Tennis project report

Introduction

The third project in reinforcement learning nanodegree is about Tennis. In this Tennis 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.

Each agent has 24 input variables for states, including the position, velocity of the ball and racket. Each agent can receive its own, local observation. Each agent has 2 continuous variables in terms of action, which correspond to the movement towards the net, and jumping.

This task is episodic, and the target goal is to obtain an average score of 0.5 over 100 consecutive episodes after taking the maximum over both agents.

We can see this is a multi-agent environment because we have two agents in this environment. To solve this project, I decided to use the deep deterministic policy gradient (DDPG) algorithm. DDPG uses actor-critic approach with deterministic policy gradient. It is perfect for project with high-dimension and continuous action space (which is our project needs).

Model design

In order to use the actor-critic approach, I created the both actor and critic networks for each agent. For both actor and critic networks, there exist both local and target networks. Therefore, we have:

<u>Agent0:</u>

Actor-local, actor-target, critic-local, critic-target

<u>Agent1:</u>

Actor-local, actor-target, critic-local, critic-target

Though we have multiple agents in this environment, the updating and learning process are the same for both agents.

<u>Actor networks structure (for both local and target networks):</u>

Input states variables -> batch normalization -> linear layer-> batch normalization -> relu activation function -> linear layer -> batch normalization -> relu activation function -> linear layer -> tanh activation function -> output

<u>Critic network structure (for both local and target networks):</u>

Input states variables -> batch normalization -> linear layer -> relu activation function -> output concatenated with action as the input for next step -> linear layer -> relu activation function -> linear layer -> output

We can see the main differences between the actor and critic network include:

1. Outputs are different. The actor network's output is the action which is a continuous variable with range between -1 and 1. The output of critic network is a number value which represents the expectation value of rewards given pair of (state, action).
2. In the middle of networks, critic network concatenates the state and action as the input for next step.

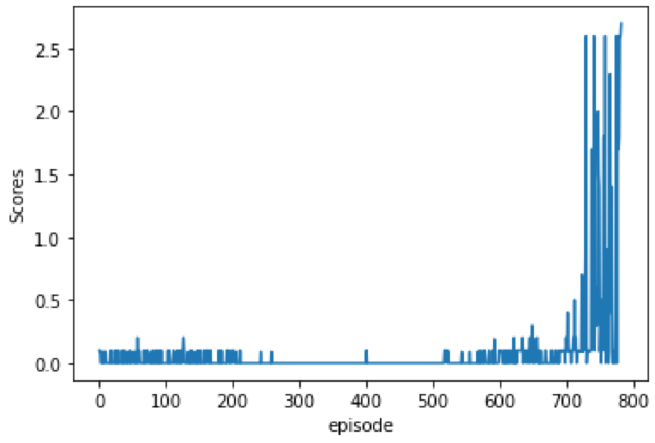
Batch normalization has been frequently applied in both actor and critic networks to increase the computing speed.

For the learning process, in both actor and critic networks, the local networks will keep play the game for each step while the target networks will update every few steps. In this model, target networks update every 20 steps. The replaybuffer is also used to help with the updating process: all steps were stored in the replaybuffer and a batch of steps were sampled from the replaybuffer and used to update the target networks. This update process is defined in the function soft_update and controlled by parameter TAU.

Results

For each episode, the scores of max rewards among the two agents were saved. Figure 1 shows how the score changes by episode. We can see, at the first few hundreds of episodes, the agents did not learn much from the training until the 600 episodes. The agents started to learn but the learning process is not as stable as we see in the previous two projects. The score values changed between 0 to 2.5. The whole training process was forced to stop whenever the average score of 100 episodes reached to 0.5.

Figure 1: Scores value change by episode.



Future direction

Except of the DDPG I applied here, there are many other approaches we can apply in multi-agent reinforcement learning environment, such as D4PG, A2C, A3C, and so on. Also, try larger networks and tune parameters such as learning rate, learning rate decay, update frequency, batch size, may also help. But tuning parameters could be very tricky due to the high sensitivity of the model. ReplayBuffer could also be improved, such as weighted sampling rather than random sampling.