DEEP REINFORCEMENT LEARNING

PROJECT 3. Collaboration and competition.

Markus Buchholz

1. GOAL

In this project, two Agents play the tennis game. The environment is constructed in such a way, that the way that 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.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives from Environment its own, local observation. The Agent can take two different actions, corresponding to movement toward (or away from) the net, and jumping. The actions are defined in are continuous space domain, which further implies on the algorithm design.

The goal of each agent is to keep the ball in play as long as possible.

The task is episodic, and in order to solve the environment (fulfil the project requirement), the 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.

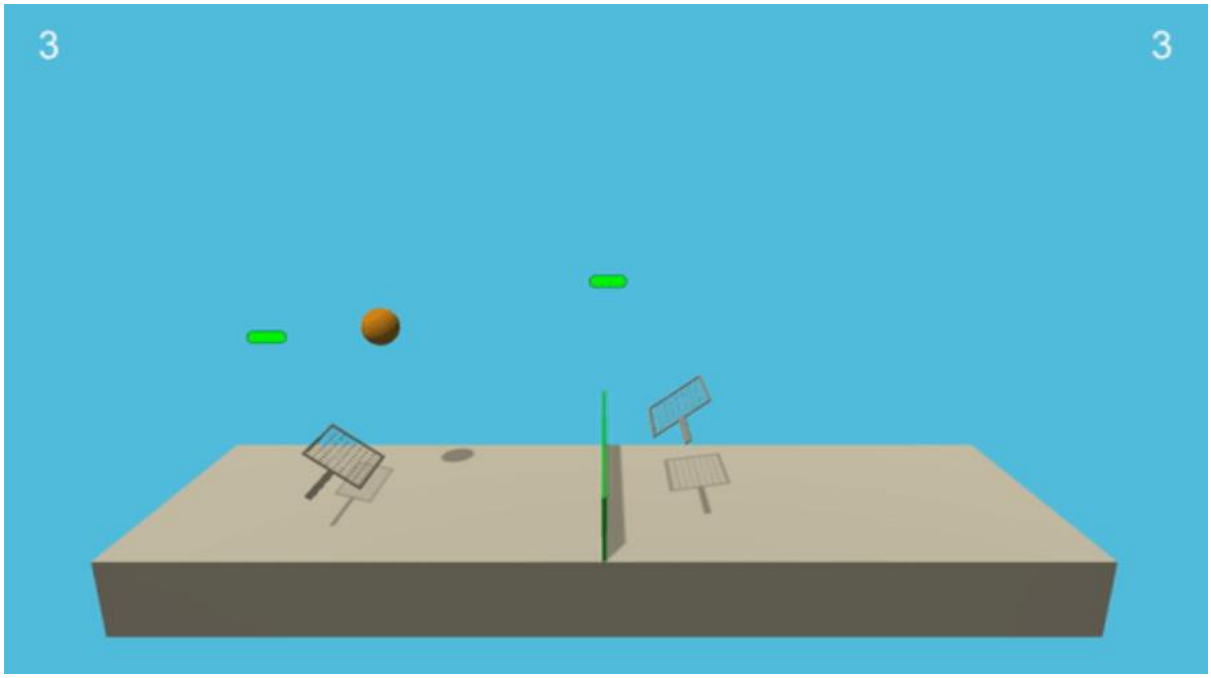

Figure 1. Collaboration and competition (Tennis game) Unity environment

2. INTRODUCTION

On previous project we deployed DDPG algorithm to control two join robot arms. As an individual human being we comprehend that close interaction, collaboration and compete with the others, can yield immense advance in our life quality and successive change in evolution of biological populations. Beside our intelligence is predefined by genes so our interactions(and gathered experience) with the other Agents (society) can reinforced our general mental capability to perceive differences, plan, solve unconventional problems, think abstractly, comprehend complex ideas. Therefore, the crucial step to building artificially intelligent systems based on deep reinforcement learning approach is to involve interaction between multiple agents. The Agent by common, consistent interaction can co-evolve together to solve the problem.
Demonstrated previously, traditional reinforcement learning approaches such as Q-Learning or policy gradient are miserable adapted to multi-agent environments. Applied algorithms allow learning the Agents individually so the environment seen from the prospective of a single agent, changes dynamically. Dynamic changes imply that the environment is no predictable (non-stationary) of the environment.

In this project, we will solve our environment by deploying the DDPG algorithm for a multi-agent purposes. The foundation of following implementation was described in our previous project, where we *exploited* separate critic for each agent. It this case, we use also separate Critic for each Agent. Unique in our solution is deployed experience memory, which is common for both Agents.
Finally, after the training the Agents (while playing the game) decide about the action independently.

3. ALGHORITM

Bearing in mind previous implementation, where one Agent controlled the robot arm, we decided to utilize the same conception for multi agent environment. Deep Deterministic Policy Gradient algorithm was evolved (DDPG).

Complete derivation of the Actor – Critic concept (DDPG algorithm) was portrayed also previously but we need to remind critical issue in order to deploy the concept well enough.
In DDPG, the Actor is used to approximate the optimal policy deterministically. That means we want always to generate the best believed action for any given state.
The Actor follows the policy-based approach and learns how to act by directly estimating the optimal policy and maximizing reward through gradient ascent. The Critic however, utilizes the value-based approach and learns how to estimate the value of different state – action pairs

Based on this simplified remind we can again express that the Agents learn by interacting with environment and adjusting the probabilities of good and bad actions (Agent learn to act). In parallel we use a Critic, which is to be able to evaluate the quality of actions more quickly (proper action or not) and speed up learning.
As a result of merge Actor – Critic we utilize two separate neural networks. The
role of the Actor network is to determine the best actions (from probability distribution) in the state by tuning the parameter θ (weights). The Critic by computing the temporal difference error TD (estimating expected returns), evaluates the action generated by the Actor.
General overview over considered algorithm used in multi agent environment can be depicted as follows,

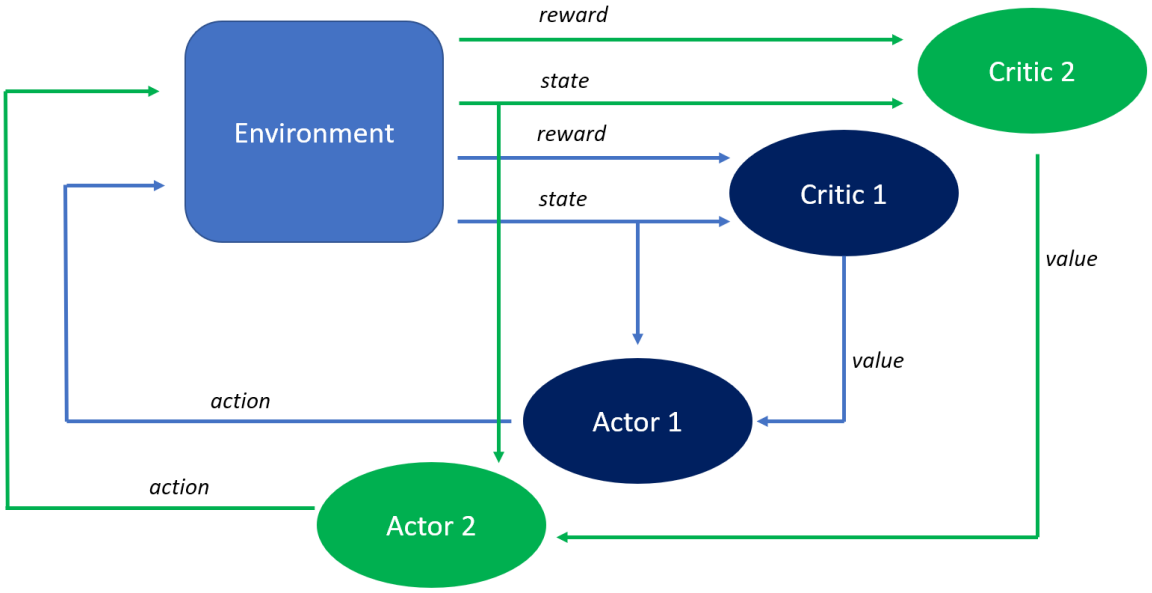

Figure 2. Multi Agent reinforcement learning architecture

The DDPG algorithm can be presented as follows

1. There are two Agents and two Critics who use separate neural network.
2. The Actor network (for each Agent) with: $a = \mu(s; \theta^{\mu})$ which takes input as a state s and results in the action a where $\theta^{\mu}$is the Actor network learning weights. The actor here is used to approximate the optimal policy deterministically. That means that the output is the best believed action for any given state. This is unlike a stochastic policy (probability distribution) in which we want the policy to learn a probability distribution over the actions. In DDPG, we want the believed best action every single time we query the actor network. The actor is basically learning the argmax a Q(S, a), which is the best action.
3. The Critic network (for each Agent) $Q(s; a, \theta^{Q})$ => $Q(s; \mu(s; \theta^{\mu}), \theta^{Q})$ which takes an input as a state s and action a and returns the *Q* value where is the Critic network θ^{Q} weights. The critic learns to evaluate the optimal action value function by using the actors best believed action.
4. We define a target network for both the Actor network $\mu(s; \theta^{\mu`})$ and Critic network $Q(s; a, \theta^{Q`})$ respectively, where $\theta^{\mu`}$and $\theta^{Q`}$ are the weights of the target Actor and Critic network.
5. Next, we perform the update of Actors network weights with policy gradients and the Critics network weight with the gradients calculated from the TD error.
6. In order, to select correct action, first we have to add an exploration noise *N* to the action produced by Actors: $\mu(s; \theta^{\mu}) + N$ (add noise to encourage exploration since policy μ is deterministic).
7. Each Agent selected action in a state, *s*, receive a reward, *r* and move to a new state, *s'*.
8. We store this transition information in experience replay buffer, common for both Agents.

9. As it is performed while we use DQN algorithm, we sample transitions from the replay buffer and train the network, and then we calculate the target *Q* value:
$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$
10. Then, we can compute the TD error as:

$$L = \frac{1}{M}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$$

Where *M* is the number of samples from the replay buffer that are used for training.

11. Subsequently, we perform the update of the Critic networks weights with gradients calculated from this loss *L*.
12. Then, we update our policy network weights using a policy gradient.
13. Next, we update the weights of Actor and Critic network in the target network. In DDPG algorithm topology consist of two copies of network weights for each network, (Actor: regular and target) and (Critic: regular and target). In DDPG, the target networks are updated using a soft updates strategy. A soft update strategy consists of slowly blending regular network weights with target network weights. In means that every time step we make our target network be 99.99 percent of target network weights and only a 0.01 percent of regular network weights (slowly mix of regular network weights into target network weights)
14. We update the weights of the target networks (Agent, Critic) slowly, which promotes greater stability (soft updates strategy):

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

4. PROJECT SETUP. RESULTS.

In Collaboration and competition project following setup of neural networks (for Agent and Critic) were involved.

Depicted below plot of rewards illustrates that the environment is solved (to fulfil the project requirement, the agents must get an average score of +0.5 over 100 consecutive episodes, after taking the maximum over both agents) while playing 1432 episodes.

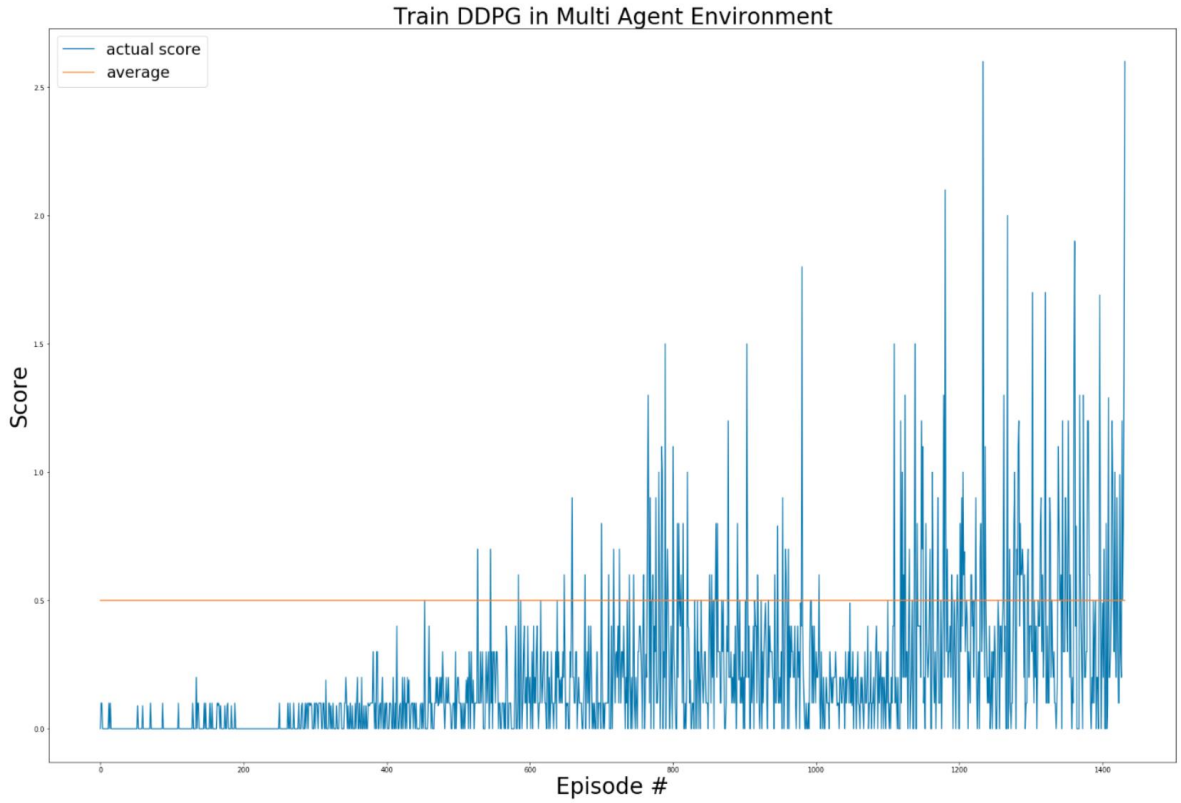

Figure 3.Training process. Results

```
Episode 1280    Total Average Score: 0.41
Episode 1290    Total Average Score: 0.43
Episode 1300    Total Average Score: 0.43
Episode 1310    Total Average Score: 0.42
Episode 1320    Total Average Score: 0.43
Episode 1330    Total Average Score: 0.45
Episode 1340    Total Average Score: 0.41
Episode 1350    Total Average Score: 0.44
Episode 1360    Total Average Score: 0.46
Episode 1370    Total Average Score: 0.47
Episode 1380    Total Average Score: 0.48
Episode 1390    Total Average Score: 0.45
Episode 1400    Total Average Score: 0.46
Episode 1410    Total Average Score: 0.45
Episode 1420    Total Average Score: 0.46
Episode 1430    Total Average Score: 0.46
Problem Solved after 1432 epsisodes. Total Average score: 0.50
```

Applied Deep Neural Network architecture (Agent):

Input layer FC1: 33 nodes in, 200 nodes out
Hidden layer FC2: 200 nodes in, 150 nodes out
Output layer FC3: 150 nodes in, 4 out – action size

Applied Deep Neural Network architecture (Critic):

Input layer FC1: 33 nodes in, 400 nodes out
Hidden layer FC2: 400 nodes in, 300 nodes out
Output layer FC3: 300 nodes in, 1 out – action size

Applied hyperparameters:

BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 512 # minibatch size
GAMMA = 0.99 # discount factor
TAU = 1e-3 # for soft update of target parameters
LR_ACTOR = 1e-3 # learning rate of the actor
LR_CRITIC = 1e-3 # learning rate of the critic
WEIGHT_DECAY = 0 # L2 weight decay
Epsilon start = 1.0
Epsilon start = 0.01
Epsilon decay = 1e-6

5. IDEAS OF FUTURE WORK

The discussed work compounds contemporary advances in deep learning and reinforcement learning. Modern combination yields exceptional results in solving challenging AI issues across a variety of domains. However, in this project a few limitations to presented approach remain. Future work should concentrate mainly on following tasks. Most notably, which was treated as a minor case was tuning of the **hyper parameters**. Future work can include implementation of **L2 regularization** (punish large weights while performing back propagation) and **dropout** (randomly switching off neurons while learning). The other thing which will be reasonable to verify is the choice of applied algorithm. Here, it is suggested to deploy the **MADDPG** algorithm (Multi Agent DDPG), where only one common Critic can be exploited (for both Agents).