Training a two joint robotic arm with deep reinforcement learning

Eduardo Di Santi

July 2019

Abstract

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

1 Introduction

The aim of the project is to train a group of robotic arms to maintain a ball in the air as much time as possible.

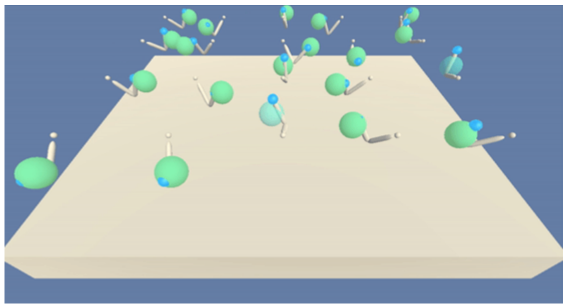

Figure 1: The environment

This environment is a version of Unity ML-Agent Reacher environment, suitable to test reinforcement learning algorithms. Solving this continuous spaces environment will need the use Agent - Critic [2] method to approximate the best police, 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 a two joint arm trying to keep the ball in the air as much time as possible. In this case,

there are multiple agent, which allows sharing experience learning to speed up learning. The environment grants a positive reward of +0.1 when the agent is able to maintain the ball in the correct position
The environment is considered solved when the average 100 rewards of of the average rewards collected by the twenty agents is bigger than 30.
The action space is a four dimension vector corresponding to the torque applied to the two joints.

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	33			according to the space state dimension
Hidden	256	Linear	ReLU	
Hidden	256	Linear	ReLU	
Output	4	Linear	ReLU	One for each action

The Critic neural network used has the following architecture

Layer	Neurons	Type	Activation	Comment
Input	33			according to the space state dimension
Hidden	512	Linear	ReLU	
Hidden	256	Linear	ReLU	
Output	1	Linear	tanh	To get values on $[-1, 1] \in R$

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 60, the environment is consider solved only averaging with one hundred, therefore, the agent must wait.

4.1 Training

The goal is to achieve an average score of 30 over last 100 episodes, with this hyper-parameters the agent solves the environment in 100 episodes with 1000 time-steps each.

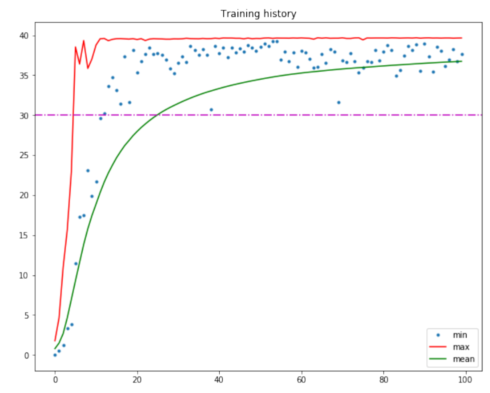

Figure 2: Training, episodes vs rewards

5 Conclusions

The agent learns fast but can improve, one of the problems is the high amount of computations needed to converge when training 20 agents.
The agent perform very well once trained, which is coherent with the learning graph. Training in a cluster with, for example, the use of Horovod can improve the 5 hours time because less agents will training on each node.

6 Future work

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`.