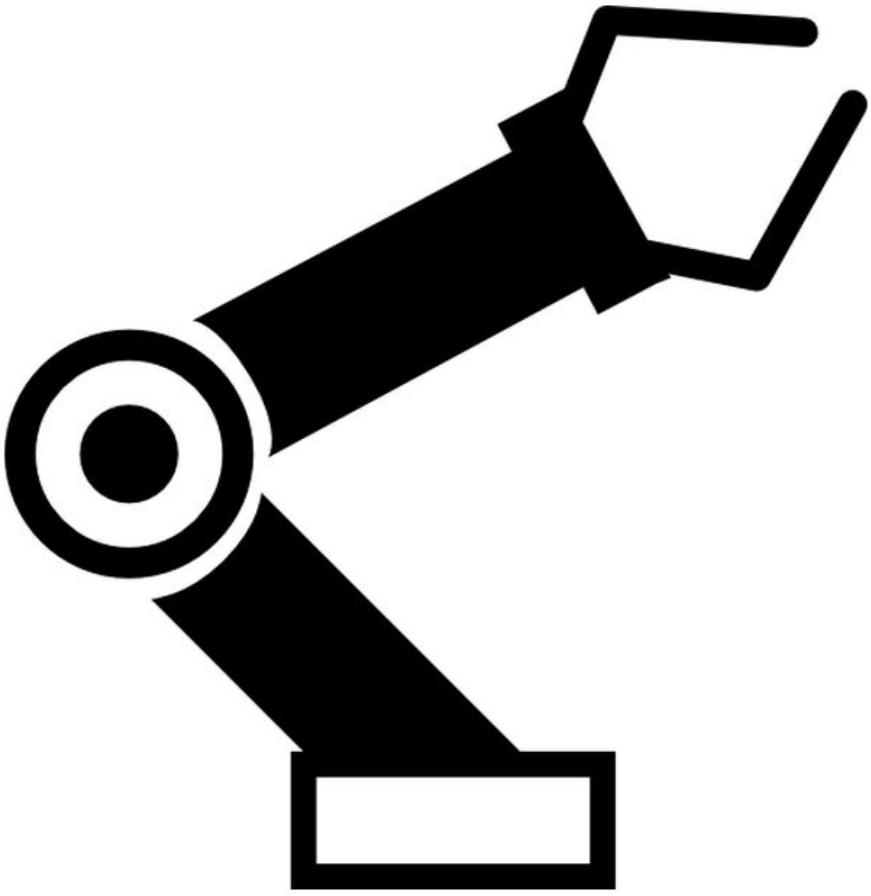
CONTINUOUS CONTROL PROJECT REPORT

Train an agent to control a robotic arm, and steer it to a target location



Dimitris Poulopoulos

08.25.2018

Udacity Deep Reinforcement Learning Nanodegree

INTRODUCTION

Unity Machine Learning Agents (ML-Agents) is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents.

For game developers, these trained agents can be used for multiple purposes, including controlling NPC behaviour (in a variety of settings such as multi-agent and adversarial), automated testing of game builds and evaluating different game design decisions pre-release.

In n this project, we develop a Deep Deterministic Policy Gradient (DDPG) agent that utilises its newly acquired skills to control a robotic arm, and steer it to a target location. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of the agent is to maintain its position at the target location for as many time steps as possible.

The state 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. To solve the environment, the agent must get an average score of +30 over 100 consecutive episodes.

LEARNING ALGORITHM

This project creates a DDPG agent, which implements the corresponding algorithm, pioneered by Lillicrap, Timothy P., et al.[1]. This network takes as input a state space 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. Then, it passes this information through a relatively simple network architecture, consisting of a few fully connected, hidden layers, with ReLU activation functions, called the Actor. The output and final layer of the network is a *tanh* activation layer, that emits the best believed action for a specific, given state. It consists of a four dimensional action vector, where every number is between -1 and 1.

The output of the actor, is fed into a second network, called the Critic, which learns to

[1] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971(2015).

evaluate the output of the Actor. That is to evaluate the optimal action value function, by using the Actor's best believed action.

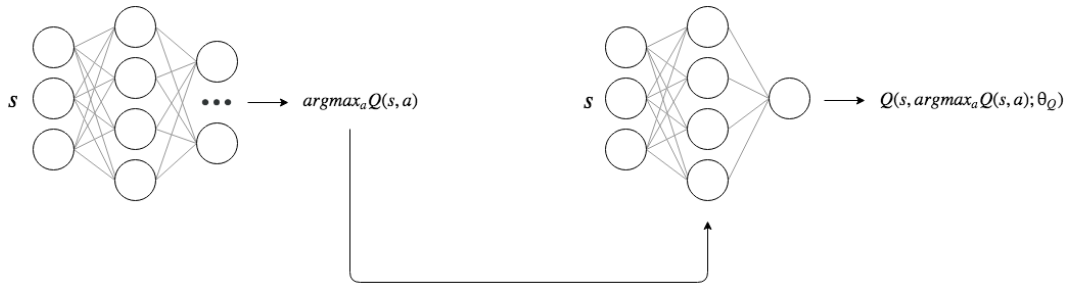


Figure 1 - DDPG Network Architecture

EXPERIENCE REPLAY

Like in the Q-Learning algorithm, to avoid getting swayed by the correlation between a sequence of tuples, we use a technique called *Experience Replay*. Thus, instead of training the agent while it observes the environment and takes action, we are keeping track of a *replay buffer* and later, during training, we sample from it at random, breaking those correlations and preventing action values from oscillating or diverging catastrophically.

The *replay buffer* is a collection of *experience* tuples - S , A , R , S' . The act of sampling a small batch of tuples from the *replay buffer* in order to learn is known as *Experience Replay*. Moreover, this technique allows us to reuse individual tuples to learn more or recall rare occurrences.

SOFT UPDATES

In DDPG the target networks are updated using a *soft update* procedure. A soft update strategy consists of slowly blending the values of regular network weights with the target network parameters, like the formula below:

$$w^{-} = \tau\, w \; + \; (1 - t)\, w^{-}$$

where τ is another hyperparameter, that determines how much of your local networks weights will be mixed with the target network weights.

RESULTS

The results of the experiment are summarised in table 1 and depicted in figure 2.

EPISODE	AVERAGE SCORE
100	1.23
200	9.63
300	28.65
310	30.20

Table 1 - Results

To consider the environment solved, the agent must get an average score of +30 over 100 consecutive episodes. For the specific network architecture this happens in 210 episodes according to the results. To achieve this, we built the network with the following hyperparameters:

- Learning rate for both Actor and Critic: 0.0001
- Batch size: 64
- Discount rate (γ): 0.99
- Soft update (τ): 0.001
- Actor $ Critic Networks:
    - Hidden layers: 2
        - First/Second hidden layer units: 128
- Buffer size: $1 \cdot 10^5$

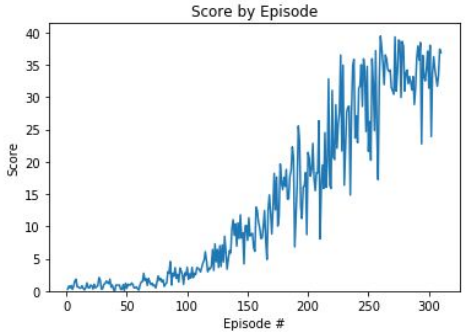


Figure 2 - Results

FUTURE IMPROVEMENTS

For future work, we could leverage another continuous space method, such as NAF [2], which provides a variant of the intuitive Q learning algorithm, for continuous action spaces. To take advantage of a parallel implementation we could try the A3C, and/or A2C algorithms[3]. We could finally examine a GAE implementation[4], to leverage the n-step bootstrapping notion, that sometimes yields superior results.

[2] Gu, Shixiang, et al. "Continuous deep q-learning with model-based acceleration." International Conference on Machine Learning. 2016.

[3] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International conference on machine learning. 2016.

[4] Schulman, John, et al. "High-dimensional continuous control using generalized advantage estimation." arXiv preprint arXiv:1506.02438 (2015).