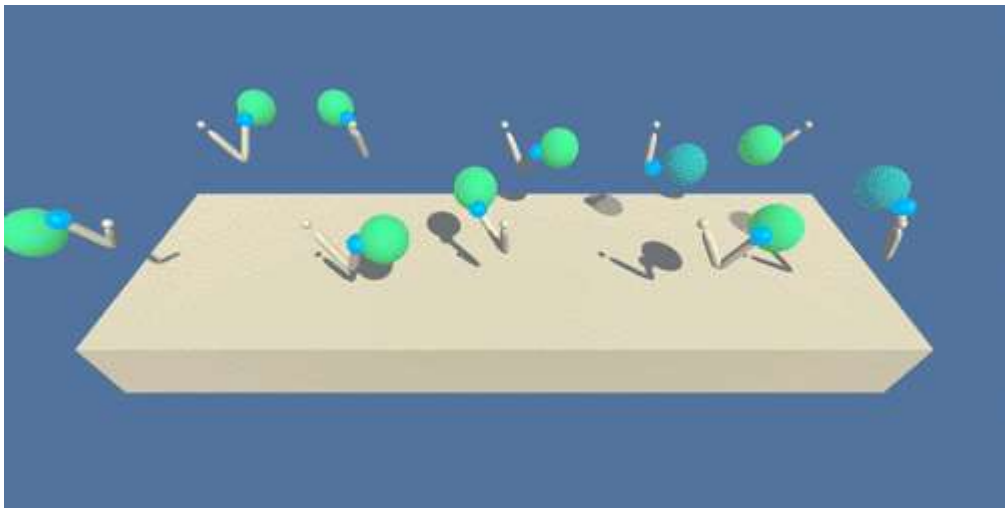
Project 2: Continuous Control

Introduction

We will work with the Reacher environment.



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.

Algorithm

In order to solve this challenge, I have used the Deep Deterministic Policy Gradient algorithm (DDPG). Deep Deterministic Policy Gradient (DDPG) is a model-free off-policy algorithm for learning continuous actions.

It combines ideas from DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network). It uses Experience Replay and slow-learning target networks from DQN, and it is based on DPG, which can operate over continuous action spaces.

Just like the Actor-Critic method, we have two networks:

Actor - It proposes an action given a state.

Critic - It predicts if the action is good (positive value) or bad (negative value) given a state and an action.

DDPG uses few more techniques not present in the DQN

Tt uses two Target networks to add stability to training

To implement better exploration by the Actor network, we use noisy perturbations, specifically an Ornstein-Uhlenbeck process for generating noise. It uses samples noise from a correlated normal distribution.

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

Critic loss - Mean Squared Error of y - Q(s, a) where y is the expected return as seen by the Target network, and Q(s, a) is action value predicted by the Critic network. y is a moving target that the critic model tries to achieve; we make this target stable by updating the Target model slowly.

Actor loss - This is computed using the mean of the value given by the Critic network for the actions taken by the Actor network. We seek to maximize this quantity.

Neural network architecture:

state_size: Dimension of each state

action_size: Dimension of each action

fc1_units: int. Number of nodes in first hidden layer – 128 Units

fc2_units: int. Number of nodes in second hidden layer – 128 Units

We set same for both actor and critic network

Other parameters are

BUFFER_SIZE = int(1e5) # replay buffer size

BATCH_SIZE = 128   # minibatch size

GAMMA = 0.99   # discount factor

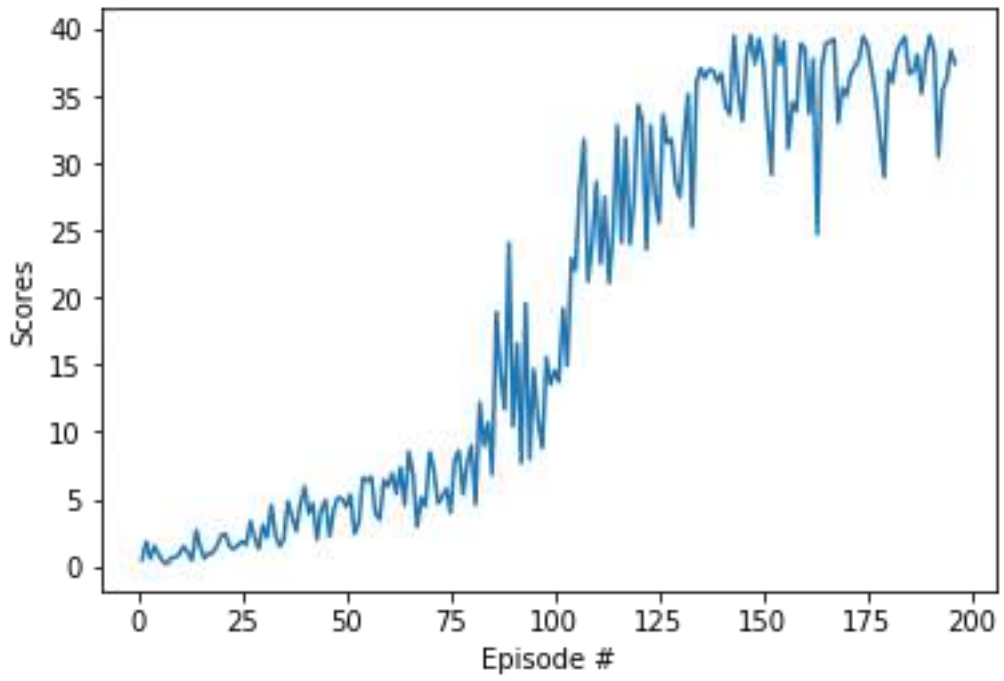
TAU = 1e-3   # for soft update of target parameters

LR_ACTOR = 2e-4   # learning rate of the actor

LR_CRITIC = 2e-4   # learning rate of the critic

WEIGHT_DECAY = 0   # L2 weight decay

Here is the training graph



The environment is considered solved, when the average (over 150 episodes) of those average scores is at least +30. In the case of the plot above, the environment was solved at episode 125, since the average of the average scores from episodes 125 to 200 (inclusive) was greater than +30.

Future :

GAE - Generalized Advantage Estimation

A3C - Asynchronous Advantage Actor-Critic

A2C - Advantage Actor-Critic

PPO - Proximal Policy Optimization

D4PG - Distributed Distributional Deterministic Policy Gradients

Refrences :

https://keras.io/examples/rl/ddpg_pendulum/