Collaboration and Competition Project (Tennis)

as part of UDACITY: Deep Reinforcement Learning Nanodegree Program

Ivan Masmitja

***Abstract*—This report describes the basic ideas behind the project number 3 (aka Tennis) conducted as part of the UDACITY nanodegree called: Deep Reinforcement Learning Nanodegree Program, where a Deep Deterministic Policy Gradients (DDPG) is used to train an actor-critic agent in order to control two rackets to bounce a ball over a net.**

Keywords — Reacher, deep deterministic policy gradients, DDPG, reinforcement learning, deep neural network, artificial intelligence.

I. Introduction

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, 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.

II. Implementation

The implementation is based on the work conducted in [1], where the DDPG algorithm was firstly introduced. They introduced this algorithm as an "Actor-Critic" method. Though, some researchers think DDPG is best classified as a Deep Q-Network method for continuous action spaces, along with Normalized Advantage Functions (NAF) [2]. Regardless, DDPG is a very successful method and it's good for you to gain some intuition.

This model-free approach can learn competitive policies for different tasks using low-dimensional observations (e.g. cartesian coordinates or joint angles) using the same hyper-parameters and network structure. In many cases, it is also able to learn good policies directly from pixels, again keeping hyperparameters and network structure constant. A key feature of the approach is its simplicity: it requires only a straightforward actor-critic architecture and learning algorithm with very few “moving parts”, making it easy to implement and scale to more difficult problems and larger networks.

The DDPG algorithm is presented below (which has obtained from [1]):

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

For this project, we have considered two identical agents, where we have considering the symmetry presented in the environment, which is presented in Fig. 1.

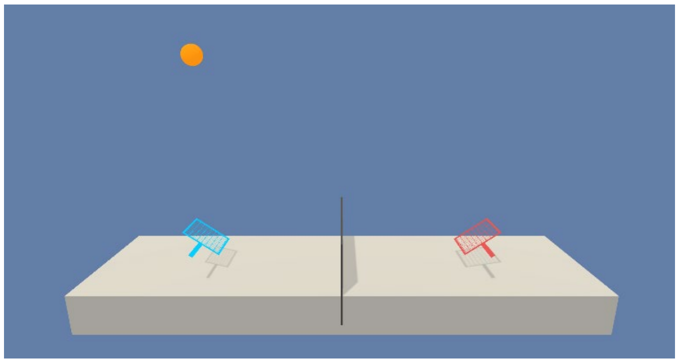

Fig. 1 Unity ML-Agents Tennis Environment.

To solve the environment, the only changes we made were:

a) To use gradient clipping when training the critic network.
b) We decided to get less aggressive with the number of updates per time step. In particular, instead of updating the actor and critic networks 20 times at every timestep, we amended the code to update the networks 10 times after every 20 timesteps.
c) As the action for jump is 1 for jump and <1 for no-jump we clipped the action using the sign function.

This was enough to solve the environment.

A. Hyperparameters

We have tested different configurations, such as number of units of the hidden layers, different learning rates, different discount factor, etc. Nonetheless, the best hyperparameters values to solve the environment were:
:

- BUFFER_SIZE = 1e5 (replay buffer size)
- BATCH_SIZE = 128(256) (minibatch size)
- GAMMA = 0.99 (discount factor)
- TAU = 1e-3(1e-2) (for soft update of target parameters)
- LR_ACTOR = 1e-4(1e-5) (learning rate of the actor)
- LR_CRITIC = 1e-3(1e-4) (learning rate of the critic)
- WEIGHT_DECAY = 0 (L2 weight decay)
- TRAIN_EVERY = 20(30) (How many iterations to wait before updating target networks)
- UPDATE_TIMES = 10(20) (Number of times we update the networks)
- fc1_units=400 (First hidden layer units)
- fc2_units=300 (Second hidden layer units)

B. Model Architecutre

The model architecture used for the DDPG agent was following a traditional Actor-Critic structure [3] and shown in Fig. 2. Two Neural Networks (NN) have been created, one for the Actor and one for the Critic.

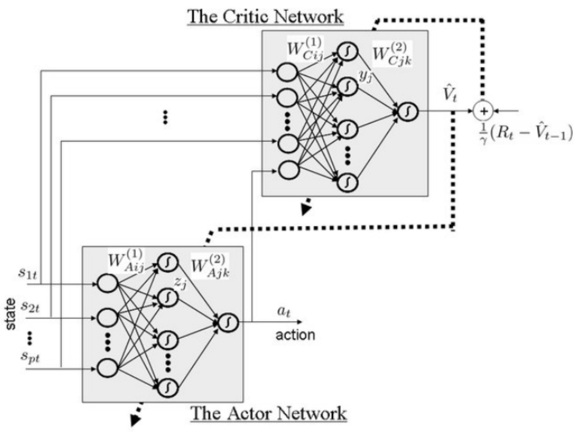

Fig. 2. the actor-critic learning architecture.

The Actor NN has 3 full connected linear layers. The first 2 layers have a ReLU activation function, whereas the last one has a tanh activation function to give an output control between -1 and 1. The first and second layers have 400 and 300 units respectively.

The Critic NN has also 3 full connected linear layers with a ReLU activation function. In this case, the output does not have the tanh activation. In addition, the second layer has been concatenated with the action provided by the Actor, as shown in Fig 2. The first and second layers have also 400 and 300 units respectively.

III. RESULTS

The above model proved to be very good at solving this problem. It achieved an average score above the score threshold of +0.5 over 100 consecutive episodes as can be observed in Fig 3, and summarized below. Here, a batch size equal to 128, a train every 20 iterations, and an update time equal to 10 have used.

Episode 2050	Average Score (averaged over agents): 0.15
Episode 2060	Average Score (averaged over agents): 0.17
Episode 2070	Average Score (averaged over agents): 0.18
Episode 2080	Average Score (averaged over agents): 0.19
Episode 2090	Average Score (averaged over agents): 0.22
Episode 2100	Average Score (averaged over agents): 0.27
Episode 2110	Average Score (averaged over agents): 0.30
Episode 2120	Average Score (averaged over agents): 0.30
Episode 2130	Average Score (averaged over agents): 0.31
Episode 2140	Average Score (averaged over agents): 0.31
Episode 2150	Average Score (averaged over agents): 0.35
Episode 2160	Average Score (averaged over agents): 0.42
Episode 2170	Average Score (averaged over agents): 0.47
Episode 2180	Average Score (averaged over agents): 0.49
Episode 2190	Average Score (averaged over agents): 0.54
Episode 2200	Average Score (averaged over agents): 0.60
Episode 2210	Average Score (averaged over agents): 0.72
Episode 2220	Average Score (averaged over agents): 0.78
Episode 2230	Average Score (averaged over agents): 0.81
Episode 2240	**Average Score (averaged over agents): 0.83**
Episode 2250	Average Score (averaged over agents): 0.79
Episode 2260	Average Score (averaged over agents): 0.74
Episode 2270	Average Score (averaged over agents): 0.69
Episode 2280	Average Score (averaged over agents): 0.68
Episode 2290	Average Score (averaged over agents): 0.63
Episode 2300	Average Score (averaged over agents): 0.51
Episode 2310	Average Score (averaged over agents): 0.37

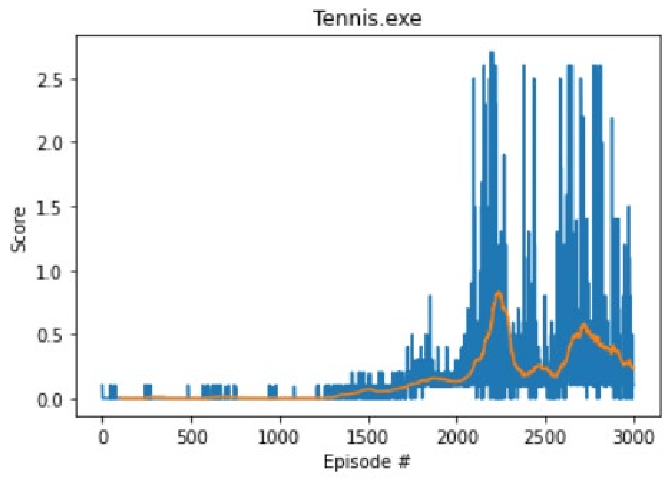

Fig. 3. (Test 1). Scores evolution over episode for collab_compet project. Red line indicates the average values over 100 iterations.

After this first test, we tried to increase the batch size, and the steps used for training and updating, in order to increase the learning rate, and therefore, the final score obtained. Specifically, we used a batch size equal to 256, a training every 30 steps, and an update time equal to 20. This improves the total score, which yielded to a maximum of 1.6376. See next figure.

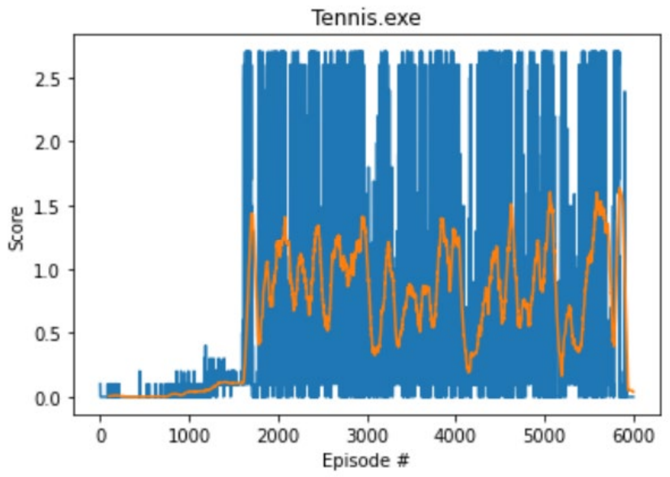

Fig. 4. (Test 2). Scores evolution over episode for collab_compet project. Red line indicates the average values over 100 iterations.

In order to reduce the oscillating observed in the score average values over 100 iterations, we tried to increase tau (for soft update of target parameters) to 1e-2. Whereas this did not reduce the oscillation, increased the learning speed, solving the environment with only ~500 episodes. The maximum average score over 100 iterations was 1.5598. See next figure.

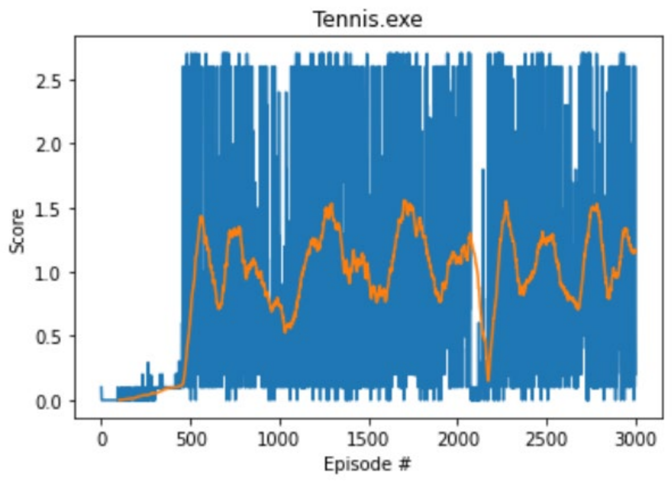

Fig. 5 (Test 3). Scores evolution over episode for collab_compet project. Red line indicates the average values over 100 iterations.

Finally, we tried to reduce the learning rate for both the actor and the critic by an order of magnitude, 1e-5 and 1e-4 respectively. This reduced the oscillation as well as increased the overall performance, reaching a maximum average score over 100 iterations to 2.4749 points. See next figure.

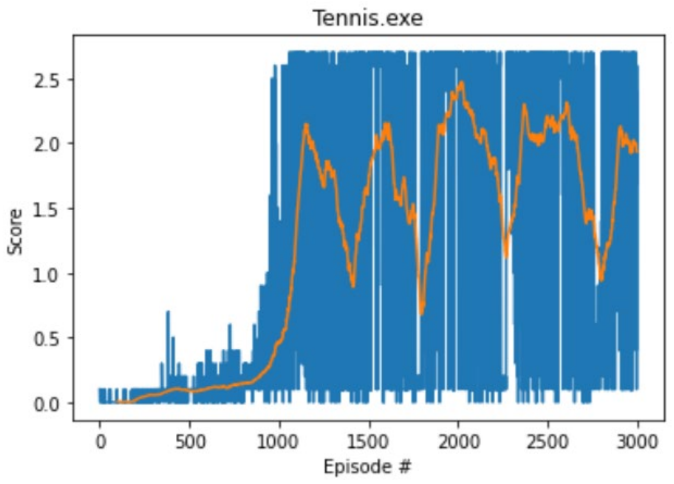

Fig. 6. (Test 4). Scores evolution over episode for collab_compet project. Red line indicates the average values over 100 iterations.

IV. Conclusions

This project can be solved using different algorithms, for example using a multi agent approach. However, due to the symmetries of the environment and the collaboration approach used to solve it. A simple DDPG method where both agents use the same Actor and Critic networks can be used. This method has shown great, which accomplished an average score (over 100 episodes) of 2.315.

V. Future work

As a future work, I will implement other algorithms that have shown great performance in continuous control scenarios. For example:

a) Proximal Policy Optimization (PPO) [4]

b) Asynchronous Advantage Actor-Critic (A3C) [5]

c) Distributed Distributional Deterministic Policy Gradients (D4PG) [6]

d) Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG) [7].

Finally, this algorithms will be implemented to obtain the optimal trajectory for an underwater autonomous vehicle in order to localize and track an underwater target [8]–[10].

References

[1] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, 2016.

[2] S. Gu, T. Lillicrap, U. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 6, pp. 4135–4148, 2016.

[3] H. Chen, G. Jiang, H. Zhang, and K. Yoshihira, "Boosting the performance of computing systems through adaptive configuration tuning," *Proc. ACM Symp. Appl. Comput.*, no. January 2014, pp. 1045–1049, 2009, doi: 10.1145/1529282.1529511.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv*, pp. 1–12, 2017.

[5] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[6] P. O. G. Radients *et al.*, "D Istributed D Istributional D Eterministic," *Int. Conf. Learn. Represent.*, pp. 1–16, 2018.

[7] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 3, pp. 2001–2014, 2016.

[8] I. Masmitja *et al.*, "Optimal path shape for range-only underwater target localization using a Wave Glider," *Int. J. Rob. Res.*, vol. 37, no. 12, pp. 1447–1462, 2018, doi: 10.1177/0278364918802351.

[9] I. Masmitja *et al.*, "Range-Only Single-Beacon Tracking of Underwater Targets from an Autonomous Vehicle: From Theory to Practice," *IEEE Access*, vol. 7, pp. 86946–86963, 2019, doi: 10.1109/ACCESS.2019.2924722.

[10] I. Masmitja *et al.*, "Mobile robotic platforms for the acoustic tracking of deep-sea demersal fishery resources," *Sci. Robot.*, vol. 5, no. eabc3701, 2020.