Attentive Relational State Representation in Decentralized Multiagent Reinforcement Learning

Xiangyu Liu and Ying Tan, *Senior Member, IEEE*

***Abstract*—In multiagent reinforcement learning (MARL), it is crucial for each agent to model the relation with its neighbors. Existing approaches usually resort to concatenate the features of multiple neighbors, fixing the size and the identity of the inputs. But these settings are inflexible and unscalable. In this article, we propose an attentive relational encoder (ARE), which is a novel scalable feedforward neural module, to attentionally aggregate an arbitrary-sized neighboring feature set for state representation in the decentralized MARL. The ARE actively selects the relevant information from the neighboring agents and is permutation invariant, computationally efficient, and flexible to interactive multiagent systems. Our method consistently outperforms the latest competing decentralized MARL methods in several multiagent tasks. In particular, it shows strong cooperative performance in challenging StarCraft micromanagement tasks and achieves over a 96% winning rate against the most difficult noncheating built-in artificial intelligence bots.**

***Index Terms*—Agent modeling, attentive relational encoder (ARE), decentralized learning, multiagent reinforcement learning (MARL), state representation.**

I. Introduction

MULTIAGENT learning has became a key toward artificial general intelligence (AGI) [1], [2]. Many real-world problems involve multiple agents with partial observability and limited communication [3]. Agents can extract useful features from neighboring agents to make optimal decisions and cooperation emerges in the group. Typical examples include the swarm robotics [4], collaborative filter [5], traffic signal control [6], and social network analysis [7].

For the learning protocol in the multiagent system, although we can adopt a centralized controller (also called as Joint Action Learner [8]) that receives the global rewards and determines the action for each agent, it may be costly to set up such a controller and sometimes it does not exist, such as intelligent transportation systems [9]. In this article, we focus on the decentralized protocol in multiagent reinforcement learning (MARL), where agents are connected by a time-varying topology structure, and they aggregate information from all their neighbors. This also promotes the scalability and robustness of multiagent systems.

However, when decentralized artificial intelligence (AI) agents are trained in an interactive environment, it is tricky to handle the state representation issue because the neighborhoods are highly flexible and scalable. Previous approaches typically represent the aggregated state by fixing the number of local team members and simply concatenating the information received from neighboring agents, as the input dimension must be invariant in neural-network policies and other machine-learning models. We argue that these formulations lack flexibility.

1) Concatenation leads to the linear increase in the input dimension, which scales poorly in large system size.
2) It cannot adapt to the change in population. Other individuals may join or quit the group of local interaction, leading to a varying dimension of representation.
3) The concatenation method requires the fixed order of input, which hinders the property of permutation invariance in the local group.
4) Incorporating all information without thinking is inefficient, as some features are worthless for decision making. Other approaches resort to some intuitive methods, e.g., averaging/max operation and histogram count. But all these methods are prone to losing valuable information to varying degrees and are problem specific.

In this case, an intriguing question is that can we design a general framework that fully utilizes the neighboring information to construct a stationary state representation, which is also invariant to the permutation and arbitrary size in multiagent learning?

In this article, we propose a compact neural-network-based architecture of representation, attentive relational encoder (ARE), for efficiently aggregating information by using an attention mechanism. The core idea is that an agent should know what other neighbors should pay attention to. ARE generates the attention weights pairwise and pools the local neighborhood to aggregate all information. We highlight that

Manuscript received October 16, 2019; revised January 14, 2020; accepted March 3, 2020. Date of publication March 27, 2020; date of current version January 10, 2022. This work was supported in part by the Science and Technology Innovation 2030—"New Generation Artificial Intelligence" Major Project under Grant 2018AAA0102301 and Grant 2018AAA0100302, in part by the National Natural Science Foundation of China under Grant 61673025 and Grant 61375119, and in part by the National Key Basic Research Development Plan (973 Plan) Project of China under Grant 2015CB352302. This article was recommended by Associate Editor H. M. Schwartz. *(Corresponding author: Ying Tan.)*

The authors are with the Key Laboratory of Machine Perception (Ministry of Education) and Department of Machine Intelligence, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: xiangyu.liu@pku.edu.cn; ytan@pku.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2020.2979803.

Digital Object Identifier 10.1109/TCYB.2020.2979803

by modeling the attention between different neighbors, ARE can actively select relevant information discriminately. By pooling, it constructs a unified state representation for learning policies. With this embedding, we condition the policy and train them simultaneously by deep reinforcement learning (DRL). The compact representation makes the learned policy robust to the changes in the multiagent system and also reduces the search space for the policy learning method. Enabling learning in this framework opens up the possibility of applying learning-based methods to multiagent interacting environments where neighbors need to be modeled explicitly and both the quantity and identity are changeable over time.

Our approach is fully decentralized in training and execution and makes no assumptions about communication channels, but each agent is given access to the state features of neighboring agents. We use a multiagent variant of Deep Q Network [10] to train the policy, which introduces the important sampling technique to address the nonstationary problem in multiagent learning [1], [11]. The effectiveness of the proposed representation framework is demonstrated using extensive empirical evaluations on three multiagent games, Catching, Spreading, and StarCraft II micromanagement scenarios, which explicitly require modeling the neighborhood. The proposed method achieves superior results compared against other baselines. To our knowledge, this is the first formalization of using the attentional aggregation method for state representation in decentralized multiagent learning.

This article is organized as follows. The related work and difficulties are presented in Section II. Our method ARE is presented in Section III. Description of the experimental evaluation scenarios and results are provided in Section IV. Finally, we summarize our findings in Section V.

II. Related Work

A. Multiagent Reinforcement Learning

Learning in the multiagent system is essentially more difficult than in the single-agent cases, as multiple agents not only interact with the environment but also with each other [1], [12], [13]. Directly applying the single-agent RL algorithms to the multiagent system as a whole is a natural approach, which is called the centralized MARL (also called joint action learner [8]). Centralized MARL models the interaction between agents by tightly coupling everything inside the model. Although feasible in execution, it suffers from the curse of dimensionality [1] due to the large-scale joint input space and action space. Thus, decentralized structure has more advantages toward scalability, robustness, and speedup [14]–[17].

In the decentralized MARL, a lot of attention has been given to the problem of modeling other agents [13]. In this article, we focus on how to aggregate the information collected from multiple agents, and we make a short survey on the information aggregation approaches in MARL.

B. Feature Aggregation in MARL

Concatenation: Concatenation is the simplest and most popular approach in multiagent RL. By concatenating other features, the augmented state contains all necessary information for decision making. MADDPG [18] constructs the critic for each agent by concatenating other agents' observations and actions, from which the agents can effectively train their actors. A centralized critic is also used in COMA [19], to implement difference reward by comparing with a counterfactual baseline. These methods are under the paradigm of *centralized learning with decentralized execution* [16], [18]–[21] which is inspired from DEC-POMDPs [22]. However, concatenation will make the dimension increase linearly, which scales poorly to the large size system. Also, the agent number and identities must be fixed, which is impractical in changeable environments.

Mean Embedding (ME): ME is a workable approach when dealing with a variable dimension problem. By calculating a mean representation, the output has an invariant dimension no matter how many agents are involved. CommNet [23] learns the communication model by rescaling the communication vector by the number of agents to aggregate information. Yang *et al.* [24] introduced the mean-field theory to MARL. The interactions within the group are approximated by those between a single agent and the average effect from the overall population or neighboring agents. Hüttenrauch *et al.* [25] also used the ME method to tackle the representation learning problem in the swarm system. The ME has the advantage of scalability, including dimension and permutation invariance. However, the mean computation is isotropic. The agent has no knowledge of each of its neighbors when pooling averagely around its local view, which may cause ambiguous estimation in many multiagent tasks where pairwise interactions are important for cooperative decision making.

Spatiotemporal Integration: Several related works aggregate information from other agents using some temporal or geometric structures. Peng *et al.* [26] used bidirectional RNNs to establish a communication protocol. Though ensuring the propagation of information, RNNs are not inherently symmetric since they process the input in a sequential manner, which usually maintains certain social conventions and roles by fixing the order of the agent. Hüttenrauch *et al.* [27] used a local histogram that encodes the neighborhood relation to transmit information, while the design of histogram is problem specific. Another spatial integration method is linear programming. An example is ORCA [28], which solves a low-dimensional linear program for reciprocal collision-free velocity computation. Although perfect in theory, it is time consuming for larger population computation.

Attention-Based Aggregator: The attention mechanism was originally proposed for natural language processing [29]. Recent research has applied an attention mechanism in multiagent modeling [30]–[32]. The attention mechanism can differentiate between agents and highlight the significant interacting agents. The closest work to ours is that of Hoshen [33], who proposed an interaction network (VAIN) modeling high-order interactions by the Kernel function while preserving the structure of the problem, which is in linear complexity in the number of vertices. However, the Kernel function is an independent embedding that cannot model complex interaction. Also, VAIN is experimented in predictive

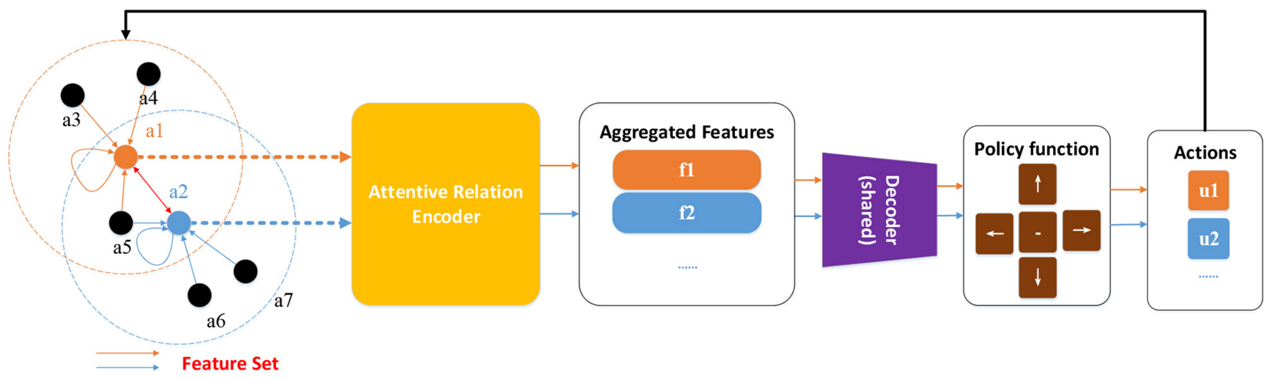


Fig. 1. Overview of the ARE in policy learning.

case, while the capability will be challenged in multiagent policy learning. We will conduct further analysis and comparisons between our methods and VAIN in Section IV. Other works focus on combining the attention mechanism with communication in MARL [30], [31].

As the topology of the multiagent system usually has an underlying graph structure [34], [35], many useful insights can be derived from the graph learning community [36], [37]. This article is partly inspired by the graph attention network (GAT) [7], which introduces an attention-based method to perform node classification of the graph-structured data. However, unlike the static graph considered in GATs, the topology of the multiagent system is changing over time as agents join or leave the local group. Also, GATs are applied for learning representation with a supervised label, while we focus on how to aggregate variable neighboring information to help cooperative decision making.

III. Proposed Method

In this section, we first introduce the problem background and then elaborate on the proposed ARE and the training method.

A. Background

We consider multiple agents operating in a partially-observable stochastic environment, modeled as a partially observable Markov decision process (POMDP). A stochastic game G is defined by a tuple $<S, U, P, r, Z, O, N, A, \gamma>$, where N agents, $A = \{a_1, a_2, \dots, a_N\}$, are in an interactive environment. $s \in S$ is the true state of the environment. At each time step, all agents simultaneously execute actions yielding a joint action $\mathbf{u} \in U$, then receive observation $\{o_i\}$ determined by observation function $O(s, u) : S \times U \to Z$, and rewards $r(s, u) : S \times U \to \mathbb{R}$ for profits. $P(s'|s, u) : S \times U \times S \to [0, 1]$ is the state transition probability function, and γ is the discount factor. We denote joint quantities over agents in bold, joint quantities other than a specific agent a with the subscript $-a$, i.e., $\mathbf{u} = [u_a, \mathbf{u}_{-a}]$. All agents take the goal of maximizing the discounted reward of r_t.

We consider the parameter-sharing decentralized control [38]. For simplicity and focusing on the representation problem, we assume that each agent can perceive the features of its neighbors in a local sensing range, and there is no other communication protocols.

An overview of the inference flow is illustrated in Fig. 1. All agents may behave in a possibly time-varying relation network $\mathcal{G}_t = (A, E_t)$, where E_t stands for the set of neighborhood links connecting agents at the time t. In an agent-centric view, we denote the neighboring feature set for the agent i as $\mathcal{N}_i = \{o_j\}_{j \in \mathcal{G}^i}$, where $\mathcal{G}^i$ is the subgraph of $\mathcal{G}$ induced by all agents adjacent to agent i (we leave out t for brevity). Therefore, our task is to design a function f with trainable weights θ to map the neighborhood feature set to a fixed size of aggregated high-level features, y: $y_i = f(\mathcal{N}_i, \theta)$, where $i \in 1, 2, \dots, N$. Our main contribution is this aggregation module, which corresponds to the ARE module as illustrated in Fig. 1 (in yellow) and will be elaborated in Section III-B. Then, these aggregated features are fed into a shared decoder to the control policy, which must select a discrete environment action $a \sim \pi_\theta$ in order to maximize the reward r. The training details will be presented in Section III-C.

B. Attentive Relational Encoder

We propose a compact neural-network-based architecture, ARE, to aggregate the information from neighboring agents group, whose size is changeable either due to the join or quit of agents. The basic idea of our ARE module is to learn an attention score for each neighbor's feature in the entire neighborhood set. The learnt score can be regarded as a credit that automatically selects useful latent features. For example, within a team of robots moving toward their separate goals, one robot may not care about some neighbors which are behind its moving direction although they are very close. The selected features are then pooled across all elements of the set to aggregate the information and finally served as the state representation for the subject agent.

Fig. 2 illustrates the main components of our approach and its execution flow. ARE consists of three encoders, E^f, E^c, and E^a, where $\{f, c, a\}$ stand for feature embedding, communication embedding, and attention embedding. In particular, as shown in Fig. 2, we first feed all the original features (self-feature as well as neighboring features) into two shared encoders E^f and E^c. E^f can be regarded as an intrinsic encoder, which keeps valuable latent features for constructing representation, while E^c is an extrinsic encoder, which reserves the

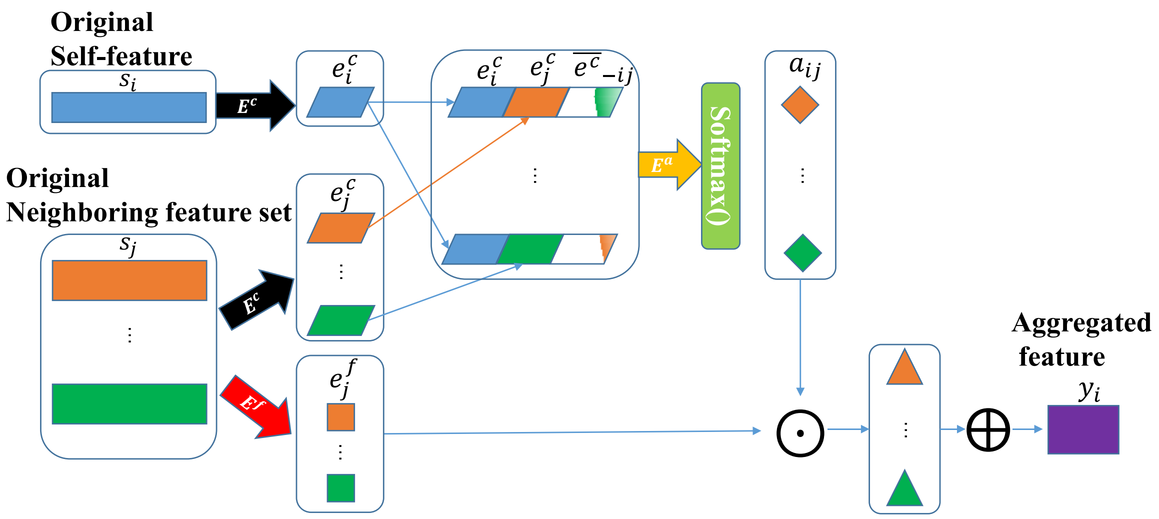


Fig. 2. ARE module on aggregating neighboring features.

crucial information for interactive relation modeling. Thus, we obtain two streams of latent vectors, e^f and e^c

$$e_i^f = E^f(o_i) \tag{1}$$
$$e_i^c = E^c(o_i). \tag{2}$$

Second, ARE computes attention scores using the latent vectors e^c for each corresponding neighboring agent through E^a, taking the self feature e_i^c, the corresponding neighbor's feature e_j^c, as well as an ME for other neighboring agents in $\mathcal{G}_i$ other than the agents i and j

$$e_{ij}^a = E^a\left(e_i^c, e_j^c, \overline{e^c}_{-ij}\right) \tag{3}$$
$$\overline{e^c}_{-ij} = \frac{\sum_{k\in\mathcal{G}_i-\{i,j\}} e_k^c}{\|\mathcal{G}_i - \{i,j\}\|}. \tag{4}$$

It is worth emphasizing that the self feature is also included in the neighboring feature set in ARE to evaluate the attention score for each agent itself. We will discuss the efficacy of this setting in Section IV-B4.

We add another channel of ME $\overline{e^c}_{-ij}$ besides pairwise features, in order to model the other neighbors' effect on the pairwise interaction. The output of the function E^a is a set of learnt attention activations $\{e_{ij}^a\}_{j\in\mathcal{G}_i}$. This procedure is similar to the query-key system [39]

$$e_{ij} \propto \phi\left(e_i^T W_k^T W_q e_j\right) \tag{5}$$

where each sender broadcasts a key transformed by W_k, while the receiver broadcasts a query transformed by W_q. The multiplication of these two parts interprets the relevance or utility of the message. However, we implement this by a neural layer E^a, where the high-level hidden state in neural net can model more abundant interactions between two agents than the query-key system, and generate the attention scores for aggregation.

Third, the learnt attention activations are normalized across the neighborhood set computing a set of attention weights $\overrightarrow{a_i} = \{a_{ij}\}_{j\in\mathcal{G}_i}$. We choose *softmax* as the normalization operation, so the attention weight for the jth neighboring feature is

$$a_{ij} = \frac{\exp\left(e_{ij}^a\right)}{\sum_{k\in\mathcal{G}_i}\exp\left(e_{ik}^a\right)}. \tag{6}$$

Subsequently, the computed attention weights are multiplied by their corresponding intrinsic latent features in e^f, generating a new set of deep weighted features. Finally, these weighted features are pooled by summing up across the neighborhood set, producing a fixed size of aggregated features which are then fed into a shared decoder to the downstream control policy, as illustrated in Fig. 1

$$y_i = \sum_j a_{ij} e_j^f \tag{7}$$
$$\pi_i = \text{decoder}(y_i). \tag{8}$$

In essence, as the weighted features can be parallelly computed and pooled, the output of the ARE module y_i is permutation invariant with regard to the input order. We formalize this property in Lemma 1.

Lemma 1 (Permutation Invariance): The ARE aggregation process (1)–(7) can be abstracted as follows:

$$\left[y_1,\ldots,y_k,\ldots,y_N\right] = f(o_1,\ldots,o_k,\ldots,o_N,\theta). \tag{9}$$

Then, for any permutation Φ, f is permutation invariant, i.e.,

$$f(o_1,\ldots,o_k,\ldots,o_N,\theta) = f\left(o_{\Phi(1)},\ldots,o_{\Phi(k)},\ldots,o_{\Phi(N)},\theta\right). \tag{10}$$

The proof of the lemma is in Appendix A.

We here highlight the specific form of the attention weight. In (3), the attention embedding is generated in the scalar value form. To model complex interaction, we can design e_{ij}^a as vector. Therefore, the attention score a_{ij} in (6) is also vector and (7) is revised

$$y_i = \sum_j \left(a_{ij}\cdot W_a\right) \odot e_j^f \tag{11}$$

where we first unify the dimension by multiplying a matrix W_a, then do the *Hadamard product* with e_j^f. For simplicity, we set the dimension of the attention vector a_{ij} to be the same with e_j^f, thus W_a becomes an identity matrix, and can be ignored.

Design Discussion: In terms of the flexibility in multiagent state representation learning, the ARE architecture is designed with the following desirable properties and advantages over existing approaches.

1) *Computational Efficiency:* ARE is computationally high efficient since all operations are parallelizable across the neighboring pairs and all modules are shared.
2) *Quantity Invariance:* Although the size of the neighboring feature set can be arbitrary, the output representation is still irrelevant as sum pooling [in (7)] is utilized. This property makes ARE scalable to the changeable and dynamic interactive environments.
3) *Permutation Invariance:* See Lemma 1.
4) *Differentiation Ability:* Our method is capable of differentiating the utility of multiple neighbors. By feeding each neighbor's feature together with self feature to the attention module and applying the attention mechanism on these features, ARE is able to attach importance to more relevant neighbors' features.

Our motivation is that whereas the original aggregation problem may be stated in a finite-dimensional feature space, it often happens that it is nontrivial to discriminate the relation between the elements in the set, and it is infeasible to do aggregation operations in that space (like averaging, concatenation, linear program, etc.). For this reason, we propose that the original feature space can be mapped into a much higher dimensional space using neural network, presumably making the aggregation easier and reasonable in that space, e.g., by sum pooling. Furthermore, we introduce the attention mechanism to play the role in improving the aggregation process by automatically selecting useful latent features across the set. We will analyze the learned attention and its relation to the actual behavior in Section IV-B.

C. Training: Multiagent Deep Q-Network

The ARE module is trained end-to-end by reinforcement learning. The learning algorithm is mainly based on Deep Q-Learning. Q-Learning [40] is a model-free, off-policy algorithm which learns the state–action value function by TD-targets. DQN [10] uses deep neural network for function approximator. While training, it samples from a replay buffer to eliminate the dependency between sequence data and bootstraps the immediate reward plus the expected future reward from the next state with a separate target model

$$\phi \leftarrow \phi + \alpha \sum_i \nabla_\phi Q_\phi(o_i, a_i)\big[Y_i - Q_\phi(o_i, a_i)\big] \tag{12}$$

$$Y_i = r(o_i, a_i) + \gamma Q_{\phi'}\left(o_i', \operatorname*{argmax}_{a'} Q_\phi\big(o_i', a_i'\big)\right) \tag{13}$$

where α is the learning rate and Q_ϕ is a parameterized model for Q function with the learning parameters ϕ, which estimates the state–action value function. In the ARE framework, this Q model corresponds to the decoder to the control policy, which takes in the upstream aggregated feature y_i generated by (11) and computes Q values for each action by a forward run. Here, (13) is a form of Double Q-learning [41], a more stable version considering overestimation, in which the current network ϕ is used to evaluate action and the target network ϕ' is used to evaluate value. Every C updates, the current Q network is cloned to update the target network. For the exploration scheme in RL, we also adopt the ϵ-*greedy* strategy [10]. The behavior policy follows the greedy policy with probability $1-\epsilon$ and selects a random action with probability ϵ.

However, several challenges will arise when adapting DQN methods to the multiagent environment, including the scalability (curse of dimensionality) for state representation, and nonstationary in learning [1]. The ARE structure is proposed to address the scalability problem. For the nonstationary problem, it becomes more serious when using DQN in decentralized training, as the replay buffer no longer reflects the dynamics when an experience data being sampled. Therefore, we borrow the ideas from [11] to tackle this difficulty, which used importance sampling. *Importance sampling* corrects the bias when gathering the off-environment data from experience replay. So we augment the data tuple in replay buffer by adding the policy distribution (sampling probability for the actions) during exploration. Then, (12) is modified as follows:

$$\phi \leftarrow \phi + \alpha \sum_i \frac{\pi_{-a_i}^{t_r}\big(\mathbf{u}_{-a_i}|s\big)}{\pi_{-a_i}^{t_i}\big(\mathbf{u}_{-a_i}|s\big)} \nabla_\phi Q_\phi(o_i, a_i)\big[Y_i - Q_\phi(o_i, a_i)\big] \tag{14}$$

in which t_r and t_i are the time of replay and the time of collection, respectively. The importance weight is calculated by $\pi_{-a}^t(\mathbf{u}_{-a}|s) = \prod_{j\in -a} \pi_j(u_j|o)$. This helps to disambiguate the sampled data from the replay buffer and correct the bias for the update direction. In addition, the update of ARE parameters θ also follows (14), as the Q network and ARE are fully differentiable and can be trained by Q-learning signal end to end.

D. Implementation Details

For completeness, the detail of the proposed method is shown in Algorithm 1. The encoders E^f, E^c, and E^a are shared and parameterized by fully connected layers followed by nonlinear activation functions. The decoder to the control policy is implemented by the dueling architecture [42], where the value and advantage streams both have a fully connected layer and the final hidden layers of the value stream having one output and the advantage as many outputs as there are valid actions. The exponential linear unit (ELU) [43] is inserted between all adjacent layers. We also use the Double Q formulation [41] and prioritized experience replay [44] to improve the training performance. The learning algorithm is based on (13) and (14) and demonstrated in Section III-C.

To make the computation more efficient with the experience replay in DQN, we augment the replay data with an adjacent matrix $I_t \in \mathbb{R}^{N\times N}$ for each time step

$$I_t = \begin{bmatrix} 1 & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1N} \\ \alpha_{21} & 1 & \alpha_{23} & \dots & \alpha_{2N} \\ \dots & \dots & \dots & \dots & \dots \\ \alpha_{N1} & \alpha_{N2} & \alpha_{N3} & \dots & 1 \end{bmatrix} \tag{15}$$

Algorithm 1 ARE-Based Multiagent Decentralized DQN

```
Input: N agents; initial values of the encoder/decoder and their target network parameters: $\theta : \{E^f, E^c, E^a\}$, $\phi$, $\theta'$, $\phi'$; $\epsilon$ for exploration; empty experience replay $\mathcal{D}$; R rounds for multihop aggregation; target network update frequency $C$
1:  for $episode = 1$ to $M$ do
2:    Update $\epsilon-$greedy for exploration
3:    Receive initial observation set $O = \{o_i\}$
4:    for $t = 1$ to $max - episode - length$ do
5:      for $round = 1$ to $R$ do
6:        for each agent $i$ do
7:          Compute intrinsic feature $e^f$, and extrinsic feature $e^c$ based on Eq. (1)(2)
8:          Receive neighboring feature set $\{e^c_j\}_{j \in \mathcal{G}^i}$
9:          for each neighbor $j$ do
10:           Compute its attention score $a_{ij}$ based on Eq. (3)(4)(6)
11:         end for
12:         Compute the aggregated feature $y_i$ based on Eq. (11)
13:       end for
14:     end for
15:     Select action with the current policy $u_i = \text{argmax}_u Q_\phi(y_i, u)$ and $\epsilon$-greedy exploration
16:     Execute actions $\mathbf{u} = \{u_1, ..., u_N\}$ and observe reward $r$ and new observation set $O' = \{o'_i\}$
17:     Store $(o_i, u_i, r, o'_i)$ and $\{\pi^{t_i}_{-i}(u_j|o_j)\}_{j \in \mathcal{G}^i}$ in replay buffer $\mathcal{D}$
18:     $O \leftarrow O'$
19:   end for
20:   Sample a random minibatch of $S$ samples $(o_i, u_i, r, o'_i)$ and $\{\pi^{t_i}_{-i}(u_j|o_j)\}_{j \in \mathcal{G}^i}$ from $\mathcal{D}$
21:   Update $\theta$ and $\phi$ based on Eq. (13)(14)
22:   Every $C$ steps reset the target network $\theta' = \theta, \phi' = \phi$
23: end for
```

where α_{ij} indicates whether the agents i and j are neighbors at the time t. Therefore, we can simply multiply the neighboring feature set from all agents with this adjacent matrix. Besides, we implement *multihops* layers [45] by running multiple rounds of the aggregation procedure (lines 6–13 of Algorithm 1), until getting a final aggregated state representation for the policy decoder. In this way, each agent will have the potential to model high-order neighbors' interactions. We will discuss these settings in Section IV. The details of the complexity analysis of Algorithm 1 are discussed in Appendix B.

IV. Experiments

To justify the effectiveness of the proposed architecture, we first conduct experiments on two toy multiagent tasks: 1) *Multiagent Catching Game* and 2) *Multiagent Spreading Game*. Both tasks require multiple agents to interact with each other to achieve certain goals. Finally, we evaluate ARE on a realistic video game: StarCraft II micromanagement, where a group of agents trying to defeat the enemies controlled by the built-in bot.

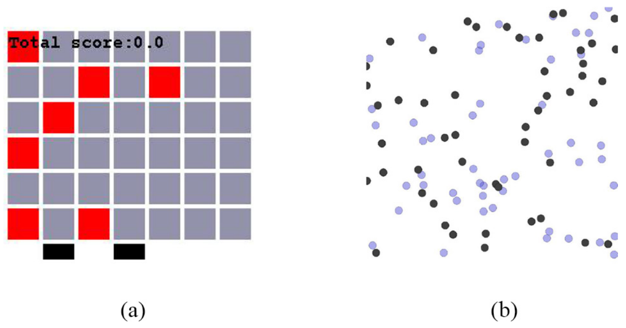


Fig. 3. Toy problems. (a) Catching Game. The two black paddles are trying to collect the red balls. (b) Spreading Game. Fifty agents in purple are trying to cover 50 black landmarks.

A. Baselines

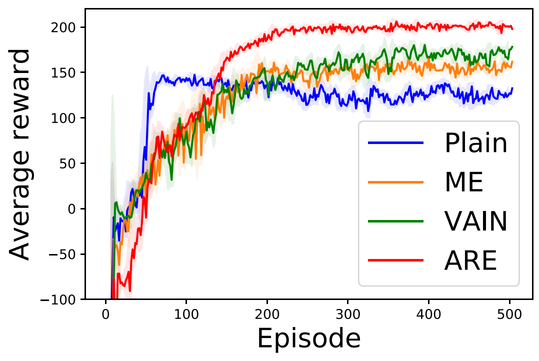
We describe three baseline models to compare against our method. Note that we only consider decentralized learning protocol.

Plain: A simple baseline is where the agent only uses self feature without considering neighboring agents. This is also known as independent Q-learning (IQL) [46]. We can write the output as $y_i = \theta(x_i)$. The advantage of this plain model is light, flexible, and naturally suited for partially observable settings, but it is not able to coordinate with other agents.

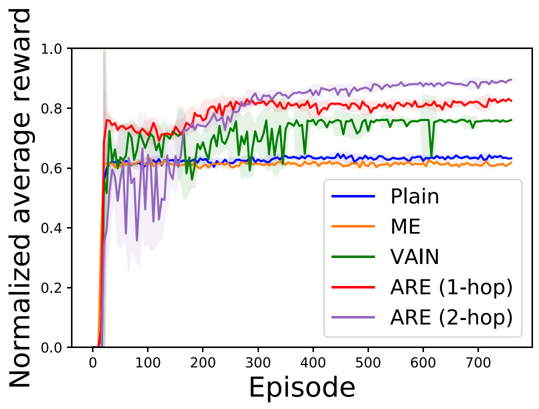
ME: As discussed in Section II-B, ME [23], [25] tries to obtain an integrated representation vector for each agent by averaged pooling over all neighbors' features. The averaged feature is then concatenated with self feature. The output can be written as $y_i = \theta(\sum_{j \neq i} \psi_{\text{me}}(x_j), \phi(x_i))$. Although it is permutation invariant, the mean computation ignores the difference between multiple neighbors. Especially in the non-additive cases, agent must explicitly model each neighbor's influence.

VAIN: VAIN [33] is an attention-based encoding architecture for predictive modeling of multiagent systems. VAIN learns a communication vector and in addition, an attention vector for each agent. For every interaction pair, the attention weights are constructed by a Kernel function. The output is given by $y_i = \theta(\sum_{j \neq i} e^{|a_i - a_j|^2} \psi_{\text{vain}}(x_j), \phi(x_i))$. We argue that the Kernel function is an independent embedding that cannot model complex interaction. In multiagent tasks, agent often first looks at its neighbor's status and then decides how much importance to attach. Also, VAIN is experimented in predictive case, while the capability will be challenged in multiagent policy learning.

All the baseline architectures and ARE are trained by the multiagent DQN, as described in Section III-C. We use a discount factor of 0.99, L2 regularization $\lambda = 5e^{-4}$, gradient clip, and an Adam optimizer. The hidden layer dimensions of the encoder and attention in ARE are $(64, 32)$ for Catching, $(64, 64)$ for Spreading, and $(128, 64)$ for SC2 combat tasks. Unless stated otherwise, all results presented are the average performance across 20 random simulation runs.



(a)



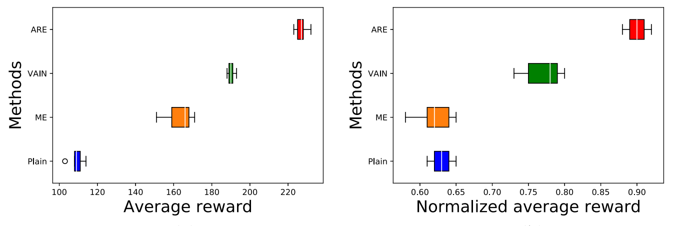
(b)

Fig. 4. Performance on two toy test problems. (a) Training reward on Catching Game by Plain (blue), ME (orange), VAIN (green), and ARE (red). (b) Training reward on Spreading Game by Plain (blue), ME (orange), VAIN (green), 1-hop ARE (red), and 2-hop ARE (purple).

B. Multiagent Catching Game

1) Environment Description: We first test on a simple task, *Multiagent Catching Game*, which is inspired by the game *Catch* introduced in [47]. This game is played on a screen of binary pixels and the goal is to move a paddle to catch balls that are dropped from the top of the screen. We extend this game to the multiagent settings, as shown in Fig. 3(a), where N paddles (in black) need to coordinate with each other to catch as many balls (in red) as possible. In this game, each agent has to explicitly consider its neighbors to make optimal decisions, catching more balls for the team. Taking illustration in Fig. 3(a) as an example, if the left agent observes two targets in each side of its view, while also another agent on its right side, it may choose to move left, allowing the neighboring agent to catch its only target. Only in this plan, the overall reward is optimal. See Appendix C-A for details on observation definition, reward structure, action space, and training.

2) Training Performance: We train ARE and the baselines with eight agents ($N = 8$). The game ends after 100 steps. The learning curve of 500 episodes is plotted in Fig. 4(a), in terms of team reward in each episode. The boxplot of the final performance is also illustrated in Fig. 5(a). As the figures show, all methods which explicitly model the neighborhood obtain higher team reward than Plain representation. This demonstrates that neighboring information is a critical part of higher



(a) (b)

Fig. 5. Comparison of the final performance via boxplot obtained by Plain (blue), ME (orange), VAIN (green), and ARE (red) on two toy test problems. (a) Catching Game. (b) Spreading Game.

TABLE I
COMPARISON BETWEEN BASELINES ON PERFORMANCE METRICS IN CATCHING GAME. [p_1 AND p_2 ARE DEFINED IN (16) AND (17)]

Methods	Metrics: % of collection	p_1	p_2
Plain	76.4%	0.701	0.201
ME	85.1%	0.831	0.149
VAIN	87.0%	0.841	0.133
ARE	**90.6%**	**0.989**	**0.128**

scores in this task. However, the Plain method converges more quickly, mainly due to its smaller state space. During simulation, we notice that Plain agents usually obtain overlapped in the same position, which indicates they simply learn to pursue their nearest target. Attention-based representations (VAIN and ARE) are apparently better than ME. This confirms that attaching different importance on different neighbors is crucial for coordinated decision making in frequently interactive environment. Our method outperforms the VAIN baseline by a clear margin, due to the different mode of generating attentional weights. As demonstrated in Section IV-A, VAIN models the interaction between two agents by independent Kernel function: $e^{\|a_i-a_j\|^2}$, while our method uses more expressive encoder $E^a(a_i, a_j, \bar{a}_{-ij})$. The hidden state can model the interaction by investigating self feature and neighbor-feature concurrently.

3) Coordination Effect: We further test the coordination performance by quantifying two measurements, p_1 and p_2. One obvious measurement is the percentage of view range, which is the ratio between the group observation space and the global space

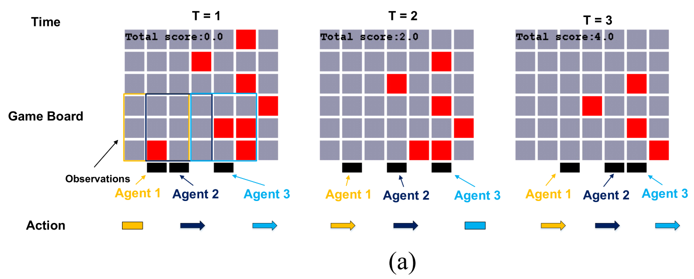
$$p_1 = \frac{\bigcup_{i\in A} |O_i|}{|S|}. \tag{16}$$

The other measurement p_2 is the ratio between the average amount of targets collected per agent (G_i) and the total amount of targets collected (G_{game}), in one episode

$$p_2 = \frac{E_{i\in A}[G_i]}{G_{\text{game}}}. \tag{17}$$

Note that G_{game} is not the simple summation of G_i, because there is a possibility that multiple agents collect the same target. Therefore, for a better policy, p_2 should be minimized. We also notice the best performance for p_2 should be $(1/|N|)$. However, it cannot be guaranteed that an arbitrary target is always accessible to a "free" agent.

We present the quantitative results of $p1$ and $p2$, and also the average percentage of targets collected in one episode,



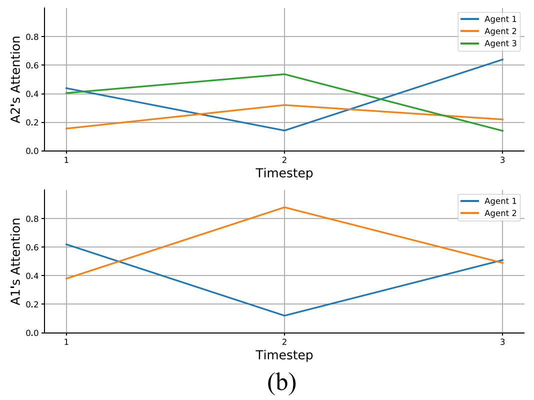


Fig. 6. Illustration of learned attention in the ARE module. (a) Three sampled screenshot of the environment. The first row is the time step. The second line is the game board. Each agent has 3 × 3 observation. The third line is the action generated by the trained policy. (b) Attention weights of Agent 2 and Agent 1 for each time step. (Note: Agent 1 is only adjacent to Agent 2 in all the three steps.)

in Table I. We notice that these metrics are consistent with the training performance. Our method outperforms in both $p1$ and $p2$. As the game randomly generates targets, agents learn to be apart from each other to maximize the coverage, in order to collect more targets and avoid getting into the same area. This is surprising because we did not explicitly encourage the agents to be separated, and by modeling pairwise attention, our architecture learns this strategy for mutual benefits.

4) Attention Analysis: We further try to gain more insights into the learned attention of our model and its relation with the actual behavior. We present an example scene occupied by three agents, whose observation range is 3 × 3 pixels, in Fig. 6(a). We implement a scalar attention variant of ARE and plot the attention weights of Agent 2 and Agent 1 for each time step, in Fig. 6(b). Interestingly, we first notice that the agent may not make much of his attention to itself. For example, in $T = 1$ for Agent 2, after checking its two neighbors' observation, it "realizes" that its only target is shared with Agent 1, while Agent 3 may struggle to catch many more targets. So it chooses to "concern" more on its neighbors, thus the attention weight to itself is the lowest, and it moves right. Furthermore, the attention weight of a neighboring pair is not symmetric. In $T = 2$, Agent 1 has no targets in its view and attaches more attention to Agent 2. In contrast, for Agent 2, it is busy at the dropping target on its right side, and nearly ignores Agent 1's information after comparing its features.

TABLE II
COMPARISON BETWEEN BASELINES ON PERFORMANCE METRICS IN SPREADING GAME

Methods	**Metrics** reward	collisions	landmarks (%)
Plain	0.63	5.02	92.5%
ME (1-hop)	0.62	**0.06**	36.4%
ME (2-hop)	0.65	**0.05**	36.3%
VAIN (1-hop)	0.71	1.82	98.1%
VAIN (2-hop)	0.76	1.80	97.2%
ARE (1-hop)	**0.83**	1.86	**98.5%**
ARE (2-hop)	**0.90**	**1.20**	**98.3%**

C. Multiagent Spreading Game

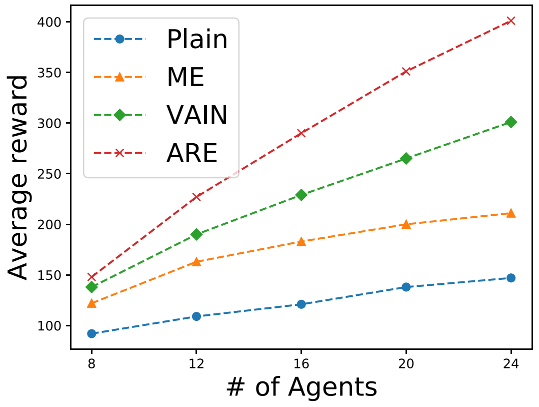
1) Environment Description: Multiagent Spreading Game is a more complex continuous state space scenario based on the multiagent particle environment [18], where N agents must reach N landmarks, as illustrated in Fig. 3(b). Each agent should avoid collision with other moving agents while making progress toward its goal. See Appendix C-B for details on observation definition, reward structure, and action space.

2) Training Performance: We trained ARE and the baselines with 50 agents ($N = 50$). Fig. 4(b) shows the learning curve, and Fig. 5(b) illustrates the boxplot of the final performance. These results are shown in terms of the 0–1 normalized mean score for each game. Our model clearly outperforms the other three baselines. Table II shows the average mean reward, number of collisions, and percentage of occupied landmarks at the end of the random test game. We also add 2-hop variants to the comparison. Although Plain agents cover most of the landmarks, it leads to many collisions. This confirms that Plain agents only learn to pursuit the landmark aggressively, which is reasonable as we did not provide other agents' state to the input. ME agents (both 1-hop and 2-hop) perform even worse than Plain representation. From the fewer collisions and lower percentage of occupied landmarks in Table II, we make sure that ME agents keep still and show conservative strategy. This is mainly because the mean computation is isotropic and incurs the loss of important information that could help cooperative decision making. The ME agent has no knowledge of the velocity field of each of its neighbors and only learns to keep still to avoid the emergent collision. Therefore, we draw a conclusion that ME is more appropriate for the symmetric tasks, where the interactions between agents can be accumulated in an additive way, like pattern formation in multirobot system [48], pursuit evasion and rendezvous in swarm system [25].

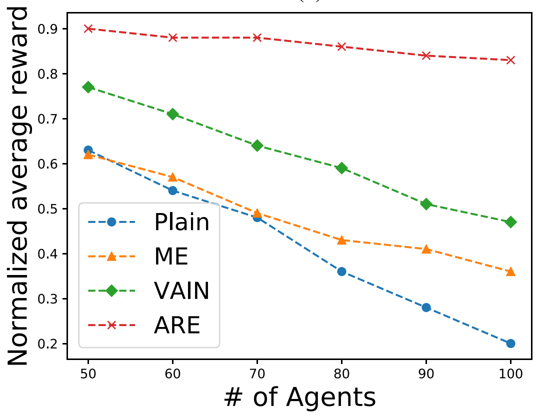
The VAIN architecture performs close to 1-hop ARE. As expected, we find that the 2-hop structure of ARE achieves better performance than the 1-hop structure. One possible reason is that by aggregating second-order neighbors' features, the control policy is able to condition on the entire state representation of both the central agent and the neighboring agents. Knowing what its neighboring situation is, an agent can achieve a more elaborate plan.

D. Scaling Test

To investigate the scalability of ARE and the baselines, we directly use the trained models under the settings of $N = 8$



(a)



(b)

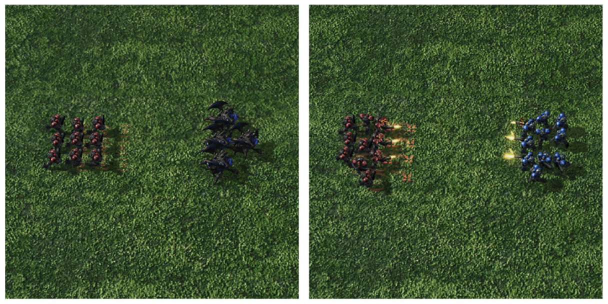
Fig. 7. Performance of baselines as a function of the number of agents during execution. Each data point is generated by evaluating a method with a fixed number of agents. (a) Average reward in Catching. (b) Normalized average reward in Spreading.

to $N = 20$ in a 40×40 map Catching task, and $N = 50$ to $N = 100$ in Spreading task. Fig. 7 shows the scaling performance.

We can see for all approaches in the Catching task, the insertion of new agents increases the episode reward. Our method outperforms the other three approaches with a higher reward growth rate. Despite a large amount of agents are present in the environment, they coordinate with each other to collect more targets, resulting in collective behaviors. For Spreading task, although the dense scenario may cause more collisions, our method practically maintains the performance, while the other baselines present the decline. This result is appealing. As we can train the decentralized policy in small-scale environments and it can generalize to large-scale multiagent system, like the swarm system, which has a great potential in real applications.

E. Combat Task: StarCraft II Micromanagement

1) Environment Description: StarCraft is a well-known real-time strategy game in a complex, stochastic environment whose dynamics cannot be easily modeled [49]. *Micromanagement* refers to a subtask of the entire game, which involves a local combat battle between two groups. The player must control a group of units to move them around the map and defeat the enemies. Recently, SC2 has drawn a lot of attention to the reinforcement learning community [50]–[52].



(a) (b)

Fig. 8. Two SC2 micromanagement scenarios. Left: Nine Marines versus four Roaches; and Right: Ten Marines versus Ten Marines. ARE and all baselines control the left side group (in red).

TABLE III
COMPARISON BETWEEN BASELINES ON PERFORMANCE METRICS IN TWO SC2 COMBAT GAME. THE REWARD IS NORMALIZED EPISODE REWARD

Methods	9M vs. 4R		10M vs. 10M	
	% of Win Rate	Reward	% of Win Rate	Reward
Plain	46.4%	14.70	58.9%	13.91
ME	55.1%	15.31	56.8%	13.09
VAIN	83.0%	18.41	80.3%	15.22
ARE	**96.6%**	**19.02**	**98.4%**	**19.13**

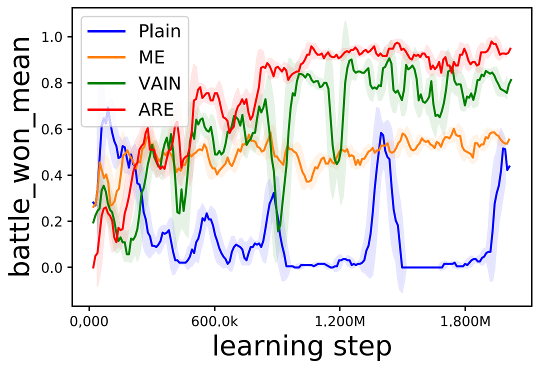
It offers a platform and opportunity to explore many challenging new frontiers.

We evaluate our method on two micromanagement tasks in SC2, nine Terran/Marines versus four Zerg/Roaches (9M_4R) and ten Terran/Marines versus ten Terran/Marines (10M_10M) (shown in Fig. 8 and we control the former group in "xx_xx"). These two tasks are a little hard even for human players to win without mastering some operation skills. We formulate this combat game into the multiagent decentralized learning problem, where the centralized player in the SC2 game is replaced by a group of AI agents, each assigned to one unit. Each agent observes local information on the unit it controls and must select from a set of actions, like move, stop, and attack to maximize its team reward, i.e., eliminate all enemies and win the game. See Appendix C-C for details on observation definition, reward structure, and action space.

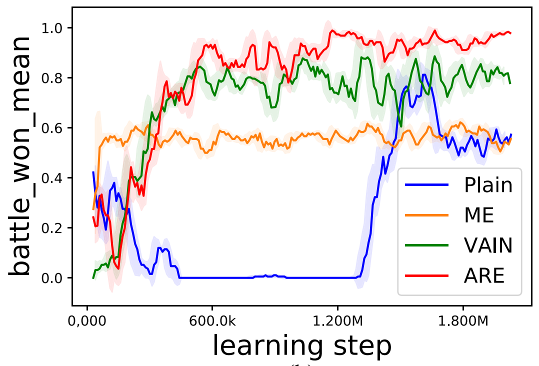
2) Training Performance: Fig. 9 demonstrates the improvement process of winning rates on these two tasks by evaluating 100 rounds during the phase of training. Table III demonstrates the performance of final trained models on both the winning rate and the episode reward. ARE achieves over 96% winning rate against the most difficult noncheating built-in AI bots. Compared with other baselines on this more challenging task than the toy problem aforementioned, our method clearly shows better performance, stable convergence, which highlights the importance of such a design facilitating the information aggregation and utility from neighbors.

3) Analysis of Learned Policy: In this part, we focus on the emerging multiagent high-level intelligent behaviors of our model. We capture a few screenshots when running our trained model in SC2 combat games. We briefly conduct a qualitative analysis of the learned policy, as shown in Fig. 10.

Focus Fire: We first observe that the team can efficiently attack the enemies by focusing each agent's fire on a particular



(a)



(b)

Fig. 9. Comparing winning rates of different methods on two combat scenarios. (a) Nine Marines versus four Roaches: Plain (blue), ME (orange), VAIN (green), and ARE (red). (b) Ten Marines versus ten Marines: Plain (blue), ME (orange), VAIN (green), and ARE (red).

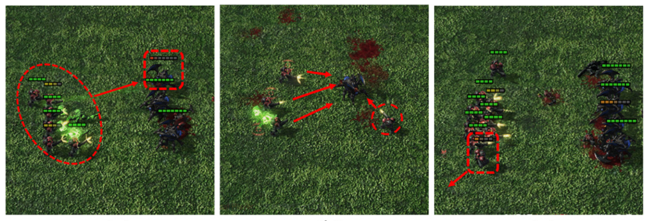

(a) (b) (c)

Fig. 10. Screenshots in the SC2 combat games. (a) Focus Fire. (b) Draw Fire. (c) Injured Back.

enemy unit. In Fig. 10(a), all alliances (left side) attack one dying enemy. Focus fire is a necessary tactic for attacking a group of players in combat tasks because the total threat from the enemy group has a positive correlation with the number of living units.

Draw Fire: Draw Fire is another high-level cooperation strategy that is often used by master players in real-time strategy games. As shown in Fig. 10(b), one agent (on the right side) draws the attention of the only enemy and other alliances focus their fires meanwhile. This plan minimizes the team-level damages until the only target is killed. Our model rapidly learns this tactic mainly because that by modeling the attention between its neighbors, each agent also models its relative position on the battlefield and knows the best response.

Injured Back: As shown in Fig. 10(c), the injured agent (downward) retreats for not being eliminated. This is reasonable because we give a negative reward when the agent is destroyed by enemies (as described in Appendix C-C). Much to our surprise, however, we continually observe that some injured agents retrieve back and go behind the frontier full-health alliance. They still keep outputting harms at the border of their firing range, which is a clever strategy for preserving power.

V. Discussion

A. Strengths

The main advantages of the proposed method are its simplicity and flexibility. ARE is invariant to the agents' number, permutation invariant to the input order. To some degree, aggregating neighbors' information is equivalent to enlarge the observation space or compute some sort of summary statistic about the state of other agents. But we provide a compact neural-network-based aggregation architecture with an attention mechanism, which is capable of differentiating the utility of multiple neighbors. ARE can be trained with any reinforcement learning algorithm and can be easily integrated with other network architecture. It is also scalable to the changeable and dynamic interactive environments.

B. Limitations

The proposed approach is based on the assumption that the agents are identically distributed and the learning process is independent. Hence, this framework is expected to be more suitable for independent games, in which the agents share a common environment and have similar abilities, and their contributions to the goal are additive. So we expect a degradation of performance for asymmetric heterogeneous agents games, or there are complex coupling effects between multiple agents. Also, we assume all neighbors' states are fully observable. Thus, our method cannot be used to intent recognition or agnostic opponent modeling [53]. It lacks the ability to infer other agents' goals. In a large-scale environment, how to model other agents' intentions or infer their goals is still a big challenge.

VI. Conclusion

ARE has been proposed for learning-aggregated state representation in multiagent decentralized learning. ARE is a compact neural-network-based architecture, which efficiently and selectively aggregates information by using an attention mechanism and can be trained end-to-end by reinforcement learning. Unlike the existing methods, it is permutation invariant, computationally efficient, and flexible to the arbitrary number of neighbors in the frequently interactive environment. It is remarkable that, in both synthetic multiagent tasks and real-time strategic StarCraft games, the ARE architecture led better performance over several baselines and also demonstrated coordinated decision making and strong scalability. We hope that this framework could shed some light on the future research of multiagent learning in large-scale scenarios and real-world problems.

Appendix A
Proof of Lemma 1

As the weighted features can be parallelly computed and pooled, the output of the ARE module y_i is permutation invariant with regard to the input order. We present the simple proof here.

Proof: Recall that the aggregation process can be abstracted as follows:

$$\left[y_1, y_2, \ldots, y_k, \ldots, y_N\right] = f(o_1, o_2, \ldots, o_k, \ldots, o_N, \theta). \quad (18)$$

In above (18), the kth entry of the output y is computed as follows:

$$\begin{aligned} y_k &= \sum_{i=1}^{N}\left(e_i^f \times a_{ki}\right) = \sum_{i=1}^{N}\left[e_i^f \times \frac{\exp\left(e_{ki}^a\right)}{\sum_{j=1}^{N}\exp\left(e_{kj}^a\right)}\right] \\ &= \sum_{i=1}^{N}\left[e_i^f \times \frac{\exp\left(\theta\left(e_k^c, e_i^c\right)\right)}{\sum_{j=1}^{N}\exp\left(\theta\left(e_k^c, e_j^c\right)\right)}\right] \\ &= \frac{\sum_{i=1}^{N}\left[e_i^f \times \exp\left(\theta\left(e_k^c, e_i^c\right)\right)\right]}{\sum_{j=1}^{N}\exp\left(\theta\left(e_k^c, e_j^c\right)\right)}. \end{aligned} \quad (19)$$

We leave out some unrelated elements for abbreviation. Note that in above (19), both the numerator and denominator are a summation of a permutation-equivalent term. Therefore, the aggregated feature y_k and also the combined vector y are permutation invariant to the neighborhood set $\{o_1, o_2, \ldots, o_k, \ldots, o_N\}$. ■

Appendix B
Complexity Analysis of ARE Model

ARE is computationally high efficient since all operations are parallelizable across the neighboring pairs and all modules are shared. The time complexity of a single ARE process can be expressed as $\mathcal{O}(N \cdot (t_1 + t_2) + |E| \cdot t_3)$, where N is the number of agent, $|E|$ stands for the number of all local connectivity edges, and t_1, t_2, and t_3 stand for the time consuming of running each ARE module, which correspond to (1)–(3). In terms of implementation, the individual feature computations are fully independent and can be parallelly computed. Thus, the total time complexity can be reduced to $\mathcal{O}(\max\{t_1, t_2\} + \max_i\{|E_i| \cdot t_3\})$. If we apply K-hop aggregation (multihop layers), the storage and parameter requirements will have a multiplication by a factor of K. We also leverage the sparse matrix operations to save the adjacent matrix (15), which reduce the storage complexity to linear in terms of the number of agents and local connectivity edges.

Appendix C
Details of Experimental Settings

A. Multiagent Catching Games

As illustrated in Fig. 3(a), if a ball is successfully catched by the paddle, a reward of 1 will be given, and −1 otherwise. For each time step, all balls move down one unit. The action space consists of {*move(left)*, *move(right)*, *stay*}. We assume that the game generates random balls from the top of the screen, but no more than the number of agents. Each agent's observations are local 5×5 pixels plus the position (x). To spread the neighbors' reward and achieve the coordination effect, we reshape the reward function as

$$r_i = n_i^{SC} + \lambda_1 \times \frac{n_i^{NC}}{\|\text{Neighbors}_i\|} - \lambda_2 \times \frac{n_i^{ND}}{\|\text{Neighbors}_i\|} \quad (20)$$

where n_i^{SC} is the number of self-collected targets, and n_i^{NC} and n_i^{ND} are the number of neighbors-collected targets and all dropped targets in the observation space, respectively. We normalize these two measures and multiply them with factors. To avoid dividing 0, the last two items will be calculated only in neighboring situations. In our experiments, we set $\lambda_1 = \lambda_2 = 0.2$. To maximize this return, each agent needs to catch the target of its own, remain targets for its neighbors, and minimize the missing targets. Reward shaping is a specialized topic in RL, but here we find these settings work well and leave the potential improvements for future work.

B. Multiagent Spreading Games

As illustrated in Fig. 3(b), agents are rewarded based on how far it is from each landmark and are penalized if they collide with other agents. In our experiments, we modify the game settings with that each agent will be given its goal position when initialized, and this game becomes a navigation task.

Each agent can observe other agents and landmarks in a limited sensing range. The discrete action set consists of {*stop*, *move(up)*, *move(down)*, *move(left)*, *move(right)*}. To enable efficient learning on obstacle avoidance, we add a small positive constant to the priorities of "crowded" data for proportional prioritized sampling in experience replay, as a large proportion of the replay data is the trivial situation (where no neighbors are around), especially in the early phases of training.

C. StarCraft II Micromanagement

We adopt the observation settings in SC2 for multiagent learning similar to existing work [19], [26]. To be specific, the local observation for an agent consists of move features (Boolean value, which indicates available actions), enemy features (relative position and health ratio for each enemy), and property features (health ratio and last action). The action consists of {*no-operation*, *stop*, *move(direction)*, *attack(id)*}. We define the reward as follows:

$$\begin{aligned} r_i^t = &-\Delta h_{j\in\mathcal{G}_a(i)}^t + \Delta h_{k\in\mathcal{G}_e(i)}^t \\ &- \left\|\text{Death}_{\mathcal{G}_a(i)}\right\|^t \times r_{\text{neg}} + \left\|\text{Death}_{\mathcal{G}_e(i)}\right\|^t \times r_{\text{pos}} \end{aligned} \quad (21)$$

where $\Delta h^t(\cdot) = h^t(\cdot) - h^{t-1}(\cdot)$ indicates the difference of the heath ratio between two consecutive time steps. Note that $\mathcal{G}_a(i)$ is the neighboring group of the controlled agents and $\mathcal{G}_e(i)$ is the enemy group. After calculating the health ratio change, we add another two terms encouraging more enemies destroyed and less teammates destroyed. We set the value $r_{\text{neg}} = r_{\text{pos}} = 10$.

The proposed method was tested using the PySC2 platform[1] by DeepMind [50].

References

[1] L. Busoniu, R. Babuska, and B. D. Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 156–172, Mar. 2008.

[2] D. Ye, M. Zhang, and A. V. Vasilakos, "A survey of self-organization mechanisms in multiagent systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 3, pp. 441–461, Mar. 2017.

[3] D. S. Bernstein, S. Zilberstein, and N. Immerman, "The complexity of decentralized control of Markov decision processes," in *Proc. 16th Conf. Uncertainty Artif. Intell. (UAI)*, Jun./Jul. 2000, pp. 32–37.

[4] Y. Tan and Z.-Y. Zheng, "Research advance in swarm robotics," *Defence Technol.*, vol. 9, no. 1, pp. 18–39, 2013.

[5] J. S. Breese, D. Heckerman, and C. M. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proc. 14th Conf. Uncertainty Artif. Intell. (UAI)*, Jul. 1998, pp. 43–52.

[6] T. Tan, F. Bao, Y. Deng, A. Jin, Q. Dai, and J. Wang, "Cooperative deep reinforcement learning for large-scale traffic grid signal control," *IEEE Trans. Cybern.*, early access.

[7] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[8] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proc. 15th Nat. Conf. Artif. Intell. 10th Innov. Appl. Artif. Intell. Conf. (AAAI)*, Jul. 1998, pp. 746–752.

[9] W. Li, "Notion of control-law module and modular framework of cooperative transportation using multiple nonholonomic robotic agents with physical rigid-formation-motion constraints," *IEEE Trans. Cybern.*, vol. 46, no. 5, pp. 1242–1248, May 2016.

[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[11] J. N. Foerster *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, Aug. 2017, pp. 1146–1155.

[12] Y. Shoham, R. Powers, and T. Grenager, "If multi-agent learning is the answer, what is the question?" *Artif. Intell.*, vol. 171, no. 7, pp. 365–377, 2007.

[13] S. V. Albrecht and P. Stone, "Autonomous agents modelling other agents: A comprehensive survey and open problems," *Artif. Intell.*, vol. 258, pp. 66–95, Feb. 2018.

[14] C. Guestrin, M. G. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *Proc. 19th Int. Conf. Mach. Learn. (ICML)*, Jul. 2002, pp. 227–234.

[15] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, Jul. 2018, pp. 5867–5876.

[16] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, Aug. 2017, pp. 2681–2690.

[17] S. Su, Z. Lin, and A. Garcia, "Distributed synchronization control of multiagent systems with unknown nonlinearities," *IEEE Trans. Cybern.*, vol. 46, no. 1, pp. 325–338, Jan. 2016.

[18] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor–critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst. 30th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2017, pp. 6382–6393.

[19] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. 32nd AAAI Conf. Artif. Intell.*, Feb. 2018, pp. 2974–2982.

[20] P. Sunehag *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proc. 17th Int. Conf. Auton. Agents MultiAgent Syst. (AAMAS)*, Stockholm, Sweden, Jul. 2018, pp. 2085–2087.

[21] H. Mao, Z. Gong, Y. Ni, and Z. Xiao, "ACCNET: Actor-coordinator-critic net 'learning-to-communicate' with deep multi-agent reinforcement learning," 2017. [Online]. Available: arXiv:1706.03235.

[22] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs* (Springer Briefs in Intelligent Systems). Cham, Switzerland: Springer, 2016.

[23] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning multiagent communication with backpropagation," in *Proc. Adv. Neural Inf. Process. Syst. 29th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2016, pp. 2244–2252.

[24] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, Jul. 2018, pp. 5567–5576.

[25] M. Hüttenrauch, A. Sosic, and G. Neumann, "Deep reinforcement learning for swarm systems," *J. Mach. Learn. Res.*, vol. 20, pp. 1–31, Feb. 2019.

[26] P. Peng *et al.*, "Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games," *CoRR*, vol. abs/1703.10069, 2017. [Online]. Available: http://arxiv.org/abs/1703.10069

[27] M. Hüttenrauch, A. Sosic, and G. Neumann, "Local communication protocols for learning complex swarm behaviors with deep reinforcement learning," in *Proc. 11th Int. Conf. Swarm Intell. (ANTS)*, Oct. 2018, pp. 71–83.

[28] J. van den Berg, S. J. Guy, M. C. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Robot. Res. 14th Int. Symp. (ISRR)*, Aug./Sep. 2009, pp. 3–19.

[29] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, May 2015.

[30] J. Jiang and Z. Lu, "Learning attentional communication for multi-agent cooperation," in *Proc. Adv. Neural Inf. Process. Syst. 31st Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2018, pp. 7265–7275.

[31] A. Das *et al.*, "TarMAC: Targeted multi-agent communication," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Jun. 2019, pp. 1538–1546.

[32] H. Mao, Z. Zhang, Z. Xiao, and Z. Gong, "Modelling the dynamic joint policy of teammates with attention multi-agent DDPG," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst. (AAMAS)*, Montreal, QC, Canada, May 2019, pp. 1108–1116.

[33] Y. Hoshen, "VAIN: Attentional multi-agent predictive modeling," in *Proc. Adv. Neural Inf. Process. Syst. 30th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2017, pp. 2698–2708.

[34] Z. Bu, Z. Wu, J. Cao, and Y. Jiang, "Local community mining on distributed and dynamic networks from a multiagent perspective," *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 986–999, Apr. 2016. [Online]. Available: https://doi.org/10.1109/TCYB.2015.2419263

[35] Z. Ji and H. Yu, "A new perspective to graphical characterization of multiagent controllability," *IEEE Trans. Cybern.*, vol. 47, no. 6, pp. 1471–1483, Jun. 2017. [Online]. Available: https://doi.org/10.1109/TCYB.2016.2549034

[36] F. Chen, Y. Wang, B. Wang, and C. J. Kuo, "Graph representation learning: A survey," *CoRR*, vol. abs/1909.00958, 2019. [Online]. Available: http://arxiv.org/abs/1909.00958

[37] Y. Liu, W. Wang, Y. Hu, J. Hao, X. Chen, and Y. Gao, "Multi-agent game abstraction via graph attention neural network," *CoRR*, vol. abs/1911.10715, 2019. [Online]. Available: http://arxiv.org/abs/1911.10715

[38] J. K. Gupta, M. Egorov, and M. J. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Proc. Auton. Agents Multiagent Syst. (AAMAS)*, May 2017, pp. 66–83.

[39] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. 30th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2017, pp. 5998–6008.

[40] C. J. C. H. Watkins and P. Dayan, "Technical note Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[41] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, Feb. 2016, pp. 2094–2100.

[42] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33nd Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, Jun. 2016, pp. 1995–2003.

[43] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUS)," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, May 2016.

[44] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016. [Online]. Available: http://arxiv.org/abs/1511.05952

[1]https://github.com/deepmind/pysc2

[45] N. K. Tran and C. Niederée, "Multihop attention networks for question answer matching," in *Proc. 41st Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, Ann Arbor, MI, USA, Jul. 2018, pp. 325–334.

[46] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, Jun. 1993, pp. 330–337.

[47] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *Proc. Adv. Neural Inf. Process. Syst. 27th Annu. Conf. Neural Inf. Process. Syst.*, Dec. 2014, pp. 2204–2212.

[48] Y. Chen and Z. Wang, "Formation control: A review and a new consideration," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Aug. 2005, pp. 3181–3186.

[49] S. Ontañón, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in StarCraft," *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 4, pp. 293–311, Dec. 2013.

[50] O. Vinyalso *et al.*, "StarCraft II: A new challenge for reinforcement learning," *CoRR*, vol. abs/1708.04782, 2017. [Online]. Available: http://arxiv.org/abs/1708.04782

[51] P. Sun *et al.*, "TStarBots: Defeating the cheating level Builtin AI in StarCraft II in the full game," *CoRR*, vol. abs/1809.07193, 2018. [Online]. Available: http://arxiv.org/abs/1809.07193

[52] D. Lee, H. Tang, J. O. Zhang, H. Xu, T. Darrell, and P. Abbeel, "Modular architecture for StarCraft II with deep reinforcement learning," in *Proc. 14th AAAI Conf. Artif. Intell. Interactive Digit. Entertainment (AIIDE)*, Nov. 2018, pp. 187–193.

[53] H. He and J. L. Boyd-Graber, "Opponent modeling in deep reinforcement learning," in *Proc. 33nd Int. Conf. Mach. Learn. (ICML)*, Jun. 2016, pp. 1804–1813.

Xiangyu Liu received the Bachelor of Engineering degree in intelligent science and technology from Nankai University, Tianjin, China, in 2015. He is currently pursuing the Ph.D. degree in computer science with the Key Laboratory of Machine Perception (Ministry of Education), Department of Machine Intelligence, EECS, Peking University, Beijing, China.

His research interests include multiagent reinforcement learning, swarm intelligence, and swarm robotics.

Ying Tan (Senior Member, IEEE) received the B.Eng., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1985, 1988, and 1997, respectively.

He is a Full Professor and a Ph.D. Advisor with the School of EECS, and the Director of Computational Intelligence Laboratory, Peking University, Beijing, China. He is the inventor of Fireworks Algorithm. He worked as a Professor with the Faculty of Design, Kyushu University, Fukuoka, Japan, in 2018, a Senior Research Fellow with Columbia University, New York, NY, USA, in 2017, and a Research Fellow with the Chinese University of Hong Kong, Hong Kong, in 1999 and from 2004 to 2005. His research interests include computational intelligence, swarm intelligence, deep neural networks, machine learning, data mining, and intelligent information processing for information security and financial prediction. He has published more than 350 papers in refereed journals and conferences in these areas, and authored/coauthored 12 books, including *Fireworks Algorithm* (Springer in 2015), and *GPU-Based Parallel Implementation of Swarm Intelligence Algorithm* (Morgan Kaufmann and Elsevier in 2016), and received five invention patents.

Prof. Tan won the Second-Class Natural Science Award of China in 2009 and the Second-Class Natural Science Award of Ministry of Education of China in 2019 and many best paper awards. He serves as the Editor-in-Chief for the *IASEI Transactions on Swarm Intelligence* and the *International Journal of Computational Intelligence and Pattern Recognition*, and an Associate Editor for the IEEE TRANSACTIONS ON CYBERNETICS, the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEM, *Neural Networks*, and the *International Journal of Swarm Intelligence Research*. He also served as an Editor of Springer's Lecture Notes on Computer Science for 40+ volumes, and a guest editors of several referred journals, including the IEEE/ACM TRANSACTIONS ON COMPUTATIONAL BIOLOGY AND BIOINFORMATICS, *Information Science*, *Neurocomputing*, *Natural Computing*, and *Swarm and Evolutionary Optimization*. He has been the Founding General Chair of the ICSI international conference series since 2010 and the DMBD conference series since 2016.