The Tennis Unity Environment with Multi Agent Deep Deterministic Policy Gradient (MADDPG)

Paolo Recchia

1 Introduction to Multi Agents environment

If within an environment we have multiple agents it's possible to follow two different approaches

Independents agents each agent observe its own environment and it learns its own policy regardless the existence of other agents. All the agents learn simultaneously (in parallel) and independently. Since each agent has a different description of the environment, this approach could lead to a non stationary description of the environment.

Interacting agents in this case a single policy is learned for all agents, in addition each agent observe the same environment and return a single reward

$$\pi : \mathcal{S} \to \mathcal{A}_1 \times \cdots \times \mathcal{A}_n \tag{1}$$
$$R : \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_n \to \mathbb{R}; \tag{2}$$

However, in this approach, the joint action space will increase exponentially as the number of agents increases.

If in an environment there are multiple agents, two different scenario can be observed

Competition when two or more agents are trained to maximize their own reward

Cooperation when two or more agents are trained to maximize a common reward,

in some context we could have a mixture of these two opposite scenario.

MADDPG [1] tries to fix the aforementioned issues by adapting the DDPG algorithm [2] for multiple agents. In the paper the team of OpenAI successfully trained both competitive and cooperative environment and the mixed ones too.

2 The MADDPG algorithm

The main idea of MADDPG is to split learning in two steps: the centralized training with the decentralized execution. In this framework we have many actors π_i (as the number of agents) which, by interacting with their own observation space O_i, they give a deterministic action a_i (as in the original DDPG paper [2]). Always as in DDPG we have many critics Q_i (always as the number of agents) which give the action–value function. However, differently from DDPG, at training time each critic take as input the information given by the policies of all agents.

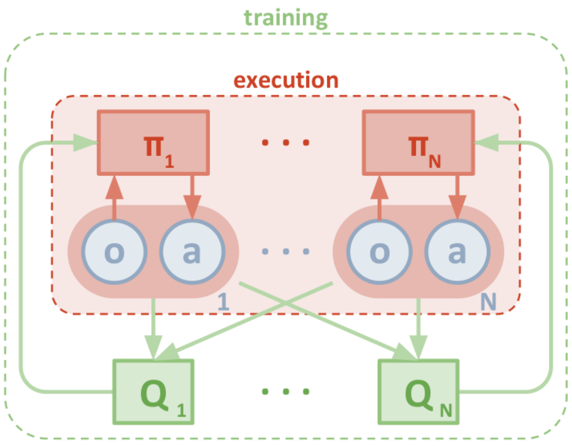

Figure 1: The general learning schema of MADDPG [1]

Exactly as in DDPG, in MADDPG are used the same key optimizations, namely the us of a softly updated target networks, a replay buffer and exploration with noise.

Let's indicate

- Q_i and Q'_i the ith critic network and critic target network respectively
- θ_{Q_i} and θ'_{Q_i} the parameters of the critic network and target networks
- μ_i and μ'_i the deterministic policy actor network and actor target network
- θ_{μ_i} and θ'_{μ_i} the parameters of the critic network and target networks
- $a = (a_1, \ldots, a_N)$ the actions of the N agents
- $s = (s_1, \ldots, s_N)$ the observations of the N
- B the memory buffer

with this notation the loss function in MADDPG reads as

$$\mathcal{L}(\theta_{Q_i}) = \mathbb{E}_{(s,a,r,s')\sim U(B)} \left[Q(s, a_1, \ldots, a_N|\theta_{Q_i})|_{a_i=\mu_i(s_i)} - y_i \right]^2 \tag{3}$$

where the symbol $\mathbb{E}_{(s,a,r,s')\sim U(B)}$ indicates the expected value calculated on a batch of experiences (s, a, r, s') uniformly sampled from the buffer memory B and y_i indicates the target values

$$y_i = r_i + \gamma Q'_i(s', a'_1, \ldots, a'_N|\theta'_{Q_i})|_{a'_i=\mu'_i(s_i)} \tag{4}$$

As in [1] the algorithm to train the agent in Tennis Unity Environment reads as

```
Algorithm 1: MADDPG [1]
Initialize buffer memory $B$ to capacity $N$
Initialize all the critic $Q_i(s, a|\theta_{Q_i})$ and actor $\mu_i(s_i|\theta_{\mu_i})$ networks with
  random weights $\theta_{Q_i}$ and $\theta_{\mu_i}$
Initialize target networks $Q'_i$ and $\mu'_i$ with weights $\theta'_{Q_i} = \theta_{Q_i}$ and
  $\theta'_{\mu_i} = \theta_{\mu_i}$
for episode $i = 1$ to Env Solved do
    Reset Tennis Unity Environment and set the input state of the first
      step $s^{(1)}$
    Initialize the random noise $\mathcal{N}$
    for time–step $t = 1$ to $T$ do
        For each agent select action by actor networks
          $a_i^{(t)} = \mu_i(s_i^{(t)}|\theta_{\mu_i}) + \mathcal{N}^{(t)}$
        The agents executes action $a^{(t)} = (a_1^{(t)}, \ldots, a_N^{(t)})$ and record
          $r^{(t)} = (r_1^{(t)}, \ldots, r_N^{(t)})$ and next state $s^{(t+1)} = (s_1^{(t+1)}, \ldots, s_N^{(t+1)})$
        Store the experience $(s^{(t)}, ^{(t)}, r^{(t)}, s^{(t+1)})$ in the buffer memory $B$
        for agent $i = 1$ to $N$ do
            Sample random mini–batch of size $D$ experiences from $B$
            Update the $i$th critic network parameters $\theta_{Q_i}$ by minimizing
              the loss function $\mathcal{L}(\theta_{Q_i})$ in (3)
            Update the $i$th actor network parameters $\theta_{\mu_i}$ by maximizing
              $Q_i(s, \mu_1(s_1|\theta_{\mu_1}), \ldots, \mu_N(s_N|\theta_{\mu_N})|\theta_{Q_i})$ with gradient ascent
        end
        Perform the soft update of each agents

            $\theta'_{Q_i} \leftarrow \tau\theta_{Q_i} + (1-\tau)\theta'_{Q_i}$
            $\theta'_{\mu_i} \leftarrow \tau\theta_{\mu_i} + (1-\tau)\theta'_{\mu_i}$

         Set $s_t = s_{t+1}$
    end
end
```

3 The Tennis Unity Environments

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.

The state space of each agent has dimensions 24 and contains the position and the velocity of the ball and the racket. For each agent 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, the 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. The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

4 The hyperparameters of the algorithm and the outcome

4.1 The hyper parameters for both networks

Discount factor γ $= 0.99$

Buffer memory size N $= 1 \times 10^5$

Batch size for random sampling D $= 128$

Parameter of soft update τ $= 6 \times 10^{-2}$

Kind of optimizer: Adam (adaptive moment estimation)

Learning rate of Adam optimizer $= 1 \times 10^{-4}$

Input layer $= 24$ (the state space size)

4.2 The actor network

The actor network is a fully forward connected neural network with

first hidden layer $= 256$ nodes with ReLU activation function

second hidden layer $= 128$ nodes with ReLU activation function

output layer $= 2$ (the number of action) with tanh activation function (since each actions has to be a value in a range $(-1, +1)$

4.3 The critic network

The critic network has a slightly different architecture

first hidden fully connected layer $= 256$ nodes with ReLU activation function

first hidden layer (not connected to the input layer) $= 2$ nodes (they take as input the action given by the actor network)

second hidden fully connected layer $= 128$ nodes with ReLU activation function

output layer $= 1$ the action value function $Q(s, a)$

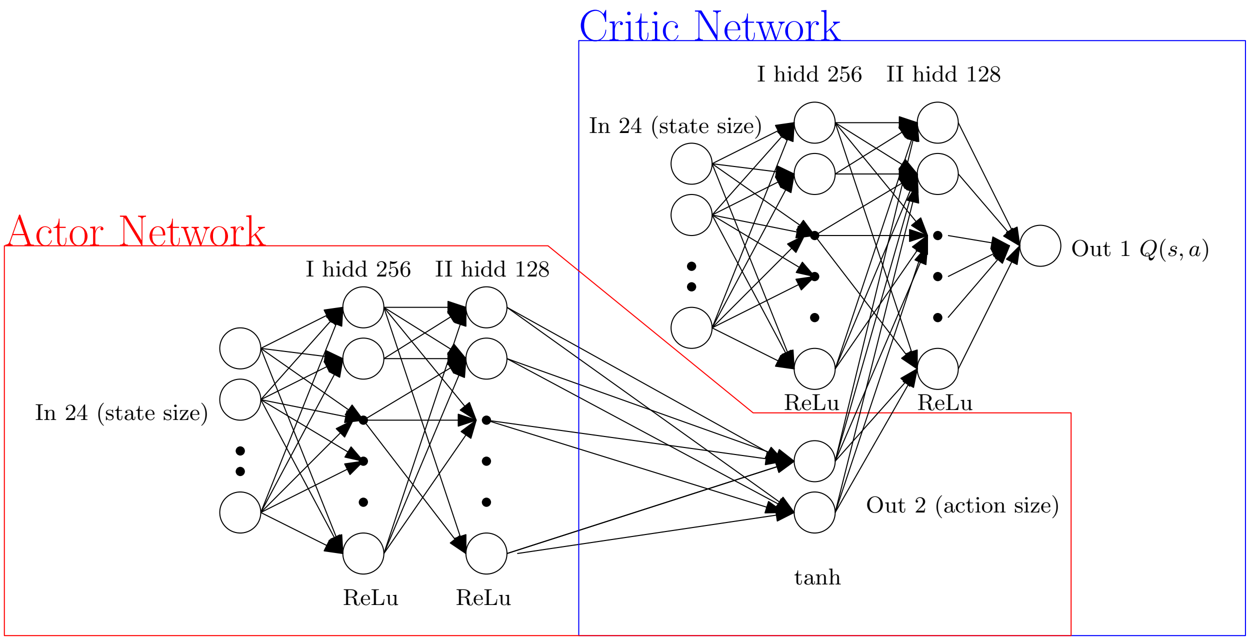

Figure 2: The Actor–Critic network

4.4 The outcome

As we can see in figure 3, by the aforementioned hyper–parameters, the agent solved the game at episode 1566, which means that the average score from episode 1567 to 1668 (included) of the max score of the 2 agents was greater than +0.5.

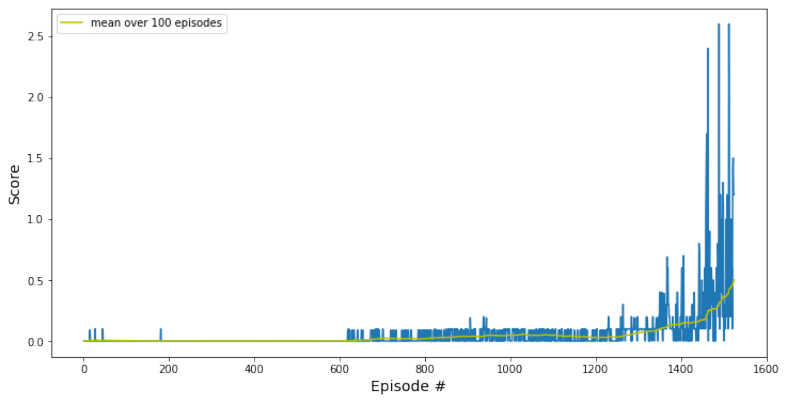

Figure 3: In this test the agent solved the environment at episode 1566 with an average score of 0.502.

5 Insights for future work

Since MADDPG share a similar architeture of DDPG, almost the same improvements and insights can be proposed. For instance we could prioritize the experiences in memory buffer as in Prioritized Experience Replay (PER) [3]. Alternative multi actor–critic agents can be proposed as in Advantage Actor–Critic (A2C) or Asynchronous Advantage Actor–Critic (A3C) [4]. True policy gradient methods could work well in this context as well. We could mention the original Proximal Policy Optimization (PPO) [5] or Trust Region Policy Optimization (TRPO) [6]. Recently the team of OpenAI has started to defeat amateur human teams at Dota 2 with their Open AI 5.

References

[1] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. arXiv preprint arXiv:1706.02275, 2017.

[2] 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. arXiv preprint arXiv:1509.02971, 2015.

[3] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

[4] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[6] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In International conference on machine learning, pages 1889–1897. PMLR, 2015.