Project 2: Continuous Control

---

1 Introduction

For this project, I have trained an Agent using the Deep Deterministic Policy Gradient algorithm which is based on the Actor Critic Methods. 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.

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.

I trained the second version which contains 20 identical agents, each with its own copy of the environment.

2 Dependencies

1. Download the environment from one of the links below. You need only select the environment that matches your operating system:
    - *Version 2: Twenty (20) Agents*
        - Linux: click here
        - Mac OSX: click here
        - Windows (32-bit): click here
        - Windows (64-bit): click here

2. (*For Windows users*) Check out this link if you need help with determining if your computer is running a 32-bit version or 64-bit version of the Windows operating system.

   (*For AWS*) If you'd like to train the agent on AWS (and have not enabled a virtual screen), then please use this link (version 1) or this link (version 2) to obtain the "headless" version of the environment. You will not be able to watch the agent without enabling a virtual screen, but you will be able to train the agent. (*To watch*

the agent, you should follow the instructions to enable a virtual screen, and then download the environment for the Linux operating system above.)

The environment can be configured as

env = UnityEnvironment(file_name='/data/Reacher_Linux_NoVis/Reacher.x86_64')

3 How to Execute this solution

Follow the solution in the file **Navigation Project Solution v2.ipynb** to train the agent to pick up the Yellow bananas.

4 Implementation Approach

The Deep Deterministic Policy Gradient algorithm with Experience Replay is implemented. The details of the algorithm are given in this research paper.

https://arxiv.org/abs/1509.02971

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

The following steps were taken to implement the algorithm

4.1 Define the class for the Ornstein-Uhlenbeck Noise

We define a class for the Ornstein - Uhlenbeck Noise. The Ornstein-Uhlenbeck process, or OU process, is a stochastic process that models mean-reverting behavior, meaning it tends to revert to a long-term mean, while also experiencing random fluctuations.
The OU Noise class has been defined as given in the class example.

[Reference:
https://en.wikipedia.org/wiki/Ornstein%E2%80%93Uhlenbeck_process#:~:text=In%20mathematics%2C%20the%20Ornstein%E2%80%93Uhlenbeck,Ornstein%20and%20George%20Eugene%20Uhlenbeck.]

4.2 Define the class for Experience Replay

I used the example given in the class to create Experience Replay which is the Replay Buffer for state transitions along with actions, rewards.

4.3 Define the Actor and the Critic Neural Networks

I define the Actor and the Critic Neural Networks with 128 neurons in the first hidden layer and 256 neurons in the second hidden layer, with batch normalizations and the dropout implemented.

The following hyperparameters were considered for the DQN Agent training.

BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 128 # minibatch size
GAMMA = 0.95 # discount factor
TAU = 1e-3 # for soft update of target parameters
LR_ACTOR = 1e-4 # learning rate of the actor
LR_CRITIC = 1e-3 # learning rate of the critic
WEIGHT_DECAY = 0 # L2 weight decay
TRAIN_EVERY = 20 # How many iterations to wait before updating target networks
NUM_AGENTS = 20 # How many agents are there in the environment
SEED=12345

4.4 Defining the Agent

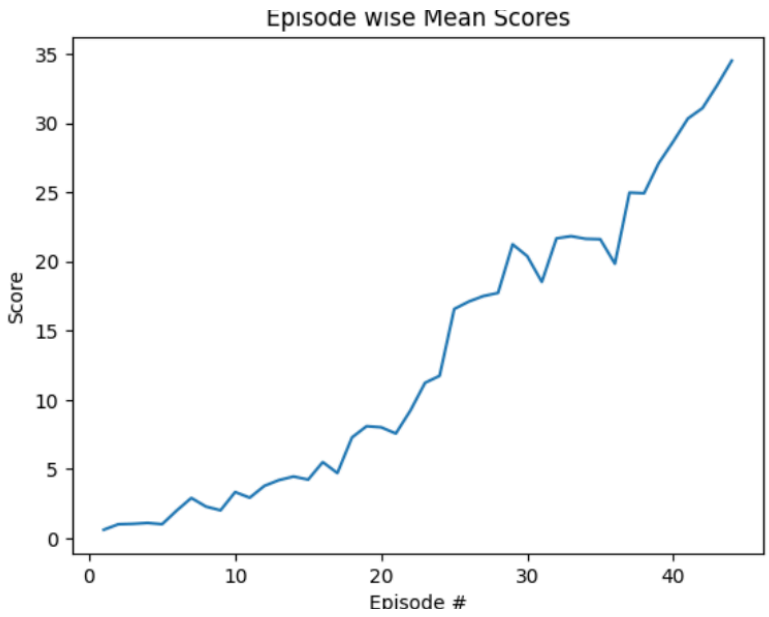
I define the Agent as given in the class. I have added the Gradient Clipping for the Critic as suggested.

4.5 Main DDPG Function

Main DDPG function is defined as mentioned in the class for the training of the algorithm.

4.6 Plot the Scores

Using the Matplotlib library, I plotted the scores. The score curve looks as below



5 Solution Details

1. Continuous_Control-MultipleAgentsv4.ipynb : The main file that contains the OU Noise, Experience Replay Buffer, Actor and Critic Neural Networks, DDPG Agent.
2. checkpoint_actor.pth - Saved model weights for the Actor Neural network
3. checkpoint_critic.pth - Saved model weights for the Critic Neural Network
4. Readme.md
5. Report Continuous Control.pdf

6 Ideas for Future Work

I would like to explore the following as ideas for future work

a. Explore Hyperparameter tuning.
b. Explore algorithms such as D4PG and PPO.