Scalable Multi-agent Reinforcement Learning for Cooperative Environment

Songyang Zhang,914502403 and Fuwei Li,914497107

Department of Electrical and Computer Engineering
University of California, Davis

Abstract: With the development of artificial intelligence, reinforcement learning has achieved huge success, especially in single agent domain. However, the traditional reinforcement learning method cannot fit well in the multi-agent cases because of the non-stationary environment and complex multi-agent interactions. Aiming to improve the robustness and scalability of reinforcement learning in the multi-agent domain, we proposes a scalable reinforcement learning framework for cooperative environment. We use centralized training with decentralized execution to increase the flexibility for the design of each agent. Besides, we use the bi-directional LSTM to deal with the scalability problem. Then we implement our algorithm in a prey-and-predator games. It shows that our framework can help improve the scalability and robustness in the multi-agent scenarios where agents are able to have various physical and informative interactions.

Index Terms: Multi-agent reinforcement learning, robustness, scalability

1. Introduction

REINFORCEMENT learning (RL) has been successfully applied to solve challenging problems in different areas, from games to system control. Especially in single agent domain, RL has shown its power and flexibility by solving complex problems. However, there are a number of important applications where interaction between multiple agents is very important. For example, multi-player games [1], multi-robot control [2] and load balancing in the system with multiple base stations [3] are all faced with the cooperation or competition between multi-agents. Thus, there has been growing interest in extending RL to the multi-agent domain where modeling or predicting the behavior of other agents in the environment is very important.

Unfortunately, traditional reinforcement learning may not fit the multi-agent problems well. One problem is that the environment will become non-stationary from the perspective of individual agents because each agent's policy will keep changing [4]. This results in the robustness challenges and limits the straightforward use of historical experience replay. Another problem is that each agent cannot obtain all the information in the environment as the their observations of other agents are partial and inaccurate [5]. Then how to estimate other agents' policy and update individual policy based on local information becomes a key problem. In addition, scalability problem is also an urgent problem to be solved in the multi-agent cases. In many applications, number of agents is dynamic as new agents will join or some old agents will leave while playing. Then we have to totally retrain our model when number of agents changes. In some situations, this retraining time is unacceptable. For example, it's really dangerous to retrain the unmanned aerial vehicle(UAE) in a aerial photography system when some new UAEs are introduced.

To address the problem mentioned above, we try to design a scalable multi-agent reinforcement learning framework to improve the robustness and scalability, especially for cooperative environment in our work. Inspired by OpenAI's work [6], we establish a multi-agent RL framework with centralized training and decentralized execution which: (1) allows agents to have their own reward function and policy; (2) leads each agent to use local information(i.e. their partial observations) at execution time; (3) is applicable to deal with both cooperative and competitive environment. In addition, we use the bi-directional long short-term memory network(Bi-LSTM) to deal with the scalability problem in the training part.

In this report, we will first introduce our scalable multi-agent reinforcement learning framework. Then we will talks more about our implementation scenarios including system setup and reward design. After that, experiment results and simulation analysis will be given. Finally, we will conclude our project and propose future work.

2. Methods

To address the multi-agent conditions, we propose a scalable framework with centralized training and decentralized execution based on bi-direction LSTM as Fig 5.

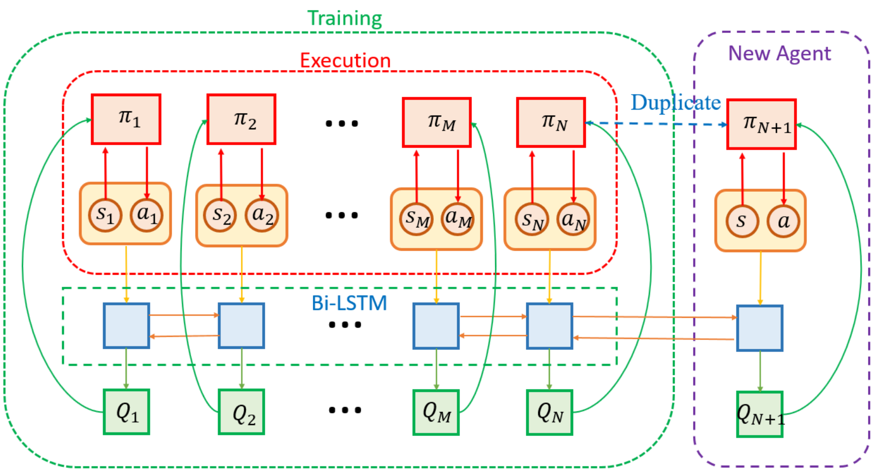

Fig. 1. Scalable Learning Framework with Centralized-training and Decentralized-execution

In our project, we would like to allow the policies to use extra information from other agents to ease training but each agent will only use their own information to execute. In addition, we use the bi-directional LSTM to deal with the cases where new agents will join the game.

In the remaining part of this section, we will discuss more about theoretical details of our framework.

2.1. Markov Environment

Different from the traditional single agent environment, in the multi-agent environment, each agent has its own observation $\mathcal{O}_i^t$ which also indicates the state of the agent s_i^t at time t. Each agent maintains one individual policy $\mu_i^{\theta_i} : \mathcal{S}_i \mapsto \mathcal{A}_i$, where each policy is parameterize by θ_i. According to the agents' policy and current states, the agent executes their own action $a_i^t = \mu_i(s_i^t)$. Then agent i will get a reward r_i^t and observe the new state s_i^{t+1} after executing the action a_i^t.

For the environment, the state of the environment consists of all the agents' states $\{s_1, s_2, \cdots, s_n\}$. The action consists of all the agents' actions $\{a_1, a_2, \cdots, a_n\}$. When considering all the agents together, the environment still preserve the Markov property, which is denoted as the Markov transition probability $p(\mathbf{s}^{t+1}, \mathbf{r}^t|\mathbf{s}^t, \mathbf{a}^t)$, where $\mathbf{s}^t = [s_1^t, s_2^t, \cdots, s_n^t]$, $\mathbf{r}^t = [r_1^t, r_2^t, \cdots, r_n^t]$ and $\mathbf{a}^t = [a_1^t, a_2^t, \cdots, a_n^t]$. Besides, an individual reward r_i, instead of one same global reward, is designed for each agent to inspire them to catch the prey actively.

2.2. Q Value Function

Based on the settings above, we can define the Q value function and state value function as that in the general reinforcement learning settings. For agent i, the state value $v_i(\mathbf{s})$ is defined as the expected discounted-cumulative rewards R_i for state $\mathbf{s}$, where $R_i = E[\sum_{t=0}^{T} \gamma^t r_i^t | \mathbf{s}]$ and γ is the discount for the reward. The Q value function is defined as $Q_i(\mathbf{s}, \mathbf{a}) = E[\sum_{t=0}^{T} \gamma^t r_i^t | \mathbf{s}, \mathbf{a}]$. The expectation is taken over the environment and the policy. Since we are using deterministic policy, the Q value function is equal to the state value function.

$$v_i(\mathbf{s}) = Q_i(\mathbf{s}, \mu(\mathbf{s})) = E[\sum_{t=0}^{T} \gamma^t r_i^t | \mathbf{s}, \mu(\mathbf{s})] \tag{1}$$

where $\mu(\mathbf{s}) = [\mu_1(s_1), \mu_2(s_2), \cdots, \mu_n(s_n)]$.

2.3. Multi-agent Actor Critic

Different from the objective function described in [7] where each agent tries to maximize its self state value function, we are trying to maximize the mean state value over all the agents in our cooperative scenario. Formally, we want to maximize

$$\frac{1}{n} \sum_{i=1}^{n} v_i(\mathbf{s}) \tag{2}$$

In our work, the critic-actor algorithm is used to learn the optimal policy of each agent and optimal Q value function by iteratively updating the actor network and critic network, where actor network refers to the policy function and critic network refers to the Q value function. At time t for agent i, we minimize the following loss function:

$$\min_{\beta} \quad (\tilde{R}_i^t - Q_i^{\beta}(\mathbf{s}_t, \mathbf{a}_t))^2 \tag{3}$$

where, $\tilde{R}_i^t$ is the estimated reward for agent i at time t with state $\mathbf{s}$ and action $\mathbf{a}$ which is the same as [8]:

$$\tilde{R}_i^t = r_i^t + \bar{Q}_i(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \tag{4}$$

where $\bar{Q}(\cdot)$ is the target Q value function with its parameters periodically updated using the recent parameters of Q function.

To optimize the policy parameters, we use the policy gradient algorithm in our work. Our objective function is defined as the mean Q value function of all the agents. The gradients for agent k with the policy parameter θ_k can be written as:

$$\nabla_{\theta_k} \frac{1}{n} \sum_{i=1}^{n} v_i(\mathbf{s}) = \int_{s'} \rho(s') \nabla_{\theta_k} \mu_k(s_k) \nabla_{a_k} \frac{1}{n} \sum_{i} Q_i(s', a')|_{a'=\mu(s')} \mathrm{d}s' \tag{5}$$

where $\mathbf{s}'$ is the state after state $\mathbf{s}$ flowing policy μ and $\rho(s')$ is the discounted probability of s' . The details of the derivation can be found in the appendix. As in [9], we also keep a replay buffer which contains experience $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})$ tuples. At each training stage, we randomly sample a batch of experiences and use them in the actor and critic algorithm.

2.4. Bi-directional LSTM

As we describe above, the inputs of the Q function for each agent are the states and actions of all the agents. As new agents are added into the environment, the dimension of the input of Q function will grow and also we need add new Q functions for the new added agents. How to address this scalability problem is important. One naive way is re-training the whole network.

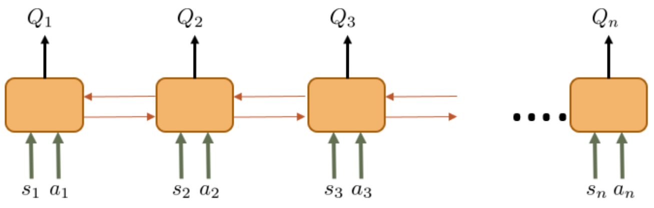

Fig. 2. LSTM structure of the Q value function

However, training the whole network is computational and time consuming. Here we resort to the LSTM network[10]. LSTM network have the properties with dynamic input and dynamic output. Fig.2 demonstrates the structure the Q value function for each agent. Each LSTM cell has the same structure and same parameters. The input of i'th LSTM cell is agent i'th state and action and the corresponding output is the Q value for agent i. With this structure, we can flexibly add new agents into our environment.

3. Scenarios

To test the performance of multi-agent RL framework, we implement our algorithm in a cooperative scenario called prey-and-predator game. In this game, several slow-moving predator are working together to chase a quick-moving prey in a limited space. Once the predators catch the prey, they will get a reward. If they run out of the limited space, they will be punished. New predators will be added into the environment while playing. We hope that the predators can catch the prey as fast as possible.

3.1. Game Setup

We construct a limited space shaped as 2×2 square for the scenario. Predators and prey will be initialized in a random position in the limited space. There also some landmarks randomly located in the space to block the move of predators and preys. Once a prey is caught or the predator runs out of the space, then the game is done and the world will be reset.

3.2. Prey

As we mainly explore the RL in a cooperative scenario, we consider the prey is part of environment. It has a fast speed and fixed policy. It can move in a random direction with a probability ϵ or move against the direction of predators' gravity with a probability $1 - \epsilon$. We also have two options to deal with the case that the prey is going to move out of the limited space. The first is that the prey will be knocked back. The second is that the space is connected and the prey will show up in a corresponding position in the other side of the space if it goes out. For convenience, we use the first option in our experiment.

3.3. Agent

In our scenario, agents are the predators. They have a slower speed but they can cooperate with each other to catch the prey. We define the state of each agent as its partial observation of environment, which denotes as s_i. Based on the state information s_i, each agent has its own policy μ_i. According to the policy $\mu_i(a_i|s_i)$, the agent takes actions a_i and get their reward r_i at each step.

- **State** s_i : The partial observations of each agent includes its own position $P_i = (x_i, y_i)$, the prey's position $P_p = (x_p, y_p)$, the angle θ_i of the direction between agent i and the prey, and the position of some other agents $\{P_m, ...P_n\}$. Then $s_i = \{P_i, P_p, \theta_i, P_m, ...P_n\}$.
- **Action** a_i : The action of each agent is the direction where the agent should go. It's a angle range in $[-\pi, \pi]$ which will translated into a five dimension vector later.

- **Reward** r_i : In our framework, each agent can have their own reward function. We define r_i as:

$$r_i = \begin{cases} -0.1 \times \sqrt{(x_i - x_p)^2 + (y_i - y_p)^2} & \textit{not caught} \\ individual\ Bonus(i) + Global\ Bonus & \textit{caught by agent } i \\ Global\ Bonus & \textit{caught by other agent} \\ -10 & \textit{agents go out of space} \end{cases} \tag{6}$$

where the agent's $Individual\ Bonus(i)$ can be different for different agent i.

4. Simulation Results

To test the performance of our algorithm, we first implement it in the condition where three predators work together to catch the prey.

The mean Q values is shown in Fig.3. We can see that the Q value increases and then converge in our algorithm. When replaying the the policy learned by our algorithm, we can see that the predators will move in different actions based on the reward design to catch preys

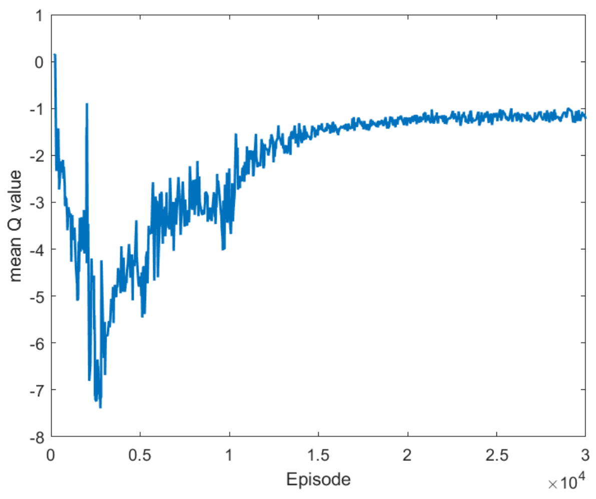

Fig. 3. Mean Q Value

We also check the rates of catching the preys. We sum the number of successful catching in each 100 rounds and then calculate the rate. As we can see from the left picture of Fig.4, the rate increase with the process of our algorithm. And it will finally converge approximately to 1. The result performs well as our expectation in our algorithm.

More, we also simulate the steps to catch the prey in each time. As it shows in the right of Fig.4, we can see that the steps spent to catch the prey decreases with playing more rounds. The predators learn to catch the preys quicker and quicker

To test the scalability of our algorithm, we set the experiment as following: (1) Three Predators join the game in the beginning to chase the prey; (2) Then three more predators join the game after 30000 episodes. We compare our results with totally retraining the network. As we can see in Fig.5, the Q value increases after three new predators join the game after 30000 steps. In addition, our algorithm converges faster and has a higher Q value than retraining the network. As our reward is designed as negative distance between the predators and agents, it means that the performance is better if the Q value is larger. From the simulation results, we can see that our algorithm has a good scalability

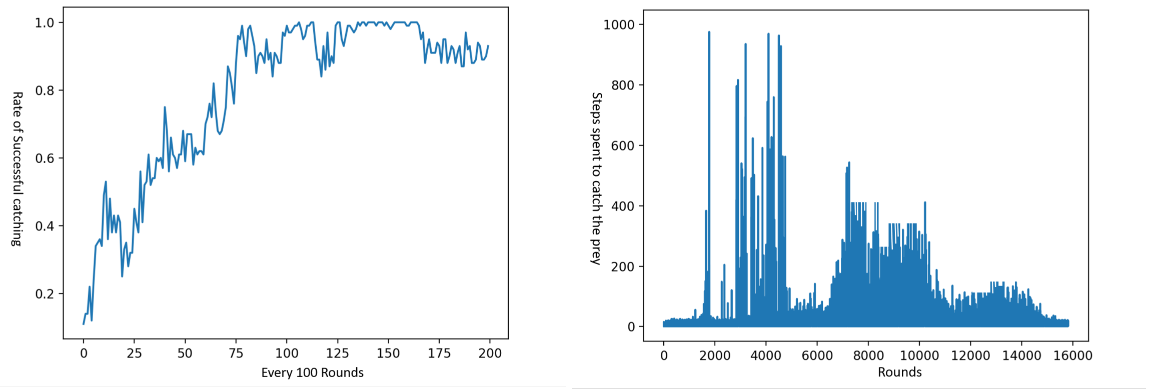

Fig. 4. The predators work together to catch one prey

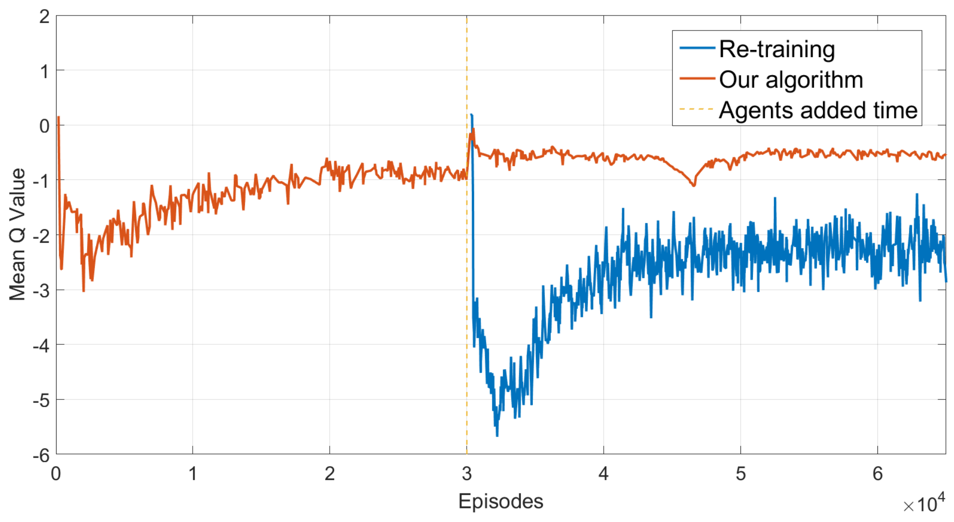

Fig. 5. Three agents play from the beginning and three new agents join the game at 30000 steps

5. Conclusions

To improve the scalability, flexibility and robustness of reinforcement learning in the multi-agent conditions, we proposed a scalable framework with centralized training and decentralized execution based on bi-directional LSTM. In our framework, we can design different reward functions for each agent and each agent can have their own policy. This increases the flexibility in different multi-agent problems, no matter in cooperative cases or competitive cases. From the results shown in the simulation part, we can see the cooperative actions and a high rate of finishing the objective tasks in our framework. The framework is suitable in multi-agents problems. In addition, with the use of bi-directional LSTM, we can obtain a high scalability in our algorithm. Our algorithm can converge faster when the new agents are introduced into the play.

Although we mainly focus on the cooperative scenarios in our project, it's also possible to implement our algorithm in a mixed scenarios with both cooperation and competition because we can design different rewards for different agents in our framework. Still in the predator-and-prey games, we can also consider prey is also an agent but it has a quite opposite goal and reward functions. As the time is limited, we will leave the mixed cases in the future work.

References

[1] S. Wender and I. Watson, “Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar,” in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 402–408.

[2] M. J. Matarić, “Reinforcement learning in the multi-robot domain,” *Autonomous Robots*, vol. 4, no. 1, pp. 73–83, 1997.

[3] A. Schaerf, Y. Shoham, and M. Tennenholtz, “Adaptive load balancing: A study in multi-agent learning,” *Journal of artificial intelligence research*, vol. 2, pp. 475–500, 1995.

[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[5] A. Servin and D. Kudenko, “Multi-agent reinforcement learning for intrusion detection,” *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pp. 211–223, 2008.

[6] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” *arXiv preprint arXiv:1706.02275*, 2017.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[8] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 387–395.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

Appendix

1) **The derivative of the policy gradient:**
For notation convenience, here our objective function is maximizing the total value function of all the agents. The gradient for agent k with the policy parameter θ_k is:

$$
\begin{aligned}
&\nabla_{\theta_k} \sum_{i=1}^{n} v_i(\mathbf{s}) \\
&= \nabla_{\theta_k} \sum_i Q_i(\mathbf{s}, \mu^\theta(\mathbf{s})) \\
&= \nabla_{\theta_k} \int_{\mathbf{S}'\mathbf{r}} [\sum_i [r_i + v_i(\mathbf{s}')] p(r_1, r_2, \cdots, r_n, |\mathbf{s}, \mu^\theta(\mathbf{s})] \,\mathrm{d}s' \mathrm{d}\mathbf{r} \\
&= \nabla_{\theta_k} \int_{\mathbf{r}} \sum_i r_i p(r_i|\mathbf{s}, \mu^\theta(\mathbf{s}))\, d\mathbf{r} + \nabla_{\theta_k} \int_{s'} \sum_i v_i(s') p(\mathbf{s}'|\mathbf{s}, \mu^\theta(\mathbf{s})) \mathrm{d}s' \\
&= \int_r \sum_i r_i \nabla_{a_k} p(r_i|s,a) \nabla_{\theta_k} \mu^k(s_k)|_{a=\mu^\theta(s)} dr + \int_{s'} \sum_i \nabla_{a_k} p(s'|s,a) \nabla_{\theta_k} \mu^k(s_k)|_{a=\mu(s)} v_i(s') ds' \\
&+ \int_{s'} \sum_i p(s'|s,a) \nabla_{\theta_k} v_i(s') ds' \\
&= \nabla_{\theta_k} \mu^k(s_k)\, \nabla_{a_k} \left[\int \sum_i r_i + v_i(s') p(s', r|s,a) dr ds \right] |_{a=\mu(s)} + \int_{s'} \sum_i p(s'|s,a) \nabla_{\theta_k} v_i(s') ds' \\
&= \nabla_{\theta_k} \mu^k(s_k) \nabla_{a_k} \sum_i Q_i(s,a)|_{a=\mu(s)} + \int_{s'} p(s'|s,a) \nabla_{\theta_k} \sum_i v_i(s') ds' \\
&= \nabla_{\theta_k} \mu^k(s_k) \nabla_{a_k} \sum_i Q_i(s,a)|'_{a=\mu(s)} \\
&+ \int_{s'} p(s'|s,a) \left[\nabla_{\theta_k} \mu^k(s_k) \nabla_{a_k} \sum_i Q_i(s') \right] ds' \\
&+ \int_{s'} p(s'|s,\mu(s)) \int_{s''} p(s''|s', \mu(s')) \nabla_{\theta_k} \sum_i v_i(s'') ds'' ds' \\
&\cdots \\
&= \int_{s'} \sum_{t=0}^{\infty} p(s \to s', t) \nabla_{\theta_k} \mu^k(a_k) \nabla_{a_k} \sum_i Q_i(s', a')|_{a'=\mu(s')} ds' \\
&= \int_{s'} \rho(s') \nabla_{\theta_k} \mu_k(s_k) \nabla_{a_k} \sum_i Q_i(s', a')|_{a'=\mu(s')} ds'
\end{aligned}
$$

2) **Code of our algorithm:**
The codes of our algorithm can be found on github:
Code Link: $https://github.com/livey/scalable_maddpg$