Competition and Collaboration Project - Tennis Players

1 Framework

1.1 Environment objective

There are 2 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 8 variables corresponding to the position and velocity of the ball and racket. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

So under this framework the agents will try to keep the ball on the court as long as possible to maximize the possible rewards,id est, strictly speaking this task is a competitive task because the agents have to learn how to play along the other in harmonious manner so both can benefit, otherwise in a competitive environment they would have to learn how to beat the other agent or learn how to make the other fail. To tackle this task a multi-agent version of the Deep Deterministic Policy Gradient (DDPG) is used.

1.2 DDPG

In the previous project I added a briefly explanation and the pseudo code for DDPG [2], refer to section 1.2 in case you need a refresh.

2 DDPG implementation

The agents are trained using a common replay buffer that feeds a single pair of Actor-Critic networks, so a single policy is used by both agents. This can lead that the agents could have very similar styles of gaming. To avoid this situation a separated replay buffer could be used for each agent and use different networks to train them, to keep it simple in this exercise I went for the first approach.

At each step the environment returns the rewards and next states for each agent these are stored along with previous states and the taken actions into the replay buffer, the agent keep taking actions and repeating this process until after a certain number of steps, then a sample batch is taken from the buffer and the learning process starts. After compute the updates for each network(actor and critic) the weights are propagated to the agents.

2.1 Hyper-parameters

Hyper-parameters		
Parameter	Value	Description
BUFFER_SIZE	100,000	replay buffer size
BATCH_SIZE	128	minibatch size
GAMMA	0.99	discount factor used in TD targets
TAU	.001	for soft update of target parameters
LR_ACTOR	.003	learning rate used by the optimizer algorithm of the actor neural network
LR_CRITIC	.003	learning rate used by the optimizer algorithm of the critic neural network
UPDATE_EVERY	1	how step are needed to update the networks
NUMBER_UPDATES	1	number of time the networks are updated in each leaning phase
HIDDEN_LAYER_1	48	number of nodes for the first fully connected layer in critic and actor's network
HIDDEN_LAYER_2	24	number of nodes for the second fully connected layer in critic and actor's network
HIDDEN_LAYER_3	12	number of nodes for the second fully connected layer in critic and actor's network

2.1.1 Network's architecture

Actor Network		
Layers	Nodes	Activation function
Input layer	24	Leaky Rectifier Linear Unit
First hidden layer	48	Leaky Rectifier Linear Unit
Second hidden layer	24	Leaky Rectifier Linear Unit
Third hidden layer	12	Hyperbolic Tangent
Output layer	2	NA

To the output of the actor network the noise coming from an Ornstein–Uhlenbeck process is added and then clipped between -1 and +1. Another

Critic Network		
Layers	Nodes	Activation function
Input layer	24	Leaky Rectifier Linear Unit
First hidden layer	50	Leaky Rectifier Linear Unit
Second hidden layer	24	Leaky Rectifier Linear Unit
Third hidden layer	12	Identity
Output layer	1	NA

A thing to notice is that after feeding the input layer of the critic-network with states, the output of that layer and the actions coming from the output of the actor-network are concatenated to finally feed the second layer of the critic-network. Also although the observation space consists of 8 variables, the observation returned by the environment are 24 features vectors for each agent, this because is stacking the 3 observations frames that will comprise an experience-

2.1.2 Noise

To each action suggested by the actor-network the algorithm noise which follows the next equation:

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

This stochastic process is known as Ornstein–Uhlenbeck and the W_t associated process(the Wiener process) can be seen as the limit of a random walk, so the sample taken from it should be modeled as $Normal(0, 1)$ to avoid add a positive bias, which it happens when sample from a uniform distribution given that actions must lay in $[-1, 1]$.

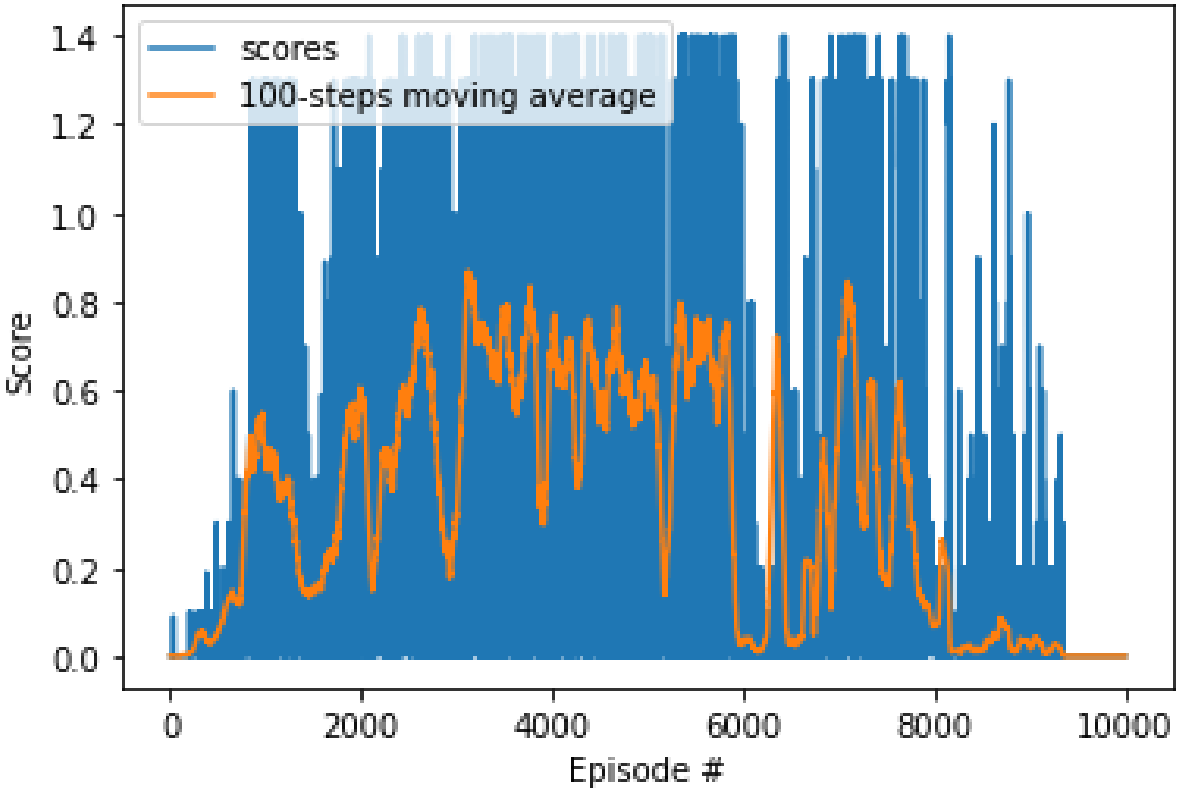
3 Results

To train the agents 10000 episodes were run (with a maximum of 500 steps per episode) and the weights for the first time the agent gets at least a .5 average over 100 episode are saved, and also the weights for the best performance achieved. Following it can be observed how the agents perform through the training phase, their average performance over a window of 100 hundreds episode every 100 episodes.

DDPG Agents	
Episode	Average Score
100	.0009
200	.0019
300	.0057
400	.0511
500	.0294
600	.0548
700	.1267
800	.1188
900	.4054
1000	0.4802
1100	0.4420
1200	0.4293
1300	0.3814
1400	0.2867
1500	0.1432
1600	0.1505
1700	0.1860
1800	0.2124
1900	0.4170
2000	0.5566
2100	0.5836
2200	0.1830
2300	0.3682
2400	0.3943
2500	0.5562
2600	0.5958
2700	0.7128
2800	0.6809
2900	0.4072
3000	0.2594
3100	0.3346
3200	0.8219
3300	0.7288
3400	0.7366
3500	0.6343
3600	0.7665
3700	0.5583
3800	0.7063
3900	0.7788
4000	0.2986
4100	0.7128
4200	0.6233
4300	0.6925
4400	0.4286
4500	0.7025
4600	0.5549
4700	0.6578
4800	0.6559
4900	0.6156
5000	0.5677

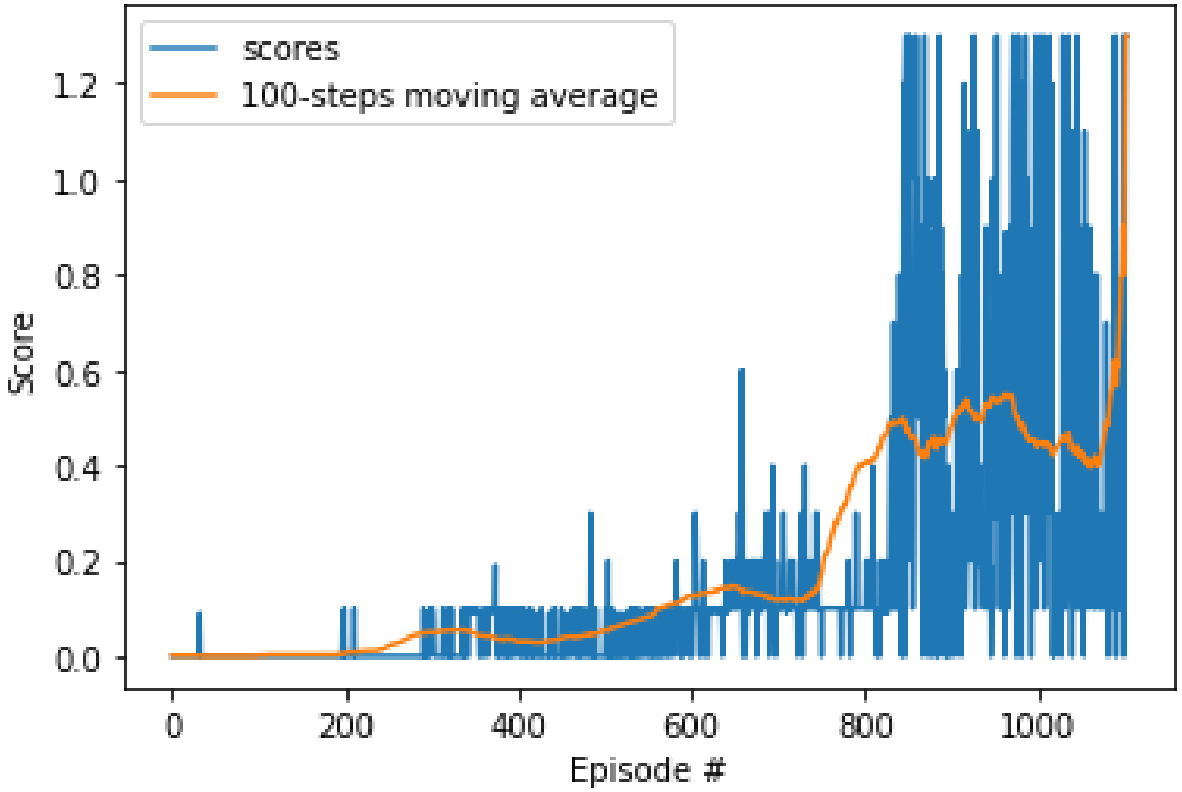
DDPG Agents	
Episode	Average Score
5100	0.5649
5200	0.5378
5300	0.2333
5400	0.6658
5500	0.6267
5600	0.7198
5700	0.6916
5800	0.6407
5900	0.6928
6000	0.2270
6100	0.0390
6200	0.0190
6300	0.0170
6400	0.3818
6500	0.4109
6600	0.0270
6700	0.0470
6800	0.2020
6900	0.3299
7000	0.2160
7100	0.5750
7200	0.7910
7300	0.4930
7400	0.5083
7500	0.4135
7600	0.1663
7700	0.5383
7800	0.4196
7900	0.2281
8000	0.1082
8100	0.0680
8200	0.2429
8300	0.0110
8400	0.0200
8500	0.0100
8600	0.0180
8700	0.0400
8800	0.0800
8900	0.0150
9000	0.0400
9100	0.0160
9200	0.0330
9300	0.0110
9400	0.0220
9500	0.0000
9600	0.0000
9700	0.0000
9800	0.0000
9900	0.0000
10000	0.0000

Environment solved in 903 episodes! Average Score: 0.5003
The 10000 episodes were executed in 171.6 minutes.



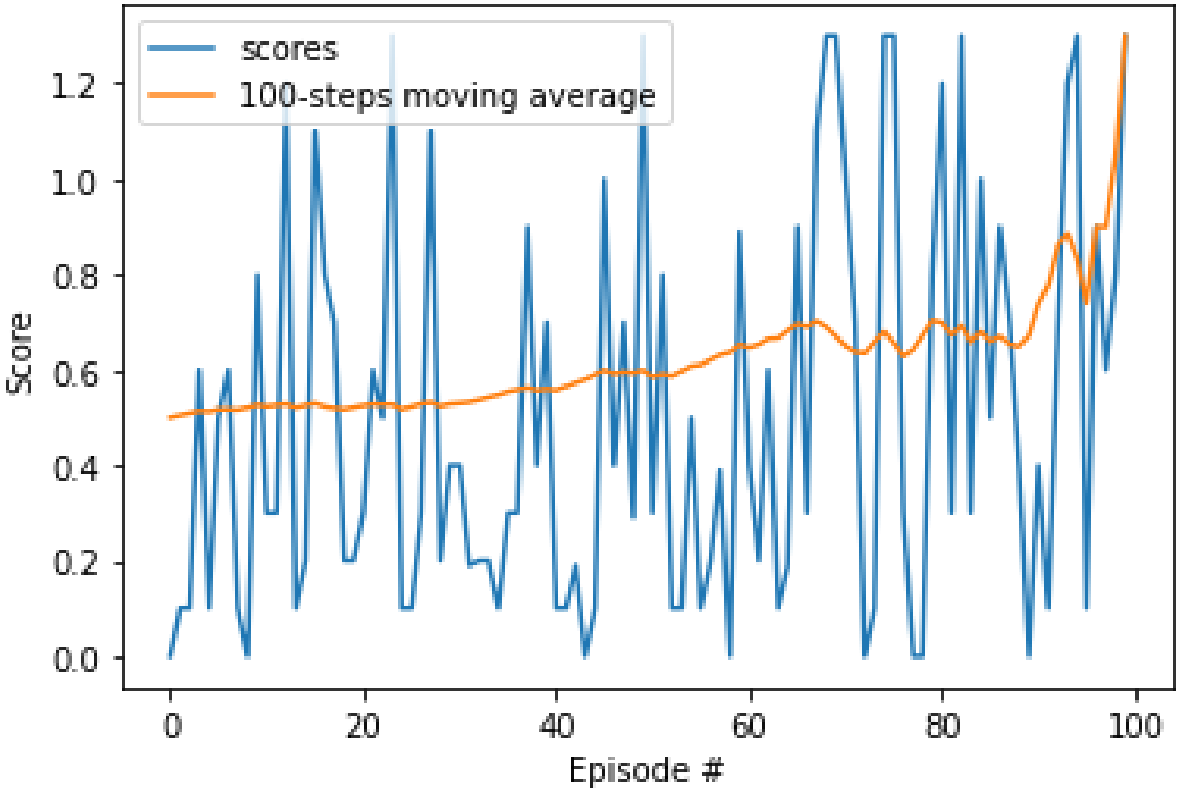
It can be observed that there are long streak with high average score over the first 6000 episodes, even that the training falls once in a while it recovers quickly, this can be observed most of the time except in the last episodes where it seems it does not recover.

3.1 Results until the environment is solved



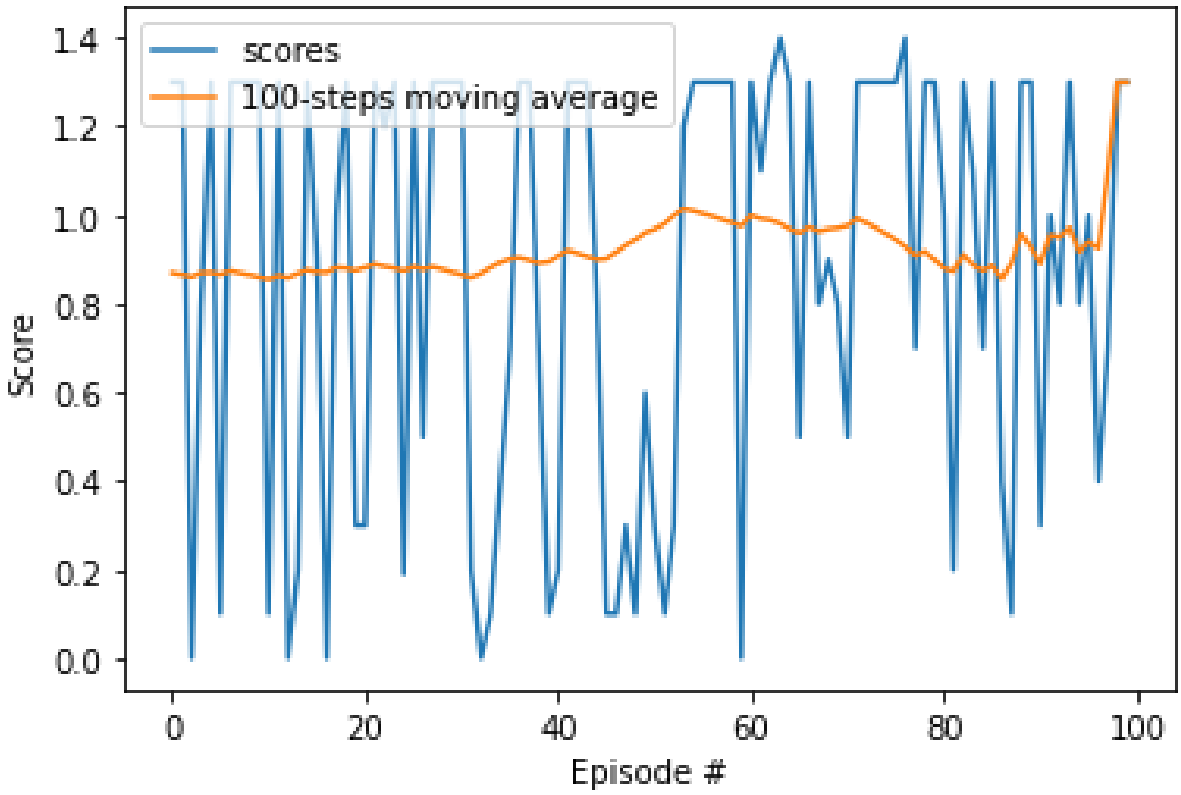
The agents need more data the first 700 episodes and then training goes upwards pretty steady. The

next is a zoom of the 100 episodes that solved the environment, beginning at episode 903.



3.2 Results for the best solution achieved

The results during the training stage are:
The best episode scored: 1.4000000208616257
The best 100 episode average scores: 0.8699000129848719
The best average score was reached at episode 3117
Graph of the episodes that got the best average score:



4 Future Work

The following steps would be:

Tackle the challenge for the soccer environment which needs a collaborative-competitive approach not just collaborative, given that in soccer game the teams need to score as much as possible avoiding the other team scores.

References

[1] Timothy P. Lillicrap, Jonathan J. Hunt, et al(2016). CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING. Google Deepmind. London, UK.

[2] Myself (2020). Continuous Control Reachers Report. https://github.com/chuquikun/Continuous$_{C}ontrol_{P}roject-Reacher/blob/main/report.pdf$