Report Project2: Continuous Control

1. Introduction:

The class of Reinforcement Learning (RL) algorithms that output a policy are called Policy Based Learning Algorithms. This class of Policy-Based algorithms are different from Q-Learning or Deep Q-Network (DQN), the continuous state function approximation equivalent of Q-Learning.

Why do we need to use Policy Based Learning Algorithms if we have DQN? DQN (or Q-Learning, SARSA etc.) work for only discrete action spaces. But there are problems where we the actions are a continuous value over a defined space. For example, the degree to which we want to steer a car to left is a continuous valued action. We may discretize the action space and use DQN, but that doesn't seem like a very robust approach, especially when the degrees of freedom increase. Hence, we turn to Policy-Based Learning Algorithms.

2. Double Deterministic Policy Gradient (DDPG) Algorithm:

DDPG is an off-policy actor-critic algorithm using deep function-approximators that can learn policies in high-dimensional, continuous action spaces [1]. DDPG combines actor-critic based methods with DQN. The policy $\mu(s|\theta^{\mu})$ is estimated by the actor. This policy maps an input state vector to an action. Since the policy can have a lot of variance, the critic, DQN, is used to reduce the variance of the actor. The critic learns the $\mathrm{Q}(\mathrm{s},\mathrm{a})$ function.
Just like in DQN, a mini-batch of transitions is selected from the Replay Buffer. At each time step, after sampling the transitions, the actor and critic learn and update the neural network weights. Similar to DQN, in DDPG, both the actor and critic have 2 sets of neural networks. The weights of the target actor and critic neural networks are learned using a soft update rule.

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

And finally, while sampling an action noise (N) was added as a means to provide for exploration.

$$\mu' = \mu(s_t|\theta^{\mu}{}_t) + N$$

Where, N is the Ornstein-Uhlenbeck process noise. This noise has 3 parameters: Mu, Sigma and Theta.

2.1 Actor Neural Network Architecture:

The input layer takes the state observation, 33 units. There are 2 hidden layers, 1st hidden layer has 200 units and 2nd hidden layer has 100 units. And the output layer represents the continuous action vector, with 4 units. ReLu activation was used after each hidden layer. After the 1st hidden layer batch normalization was used to reduce the co-variance shift [2]. And finally, at the end of each output layer, tanh activation was used to maintain the action value in the range -1 to 1.

2.2 Critic Neural Network Architecture:

The critic neural network is used to get the Q value of the corresponding action that the actor neural network determines. Hence, the critic neural network takes both the state and action to estimate the Q value. The critic neural network has 2 hidden layers, 1st with 200 units and 2nd with 100 units. Note that the input layer takes the state observation vector. And the 1st hidden

layer we use batch normalization to reduce the co-variance shift. The 2nd hidden layer takes the outputs of the 1st hidden layer and concatenates the action vector and takes the concatenated vector as input and outputs 100 units. Note that ReLu activation is used after both the hidden layers. Note that the output layer doesn't need activation since the critic NN is used to estimate the Q value.

2.3 Hyperparameters:

The following were the optimal tuned hyperparameter settings:

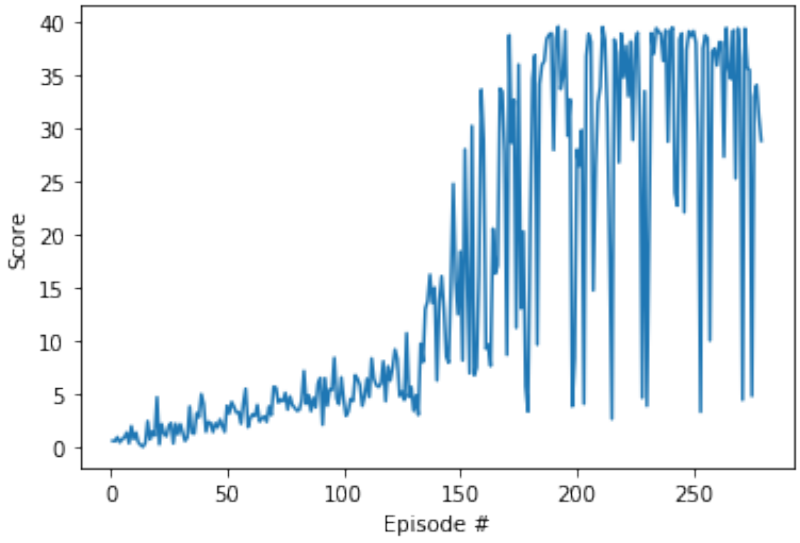
Hyperparameter	**Value**
Replay Buffer Size	1e6
Batch Size	1024
Gamma	0.99
Tau (Soft Update rate)	1e-3
Actor Learning Rate	1e-4
Critic Learning Rate	1e-3
OUNoise Mu	0
OUNoise Theta	0.15
OUNoise Sigma	0.05

2.4 Pseudocode:

Pseudocode is on Page 5 of 1st reference (Continuous control with deep reinforcement learning). The code was based from Udacity's Deep Reinforcement Learning Repo [3]. Modifications were made to the NN architectures in the model.py file.

3. Results:

The environment was solved in 279 episodes. Following is the plot of the rewards:



4. Ideas for future work:

- Critic neural network architecture modification. Explore the feasibility of having both the state and action vector as a concatenated vector to the input layer, as opposed to feeding the action vector after input layer, as input to the 1st hidden layer.
- Use other techniques like Proximal Policy Optimization (PPO) and REINFORCE to solve this problem. And to benchmark multiple solutions with DDPG.
- Express the problem as a Policy optimization problem. Use a standard Neural Network to express a policy function. And see how various optimization techniques, like Randomized Hill Climbing, Simulated Annealing etc., solve the environment.

5. References:

1 - Continuous control with deep reinforcement learning, Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra https://arxiv.org/pdf/1509.02971.pdf
2 - https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
3 - https://github.com/udacity/deep-reinforcement-learning/tree/master/ddpg-pendulum