Udacity Project3 MARL report

Goal of the project

To maximize the reward for both agents in the tennis game.
The reward is set to keep two agents playing. So the reward of hitting the ball is +0.1. If an agent lets a ball hit the ground or out of bounds, the reward is -0.01.

The task is said to be solved when the average reward reached 0.50 over a consecutive 100 episodes.

For more environment details, see README.md

Introduction to DDPG

Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

Learning algorithms : MADDPG

For agent i, the expected return gradient is:

(Q is the centralized action-value function with all agents' action and addition info : x)

x could be: all agent's observations, or other addition state info…..

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s\sim p^{\boldsymbol{\mu}}, a_i\sim \boldsymbol{\pi}_i}[\nabla_{\theta_i} \log \boldsymbol{\pi}_i(a_i|o_i) Q_i^{\boldsymbol{\pi}}(\mathbf{x}, a_1, ..., a_N)].$$

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_i) = \mathbb{E}_{\mathbf{x}, a\sim\mathcal{D}}[\nabla_{\theta_i}\boldsymbol{\mu}_i(a_i|o_i) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, ..., a_N)|_{a_i=\boldsymbol{\mu}_i(o_i)}],$$

The actions would be estimated using a learned approximate policy of other agents. This solves the problem of non stationary environments.

The overall pseudocode is as follows:

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

Architecture of the network

The agent's network:

```
Actor(
  (fc1): Linear(in_features=24, out_features=100, bias=True)
  (bn1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=100, out_features=75, bias=True)
  (fc3): Linear(in_features=75, out_features=75, bias=True)
  (fc4): Linear(in_features=75, out_features=2, bias=True)
)
```

```
Critic(
  (fcs1): Linear(in_features=24, out_features=100, bias=True)
  (bn1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=102, out_features=100, bias=True)
```

```
  (fc3): Linear(in_features=100, out_features=1, bias=True)
)
```

Hyperparameters

If there is need to fine tune the parameters, directly change it in Tennis.ipynb

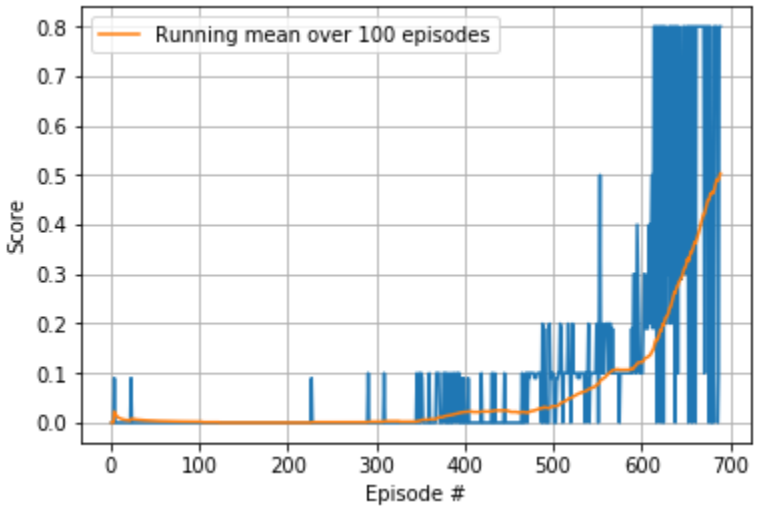
```
BUFFER_SIZE = int(1e5)
BATCH_SIZE = 128
GAMMA = 0.995
TAU = 0.002
LR_ACTOR = 2.0e-3
LR_CRITIC = 1.5e-3
WEIGHT_DECAY = 0
```

Training Implementation

- `Tennis.ipynb` :
  - The Actor and Critic classes each implement a *Target* and a *Local* Neural Networks used for the training.
  - MADDPG part uses the DDPG model to initiate and update agents.
  - The result shows with our parameters, the goal can be achieved.
- `model.py`: Implement the **Actor** and the **Critic** classes.
  - This file contains the Actor and Critic Networks for DDPG Agents

Results

```
Episode 100 Average Score: 0.0018
Episode 200 Average Score: 0.0000
Episode 300 Average Score: 0.0019
Episode 400 Average Score: 0.0216
Episode 500 Average Score: 0.0333
Episode 600 Average Score: 0.1218
Episode 689 Average Score: 0.5034
Environment solved in 589 episodes! Average Score: 0.5034
```



Future improvements

1. Try On-policy algorithms
    a. There is a recent paper about PPO in cooperative multi-agent learning schemes.
        i. Chao Yu et al. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games(2022)