MADDPG solving multi-agent unity environments

Abdelkarim Eljandoubi

September 29, 2022

1 Introduction

In this project, I trained a MADDPG multi-agent to solve two types of environments.

1.1 Tennis 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 the 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 24 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 the movement toward (or away from) the net, and jumping.

1.2 Soccer environment

In this discrete control environment, four agents compete in a 2 strikers vs 2 goalies in soccer game. The goal for a Striker is to get the ball into the opponent's goal and for Goalie is to keep the ball out of the goal. A striker/goalie receive a reward of ± 1 when ball enters goal and $\mp 10^{-3}$ for existential.

1.3 Solving

An environment is considered solved, when an average score of $+0.5$ over 100 consecutive episodes, for each agent is obtained.

2 Implementations

I have started from the code provided in Udacity as a solution for the coding exercise which implements the basic form of DDPG [1] for the agent.

2.1 Trainer

I generalised the code to implement MADDPG [2] the multi-agent version of [1]. Moreover, I adapted it to interact with continuous and discrete action space of various sizes.

2.2 Architectures

I used here the same architectures from the coding exercise with small changes. In fact, both actor and critic are fully connected networks with two hidden layers. For an agent i:

actor
- the first hidden layer has 400 units and the second 300.
- all activation functions are ReLU except the output layer which is tanh if the space is continuous else $\operatorname{argmax} \circ \operatorname{softmax}$.

critic
- the first hidden layer has 500 units and the second 300.
- all activation functions are ReLU except the output layer. $\operatorname{argmax} \circ \operatorname{softmax}$.

3 Results

Throughout experiences, I set

- replay buffer size to 10^6
- batch size to 128
- the target update frequency to 1
- number of optimization step to 1
- the actor learning rate to 10^{-4}
- the critic learning rate to 10^{-3}
- $\gamma = 0.99$
- $\tau = 10^{-3}$

Also, I trained the networks by the Adam optimizer because of its superior experimentally performance.

3.1 Tennis

The agents succeeded to solve this environment with 2063 episodes (see fig. 1).

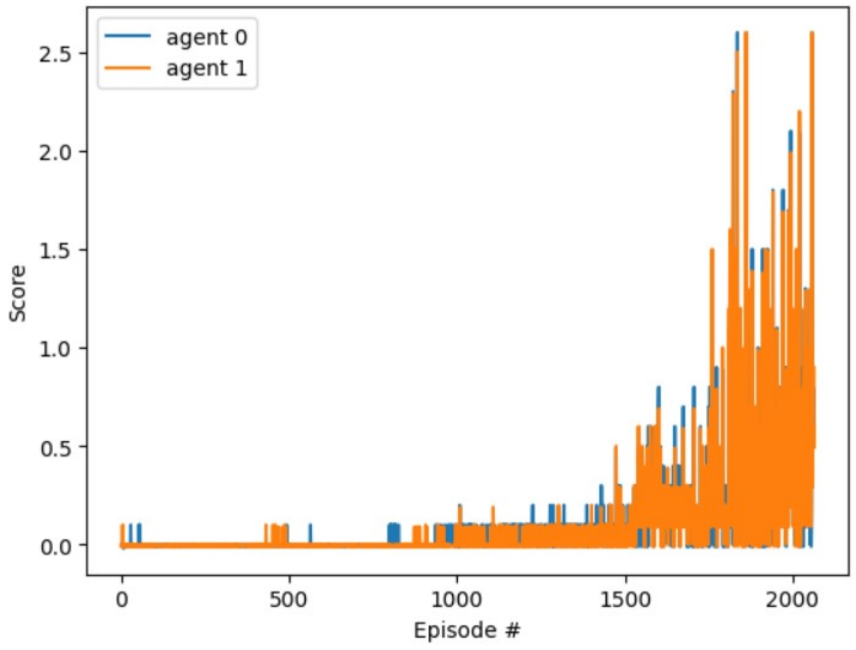

Figure 1: The score plot for Tennis.

3.2 Soccer

Need more investigation and will be updated soon.

4 Future work

In addition to the current work, we can do the following to improve performance by :

- adding prioritized experience replay from [4].
- implementing QMIX from [3]

References

[1] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016.

[2] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[3] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. *CoRR*, abs/1803.11485, 2018.

[4] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.