Training a two player tennis environment for cooperative playing with deep reinforcement learning

Eduardo Di Santi

August 2019

Abstract

This document shows how to train a deep reinforcement learning agent with multi agent DDPG to solve the Tennis environment.

1 Introduction

The aim of the project is to train a two player tennis agent.

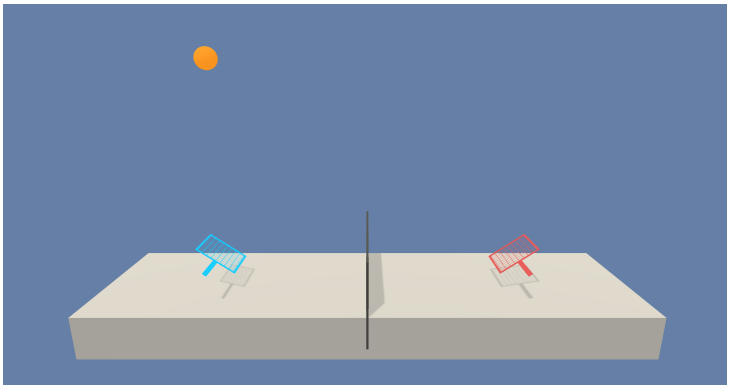

Figure 1: The environment

This environment is a version of Unity ML-Agent Tennis environment, suitable to test reinforcement learning algorithms. Solving this continuous spaces environment will need the use Agent - Critic [2] method to approximate the best policy, in this case the use of Deep Deterministic Policy Gradient (DDPG) algorithm.

2 Environment

The environment is an inertial plus gravity simulation of a ball and two players with represented by rackets. The goal is to keep the ball in the air as much time as possible following the regular tennis rules. In this case, there are two agents and they learn by self playing. The environment grants a positive reward of +0.1 when the agent is able to hit the ball over the net and -0.1 when the ball hits the player ground or go out of bound following the tennis rules
The environment is considered solved when the average 100 rewards of of the average rewards collected by the twenty agents is bigger than 0.5.
The action space is a 2 dimension vector corresponding to jump and going towards or against the net.
Task is episodic and rewards are given during the episode.

3 Algorithm

The agent learns using an Actor - Critic [1] approach and a DDPG algorithm for estimate the value action function .
This algorithm has two neural networks, one for the Actor and One for the Critic. The Actor tries to approximate the best policy in a deterministic way . Because the Agent is deterministic, noise has to be added to encourage the exploration.
The policy network weights are updated using a policy gradient.
The Critic gets the State and Action and returns the Q value, which is the action - value function (using DQN) approximated on the NN weights.

3.1 Neural networks architecture

The Agent neural network used has the following architecture

Layer	Neurons	Type	Activation	Comment
Input	8			according to the space state dimension
Hidden	256	Linear	ReLU	
Hidden	256	Linear	ReLU	
Output	1	Linear	tanh	To get values on $[-1, 1] \in R$

The Critic neural network used has the following architecture

Layer	Neurons	Type	Activation	Comment
Input	8			according to the space state dimension
Hidden	512 + action vector size	Linear	ReLU	
Hidden	256	Linear	ReLU	
Output	4	Linear	ReLU	One for each action

3.2 Hyperparameters

The hyper parameters are the following:

Parameter	Value	Description
γ	0.99	Discount factor
τ	0.001	Soft-update ratio
Local network update interval every	4 time steps	
Remote network update interval every	4 time steps	
Replay buffer size	1e6	
Mini batch size	128	
ϵ - start	1	Start value
ϵ - min	0.01	Minimum value
ϵ - decay	1e-6	Decay rate
Learning rate for ACTOR	1e-3	
Learning rate for CRITIC:	1e-4	
L2 weight decay	0	
Learn every	20 time-steps	
Number of learning passes	10	
Ornstein-Uhlenbeck noise σ parameter	0.2	
Ornstein-Uhlenbeck noise θ parameter	0.15	

4 Results

The agent learns and achieve the goal on 100 episodes, but converges around episode 40, the environment is consider solved only averaging one hundred scores, therefore, the agent must wait.

4.1 Training

The goal is to achieve an average score of 0.5 over last 100 episodes, with this hyper-parameters the agent solves the environment in 100 episodes with 1000 time-steps each.
The algorithm waits to consider the environment solved, but will save the best model after every episode.

Every episode consist in 1000 time steps maximum, after episode 80 the agent losses some precision, reducing the time steps does not help on this behaviour (but in other step) and can worth to research why this happens. The final result is not affected because the best model is saved along with the final model.
The training history depiction is shown in Figure 2.

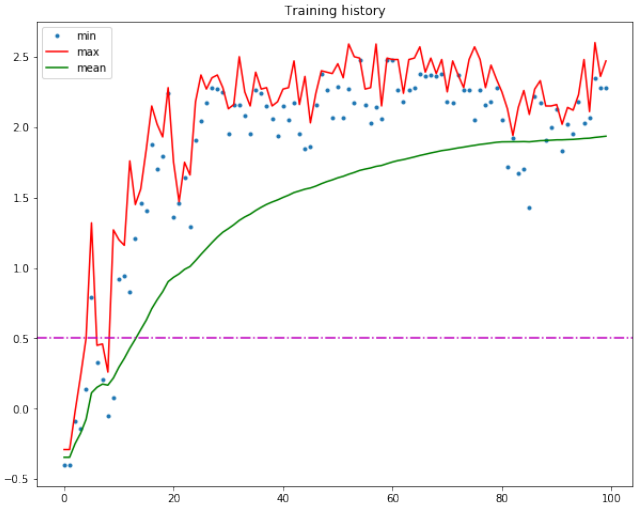

Figure 2: Training, episodes vs rewards

4.2 Playing

The agent plays very well against itself and against a human user proving that the policy is reliable.

5 Conclusions

The agent learns very fast but play collaboratively only.
The agent perform very well once trained, which is coherent with the learning graph.

6 Future work

This task is cooperative, so, the agent doesn't learn to compete, means, play to win. The agent will perform well in preventing the ball to fall or go out of bounds, but wont play to win.
To solve this the solution could be:

- Granting a rewards when the other player fails, if the environment cannot be modified, is enough to feed the last reward to the neural network.
- Feeding the state of the the other player to the neural network

Check the possibility to consider the environment solved before, maybe some hyper parameter tuning can allow the agent to converge faster.

References

[1] Jonathan J. Hunt Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv* (2015). DOI: https://arxiv.org/pdf/1509.02971.

[2] Vijay R. Konda John N. Tsitsiklis. "Actor-Critic Algorithms". In: *MIT* (). DOI: http://web.mit.edu/jnt/www/Papers/J094-03-kon-actors.pdf.