Project 3: Collaboration and Competition

Timothy Bernard

August 2022

1 Problem Description and Algorithm Overview

The purpose of this project was to train two separate agents to volley a tennis ball back and forth (keeping it in the air). The observation space is 24 continuous variables (for each agent) consisting of position and velocity for the racket and ball (multiplied by 3 for each agent since there are three observed "frames" per observation). This problem is one of continuous control for multiple agents, so I decided to implement the solution using ideas from the Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments paper [1]. MADDPG essentially builds off of DDPG [2] for multiple agents working cooperatively/ competitively. As such it still uses ideas such as a replay buffer, actor-critic setup, Q-value neural network to evaluate policy etc. One of the key ideas with MADDPG is to use centralized training: all of the agents share experience during training. Practically, this results in a shared experience-replay buffer during training which has the joint-state, joint-next-state and rewards and actions for all agents. Another key point is "decentralized execution" where each agent uses only local observations at execution time (this is an explicit goal stated in the methods section 4.1 of the paper [1]). With these two goals in mind it acts as an extension to DDPG where the critic is augmented with information about the policies of others (it uses other agents' subsequent actions for the target and their current actions for the current inference together to calculate the critic loss, for example). This is in addition to using the joint states, which incorporate all agents' observations - all the while still only give the actor local information during execution.

2 Implementation

This algorithm was implemented using Python3 and PyTorch, to train a policy according to MADDPG to achieve the goal of a +0.5 score across 100 consecutive episodes. The implementation builds off of the ddpg-pendulum and ddpg-bipedal algorithms from the Udacity Deep Reinforcement Learning GitHub repo [3], and adapts them to use MADDPG. All of the DDPG-specific hyperparameters were able to be directly used from my previous project Continuous Control:

Table 1: **Hyper parameters**

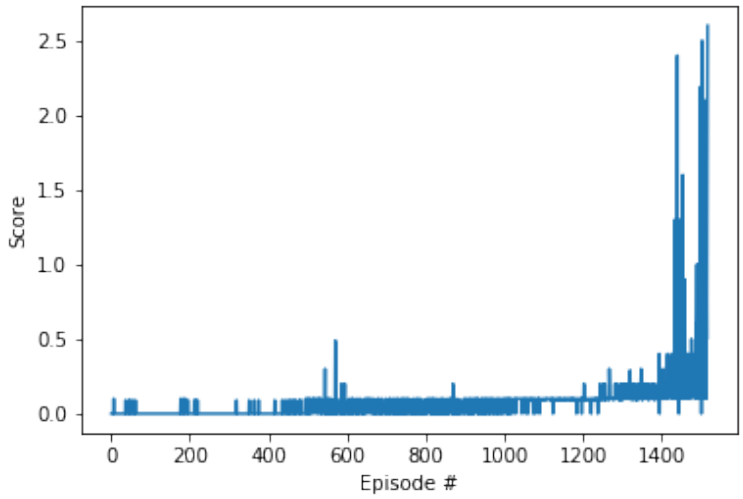
Replay Memory Buffer	100,000
Mini-batch Size	128
Discount Factor	0.99
Tau (for soft update of parameters)	1e-3
Learning Rate Actor	2e-4
Learning Rate Critic	2e-4
Max Number of Episodes	5000
Max Number Time-steps per Episode	Unlimited
L2 Weight Decay	0.0

I kept them the same since the algorithm really just combines multiple DDPG Agents and this intuition fortunately succeeded. Of note, the episodes were allowed to go on indefinitely (so no max time-steps per episode), this extra time for the agents to "play" helped the algorithm converge faster for me. I also set the maximum number of time-steps to be quite high since the agent did not see significant improvements in training until over 1000 episodes in. This can be seen more clearly in the results section. Special thanks to Udacity for the standard DDPG examples, Andrei from the knowledge center for the help in using torch.chunk for speed and for the psuedo code and hints provided from the mentors!

The architectures for the actor and critic networks (and therefore their target networks) were fully-connected multi-layer perceptrons. The actor network (to estimate the policy) had one hidden layer with 256 nodes. The critic had 3 layers (256, 256, and 128 nodes respectively). Relu activations were used for introducing nonlinearity and tanh was used at the last layer, forcing the output into the interval $(-1, 1)$ for the actor network. The weights for both were initialized from uniform distributions according to [2]. Beyond this, the critic network simply had to be adapted to take in twice the number of action-variables and twice the number of state-variables.

3 Results

Figure 1: Training Results



The algorithm was able to make the agent reach an average score of +0.5 over the past 100 episodes in 1516 episodes.

4 Future Ideas

One future idea would be to actually properly implement the target evaluation for the critic update, since my current implementation actually uses all rewards, but the algorithm as seen in the appendix of [1] (and intuitively) uses just the current agent in question's reward! Another possibility would be to use the suggestion from the future work section of [1] and use an ensemble of policies for each agent to improve performance. Lastly, it would be good to use prioritized replay [4] in order to more effectively select more "important" tuples (S, A, R, S') for training based on how "un-expected" an experience transition is.

References

[1] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2017. [Online]. Available: https://arxiv.org/abs/1706.02275

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[3] A. C. et. al, "Deep deterministic policy gradient, pendulum and bipedal," 2018. [Online]. Available: https://github.com/udacity/deep-reinforcement-learning

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. [Online]. Available: https://arxiv.org/abs/1511.05952