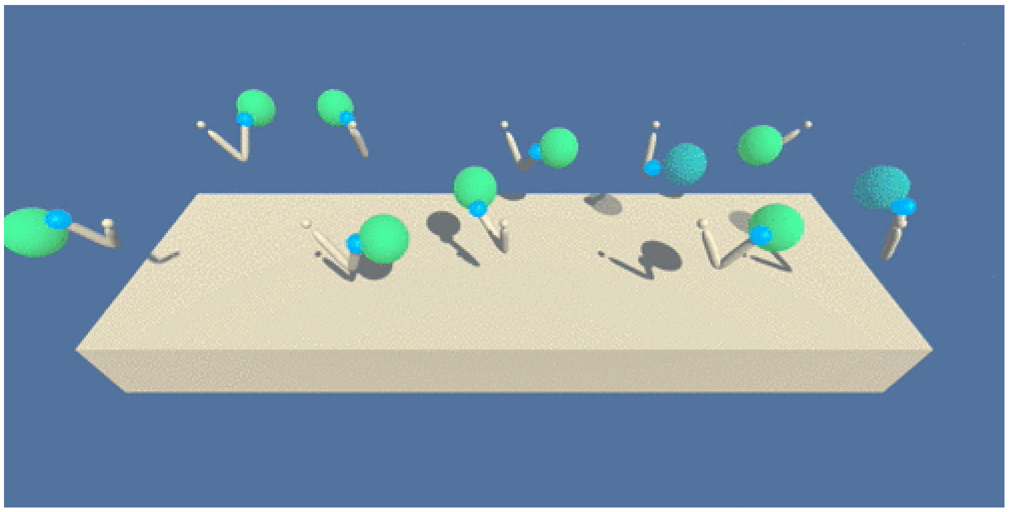
Deep Reinforcement Learning Nanodegree Project 2 - Continuous Control

Jean Salvi DUKUZWENIMANA

June 24 ,20201

Abstract

This project was solved using a deep reinforcement learning agent.it uses Actor-Critic methods to train an agent to move a double-jointed arm to a target location.



In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

Thus, the agent goal is to maintain its position at the target location for as many time steps as possible. The objective of the project is to train 20 identical agents at the same time, each with its own copy of the environment, to get an average score of +30 over 100 consecutive episodes, and over all the agents. To this end, in this work we implement two different agents based on actor-critic DRL methods: Deep Deterministic Policy Gradients (DDPG) and Proximal Policy Optimisation (PPO). After training, the performance of the agents was compared in order to evaluate which agent was able to achieve the objective in the least number of episodes, and which was able to achieve the objective.

Learning algorithm

Deep Deterministic Policy Gradient (DDPG) , which is an actor-critic approach, was used as the learning algorithm for the agent. This algorithm is quite similar to DQN, but also manages to solve tasks with continuous action spaces. As an off-policy algorithm DDPG utilizes four neural networks: a local actor, a target actor, a local critic and a target critic Each training step the experience (state, action, reward, next state) the 20 agents gained was stored.during backpropagation if the same model was used to compute the target value and the prediction, it would lead to computational difficulty. During the training, the actor is updated by applying the chain rule to the expected return from the start distribution. The critic is updated as in Q-learning, i.e. it compares the expected return of the current state to the sum of the reward of the chosen action + the expected return of the next state.

Few things had to be adapted : the step function as we know have simultaneously 20 agents that return experiences and the noise as we have to apply a different noise to every agent (at first I did not change the noise, the training was running but the agent did not learn anything).

Hyperparameters

The following hyperparameters were used:

- BUFFER_SIZE = int(1e5) # replay buffer size
- BATCH_SIZE = 128   # 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

Neural networks

The actor model is a simple feedforward network:

- First layer : input size = 33 and output size = 128
- Second layer : input size = 128 and output size = 128
- Third layer : input size = 128 and output size = 4

The critic model:

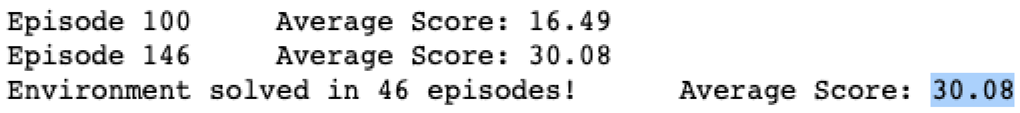
- First layer : input size = 33 and output size = 128

- Second layer : input size = 134 and output size = 128
- Third layer : input size = 128 and output size = 1

The second layer takes as input the output of the first layer concatenated with the chosen actions.

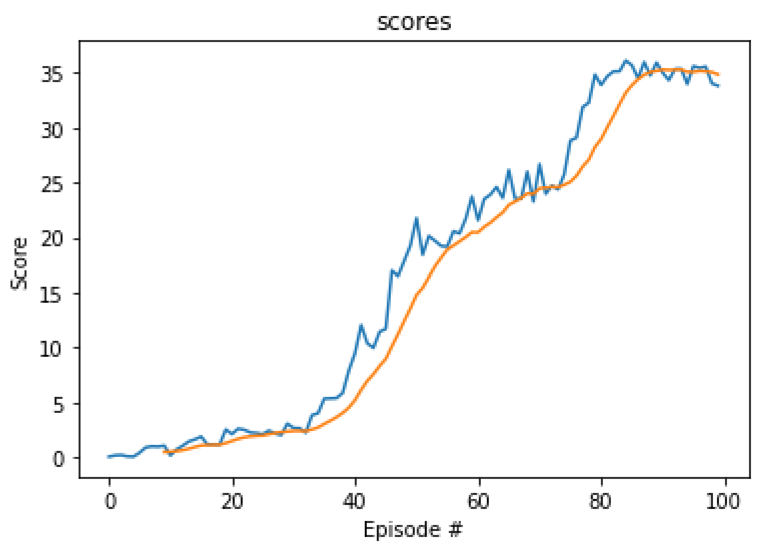
Results

The agent was able to solve the environment after 46 episodes achieving an average score of 30.08.



```
Episode 100	Average Score: 16.49
Episode 146	Average Score: 30.08
Environment solved in 46 episodes!	Average Score: 30.08
```

Here is the graph of the score evolution



possible future improvements

The algorithm could be improved in many ways. batch normalisation could be added,To improve the stability of the model.also one could implement some DQN improvements, for example Prioritized Experience Replays which would improve the learning effect gained from the saved experience.