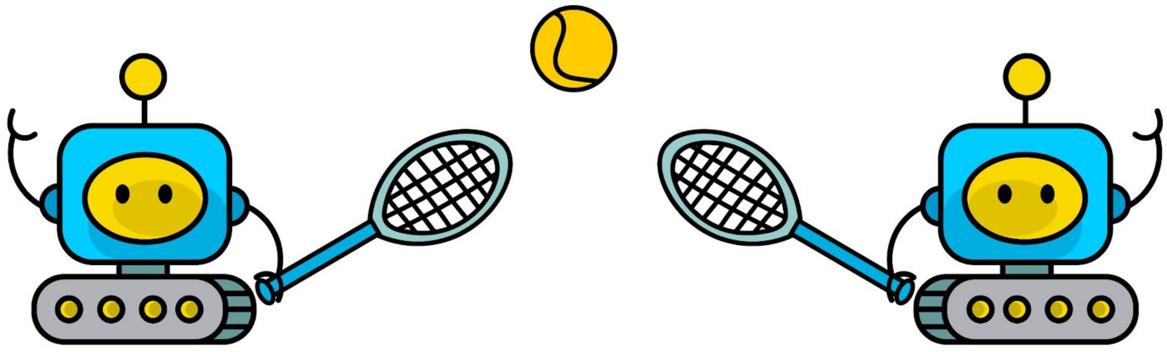
Phase 3 Deep Reinforcement Nanodegree Scholarship from Facebook

Nanodegree Program by Udacity

9/03/2019

PROJECT COLLABORATION AND COMPETITION

By Sabrina Palis



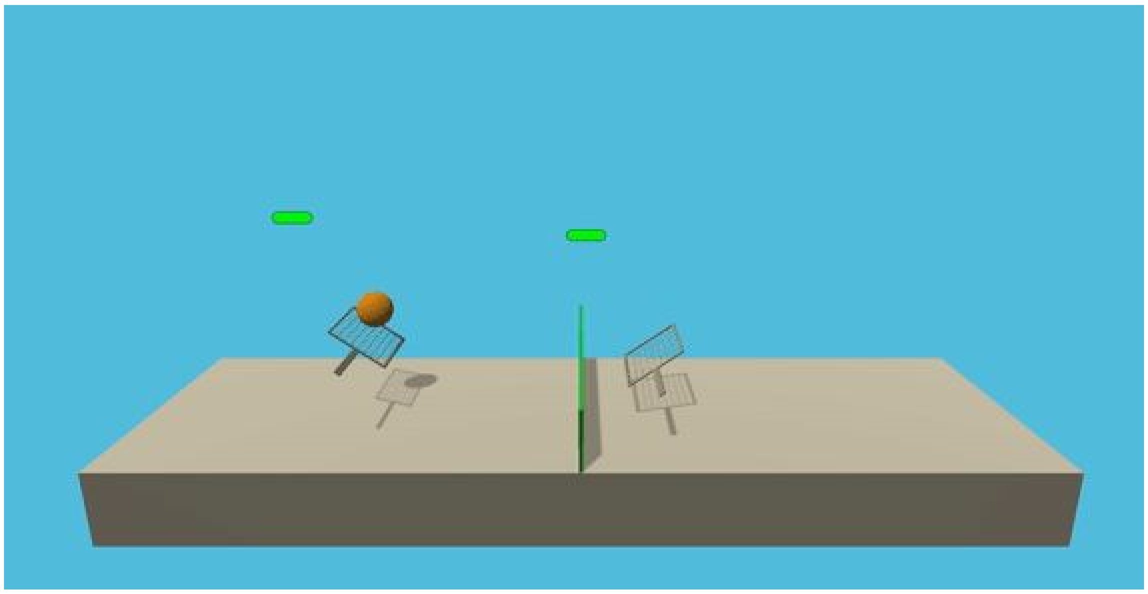
Project description and specifications by Udacity

Image Credit Sabrina Palis

For this project, we will work with the Tennis Unity 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 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.


Screenshot of the Unity Tennis Environment

Solving the Environment

The task is episodic, and in order to solve the environment, your 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.
- This yields a single **score** for each episode.

The environment is considered solved, when the average (over 100 episodes) of those **scores** is at least +0.5.

Implementation: MADDPG

Rationale

The Collaboration and Competition project is the third project of the Deep Reinforcement Learning Nanodegree, DRLND, by Udacity. It is the opportunity to **demonstrate an understanding of the relevancy multi-agents systems as a way to solving AI.**

If we want to build agents that will have to be used in the real world, i.e. a multi-agent world, we want them to be able to learn from one another and from human beings in a very complex environment.

In many environments, agents have to show a mixture of competitive and collaborative behaviour.

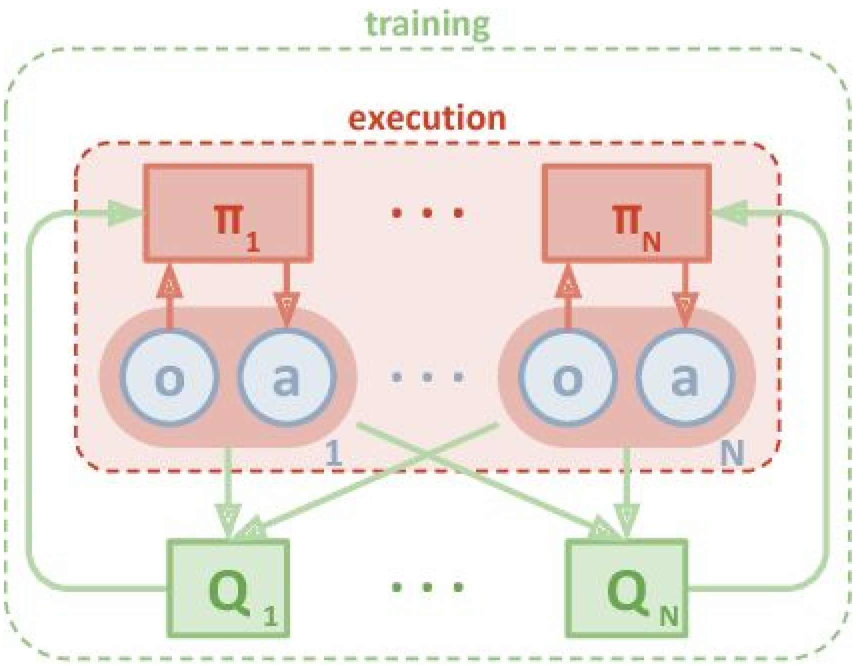
Learning algorithm

Centralized training with decentralized execution

This implementation follows the guidance of the "Multi Agent Actor Critic for Mixed Cooperative Competitive environments " .

DDPG is an off-policy actor critic algorithm that uses the concept of target networks. The input of the action network is the current state while the output is a real value or vector representing an action chosen from a continuous action space.

The framework of **centralized training with decentralized execution** is adopted in this implementation. This implies that some extra information is used to ease training, but that information is not used during the testing time.



Multi-Agent Actor-Critic for MixedCooperative-Competitive Environments
arXiv:1706.02275v3 [cs.LG] 16 Jan 2018

During training, the critic for each agent uses extra information like states observed and actions taken by all the other agents.

There is one actor for each agent. Each actor has access to only its agent's observations and actions.

During the execution time, only the actors are present and hence all observations and actions are used.

Learning critic for each agent allows us to use a different reward structure for each. Hence the algorithm can be used in all, cooperative, competitive, and mixed scenarios.

MADDPG

The MADDPG algorithm uses **two deep neural networks**, one is the actor and the other the critic. The **actor** is used to approximate the optimal policy deterministically by always outputting the best action for any given state. The **critic** learns to evaluate the optimal action value function by using the actor's best believed action argmaxa Q(s,a).

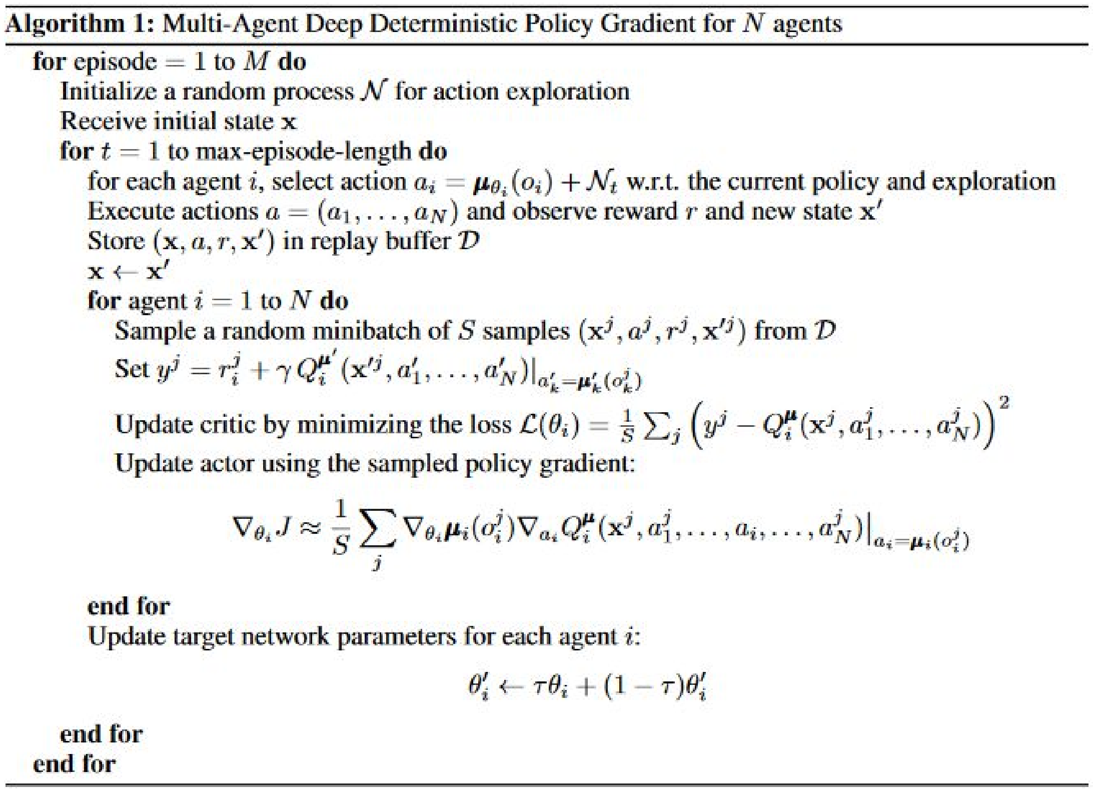
The DDPG includes **two copies of the network weights** for each network: a regular for the actor, a regular for the critic, a target for the actor, and a target for the critic.

In DDPG the target networks are updated using a **soft update strategy** which consists in slowly blending your regular network weights with your target network weights. Every time step, the target network is 99.99% target network weights and only 0.01% of the regular network weights. This soft update strategy helps to ensure the learning stability of the DDPG.

A **replay buffer** allows the DDPG agent to learn offline by gathering experiences.

Noise is added through the **Ornstein Uhlenbeck process** to get a better exploration, thus accelerating convergence. The OU process is considered to be **smooth in time**, as opposed to pure Gaussian noise which is not temporally correlated and therefore less suitable for control tasks.

Pseudocode



```
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 $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
            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:
                $\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$
        end for
        Update target network parameters for each agent $i$:
                $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    end for
end for
```

Multi-Agent Actor-Critic for MixedCooperative-Competitive Environments
arXiv:1706.02275v3 [cs.LG] 16 Jan 2018

Network architecture

A simple architecture yielded good results with fully-connected linear layers and ReLu activations.

Note that:

- the actor output is a tanh layer scaled to be between $[-b,+b]$, $b \in R$.
- the critic network takes both the state and the action as inputs, and the action input skips the first layer. This is achieved using torch.cat.

Actor

Linear → ReLu → Linear → ReLu → Linear → Tanh

Critic

Linear → ReLu → Linear → ReLu → Linear

Note the use of torch.cat to concatenate along an existing dimension.

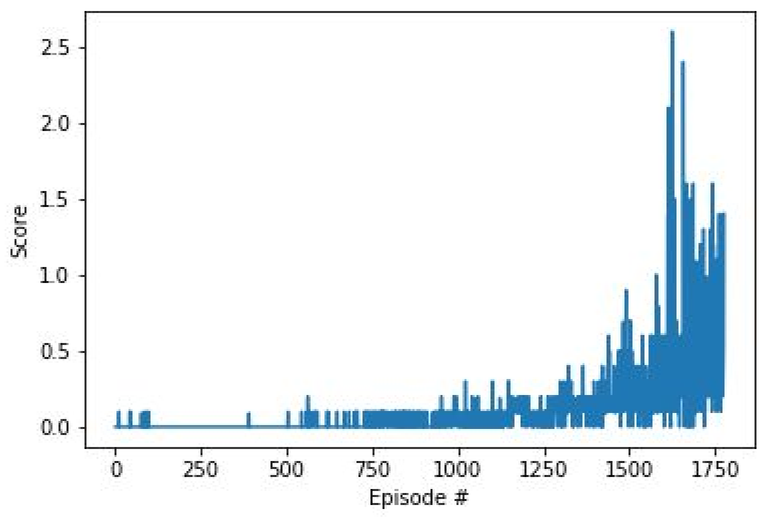
```
xs = F.relu(self.fcs1(state))
x = torch.cat((xs, action), dim=1)
```

Chosen hyperparameters

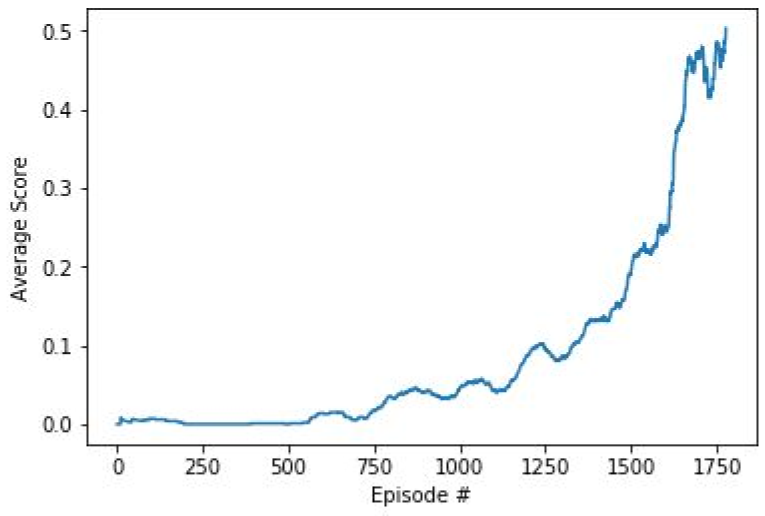
● BUFFER_SIZE = int(5e5)	replay buffer size
● BATCH_SIZE = 256	minibatch size
● GAMMA = 0.99	discount factor
● TAU = 1e-3	for soft update of target parameters
● LR_ACTOR = 1e-4	learning rate of the actor
● LR_CRITIC = 1e-3	learning rate of the critic
● WEIGHT_DECAY = 0	L2 weight decay
● OU sigma = 0.2	standard deviation for the OU noise
● OU theta = 0.15	theta noise

Plot of rewards

The following plots of rewards illustrate that the agent is able to receive an average reward (over 100 consecutive episodes, after taking the maximum over both agents) of at least +0.5. The Tennis Environment Option 2 was solved in 1776 episodes with an average score of +0.503.



MADDPG Tennis Environment Scores



MADDPG Tennis Environment Average Scores

Ideas for future work

The literature dedicated to the MARL field provides challenging ideas for future work, depending on the particular aspect one wants to improve. Here are a few I would like to implement:

- Use MADDPG-MD to improve cooperation by extending dropout technique to robustify communication when applied in multiagent scenarios with direct com-munication.
- Train PPO and A3C to compare learning performances.