Project Continuous Control

Introduction

The aim of the project is to train an agent to move a robotic arm using continuous control. We use the Reacher Environment from Unity for the project (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 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 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.

Solution

The algorithmic approach used to solve the problem is to use the Deep Deterministic Policy Gradients (DDPG) algorithm. DDPG uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a target policy network. DDPG is an Actor-Critic method and in DDPG, the Actor directly maps states to actions. The target networks are time-delayed copies of their original networks that slowly track the learned networks. Using these target value networks greatly improve stability in learning.

The algorithm uses the following main components:

1. Experience replay: we save all the experience tuples (state, action, reward, next_state) and store them in a finite-sized cache — a “replay buffer.” Then, we sample random mini-batches of experience from the replay buffer when we update the value and policy networks.

2. Actor & Critic network updates: The updated Q value is obtained by the Bellman equation. In DDPG, the next-state Q values are calculated with the target value network and target policy network. Then, we minimize the mean-squared loss between the updated Q value and the original Q value.

3. Target network updates: We make a copy of the target network parameters and have them slowly track those of the learned networks via soft updates

4. Exploration: For continuous action spaces, exploration is done via adding noise to the action itself. We use *Ornstein-Uhlenbeck Process* to add noise to the action output (same as used by the original DDPG authors)

The flowchart for the DDPG algorithm [Lillicrap et al, 2015] is given below:

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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 Architectures

The Actor networks use two fully connected layers with 256 and 256 units with relu activation, batch normalization and tanh activation for the action space. The network has an initial dimension the same as the state size.

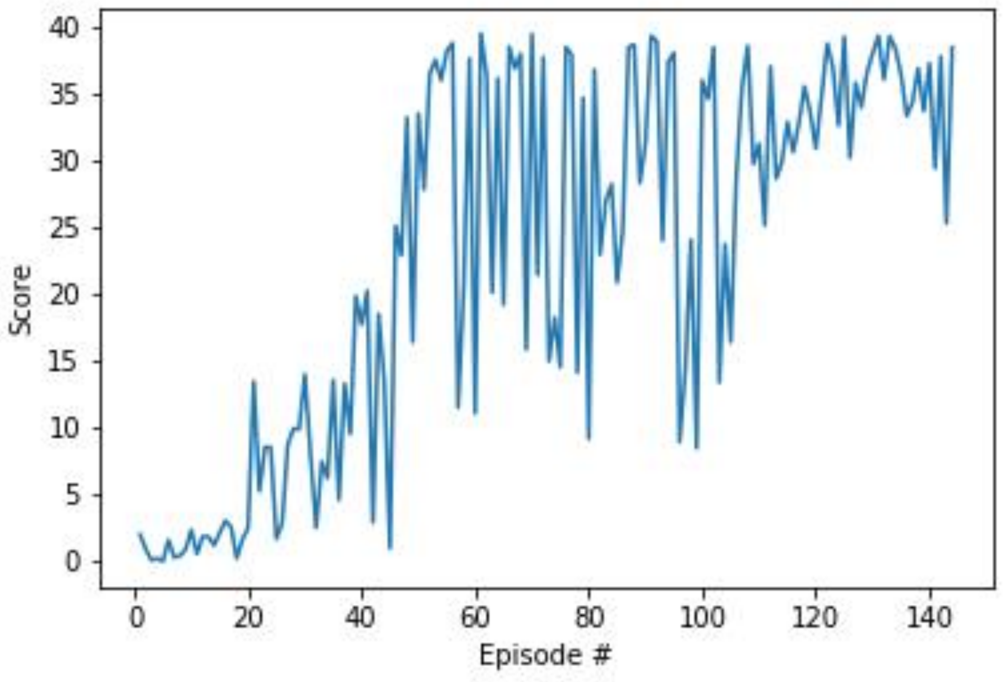
The Critic networks utilize two fully connected layers with 256 and 256 units with relu activation and batch normalization.

Hyperparameters

Name	Description	Value
buffer_size	replay buffer size	1e6
batch_size	mini batch size	128
discount	discount_factor	0.99
target_mix	factor to weight nw weights	1e-3
lr_actor	learning rate for actor	3e-4
lr_critic	learning rate for critic	3e-4
noise_mu	Mean for noise process	0
noise_theta	Theta for noise process	0.15
noise_sigma	Sigma for noise process	0.05
actor_fc1	Number of neurons in first fully connected layer for the actor	256
actor_fc2	Number of neurons in second fully connected layer for the actor	256
critic_fc1	Number of neurons in first fully connected layer for the critic	256
critic_fc2	Number of neurons in second fully connected layer for the critic	256
max_episodes	Maximum number of episodes in training	1000
max_steps	Maximum steps in each episode	1e6

Results

The figure below shows the performance of the algorithm to solve the environment. We achieved average score of +30 from iteration 44.



Future Work

Using multiple copies of the agents to learn the parameters was not explored in this project. This would be a next step. With respect to the learning algorithm itself – Proximal Policy Optimization (PPO) and Distributed Distributional Deterministic Policy Gradients (D4PG) algorithm could be explored.