Report for Udacity Reinforcement Learning Nanodegree - P2 Continuous Control

Project Setup and Learning Algorithm

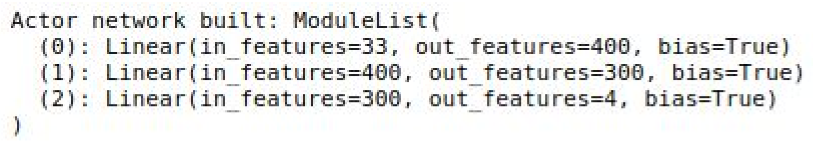
This project involves training a RL agent that controls a double jointed arm. The goal of the agent is to move the arm to a target location. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. So the agent gets trained to maintain its position at the target location 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.

There are two environments available - one with a single agent and the other with multiple (20) agents. This submission uses the multiple agent environment. For the multiple agent environment, the scores from all the agents are averaged and envtt is considered solved when this average score is +30 over 100 consecutive episodes.

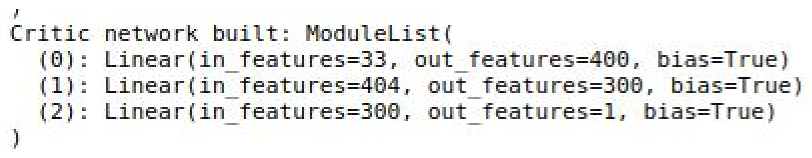
The Deep Deterministic Policy Control (DDPG) has been used as the Reinforcement Learning Algorithm. DDPG agent uses an actor and a critic network to learn policies in high-dimensional, continuous action spaces. Here the critic is a value based learner and actor is a policy based learner. The actor network specifies an action based on the current state of the environment. The actor returns a deterministic actions which is the best action under the current state. The critic network calculates an advantage to evaluate the actions made by the actor.
My implementation of this project uses the Agent , OUNoise and the Replay Buffer class provided in the project DDPG-Pendulum. I have made some modifications the neural network used and the functions to train the DDPG agent to apply to this use case.

I found good success by using a relatively simple neural network that also happens to be faster to train. The actor neural network consists of 2 linear layers, the first with 400 neurons and the second with 300 neurons. The actor network takes as input the state size of 33 and the output layer has 4 neurons corresponding to the 4 actions. The Actor network uses RELU as the activation function for the linear layers. The final layer uses tanh as the activation function to squash outputs b/w -1 and +1.
I found that the performance of the actor network improved significantly after I used Batch Normalization in the actor network.
The Actor Network is shown below:



```
Actor network built: ModuleList(
  (0): Linear(in_features=33, out_features=400, bias=True)
  (1): Linear(in_features=400, out_features=300, bias=True)
  (2): Linear(in_features=300, out_features=4, bias=True)
)
```

The critic neural network consists of 2 linear layers, the first with 400 neurons and the second with 300 neurons. The action predictions from the action network are concatenated with the first layer of the critic neural network. The critic network takes as input the state size of 33 and the output layer has 1 neuron. The critic network uses RELU as the activation function for the linear layers. The final layer uses sigmoid as the activation function.



```
,
Critic network built: ModuleList(
  (0): Linear(in_features=33, out_features=400, bias=True)
  (1): Linear(in_features=404, out_features=300, bias=True)
  (2): Linear(in_features=300, out_features=1, bias=True)
)
```

The mean square loss is used as the loss function to train the critic neural network. Adam is used as the optimizer with a learning rate of 1e-3. The actor network is also trained using the Adam Optimizer with a learning rate of 1e-3. Soft update with tau = 1e-3 is used to update the weights of the target actor and critic networks.

To do exploration, DDPG adds random noise to the action predictions from the actor network. The noise is reduced over time to limit the amount of exploration as the agent learns.

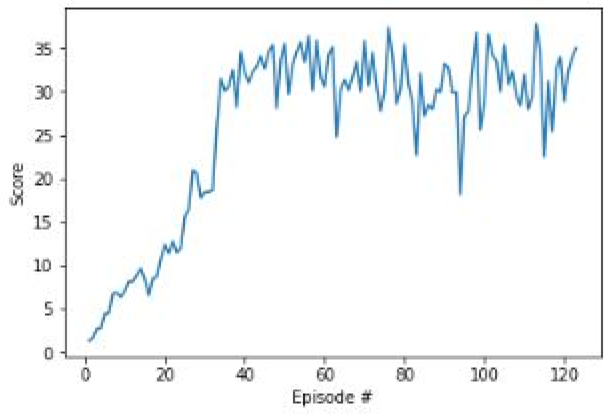
Since I have used the multiple agent environment, the state, action, reward, next state, done from all the agents is stored in the replay memory. A batch size = 128 experiences are randomly picked up to train the neural network.

Training the Neural Network

The DDPG Network was trained for max of 2000 steps or till the average reward of 30 over 100 episodes was reached. It took the model 123 episodes to reach an average reward > 30. The plot of score by epoch is shared below:

```
Episode 108 (02m10s)    Score: 29.56    Moving average: 26.85
Episode 109 (02m10s)    Score: 28.44    Moving average: 27.07
Episode 110 (02m10s)    Score: 32.00    Moving average: 27.32
Episode 111 (02m10s)    Score: 27.98    Moving average: 27.52
Episode 112 (02m10s)    Score: 29.54    Moving average: 27.73
Episode 113 (02m10s)    Score: 37.79    Moving average: 28.02
Episode 114 (02m10s)    Score: 34.31    Moving average: 28.27
Episode 115 (02m10s)    Score: 22.54    Moving average: 28.41
Episode 116 (02m10s)    Score: 31.22    Moving average: 28.66
Episode 117 (02m10s)    Score: 25.45    Moving average: 28.83
Episode 118 (02m10s)    Score: 32.67    Moving average: 29.06
Episode 119 (02m10s)    Score: 33.97    Moving average: 29.30
Episode 120 (02m10s)    Score: 28.85    Moving average: 29.46
Episode 121 (02m10s)    Score: 32.27    Moving average: 29.67
Episode 122 (02m10s)    Score: 33.99    Moving average: 29.88
Episode 123 (02m10s)    Score: 35.01    Moving average: 30.12

Environment solved in 123 episodes!     Average Score: 30.12
```



Once training was finished, the best weights were saved. I then tested the trained agent on Unity Environment by playing games. The agent was always able to get a score of at least 30 points.

Opportunities for Improvement

The performance of DDPG agent can be further improved. Some of these improvements include:

1. Prioritized Experience Replay - The idea behind Prioritized Experience Replay is to sample events from the replay buffer based on importance probabilities instead of randomly. The intuition is that some past experiences that occur infrequently may be more important for learning. Past experiences with higher TD error delta could be given higher priority
2. Experiment with other algorithms - While DDPG is a good algorithm, tuning the DDPG algorithm required a lot of trial and error. Perhaps another algorithm such as Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO) or Distributed Distributional Deterministic Policy Gradients (D4PG) would be more robust.