Continuous Control Project

1. Introduction:

This is an Implementation of a deep reinforcement agent for a double-jointed arm that can move to target locations. The goal is to maintain the position of the arm in the location of the target for as long as possible. 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.

2. Environment:

- The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm.
- The action space is 4 dimensional vector, action = $[x_1, x_2, x_3, x_4]$ where $x_i \in [-1, +1]$ with $\in \{1,2,3,4\}$, corresponding to torque applicable to two joints. Every entry in the action vector must be a number between -1 and 1.
- This task is episodic and the environment is considered to be solved if we reach an average reward of 30 or plus over 100 episodes.

3. The algorithm:

- For this task I chose to use the *The Deep Deterministic policy gradient (DDPG)* algorithm over Deep Q Network (DQN). Because DDPG works on continuous state and action space whereas DQN is meant to solve discrete action space problems.
- Critic Value Model computes the Q-values for any given (state, action) pair. After that, the gradient of this Q-value is computed with respect to the corresponding action vector which is then fed in as input for the training of the Actor Policy Model. Since there are two networks, that's why the model is sample efficient.
- The agent uses two networks:
  i) The actor network: given the current state the actor is trained to output an approximation of the optimal policy action(i.e.) (Argmax of probabilities of actions of a given state)
  ii) The Critic network: this one learns to evaluate the optimal action value function by using the actors best believed action.
- For both the Critic and the Actor neural networks I used 4 layers of fully connected neural networks.
- **The reply buffer** : Just like the DQN , this algorithm uses a reply buffer and sample from it randomly in order to break the correlation that exist because of the consecutive experiences .

- **Soft Updates**: In DDPG we have 4 networks:
  i) A regular (local) copy of actor network.
  ii) A target copy of the actor network.
  iii) A regular (local) copy of the critic network.
  iv) A target copy of the critic network.
- The regular network is the most up to date network since it's the one used for training and the target network is the one used for prediction to stabilise the train.
- For the soft update works we mix in 0.01% of the regular network weights with targets network weights.

Hyper-parameters:

- GAMMA = 0.99 # discount factor
- TAU = 1e-3 # soft update of target network weights
- LR_ACTOR = 1e-4 # learning rate of the actor
- LR_CRITIC = 1e-4 # learning rate of the critic
- BUFFER_SIZE = int(1e5) # reply buffer
- BATCH_SIZE = 128 # minibatch size
- WEIGHT_DECAY = 0 # L2 weight decay

I have used Continuous control with deep reinforcement learning research paper hyper-parameter values.

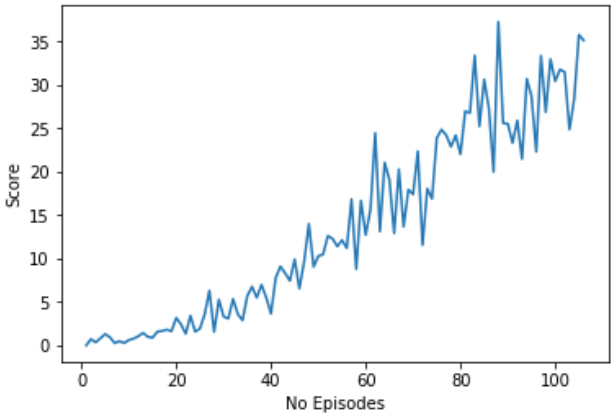
- n_episodes (int): maximum number of training episodes
- max_t (int): maximum number of timesteps per episode
- num_agents: number of agents in the environment

Where `n_episodes=200`, `max_t=1000`

Upon running this numerous times it became apparent that the environment would return done at 1000 timesteps. Higher values were irrelevant.

Results:

Environment was solved in 106 episodes with an average reward of 31.13.
The following is a plot showing the evolution of rewards during the training.



5. Improvements:

There are several ways to improve this task among them:

- Trying out algorithms like PPO, A3C, and D4PG that use multiple (non-interacting, parallel) copies of the same agent to distribute the task of gathering experience.
- As mentioned in this paper the DDPG requires a large number of training episodes to find a solution and this was observed during the training , So trying a model free approach may help.
- Trying out dropouts, various weight initialization and further hyper-parameters like cost functions that may yield better results.