Project 3: Collaboration and Competition

Yafeng Cheng

April 15, 2019

This is a report to describe the implementation of the Actor-Critic algorithm to solve the Collaboration and Competition 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 .

In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.
- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

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

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 64 units.

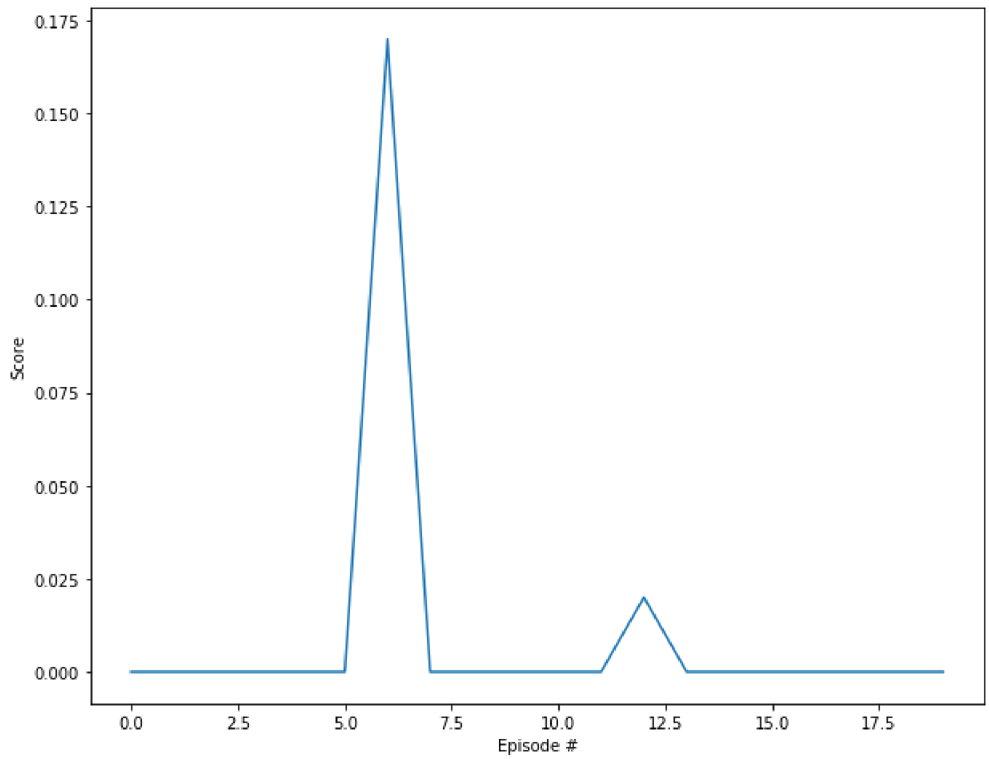
2.2 The choice of hyper-parameters

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	[64, 64]	neural network hidden layer and cells in each layer for actor
critic_hiden_layer_size	[64, 64]	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	5	update frequency

3 Outcome

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



4 Conclusion and future work