Project 2: Continous Control

Ke Wang

Nov. 2018

1 Introduction

In this project, I developed a deep deterministic policy gradient algorithm to solve a continuous control problem in Unity environment.

1.1 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.

The version of environment we adopt in this project contains 20 identical agents, each with its own copy of the environment.

1.2 Solving criterion

The RL agents must get an average score of +30 (over 100 consecutive episodes, and over all agents).

After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 20 (potentially different) scores. We then take the average of these 20 scores. This yields an average score for each episode (where the average is over all 20 agents). The environment is considered solved, when the average (over 100 episodes) of those average scores is at least +30.

2 Algorithm

I adopted a Deep Deterministic Policy Gradient described in Algorithm 1.

Algorithm 1 Deep Deterministic Policy Gradient

```
1: Randomly initialize the weights for local critic network and local actor network
2: Randomly initialize the weights for local critic network and local actor network
3: Initialize replay buffer $R$
4: for $episodei = 1$ to $M$ do
5:     Reset Environment
6:     Initialize a random process N for action exploration
7:     Receive initial observation state $s_1$
8:     for $t = 1$ to $T$ do
9:         Select action at $= \mu + N_t$ according to the current policy and exploration noise
10:        Execute action at and observe reward $r_t$ and observe new state $s_{t+1}$
11:        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
12:        Sample a random minibatch of M transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
13:        Update the critic networks
14:        Update the actor policy
15:        Soft update the target networks
```

3 Neural network

3.1 Actor network

I proposed a four-layer fully connected linear neural network for actor network. The first layer is input layer. The second and third layers are hidden layers with a size 400. The last layer is the output layer with size 4. The last layer has a size 300. The first three layers adopt leaky relu as their activation function. The last layer adopt tanh as its activation function to bound the output. A batch normalization layer is added in between third and last layers.

3.2 Critic network

I proposed a four-layer fully connected linear neural network for critic network. The first layer is input layer. The second and third layers are hidden layers with a size 400. The last layer is the output layer with size 4. The last layer has a size 300. The first three layers adopt leaky relu as their activation function. The last layer does not have activation function.

4 Results

The of parameters of the algorithm are:

- replay buffer size: 100000
- minibatch size: 128

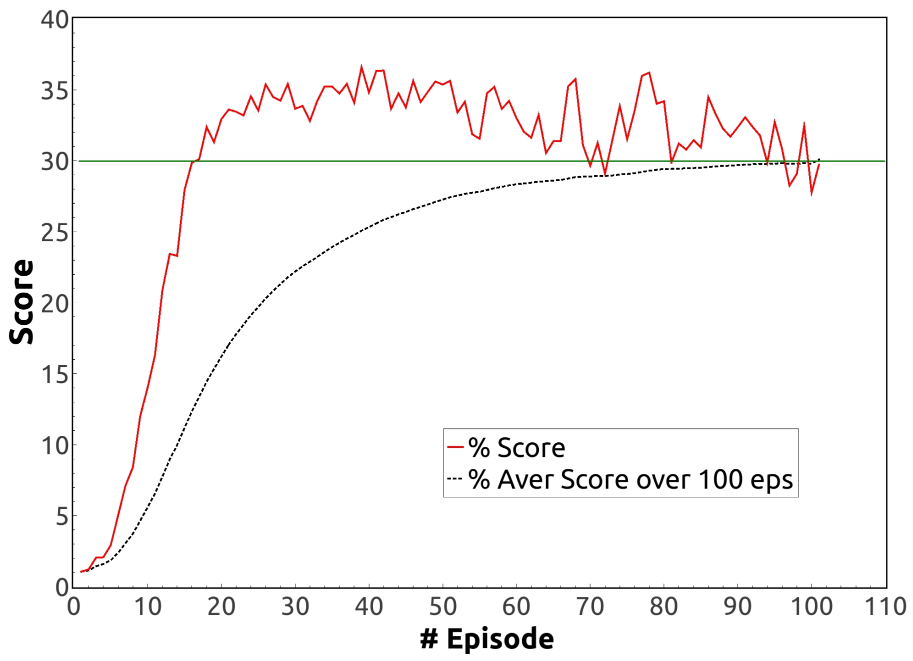

Figure 1: Rewards per episode

- discount factor: 0.99
- soft update of target parameters: 0.001
- learning rate of the actor: 1e-5
- learning rate of the critic: 1e-4

The learning curve (a plot of rewards per episode) is shown in Figure 1. The problem is solved after 101 episodes.

5 Conclusion

In this project, I developed a deep deterministic policy gradient algorithm to solve a continuous control problem in Unity environment. The problem is solved after 101 episodes.

Next step, I want to try prioritized experience replay and other multi-agent reinforcement learning algorithms.