PROJECT REPORT

UDACITY

DEEP REINFORCEMENT LEARNING NANODEGREE

Project 3: Collaboration and Competition

AUTHOR:
Thomas Teh

Date: November 7, 2018

1 Multi-Agent DDPG

In this project, I tackled the Unity Tennis environment using a multi-agent deep reinforcement framework. The tasks of the agents are to keep the ball within bounds and up in the air as long as possible. Under the multi-agent setting, the two agents that interact with each other are DDPG agents (see 5).
It is important to note that the interaction between the agents are neither cooperative nor competitive following reasons:

- The agents do not have incentives to outperform each other in terms of scores.
- The agents do not formulate a shared strategy to keep the ball in play as long as possible.

Each agent chooses an action that maximizes the rewards independently, but their objectives are such that it results in cooperative play. When one agent successfully keeps the ball in play, the other agent benefits by having the opportunity to increase its reward by also keeping the ball in play.

2 The MADDPG Architecture

The MADDPF architecture was introduced by (Lowe et al., 2017) as shown in Figure 1. While the algorithm utilizes the DPPG algorithm for each agent, it is important to note that the actor networks for each agent takes in observations of that particular agent as input. The actor networks of the other agents do not have access to those observations.

However, the critic networks of the agents have access to the observations and actions of **all** agents as inputs. All the agents will contribute to the experience replay buffer and this will allow the agents to learn from one another's experiences.

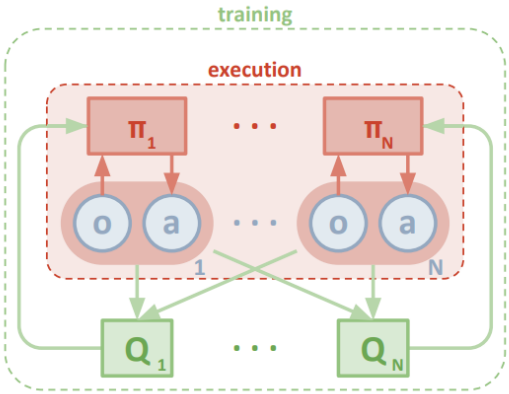

Figure 1: MADPGG: multi-agent decentralized actors and centralized critics approach(from (Lowe et al., 2017))

2.1 The Actor Network

The actor network, as shown in Figure 2 consists of the following:

- Input layer: The input to the network are the states of the environments observable by the **individual** agent.
- Hidden layer: The two hidden layers are made up of 512 and 256 neuron units respectively with exponential linear units.
- Output layer: The network outputs the action taken by the agent, which consists of 2 numbers

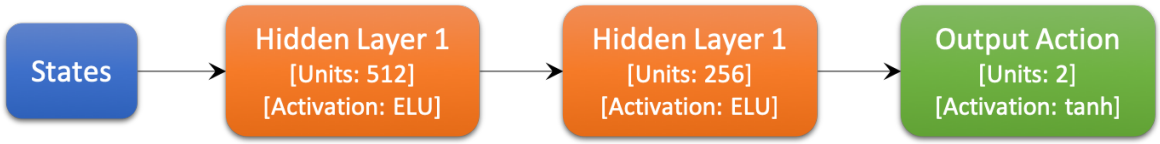

Figure 2: Architecture of the actor network.

2.2 The Critic Network

The critic network, as shown in Figure 3 consists of the following:

- Input layer: The input to the network are the states of the environments observable by the agents.
- Hidden layer: Hidden layer 1 takes in the states observable by the agent as input. Hidden layer 2 takes both the output from hidden layer 1 and the action as inputs. Hidden layer 3 takes the output of hidden layer 2 as input.
- Output layer: The network outputs the value function of the state-action input.

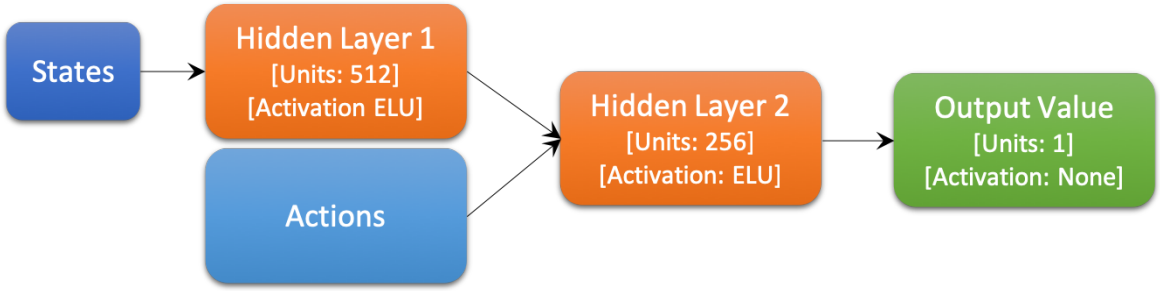

Figure 3: Architecture of the critic network.

While in (Lillicrap et al., 2015), the authors mentioned the application of batch normalization with ReLU activation functions for their actor-critic networks. From

my experience in Project 2, I found that the training process is more stable by using exponential linear units (ELUs) introduced by Clevert et al. (2015). The ELUs retain most of the characteristics of ReLUs but they have an implicit regularization effect which made training much more stable.

3 Hyperparameters

The hyperparameters used in training the MADDPG algorithm are listed below:

Hyperparameter	**Value**
Actor learning rate, α^{μ}	0.00015
Critic learning rate, α^{Q}	0.00100
Discount factor, γ	0.99500
Buffer size	150,000
Minibatch size	256
Soft update weights, τ	0.0020
Update frequency	Every 1 time step
Number of updates	1 update
Initial weight for noise	1.0
Noise weight decay	0.999990
Long term mean for noise, μ_{ou}	0.00
Mean reversion rate for noise, θ_{ou}	0.15
Volatility for noise, σ_{ou}	0.25

4 Results

The average rewards during the training process is shown in Figure 4. The MADDPG algorithm achieves a mean score of 2.00 (exceeding the required 0.50), across 100 episodes after 737 episodes.

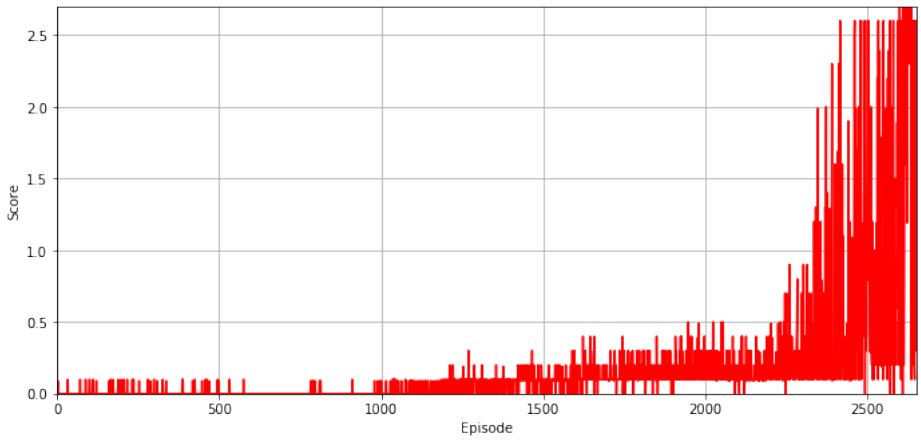

Figure 4: Average reward vs number of episodes

5 Ideas for Future Work

The solution of the problem can potentially improved by applying different algorithms within the multi-agent reinforcement learning framework. It is possible to replace each individual agents with a different algorithm such as A3C and PPOs.

Also, a more interesting future work to be done is to deploy the multi-agent deep reinforcement learning in a pure competitive or collaborative game play. The initial intuition is that there is a need for an overall loss function across agents that consists of competing objectives (in a similar fashion as the generative adversarial networks, in which the loss function expresses the competition between the discriminator and the generator).

Appendix: Deep Deterministic Policy Gradient

DDPG, as introduced by Lillicrap et al. (2015), was use to solve this problem. The pseudo-code for DDPG is given in Algorithm 1. The action-value function describes the expected return after taking an action a_t in state s_t and thereafter following policy π:

$$\begin{aligned} Q^{\pi}(s_t, a_t) &= \mathbb{E}_{\pi}\left[R_t | s_t, a_t\right] \\ &= \mathbb{E}_{\pi}\left[\sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i) | s_t, a_t\right] \end{aligned}$$

The action-value function follows the Bellman equation:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left[r(s_t, a_t) + \gamma \mathbb{E}_{\pi}\left[Q^{\pi}(s_{t+1}, a_{t+1})\right]\right]$$

Given that the target policy is deterministic, we can described it as $\mu : \mathcal{S} \rightarrow \mathcal{A}$

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{\pi}\left[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right]$$

In DDPG, we use non-linear function approximator to learn both the $Q(s_t, a_t)$ action-value function and the policy $\mu(s_t)$. Firstly, the action-value function is learned using Q-learning, whereby $\mu(s_t) = \arg\max_{a \in \mathcal{A}} Q(s_t, a_t)$. Similar to the DQN algorithm introduced by Mnih et al. (2015), the action-value function $Q(s_t, a_t)$ is learned by minimizing the following loss function:

$$L(\theta^Q) = \mathbb{E}_{\pi'}\left[(Q(s_t, a_t) - y_t)^2\right]$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) | \theta^Q)$$

Secondly, the actor network is learned by using the DPG algorithm introduced by Silver et al. (2014). The actor is updated by the following:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{\pi'}\left[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{\pi'}\left[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned}$$

One final issue on DDPG is exploration. Exploration is introduced to the algorithm by including some noise in the action.

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_\sqcup.$$

$\mathcal{N}_t$ is an Ornstein-Uhlenbeck process

$$d\mathcal{N}_t = \theta_{ou}(\mu_{ou} - \mathcal{N}_t)dt + \sigma_{ou} dW_t$$

where θ_{ou} is the mean-reversion rate, μ_{ou} is the long-term mean, $sigma_{ou}$ is the volatility and W_t is a Brownian motion.

Initialization:
Randomly initialize critic network $Q\left(s, a|\theta^Q\right)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$.
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
Selection action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$ according to the current policy and exploration noise.
Execute action a_t and observe reward r_t and observer 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 $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$.
Update the critic network by minimizing:

$$L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2.$$

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
end

Algorithm 1: Deep Deterministic Policy Gradient (DDPG)

References

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*. pages 4

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. pages 3, 5

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390. pages 2

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529. pages 5

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *ICML*. pages 6