Tennis Collaboration and Competition Project

Ennio Nasca

July 5, 2020

1 Introduction

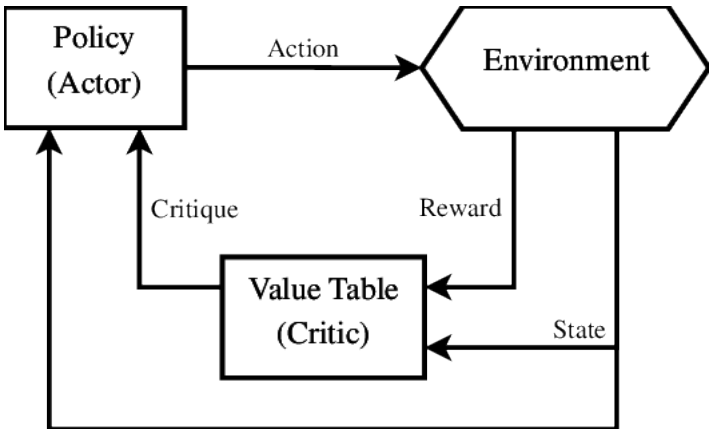
In this report we illustrate how the Multi Agent Deep Deterministic Policy Gradient (MADDPG) algorithm has been applied to the *Tennis* environment. This project is part of the Deep Reinforcement Learning Nanodegree program developed by Udacity. In this repost we provide a description of the model architecture along with the chosen hyperparameters used in the agent definition. Finally, we highlight some future works that can lead to potential improvements upon the current results.

2 Implementation

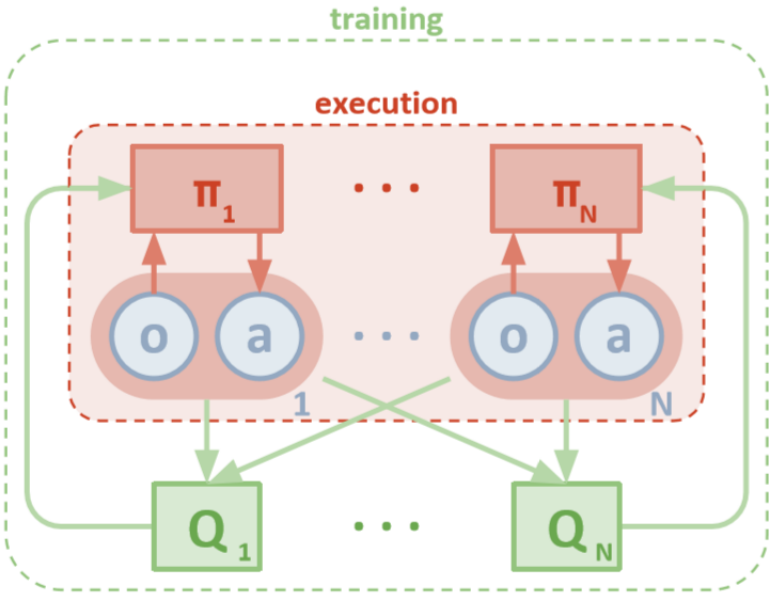
2.1 Agent

The agent is implemented using Multi Agent Deep Deterministic Policy Gradient (MADDPG). This algorithm, as the name suggests, proposes a multi-agent the extension to competitive-collaborative environments of the DDPG model. The approach relies on the definition of a Markov game with N agent. The main difference between a MDP and a Markov game is in the set of actions $A_1, \ldots, A_N$ and a set of observations $O_1, \ldots, O_N$ for each agent.

In the Actor-Critic paradigm, there are two entities, namely the Actor and the Critic, collaborating to achieve the agents' results. Basically, the actor is responsible for learning the policy $\pi(a|s)$, i.e. which actions to pick action given the state, while the critic learns to calculate the action-value function $Q(s, a)$ in order to give the actor a measure of how good is choices are.



In the MADDPG model the actors learn only from local information, i.e. their own observed state, while the critic have access to the full set of observations and actions.



The MADDPG is heavily based on the DDPG approach, which takes inspiration from DQN because it uses:

- **Experience replay buffer** to handle data correlation between consecutive samples.
- **Two neural networks** with identical architecture (local and target networks) for each actor and critic.

but at the same times DDPG introduces some variation with respect to DQN:

- **Soft update** is adopted to updated each target network, instead of performing hard copies of the whole network every n-th iteration.
- **Ornstein–Uhlenbeck process** is used to add noise to the actions, thus creating incentives to exploration.

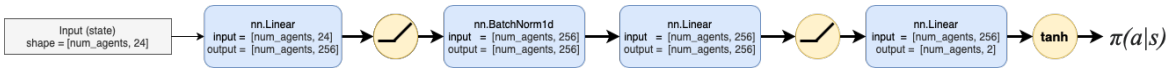
The MADDPG implementation for this project is based on the one provided by Udacity for its Nanodegree, but a few tweaks and chnages were necessary in the code and hyperparameters in order to make the agent work.

The chosen hyperparametes are described above:

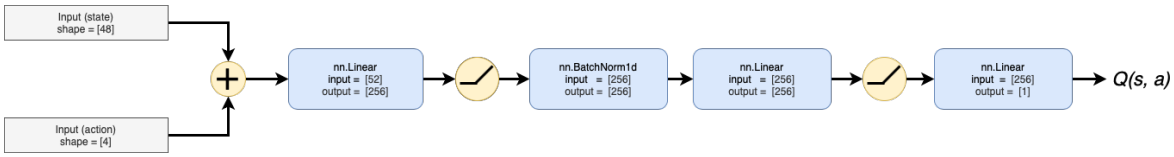
```
BUFFER_SIZE = int(1e5)     # replay buffer size
BATCH_SIZE = 256           # minibatch size
GAMMA = 1                  # discount factor
TAU = 3e-3                 # for soft update of target parameters
LR_ACTOR = 5e-4            # learning rate of the actor
LR_CRITIC = 5e-4           # learning rate of the critic
WEIGHT_DECAY = 0           # L2 weight decay
UPDATE_EVERY = 1           # how often to update the network
N_UPDATES_PER_STEP = 1     # number of updates to perform at each learning step
GRAD_CLIP = 1              # Value at which clip the gradient during backprop
EPSILON_DECAY = 0.9        # decay the noise factor
```

2.2 Model architecture

The architecture for the actor network is as follows:



The architecture for the critic network is as follows:



3 Results

The agent was able to solve the Tennis environment in 1631 episodes, and it took around 1h30min to train on my local machine. All the details can be seen in the project notebook. It can be seen from the image below that the agent would perform quite bad in early episodes of the training process to later on, around episode 1400, find the optimal strategy that leads to high scores (above 2.0).

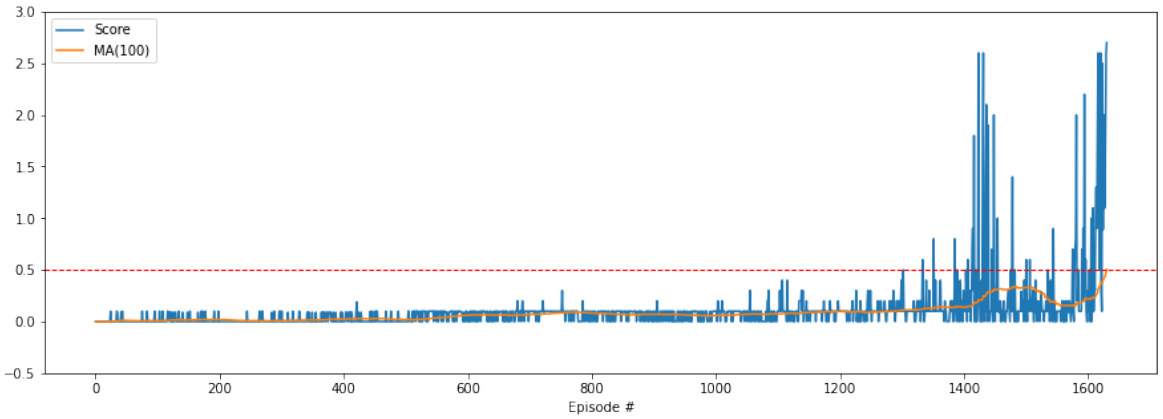


Figure 1: The figure shows the individual episode score (in blue), and the moving average of the last 100 episodes' scores (in orange), of the training process. The threshold of 0.5 for the moving average (in red), is the value for which the environment is considered solved.

4 Ideas for Future Work

There are several improvements that could be applied to the current implementation of MADDPG. For example, the use of **Prioritized Experience Replay**, which favors the sampling of experience tuples that have a larger TD error. Also we could try to simplify the architecture and extend the use of *Batch Normalization* to smooth and speed up the learning curve.

DDPG is one of the most classic actor-critic algorithms and proved to be a valid solution for the task at hand. Nonetheless, other algorithms (like PPO, TRPO, A3C, D4PG) could be implemented for practicing purposes and ideally lead to faster convergence.