Project 2 - Deep reinforcement learning - Continuous Control

Gianluca Scarpellini (gianluca@scarpellini.dev)

April 8, 2020

Contents

1 Introduction

Deep Reinforcement Learning is today becoming a relevant field of research. The current work aims to evaluate the ability of value-based methods of solving a simple task in a constrained environment. In particular, we implemented Deep DPG algorithm as presented during Udacity "Deep Reinforcement Learning" as well as in the original paper. We describe the environment in section 2. A more in depth explanation of the approach is presented in section 3. In section 4 we comment the results obtained with the algorithm. Finally, we take our conclusions in section 5.

2 Environment and task

The presented project is developed in order to solve the "Reacher" environment with multiple agents (20 agents were employed for this specific project). The tasks consists in moving a double-jointed arm to target locations. The reward is $+0.1$ for each correct movement. The agent goals is to maintain the right position as along as possible. The observation space consists of 33 values (position, rotation, velocity, angular velocities for each arm). The space action is **continuous**; each action is contained between -1 and $+1$.

3 Algorithm and Implementation

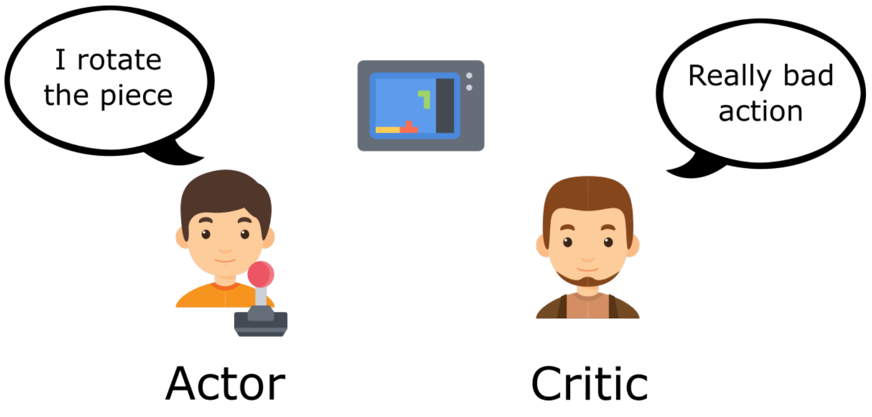

Figure 1: Actor-critic approach

We used an actor-critic approach to solve the environment. Actor-critic methods merges the idea of **policy-based** and **temporal-differences** in order to mitigate their issues both in variance and bias. DDPG exploits 2 different function approximators (such as neural networks, CNN, ...). The policy produces by the actor is **deterministic**: the action per state is the optimal one. The critic evaluates the deterministic policy and updates its parameters using TD-error. The **deterministic policy gradient algorithm** is employed to update the actor weights following equation in figure 2.

3.1 Improvements

We employed a few improvements proposed in the original DDPG article in order to stabilize training.

$$
\begin{aligned}
\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}\right] \\
&= \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta_\mu}\mu(s|\theta^\mu)|_{s=s_t}\right]
\end{aligned}
$$

Figure 2: DPG equation

- **Experience Replay**: The experience replay was originally proposed for Deep Q-learning approach. It employs a buffer of finite size in order to sample randomly from the state-action-rewards tuple space. Independent mini-batches favor a more stable training.
- **Soft updates**: In DDPG the target networks are updated using a **soft copy** of the old weights. The result is a more smooth updating of neural networks parameters.
- **Parallelism**: We adapt DDPG in order to exploit a multi-agent environment. The behavior of the algorithm is the same; the experience buffer and the updates benefit from the high independence of the experiences coming from different parallel agent.

3.2 Hyperparameters

For the experiment we used the following hyperparameters:

Hyperparameter	Value
Replay buffer size	1e6
Batch size	1024
γ	0.99
τ	1e-3
Actor lr	1e-4
Critic lr	3e-4
Update every	20
Update times	10
Episodes	500

4 Results

In figure 3 it's presented the algorithm result per episode. In particular, we were able to solve the environment in less then 50 epochs. The scores kept growing until an optimal maximum of 35, probably due to limited time play per episode. The learning has been stable until 350 when it starts reaching a local minimal. Different approaches and improvements are discussed in section 5.

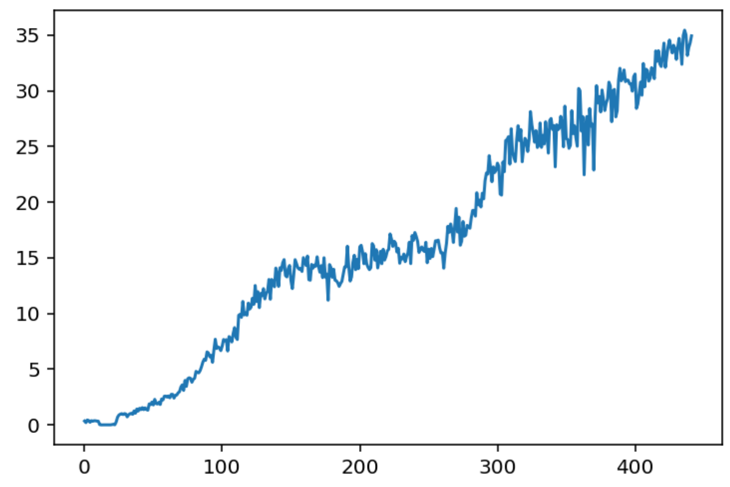

Figure 3: Learning scores per epoch

5 Conclusion and further

We developed a pipeline in order to solve 'reacher' environment with Deep Reinforcement Learning. In particular, we implemented **Deep Deterministic Policy Gradient** following the paper specification. DDPG is an off-policy actor-critic algorithm which has proven stability and optimal results in multiple tasks. We got the algorithm to hack the environment in less then 500 epochs. We believe better results in terms of training speed could be achievable using more advances algorithms like PPO for continuous action. As a matter of fact, PPO could better benefit from the parallelism offered by the environment. We would consider integrating **Prioritized Replay** as a way to optimally sample mini-batches from the queue.