

# Deep Reinforcement Learning Specialization – Project 3: Collaboration and Competition

Vlad Negreanu

## 1 Summary

The project consists in training two agents that 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 **24** variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. **2** continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic. 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 agents score of **+0.5** (over **100** consecutive episodes, after taking the maximum over both agents).

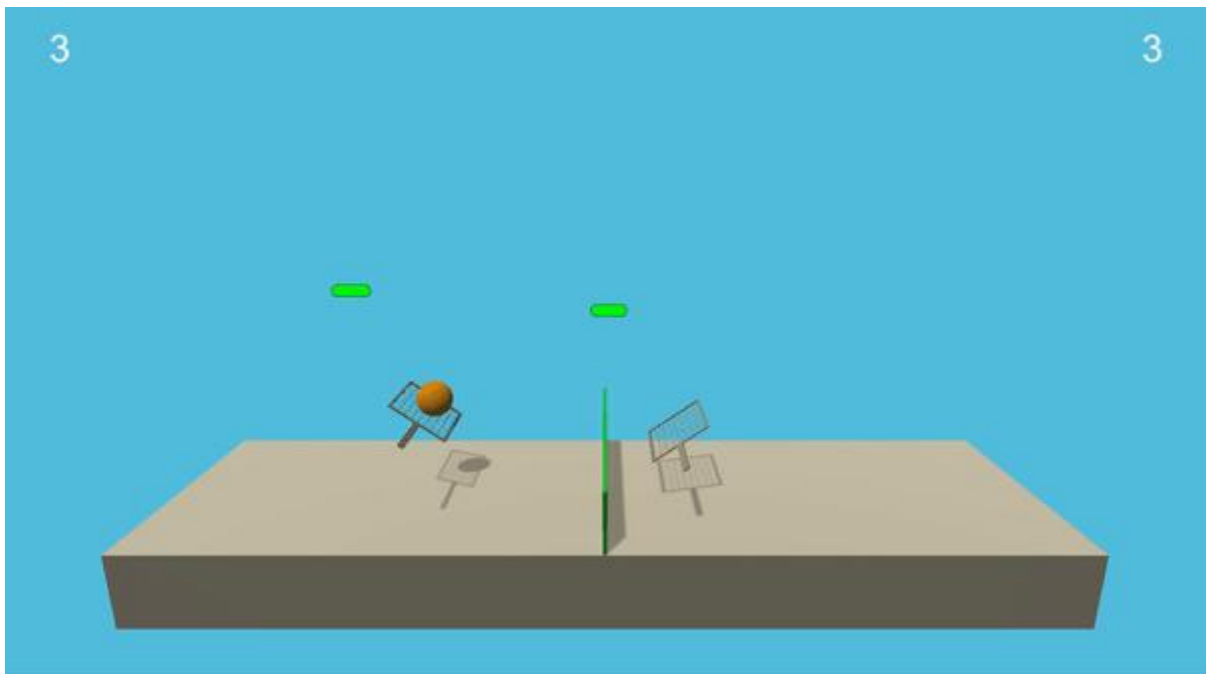


Fig.1 Environment overview

## 2 Reinforcement learning approach

## 2.1 Possible methods

- **Value-based methods (VB)**
- **Policy-based methods (PB)**
- **Actor-Critic methods (AC)**

The **Value-Based (VB)** control method is useful because the agent can use its experiences learned from the environment to estimate an optimal action-value function. **Value-based** methods are used for concrete spaces.

The **Policy-Based (PB)** control method is useful because it can learn directly the optimal policy, without having to maintain a separate value function estimate. **Policy-based** methods can learn either stochastic or deterministic policies so they can be used to solve environment in either finite or continuous spaces.

The **Actor-Critic (AC)** is a hybrid of value and **policy-based** methods in which the **actor** estimates the policy and the **critic** estimates the value function.

## 2.2 Algorithm decision and neural networks description

Algorithm used is **DDPG** (Deep Deterministic Policy Gradient), which is a actor-critic algorithm, combining **DPG** with **DQN** ( Deep QNetwork). **DQN** stabilizes the learning of the Q-function by using experience replay.

The original **DQN** works in discrete space, and **DDPG** extends it to continuous space with the actor-critic framework while learning a deterministic policy.

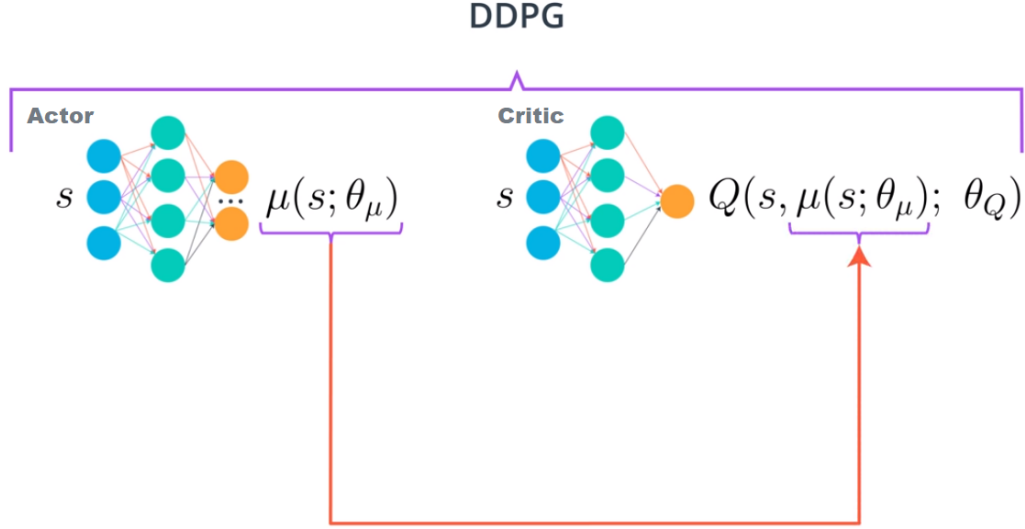


Fig.2 DDPG overview

---

**Algorithm 1** DDPG algorithm

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .  
Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q$ ,  $\theta^{\mu'} \leftarrow \theta^\mu$   
Initialize replay buffer  $R$   
**for** episode = 1, M **do**  
  Initialize a random process  $\mathcal{N}$  for action exploration  
  Receive initial observation state  $s_1$   
  **for** t = 1, T **do**  
    Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise  
    Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$   
    Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$   
    Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$   
    Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$   
    Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$   
    Update the actor policy using the sampled policy gradient:

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

Update the target networks:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{aligned}$$

**end for**  
**end for**

---

Fig.3 DDPG Algorithm

## Actor neural network

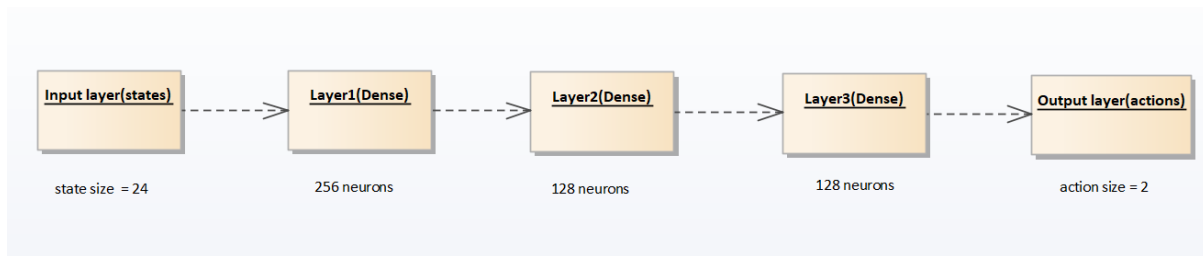


Fig.3 Actor neural network description

## Critic neural network

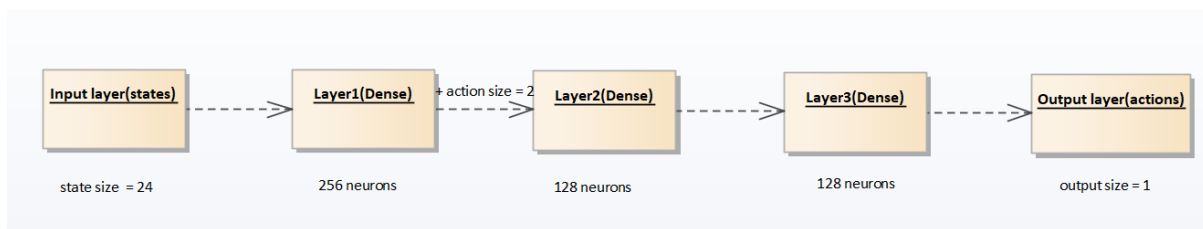


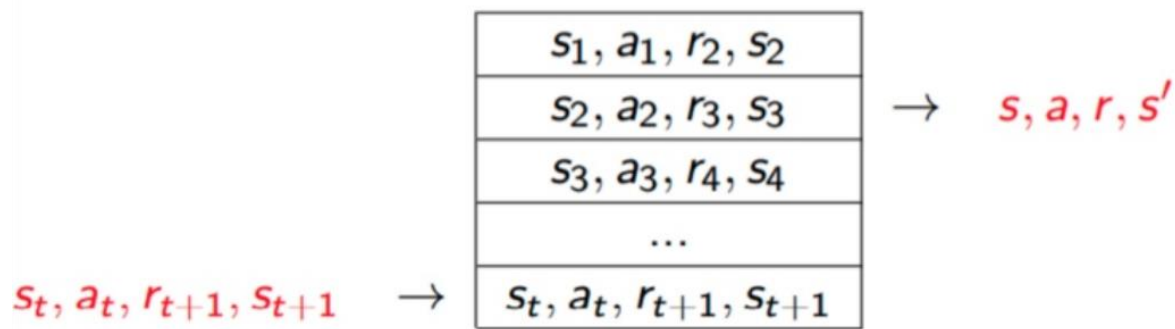
Fig.4 Critic neural network description

## 2.2 Stabilization issues of the neural networks

Since neural networks are considered unstable when representing action values one of the features to be used to reduce the instabilities is **Experience Replay**.

### 2.2.1 Experience Replay

To interact and learn from the environment a sequence of tuples needs to be stored. For this we keep track of a so called **replay buffer** that contains a collection of tuples  $(S, A, R, S')$  gradually added to the buffer as we are interacting with the environment. Buffer size used is **1000000** so we can store 1000000 experiences at a time.



### Replay Buffer – fixed size

Fig.4 Replay buffer with fixed size

#### 2.2.2 Soft target updates

In DDPG we have a regular network for both the actor and the critic. We need to have a copy of these networks. We have a target actor and target critic. The update of the networks is done by slowly blending your regular network weights with your target network weights as follows:

$$\begin{aligned}\theta^{Q'} &\leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1 - \tau)\theta^{\mu'}\end{aligned}$$

with  $\tau \ll 1$  usually 1% (0.01).

#### 2.2.3 Noise for exploration

An exploration policy  $\mu'$  is constructed by adding noise  $N$ :

$$\mu'(s_t) = \mu(s_t | \theta^{\mu_t}) + N$$

We use the **Ornstein-Uhlenbeck process** to model the velocity of a Brownian particle with friction which results in values centered around 0.

Start with Brownian motion:

$$dW_t = N(0, dt)$$

then we add the friction:

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t$$

where  $\theta$  controls the amount of friction to pull the particle towards the global mean  $\mu$ . The parameter  $\sigma$  controls the scale of the noise.

After discretizing the formula above we have:

$$X_{n+1} = X_n + \theta(\mu - X_n)\Delta t + \sigma\Delta W_n$$

## 2.3 Other algorithms and parameters

- **Adam** optimizer algorithm since it uses the best properties of the AdaGrad and RMSProp algorithms to provide an **optimization** algorithm that can handle sparse gradients on noisy problems.
- **Learning rate actor**  $\alpha$  initialized to  $3e-4$ .
- **Learning rate critic**  $\alpha$  initialized to  $3e-4$ .
- **Discount rate**  $\gamma$  initialized to 0.99 very close to 1 meaning that the return objective takes future rewards into account more strongly as the agent progresses through the environment.
- **Minibatch size** is 1024
- **Replay Buffer size** was set to  $10^6$
- **Ornstein-Uhlenbeck noise** process parameters are  $\theta = 0.15$  and  $\sigma = 0.2$
- **Number of agents** was set to 2.

## 3 Result

The agent is attempted to be trained initially over **4000** episodes with each episode having **1000** timesteps. We consider the environment to be solved when the average over the last **100** episodes is greater or equal than **0.5**. In the figure below there is the plot of the score obtained per episodes.

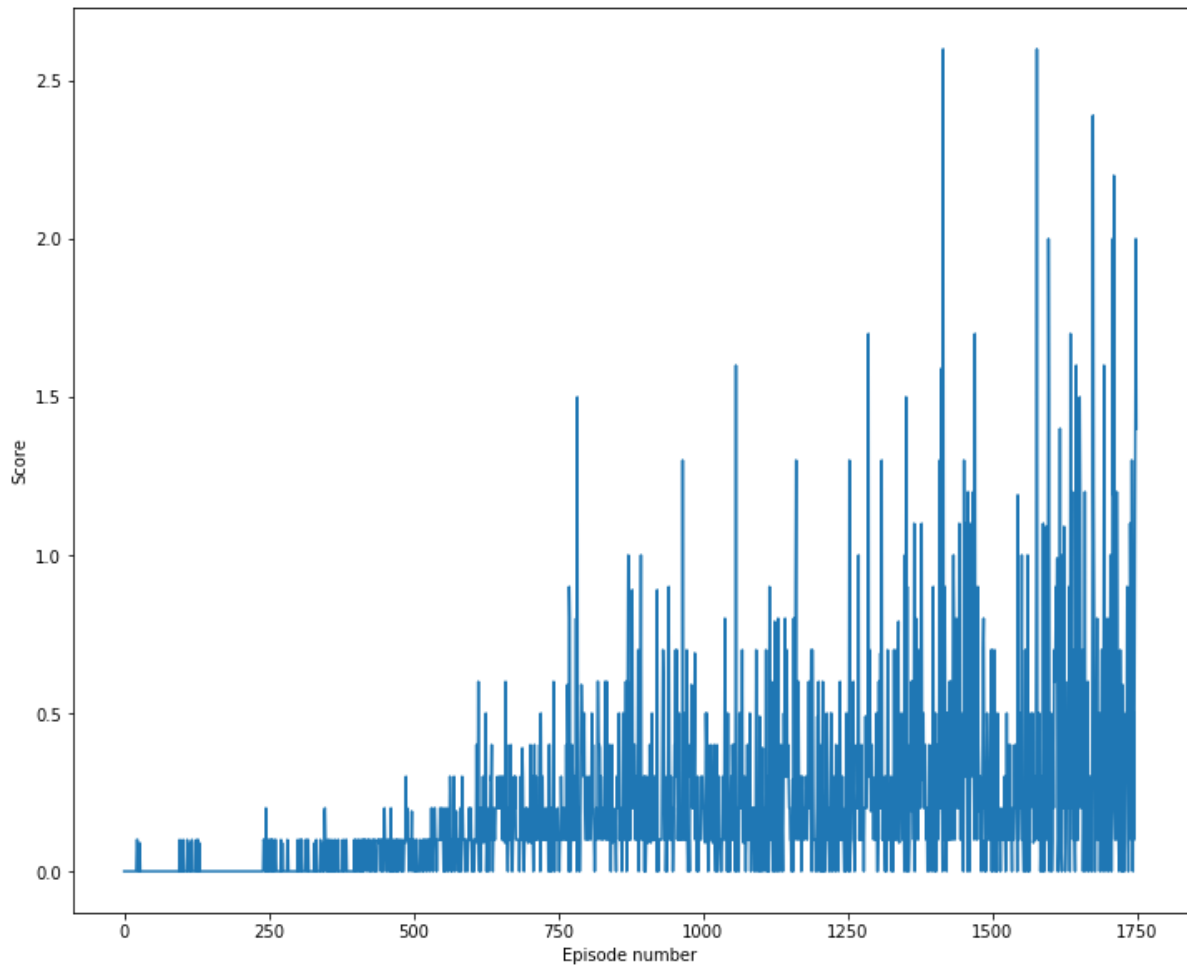


Fig.7 Scores per episode

## 4 Improvements

- **architectural improvement** can be achieved using **TD3**, **SAC** algorithms. Where the actor is updated with delay once every few iterations and there are two critic neural networks.
- **stabilization improvement** the main idea is to give much more perspectives to the experiences procedure since it is important for the learning. A possible improvement can be **Prioritized experience replay**. Also the usage of **gaussian noise** might help instead of **OUNoise** method.