Udacity DRLND Project #3
Collaboration and Competition

Jing Zhao 11/13/2018

Introduction

In this report, I described in detail the implementation of a deep deterministic policy gradient (DDPG) algorithm for solving the collaboration and competition (Tennis) problem in the Unity ML-Agents environment. I first introduced the environment followed by a deep dive into the learning algorithm. The environment is solved in 1058 episodes. A plot of rewards per episode is showed to illustrate the training process, along with some discussions around why the DDPG algorithm was particularly suitable for this problem. A couple of ideas for future improvement are presented at the end.

Environment

The Tennis environment is provided by the DRLND's course website. 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. Figure 1 is a screenshot of the environment.

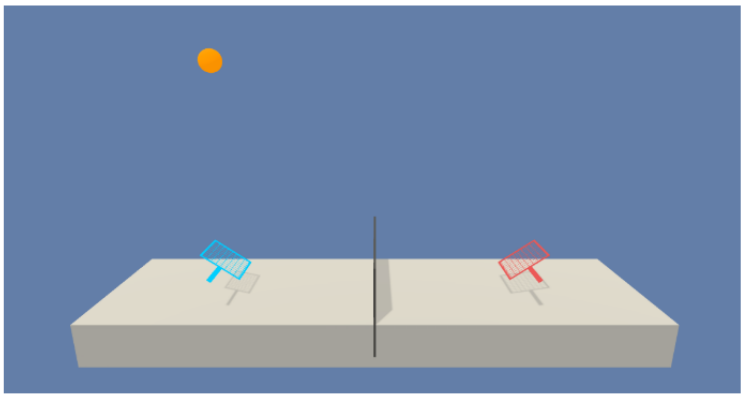

Figure 1. A screenshot of the Collaboration and Competition (Tennis) environment.

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, my 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.

Learning Algorithm

I used the DDPG algorithm [1] to solve this environment. DDPG is basically an actor-critic method. The actor network maps states to the continuous action space, while the critic network approximates the Q-value of state-action pair. The goal is to maximize the Q-value by adjusting the weights of actor and critic networks. Inspired by the success of deep Q-network (DQN) [2], updating critic network is also driven by minimizing the temporal-difference from one time-step to the next. The two significant improvements of DQN over traditional online Q-learning - namely experience replay and fixed target network – are adopted in the DDPG algorithm. A pseudocode from [1] is shown below for reference.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' 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

Note that, we can effectively turn this two-agent play problem into a self-play scenario if the agent only receives its own local observation at every time step. This is desirable because the nature of this problem is collaboration only – no competition. The optimal equilibrium is reached when the competency of two agents are at the same level. Suppose one agent was significantly better than the other, the game couldn't last long at all because the weaker player would always miss the ball. As a result, the total score would be quite low.

Therefore, I decided to train two agents using the same actor and critic networks. Specifically, I built two 2-layer feedforward neural networks (NN) for the actor and critic, respectively. Table 1 shows the hyperparameters of these networks along with some other learning parameters used in the DDPG algorithm.

Table 1. Hyperparameters of DDPG and other learning parameters

Parameter	Description	Value
State_size	Dimension of the observation space	24
Action_size	Dimension of the action space	2
BUFFER_SIZE	Replay buffer size	1,000,000
BATCH_SIZE	Minibatch size	1024
GAMMA	Discount factor	0.99
TAU	For soft update of target parameters	1e-3
LR_ACTOR	Learning rate of actor network	5e-4
LR_CRITIC	Learning rate of critic network	1e-3
UPDATE_EVERY	How often (timesteps) to update the target network	2
UPDATE_RUNS	If update, how many times in a row	1
N_episodes	Total episodes in training	2000
Max_t	Maximum time steps in each episode	1000
[state_size, fc1_units, fc2_units, action_size]	Units in each layer of the actor network	[24, 128, 64, 2]
[[state_size, action_size], fcs1_units, fc2_units, output_size]	Units in each layer of the critic network	[[24, 2], 64, 64, 1]
Eps_start	Start value of noise (exploration) decay	1.0
Eps_end	End value of noise (exploration) decay	0.1
Eps_decay	Rate of exponential noise (exploration) decay	0.995

It is worth noting that learning hyper-parameters listed in Table 1 are almost identical to those used in my second project except for UPDATE_EVERY and UPDATE_RUNS. More frequent updates are necessary simply because each episode can be very short especially at the beginning of training. Obviously, state_size and action_size must also be adjusted accordingly.

Result and Discussion

I leveraged the DDPG Python implementation from the course GitHub repository (ddpg-pendulum) for this project. Figure 2 below shows the score of each episode (in blue) as well as the moving-average across past 100 episodes (in red dash line). Note that these scores are taking maximum over two agents. According to the project description, this environment is “solved” once the agent is able to receive an average reward (over 100 episodes) of at least +0.5. I achieved this goal in 1058 episodes.

```
Episode 1158    Average Score: 0.51     Score: 2.39
Environment solved in 1058 episodes!    Average Score: 0.51
```

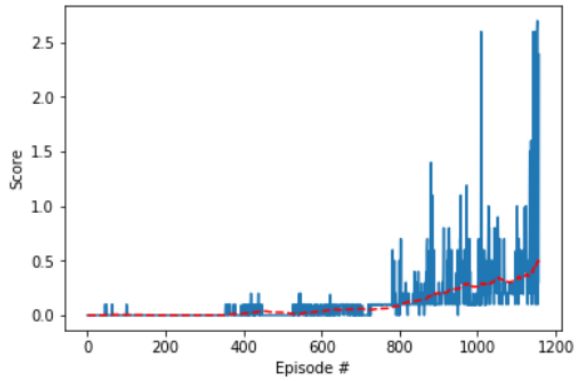

Figure 2. Maximum score over two agents (blue) and its 100-point moving average (red) during training using a DDPG algorithm. The environment is solved in 1058 episodes.

I must admit that it surprised me at the beginning when I solved the environment just by "naively" applying DDPG algorithm to this problem. However, I came to realize later that this was somewhat expected because the problem I was asked to solve. First, the two tennis players are physically separated. They are only allowed to move in spaces that are mutually exclusive, namely, two sides of the court. Second, the odds of hitting a ball is largely determined by its trajectory - not much by the opponent's movement especially after it hits the ball. Above two unique attributes of the environment result in a "weak interdependency" between two players to a degree that one can almost regard them as two independent entities. This is exactly what I did in my implementation that I chose to only include one player's actions in the critic network. However, this working implementation may not work well if a "strong interdependency" exists between two agents. For instance, each agent is allowed to freely cross the net and play at its opponent's home court. Or, the two agents can be physically close to each other so that one doesn't have much time to respond to the other's action. These conditions would for sure make the problem harder. However, if we take the "coach's view", the two players are still teammates not competitors. They share the common goal which is to keep the ball up in the air as much time as possible. Therefore, we can design a "team agent" which is made up by the two players. This team agent's state space and action space would be the combination of each player's own. The beauty of this view is that we may still be able to use DDPG and train a single set of actor & critic network to achieve satisfactory performance.

In addition, I'd like to comment on the benefit of sharing the same actor network between two agents. Inspired by the self-play concept, we can think of one player as the "clone" of the other player. The only difference between the two is that they play at different side of the tennis court. We would expect a human player to perform consistently regardless which side she stands on the court. By the same token, we should expect the actor network to generalize its learning to achieve comparable performance on both sides. For this reason, only one actor network should suffice.

Ideas for Future Work

The current DDPG based working implementation may not perform well in a mixed cooperative-competitive environment such as in a zero-sum game where multiple agents have competing goals. This

is because each agent's policy is changing as training progresses, and the environment becomes non-stationary from the perspective of any individual agent. I'm interested in implementing the MADDPG algorithm from [3] and see how it performs against my current DDPG version. Additionally, I plan to solve the more difficult Soccer environment.

Reference

[1] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. *Continuous control with deep reinforcement learning*. ICLR 2016

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. *Human-level control through deep reinforcement learning*. Nature, 518 (7540):529–533, 2015.

[3] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. *Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments*. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.