P2 Continuous Control

Introduction:

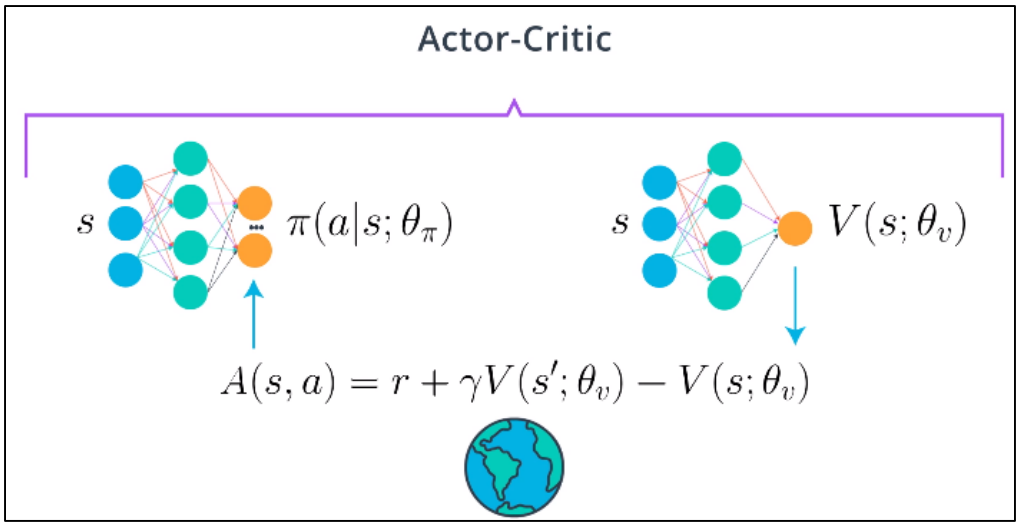
For this project, I will work with the Reacher environment. https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Examples.md#reacher

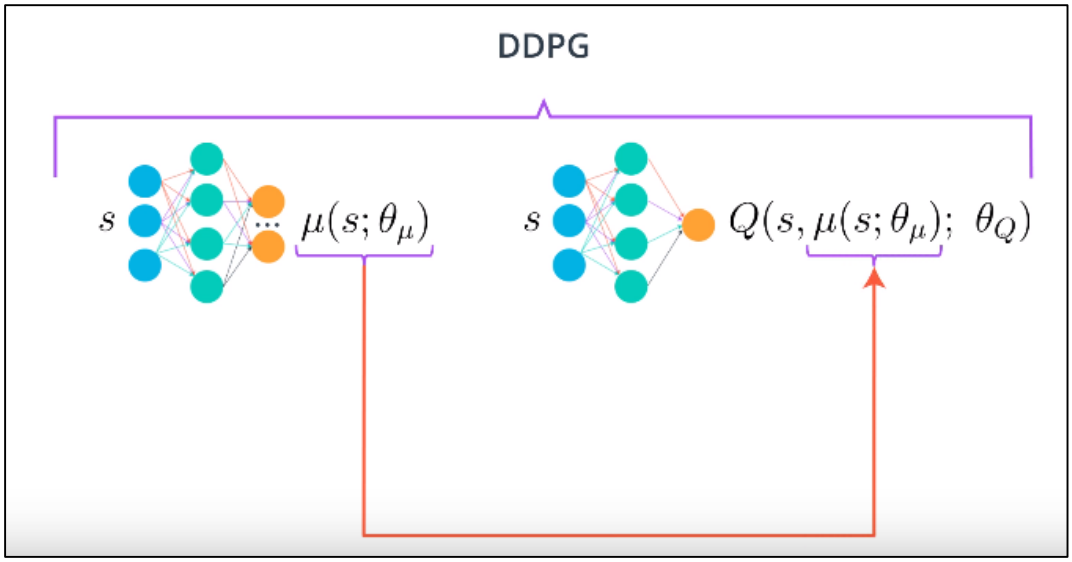
In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.

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.

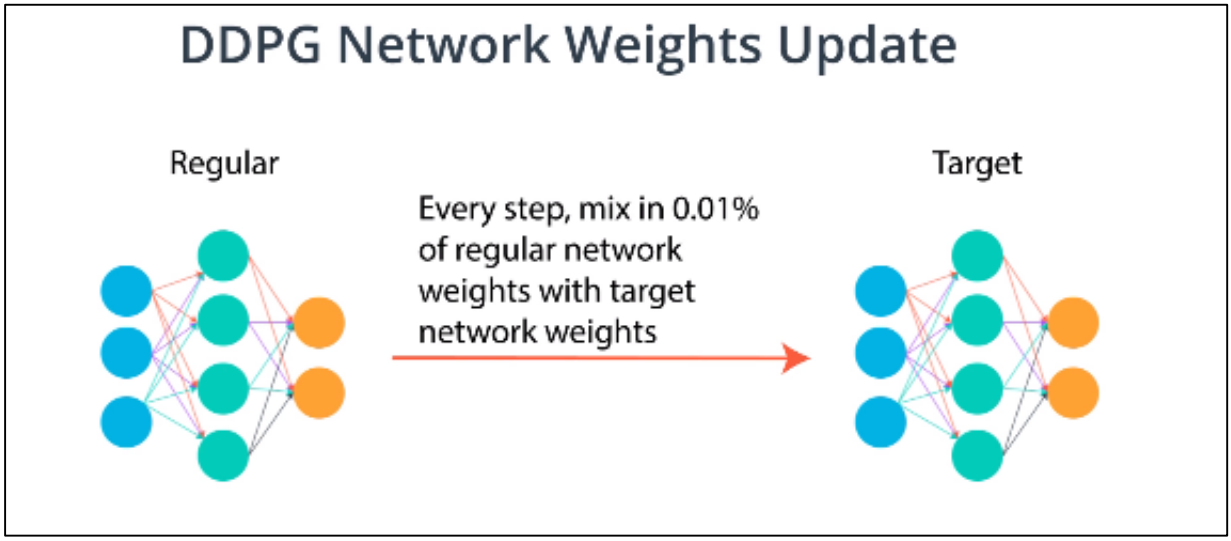
DDPG: Deep Deterministic Policy Gradient, Continuous Action-space

The DQN network can be used as long as the action space is discrete however our environment has continuous action space. Here I am using DDPG which is one of Actor-Critic Methods.





Both actor and critic has 2 copies, local and target and in each step we use soft update to modify the target weights.



In Reinforcement learning for discrete action spaces, exploration is done via probabilistically selecting a random action (such as epsilon-greedy or Boltzmann exploration). For continuous action spaces, exploration is done via adding noise to the action itself (there is also the parameter space noise but we will skip that for now). In the DDPG paper, the authors use *Ornstein-Uhlenbeck Process* to add noise to the action output (Uhlenbeck & Ornstein, 1930):

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$$

Training Procedure:

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:

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

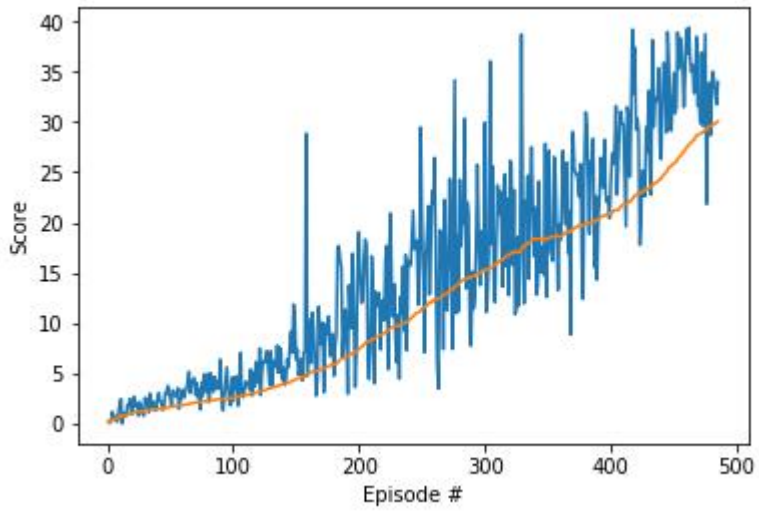
  end for
end for

Reference:

https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b

Training Results:

I was able to solve the environment in 385 episodes:



Ideas for Future Work:

My Hyper parameter tuning took huge time to achieve the mention above results, I had totally different results by tuning the seed, the training on GPU was converging slower compared to GPU also batch size was giving good results at 128.

I am planning to further optimize my hyper parameter to solve it in less number of episodes by tuning the noise parameters and using bigger neural network.

I also having plan to solve it using other Actor-Critic network and solve the 20agents version.