Project 3: Collaboration and Competition

May 12, 2021

Contents

1 Project Overview

In this project, we implement the Multiagent Deep Deterministic Policy Gradient reinforcement learning technique of Lowe et. al. (2017) to train agents to play tennis against eachother. The environment comes from OpenAI's Tennis environment. The observation space consists of 3 stacked frames of 8 variables corresponding to the position and velocity of the ball and racket. The action space consists of two continuous variables - one representing the agent's movement towards or away from the net and the other is for jumping, and each of the 2 agents receives its own local observation.

The reweards are structured episodically where hitting the ball over the net triggers a reward of +0.1 and if the ball hits the ground or the agent send the ball out of bounds on its oppenet's side a reward of -0.01 is received. The environment is considered solved when the average winning score of a match between the two agents equals +0.5 over 100 consecutive episodes.

2 Background

One of the limitations of Q-learning in competitive environments is that the rewards of the game evolve in a way that is not explainable by the agent's own policy. To see why, imagine two agents are playing tennis and the first agent always hits a forehand. This would cause the second agent to learn that hitting the ball to the first agent's weak side is optimal. However, if the second agent were to change his strategy midway through the game, all of the experiences in the first agent's replay buffer used to train his network would be based on his opponent's old policy, so it would either not converge to the optimal strategy, or do so very slowly.

While simple multi-agent models where the agents learn independently do not work well in practice due to the environmental nonstationarity caused by the changing policies of the other agents, utilizing an actor critic framework where the agents share a critic during the "reflection phase" of learning but act independently fixes the stale replay buffer problem, as now each agent has access to a centralized replay buffer.

3 Learning Algorithm

3.1 Network Architecture

3.1.1 Actor

- **Input Layer:** 24 dimensional layer
- **Hidden Layer 1**: 400 dimensional with relu activation
- **Hidden Layer 2:** 250 dimensional with relu activation
- **Output Layer:** 2 dimensional, corresponding to the 2 actions with tanh activation

3.1.2 Critic

- **Input Layer:** 24 dimensional layer
- **Hidden Layer 1:** 400 dimensional with relu activation
- **Hidden Layer 2:** 250 dimensional with relu activation
- **Output Layer:** 1 dimensional, corresponding to the 1 core with no activation, as the output is predicting the score

3.2 Parameters

- **BUFFER_SIZE** R= int(1e6); Number of episodes to retain in replay buffer.
- **BATCH_SIZE** N = 256; minibatch size
- **GAMMA** γ= 0.99; inter-temporal discount factor for future rewards
- **TAU** τ= 1e-3 ; for soft update of target parameters

- **LR_ACTOR** α_μ= 1e-3 ; Rate at which the actor learns during the gradient descent step.
- **LR_CRITIC** α_Q= 1e-3 ; Rate at which the critic learns during the gradient descent step.
- **WEIGHT_DECAY** = 0; L2 weight decay
- **LEARN_EVERY** = 10; Every 10 episodes, update network paramaters
- **M**=2,000: Maximum number of episodes
- **T**=1,000: Maximum length of any single episode

3.3 Algorithm

Following the structure of Lowe et. al. (2017) we have:

Algorithm 1 MADDPG algorithm

Randomly initialize critic network $Q\left(s, \boldsymbol{a}|\boldsymbol{\theta^Q}\right)$ and actor $\boldsymbol{\mu}\left(\boldsymbol{s}|\boldsymbol{\theta^\mu}\right)$ with weights $\boldsymbol{\theta^Q}$ and $\boldsymbol{\theta^\mu}$ for each agent
Initialize target network $\boldsymbol{Q'}$ and μ' with weights $\boldsymbol{\theta^{Q'}} \leftarrow \boldsymbol{\theta^Q}$, $\boldsymbol{\theta^{\mu'}} \leftarrow \boldsymbol{\theta^\mu}$
Initialize replay buffer R of size D
for episode = 1, M **do**

- Initialize a random process $\mathcal{N}$ for action exploration
- Receive initial observation state $\boldsymbol{s_1} = (s_{1,1}, s_{2,1}, ..., s_{n,1})$
- **for** t=1, T **do**
  - For each agent a_i,select action $a_{i,t} = \mu_i\left(s_t|\theta^\mu\right) + \mathcal{N}_t$ according to the current policy and exploration noise
    - The noise in this case is sample from an Ornstein-Uhlenbeck process (Uhlenbeck & Ornstein, 1930) in order to induce further exploration.
  - Execute action $\boldsymbol{a_t} = (a_{1,t}, a_{2,t}, ..., a_{n,t})$ and observe reward $\boldsymbol{r_t} = (r_1, r_2, ..., r_n)$ and observe new state $\boldsymbol{s_{t+1}} = (s_{1,t}, s_{2,t}, ..., s_{n,t})$
  - Store transition $(\boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{r_t}, \boldsymbol{s_{t+1}})$ in the shared replay buffer R
  - Sample a random minibatch of N transitions $(\boldsymbol{s_j}, \boldsymbol{a_j}, \boldsymbol{r_j}, \boldsymbol{s_{j+1}})$ from R
  - Set $\boldsymbol{y_j} = \boldsymbol{r_j} + \gamma Q'\left(\boldsymbol{s_{i+1}}|\boldsymbol{\mu'}\left(\boldsymbol{s_{i+1}}|\boldsymbol{\theta^{\mu'}}\right)|\boldsymbol{\theta^{Q'}}\right)$
    - This is the famous Bellman error term. There are 3 steps implicit in this representation:
      1. Use the actor network to choose action a_i in state s_i and receive reward r_i.
      2. Use the critic network to evaluate the value of taking action a_i in state s_i.
      3. These $y_i's$ can be thought of as expected values of the critic's action-value function
  - Update critic by gradient descent with learning rate α_Q by minimizing the loss:

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

    - In this step we update the weights in our critic network to best match the $(y_i's)$ in the last step.
  - Update actor policy with learning rate $\alpha_\mu$using the sampled policy gradient

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

    - In this step, essentially we observe all of the episodes N in our minibath sampled from the replay buffer R, and simply update the weights in the network to better align the state,action pairs with the ultimate reward of the game.
    - This step is similar to a supervised learning problem where we have "images" (s_i, a_i), predicted rewards or "labels" $Q\left(s_i, a_i|\theta^Q\right)$ and actual labels for the reward r_i received at the end of episide i.
  - Perform a soft update of the target networks:

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\,\theta^{Q'}\\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\,\theta^{\mu'}\end{aligned}$$

- **end for**

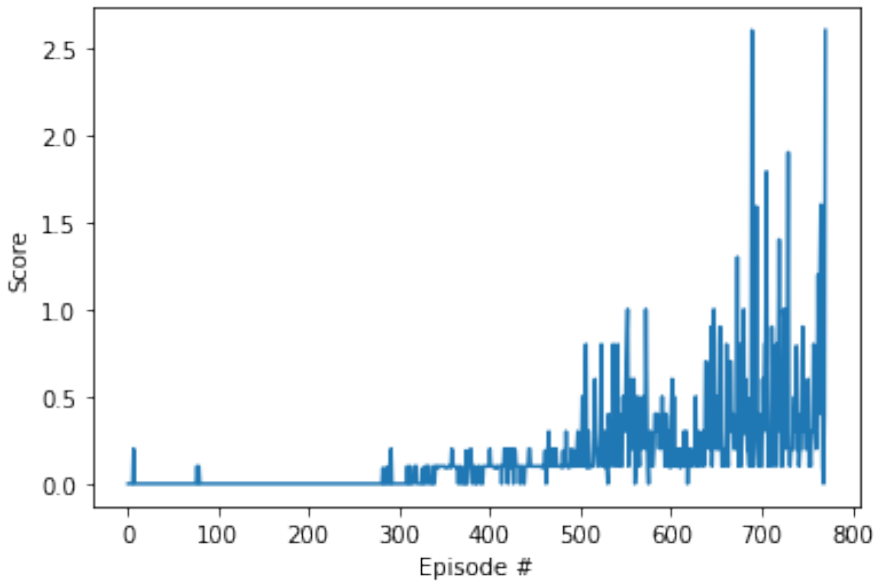
end for

4 Learning Curve

The aforementioned algorithm was able to solve the environment in 770 episodes.

Episode	Average* Score
100	0.00
200	0.00
300	0.00
400	0.06
500	0.12
600	0.31
700	0.35
770	0.52

* Average is over trailing 100 episodes



5 Future Work and Discussion

The first major area for improvement would be to consider a different learning algorithm. The current state of the art in the multi-agent learning space is Chao et al (2021) implementation of the multi agent proximal policy optimization, which tends to train faster than actor critic methods, achieves state-of-the-art performance on several benchmarks, and has an advantage in network parsimony as it only requires a policy network to be trained.

Within the MADDPG algorithm, there is always room for tuning the hyperparamters. In this case they were chosen manually on the basis of trial-and-error, however, a more systematized grid search may yield faster convergence.

Another area of exploration would be whether a prioritized experience replay in the replay buffer would aid in training.

Next, we can consider restructuring the reward function. The 10-to-1 ratio of reward to penalty seems a bit arbitrary; given how tennis actually works, where a rally can last indefinitely - leading to an indefinite amount of reward - but hitting the ball out of bounds can only happen once in an episode, I would consider reversing the reward structure.

6 Bibliography

Yu, Chao, et al. "The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games." arXiv preprint arXiv:2103.01955 (2021).

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., and Mordatch, I. (2017). *Multi-agent actor-critic for mixed cooperative-competitive environments.* arXiv preprint arXiv:1706.02275.

Uhlenbeck, George E and Ornstein, Leonard S. *On the theory of the brownian motion.* Physical review, 36(5):823, 1930.