Implicit Cooperative Learning on Distribution of Received Reward in Multi-Agent System

Fumito Uwano[a]
Department of Computer Science, Okayama University, 3-1-1, Tsushima-naka, Kita-ku, Okayama, Japan

Keywords: Multiagent System, Reinforcement Learning, Neural Network, Implicit Learning, Normal Distribution.

Abstract: Multi-agent reinforcement learning (MARL) makes agents cooperate with each other by reinforcement learning to achieve collective action. Generally, MARL enables agents to predict the unknown factor of other agents in reward function to achieve obtaining maximize reward cooperatively, then it is important to diminish the complexity of communication or observation between agents to achieve the cooperation, which enable it to real-world problems. By contrast, this paper proposes an implicit cooperative learning (ICL) that have an agent separate three factors of self-agent can increase, another agent can increase, and interactions influence in a reward function approximately, and estimate a reward function for self from only acquired rewards to learn cooperative policy without any communication and observation. The experiments investigate the performance of ICL and the results show that ICL outperforms the state-of-the-art method in two agents cooperation problem.

1 INTRODUCTION

Multiagent Reinforcement Learning (MARL) controls some agents in groups to learn cooperative action, such as in warehouses where robot agents cooperate with each other to manage the delivery of supplies. In this case, MARL must decrease the complexity of communication to achieve the desired cooperation and enable the robots to solve real-world problems. In previous work, Kim et al. discussed a practical scenario for each agent to communicate with other agents in real-world reinforcement learning tasks and proposed a multiagent deep reinforcement learning (DRL) framework called SchedNet (Kim et al., 2019). Du et al. expanded the focus to the dynamic nature of communication and the correlation between agents' connections to propose a learning method to obtain the topology (Du et al., 2021). Those works are efficient and straightforward, but the agents themselves cannot do complex tasks based on real-world problems, especially in a dynamic environment. In contrast, Raileanu et al. proposed self–other modeling (SOM) method to enable agents to learn cooperative policy through predicting others' purpose or goals based only on the observation (Raileanu et al., 2018). Ghosh et al. argued that the premise of SOM requires the behaviors and types of all agents be presented as a problem and proposed AdaptPool and AdaptDQN as cooperative learning methods without using this premise (Ghosh et al., 2020). However, the proposed framework rely on the conception of step to make agents learn synchronously. It is hard to apply it for real-world problem.

Given this background, this paper proposes Implicit cooperative learning (ICL) method to enable agents to learn collective actions based on only reward signal calculation. In particular, the ICL method in which agents learn cooperative policy by estimating their appropriate reward to decrease an influence to the other agent implicitly. This paper also compared the ICL method with its baseline method and the SOM in a fully cooperative task to validate the effectiveness of the ICL method.

This paper is organized as follows: In Section 2, we introduce the background technique: the structure of game, the A3C and SOM. In Section 3, we describe the ICL method and the mathematical analysis. Furthermore, the experimental details and discussions are described in Section 4. Finally, the conclusions are presented in Section 5.

[a] https://orcid.org/0000-0003-4139-2605

Uwano, F.
Implicit Cooperative Learning on Distribution of Received Reward in Multi-Agent System.
DOI: 10.5220/0011593500003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 1*, pages 147-153
ISBN: 978-989-758-623-1; ISSN: 2184-433X

2 BACKGROUND

2.1 Problem Formulation

We consider a decentralized Partially observable Markov decision process (Dec-POMDP) $\mathcal{M}$ for two agents, defined by the tuple $\langle I, \mathcal{S}, \mathcal{A}, O, O, \mathcal{T}, r, \gamma \rangle$. There are a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and a set of observations O, and a transaction function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. The agents parcially perceive their state by following the observation function $O : \mathcal{S} \times \mathcal{A} \to O$, and selects an action by the stochastic policy $\pi_{\theta_i} : \mathcal{S} \times \mathcal{A} \to [0,1]$ and receives a reward from a reward function $r : \mathcal{S} \times \mathcal{A}_i \to \mathbb{R}$. The agent attempts to maximize the accumulated reward $R = \sum_{t=0}^{T} \gamma^t r_t$, where γ is a discount factor and T is the time horizon. Figure 1 shows an example for description of the state $\mathcal{S}$ and the observation O. The state $\mathcal{S}$ represents allocentric observation shown in the middle of the figure, while the observation O represents egocentric observation, e.g., 8-neighboring features, in the bottom of that. The agent can observe the portion of state $s \in O$.

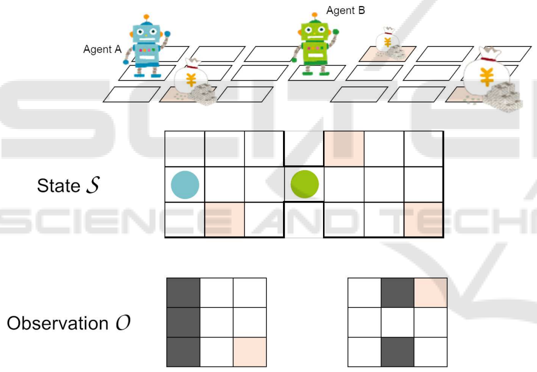

Figure 1: The problem formulation.

2.2 Asynchronous Advantage Actor–critic

The A3C algorithm (Mnih et al., 2016) is a DRL method where the system assigns the same learning components, agents, environments, and etc., into some processes each other, and then executes trials asynchronously to acquire an optimal policy immediately. This paper descriptions the details of A3C in (Fujita et al., 2019). The assigned agents execute backpropagation to update the neural network of the primary agent with the own loss of the parameters. After that learning, they initialize themselves and update the current parameters from that of the primary network and repeat the processes.

Here, we show Algorithm 1, the pseudo code of

Algorithm 1: Algorithm of A3C.

Require: Initialize thread step counter $t \leftarrow 1$
1: **while** $T > T_{max}$ **do**
2: Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
3: Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
4: $t_{start} = t$
5: Get state s_t
6: **while** terminal s_t or $t - t_{start} == t_{max}$ **do**
7: Perform a_t according to policy $\pi(a_t|s_t;\theta')$
8: Receive reward r_t and new state s_{t+1}
9: $t \leftarrow t+1$
10: $T \leftarrow T+1$
11: **end while**
12: $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \end{cases}$
13: **for** $i \in t-1, \ldots, t_{start}$ **do**
14: $R \leftarrow r_i + \gamma R$
15: $A(s_i, a_i; \theta, \theta_v) = \sum_{j=0}^{k-1} \gamma^j r_{i+j} + \gamma^k V(s_{i+k};\theta_v) - V(s_i;\theta_v)$
16: Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta') A(s_i, a_i; \theta, \theta_v) + \beta \nabla_{\theta'} H(\pi(s_i;\theta'))$
17: Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2 / \partial\theta'_v$
18: **end for**
19: Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$
20: **end while**

A3C. Generally, state s or sensed information to detect the state is input to the network and the policy π or state–action value $Q(s,a)$ is output from the network in DRL. A3C approximates the policy $\pi(a_t|s_t;\theta)$ and the state value $V(s_t;\theta'_v)$ appropriately throughout the entire processes. In particular, A3C updates the network parameters by following equations (the 16th and 17th lines in Algorithm 1):

$$d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi \left(a_t|s_t;\theta'\right) A(s_t, a_t; \theta, \theta_v) + \beta \nabla_{\theta'} H(\pi\left(s_t;\theta'\right)), \quad (1)$$

$$d\theta_v \leftarrow d\theta_v + \frac{\partial(R - V(s_t;\theta'_v))^2}{\partial\theta'_v}, \quad (2)$$

where the losses θ' and θ'_v in each process. There is an arbitrary step i to labeled the state, action and reward by s_i, a_i, and r_i, respectively. the parameter θ' is updated using the entropy function $H(\pi(s_i;\theta'))$ multiplied by the factor β. In the advantage function $A(s_i, a_i; \theta, \theta_v)$, the calculation includes the future reward multiplied by the discount factor γ.

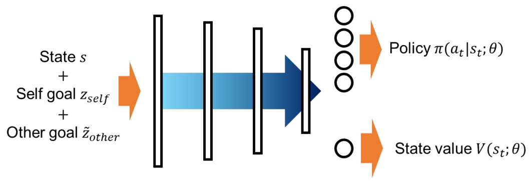

Figure 2: Neural network in SOM.

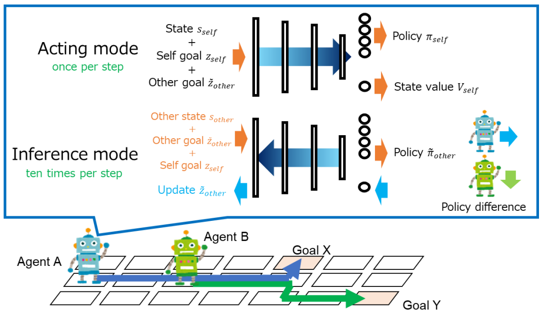

Figure 3: The process in SOM.

2.3 Self-Other Modeling (SOM)

The SOM enables an agent to estimate the other agent's goal by its neural network and input a self-state, a self-goal, and the estimated goal to learn cooperative policy. Figure 2 shows an example of the SOM model. This model learns from an observed state, the self goal z_{self}, and the self estimated other goal $\tilde{z}_{other}$, involving backpropergation of the difference between the estimated other agent's behavior and the actual one each step to update $\tilde{z}_{other}$.

Equation (3) denotes the neural network model as approximation function in the SOM. The SOM estimates parameter θ asymptotically.

$$\begin{bmatrix} \pi \\ V \end{bmatrix} = f(s_{self}, z_{self}, \tilde{z}_{other}; \theta). \tag{3}$$

Then, the SOM has two network models as follows:

$$f_{self}(s_{self}, z_{self}, \tilde{z}_{other}; \theta_{self}) \tag{4}$$

$$f_{other}(s_{other}, \tilde{z}_{other}, z_{self}; \theta_{self}). \tag{5}$$

The former is used for its learning and the latter is used for estimating the other agent's goal. Note that the SOM uses the two with the same parameter θ_{self}, that is, the two is the same but the inputs are different each other.

Figure 3 shows the process in the SOM in an example of maze problem where two agents observe their coordinates as portions of states and move with four directions, up, down, left, and right. In this problem, the agents aim to reach the different goal each other with the minimum number of steps. The SOM has two modes called the acting mode and inference mode. In the acting mode, the agent updates the parameter θ_{self} using backpropergation of the loss based on the reward and the output from the state s_{self} and the goals z_{self} and z_{other} every step in each episode. In the inference mode, the agent outputs the estimated other policy $\tilde{z}_{other}$ to update the other agent's goal $\tilde{z}_{other}$ using backpropergation with the actual action ten times for ever step. Note that the number of processing the inference mode can be changed rather than 10, but this description is using 10 along to the experiment in this paper.

The SOM has the complete information, e.g., the other agent's state. By contrast, the ICL method has a neural network in which input only the self state and the estimated goal to learn cooperative policy by the internal reward without whole of the other goal estimation.

3 IMPLICIT COOPERATIVE LEARNING (ICL)

The ICL method enables an agent to avoid unexpected interaction with the other agent to maximize the reward function by maximizing a self-reward function. The ICL method estimates the reward function as standard normal distribution and divides the reward function into two parts for an acquired reward and a reward interfered with by the other agent. The ICL method assumes that agents have the same strategy each other implicitly and learn cooperative policy by integration of both agents' optimization.

3.1 Mechanisms

We replaced the acquired reward as a scalar to the function as follows in the ICL:

$$f_g(r) = \frac{1}{\sqrt{2\pi}\sigma_g} \exp\left(-\frac{(r-\mu_g)^2}{2\sigma_g^2}\right), \tag{6}$$

where the variables μ_g and σ_g denote a mean and a standard deviation for the arbitrary goal g from the rewards acquired throughout the entire episodes, respectively. And then, the reward value is denoted by r.

Figure 4 shows an example where two agents in one maze with two goals as the same rules of Figure 2. The speech balloon represents the agent A's inside process with two reward functions estimated the own parameters μ and σ. The agent proceeds estimating the reward functions at the end of each episode. After having acquired reward, the agent update the parameters μ and σ for the corresponding goal. To update μ

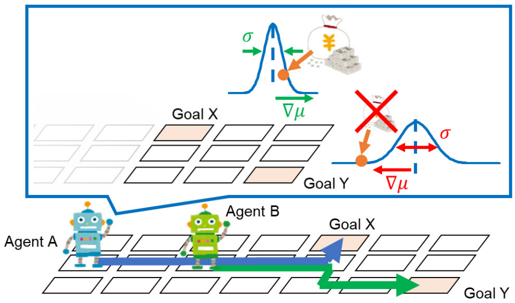

Figure 4: The mechanisms in ICL.

and σ recursively, the ICL method adopts the following equations:

$$\mu_{n+1} \leftarrow \frac{n}{n+1}\mu_n + \frac{x_{n+1}}{n+1}. \tag{7}$$

$$\sigma_{n+1}^2 \leftarrow \frac{n}{n+1}(\sigma_n^2 + \mu_n^2) + \frac{x_{n+1}^2}{n+1} - \mu_{n+1}^2, \tag{8}$$

where the variable n denotes a number of update.

This figure illustrates two cases: the agent A can reach the goal X without any interference and cannot reach the goal Y by the agent B's interference. In the former case, the agent A update the variables μ and σ in the reward function for the goal X to shape it being keen and narrowing. In the latter case, the agent A cannot acquire any amount of reward to decrease the average value μ and shape it dull.

For each learning part, the ICL method designs a reward for each agent as shown in Equation (9). The variable ir is an internal reward by which an agent learn actually.

$$ir = \frac{\mu}{\sqrt{2\pi}\sigma}. \tag{9}$$

The value of ir is on a peak of a normal distribution, which an agent aims to decrease the standard deviation σ and increase the averaged reward value μ. If the other agent interferes with, the standard deviation σ is decreased. Thus, the agent implies to learn appropriate policy with avoiding the other agents each other by maximizing the average reward value μ with decreasing the standard deviation σ simultaneously.

3.2 Mathematical Analysis

We analyzed the ICL mechanisms in one condition that the reward function can be separated by three functions approximately shown as follows:

$$R(\tau_{self}, \tau_{other}) \approx r(\tau_{self}) + r(\tau_{other}) + r_i(\tau_{self}, \tau_{other}),$$

where R, r, and r_i denote a total reward function and reward functions for self and the other agents, and their interaction, respectively. τ_{self} and τ_{other} denote potential variables which can be routes, sequential state-action pairs, etc. to represent the agents' trajectories and difference each other. However, this paper adopts the trajectory, that is, the reward must be decided based on the agents' routes.

First, we derive the total derivative $r_i(\tau_{self}, \tau_{other})$ as follows:

$$\begin{aligned} &dr_i(\tau_{self}, \tau_{other}) \\ &= \left(\frac{\partial R(\tau_{self}, \tau_{other})}{\partial \tau_{self}} - \frac{\partial r(\tau_{self})}{\partial \tau_{self}}\right) d\tau_{self} \\ &+ \left(\frac{\partial R(\tau_{self}, \tau_{other})}{\partial \tau_{other}} - \frac{\partial r(\tau_{other})}{\partial \tau_{other}}\right) d\tau_{other}. \end{aligned} \tag{10}$$

If the derivative $dr_i(\tau_{self}, \tau_{other})$ is zero, the following two conditions are hold:

$$\frac{\partial R(\tau_{self}, \tau_{other})}{\partial \tau_{self}} = \frac{\partial r(\tau_{self})}{\partial \tau_{self}}. \tag{11}$$

$$\frac{\partial R(\tau_{self}, \tau_{other})}{\partial \tau_{other}} = \frac{\partial r(\tau_{other})}{\partial \tau_{other}}. \tag{12}$$

These conditions show the derivative values of the total reward function and the partial reward function are the same each other in each agent's potential variable, that is, the influence of the total reward function is completely according to that of the partial reward function. Therefore, the ICL method makes an agent decrease a variance of a reward throughout episodes to prevent from interception with the other agent. Furthermore, the variance approximately is zero, the derivative value $dr_i(\tau_{self}, \tau_{other})$ is also zero.

4 EXPERIMENT

4.1 Experimental Setup

To investigate the effectiveness, this paper compared the ICL method with the SOM and A3C in Coin Game, respectively. The ICL method employed the A3C algorithm based on (Fujita et al., 2019). The evaluation criteria are the spent step until all agents have reached the goal and the agents' acquired rewards.

4.2 Coin Game

Coin Game (Raileanu et al., 2018) is a fully cooperative task, which agents aim to gain maximum reward when they have both of their goals rather than only their goal. In this task, every agent is assigned of a color of coin and its policy is evaluated by Equation

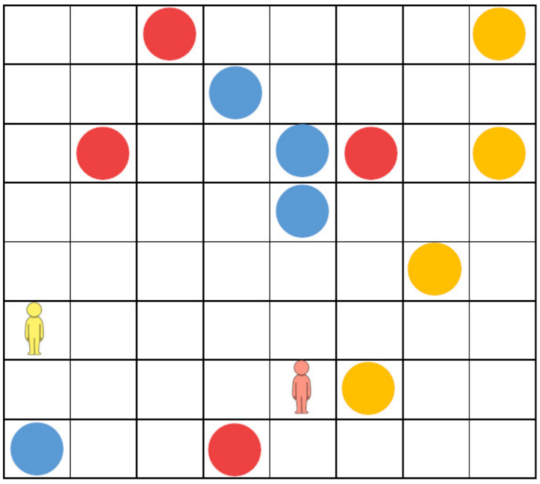

Figure 5: An example of Coin Game.

(13) as follows in the color assignment:

$$R_{coin} = \left(n^{self}_{C_{self}} + n^{other}_{C_{self}}\right)^2 + \left(n^{self}_{C_{other}} + n^{other}_{C_{other}}\right)^2 - \left(n^{self}_{C_{neither}} + n^{other}_{C_{neither}}\right)^2, \quad (13)$$

where R_{coin} denotes a reward to share both of agents. The words $self$ and $other$ indicate the agent which actually gain the reward and the other agent, respectively. Thus, C_{self} and C_{other} indicate the color of coins assigned to $self$ and $other$ agents, respectively, and $n^{self}_{C_{self}}$, $n^{self}_{C_{other}}$ denote the numbers of coins assigned to the $self$ and $other$ agents in the coins gathered by the $self$ agent, respectively. The number $n^{self}_{C_{neither}}$ denotes the number of coins assigned to no-one in the coins gathered by the $self$ agent. The numbers of coins $n^{other}_{C_{self}}$, $n^{other}_{C_{other}}$, and $n^{self}_{C_{neither}}$ are denoted in the same manner of the $self$ agent.

Figure 5 shows a circumstance of Coin Game in one episode. There represent two humans as agents and 12 circles as coins in a 8×8 grid world, and the color of agents and coins shows the assigned colors. The agents explore to gather the same color coins in an entire episode and gain a reward at the end of the episode. After that, the location and color of agents and coins are changed and the agents go on to the next episode. The agents are assigned of different colors each other, and the colors are changed in each episode in Coin Game. In the ICL method, let the potential variable τ be a pair with assigned color and self-gathered coins.

4.3 Training Details

The parameters are summarized on Table 1. Totally, 0.2 billion steps are executed (first line) and agents learn up to 10 steps in one episode (second line). An episode is finished within 10 steps if all coins have been gathered. The number of processes is 16 (third line). The parameters α, γ, and β are set by 0.0007, 0.99, and 0.01 (i.e., fourth, fifth, and sixth lines, respectively). The total number of steps in an episode and the number of processes are the same as SOM.

The ICL method's neural network in which two hidden layers, dense and LSTM layers. The dense layer is connected from the input layer and connects to the LSTM layer. The LSTM layer connects to two output layers for a policy and a state value. The parameter optimization using Adam (Kingma and Ba, 2017). The hidden layer dimension was 64. The network is the same as SOM. As for the input features, there are seven features (two agents, three coins, a wall and an empty state) in each of 64 grid squares and the input has 448 features totally. There are six features, less than ICL by one feature, in (Raileanu et al., 2018), however ICL has "nothing" class whereas SOM classify it by decreasing all features to zero, that is, both have the same features actually. By the way SOM has six features (two agents, three coins, a corner.) Furthermore, there are addition input features show self and other colors (i.e., six-dimension vector) in the SOM. The ICL has only self-color as additional features with three dimensional vector.

4.4 Results

Figure 6 shows the performance of ICL and A3C. The vertical and horizontal axes indicate a total value of reward and each episode, respectively. Both of the results are moving average in 10,000 episodes. This figure shows the ICL method outperforms the A3C and the result converged to over 13. Figure 7 shows the performance of the SOM and the other baseline methods as shown in (Raileanu et al., 2018). Although this paper did not explain the detail of the other baseline methods, the TOG, or the high performance method, is not considered because it utilizes a given and optimal values in a part of parameters. The vertical and horizontal axes indicate a total value of reward and each episode, respectively. The scale of episode in ICL and A3C is decuple than SOM, however SOM

Table 1: Experimental parameters.

Total number of steps in all episodes	20,000,000
Total number of steps in an episode	10
Processes	16
Learning rate α	0.0007
Discount rate γ	0.99
Rate β	0.01

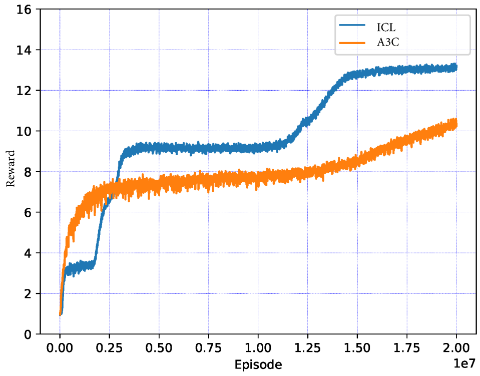

Figure 6: Coin performance in ICL and A3C.

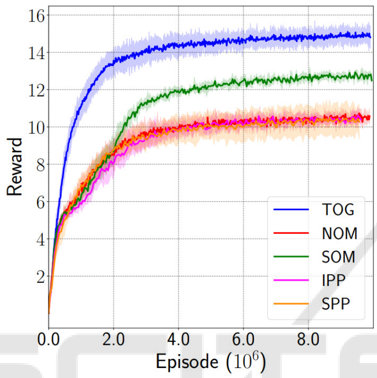

Figure 7: Coin performance in SOM. (Raileanu et al., 2018).

executes optimization update of its parameter for 10 times in every step. Thus actual scale is the same each other and the number of ICL and A3C update parameter is less than SOM. Each result is averaged in 5 runs and the standard deviation is shown as the light color area around the line. This figure shows the result of SOM converged to under 13. That result is naturally smaller than that of the ICL.

4.5 Discussion

4.5.1 Stability of Learning

The result of ICL method is moving average because that is not stably by comparing that of the SOM. However, the difference of each framework, without learning performance, causes that effect. Concretely, that is because the SOM enables agents to update their parameters for 10 times at each step, the trajectory is stable in Figure 7. However, that seems to become serious problem in real world. That is because there is no conception of step. For example, there are two robots in a warehouse. The robots carry supplies on to the appropriate shelf and arrange them cooperatively. In this circumstance, each agent makes a delay by the update parameter. Thus, the SOM required agents achieve synchronous actions.

That unstableness can be solved by the Maximum entropy inverse reinforcement learning (MEIRL) technique in (Ziebart et al., 2008). In particular, the ICL utilizes the potential variable τ similar to the expert demonstration. That is one of the future works.

4.5.2 Convergence

The trajectory of the A3C and the SOM goes on in smooth. However, that of the ICL goes on in steps. That shows the ICL transited through three kinds of equilibrium. In the first equilibrium, the agents gained about three as a value of reward. In the next equilibrium, the agents gained about nine in a long term. That result suggests the agents can averagely gather two coins. In the final equilibrium, the agents gained about thirteen in which they can gather averagely two and a half coins. The ICL avoid unstable policy which causes a bad effect to an interaction between agents and keeps a new valuable equilibrium, unlike other methods. That is why the result of the ICL has small variance and outperform than that of A3C.

4.5.3 Limitation

Agents should have gained 32 value of reward in Coin Game. However, the ICL method could not achieve that. That reason is the difference or lack of premise. The result shows us right of the premise that the self-reward function follows a normal distribution and the influence of interaction could be in the standard deviation. However, a noise in the total reward cannot always follow a normal distribution. In short, all of the standard deviation do not represent the influence of interaction. At least, a normal distribution cannot represent change of the total reward function by the other agent's learning and that change is a part of the standard deviation. The ICL has limitation in terms of a premise of distribution.

To tackle this issue, there is a way in how to utilize kernel density estimation. However, the computational complexity intercepts the ICL method. Thus, the more suitable parametric distribution should be considered in the future.

5 CONCLUSION

This paper proposed the ICL method in which agents learn cooperative policy by estimating their appropriate reward to decrease an influence to the other agent implicitly. Concretely, the ICL method makes an agent separate three partial reward functions for self and the other agents, and an interaction with the agents approximately, and estimates a reward function for self based on only acquired rewards to learn without factors of the other agent and an interaction with the other agent. The experiments compared the ICL method with A3C and the SOM as baseline methods. The results show (1) the ICL method outperformed all of the other methods; and (2) the ICL method can avoid unstable policy which causes a bad effect to an interaction between agents and keeps a new valuable equilibrium.

This paper showed the ICL method has two kinds of limitation: learning stability and premise of distribution. To overcome the limitation, we will apply the MEIRL method for the ICL method in the future. After that, we will examine a suitable distribution for the reward function. We premised a distribution combined two distributions for two agents, and each distribution is enough to be a normal distribution.

ACKNOWLEDGEMENTS

This research was supported by JSPS Grant on JP21K17807 and Azbil Yamatake General Foundation.

REFERENCES

Du, Y., Liu, B., Moens, V., Liu, Z., Ren, Z., Wang, J., Chen, X., and Zhang, H. (2021). *Learning Correlated Communication Topology in Multi-Agent Reinforcement Learning*, page 456–464. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC.

Fujita, Y., Kataoka, T., Nagarajan, P., and Ishikawa, T. (2019). Chainerrl: A deep reinforcement learning library. In *Workshop on Deep Reinforcement Learning at the 33rd Conference on Neural Information Processing Systems*, Vancouver, Canada.

Ghosh, A., Tschiatschek, S., Mahdavi, H., and Singla, A. (2020). *Towards Deployment of Robust Cooperative AI Agents: An Algorithmic Framework for Learning Adaptive Policies*, page 447–455. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC.

Kim, D., Moon, S., Hostallero, D., Kang, W. J., Lee, T., Son, K., and Yi, Y. (2019). Learning to schedule communication in multi-agent reinforcement learning.

Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783.

Raileanu, R., Denton, E., Szlam, A., and Fergus, R. (2018). Modeling others using oneself in multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4257–4266, Stockholmsmassan, Stockholm Sweden.

Ziebart, B. D., Maas, A., Bagnell, J., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI2008)*, pages 1433–1438, Chicago, USA. AAAI.