Report for Udacity DRLND Project 3

Collaboration and Competition (Tennis)

Introduction

This report describes the Competition and Collaboration project results using the Unity Tennis environment. This report demonstrates an implementation of the MADDPG algorithm which is able to successfully complete the project requirements.

In this 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.

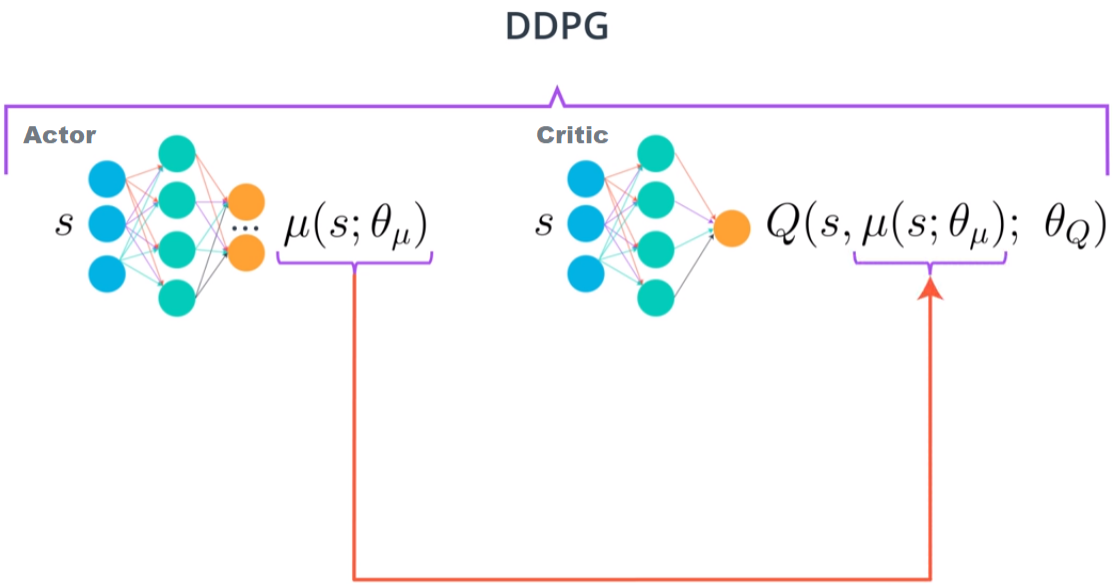
The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping.

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 to be solved, when the average (over 100 episodes) of those scores is at least +0.5.

DDPG and MADDPG

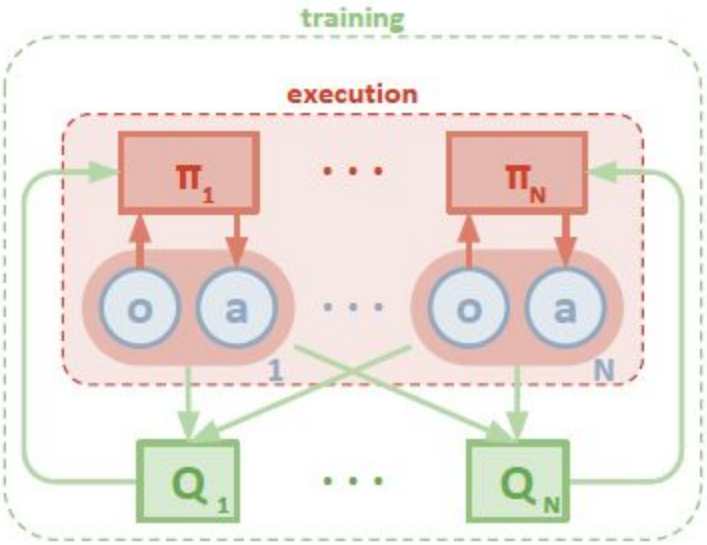
DDPG, short for Deep Deterministic Policy Gradient, is a model- free off-policy actor-critic algorithm, combining DPG with DQN. Recall that DQN (Deep Q- Network) stabilizes the learning of the Q-function by using experience replay and a frozen target network. The original DQN works in discrete space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy.

In DDPG, we use 2 deep neural networks: one is the actor and the other is the critic:



In this report, we will focus on the modification made on the standard DDPG to solve the Tennis environment.

The main difference is that all agents share the same replay buffer memory. By sharing a common replay buffer, we ensure that we explore all the environment. Note that the agents still have their own actor and critic networks to train as the picture below.



During execution step (actor process) each agent uses its own independent actor, observes the state o by itself, and outputs the determined action a. At the same time, the

actor only uses the data generated by itself. Each agent also corresponds to a critical, but the critic receives all data from actors at the same time. We refer it as data centralized critic, and train critic by using all data. This kind of centralized criticism differs from general centralized criticism, and the main difference in that there is N numbers of critics existing.

Here is MADDPG algorithm defined in the paper Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments

Multi-Agent Deep Deterministic Policy Gradient Algorithm

For completeness, we provide the MADDPG algorithm below.

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

Implementation

The MADDPG Agent contains two instances of a DDPG Agent and also defines the Replay Buffer. The replay buffer is defined using a buffer size of 1e6 and a batch size of 256.

The DDPG Agent follows the actor critic model. The learning rate for the critic is set as 1e-3 and the learning rate for the actor is set as 1e-4. Both the actor and the critic employ two hidden layers of 400 and 300 units respectively. The actor only sees its own

observations. Therefore, the actor model looks like: 256x24 -> 256x400 -> 256x300 -> 256x2

The critic sees the observations of both agents and the actions of both agents. The critic employs a batch normalization layer following the first fully-connected layer and it is the output of the batch normalization layer that is merged with the actions. Therefore, the critic model looks like: 256x48 -> 256x400 -> 256x404 -> 256x300 -> 256x1

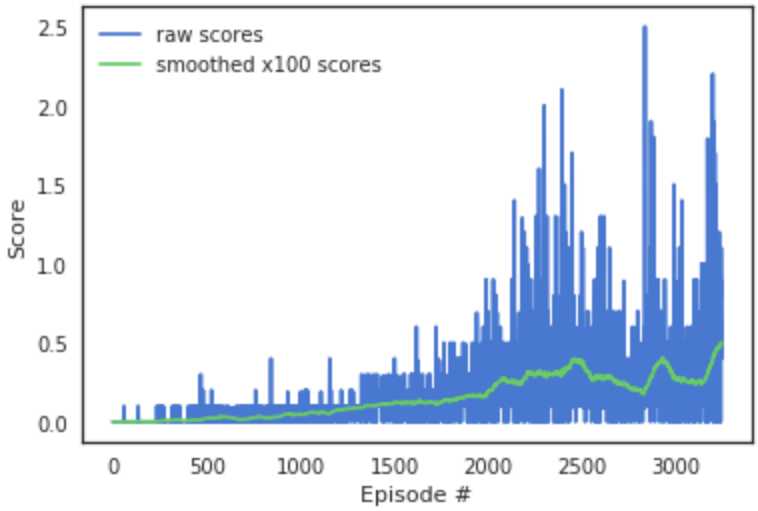
Other Hyper parameters

soft update hyper parameter tau set to 0.01

future rewards discount gamma set to 0.99

Results

The agents are trained until they solve the environment, that is to say when one the two agent obtains an average reward of at least +0.5 over the last 100 episodes. The agent achieves an average score of +0.5029 in just 3241 episodes.



Future ideas

The following techniques are worth pursuing for faster convergence or higher peak performance:

- hyperparameter search
    - the mini-batch size of 256 slows computations versus a smaller batch size and it is likely unnecessary
    - the learning rates selected may not be optimal and may result in slower convergence
- architecture search
    - it is possible that a deeper model or adding / removing units in each of the hidden layers may improve performance
- prioritized experience replay
    - replay of states and actions seen to be helpful to convergence may improve performance

Try to combine PPO Proximal Policy Optimization by Open AI for other lager continuous state and action space cases with more agents:

MADDPG: Each Critic needs to observe the state and actions of all agents. It is not efficient for a large number of uncertain agent scenarios, and when the number of agents is particularly large, the state space is too large. Each agent corresponds to an actor and a critic. When there are a large number of agents, there are a model of huge size. It might also combine PPO to DDPG.