Unity ML- Collaboration and Competition

DRL Nanodegree Project 3

In this environment, two agents try to play a racket game. 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, each agent tries to keep the ball in the game as much as possible.

The observation space of each agent consists of 24 element vector corresponding to the position and velocity of the ball and racket. In total, there are 48 elements for two agents. Each agent receives its own, local observation. Two continuous actions are available that are between -1 and 1 in total a four element vector represents the two agents' actions.

The game task is episodic, and in order to solve the environment, any of the agent your must get an average score of +0.5. Hence, the better playing agent must get a higher score which should also be reported. Specifically, each agent adds its discounted rewards and at the end of the episode, the higher rewarded agent's score must be selected and assumed as the overall score of the task.

We have decided to solve this task using the Deep Deterministic Policy Gradients [1] algorithm. The algorithm is actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. While the actor learns the trajectories, the critic learns estimated value functions similar to Q-learning. Although, the original version of the DDPG algorithm is designed for the single agent version. Modifying the algorithm that receives observations from both agents were able to solve the current task. The original single agent version of the DDPG is given below:

Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$
        Update the target networks:
            $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
            $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
    end for
end for
```

The algorithm consists of two neural networks. The actor network estimates the policy given the state of the agent. The critic network estimates the state-action value. They are separate neural networks and have different weights. Also, there exists target actor and critic networks which allow the agent to accumulate immediate experiences by slowly updating the target networks and reduce overall variance of the agent behavior.

The experiences that the agents acquire from the environment are stored in a buffer called replay buffer. By sampling from the replay buffer, the agents can sample same experiences multiple times and in addition can keep rare events in the buffer. This improves the training to a large extent.

All the agents take actions (a_t) and observe the rewards (r_t) and new states (s_{t+1}). Then, the transitions are stored in the replay buffer R.

Then the agents sample from the replay buffer and set the target state-action value function from the target networks:

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

Then the critic loss which is the mean squared loss is calculated as target value and the local network:

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

Next, the actor policy is updated as:

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

Finally, the target actor and the critic networks are updated as:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

The update is called the soft update because the target networks are updates at each step but with a fraction of the changes from the actor and critic networks $\tau < 1$.

In order to solve the environment for two agents, we have modified the replay buffer over the the single agent model:

Modified Replay Buffer: We have modified the replay buffer to have more than one agent in a single entry. In the modified version, the replay buffer consists of two observations from each agent. Thus, each entry of the replay buffer has state, action, reward, next_state signals from each agent. The done signal is shared for both agents.

Moreover each entry in the replay buffer is:

($state_1$, $state_2$, $action_1$, $action_2$, $reward_1$, $reward_2$, $next\_state_1$, $next\_state_2$, done)

Thus, each agent

Gradient clipping of the critic network: During training of the critic network we apply gradient clipping in order to limit the weights of the critic network.

The hyperparameters that we have used to solve the algorithm is given below:

BUFFER_SIZE = 2e6 # replay buffer size

BATCH_SIZE = 64   # minibatch size

GAMMA = 0.99   # discount factor

TAU = 1e-3   # for soft update of target parameters

LR_ACTOR = 1e-4   # learning rate of the actor

LR_CRITIC = 3e-4   # learning rate of the critic

WEIGHT_DECAY = 0.0001   # L2 weight decay

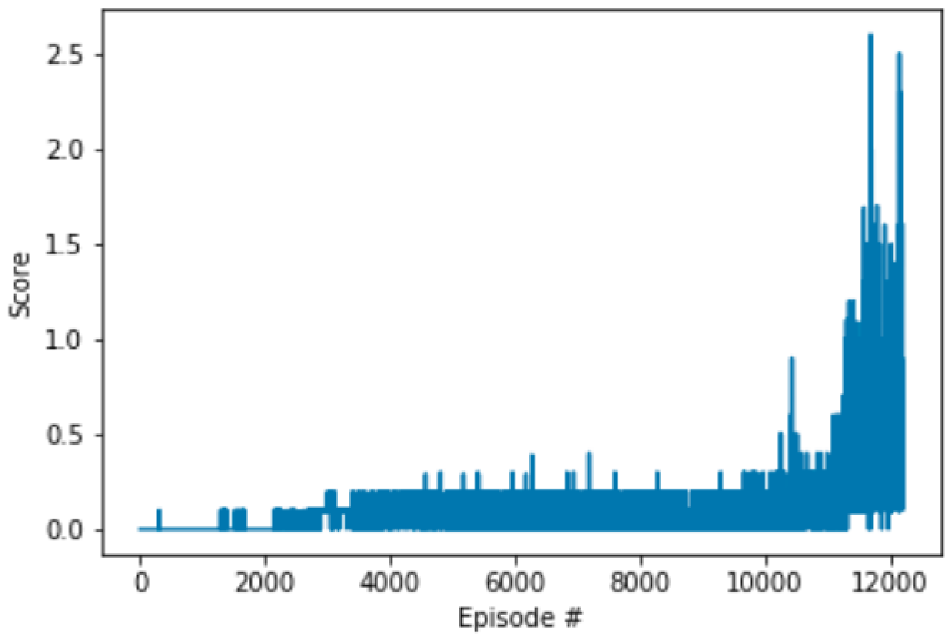
Both actor and critic neural network has RELU activation for all hidden layers. The final output layer of the actor was a tanh layer for the for available actions for two agent. Thus the actor network has 4 actions, 2 for each agent. The critic network has a single output.

Neural networks have 2 hidden layers with 400 and 300 units respectively (≈ 130,000 parameters). For the critic network, the action inputs are included at the second hidden layer. The minibatch size is 64 and the shared replay buffer size is two million entries. For the exploration, we used the same parameters of Onstein-Uhlenbeck process as as the DDPG [1] with $\theta = 0.15$ and $\sigma = 0.2$.

With the given hyperparameters, the defined goal is to get and average score of 0.5 for one of the agents on average over 100 episodes. We are able to achieve the goal in 12199 episodes. On an Intel 7700K 4.0GHz CPU with NVIDIA GTX1080 GPU, one episode takes approximately 4 minutes. Below we present the average score vs episode graph.

```
Number of episodes to run 350000
Episode 1       Average Score: 0.00
Episode 1001    Average Score: 0.00
Episode 2001    Average Score: 0.00
Episode 3001    Average Score: 0.10
Episode 4001    Average Score: 0.09
Episode 5001    Average Score: 0.10
Episode 6001    Average Score: 0.09
Episode 7001    Average Score: 0.09
Episode 8001    Average Score: 0.09
Episode 9001    Average Score: 0.07
Episode 10001   Average Score: 0.06
Episode 10101   Average Score: 0.06
Episode 10201   Average Score: 0.10
Episode 10301   Average Score: 0.14
Episode 10401   Average Score: 0.09
Episode 10501   Average Score: 0.12
Episode 10601   Average Score: 0.14
Episode 10701   Average Score: 0.14
Episode 10801   Average Score: 0.14
Episode 10901   Average Score: 0.14
Episode 11001   Average Score: 0.14
Episode 11101   Average Score: 0.16
Episode 11201   Average Score: 0.18
Episode 11301   Average Score: 0.19
Episode 11401   Average Score: 0.25
Episode 11501   Average Score: 0.27
Episode 11601   Average Score: 0.32
Episode 11701   Average Score: 0.39
Episode 11801   Average Score: 0.48
Episode 11901   Average Score: 0.37
Episode 12001   Average Score: 0.34
Episode 12101   Average Score: 0.38

Environment solved in 12199 episodes!   Average Score: 0.50
Runtime wall:  3027seconds
```

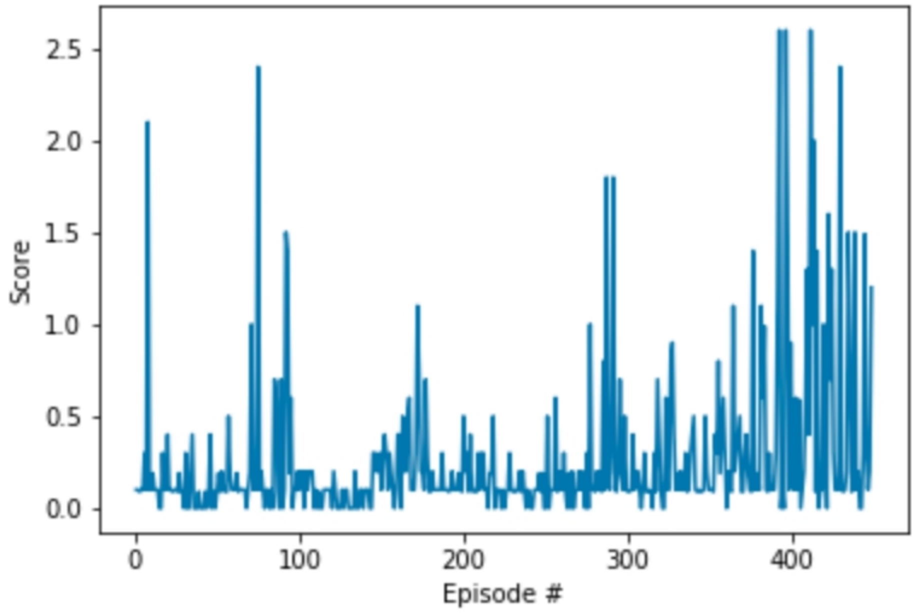


The agent starts to learn very slow however after around 9000 episodes, the learning speed increases. With the current hyperparameters, the agent can obtain average score of 0.50 in 12199 episodes.

After solving the environment, we have saved the parameters of the actor and critic networks. Then loaded them to run more episodes to observe how the agent with saved parameters can learn. The figure below shows that with transferred weights, the algorithm is able to complete the task in 448 episode.

```
Number of episodes to run 350000
loading the model...
Episode 1       Average Score: 0.10
Episode 101     Average Score: 0.21
Episode 201     Average Score: 0.16
Episode 301     Average Score: 0.19
Episode 401     Average Score: 0.33

Environment solved in 448 episodes!     Average Score: 0.50
Runtime wall:  268.0117664337158
```



As a future work, the multi-agent version of the current version could be implemented [2]. In the new version, each player agent can have policy networks whereas they can share the replay buffer and critic network. In this way, the players will collect different learning experiences, however they will contribute to the same replay buffer. By sharing the critic network, each agent will be able to obtain a better view of the value network throughout the environment. Implementation wise, each agent can act in a single python process that is obtained from the process pool and write to the replay buffer. In order to write to the replay buffer, a synchronization mechanism like a queue can be used. In this way, only one process can get the access to write to the buffer.

References:

1. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971. 2015 Sep 9.
2. Lowe R, Wu Y, Tamar A, Harb J, Abbeel OP, Mordatch I. Multi-agent actor-critic for mixed cooperative-competitive environments. InAdvances in Neural Information Processing Systems 2017 (pp. 6379-6390).