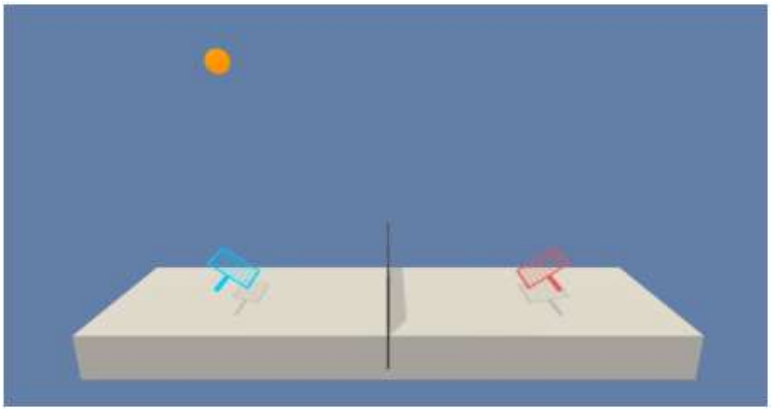
Deep Reinforcement Learning Nanodegree

Project 3 – Collaboration and Competition

1. Introduction



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 24 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, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically,

- After each episode, add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. The maximum of these 2 scores is taken.
- 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.

2. Multi-Agent Deep Deterministic Policy Gradient Algorithm

The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm is implemented to solve the Tennis environment.

2.1. Multi-Agent Actor-Critic Method

The Multi-Agent Actor-Critic method adopts a framework of centralized training and decentralized execution as shown in Figure 2-1.

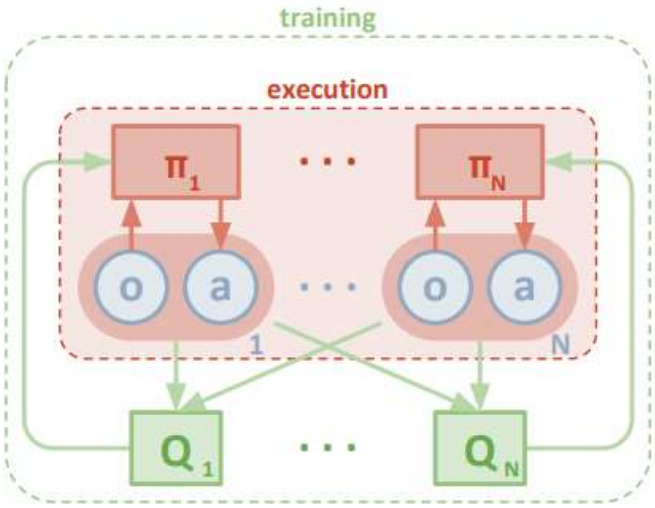


Figure 2-1 Multi-Agent Actor-Critic Method

Similar to the Single-Agent Actor-Critic method, each agent has its own actor and critic network.

During training
- actor network for each agent has access only to local / its agent's observations and actions
- critic for each agent has fully visibility of the environment – it uses local / its agent's observations and actions, plus the observations and actions of all the other agents

During execution
- actor networks are present, hence, all observations and actions are used
- critic networks are absent

The Multi-Agent Actor-Critic method can be used in cooperative, competitive and mixed scenarios.

2.2. Network Architecture

The actor network, shown in Figure 2-2, comprises of three layers:
- An input layer with 24 neurons corresponding to 24 possible observation states
- Two hidden fully-connected layers
  - Layer 1 with 256 neurons activated by a `ReLU` function
  - Layer 2 with 128 neurons activated by a `ReLU` function
- An output fully-connected layer with 2 neurons activated by a `tanh` function, corresponding to action vector with two numbers indicating (1) the movement

toward (or away from) the net and (2) jumping, where each number has a value between -1 and 1

The critic network, shown in Figure 2-3, comprises of three layers:

- An input layer with 24 neurons corresponding to 24 possible observation states
- Two hidden fully-connected layers
  - Layer 1 with 256 neurons activated by a `ReLU` function
  - Layer 2 with 128 neurons activated by a `ReLU` function where the input is the concatenation of layer 1 output and action vector
- An output fully-connected layer with single neuron, activated by a `ReLU` function, corresponding to a scalar Q value

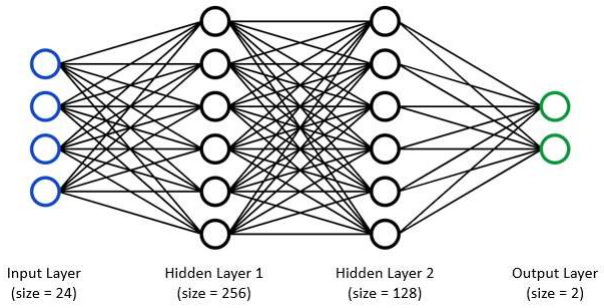


Figure 2-2 Actor Network

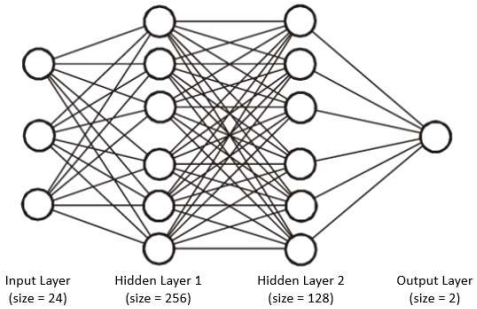


Figure 2-3 Critic Network

Noise is added using an Ornstein-Uhlenbeck process to the action values at each time step, and it decays over time. That is, more noise is introduced earlier in the training process (i.e. higher exploration) and decreases over time as the agent gains more experience (i.e. higher exploitation).

To stabilize the training, three techniques are used:

a. Experience Replay

During training, the sequence of observations can be highly correlated. Each agent has its own (circular) replay buffer to store each local experience tuple (S, A, R, S') as the agent interacts with the environment. This decouples the occurrence of experience entries across agents.

Using experience replay to sample a small batch of tuples from the replay buffer randomly out-of-order breaks the correlation between consecutive experience tuples. This allows the agent to learn more from individual tuples multiple times, recall rare occurrences, make better use of experiences and stabilize the action space - prevent action values from oscillating or diverging catastrophically.

b. Fixed Target

Introduced for DQN, the parameters of the network is shifted based on a constantly moving target, and this can potentially lead to harmful correlations. Two separate networks with identical architectures are used, one with fixed targets and another learns the Q value. The target network weights are updated less often (or more slowly) than the online network. This decouples the target from the parameters, making the learning algorithm more stable, and less likely to diverge or fall into oscillations. Since there are two neural networks for the actor

and critic, there are two target neural networks as well, one for actor and one for critic.

c. Soft Target Updates

The target actor and critic networks are updated such that at each update step, 2% of the local network weights are mixed with the target network weights. That is, 98% of the target network weights are retained and 2% of the local network weights are added.

2.3. Hyper-parameters

The hyper-parameters used in the MADDPG algorithm are shown in Table 1.

Hyper-parameter	Name in code	Value
Discount factor, γ	GAMMA	0.99
Replay buffer size	BUFFER_SIZE	10^5
Minibatch size	BATCH_SIZE	512
Soft update parameter, τ	TAU	2×10^{-2}
Learning rate of actor	LR_ACTOR	10^{-3}
Learning rate of critic	LR_CRITIC	10^{-3}
Initial epsilon, ε_0	EPSILON	1.0
Epsilon decay, δ	EPSILON_DECAY	10^{-5}

Table 1 Hyper-parameters

3. Results

The model training completes once the score – averaged over the past 100 episodes – exceeds +0.5. The model training completes in 984 episodes as shown in Figure 3-1.

```
Episode 800:    Average Score: 0.34     Best Score: 1.3000000193715096
Episode 825:    Average Score: 0.34     Best Score: 1.3000000193715096
Episode 850:    Average Score: 0.33     Best Score: 1.3000000193715096
Episode 875:    Average Score: 0.31     Best Score: 1.3000000193715096
Episode 900:    Average Score: 0.31     Best Score: 1.3000000193715096
Episode 925:    Average Score: 0.36     Best Score: 1.3000000193715096
Episode 950:    Average Score: 0.38     Best Score: 1.3000000193715096
Episode 975:    Average Score: 0.48     Best Score: 3.300000049173832

Environment solved in 984 episodes!     Average Score = 0.50 over last 100 episodes
```

Figure 3-1 Training Average Score Over 100 Episodes

Figure 3-2 depicts the model scores by episode.

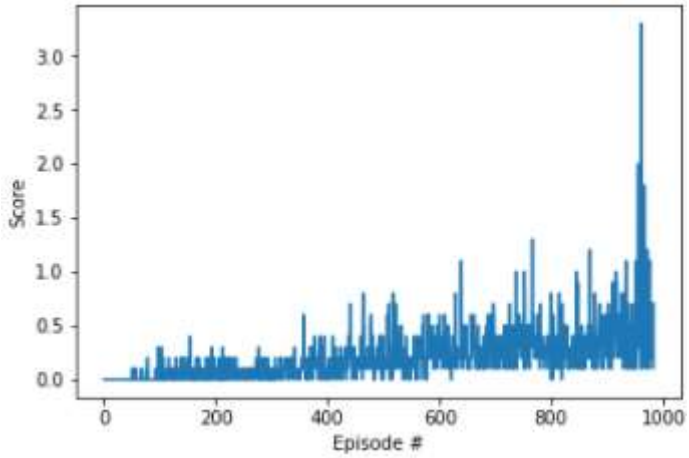


Figure 3-2 Scores by Episode

4. Potential Improvements

- Hyper-parameter tuning (e.g. number of layers and neurons, learning rates, batch and buffer sizes, noise levels)
- Investigate prioritized experienced replay