STRUCTURAL MULTI-AGENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we propose a multi-agent learning framework to model communication in complex multi-agent systems. Most existing multi-agent reinforcement learning methods require agents to exchange information with the environment or global manager to achieve effective and efficient interaction. We model the multi-agent system with an online adaptive graph where all agents communicate with each other through the edges. We update the graph network with a relation system which takes the current graph network and the hidden variable of agents as input. Messages and rewards are shared through the graph network. Finally, we optimize the whole system via the policy gradient algorithm. Experimental results of several multi-agent systems show the efficiency of the proposed method and its strength compared to existing methods in cooperative scenarios.

1 INTRODUCTION

Reinforcement learning(Shu & Tian, 2018) has been making significant progress in many applications, including playing video games(Mnih et al., 2013; Silver et al., 2016), text generation(Guo; Li et al., 2016), and robotic controls(Tedrake et al., 2004; Levine et al., 2016). These signs of improvement in single-agent learning have enabled agents to deal with tasks in high dimensional spaces. Under this circumstance, agents can fulfill their goals by modeling the environment as a whole(Doya et al., 2002). However, numerous applications are concerned with the interaction of multiple agents(Matignon et al., 2012; Mordatch & Abbeel, 2018; Leibo et al., 2017). The relationships between agents are complicated as the cooperation(Peng et al., 2017; Olfati-Saber et al., 2007) and competition(Zheng et al., 2018) with each other bring uncertainty and complexity. The ideas of deep reinforcement learning have been explored for multi-agent systems(Lowe et al., 2017). Unfortunately, conventional reinforcement learning approaches such as Q-Learning or policy gradient with independently learning agents are poorly suited to multi-agent environments(Matignon et al., 2012).

Most existing multi-agent learning methods focus on collaboration among learning agents. In some multi-agent systems, success has been made by introducing a centralized controller(Shu & Tian, 2018; Hong et al., 2018). When the agent number increases, researchers reduce computational complexity by approximating the average effect from the overall population or neighboring agents(Yang et al., 2018). Others develop methods that enable agents to learn to coordinate on their own(Morcos et al., 2018; Sukhbaatar et al., 2017; Perolat et al., 2017). These methods, however, fail to characterize systems of diverse agents as a whole and can hardly reveal the relationship between agents without any prior information. Besides such challenges, a general dilemma lies in that none of the existing algorithms deals with indefinite numbers of agents, which is a typical attribute of most dynamic systems.

In this work, we propose a general multi-agent learning structure(SMAL) to address these challenges. As shown in Figure 1, we tackle multi-agent reinforcement learning under a setting where each agent is directly interacting with a finite set of other agents. Through a chain of direct interactions, we interconnect any pair of agents globally. Unlike the agents that only have access to local information, we update their policies by considering communications with each other. Each agent has a latent variable that stands for its role in the whole system. The hidden variables of every pair of agents model the online adaptive graph network that stores the relations.

The online graph explicitly demonstrates the connections between agents by the weights of edges and informs agents with whom they should collaborate. Edges of positive weight indicate that the

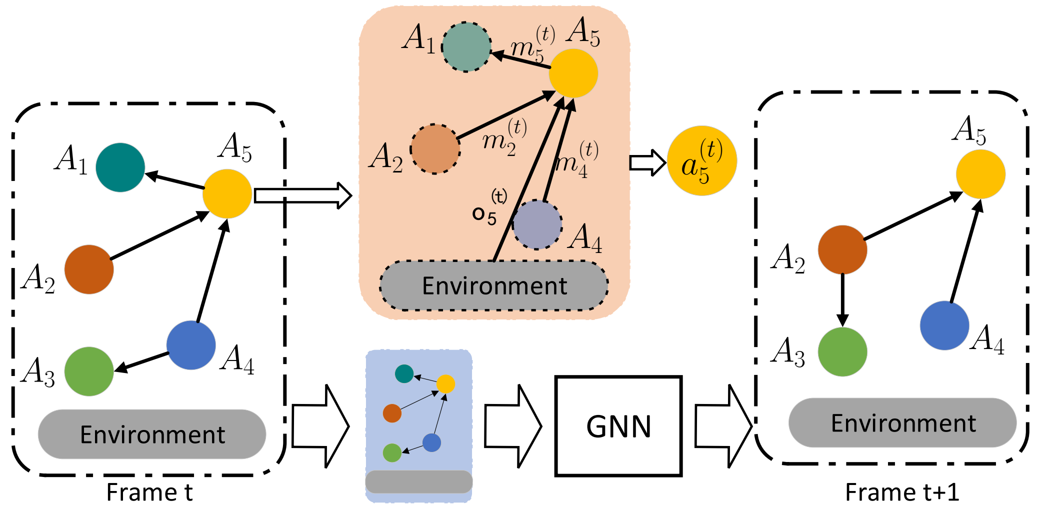


Figure 1: The main framework of the proposed Structural Multi-Agent Learning. Given the states of the multi-agent system at frame t, all agents communicate with each other through the edges rather than upload messages to the environment or global manager to achieve effective and efficient interaction. Then each agent makes a decision according to the current state and received messages. Finally, we update the graph network with a relation system which takes the current graph network and the hidden variables as inputs.

two agents should work together while a negative one suggests that they are competitors. As we use a relation network to infer edges between agents, the graph is adaptive even if the agent number fluctuates. Agents can only communicate with those adjacent on the graph, and their messages are propagated to connected agents. For highly complex systems with numerous agents, we reduce the complexity of the graph network by limiting global communication and control cost. Finally, We empirically show the success of our approach compared to existing methods in various cooperative scenarios, where agents can discover optimal coordination strategies.

The main contributions of this paper are:

(1)We propose a general framework for multi-agent learning that models the relationships between agents with a graph network.

(2)The messages and rewards are shared through the graph to achieve efficient cooperation.

2 Related Work

Multi-agent reinforcement learning. Self-agent reinforcement learning algorithms have achieved significant success in several applications. However, it's difficult to adapt these methods to multi-agent systems(Matignon et al., 2012). As the environments are non-stationary, independent-learning policy gradient methods perform poorly(Lowe et al., 2017). Under cooperative settings, a centralized critic is proved effective(Shu & Tian, 2018; Hong et al., 2018). Other methods encourage cooperation via the sharing of policy parameters(Gupta et al., 2017). The graph in our work is a variant of a centralized controller for the messages between agents. The difference lies in that the policies of agents are learned separately.

Communication in multi-agent systems. Reinforcement learning provides a way to study how communication emerges in multi-agent systems. Pioneering work by Jakob(Foerster et al., 2016) is considered as a first attempt at learning communication and language with deep learning approaches. Under cooperative settings, some communication architectures(Cao et al., 2018; Das et al., 2018; Kluge, 1983; Singh et al., 2018; Sukhbaatar et al., 2016) allow agents to achieve targeted and profitable communication. Several approaches (Choi et al., 2018; Lazaridou et al., 2018; Bogin et al., 2018) train agents to learn from raw pixel data and communicate through discrete symbols. Social influences can also be interpreted as a kind of communication(Jaques et al., 2019). Other frameworks use early and curiosity-driven developments to build a unified model of speech and tool(Oudeyer & Smith, 2016; Foerster et al., 2016). Unlike prior approaches for targeted communications, we propose a structural communication system for agents to exchange messages with their targets through a graph.

Relational forward models. Relational reasoning has received considerable attention in recent years, and researchers attempt to predict the forward dynamics of multi-agent systems with a rich relational structure(Zheng et al., 2016). Deep learning models that operate on the graph have been developed to provide insights into the relational structures behind the data. Relational forward model(Tacchetti et al., 2018) based on graph network(Battaglia et al., 2018) embeds RFM modules inside agents and updates the graph network with supervised learning. Their model takes the state of the environment as input and outputs either an action prediction for each agent or a prediction of the cumulative reward each agent will receive. The graph in our framework can be defined as a relational forward model and is updated by policy gradient flows.

3 BACKGROUND

Multi-agent Markov games. In this paper, we consider multi-agent partially observable Markov games(Littman, 1994). Under multi-agent settings, a Markov game can be defined by $\langle S, \mathcal{T}, A, \gamma \rangle$. Given environment state $s_t \in S$, each agent chooses an action $a_t \in A$. The actions of all agents are combined as a joint action $\boldsymbol{a}_t = \left[a_t^0, \dots a_t^N\right]$, producting the next state s_{t+1} according to the transition function $\mathcal{T}\left(s_{t+1} | \boldsymbol{a}_t, s_t\right)$. Each agent also receives a reward $r(s_t, a_t)$ and the target is to maximize its accumulated expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t(s_t, a_t)$, where γ is the discount factor and T is the time horizon. As the environment is partially observable, every agent chooses its action based on observation o_t rather than s_t.

Policy gradient algorithms. Policy gradient methods(Sutton et al., 2000) directly update the parameters of policies by computing the gradients of value functions. Given the objective: $J(\theta) = \mathbf{E}_{\tau \sim p_\theta(\tau)}[r(\tau)]$, where r and τ refer to rewards and trajectories, the gradient of the policiy can be written as:

$$\nabla_\theta J(\theta) = \mathbf{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(\mathbf{a}_t | \mathbf{s}_t\right) \right) \left(\sum_{t=1}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right) \right] \tag{1}$$

Here $\pi_\theta(a_t|s_t)$ is the policy, s_t and a_t are states and actions. The advantage of equation 1 is that we don't need any prior knowledge about initial distribution or environment. However, the gradient estimates are of high variance. Under multi-agent settings, an agents reward usually depends on the actions of other agents, which leads to more fluctuations in the estimation process.

Graph neural network. Graph neural networks (GNN)(Defferrard et al., 2016) extended existing neural networks for processing the data represented in graph domains. GNNs can be categorized as spectral approaches and non-spectral approaches. Non-spectral methods define convolutions directly on the graph, operating on spatially close neighbors. Spectral approaches work with a spectral representation of the graphs. The convolution operation is defined in the Fourier domain by computing the decomposition of the graph Laplacian(Zhou et al., 2018). The operation can be defined as the multiplication of a signal $x \in \mathbb{R}^N$ (a scalar for each node) with a filter $g_\theta = diag(\theta)$:

$$g_\theta \star x = U g_\theta(\Lambda) U^T x, \tag{2}$$

where U is the matrix of eigenvectors of the normalized graph Laplacian L(Chung & Graham, 1997).

4 APPROACH

In this section, we will detail the proposed structural multi-agent learning method. We first define the structure of a multi-agent system with the graph network and the interactions between agents. Then we provide the optimization process of the whole framework.

4.1 Structural multi-agent modeling

Given a multi-agent system $\{\mathcal{A}, \mathcal{G}\}$ with N agents, where A_i is the i-th agent, and $\mathcal{G} = \{v(A_i, A_j) \in [-1, 1]\}$ is the graph that represents the relations between agents(Note that A and a refers to agents and actions respectively). Positive, negative and zero weight of the graph $v_{ij} = v(A_i, A_j)$ means A_i, A_j are cooperative, competitive and independent.

Each agent can communicate with the agents adjacent on the graph directly, and the message of itself can propagate through the edges to connected agents. Thanks to the graph neural network, the agent A_i can get the encoded messengers at $(t - d + 1)$-th frame of neighbors whose order is d. Specifically, when $d = 1$, it means that agents can receive instant messages from adjacent neighbors, while messages from farther agents are propagated through frames.

4.1.1 Cooperative

First, we talk about cooperative settings. In this case, the edges of the graph $v_{ij} \in [0, 1]$. For a single agent A_i, it takes its observation, its hidden variable and messages from other agents as inputs to generate action according to policy π. We also exploit a model f to predict its hidden variable and messages to be sent to other agents. For different agents, the dimensions of messages must be the same to communicate with each other. We have:

$$a_t \sim \pi(o_t, h_t, m_{-t}) \tag{3}$$

$$\{h_{t+1}, m_{t+1}\} = f(h_t, m_t, o_t, a_t) \tag{4}$$

$$m_{-t} = \sum_{j \neq i}^{N} v_{ij} m_{jt} \tag{5}$$

Here o_t is the observation, h_t is the hidden variable, m_t is the message sent to other agents and a_t is the action. In equation 5, m_{jt} is the message given by agent j at time t, so m_{-t} is treated as a sum of reweighted messages from other agents. We omit the subscript i for clarity. For the graph $\mathcal{G} = \{v(A_i, A_j)\}$, we use a relation network ϕ to infer the edge between each pair of agents, which is defined as follows:

$$\mathcal{G}_{t+1} \sim v_{ij} = \phi(h_{it}, h_{jt}, \psi(\mathcal{G}_t)) \tag{6}$$

Where ψ is a graph neural network which encodes the current graph $\mathcal{G}_t$ into vectors as the global representation of current multi-agent system.

To adjust the whole graph network needs to traverse all possible active edges of the current multi-agent system, which might cost lots of computation when the multi-agent system scale is relatively large. Since the relationship between agents might not change frequently, we can update the graph network interval T frame or when the number of agents changes to reduce the cost of graph network adjustment.

4.1.2 Competitive

Under competitive settings, the edges of the graph $v_{ij} \in [-1, 1]$. We can change equation 5 as follows:

$$m_{-t} = \sum_{j \neq i}^{N} max(v_{ij}, 0) m_{jt} \tag{7}$$

In this way, the messages from partners are considered valuable, while messages from competitors are ignored. As competitive environments are totally different from cooperative scenarios, we mainly focus on cooperative settings in our experiments and leave further discussions of competitive settings to Appendix.

4.2 Optimization

Different from traditional multi-agent learning methods, we consider the actions of the single agent and the whole interaction network at the same time. The whole formulation of the proposed method contains two parts:

Algorithm 1: SMAL

```
Input: Training environment $E$, parameters: $\lambda, T$, learning rate $\rho$, total iterative number $\Gamma$, and
       convergence error $\varepsilon$.
Output: Parameters: $\theta, \psi, \phi$.
Initialize $\theta, \psi, \phi$;
for $t = 1, 2, \cdots, \Gamma$ do
    for Every agent $A_i$ do
        Receive observations $o_t$ from $E$;
        Generate message $m_t$ and send it to its neighbor;
        Receive messages $m_{-t}$ from its neighbor;
        Generate action $a_t$;
        Interactive with $E$ according to $a_t$ and obtain $o_{t+1}$;
    end
    if MOD(t,T)= 0 then
        for $A_i, A_j \in \mathcal{S}$ do
            Calculate $v_{ij}$ according to (6);
        end
        Update current graph $\mathcal{G}$;
    end
    Compute $J_t$ using (8);
    if $t \geqslant 1$ and $|J_t - J_{t-1}| < \varepsilon$ then
        go to Return.
    end
    Update $\theta, \psi, \phi$ according to (12);
end
Return: $\theta, \psi, \phi$.
```

$$J = J_1 + \lambda J_2 \tag{8}$$

Where J_1 is the individual loss and J_2 is the structural loss. We use λ to balance those two part. The individual loss J_1 is defined by the rewards of a single agent. In a multi-agent system, the purpose of each agent is to enlarge the reward of itself and its collaborators. So the policy gradient part need to consider the reward of its neighbor. We define the individual loss J_1 for every agent as following:

$$J_1 = -E\left[\pi(a_t|o_t, h_t, m_{-t})(R(s_t, a_t) + R_{-t})\right] \tag{9}$$

$$R_{-t} = \frac{1}{1 + \sum_{j\neq i}^{N} v_{ij}} \sum_{j\neq i}^{N} v_{ij} R(s_{jt}, a_{jt}) \tag{10}$$

Where $R(s_t, a_t)$ is the reward for a single agent, and R_{-t} is the shared reward from adjacent agents. The structural loss $J_2 = E\left[L(\mathcal{G}_{t+1})\right]$ of our method aims to minimize the global communication and control cost. In our method $L(\mathcal{G}_{t+1})$ is defined as following:

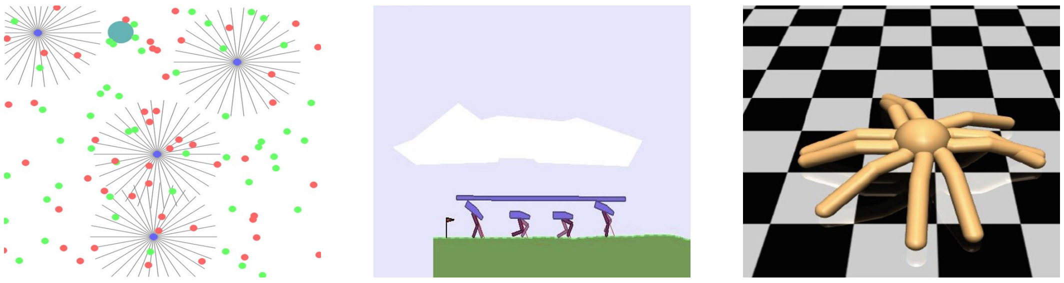
$$L(\mathcal{G}_{t+1}) = \sum_{i,j} \mathbb{I}_{v\neq 0}(v_{ij}) \tag{11}$$

Where $\mathbb{I}(\cdot)$ is the indicator function. With this loss, we can control the sparseness of the graph. In large scale systems, we want to reduce unnecessary communications between agents.

Finally, we use the stochastic sub-gradient descent algorithm to update those neural networks with learning rate ρ. Here θ refers to the parameters for policy π and predictor f of every agent.

$$\theta = \theta - \rho\frac{\partial J}{\partial \theta}, \phi = \phi - \rho\frac{\partial J}{\partial \phi}, \psi = \psi - \rho\frac{\partial J}{\partial \psi}. \tag{12}$$

Algorithm 1 summarizes the detailed procedure of our proposed SMAL method.



(a) Waterworld (b) Multi-Walker (c) Multi-Ant

Figure 2: Illustrations of the experimental environment and continuous tasks we consider. Agent numbers of the three environments above are respectively 4, 4 and 10.

5 Experiments

5.1 Environments

We conducted experiments on three multi-agent domains: Waterworld, Multi-Walker, and Multi-Ant(Gupta et al., 2017). All the three tasks are continuous control problems with partial observation in which agents should cooperate to fulfill a certain goal. We provide details of each environment below.

Waterworld. Waterworld is a type of pursuit problem in the continuous domain. The environment is based on the single-agent Waterworld domain used by (Ho et al., 2016). In this task, agents need to cooperate to capture moving food targets while avoiding poison targets. The agents move around by applying a two-dimensional force. They will receive a positive reward for achieving a food target and a negative reward for capturing a poisoned target. The observation and action spaces are all continuous. Figure 2a shows an example of four agents moving for food.

Multi-Walker. Multi-Walker is a control locomotion task based on the BipedalWalker environment from OpenAI gym(Brockman et al., 2016). The environment contains multiple bipedal walkers that can actuate the joints in each of their legs. At the start of each simulation, a large package is placed on top of the walkers. The walkers must learn how to move forward and to coordinate with other agents to keep the bag balanced while navigating complex terrain. Each agent receives a positive reward for carrying the package forward, a negative reward for falling and dropping the package. Figure 2b shows the scene of four walkers.

Multi-Ant. The multi-ant domain is a 3D locomotion task based on the robot models in (Schulman et al., 2015b). In this domain, each leg of the ant is a separate agent. Legs can sense their positions and velocities as well as those of their neighbors. Each leg is controlled by applying torque to its two joints. The legs should cooperate with neighbors to help the ant move forward as quickly as possible. The robot in Figure 2c is an ant with ten legs.

5.2 Results and analysis

Comparison with state-of-the-art methods. For comparison, we choose four baselines: Multi-agent Deep Deterministic Policy Gradient(MADDPG)(Lowe et al., 2017), Centralized MADDPG(C-MADDPG), Individualized Controlled Continuous Communication Model(IC3Net)(Singh et al., 2018) and Trust Region Policy Optimization(TRPO) (Schulman et al., 2015a) to compare with our method: Structural Multi-agent learning(SMAL). Among them, MADDPG learns coordinated behavior via centralized critics and uncentralized executors, while Centralized MADDPG exploits joint observation and action to train a centralized critic and a centralized actor. IC3Net is an efficient method that learns profitable communication by controlling continuous communication with a gating mechanism. TRPO is an effective single-agent reinforcement learning methods. We list the details for the implementation and comparison in the Appendix.

We plot the learning curves for various approaches in Figure 3. The curves show that our method (SMAL) can reach high rewards in early episodes and the optimal policies perform better than the

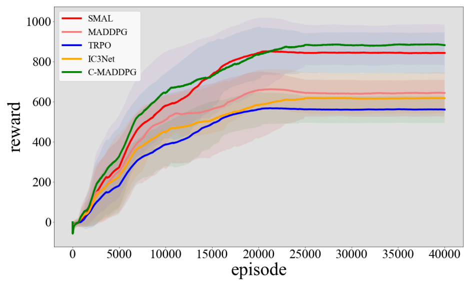


Figure 3: Total agent rewards on Waterworld

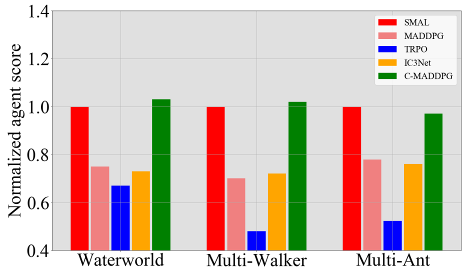


Figure 4: Comparison of normalized agent score

baselines. We observed that the agents trained by our method tend to keep close with other agents near them more frequently. In the Multi-Walker domain, the walkers learn to carry the package forward without letting it fall by the messages from the other three walkers. In the Multi-ant case, the legs learn to avoid collision with each other. The histogram in Figure 4 is the normalized agent reward(divided by the return of SMAL) in all three domains, which shows that our method can be generalized to these systems. Our method surpasses MADDPG in all three environments, indicating that the graph network is a better supervisor than the centralized critic. Also, out method outperforms IC3Net, which focuses on efficient communication and doesn't include a reward sharing mechanism. Compared to Centralized MADDPG, which trains the whole system jointly and optimizes the total reward directly, our method achieves a near-optimal result.

Ablation studies. To verify the effectiveness of sharing messages and rewards in our method, we create two baselines that only allow agents to share either messages or rewards. To clarify how messages and shared rewards help with the collaboration between agents, we evaluate their effectiveness in the three domains.

Method		Environments(Total Rewards)		
Shared Messages	Shared Rewards	Waterworld	Multi-walker	Multi-ant
×	×	472.33	30.64	67.29
✓	×	660.10	52.82	106.21
×	✓	625.59	48.76	90.80
✓	✓	**812.45**	**70.68**	**126.11**

Table 1: Ablation studies for messages and shared rewards

Table 1 shows how shared messages and rewards lead agents to cooperate with others. The model with only messages is made by removing the second term of equation 10. Meanwhile, the model that only shares rewards is by setting messages as zero. When agents aren't allowed to share messages and rewards, SMAL degenerates to a self-agent reinforcement learning method. The results demonstrate that the transmission of signals and rewards are both necessary for agents to achieve thorough consociation. These results further show that if rewards of other agents are taken into consideration, policy gradient methods can learn suboptimal policies for overall returns.

Trade-off between communication costs and Performance. To investigate the loss defined in equation 8($J = J_1 + \lambda J_2$), we alter the parameter λ to adjust the sparseness of the graph. When $\lambda = 0$, the graph learned by SMAL is fully connected regardless of the communication cost. When we increase λ, we limit the communication between agents. All the experiments above are carried out with $\lambda = 0$ to achieve complete cooperation.

Figure 5 reveals the relationship between communication costs and agents' performance. When communication is limited, agents that should be collaborators are forced to be independent. Losing potential opportunities for cooperation leads to less total rewards. How to achieve best cooperation with limited communication is an interesting problem for future work.

Effect of the graph. To further analyze the effect of the graph in our SMAL method, we visualize how graph changes in a certain episode. We use $\lambda = 1$ to ensure some agents are independent.

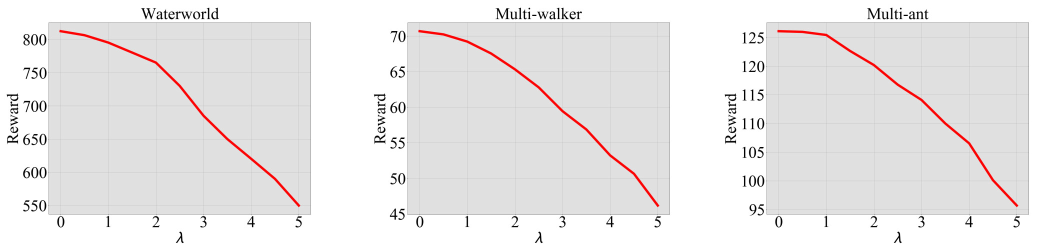


Figure 5: Trade-off between communication costs and performance, the number of edges decreases when λ increases. We choose ten values of λ for every environment.

Figure 6 demonstrates the transformation of the graph inferred during one episode on Waterworld with four agents. Wider edges in the plot refer to larger weights of messages between the agents. If there's no edge between two vertices, the edge inferred is zero. The above row shows the real situation of agents in Waterworld. We examine the case in the middle of Figure 6.

As agent 1 and 4 are around agent 2, they are more likely to cooperate to capture the food targets. In contrary, agent 3 is far away from other agents, so it keeps a distant connection. The graph we infer is consistent with this situation as the edges between agent 1,2,4 are wider, while the edges between agent 3 and other agents are narrower. The experimental results point out that introducing a graph network to manage the message flows and share rewards leads to better cooperation.

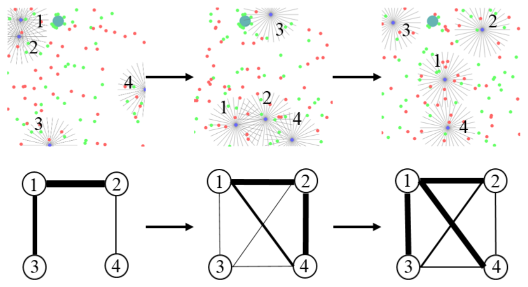


Figure 6: Graph inference

Table 2: Ablation study of graph

Method	Total Reward($\lambda = 0$)		
Environment	Waterworld	Multi-walker	Multi-ant
Graph	812.45	70.68	126.11
Mean	636.92	59.34	115.70

Quantitive improvements contributed to the graph are shown in table 2. In the second experiment, all the weights in the graph are set as one. Agents directly receive the messages from other agents and take their mean vector as the input message. With the graph to model the proportion of messages and rewards, the agents choose their actions more intelligently.

6 Conclusions and Future Work

In this paper, we have proposed a structural multi-agent learning framework. We tackle multi-agent reinforcement learning under a setting where each agent is directly interacting with a finite set of other agents. Through a chain of direct interactions, any pair of agents are interconnected globally. The messages and rewards are shared through the global graph. Experimental results on several multi-agent systems show the efficiency of the proposed method and its strength compared to existing methods in cooperative scenarios. We also show that messages and rewards are both necessary for collaboration.

For competitive settings, agents should learn to generate unique messages to inform their partners or cheat their opponents. Also, when the agent number dynamically changes, the initialization or removal of agents remains investigation. For many-agent systems, the stability and computational complexity should be taken into consideration. We leave these investigations to future work. How to apply the proposed method to real group analysis, like multi-object tracking, group action recognition, and autonomous driving, is an exciting future research direction.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NeurIPS*, pp. 5048–5058, 2017.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Ben Bogin, Mor Geva, and Jonathan Berant. Emergence of communication in an interactive world with consistent speakers. *arXiv preprint arXiv:1809.00549*, 2018.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. *arXiv preprint arXiv:1804.03980*, 2018.

Edward Choi, Angeliki Lazaridou, and Nando de Freitas. Compositional obverter communication learning from raw visual input. *arXiv preprint arXiv:1804.02341*, 2018.

Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Michael Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. *arXiv preprint arXiv:1810.11187*, 2018.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pp. 3844–3852, 2016.

Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369, 2002.

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *NeurIPS*, pp. 2137–2145, 2016.

H Guo. Generating text with deep reinforcement learning, arxiv (2015).

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *AAMAS*, pp. 66–83. Springer, 2017.

Jonathan Ho, Jayesh Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization. In *ICML*, pp. 2760–2769, 2016.

Zhang-Wei Hong, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, and Chun-Yi Lee. A deep policy inference q-network for multi-agent systems. In *AAMAS*, pp. 1388–1396. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, Dj Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pp. 3040–3049, 2019.

Werner E. Kluge. Cooperating reduction machines. *IEEE Trans. Comput.;(United States)*, 11, 1983.

Angeliki Lazaridou, Karl Moritz Hermann, Karl Tuyls, and Stephen Clark. Emergence of linguistic communication from referential games with symbolic and pixel input. *arXiv preprint arXiv:1804.03984*, 2018.

Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *AAMAS*, pp. 464–473. IFAAMAS, 2017.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(1):1334–1373, 2016.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pp. 157–163. Elsevier, 1994.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS*, pp. 6379–6390, 2017.

Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *KER*, 27(1):1–31, 2012.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. *arXiv preprint arXiv:1803.06959*, 2018.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *AAAI*, 2018.

Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

Pierre-Yves Oudeyer and Linda B Smith. How evolution may work through curiosity-driven developmental process. *Topics in Cognitive Science*, 8(2):492–502, 2016.

Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2, 2017.

Julien Perolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In *NeurIPS*, pp. 3643–3652, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pp. 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

Tianmin Shu and Yuandong Tian. Mˆ 3rl: Mind-aware multi-agent management reinforcement learning. *arXiv preprint arXiv:1810.00147*, 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *arXiv preprint arXiv:1812.09755*, 2018.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pp. 2244–2252, 2016.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, pp. 1057–1063, 2000.

Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*, 2018.

Russ Tedrake, Teresa Weirui Zhang, and H Sebastian Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *RSJ*, volume 3, pp. 2849–2854. IEEE, 2004.

Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.

Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *AAAI*, 2018.

Stephan Zheng, Yisong Yue, and Jennifer Hobbs. Generating long-term trajectories using deep hierarchical networks. In *NeurIPS*, pp. 1543–1551, 2016.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

A Appendix

A.1 Competitive settings

In this section, we provide an extension of the competitive situation we have talked about in section 4.1.2.

Under competitive settings, the edges of the graph $v_{ij} \in [-1, 1]$. The target of every agent is to maximize the total rewards of their collaborators and minimize the rewards of their competitors. We can change equation 5 as follows:

$$m_{-t} = \sum_{j \neq i}^{N} max(v_{ij}, 0) m_{jt} \tag{13}$$

The equation indicates that messages from partners are considered valuable, while messages from competitors are ignored. (Singh et al., 2018) also demonstrates that agents will stop communication under the competitive setting. Also, equation 10 should be altered:

$$R_{-t} = \frac{1}{1 + \sum_{j \neq i}^{N} |v_{ij}|} \sum_{j \neq i}^{N} v_{ij} R(s_{jt}, a_{jt}) \tag{14}$$

When v_{ij} is negative, the equation is still a reweighted combination of rewards of other agents. The revised policy gradient method aims to control the rewards of competitors and reinforce the rewards of cooperators.

However, messages in competitive settings can be more flexible. In cooperative settings, agents are sending the same messages to others. When competitors exist, agents should learn how to give out information selectively. Agents can even determine how to lie to competitors. How to generate signals in competitive scenarios is an interesting future direction.

A.2 Implementation Details

We implement our structural multi-agent reinforcement learning algorithm and test its efficiency on the environments presented in Section 5.1. The agent numbers in three systems are all four, and the max steps for each episode are 500. We choose ten random seeds for every environment.

Our policies and graph inference network are both parameterized by a two-layer ReLU MLP with 128 units per layer. In cooperative environments, the output of the graph inference network is limited to [0,1] with a sigmoid function. In all of our experiments, we use the Adam optimizer with a learning rate of 0.001 to optimize the policy network and a learning rate of 0.01 to optimize the graph network.

The latent variables and messages are initialized randomly with a length of 10. The edges inferred from the network serve as the weights to calculate a weighted average of messages for a single agent. We exploit a replay buffer Andrychowicz et al. (2017) of size 500000 and the batch size is 250. As the graph network doesn't need to be changed frequently, we update it every 10000 steps.

For comparison, the hyperparameters for all four algorithms are the same, except for TRPO we exploit a batch size of 100, which leads to better performance. For IC3Net, we trained with the setting the authors exploited in the StarCraft task (Singh et al., 2018).