Multi Agent Deep Deterministic Policy Gradients to solve Cooperative Environment

Mateo Neira

August 27, 2020

Introduction

In this project we use a modified implementation if multi agent deep deterministic policy gradient (MADDPG) to solve a collaborative environment. MADDPG is an off-policy learning algorithm for multi-agent learning in cooperative, competitive and mixed environments proposed in [4]. The algorithm is an extension to deep deterministic policy gradients (DDPG) [3] framework. DDPG is an off-policy algorithm, and samples trajectories from a replay buffer of experiences, using a function approximator for both Q-Value estimation and Policy.

For the multi-agent setting, MADDPG operates under the following constraints: 1) learaned policies of each agent can only make use of their own observations, 2) the environment dynamics model is not assumed to be differentiable , 3) no assumptions are made regarding the structure of the communication methods between agents. This is achieved by having a augmenting the critic for each agent with extra information about the policies of other agents, without modifying the actor network.

Here we implement a simplified version of the original MADDPG algorithm proposed in [4] to solve a collaboration task. The task requires two agents to play tennis with the goal of keeping the ball in play for as long as possible.

0.1 Environment

The environment used for this project is the Reacher environment from Unity, using its Machine Learning Agents Toolkit [1]. The environment consists of 2 agents that control rackets to hit a ball over a net.The goal of each agent is to keep a ball in play for as long as possible. A reward of +0.1 is received if an agent hits the ball over the net and -0.01 every time an agent lets the ball hit the ground or hits the ball out of bounds. The agent perceives a 24 dimensional state space corresponding to position, velocity, and orientation of the ball and racket. The agent can take actions by moving towards or away from the net and jumping (for a total of 2 dimensions), that range from -1 to 1. The environment is considered solved when the average score of the max between the two agents is at least +.5, averaged across 100 episodes.

1 Implementation

1.1 Deep Deterministic Policy Gradient

The implementation used follows the original algorithm proposed in [4] with the difference that the critic network of each agent only receives local information of the environment (modified MADDPG, Algorithm 1).

1.1.1 Actor-Critic Network

We used Adam [2] for learning the neural network parameters with a learning rate of 10^{-3} and 10^{-3} for the actor and critic respectively. For Q we did not use a L_2 weight decay, as it decreased performance, and used a discount factor of $\gamma = 0.99$. For the soft target updates we used $\tau = 0.001$. The neural networks used the rectified non-linearity for all hidden layers. The final output layer of the actor was a $\tanh$ layer, to bound the actions. The networks had 2 hidden layers with 128 units each. Gradient clipping was used on the Critic network in order to avoid exploding gradients, which improved learning performance. Actions were not included until the 2nd hidden layer of Q.The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$ and $[3 \times 10^{-3}, 3 \times 10^{-3}]$. This was to ensure the

Algorithm 1 MADDPG algorithm (adapted from original paper) [3])

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ for each agent.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$ for each agent.
Initialize replay buffer R for each agent.
Initialize epsilon for noise decay ϵ for each agent.
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration for (independent process for each agent)
  Receive initial observation state $\mathbf{x}$
  for t = 1, T **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration.
    Execute action $a = (a_i, ..., a_N)$ and observe reward r and observe new state $\mathbf{x}'$
    for each agent i, store transition (s_i, a_i, r_i, s_{i+1}) in R
    for agent $i = 1$ to N **do**
      sample a random minibatch of N transitions (s_j, a_j, r_j, s_{j+1}) from R
      Set $y_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimising the loss on the clipped gradient of: $L = \frac{1}{N}\sum_j (y_j - Q(s_j, a_j|\theta^Q))^2$
      Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_j \nabla_a Q(s, a|\theta^Q)|_{s=s_j, a=\mu(s_j)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_j}$$

      Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

      update epsilon: $\epsilon \leftarrow max(0.05, \epsilon - 1e^{-6})$
    end for
  end for
end for

initial outputs for the policy and value estimates were near zero, similar to the original DDPG paper. The other layers were initialised from uniform distributions $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$ where f is the fan-in of the layer. We trained with minibatch sizes of 1024, and replay buffer size of 10^6.

1.1.2 Ornstein–Uhlenbeck process

The original DDPG algorithm uses an Ornstein—Uhlenbeck process [5] to generate temporally correlated exploration. The Ornstein-–Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centred around 0. We use a $\theta = 0.2$ and $\sigma = 0.05$. Additionally a noise decay ϵ was added to slowly decrease the noise at each training step.

2 Results

With this implementation of MADDPG we were able to solve the environment in under 1,472 episodes, as seen in fig 1. Performance reaching an average reward over 100 episodes of around +1.4 in 1,600, slightly drops after 1,750 episodes, and continues to improve to reach a average reward of around +1.6.

3 Conclusions

We implemented and tested a simplified version of MADDPG to solve a cooperative environment. The main difference in our implementation is that the Q-Network for each agent only recieves local information about the environment. For this particular environment this works well and both agents are able to learn, after around 1700 episodes performance drops but recovers and continues to improve reaching a score of around +1.6 . For future extensions, we can implement MADDPG as set in the original paper with a centralised Q-network for each agent.

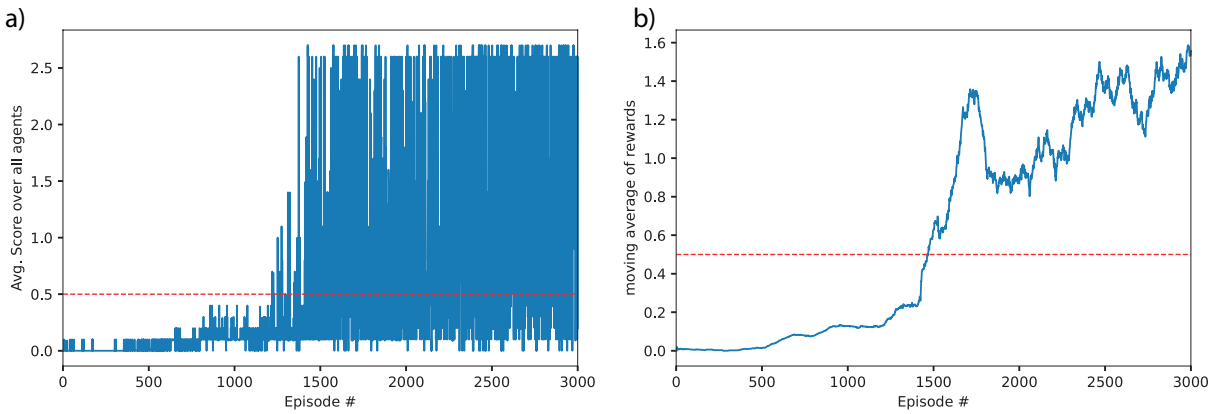

Figure 1: Training performance. a) max score over all agents for each episode. b) moving average of 100 episodes of max score over all agents.

References

[1] Arthur Juliani et al. “Unity: A general platform for intelligent agents”. In: *arXiv preprint arXiv:1809.02627* (2018).

[2] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[3] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[4] Ryan Lowe et al. “Multi-agent actor-critic for mixed cooperative-competitive environments”. In: *Advances in neural information processing systems*. 2017, pp. 6379–6390.

[5] George E Uhlenbeck and Leonard S Ornstein. “On the theory of the Brownian motion”. In: *Physical review* 36.5 (1930), p. 823.