Project Collaboration and Competition

1. Introduction

In this project there are two different agents where every agent is supposed to play tennis. The purpose of the agents is to keep playing for as long as possible without letting the ball touch the ground or shooting the ball out of the court. The action space consists of two continuous actions which are equivalent to movement regarding front, back and jumping.

Each agent gets a +0.1 reward for hitting the ball and the ball goes over the net. However, if the agent lets the ball hit the ground or shoots the ball out of bounds it receives a -0.01 reward.

The observation space consists of 24 different states for each agent which corresponds to the position and velocity of the ball and racket.

The goal of the agents is to achieve a rolling mean of more than 0.5 over 100 episodes averaged over the 2 agents. That means we that we sum up the score for every agent during an episode and then take the max of the agents' accumulated score and then compute a rolling mean over the 100 last episodes.

In Figure 1 a visual of the Tennis environment is depicted with two different competing agents.

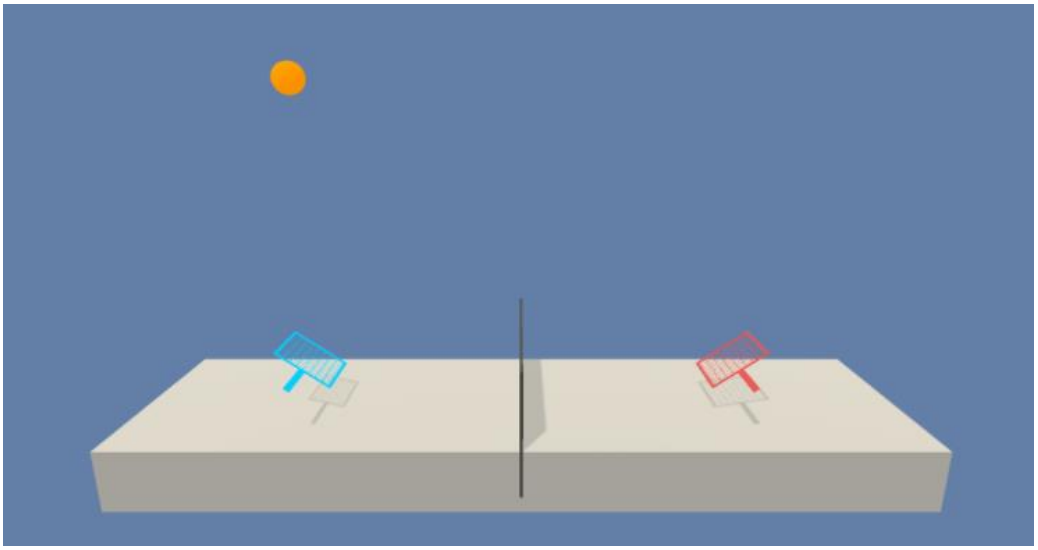

Figure 1: The Tennis environment with two agents.

MADDPG

In this project the Multi Agent Deep Deterministic Policy Gradient (MADDPG) is used. The pseudocode for this algorithm is depicted in Figure 2. Each agent has its own network and similar to the DDPG, the MADDPG algorithm consists of two networks, one for the critic and one for the actor, where each network has a target and local network which is similar to the DQN with a target network.

The policy network also as a noise term to add some exploration to the actions taken. The noise is governed by the Ornstein-Uhlenbeck process for more details see this paper: https://arxiv.org/pdf/1509.02971.pdf

However apart from the DDPG in MADDPG the critic also evaluates the other agents states and actions. Hence for each agent's critic network we also must send in the other agent's states and actions for evaluation. This is only done during training, when executing the policy the critics aren't involved.

The complete pseudocode for the MADDPG algorithm is shown in Figure 2.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **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 actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Figure 2: Pseudocode of MADDPG, from https://arxiv.org/pdf/1706.02275.pdf

2. Method and parameters

The method used in this project is the Multi Agent Deep Deterministic Policy Gradient (DDPG). The code in this project is based on the Udacity's DDPG code used to solve the bipedal environment and it can be found in this repository.

Hyperparameters used and their respective values:

- Batch size: 256
- Gamma: 0.995 (discount factor)

- Learning rate for actor and critic network: 5e-4
- Tau: 0.002 (soft update blending factor between target and local network)
- Update episode frequency: 1
- Number of updates of the actor and critic networks: 3
- Buffer size: 1e5 (Size of replay buffer)
- Optimizer: Adam with default settings

Running parameters:

- Episodes: 3000 (or until termination if target has been reached but average reward drops below 0.4.
- Time steps: Until termination
- Noise decay per timestep: 0. 999999 (the noise decay is started once an average reward of mora than 0.1 is reached).

2.1. Actor Network

Layers and units:

- Input layer: 24
- Hidden layer 1: 512
- Hidden layer 2: 256
- Output layer: 2

Activation functions: Relu between all layers except the last where we have a tanh function to get a value between -1 and +1.
Other: After the first layer we have a batch normalization function.

2.2. Critic Network

Layers and units:

- Input layer: 24*2 + 2*2(own states + other states + own actions + other actions)
- Hidden layer 1: 512
- Hidden layer 2: 256
- Output layer: 1

Activation functions: Relu for all layers except the last one that doesn't have any.
Other: After the first layer we have a batch normalization function.

3. Results

In Figure 3 the results of the run with the above-mentioned configurations are depicted. We can see that the rewards per episode seems to get above the target after around 700 episodes. The rewards per episodes is very unstable jumping between 0 and at max around 2.5. The number of 0's scores seems to drop of a bit after around 700 episodes. At around episode 1700 we see the first really high score of about 2.5, we reach a few more of those scores until around 2200 when the models starts to seem doing worse.

The rolling mean target of 0.5 is fulfilled at around 1750 episodes then the rolling mean goes up to around 0.6 and stays there for about 500 episodes. At episode 2100-2200 the average mean seem to drop quite fast down to 0.4 which is when the algorithm is terminated.

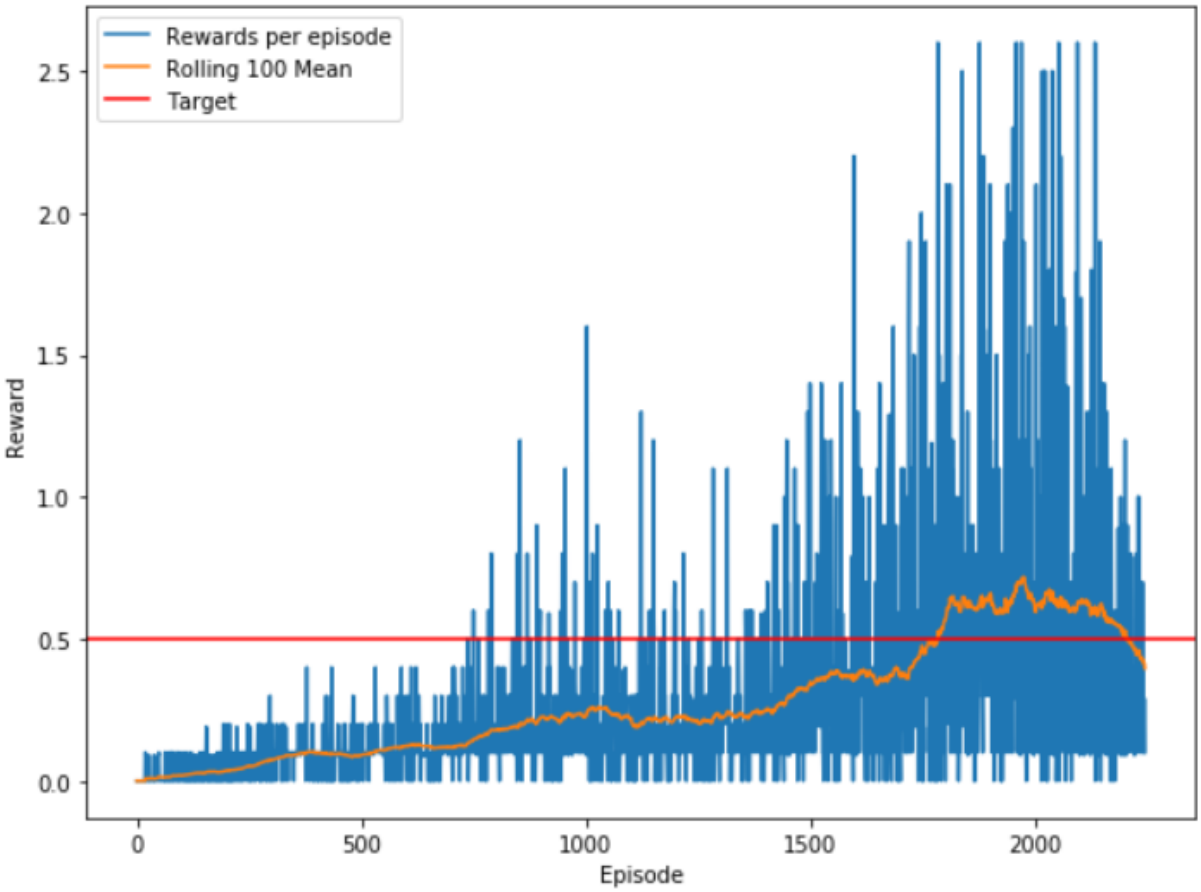

Figure 3: Rewards per episode and rolling mean over 100 compared to the target.

4. Discussions

Many different configurations of different hyperparameters were. It took me a long time before I started to get some idea about what parameters that work to accomplish the target.

Recommended steps to further investigate would be to both reduce the size of the network but also to increase it. It would be interesting to reduce the network size to get faster run times to easier find optimal settings for the hyperparameters. However, it would also be interesting to increase the network size to increase the capacity and see how large scores one could achieve. However, this would come with the cost of longer run times.

I would like to run the environment with a learning rate scheduler to have an adaptive time step dependent on the loss functions. This would help to reach the target goal faster and then hopefully stabilize the training thereafter.

In addition, the update frequency, number of updates and the blending factor should also be somewhat adaptive dependent on the loss output of the networks.

For the critic I combined the states and actions already at the input layer, it would have been interesting to see the effect of adding the action values in the second layer.

I would also do more testing with the decay rate; I believe I let the model explore too much and I should have decayed the action noise fast to let the model exploit more.