperparameter search to ensure learning is achieved quickly with better performance
- Enhance the RL learning model with Prioritized Experience replay instead of choosing experiences at random
- Try out different algorithms like PPO, D4PG, TRPO which also help in obtaining solution optimally