1 Problem statement

In this project, I train an agent to move a double-jointed arm 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 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 the 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.

The task is episodic, and in order to solve the environment, the agent must get an average score of +30 over 100 consecutive episodes.

2 Algorithgm

In this project, the Deep Deterministic Policy Gradient (DDPG) algorithm is used to train the agent. The pseudocode for this algorithm is provided below.

Algorithm 1: : Deep Deterministic Policy Gradient (DDPG).

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s, a|\theta^\mu)$ 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 data 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, r_{t+1}) in 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 $L\frac{1}{N}\sum_i(y_i - Q(s_i, a_i|\theta^Q))$;
    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
end

3 Hyperparameters

The actor and critic are both feedforward neural networks. Each network consists of 4 layers (an input layer, two hidden layers, and an output layer). The actor turns a state into an action using 33 input units (continuous values representing a state), 128 and 256 units for the first and second hidden layer, and 4 output units that resemble the continuous actions for both joints of the arm. The critic has a similar structure. Its input layer also consists of 33 input units which are fully connected with the first hidden layer of size 128. The results of this first matrix multiplication and activation are concatenated with the output of an actor-network and fully connected with a second hidden layer of size 256. Finally, the critic network yields a single value that resembles the action value.

Both networks use ReLU for their hidden layers. The actor uses tanh to obtain values in the valid range of actions. Adaptive Momentum (Adam) algorithm is used for updating the parameters of the neural network. There are always two copies of a network, referred to as the local and the target. The local versions are updated every few steps along with a soft update on the target versions.

Other hyperparameters for the DDPG algorithm are as follows

- $M = 1000$
- $T = 1000$
- $N = 500000$
- Batch size $= 512$
- $C = 10$
- Actor learning rate $= 3 \times 10^{-4}$
- Critic learning rate $= 1 \times 10^{-3}$
- $\gamma = 0.99$
- $\tau = 1 \times 10^{-3}$

4 Results

The agent was able to learn the policy within 424 episodes with threshold criteria of an average score of +30 over 100 consecutive episodes.

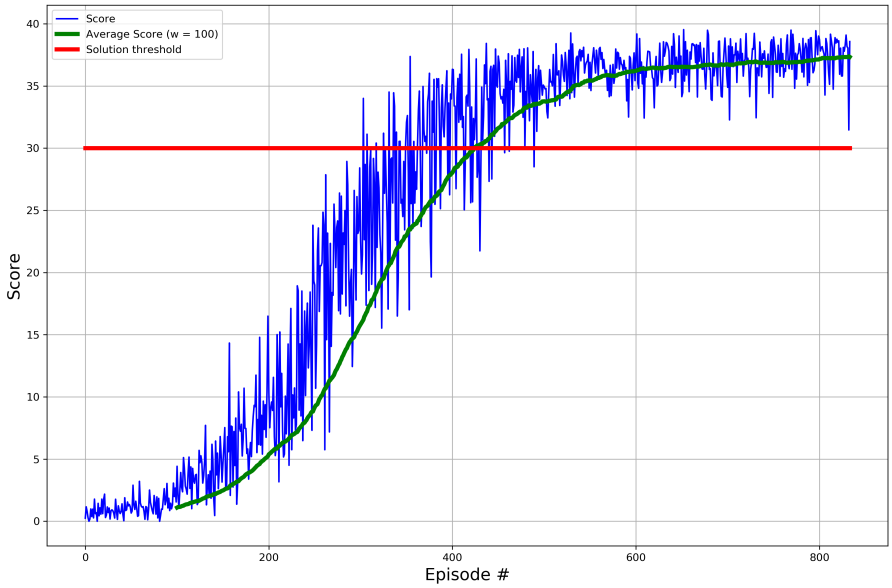

Figure 1: Trajectory of the reward function

5 Ideas for future work

Some of the potential ideas for future work are given below:

- Optimize hyperparameters of the DDPG algorithm using methods like Bayesian optimization
- Increase the resolution by shifting from a 33-dimensional state representation to a more accurate state resolution of the system and then use a convolutional neural network followed by the fully-connected layers as the agent.
- Train the agent using distributed training with 20 agents.