Multi-agent Communication with Attentional and Recurrent Message Integration

1st Zhaoqing Peng
University of Chinese Academy of Sciences
Beijing, China
pengzhaoqing16@mails.ucas.ac.cn

2nd Libo Zhang
Institute of Software, CAS
Beijing, China
zsmj@hotmail.com

3rd Tiejian Luo
University of Chinese Academy of Sciences
Beijing, China
tjluo@ucas.ac.cn

***Abstract*—Effective communication is significant for solving cooperative tasks in multi-agent domain. Agents coordinate their behaviors by appropriately modeling the communication signals or messages sent from others. To this end, agents are required to filter noise and obtain useful information from received messages, and learn to adapt to the dynamics of messages number. In this paper, we propose an attentional and recurrent message integration method (ARMI) that handles the dynamics by recurrently decoding messages, and performs attentional integration based on the relevance of each message. We evaluate our proposal on a new "predator-prey-toxin" environment where the number of agents changes, and the results outperform other competing multi-agent methods. Further investigations are also done to prove the superiority of ARMI in collaborating agents' behaviors for complex tasks and establishing interpretable communication protocol.**

***Index Terms*—multi-agent reinforcement learning, recurrent message integration, attention mechanism**

I. Introduction

Many tasks in real-world problems require the collaboration of multiple agents [1]. In multi-agent settings, to achieve self-awareness, each agent observes its state in the environment from its local view points, and this partial observability may lead to the non-stationary of each agent's perception due to other agents' actions [2]. Therefore, communication among them is essential for each agent to learn and understand the behaviors of other agents. Efficient learners must establish meaningful communication protocol to coordinate their behaviors and learn to solve complex tasks in collaboration.

The rapid progress in recent years of deep reinforcement learning [3] opens the door to a new perspective on multi-agent domain. Deep neural networks are usually adopted to produce messages to transmit information among agents, and reinforcement learning (RL) is applied to guide the optimization of learning well representations of messages through reward signals of the environment. However, the handling of generated messages for each agent remains challenges in two aspects: 1) how to enable agents to extract useful information from massive messages without prior knowledge, and 2) how to allow the scalability of agents' social interactions especially when the agent number is dynamic.

In this work, we formulate the process of handling messages using an *integration function* that inputs raw receiving messages and outputs a hidden communication vector for agent's decision making. To model the dynamics of message number and simultaneously learn the useful interactions more efficiently, we propose an *attentional and recurrent message integration* (ARMI) method that consists of two specific integration functions: 1) an attentional function that uses an additive attention value to represent the relevance of each received message w.r.t the observation, and 2) a recurrent function that utilizes the recurrent neural network (RNN) to feed in each message by the receiving order. The recurrent function shows its benefits on modeling the sequential information of the received messages and dealing with the dynamic agent number, while the attentional function enables each agent the capacity of giving more attention to relevant and useful messages according to its current observation.

To verify the effectiveness of the proposal, we evaluate it on a new "predator-prey-toxin" environment which consists of continuous state-action space and a new toxin role to increase the complexity of tasks. During training, we periodically increase the agent number to study the scalability and performance of each method under a dynamic environmental setting, and the empirical results show that our method reaches much higher score than the other methods after introducing the toxins, and demonstrates the advantage of ARMI in handling such complex cooperative tasks.

In order to further interpret the well performance of our proposal, we then visualize the attentions of the agent under an example state, and the results indicate our method can make agents learn effective interactions by paying more attention to useful and relevant messages. We also analyze how messages affect the agent's policy, which shows that the receiving agent could make decisions in view of the fellow agents' states while the communication-disabled method cannot. Finally, we visualize the communication protocol and find a significant correlation between the message and the receiver agent' action. The results prove that agents correctly interpret the received message and appropriately make decisions in view of those messages.

The remainder of this paper is organized as follows. Section 2 discusses the previous literature related to our work, which is followed in Section 3 by a formulation of the environment and an introduction to main RL techniques. The detail of our proposal is presented in Section 4. Section 5 gives insights on

Libo Zhang is the corresponding author.

the conduction of serval experiments and the numerical results. Section 6 provides some final conclusions and directions for future work.

II. Related Works

Independent learning (IL) is the simplest method to extend the single-agent RL to multi-agent domain such as [4], [5] where each independent learner only cares about its own unique policy and does not have any explicit or implicit communication. IL will face the challenge of not being able to tackle the non-stationary environments.

There exists some related works addressing communication issues: literature [6] simply concatenates all received messages and feeds them to a target network. This naive method may fail in the case where the agent number is large and also dynamic, since the input size of the first layer in the network will linearly grow as the agent number increases, which is intractable for simple feed forward network such as multi-layer perceptions (MLP). Multi-Agent Actor-Critic [7] proposes a simple extension where the critic of each agent has access to the global policies of other agents while the actor only observes its local state. BiCNet [8] unrolls the bi-directional RNN across agents, which allows to transmit the communication information by the hidden state of each recurrent cell. DIAL [9] enforces agents to learn communication protocols through considering other agents' messages generated in previous time-step. However, the message in DIAL is delayed for one time-step, which could cause the bad action decision conditioned on the old and expire states of other agents.

CommNet [10] conducts averagely pooling operation over all received messages to get a communication vector for all agents, but it puts the whole burden of modeling interactions on the message extractor. In contrast, VAIN [11] increases the diversity of receiving messages by adding attention mechanism over CommNet, and it allows different strengths of the interactions among agents. Our previous work [12] introduces prior knowledge of the environment to help optimize the attentional value of each message, but it still ignores the scalability of agent number. In this work, we are more interested in studying the message integration issues under the environment with dynamic agent number.

III. Background

Multi-agent Markov Games: In this work, we consider a zero-sum Stochastic Game (SG) with the multi-agent and partially observed settings. We formulate it as a Partially Observed Markov Decision Processes (POMDPs), denoted by a tuple $G = \langle S, U, P, R, \Omega, O, n\rangle$, in which n agents interact with the environment. Time is discrete and at each time-step, each agent takes action $a \in A$, forming the joint action space $\mathbf{a} \in \mathbf{U} \equiv U^n$, which induces a environment transiting from a global state $s \in S$ to new state $s' \in S$ based on the state transition function $P(s'|s, \mathbf{a}) : S \times \mathbf{U} \times S \to [0, 1]$. Each agent receives a local observation $o \in \Omega$ according to observation function $O(s, a) : S \times U \to \Omega$ and chooses the action with a stochastic policy $\pi(a|o) : \Omega \times U \to [0, 1]$. After performing the joint action $\mathbf{a_t}$ to the environment at the time-step t, the agents receives a joint reward vector $\mathbf{r_t} = R(s_t, \mathbf{a_t}) \in \mathbb{R}^n$ describing the performance $r_k^t \in \mathbf{r_t}$ of each agent k. The objective for each agent k is to maximize its own expected return $R_k = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_k^t]$ where $\gamma \in [0, 1)$ is the discount factor.

Deep Q-Networks and Policy Gradient: In single agent RL, the action value $Q^\pi(s, a) = \mathbb{E}_\pi[R_t|s_t, a_t = s, a]$ represents expected return for selecting action a in state s according to policy π, while $V^\pi(s) = \mathbb{E}[R_t|s_t = s]$ is the state value of state s under the policy π. In model-free RL methods, these functions are provided by the environment, where the agent learns the optimal policy π^* through trial-and-error interactions. However, due to the huge searching space of continuous states and actions, it is not feasible to directly mapping from state-action pairs to the corresponding Q-value in practice. Therefore, the recent deep Q-network (DQN) [3] introduces a function approximator such as a neural network to approximate optimal action-value function $Q(s, a; \theta)$ with parameter configuration θ, *i.e.*, $Q(s, a; \theta) \approx Q_{\pi^*}(s, a)$. At the time-step t, the parameters θ are optimized using stochastic gradient descent by minimizing the loss functions:

$$L_t(\theta_t) = \mathbb{E}(y - Q(s, a; \theta_t))^2, \tag{1}$$

where $y = R_t(s, a) + \gamma \max_{a'} Q(s', a'; \theta_{t-1})$, s' is the next state transmitted from state s, and a' is the action of the next state.

The policy gradient [13] directly parameterizes the policy $\pi(a|s; \theta)$ and updates the parameters θ in the direction of $\nabla_\theta \log \pi(a_t|s_t; \theta) R_t$. In order to reduce the variance of this estimate $\log \pi(a_t|s_t; \theta) R_t$, a relative advantage of the action w.r.t the current state is introduced by subtracting a relative baseline $b_t(s_t)$, that is, $R_t - b_t(s_t)$. We commonly use the action value as the estimate of R_t and the learned state value to represent the baseline $b(s_t)$ [14], thus the advantage of action a_t in state s_t can be defined as

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t), \tag{2}$$

the resulting gradient is $\nabla_\theta \log \pi(a_t|s_t; \theta) A(s_t, a_t)$. This approach can be achieved by actor-critic methods in RL where the policy π is produced by the actor and the baseline is generated by the critic.

IV. Method

In this work, we are devoted to solve the partially observed, fully cooperative task in multi-agent domain with deep RL, and we consider a general communication setting: 1) one agent first produces its own message and broadcast it to all the others, 2) one receives the messages from the other agents and then makes decisions on its current action. Our motivation is to make each agent capable of modeling the dynamics of received messages and learn the integration of useful messages for better coordinating its behavior with other agents.

Specifically, we consider a game with N agents, each agent i observes its local state o_i^t at the time-step t and learns a mapping that transforms current observation o_i^t into a message $m_i^t = \mathcal{M}_i(o_i^t; \theta_i^m)$ with parameters θ_i^m. Each message

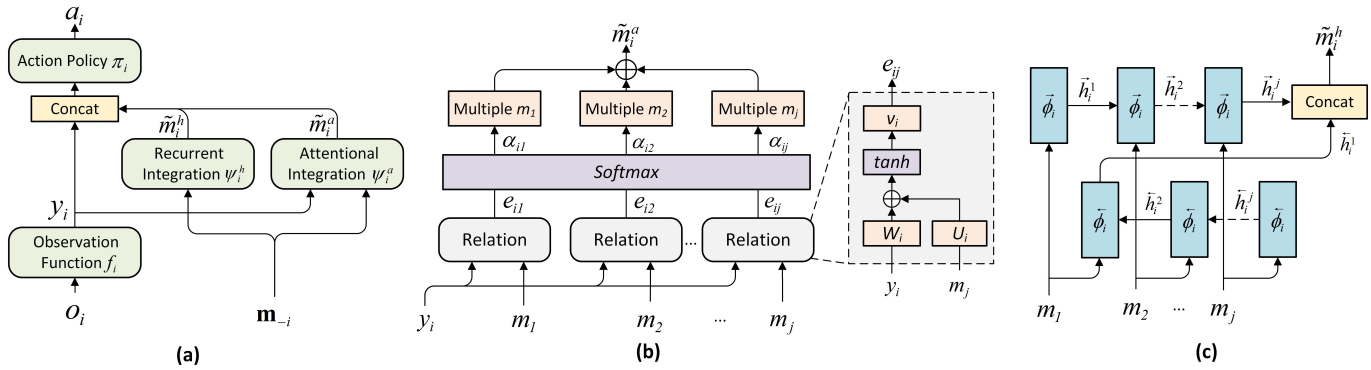

Fig. 1. (a): The message integration of agent i. (b): The attentional integration function ψ_i^a. (c): The recurrent-based integration function ψ_i^h

$m_i^t \in \mathbb{R}^l$ consists of l continuous values in each time-step. We assume that each agent i receives messages from teammate agents except for itself. Let $\mathbf{m}_{-i}^t$ denote the received message set of agent i by $\mathbf{m}_{-i}^t = \{m_1^t, m_2^t, \cdots, m_j^t, \cdots, m_N^t | j \neq i\}$. Each agent obtains a hidden observation state $y_i^t = f_i(o_i^t; \theta_i^o)$ by an observation function with parameters θ_i^o.

We propose two integration functions that deal with the received messages $\mathbf{m}_{-i}^t$ as the Fig. 1 (a) illustrates: 1) a recurrent integration function ψ_i^h that models the dynamics of received messages and integrates them into a hidden communication vector $\widetilde{m}_i^h = \psi_i^h(\mathbf{m}_{-i}^t)$ and 2) an attentional integration function ψ_i^a that outputs an attentional communication vector $\widetilde{m}_i^a = \psi_i^a(\mathbf{m}_{-i}^t, y_i^t)$ according to the relevance of each received message w.r.t its current observation. The agent action a_i is conditioned on two integrated messages and its own observation, which is given by $\pi_i(a_i | \widetilde{m}_i^h, \widetilde{m}_i^a, y_i^t; \theta_i^u)$ with parameters θ_i^u. Whether agents eventually develop good cooperative policies depends on how well these two integrated messages are learned.

A. Recurrent Integration Function

In view of the sequential order information of the messages, we make use of the RNN to implement the message integration where each received message is recurrently fed by the receiving order. Therefore, the integration function ψ_i^h is directly parameterized with RNN paramters θ_i^h, and we have $\widetilde{m}_i^h = \psi_i^h(\mathbf{m}_{-i}^t; \theta_i^h)$.

Specifically, as the Fig. 1 (c) shows, we unroll the recurrent cell ϕ_i along the dimension of the message number (also agent number). The hidden state output h_i^k of received message m_k is given by:

$$h_i^k = \phi_i(h_i^{k-1}, m_k; \theta_i^h) \tag{3}$$

where the h_i^{k-1} is the hidden state output of previous feeding message m_{k-1}. Our final hidden communication vector $\widetilde{m}_i^h$ is actually the hidden state output of the last feeding message. In practical, to allow the input information of h_i^k reachable to h_i^{k-1}, we use bidirectional-RNN architecture [15] to decode the messages in both forward and backward directions. In this case, the $\widetilde{m}_i^h$ will become two concatenated hidden vectors: $\overrightarrow{m}_i^h$ and $\overleftarrow{m}_i^h$, which are the last hidden state outputs of the forward and backward network with parameters $\overrightarrow{\theta_i}^h$ and $\overleftarrow{\theta_i}^h$ respectively.

This recurrent integration is inherently beneficial to deal with dynamics of agents. When there is a new agent k that enters the environment, we just need to dynamically instantialize one more recurrent cell and put it in the head or tail of the message flow chain for receiving message m_k. Similarly, if there is an agent k that is deactivated (exit the environment), we can skip the computing for m_k and use the previous hidden states of message (*e.g.*m_{k-1}) to replace the hidden state of m_k.

Another benefit of recurrent integration is that: the sharing of parameters between all the recurrent cells can learn the repeated similar messages more efficiently. For instance, the experience of learning the message m_k can be used to learn message m_n especially when the m_k and m_n convey similar interaction information. In these settings, to make it easier for the receiving agent to know where the message comes from, we input each message with the index of the sending agent.

B. Attentional Integration Function

Since not all the received messages are useful to make decisions under broadcast settings, we propose an attentional integration function ψ_i^a that gives more attentions to relevant and useful messages w.r.t its current observation, and our communication vector is given as $\widetilde{m}_i^a = \psi_i^a(\mathbf{m}_{-i}, y_i; \theta_i^a)$ with paramters θ_i^a. The idea behind is straightforward that if the $\widetilde{m}_i^a$ contains as much as relevant and useful information, it will be more helpful for the agent to make better decisions in view of teammate agents' states.

In this case, as Fig. 1 (b) depicts, the $\widetilde{m}_i^a$ is computed as a weighted sum of each received message m_j:

$$\widetilde{m}_i^a = \sum_{j \neq i} \alpha_{ij} m_j \tag{4}$$

where α_{ij} is the additive attentional value [16] representing the relevance of the message m_j, and is computed by:

$$\alpha_{ij} = \frac{e^{e_{ij}}}{\sum_{k \neq i} e^{e_{ik}}} \tag{5}$$

where the energy value e_{ij} is modeled by a relation function a with the input of the message m_j and the observation hidden state y_i:

$$e_{ij} = a(y_i, m_j) = v_i \tanh(W_i y_i + U_i m_j) \tag{6}$$

where W_i, U_i, v_i are the learned weight matrixes (actually denoted by θ_i^a) of attentional mechanism to evaluate how relevant between y_i and m_j. We reinforce agent i to generate a larger value of α_{ij} with the corresponding message m_j when the occurrence of m_j can bring up rewards for it.

C. Policy Learning

We use actor-critic architecture of RL to learn each agent's action policy, and the actor is actually composed of four parts: $\psi_i^a(\theta_i^a)$, $\psi_i^h(\theta_i^h)$, $f_i(\theta_i^o)$ and $\pi_i(\theta_i^u)$. For simplicity, we use θ_i to denote these parameters, thus the action policy can be revised as $\pi_i(a_i|o_i, \mathbf{m}_{-i}; \theta_i)$. The critic shares non-output layers of the actor and outputs one linear value $V_i(o_i, \mathbf{m}_{-i}; \omega_i)$ with parameters ω_i.

The actor and critic are updated according to asynchronous advantage actor-critic (A3C) [14]. At the time-step t, the actor of agent i is optimized through policy gradients as $\nabla_\theta \log \pi(a_i^t|o_i^t, \mathbf{m}_{-i}^t, \theta_i) A_i^t$ where the advantage function A_i^t is given by $R_i^t - V_i(o_i^t, \mathbf{m}_{-i}^t, \omega_i)$, and the critic is optimized with gradients $\partial(R_i^t - V_i(o_i^t, \mathbf{m}_{-i}^t; \omega_i))^2/\partial\omega_i$. The updates are performed after every action until reaching a terminal state, thus R_i^t can be computed by $R_i^t = r_i^t + \gamma V_i(o_i^{t+1}, \mathbf{m}_{-i}^{t+1}; \omega_i)$.

Since all the functions and operations are continuous and differentiable, an end-to-end training is allowed to apply for the whole model and the message extractor $\mathcal{M}_i(o_i^t, \theta_i)$ is optimized by the gradients passed down from each communicating actor through backpropagation.

V. Experiment

In this section, we conduct different experiments to verify the effectiveness and superiority of our ARMI, compared to the baseline IL, DIAL, CommNet, and VAIN. All the methods are implemented on MXNet, a scalable deep learning framework, and our game is built on Pygame development tool.

We give specific implementation of ARMI as followings: each message has the length $l = 50$, and the observation function uses one hidden MLP of 255 neurons. The sizes of attentional weights W, U, v are set to $255 \times 50, 50 \times 50, 50 \times 1$ respectively, while the recurrent-based function is modeled by one stack layer LSTM with hidden size of 255. We use a variant of stochastic gradient descent method Adam [17] to optimize the parameters with default hyperparameters and a learning rate of 5×10^{-4}. The discount rate is fix to $\gamma = 0.99$, and we start 32 threads running 32 environment instances to break the irrelevance of the samples for updating the networks.

A. Environmental Setup

We use a predator-prey pursuit problem to be our testbed, which is a typical benchmark problem of multi-agent RL [18]. Different from traditional grid-world pursuit problem, we design a continuous state-action space environment to simulate a realistic case in real-world applications. We implement a prototype game with a newly introduced toxin role to make the task more challenged.

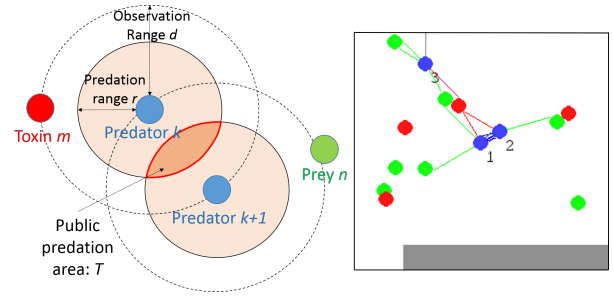

Fig. 2. Predator prey toxin environment

Our scenario is that our predators (agents) need to cooperate with each other to catch preys for earning rewards, and also avoid punishments of colliding with toxins. As Fig. 2 (Right) shows, we have n predators (blue), m toxins (red) and k preys (green) in total running in the environment. Each agent has a limited observation range and a fixed predation range as in the Fig. 2 (Left). An agent is awarded when a prey is fully covered by the public predation area, which is formed with the coordination of the fellow agent. Negative reward is given each time when an agent comes into contact with any toxin. The existence of toxins increases the difficulty of collaboration that agents are more likely to get separated while they encounter toxins, which may lead to a sparser positive reward distribution. Obviously, the optimal strategy of agents is to maintain the maximum of public predation area while cruising around together and also avoiding toxins.

State space: the observation of each agent is divided into z directions in average and for each direction, there is an eye sensor pointing in the observation depth d. Each eye sensor observes the distance to the wall, the prey, the toxin and the fellow agents. In addition, the agent perceives two additional sensors for its own speed in both x and y directions.

Action space: there are 4 actions available for the agent to control the velocity: to apply thrusters to the left, right, up and down in an accelerated speed, and the velocity is attenuating with a certain decay rate.

The environment is reset after reaching the terminal state in each episode. The max step in an episode is limited to 1,000, and we carried out the experiment with 40 epochs for each approach and one epoch contains maximum 100,000 steps.

B. Scores Evaluation

We first consider a standard environment instance with 5 predators, 20 preys and 10 toxins (5-20-10). Note that the environment is dynamic since we will add new agents into the game within certain number of epochs. In our case, the agent number starts from 2 in the 0th epoch and reaches 5 in the 24th epoch. Therefore, the training performance of each method is evaluated by its average score, which is the ratio of total rewards to the current agent number in one episode.

As Fig 3. shows, our ARMI almost obtains the highest scores in the whole training process, which indicates the agents may have a stronger learning efficiency to earn rewards and also avoids toxins.

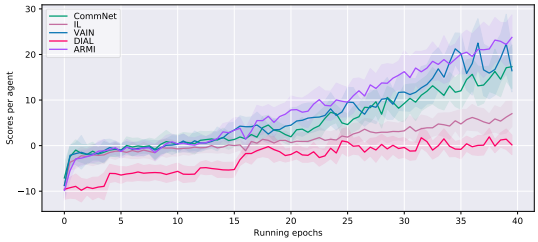

Fig. 3. Average scores per agent of each method

To suppress the interference of introduced toxin role, we further investigate the performance of each method and the random policy in an environment without toxin (5-20-0). Table I shows the statistical analysis of the average scores and collisions per agent in last 20 epochs, and we find that VAIN and ARMI achieve similar scores without the disturbing of toxins, but the superiority of ARMI comes out after toxins are introduced. This result proves that ARMI could perform better in such complex environments, which is due to the stronger negotiation skill of each agent while encountering toxins. We will give out some further discussions on the reason in following sections.

TABLE I
STATISTICAL RESULT OF THE PERFORMANCE IN LAST 20 EPOCHS

Approach	*Score (5-20-0)*	*Score (5-20-10)*	*Collision (5-20-10)*
Random	1.6 ± 1.2	-17.5 ± 6.8	18.4 ± 7.3
IL	8.3 ± 4.0	3.5 ± 2.9	14.8 ± 4.3
DIAL	3.1 ± 1.7	-0.4 ± 2.3	25.8 ± 6.8
VAIN	18.7 ± 7.1	12.1 ± 6.6	14.6 ± 4.3
CommNet	16.9 ± 6.0	9.6 ± 5.3	13.6 ± 3.9
ARMI	**19.6 ± 6.0**	**15.1 ± 6.4**	**15.1 ± 4.4**

C. Attention visualization

Let us further visualize the attentional weights generated by ARMI to understand the nature of interactions between the agents. We randomly sample a state from the environment as Fig. 4 (Left) illustrates: the agent 1, agent 2 and agent 3 are in the observation range of each other while the others are completely separated. In this state, since the public predation area is the key to get rewards, the agents should pay more attentions to the messages sent from the close agents. Therefore, these three agents should care more about the message sent from each other rather than the agent 0 and agent 4, which is correctly learned in ARMI shown in Fig. 4 (Right). Although there exist some attentional distractions for the separated agents, we leave it to future investigation.

D. Comparison with IL

We now investigate how our method works by receiving messages compared with the case without message exchanging. We examine the changes of one agent's policy only

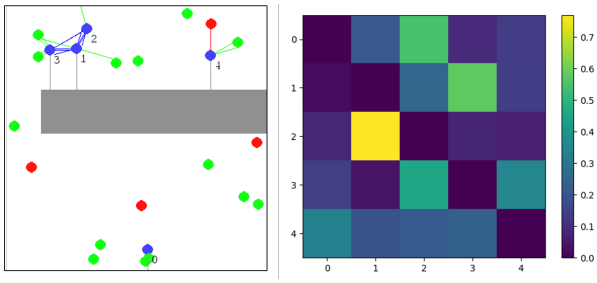

Fig. 4. A screenshot of the sampled environment (Left), and the heatmap of the current attentional weights in matrix from (Right): the color of the block at i-th row and j-th column represents the attention strength (α_{ij}) of the agent i w.r.t the message sent from agent j.

under the influence of messages sent from the fellow agent. Therefore, we consider a scenario where agent 1 perceives a prey in its left side and agent 2 in its right side, in this case, agent 2 only depends on the messages sent from the agent 1 to decide the next action. We observe the changes of the probability of choosing left action in agent 2 when the distance between them changes. The policies are collected through the trained networks of our method and IL in different epoch, and we plot them in 3-D space shown as Fig. 5.

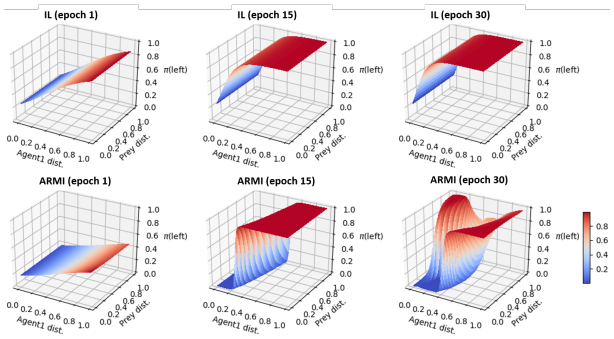

Fig. 5. 3-D mappings of the left action

As the Fig. 5 depicts, two specific patterns are well learned for agent 2 in our method: 1) the left action is more inclined to be selected when the distance with agent 1 decreases because the two agents are trying to get near with each other to keep the public predation area, 2) once the agent 1 observes the appearance of the prey, the left action of the agent 2 is dominating since it needs to move towards left side to eat the prey and get rewards. However, the second pattern is not formulated in the IL method because there is no provided information for the agent 2 to decide the next action. This result shows that the messages can serve as the information carrier that transfers the agent 1's instruction of moving left to agent 2. More importantly, the messages are correctly interrupted and understood by the agent 2, thus the behavior is successfully coordinated.

E. Communication protocol visualization

We can also visualize the communication protocol learned in the predator-prey domain by mapping the high dimensional

messages data in a two-dimension projection space. We sample 1000 points of 50-dimensional messages sent from one certain agent to another and the corresponding action policies in the final epoch of our method. According to the policy, each message can be labeled with 4 distinct values representing the taken action of up, down, left and down. However, this labelling way is discrete, deterministic and ignores the probability nature of the stochastic policy. Therefore, we map the output action policy to a continuous angle space in which each message is labeled with a clockwise angle θ w.r.t to the direction of up action, which can reflect the resulting direction of this message. The angle θ can be computed by the following equation where $\pi(a_{up})$, $\pi(a_{down})$, $\pi(a_{right})$ and $\pi(a_{left})$ are the probability of the up, down, right, left action respectively.

$$\theta = \arctan\left(\frac{\pi(a_{up}) - \pi(a_{down})}{\pi(a_{right}) - \pi(a_{left})}\right). \tag{7}$$

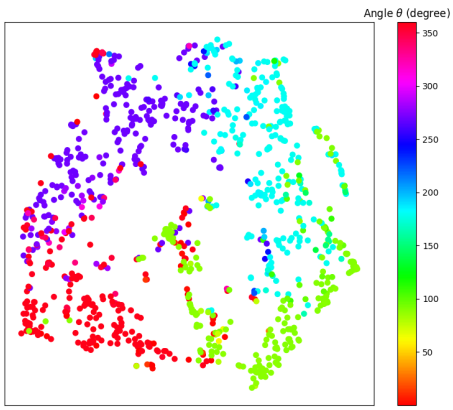

Fig. 6. t-SNE embedding representations of communicated messages

We use t-SNE [19] to map data with similar features near to each other, thus the messages with similar content will get clustered in one region of embedding space. As shown in the Fig. 6, each message is colored by the angle value θ and we find four distinct colored areas are formed in the figure to represent the four types of the message features that result in four different actions, such as the red dots correspond to the UP action, etc. This result shows a strong correlation between the received messages and the decided action in one agent, which explains the fact that the features of messages are well learned and interpreted by the receiver agent, and the communication protocol is properly established between the sender and receiver agent.

VI. Conclusion

This paper provides a novel message integration method ARMI to address the communication issues in multi-agent domain. Two specific integration functions are proposed to handle dynamics of agent number and learn effective interactions respectively. Serval experiments are conducted to verify the effectiveness of our proposal in a benchmark environment, and the results show that our method outperforms existing methods and achieves a much higher score in such environments.

Since we assume each agent feeds messages in sequence by the receiving order with the recurrent function, the forward and backward process is hard to be parallelized and sped up, which may cause intolerant running time especially for a large number of agents. Our further work may contain the simplification of integration functions, and more surveys on the imperfect messages because the transmission in real-time communication channel is usually very noisy.

References

[1] Zhaoqing Peng, T. Kato, H. Takahashi and T. Kinoshita, "Intelligent home security system using agent-based IoT devices," IEEE 4th Global Conference on Consumer Electronics, Osaka, 2015, pp. 313-314.

[2] Omidshafiei S, Pazis J, Amato C, How JP, Vian J, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," In International Conference on Machine Learning, pp. 2681-2690, 2017.

[3] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al., "Human-level control through deep reinforcement learning," Nature 518.7540 (2015): 529-533.

[4] Gupta, Jayesh K., Maxim Egorov, and Mykel Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," International Conference on Autonomous Agents and Multiagent Systems, pp. 66-83, 2017.

[5] Leibo JZ, Zambaldi V, Lanctot M, Marecki J, Graepel T, "Multi-agent Reinforcement Learning in Sequential Social Dilemmas," Proceedings of the 16th Conference on Autonomous Agents and Multi Agent Systems, 2017, pp. 464-473

[6] Hausknecht, Matthew, and Peter Stone, "Grounded Semantic Networks for Learning Shared Communication Protocols," Workshop on Deep Reinforcement Learning (NIPS), Barcelona, Spain, 2016.

[7] Lowe R, Wu Y, Tamar A, Harb J, Abbeel OP, Mordatch I, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," arXiv preprint arXiv:1706.02275, 2017.

[8] Peng P, Yuan Q, Wen Y, Yang Y, Tang Z, Long H, et al., "Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games," arXiv preprint arXiv:1703.10069, 2017.

[9] Foerster, Jakob, Yannis Assael, Nando de Freitas, and Shimon Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," In Advances in Neural Information Processing Systems, pp. 2137-2145, 2016.

[10] Sukhbaatar, Sainbayar, and Rob Fergus, "Learning multiagent communication with backpropagation," In Advances in Neural Information Processing Systems, pp. 2244-2252, 2016.

[11] Hoshen, Yedid, "Vain: Attentional multi-agent predictive modeling," In Advances in Neural Information Processing Systems, pp. 2698-2708, 2017

[12] Zhaoqing Peng, Libo Zhang, Tiejian Luo, "Learning to Communicate via Supervised Attentional Message Processing," In 31th International Conference on Computer Animation and Social Agents, 2018, in press.

[13] Sutton RS, McAllester DA, Singh SP, Mansour Y, "Policy gradient methods for reinforcement learning with function approximation," Advances in Neural Information Processing Systems, 2000.

[14] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, et al., "Asynchronous methods for deep reinforcement learning," In International Conference on Machine Learning, pp. 1928-1937, 2016.

[15] Graves, A, Mohamed, A. R., Hinton, G., "Speech recognition with deep recurrent neural networks," In Acoustics, speech and signal processing (icassp), pp. 6645-6649 , 2013

[16] Bahdanau, D., Cho, K., Bengio, Y., "Neural machine translation by jointly learning to align and translate," International Conference on Learning Representations(ICLR), pp. 1-15, 2014

[17] Kingma, Diederik P., and Jimmy Ba., "Adam: a method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014

[18] K. Noro, H. Tenmoto, A. Kamiya, "Signal learning with messages by reinforcement learning in multi-agent pursuit problem," Procedia Computer Science 35, pp. 233-240, 2014.

[19] Maaten, Laurens van der, and Geoffrey Hinton, "Visualizing data using t-SNE," Journal of Machine Learning Research, pp. 2579-2605, 2008