Udacity DRLND Project 2: Continuous Control

Introduction

For this project, I have worked with a Unity Environment called Reacher. 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.

To solve it, a DDPG agent is trained with the experience of 20 players acting in parallel games. The agents must get an average score of +30 (over 100 consecutive episodes, and over all agents).

Algorithm

The algorithm selected is Deep Deterministic Policy Gradient (DDPG), an actor-critic algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

A common replay buffer is used to store the steps of every player. From that replay buffer are periodically retrieved the experiences to train the agent. Some noise is added to the selected action in order to foment exploration. Both actor and critic present different networks. For both, separate target and local weights are maintained in order to avoid instability. To update the target networks, soft updates are accomplished periodically. Adam optimizer is employed for actor and critic.

References for most of the architectures and code used and first parameters tested was taken from previous or later lessons in the nanodegree program.

Model

Architecture

Actor:

1. Normalization. Fully connected layer – input: state size. Output: 128. Leaky ReLu
2. Fully connected layer – input: 128. Output: 128. Leaky ReLu
3. Fully connected layer – input: 128. Output: action size. Hyperbolic tangent

Critic:

1. Normalization. Fully connected layer – input: state size. Output: 128. Leaky ReLu
2. Fully connected layer – input: 128. Output: 128 + action size (concat). Leaky ReLu
3. Fully connected layer – input: 128 + action size. Output: 1.

Parameters

replay buffer size: int(1e6)

minibatch size: 256

discount factor (gamma) = 0.99

soft update of target parameters from local to target networks (tau) = 1e-3

learning rate of the actor = 1e-4

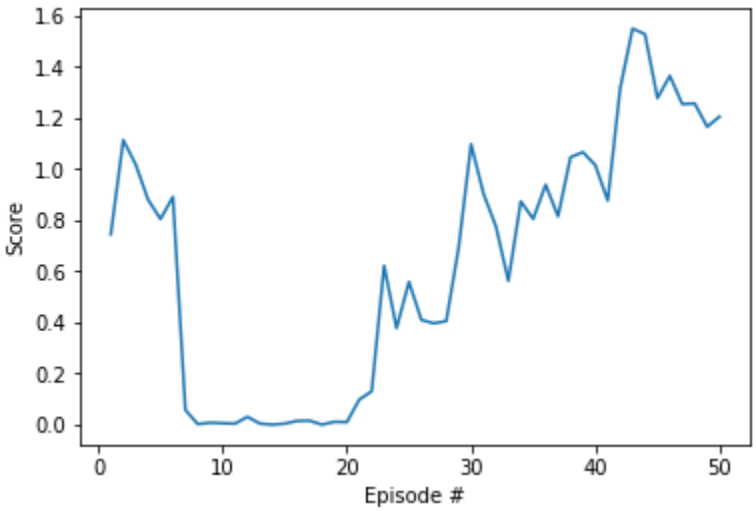
learning rate of the critic = 1e-3

Number of steps for learning (LEARN_EVERY) = 2

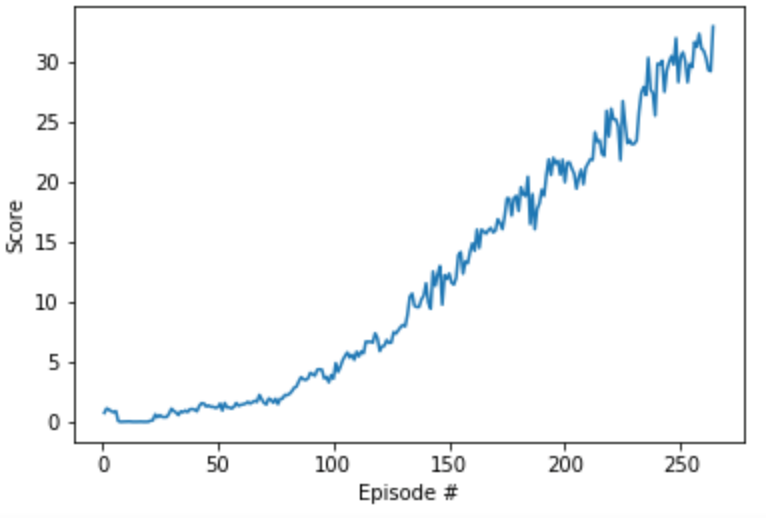
how often to execute the learn-function each LEARN_EVERY step = 1

Results

As results demonstrate, the environment was solved at episode 264. The graph presents a period of instability in the first 20 episodes in which it seems to not learn.



After that, there is an inflection point and the learning is constant until the environment is solved.



Future work

Results show that more training time leads to a significant improvement in the performance, so waiting up to a plateau would be interesting to improve current algorithm's score. Another promising way is the use of prioritized replay (PER) to select the experieces that provide with the most rich informatiom for learning. This can be achieved by selecting the experieces with higher temporal difference error. Besides, the architecture of actor and critic would be increased with more neurons per layer and maybe on depth level. Furthermore, more players in parallel gather more experiences, which increases the possibilites to visit more interesting states.

The Crawler environment will be tested to afford a harder challenge. For sure, the time required to train the agent will increase as well the initial period of instability and non-learning.