Project 3 - Deep reinforcement learning - Collaboration and competition

Gianluca Scarpellini (gianluca@scarpellini.dev)

April 10, 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 for 2 competitive agents. 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 "Tennis" environment with 2 agents in competition. The tasks consists in throwing the ball to opposing competitor field. The reward is +0.1 for each right hit and a reward of -0.01 for the wrong ones. The agents goal is to keep the ball in play as long as possible. The observation space consists of 24 variables (position and velocity of ball and racket). The action space is continuous and consists of a vector of 2 continuous value corresponding to left-right movement and jump. The task is episodic. The environment is considered solved if an average score of at least **0.5** is maintained for 100 epochs.

3 Algorithm and Implementation

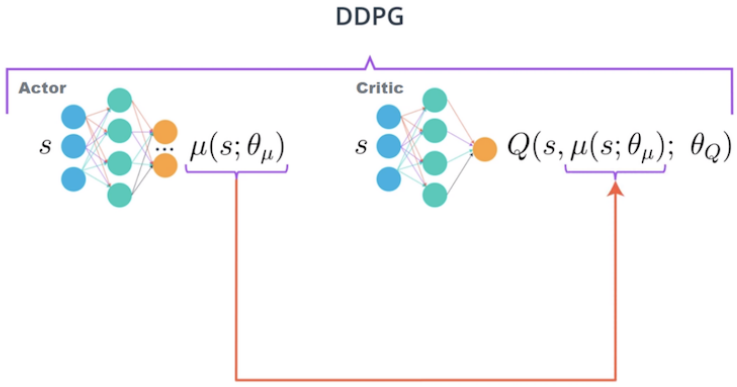

Figure 2: DDPG structure

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.

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

Figure 2: DPG equation

3.1 Improvements

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

- **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. The states of both agents are pushed in the same buffer for more stability.
- **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	1e-3
Update every	20
Update times	10
Episodes	5000

3.3 Neural Network

I implemented 2 neural networks as function approximators for both Actor and Critic as summarized in tables 1 and 2. The actor neural network is a Multi-layer perceptron with 1 hidden layer. I used **leaky relu** as non-linearity for the input layer and the hidden layer, while I used **tanh** for the ouputs of the last layer. The critic network is a Multi-layer perceptron too with multiple-input. I decided to insert a Batch Normalization layer as initially proposed by the Deep Q-learning algorithm in order to normalize the observations input. The first layer output is concatenated with the actions input in order to merge the mutual information. 2 more hidden layers followed by leaky relu further modified the input space. The linear layer as output is returned without non-linearity.

Table 1: Actor network

Actort Network	Value
Fc1	256
fc2	128

Table 2: Critic network

Critic Network	Value
fc1	256
fc2	128
fc3	128

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 2.0, probably due to limited time play per episode. The learning was flat until when it started reaching a local maximum. There was some noise in the precess, probably due the multi-agent environment. Different approaches and improvements are discussed in section 5.

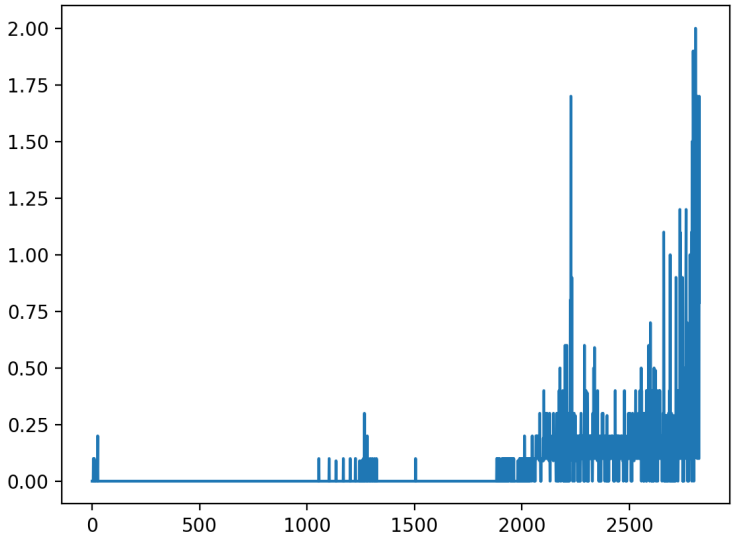

Figure 3: Learning scores per epoch

5 Conclusion and further

We developed a pipeline in order to solve 'tennis' 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 believe better results in terms of training speed could be achievable using more advances algorithms like PPO for continuous ac-

tion. As a matter of fact, PPO could better benefit from the parallelism offered by the environment. We can improve the experience sampling by using a priority based sampling technique to help the agent with the exploration/exploitation problem. More advance solutions could involve D4PG algorithm or Hindsight Experience replay.