Solving Continuous Control Environment by Deep Deterministic Policy Gradient

Xiongwen Qian

1. Problem

This project aims to solve a continuous control problem in a 3D Unity Environment in which 20 double-jointed arms are trained to trace 20 moving targets. For each arm, the goal is to maintain reaching its moving target as long as possible. A reward of +0.1 is provided for each step that the agent's hand is in the goal location.

The state space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of each arm. Each action is comprised of four numbers, corresponding to the torques applied to the two joints. Each number is within the range -1 to 1.

2. Learning Algorithm

Deep Deterministic Policy Gradient (DDPG) is used to solve the problem.

a) Algorithm Overview

DDPG can viewed as an extension to the Deep Q-Learning Network to handle actions in the continuous space. Overall, it consists of an actor and a critic. The actor maps the perceived states to actions by a neural network. The output of the network is the actions that should be taken. The critic is similar to Deep Q-Learning Network (DQN). It builds a mapping from the input state-action pair to its Q-value by another neural network.

The actor's neural network is trained by a gradient ascent method since the objective is to maximize the state-action Q-value. The critics function is trained by a gradient descent method since the objective is to minimize the difference between the target Q-value and the local Q-value.

b) Neural Network Structure

For the actor's neural network, the input is a state vector of 33 units. 2 Hidden layers are used. The first hidden layer has 400 units, and the second has 300 units. The output layer has 4 units representing the values of the 4 actions. Batch normalization is used after the input vector goes through the connection to the first hidden layer. It ensures that the inputs to a layer are within the same range which helps the training of the network greatly. The activation function is Relu except for the last layer whose activation function is tanh.

For the critic's neural network, one input is a state vector of 33 units. It is connected to the first hidden layer which has 400 units. Also, batch normalization is used. The

other input, the action vector, is concatenated with the first hidden layer before the concatenated layer is connected to the second hidden layer. The output layer has one unit, the Q value.

Both the actor and the critic have two copies of networks, one for calculating the target Q-value and one for the local Q-value update.

c) Learning between Interval
In one episode, the learning process is called periodically rather than after each time step. Usually after one time step, not many more new samples are generated, so it is better to call the learning process in a periodic way. When a learning process is carried out, it is repeated several times in the same spirit of Proximal Policy Optimization (PPO).

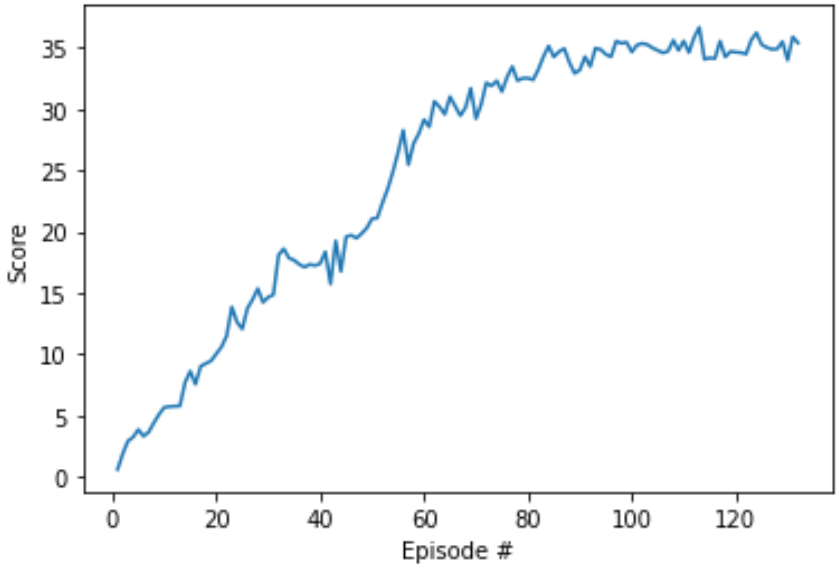
d) Random Process for Action
It turns out to be very helpful to reset the mean for the random process that is added to the actions after each learning process. Maybe because such operation ensures that the random process does not deviate from the mean too much, so the whole training becomes more stable, which has been verified by experiments.

e) Experiment Replay and Soft Updates
Experiment Relay and Soft Updates techniques are used in the learning algorithm. Experiment Relay helps to break the correlation between training experiences. Soft Updates copies the weights of the local network to the target network in a tiny but steady way, so the weights of the target network keeps continuously relatively stable.

f) Hyperparameters
Buffer size for the replay buffer = 1,000,000
Batch size for every training = 128
Discount factor γ = 0.99
Parameter for soft update τ = 0.0001
Learning interval = 20
Learning repeat times each learning process = 10

3. Result

The environment is solved in 150 episodes.



4. Ideas for Future Work

The whole training process takes a lot of time. It is desirable to further improve the training efficiency of the algorithm. In the current setting, the experiences saved in the replay buffer are treated equally. Each of the experience has the same chance to be selected. However, some of the experiences may be more important for the training because they contain more valuable information. One potential way is to apply Priority Experienced Replay method in which more important experience will be sampled more often. The main idea is to use the TD error in the process of updating the critic network as the criterion to assess the importance of experiences. More details can be found in the paper A novel DDPG method with prioritized experience reply.