Collaboration and Competition with Deep Deterministic Policy Gradient

This report presents the description of the implementation of the Continuous Control in the Tennis Unity ML-Agents environment using Deep Deterministic Policy Gradient (DDPG).

In the Tennis environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores.
- This yields a single score for each episode.

The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

1. Learning Algorithm

The project used the Deep Deterministic Policy Gradient algorithm to solve the Tennis environment problem for two Agents. DDPG was introduced as an actor-critic method that performs well in environments with a continuous action space, which is a known limitation of the popular DQN algorithm.

The algorithm consists of two different types of neural networks: an actor and a critic. The actor network is used as an approximate for the optimal deterministic policy. As a result, only one action is returned for each state, which is considered to be the best action at that state. The critic, on the other hand, learns how to evaluate the optimal action-value function by using actions from the actor.

The pseudocode of this actor critics method used to train the two agents and obtained from the DDPG paper is showed in below graph.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

The hyperparameters combination which let arrive to the solution for average +0.5 reward for the two agents.

BUFFER_SIZE = int(1e5)	# replay buffer size
BATCH_SIZE = 128	# minibatch size
GAMMA = 0.99	# discount factor
TAU = 1e-3	# for soft update of target parameters
LR_ACTOR = 1e-3	# learning rate of the actor
LR_CRITIC = 1e-3	# learning rate of the critic
WEIGHT_DECAY = 0	# L2 weight decay

The DDPG algorithm uses two neural networks for every agent. The Actor Neural Network has the next characteristics.

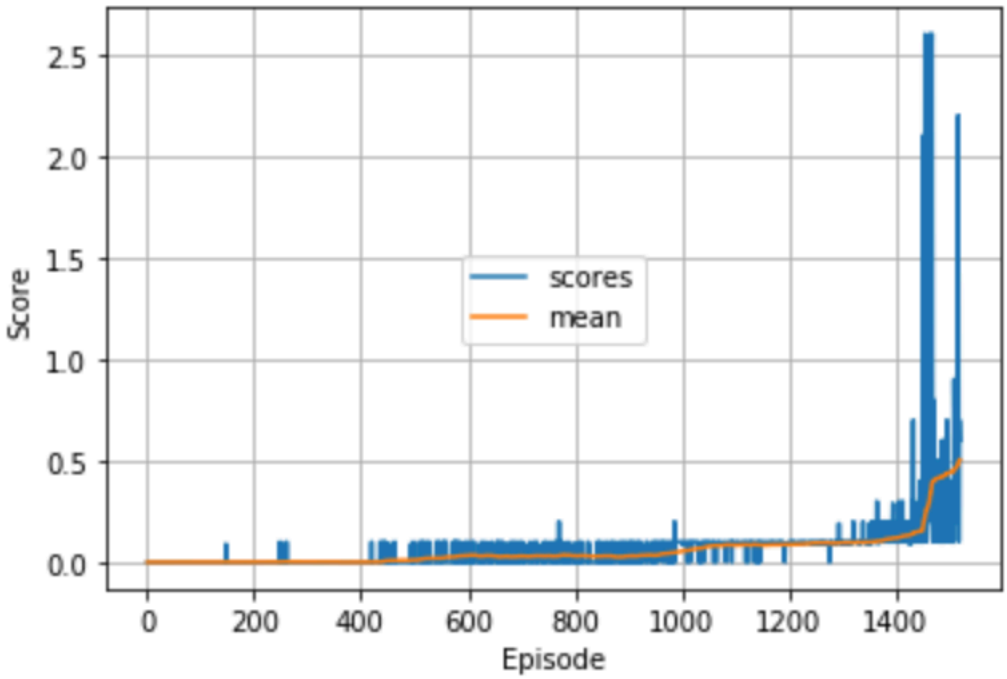
- The model has 3 fully connected layers.
- The first layer takes in the input state which passes through a Relu activation after that a batch normalization is applied.
- The second layer take the batch normalized input and passes through a Relu activation.
- The third layer output as single value representing the action chosen.
- The optimizer of this network is Adam.

The Critic Neural Network maps (state, action) pairs with Q-values and it has similar architecture than the Actor.

- The model has 3 fully connected layers.
- The first layer takes the state and passes through a Relu activation, then applied batch normalization and concatenate with the action size.
- The second layer passes through a Relu activation.
- The last layer calculates the Critic's output which estimates the Q-value of the current state and the action given by the actor.
- Also, in this case the optimizer used is Adam.

2. Plot of Rewards

In this project, the two agents received an average of +0.5 for 1419 episodes and the next graph presents the evolution of the scores to solve the problem. The graph shows how is increasing in the mean score for every 100 episodes.



3. Ideas for Future Work

Some improvements to be done to the actual project are the following.

- Solve using Multi-Agent Actor-Critic for Mixed Cooperative-Competitive algorithm.
- Try more deep neural networks
- Fine tuning of the model hyperparameters