Udacity

Deep Reinforcement Learning Nanodegree

Project 2—Continuous Control

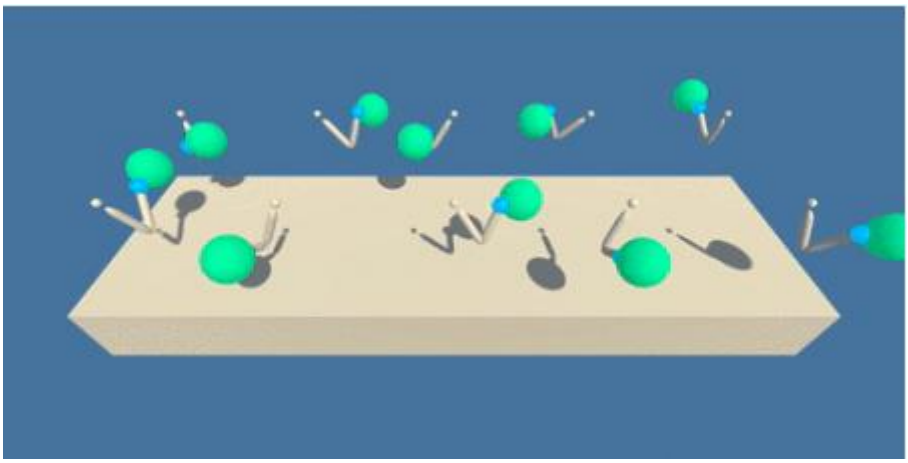
Liu Tianze

I. Introduction

In this project, I used DDPG to train the agent to achieve the goal of version 1.

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.



II. DDPG

For architecture, I referenced the DDPG exercise and did not change the basic network architecture too much.

In DDPG, there was an actor-critic architecture. The actor network was used to determine the best action for a specific state through policy gradient method and the critic network was used to evaluate the policy determined by actor by TD(Temporal Difference) error. In other words, when the actor used policy gradient to find the best action through gradient ascent, the critic network would tell the actor whether the direction of gradient ascent was right or not.

The formulas used for updating weights for actor and critic networks were as follows:

Actor：

Update the actor policy using the sampled policy gradient：

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a \mid \theta^Q) \mid_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu) \mid_{s_i}$$

Critic：

$$Set\ y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$$

Update critic by minimizing the loss：$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

I also add noise (OU Noise) to their actions at training time in order to make DDPG policies explore better.

After many trial and error, I found that 2 FC layers was enough. For neurons of FC layers, I have experimented many different combinations. For example, 512/512, 512/256, 256/256, 256/128 and so on. I found that 256/256 could have the best results. Hence, too many or too few neurons would not result in a better performance.

The hyperparameters of DDPG were as follow:

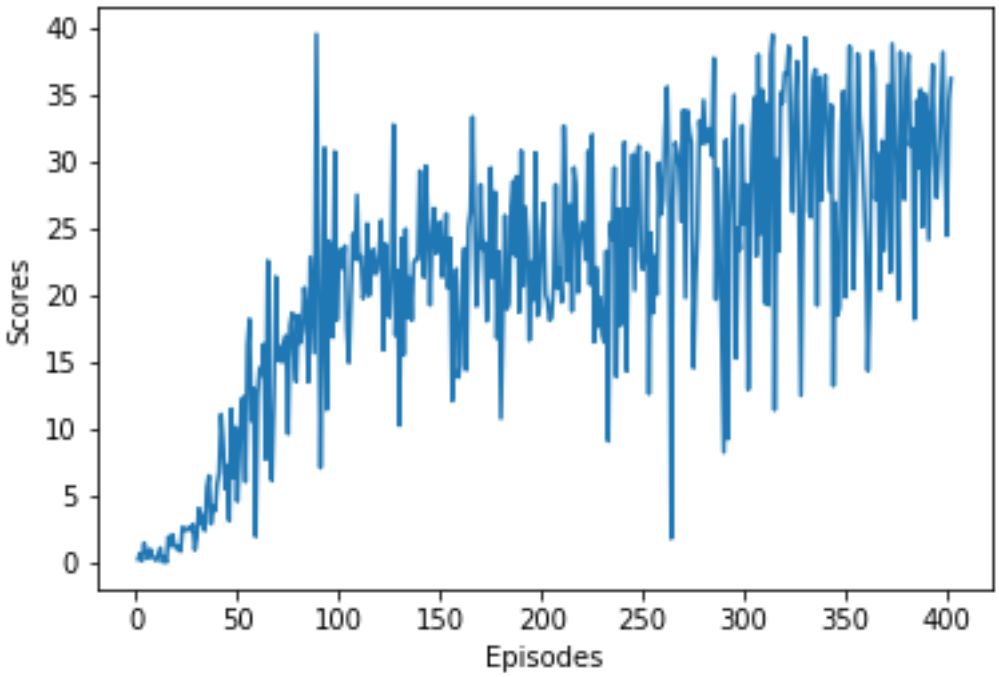
Replay buffer size	100000
Batch size	64
Gamma	0.99
TAU	0.001
Learning rate of actor	0.0002
Learning rate of critic	0.0001
Weight decay	0

The agent used replay buffer as length 100,000 and it could have the ability hold important experiences during the training. After experimenting, I found batch size 64 was suitable for training. The soft update of target parameters (TAU) was 0.001. Furthermore, I wanted the agent to focus on the long term returns so the discount factor gamma was set to 0.99.

III. Results

From the results of my project, it achieved 30.00 score in average below 500 episodes. At first, the average score was around 0. After about 100 episodes, the average score was closed to 10.00. After 300 episodes, the average score surpassed 20.00 in average. Finally, it achieved the average score of 30.01 at 402 episodes.

The training progress of DDPG model was as follow:



IV. Further work

Although I have achieved the goal, there was still some room to improve. For example, I could use D4PG as my network, maybe it would make me get better results because it used distributed probabilities to represent Q-Value. In addition, I could also use A3C to complete the second version of this project.