Report

Lifeng Wei

December 2018

1 Introduction

This report summaries my explorations for solving the **Tennis** environment. First it briefly discuss the algorithms applied. Then it shows some of my experimental results for visualization. In the last some possible improvements are also discussed.

The description for the environment can be found in the Readme.md file and will not be repeated here.

2 Algorithm

2.1 MADDPG

The full name of MADDPG [3] is Multi-Agent Deep Deterministic Policy Gradient. You can take it as the generalization of DDPG [2], in for environments where there are multiple agents. If you are not familiar with DDPG, please check the origin paper. But for a brief catch up, another report in the **Continuous Control** repository should help.

2.1.1 Difference from DDPG

One might say that for multiple agents, we can train them separately, using DDPG or PPO or TRPO etc. Actually that works and **for now this environment is solved by two separate DDPG agents sharing the same policy**. But using MADDPG provides extra benefits, which allows MADDPG to solve more complicated environment, train with higher efficiency, get a more stable solution and perhaps a higher reward.

2.1.2 Centralized training with Decentralized execution

This is the biggest difference between MADDPG and DDPG. The main idea of centralized training with decentralized execution is to train the agent with extra information to accelerate training but each agent only has its local information to choose an action. The most direct setting is to give the critic access to all information, including information of other agents, while the actors only have access to their own local observations.

For example, if we have n agents, with state information $s_1, s_2, \ldots, s_n$ and actions $a_1, a_2, \ldots, a_n$. Here the actions are determined by

$$a_i = \mu(s_i|\theta^{A_i}) + \mathcal{N}_i$$

The first term is actor's decision and the second term is a noise. Each agent has critic parameter θ^{Q_i}. Then the estimated future reward is:

$$Q(s_1, s_2, \ldots, s_n, a_1, a_2, \ldots, a_n|\theta^{Q_i})$$

This formulation has two major benefits.

- First, different agents can easily have different reward structures by simply changing the parameters. This includes totally conflicting reward systems, which means the agents are competitors.
- Second, this setting makes the distribution of reward stabilized. The reason behind is that the future of i^{th} agent is not only determined by its own policy. The change of one agent's policy will also influence the expected future reward of other agents. So if the critic only has access to the agent's own local information, the reward signals will have a larger variation and extremely unstable, which makes the training much harder. But with knowing all information, especially the actions of other agents, the reward signal will be stable since in the MDP setting we know the distribution of following states if we know the current state and actions.

Under this setting we train the critic by minimizing

$$\mathbb{E}_{from\ experience}(r_{t,i} + \gamma \cdot Q'(S_{t+1}, \hat{A}_{t+1}|\theta^{Q_i}) - Q(S_t, A_t|\theta^{Q_i}))^2 \quad i = 1, 2, \ldots n$$

γ is the discount factor. $r_{t,i}$ means the reward for agent i at time t. $S_t = (s_{t,1}, s_{t,2}, \ldots, s_{t,n})$ is the joint of state information of all agents at time t. Similarly $A_t = (a_{t,1}, a_{t,2}, \ldots, a_{t,n})$ is the joint of all actions at time t.
$\hat{A}_t = (\mu(s_{t+1,1}|\theta^{A_1}, \mu(s_{t+1,2}|\theta^{A_2}), \ldots, \mu(s_{t+1,n}|\theta^{A_n}))$ which is the noiseless actions that actors would take. Q' means it's target network.

The actors are trained by maximizing

$$\mathbb{E}_{from\ experience}Q(S_t, \hat{A}_t^i(\mu_{t,i})) \quad i = 1, 2, \ldots, n$$

Here $\hat{A}_t^i(\mu_{t,i}))$ means replace $a_{t,i}$ in A_t by $\mu_{t,i} = \mu(s_{t,i}|\theta^{A_i})$.

This is only the basic setting. We can see that in this training every agent needs to know the policy of other agents in order to train its critic. This assumption can be removed with the following improvement.

2.1.3 Inferring Policies of Other Agents

This idea means each agent has some 'prediction' for the actions that other agents would take. Under this setting, the critic is trained with these inferred actions and these predictions are improved by imitating the actual actions taken by other agents.

2.2 DDPG

In this section we will provide the design of our DDPG agents.

2.2.1 Network Structure

- Actor network
  The structure of actor is exactly the same as mentioned in the original DDPG paper [2]. It uses Relu [4] as activation and Batch normalization [1] to accelerate training. The structure is:

  FC(state size, 400)→Relu→Batch Normalization→FC(400,300)→Relu→Batch Normalization→FC(300, action size)

- Critic network
  Here we tested two different structures of critic. The main difference between them is the place to include action. The structures are:

  1. Structure 1 (activated by 'critic=1')
     FC(state size, 400)→Relu→Batch Normalization→Add action, FC(400+action size, 300)→Relu→FC(300,1)
  2. Structure 2 (activated by 'critic=2')
     FC(state size, 400)→Relu→Batch Normalization→FC(400, 300)→Relu→Batch Normalization→Add action, FC(300+action size,1)

In the experiment I found that the second structure does not learn at all. I guess the reason is that action directly participates in the final prediction of reward. There is not enough interaction and non-linearity relationship between state and action.

2.2.2 Network Updates

Here we illustrate in details how networks are updated in each learning step. First we explain our notations:

- From replaybuffer we get state S_i, action a_i, reward r_i, state after action S_i'. $i = 1, 2, \ldots, N$
- Actor networks $A(s|\theta^A_{local})$, $A(s|\theta^A_{target})$, $Q(s, a|\theta^Q_{local})$, $Q(s, a|\theta^Q_{target})$
- Reward discount rate γ. Soft update parameter τ. Learning rate for local actor η_1. Learning rate for local critic η_2.

First we update the critic network with TD method. You can do TD(λ) but here I just used TD(0).

$$\theta^Q_{local} = \theta^Q_{local} - \frac{1}{N}\nabla_{\theta^Q_{local}} \sum_{i=1}^{N} (r_i + Q(S'_i, A(S'_i|\theta^A_{target})|\theta^Q_{target}) - Q(S_i, a_i|\theta^Q_{local}))$$

$$\theta^A_{local} = \theta^A_{local} + \frac{1}{N}\nabla_{\theta^A_{local}} \sum_{i=1}^{N} Q(S_i, A(S_i|\theta^A_{local})|\theta^Q_{local})$$

After updating these two local networks, we update two target networks:

$$\theta^A_{target} = \tau * \theta^A_{local} + (1-\tau) * \theta^A_{target}$$

$$\theta^Q_{target} = \tau * \theta^Q_{local} + (1-\tau) * \theta^Q_{target}$$

2.2.3 Other details

In practice there are also some other details, including choice of hyperparameters, applied. I list them here:

- Network Initialization: The initialization of networks' weights are not totally random. As suggested in the DDPG paper, all weights in fully connected layers, except for the output layer, is initialized uniformly in interval $(-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}})$. Here f is the input size of the layer. For the output layer, actor's network is uniformly sampled from $(-10^{-3}, 10^{-3})$ while critic's network is from $(-10^{-4}, 10^{-4})$.

- Hyperparameters: $\eta_1 = 0.0001$, $\eta_2 = 0.001$, $\tau = 0.001$, batch size=128, replaybuffer size=300,000

- Learning speed: 1 time learning for each timestep

- Gradient Clipping: When the L2-norm of gradient of local critic network is larger than 1, I would shrink it to 1.

- Decreasing level of noise: The level of noise is always getting smaller. At the i^{th} episode, the noise is original noise $\times \frac{1}{\sqrt{i}}$

3 Result

Here I present several independent results to show the instability of DDPG. In each of these plots there are two lines. The yellow line is the benchmark (0.5) and the blue line is the trajectory of mean rewards of past 100 episodes. The range of x axis is from 0 to 5000, which means there are in total 5000 episodes. We can see the learning speed varies in a large range for the agents. Some of them learns fast, solving the problem in around 2000 episodes. Some of them

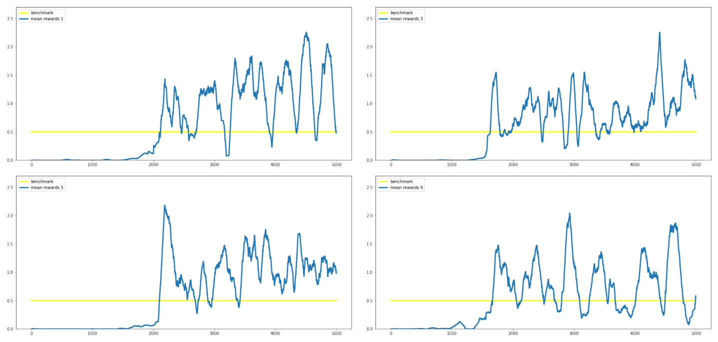

Figure 1: These ones learn fast

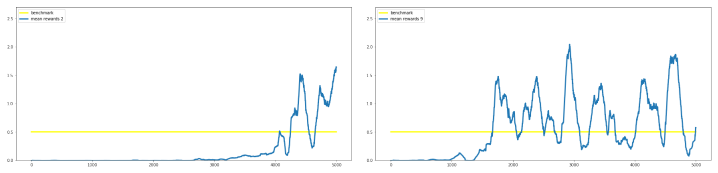

Figure 2: These ones learn slow

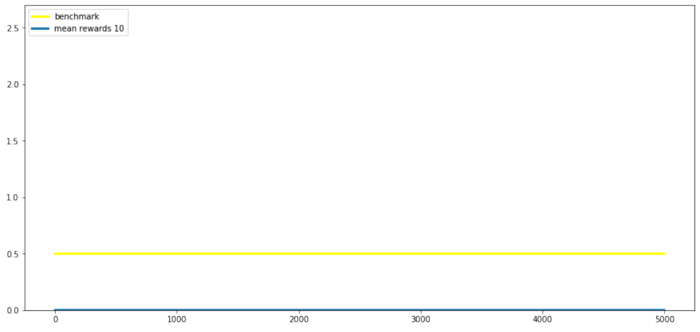

Figure 3: This one doesn't learn

are slow, solving the problem near 4000 episodes and one of them cannot even learn!

But there is still one thing in common for the successful agents. Their learning is not stable. The trajectories keep going up and down, which means they constantly learn wrong things.

4 Future Work

Next I will see how MADDPG could improve the performance of DDPG agents. I will also try to get an average reward over 2.5. Lastly, I will compare the performance of DDPG and MADDPG in the soccer environment.

References

[1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[3] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[4] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.