Continuous Control using Reinforcement Learning

Task Overview:

In this project, we train double-jointed arms to 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 the agents is to maintain its position at the target location for as many time steps as possible. We are training with 20 agents simultaneously, as shared experience replay helps in making the training faster.

State and action space:

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.

Learning Algorithm:

We use the DDPG algorithm to train actor critic networks.

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

Network Structure:

It consists of 2 networks:

Actor Network:

We've used a fully connected neural network with 3 layers that takes in 33 inputs (the current state of the agent) and generates 4 outputs (the action values). We have used relu activation function after the first two layers and tanh activation after the 3rd layer.

Critic Network:

This is a slightly different network where it takes the state and action as input and directly gives us the Q-value. Initially the states are fed into the first fully connected layer of the network followed by relu activation. The output of the previous step is concatenated with action and passed through another 2 fully connected layers.

Replay Buffer:

We have used a replay buffer that utilizes the concept of experience replay. We gather the experiences of the agent in a deque. The deque size is set to 100000. We randomly sample a mini-batch of size 1024 from the buffer pool and do a supervised training of the online network against the target network. Finally, a soft update is done by copying the parameters of the online network to the target network.

Ornstein-Uhlenbeck process:

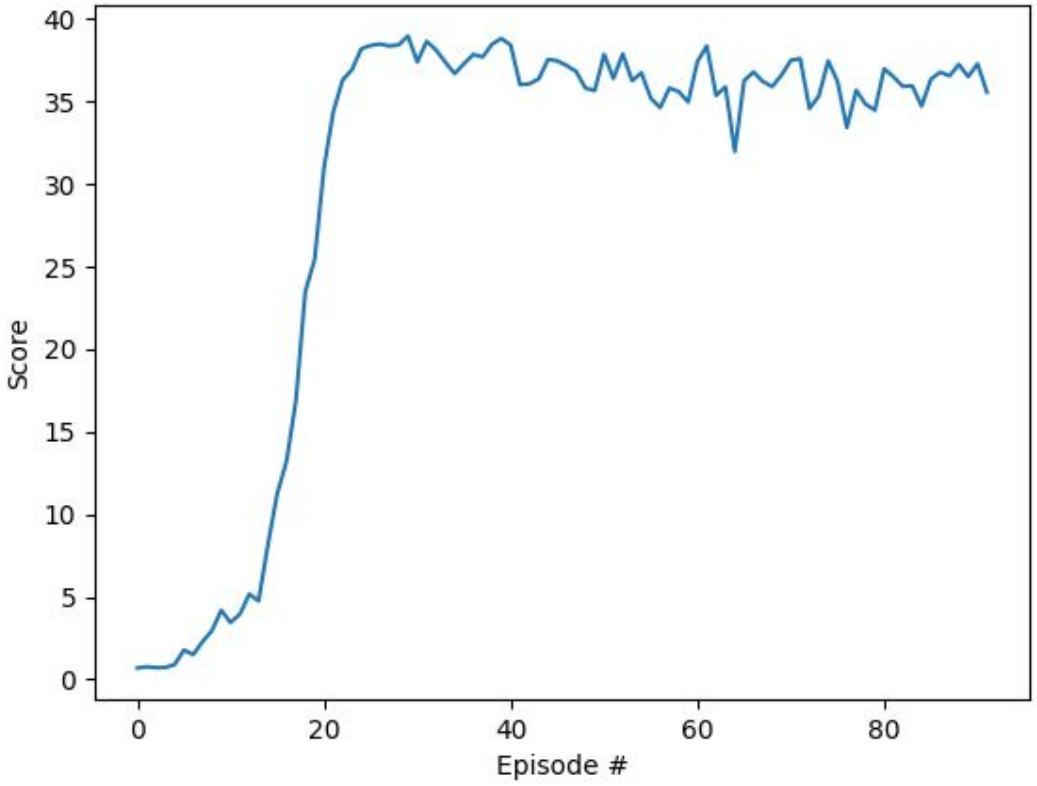
We introduce an additional noise to the actions before interacting with the environment, this encourages exploration of the agents.

Training details:

The learning rate of training the actor network and the critic network was set to 1e-3. The discounting factor was set to 0.99. The average reward is calculated for the last 100 episodes of the training and monitored.

Results:

The agents received an average score of 30.04 in the 92^{nd} episode.



Ideas for future work:

Prioritized Experience Replay could be implemented to further enhance the performance of the agent, so as to replay important transitions more frequently, and therefore learn more efficiently. We can assign priority weights for each transition step, which will be directly proportional to the TD error of the actor. Instead of randomly sampling from the buffer pool, we could sample based on the priority weights.