Project Report: Collaboration and Competition

Rui Ding

March 2020

1 Introduction

In this project, I applied two DDPG agents which is structured have the same network structure and their learning experiences are exactly opposite to each other in a zero sum game. The two agents are trained to cooperate and control the rackets to bounce a ball for as long as possible. The environment is built through the Unity Tennis environment.

2 Description of Environment

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.

3 Algorithm

In this project, I applied a modified DDPG algorithm for the two agents. Each agent is instantiated with an actor network (and a target actor network), a critic network (and a target critic network), and a replay buffer with capacity 100000.

The DDPG algorithm works as follows. The actor network takes in a state as input and outputs a deterministic action vector, in this case of dimension 2. Hence denote the deterministic policy by:

$$\mu : S -> A$$

where the policy $\mu(s|\theta^\mu)$ is parametrized by θ^μ. The Q values corresponding to such a policy is:

$$Q^\mu(s_t, a_t) = E_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))]$$

The Q network is parametrized by θ^Q and is learned by the critic using Q-learning with the following loss function:

$$L(\theta^Q) = E_{s_t, a_t, r_t}[(r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) - Q(s_t, a_t|\theta^Q))^2]$$

The actor is updated using the following policy gradient which is obtained by applying chain rule:

$$\nabla_{\theta^\mu} J = \nabla_{\theta^\mu} E_{s_t}[Q(s, a|\theta^Q)|s = s_t, a = \mu(s_t|\theta^\mu)] = E_{s_t}[\nabla_a Q(s_t, \mu(s_t)|\theta^Q) \nabla_{\theta^\mu} \mu(s_t|\theta^\mu)]$$

Target networks are used for both the actor(μ') and the critic(Q') for stable training. The target networks are updated following a soft update rule with hyperparameter τ such that in every learning step:

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

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

Another key component of the algorithm is the noise process, which is defined in the original paper by an Ornstein-Uhlenbeck process that progresses through the entire episode. The exploration policy is set to be $\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \epsilon N_t$.

The two agents on trained on their respective local observations of the episode and the rewards are opposite to each other. When any of the two agents experiences a terminal state, the current episode ends. The noise process is refreshed for every new episode with a decay in the scaling factor ϵ, which is initiated to be 1.0, if the current episode yields an improvement in the average score.

4 Training Results

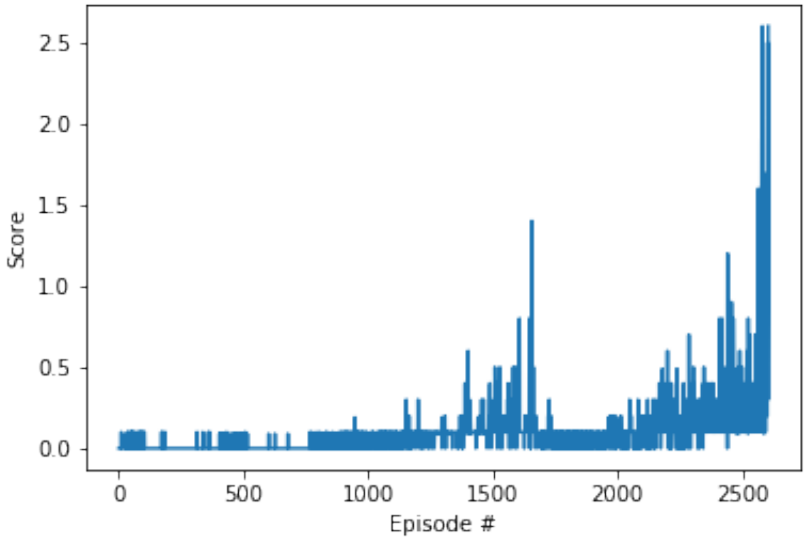
For this project I tried several sets of hyperparameters and ended up with drastically different results. It is clear that without a good choice of learning rates, batch size, exploration noise, and network structures, the agent might not learn at all or will learn up to some threshold and decay backwards.

I selected to implement both the actor and critic networks (local and target) with neural networks composed of two hidden layers, both having 128 cells. This is different from the original implementation but works well for this task,

which has input dimension 24. The actor networks has output dimension 2 corresponding to an action. The critic network takes in the action vector in the second layer and outputs a scalar value. ReLu activation is used for all activation functions.

For the DDPG agent, the batch size is selected to be 128, $\gamma = 0.99$, $\tau = 0.001$. The learning rate for actor and critic networks are both 0.0002.

I choose an epsilon decay factor of 0.001 for every episode that makes an improvement on the average score over the last 100 episodes. The agents made small improvements and reached an oscillation state around episode 1500, it is then able to improve further and solve the environment in 2505 episodes and reach an average score of 0.5 over the last 100 episodes. The score plot is below.



The weight of the local actor and critic networks in the final solution is saved as the files "checkpoint_actor_0.pth" and "checkpoint_critic_0.pth" for the first agent and "checkpoint_actor_1.pth" and "checkpoint_critic_1.pth" for the second agent.

5 Future Extensions

The two agents in this project are trained through collaborative self-play and does not involve the concept of competition. One possible extension is the train the agents with different objectives such that they try to beat each other instead of trying to cooperate and continue the game. It is interesting to compare the results between these two training results.

6 References

1. Course Materials and Codes

2. Multi-Agent Actor Critic for Mixed Cooperative-Competitive Environments, Lowe et al, 2017.