Project 2: Continuous Control

Yafeng Cheng

April 15, 2019

This is a report to describe the implementation of the Actor-Critic algorithm to solve the Continuous Control project.

1 The project introduction

The environment of the project is created using the Unity Machine Learning Agents v0.4. To create the environment on the local machine, we can follow the instruction on the following link .

The goal of the project is to train 20 double-jointed arms to move to target locations. A reward of +0.1 is provided for each step that one 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.

2 Algorithm

The algorithm used in this project is Deep Deterministic Policy Gradients (DDPG) agent. The algorithm is as following:

Algorithm 1: DDPG

Initialization:;
Create target actor $A_t(S)$ and local actor $A_l(S)$ with weights θ_t and θ_l , respectively. The input S represents the state at any time step. Both models output a vector with size same as the dimension of the action space;
Create target critic $C_t(S, A)$ and local critic $C_l(S, A)$ with weights η_t and η_l , respectively. The input S represents the state at any time step, while the input A represents the action at any time step. Both models output a one dimensional vector representing the value of the state action pair;
Let T be the maximum number of time steps each work can play in one episode;
Let E be the number of episodes each work can play;
while $t < T$ **do**
    Reset environment;
    while $t < T$ **do**
        Let each worker interact with the environment N time steps using target actor $A_t(S)$ and random noise generated from a Ornstein-Uhlenbeck process. Collect all the tuples $(S_{i,j}, A_{i,j}, R_{i,j}, S'_{i,j})$ for the i$-th$ worker and j$-th$ time step;
        Define $y^c_{i,j} = R_{i,j} + \gamma C_l(S'_{i,j}, A'_{i,j})$ as the observation and $\hat{y}^c_{i,j} = C_l(S_{i,j}, A_{i,j})$ as the fitted value. Define loss $L_c = \sqrt{\sum_{i,j}(y^c_{i,j} - \hat{y}^c_{i,j})^2}$;
        Define the loss for actor $L_a = -\mathrm{E}\left[y^c_{i,j}\right]$;
        Update the local critic $C_l(S, A)$ with the above defined loss L_c. Soft update the target critic $C_t(S, A)$;
        Update the local actor $A_l(S)$ with the above defined loss L_a. Soft update the target actor $A_t(S)$;
    end
end

There are two tweaks applied in the algorithm turn out to have a high impact on the learning.

- The noise generated from a Ornstein-Uhlenbeck process: the noise is correlated with the time steps, which is much more complex than a simple Gaussian noise.
- The batch normalization: given the large reply buffer, the batch normalization plays an very important role in accelerating the learning.

There are also other important configurations of the algorithms such as the relatively big batch size, suitable update frequency, learning rate, and size of the neural network.

2.1 The architecture of the neural networks

All the models used in the algorithm $A_t(S)$, $A_l(S)$, $C_t(S)$, $C_l(S)$ are feed forward neural network with two hidden layers. Each hidden layer has 512 and 256 units, respectively.

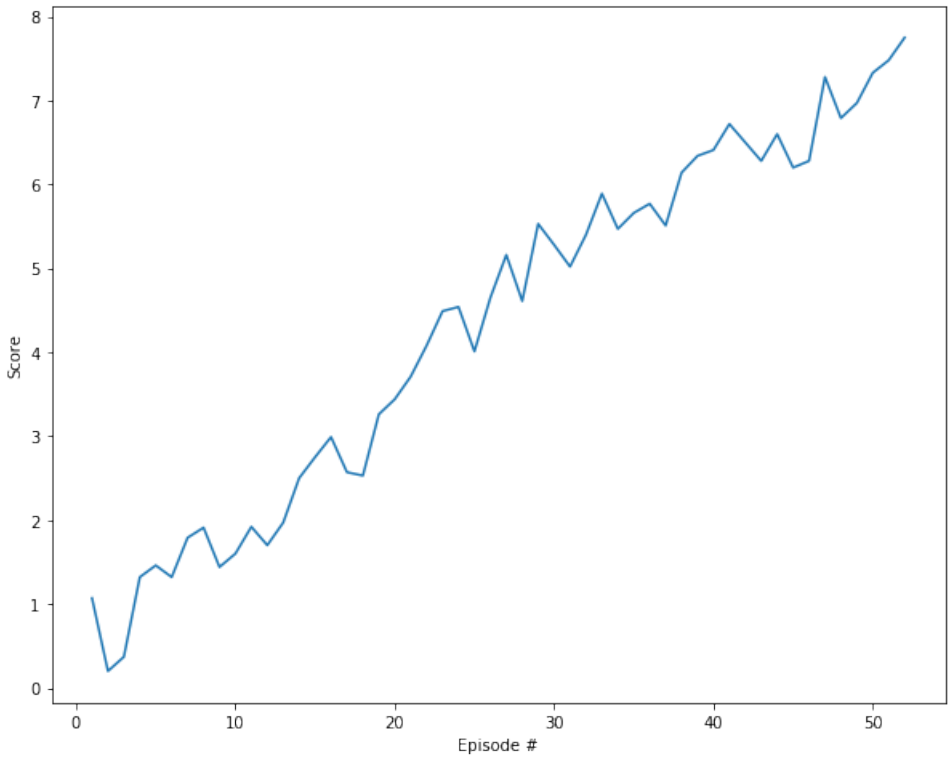
2.2 The choice of hyper-parameters

The hyperparameters used in the algorithm are listed in the following table:

argument	value	explanation
n_episodes	80	
tau	0.001	update rate?
actor_LR	0.0001	learning rate for actor
critic_LR	0.0003	learning rate for critic
gamma	0.99	decay rate for future return
max_t	2000	maximum number of steps in one episode
actor_hiden_layer_size	[512,256]	neural network hidden layer and cells in each layer for actor
critic_hiden_layer_size	[512,256]	neural network hidden layer and cells in each layer for critic
actor_hiden_layer_act	[nn.ReLU(),nn.ReLU()]	neural network activation function in each hidden layer for actor
critic_hiden_layer_act	[nn.ReLU(),nn.ReLU()]	neural network activation function in each hidden layer for critic
buffer_size	int(1e6)	replay buffer size
batch_size	512	batch size
update_every	4	update frequency

3 Outcome

The plot of the score from the first 52 episodes is shown as following:



4 Conclusion and future work

Based on my current experience, it is difficult to achieve success for this project. The number of different configurations I tested is more than 20 with the focus on the neural network architecture, and the noise generator. The current result is the best configuration so far. There are many possible changes one can make in the algorithm. One of the

bottle neck for testing is the computation time. As each setting takes a long time or a large GPU to test, it is difficult to find the best configuration for the problem.

I would like to try to test Advantage Actor Critic on this project in the future.