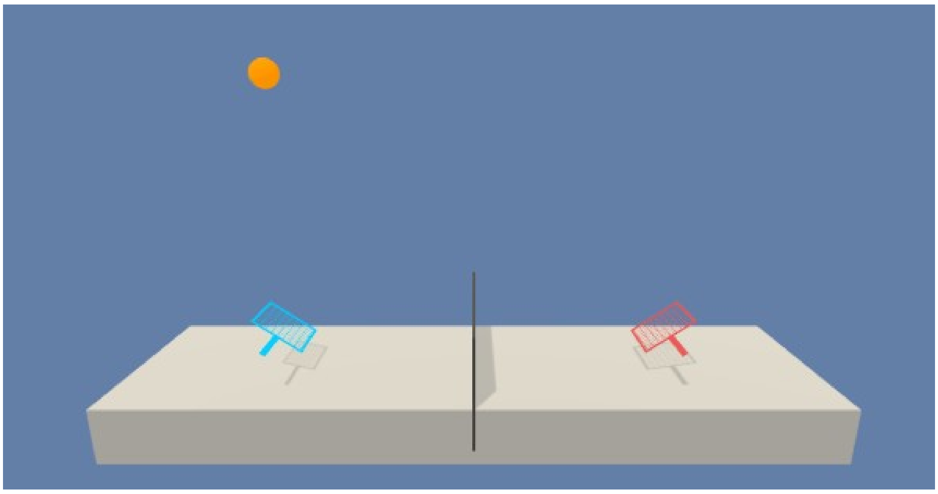
22.03.2020

Udacity Nanodegree Deep Reinforcement Learning
Sören Klingner Report

Collaboration & Competition

1. Introduction

The goal of this project is to train 2 agents in the Unity Tennis environment to play tennis. 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. Thus, the goal of each agent is to keep the ball in play.

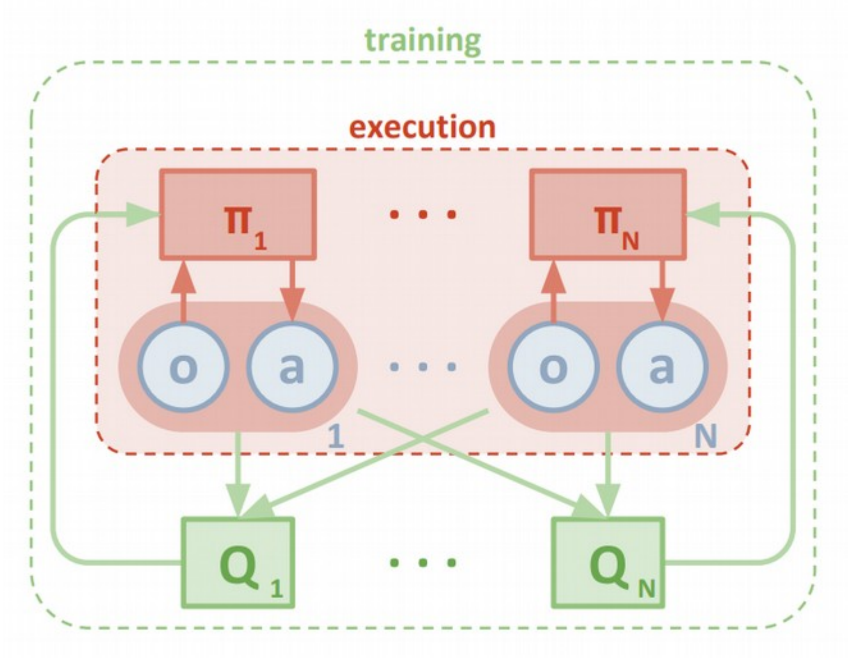


The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

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.

2. Algorithm

The agent is trained with the Multi Agent DDPG. The full algorithm is described in the methods section of the paper[1]. The information flow in a MADDPG algorithm is described below.



The environment was solved by tuning the DDPG algorithm described in the second project of this course (see continuous control). In detail, an agent called MADDPG (Multi-Agent-DDPG) handles the training of two DDPG agents, each one of them corresponding to one of the two players. This MADDPG agent handles a common replay buffer for its two sub-agents.

Each one the DDPG agent has:

- An actor seeing the state corresponding to its player and performing an action for it
- A critic seeing the whole picture: The state of both agents, and estimating the action-value function.

The training procedure goes like this:

- States, actions, rewards, next states are collected in a single replay buffer.
- Each agent samples a batch of experiences from the replay buffer and asks the other agent to propose an action for each state and next states
- Each agent is trained like DDP agents, with minor differences. The critic takes as input the states of all agents. The best actions and best next actions are estimated thanks to both agents.

1 https://arxiv.org/abs/1706.02275

3. Hyperparameters and Architecture

Hyperparameters

BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 128 # minibatch size
LR_ACTOR = 1e-3 # learning rate of the actor
LR_CRITIC = 1e-3 # learning rate of the critic
WEIGHT_DECAY = 0 # L2 weight decay
LEARN_EVERY = 1 # learning timestep interval
LEARN_NUM = 1 # number of learning passes
GAMMA = 0.99 # discount factor

TAU = 7e-2 # for soft update of target parameters
OU_SIGMA = 0.2 # Ornstein-Uhlenbeck noise parameter, volatility
OU_THETA = 0.12 # Ornstein-Uhlenbeck noise parameter, speed of mean reversion
EPS_START = 5.5 # initial value for epsilon in noise decay process in Agent.act()
EPS_EP_END = 250 # episode to end the noise decay process
EPS_FINAL = 0 # final value for epsilon after decay

Actor neural network

The Actor neural network consists of two fully connected layers.

- The input has 24 units to observe a state with a dimension of 24.
- The first hidden layer has a size of 256 units and the second layer layer has 128 units.
- The output has 2 units for one dimension movement and jumping.

Critic neural network

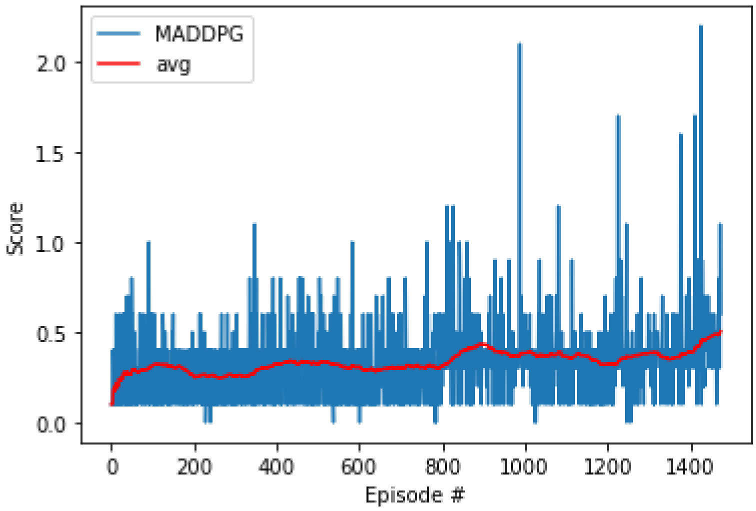
The Critic neural network consists of two fully connected layers.

- The input has 24 units to observe a state with a dimension of 24.
- The first hidden layer has a size of 256 units and the second layer layer has 128 units + 2 (actions).
- The output has 1 unit.

3. Results

Given the chosen architecture and parameters, our results are:

```
Episode 10       Max: 0.400000    Average: 0.190000
Episode 100      Max: 1.000000    Average: 0.311000
Episode 200      Max: 0.500000    Average: 0.254000
Episode 300      Max: 0.500000    Average: 0.252900
Episode 400      Max: 0.800000    Average: 0.322000
Episode 500      Max: 0.400000    Average: 0.327000
Episode 600      Max: 0.400000    Average: 0.305000
Episode 700      Max: 0.600000    Average: 0.303000
Episode 800      Max: 0.600000    Average: 0.308900
Episode 900      Max: 0.500000    Average: 0.429000
Episode 1000     Max: 0.700000    Average: 0.382000
Episode 1100     Max: 0.500000    Average: 0.364000
Episode 1200     Max: 0.700000    Average: 0.321000
Episode 1300     Max: 0.600000    Average: 0.382000
Episode 1400     Max: 0.800000    Average: 0.388000
Episode 1410     Max: 0.500000    Average: 0.388900
Episode 1420     Max: 1.700000    Average: 0.414900
Episode 1430     Max: 2.200000    Average: 0.455900
Episode 1440     Max: 0.700000    Average: 0.457900
Episode 1450     Max: 0.700000    Average: 0.471800
Episode 1460     Max: 0.600000    Average: 0.486800
Episode 1470     Max: 0.600000    Average: 0.483800
Environment solved in 1375 episodes. Average: 0.503800 over past
100 episodes
```



These results meets the project's requirements as the agents are able to receive an average score of 0.504 over the last 100 episodes, from episode 1375.

4. Ideas for Future Work

Further improvements could be:

- Further tune the hyper-parameters to converge faster, automated grid based
- Implement a prioritized replayed-buffer
- Train two pairs of agents, and then switch the partners, to see if they are able to play with another player.