ACE: Cooperative Multi-agent Q-learning with Bidirectional Action-Dependency

Chuming Li[1, 2*], **Jie Liu**[2*], **Yinmin Zhang**[1, 2*], **Yuhong Wei**[3], **Yazhe Niu**[2, 3], **Yaodong Yang**[4†],
Yu Liu[2, 3†], **Wanli Ouyang**[1, 2]

[1] The University of Sydney, SenseTime Computer Vision Group, Australia, [2] Shanghai Artificial Intelligence Laboratory,
[3] SenseTime Group LTD, [4] Institute for AI, Peking University
chli3951@uni.sydney.edu.au, liujie@pjlab.org.cn, {yinmin.zhang, wanli.ouyang}@sydney.edu.au, {weiyuhong,
niuyazhe}@sensetime.com, yaodong.yang@pku.edu.cn, liuyuisanai@gmail.com,

Abstract

Multi-agent reinforcement learning (MARL) suffers from the non-stationarity problem, which is the ever-changing targets at every iteration when multiple agents update their policies at the same time. Starting from first principle, in this paper, we manage to solve the non-stationarity problem by proposing bidirectional action-dependent Q-learning (ACE). Central to the development of ACE is the sequential decision making process wherein only one agent is allowed to take action at one time. Within this process, each agent maximizes its value function given the actions taken by the preceding agents at the inference stage. In the learning phase, each agent minimizes the TD error that is dependent on how the subsequent agents have reacted to their chosen action. Given the design of bidirectional dependency, ACE effectively turns a multi-agent MDP into a single-agent MDP. We implement the ACE framework by identifying the proper network representation to formulate the action dependency, so that the sequential decision process is computed implicitly in one forward pass. To validate ACE, we compare it with strong baselines on two MARL benchmarks. Empirical experiments demonstrate that ACE outperforms the state-of-the-art algorithms on Google Research Football and StarCraft Multi-Agent Challenge by a large margin. In particular, on SMAC tasks, ACE achieves 100% success rate on almost all the hard and super hard maps. We further study extensive research problems regarding ACE, including extension, generalization and practicability. Code is made available to facilitate further research.

Introduction

Cooperative multi-agent reinforcement learning (MARL) aims to learn a good policy that controls multiple agents and maximizes the cumulative return in a given task. It has great potential in various real-world tasks, such as robot swarm control 2017, autonomous driving 2020; 2016 and multi-player games 2019; 2021. A major challenge of MARL is the complex joint action space. In multi-agent tasks, the joint action space increases exponentially with the number of agents. Hence, for the sake of scalability, existing MARL algorithms usually learn an individual policy to select the action for every single agent. In MARL algorithms, the reward signal is affected by other agents' behavior. However, the environment of multi-agent task is non-stationary 2019; 2021 to every single agent, where the policies of agents keep changing during the learning process. This non-stationary problem breaks the Markov assumption in single-agent RL algorithms and causes endless adaptation of multiple agents according to each other's change of policy. In value-based methods, the non-stationary problem shows up as that the value of the individual action can not be estimated accurately.

To solve the non-stationary problem, we introduce bidirectional action-dependency to estimate the action value of every single agent accurately. We cast multi-agent decision-making process as a sequential decision-making process, where only one agent makes a decision at a time. In this sequential process, the bidirectional action-dependency is embodied in two aspects. In the forward direction, the evaluation of an agent's action value is dependent on the preceding agents' actions in the decision-making sequence. While in the backward direction, the target to update an agent's action value is dependent on how subsequent agents react to the preceding actions. We formulate this bidirectional dependence by transforming a multi-agent Markov Decision Process (MMDP) 1994 into a single-agent Markov Decision Process (MDP), called sequentially expanded MDP (SE-MDP). In SE-MDP, a decision $\boldsymbol{a}^t$ based on a state s^t is expanded to multiple intermediate states $\left[s^t_{a_1}, ..., s^t_{a_{1:n}}\right]$, named SE-state. The SE-state $s^t_{a_{1:i}}$ is defined as the state s^t in the original MMDP along with the decisions $a_{1:i}$ made by the preceding agents. Only one agent makes a decision at each SE-state. After each agent makes the decision, the state transits to the next one, which includes the new decision. This transformation validates that the proposed bidirectional action-dependency does circumvent the non-stationary problem.

With the introduced bidirectional dependency, we propose a simple but powerful method, called *bidirectional ACtion-dEpendent deep Q-learning* (ACE). ACE is compatible with the abundant Q-learning methods for single-agent tasks, and naturally inherits their theoretical guarantee of convergence and performance. For practical implementation, we identify an efficient and effective network representation of the SE-state. We first generate the embeddings for all units in the task as well as the embeddings for their available actions. Then, we combine the embedding of each unit with the corresponding action embedding to construct the embedding for every SE-state. This design is quite efficient, because the em-

[*] Equal contribution

[†] Corresponding author

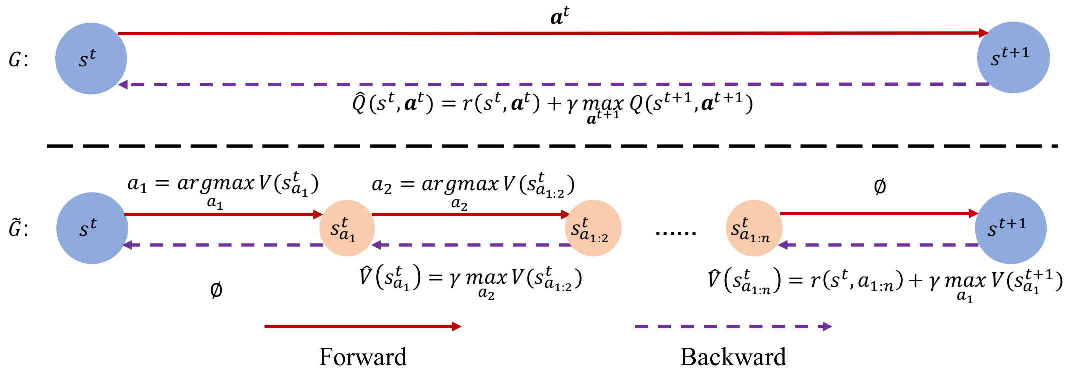

G:
s^t
$\boldsymbol{a}^t$
s^{t+1}
$\hat{Q}(s^t, \boldsymbol{a}^t) = r(s^t, \boldsymbol{a}^t) + \gamma \max_{\boldsymbol{a}^{t+1}} Q(s^{t+1}, \boldsymbol{a}^{t+1})$
$\tilde{G}$:
$a_1 = \underset{a_1}{argmax}\, V(s^t_{a_1})$
$a_2 = \underset{a_2}{argmax}\, V(s^t_{a_{1:2}})$
$s^t_{a_1}$
$s^t_{a_{1:2}}$
......
$s^t_{a_{1:n}}$
$\emptyset$
$\hat{V}(s^t_{a_1}) = \gamma \max_{a_2} V(s^t_{a_{1:2}})$
$\hat{V}(s^t_{a_{1:n}}) = r(s^t, a_{1:n}) + \gamma \max_{a_1} V(s^{t+1}_{a_1})$
Forward
Backward

Figure 1: Comparison between the original MMDP (above) and the transformed SE-MDP (below). A single transition in MMDP is expanded to n sequentially expanded states in SE-MDP.

beddings of all SE-states along a sequential decision-making process are constructed with additive combination among the same set of unit and action embeddings. This set is computed only once before every sequential decision-making process, and the additive combination brings in negligible cost. Moreover, an interaction-aware action embedding is developed to describe the interaction among units in the multi-agent task, which further improves the performance of ACE.

We evaluate the performance of ACE on both a toy case and complex cooperative tasks. In the toy case, ACE demonstrates its advantage in converging to the optimal policy against the popular value-factorization methods. Because it bridges the gap of the optimal actions between the joint and individual Q-function, which widely exists in value-factorization methods. For complex tasks, we choose two benchmark scenarios in Google Research Football (GRF) 2020 environment and eight micromanagement tasks in StarCraft Multi-Agent Challenge (SMAC) 2019. Empirical results show that ACE significantly outperforms the state-of-the-art algorithms on GRF, and achieves higher sample efficiency by up to 500%. On SMAC, ACE achieves 100% win rates in almost all the hard and super-hard maps. Other advantages of ACE are verified with comprehensive experiments, including generalization, extension and practicability. Surprisingly, ACE also indicates better generalization performance compared with other baselines when transferred to a new map with a different number of agents in SMAC.

Related Work

To solve the widespread cooperation tasks, many multi-agent reinforcement learning (MARL) algorithms have been proposed recently. According to the extent of centralization, these works can be divided into two categories, independent learning scheme and action-dependent learning scheme.

First, many works tend towards a fully independent learning scheme 2022, where agents make decisions with their independent value functions or policies. One typical category assigns independent actor to each agent by directly transferring the actor-critic methods to multi-agent scenarios 2018; 2017; 2021; 2021. Another line is value-based methods 1993; 2017; 2018; 2020; 2019; 2020; 2020; 2020. To avoid the non-stationary problem, they usually develop different factorized value functions following the IGM principle 2019, which requires that the individually optimal actions are consistent with the jointly optimal actions. We remark that existing value factorization methods following the IGM principle either suffer from the structural constraints, like VDN and QMIX, or introduce secondary components along with additional hyperparameters, like QTRAN, WQMIX and QPLEX. However, the optimal joint action often changes due to the discovery of a better policy, resulting in the mismatch between the optimal joint Q function and individual functions during training. This means that individual Q functions require more iterations to recover the satisfaction of IGM, and the policy explores the environment with sub-optimal actions, leading to low sample efficiency. To avoid the issues, this paper focuses on directly estimating the value of each action, rather than following the IGM principle to construct factorization function classes.

Second, the action-dependent learning scheme 2019; 2021a; 2022; 2021b; 2022; 2022; 2022 is more centralized. One perspective is action-dependent execution, where the agent makes decisions with dependency on other agents' actions. CGS 2022 proposes a graph generator to output a directed acyclic graph which describes the action dependency. Each node in the graph represents an agent whose policy is dependent on the action of agents on its parent nodes. However, each agent's decision is only dependent on part of the previous agents in the topological sort of the generated DAG, and the policy update is independent on the reaction of the subsequent agents. It means the non-stationary effect is not totally removed. In another perspective, action-dependency is introduced in policy update rather than execution. Multi-agent rollout algorithm 2019 and HAPPO 2021a follow a update to sequentially update the policy of each agent with the others fixed, thus avoiding the conflicting update directions of individual policy updates. This paradigm is an implicit rather than full action-dependency, because the policy does not explicitly depends on the actions of the preceding agents. As an extra difference, ACE is the first value-based MARL method that achieves remarkable performance following the action-dependent learning scheme. More elaborate discussion with relative works are referred to the Appendix.

Problem Formulation

In this paper we take Multi-agent Markov Decision Process (MMDP) 1994 to model cooperative multi-agent tasks. An MMDP is a tuple $\mathcal{G} = \langle \mathcal{S}, \mathcal{N}, \mathcal{A}, P, r, \gamma \rangle$, where $\mathcal{S}$

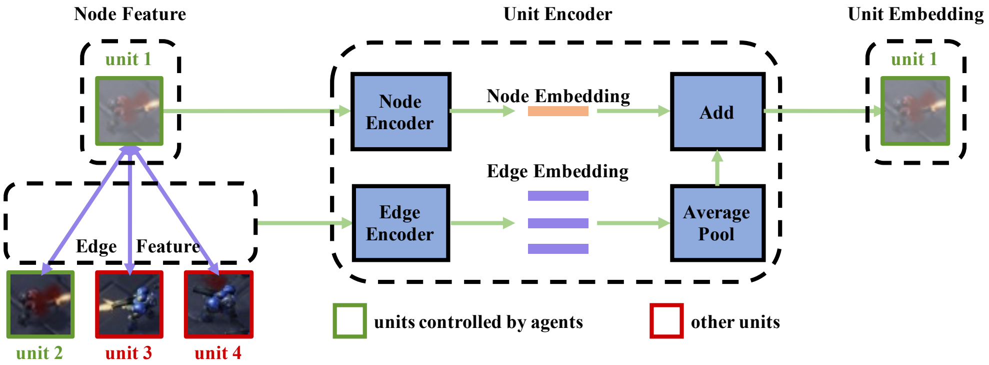


Figure 2: The schematic of the unit encoder. The node embedding is obtained by the node encoder, and the edge embedding (for the unit and its interacted units) is obtained from the edge encoder. The average-pooled edge embedding is added to the node embedding to provide unit embedding.

is the space of global state and $\mathcal{N}$ is the set of n agents. $\mathcal{A} \equiv \mathcal{A}_1 \times, ..., \times \mathcal{A}_n$ is the joint action space consisting of each agent's action space $\mathcal{A}_i$. At each step, the global state s is transformed to each agent i's input, and each agent i selects an action $a_i \in \mathcal{A}_i$. Then, with the joint action $\boldsymbol{a} = [a_1, ..., a_n]$ and the transition function $P\left(s' | s, \boldsymbol{a}\right)$, the process transits to the next state s' and returns a reward $r\left(s, \boldsymbol{a}\right)$. The target we consider is to learn an optimal policy $\boldsymbol{\pi}\left(\boldsymbol{a} \mid s\right)$ which maximizes the expected return $\mathcal{R} = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s^t, \boldsymbol{a^t}\right)\right]$.

Method

Bidirectional Action-Dependency

In this section, we consider a sequential decision-making scheme: all agents make decisions sequentially. The bidirectional action-dependency has two directions. In the forward direction, each agent's decision depends on the state and their preceding agents' actions. Inversely, in the backward direction, the update of the Q-value for an agent's action depends on how its successor reacts to the preceding actions.

We formalize this bidirectional dependency by transforming the original MMDP $\mathcal{G}$ into a single agent MDP $\widetilde{\mathcal{G}}$. In $\widetilde{\mathcal{G}}$, the state transits along the decision-making sequence. Specifically, a intermediate transition happens each time when a single agent in the sequence selects its action. The intermediate state is defined as the original state s^t along with the actions of the agents which have made their decisions, denoted as $s^t_{a_{1:i}}$. At each intermediate transition, an agent i receives its intermediate state $s^t_{a_{1:i-1}}$ and produces its action a_i, then the intermediate state intermediately transits to $s^t_{a_{1:i}}$ with a reward 0. After the last agent n makes decision and the intermediate state intermediately transits to $s^t_{a_{1:n}}$, a psuedo agent produces an empty action and the intermediate state transits from $s^t_{a_{1:n}}$ to s^{t+1}, with the reward $r\left(s^t, \boldsymbol{a}^t\right)$ defined in the original MMDP $\mathcal{G}$. With the above definition, a transition $\left(s^t, \boldsymbol{a}^t, r\left(s^t, \boldsymbol{a}^t\right), s^{t+1}\right)$ of $\mathcal{G}$ is expanded into a sequence of intermediate transitions $\left(s^t, a^t_1, 0, s^t_{a_1}\right), \left(s^t_{a_1}, a^t_2, 0, s^t_{a_{1:2}}\right), ..., (s^t_{a_{1:n-1}}, a_n, r(s^t, \boldsymbol{a}^t), s^{t+1})$ in $\widetilde{\mathcal{G}}$. We define $\widetilde{\mathcal{G}}$ as the sequential expansion of $\mathcal{G}$ and name this MDP as sequentially expanded MMDP (SE-MMDP). Similarly, we define the intermediate state $s_{a_{1:i}}$ as sequentially expanded state (SE-state), of which the space is represented by $\widetilde{\mathcal{S}}$.

As depicted in Figure 1, the formulation of SE-MDP validates that the bidirectional action-dependency does circumvent the non-stationary problem. In SE-MDP, the forward dependency is manifested in that the preceding actions are incorporated in the SE-state. It means the changeable behavior of the preceding agents are tracked in the value estimation of each SE-state. As for the backward dependency described by dashed lines in Figure 1, the target value of an agent's action a_i in the Bellman operator depends on its successor's reaction to the preceding actions, *i.e.,* the best selection of a_{i+1}, which also tracks the successor's behavior.

Bidirectional Action-Dependent Q-learning

The formulation of sequential expansion $\widetilde{\mathcal{G}}$ circumvents the non-stationary problem, which enables us to easily adopt different single-agent algorithms to solve $\widetilde{\mathcal{G}}$. Based on the formulation of sequential expansion, this section introduces the proposed bidirectional ACtion-dEpendent Q-learning (ACE), which transfers existing single-agent value-based methods to multi-agent scenarios with minimalist adaptation and inherits their theoretical guarantee of convergence and performance.

Value-based methods usually learn the function $Q : \mathcal{S} \rightarrow \mathbb{R}^{|A|}$ to build the mapping from the state to the estimated return of actions, and select the action with the maximum Q value during execution. However, in SE-MDP, once making the decision a_{i+1} for the $i+1$th agent on the current SE-state $s^t_{a_{1:i}}$, we can direct intermediate transition to the next SE-state $s^t_{a_{1:i+1}}$ without interacting with the environment. Hence, we take a step forward and use the value function $V : \widetilde{\mathcal{S}} \rightarrow \mathbb{R}$ to estimate the return of the SE-state rather than the action, and use the values $V\left(s^t_{a_{1:i+1}}\right)$ of all possible next SE-states rolled out via different actions a_{i+1} to select the optimal action.

Decision with Rollout Specifically, to make decision at an SE-state $s^t_{a_{1:i}}$, we use agent $i+1$'s action space A_{i+1} to roll out to all possible next SE-states $s^t_{a_{1:i+1}}$, and select the action $a^t_{i+1} = \arg\max_{a_{i+1}} V\left(s^t_{a_{1:i}, a_{i+1}}\right)$, which leads to the next SE-state with the optimal value $V\left(s^t_{a_{1:i+1}}\right)$.

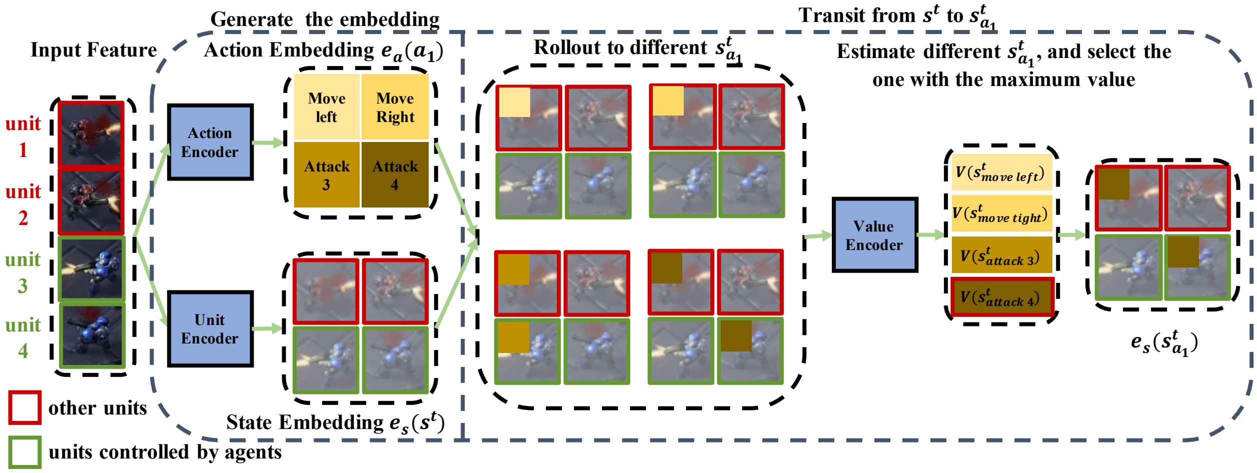


Figure 3: Schematic of the pipeline of ACE, which takes SMAC as an instance. There are four units in the map. Units 1 and 2 are controlled by the RL agent, and units 3 and 4 are enemies controlled by the environment. At first, the initial state embedding is generated, consisting of the initial embedding for all units obtained from the unit encoder, as well as the action embedding of all actions obtained from the action encoder (only the action embedding of unit 1 is shown in the figure, where actions *attack 3* and *attack 4* mean unit 1 attacking unit 3 and 4 respectively). Then, agent (unit) 1 is the first one to make the decision, thus its action embeddings are incorporated into the initial unit embeddings to rollout to the embeddings of different new SE-states $e\left(s^t_{a_1}\right)$ (4 rolled out SE-states in the figure). Afterwards, all of these new SE-states are evaluated by the value encoder. Finally, the SE-state with the maximum value is retained and used by the next rollout for the action of agent 2.

Update with Rollout Our value function V is updated by the standard Bellman backup operator in single agent RL. At an SE-state $s^t_{a_{1:i}}$, to obtain the target value to update the value $V\left(s^t_{a_{1:i}}\right)$, we also rollout to all possible next SE-states $s^t_{a_{1:i+1}}$, estimate their values $V\left(s^t_{a_{1:i+1}}\right)$ and select the maximum value as the target value. For the final SE-state $s^t_{a_{1:n}}$ in a decision sequence, we roll out at the first SE-state s^{t+1} in the next decision sequence, i.e., the next state in the original MMDP $\mathcal{G}$. The update of V is formalized as Eq 1, with $\hat{V}\left(s^t_{a_{1:i}}\right)$ denoting the Bellman target of $V\left(s^t_{a_{1:i}}\right)$.

$$\hat{V}\left(s^t_{a_{1:i}}\right)=\begin{cases}\max_{a_{i+1}}\ \gamma V\left(s^t_{a_{1:i},a_{i+1}}\right), & \text{if } i<n\\ \max_{a_1}\ r\left(s^t,a_{1:n}\right)+\gamma V\left(s^{t+1}_{a_1}\right), & \text{if } i=n\end{cases} \tag{1}$$

Network Representation

Deep Reinforcement Learning (DRL) methods usually benefit from the good generalization ability of a deep neural network (DNN), which encodes the state to a vectorized embedding and maps the embedding to the estimated return of the state or action. As the design of representation has a great effect on the efficiency and performance of the algorithm, we will discuss two concerns in the design of the network representation of the SE-state $s^t_{a_{1:i}}$ with DNN.

Decomposed State Embedding Firstly, a transition $\left(s^t,\boldsymbol{a}^t,r\left(s^t,\boldsymbol{a}^t\right),s^{t+1}\right)$ in the original MMDP $\mathcal{G}$ corresponds to n intermediate transitions in the sequential expansion $\widetilde{\mathcal{G}}$ and each intermediate transition requires to evaluate $|A_i|$ next states, resulting in $\sum_{i=1}^n|A_i|$ total states for evaluation. Direct computing all states' embedding from scratch will bring unacceptable computational cost, thus the first principle we follow in the representation of SE-state is: all the state embeddings $e_s\left(s^t_{a_{1:i}}\right)$ along the sequential decision-making are decomposed into a shared embedding $e_s\left(s^t\right)$ of the initial state s^t, as well as a shared set of embeddings $e_a\left(a_1\right),...,e_a\left(a_n\right)$ of available actions $a_1,...,a_n$, all generated by the same action encoder. Then, the state embedding $e_s\left(s^t_{a_{1:i}}\right)$ is obtained by combining the initial state embedding $e_s\left(s^t\right)$ and the corresponding action embeddings $e_a\left(a_1\right),...,e_a\left(a_i\right)$. In this decomposition, the original state only requires to be encoded once rather than $\sum_{i=1}^n|A_i|$. Moreover, the combination is additive and introduces negligible cost. **Secondly**, a multi-agent task involves interaction among multiple units, including cooperative interaction among agent-controlled units, like healing an allied unit in SMAC, and interaction between agent-controlled and environment-controlled units, like attacking an enemy unit in SMAC. We follow two designs, unit-wise state embedding and interaction-aware action embedding, to describe the interactions in the state and action embedding.

Unit-wise State Embedding For state embedding, we use a **unit encoder** to generate the unit-wise embedding $e_u\left(u_i\right)$ of each unit u_i in the environment, which forms the initial state embedding $e_s\left(s^t\right)=\left[e_u\left(u_1\right),...,e_u\left(u_m\right)\right]$. Here m is the number of units. We assume that the first n units are controlled by the RL agent and the rest $(m-n)$ ones are controlled by the environment. We do not fuse the unit embeddings to a global state embedding, but retain them to facilitate the description of the interactions among units. The input feature of each unit includes the **node feature** and **edge feature**. The node feature is the state of each unit, e.g., the health and shield in SMAC and the speed in GRF, and the edge feature is the relation between the units, e.g., the distance between units in SMAC. Our unit encoder takes a fairly simple architecture, depicted in Figure 2. The node and edge feature are separately encoded by two encoders to generate the corresponding embedding. In this paper, we take a fully

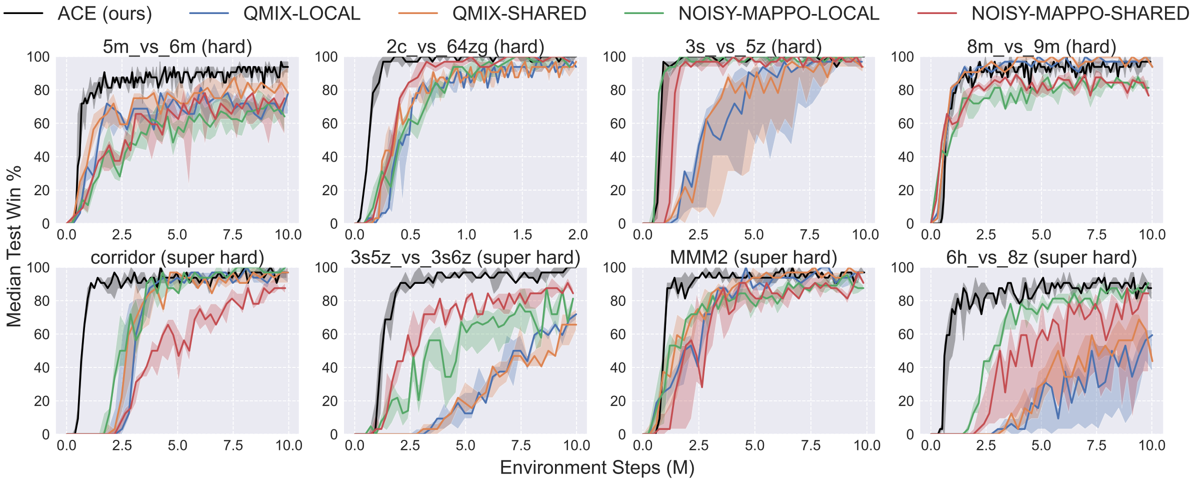


Figure 4: Comparison of ACE against baselines on four super hard and four hard SMAC maps.

connected layer along with a ReLU 2018 as the encoder. The resulted edge embedding average-pooled and then added to the node embedding to obtain the final unit embedding.

Interaction-aware Action Embedding To make the state embedding $e_s\left(s^t_{a_{1:i}}\right)$ aware of the unit interactions, we develop a two-fold **interaction-aware action embedding**. Given an action a_i that is executed by unit u_i and involves the interaction with some target units, its action embedding consists of an **active embedding** and a **passive embedding**, formalized by $e_a\left(a_i\right) = \left[e^a_a\left(a_i\right), e^p_a\left(a_i\right)\right]$. The active embedding $e^a_a\left(a_i\right)$ encodes the effect of action a_i on the unit u_i itself, and the passive embedding $e^p_a\left(a_i\right)$ encodes the effect of action a_i on the target units. For actions without interaction, it only has an active embedding $e^a_a\left(a_i\right)$, formalized by $e_a\left(a_i\right) = \left[e^a_a\left(a_i\right)\right]$.

After the generation of the original state embedding $e_s\left(s^t\right) = \left[e_u\left(u_1\right), ..., e_u\left(u_m\right)\right]$ and the action embedding $e_a\left(a_1\right), ..., e_a\left(a_i\right)$, we use an additive combination of the unit and action embedding to construct the state embedding $e\left(s^t_{a_{1:i}}\right)$ of the intermediate SE-states $s^t_{a_{1:i}}$, formalized by $e_s\left(s^t_{a_{1:i}}\right) = \left[e_u\left(u_{1,a_{1:i}}\right), ..., e_u\left(u_{m,a_{1:i}}\right)\right]$. The element $e_u\left(u_{j,a_{1:i}}\right)$ denotes the combination of the initial unit embedding $e_u\left(u_j\right)$ of unit j and the embeddings of its associated actions among $a_{1:i}$. The rule of combination is: for each action a_i, its active action embedding $e^a_a\left(a_i\right)$ is added on the unit embedding $e_u\left(u_i\right)$ of it executor u_i; if a_i involves an interaction with some target unit, its passive action embedding $e^p_a\left(a_i\right)$ is added on $e_u\left(u_j\right)$ to describe the interaction. The definition of $e_u\left(u_{j,a_{1:i}}\right)$ is formalized by:

$$e_u\left(u_{j,a_{1:i}}\right) = \begin{cases} e_u\left(u_j\right) + \sum_{e^p_a\left(a_k\right) \in P\left(a_{1:i}\right)_j} e^p_a\left(a_k\right), \\ \qquad\qquad\qquad\qquad \text{if } i < j \\ e_u\left(u_j\right) + e^a_a\left(a_j\right) + \sum_{e^p_a\left(a_k\right) \in P\left(a_{1:i}\right)_j} e^p_a\left(a_k\right), \\ \qquad\qquad\qquad\qquad \text{if } i >= j \end{cases} \tag{2}$$

where $P\left(a_{1:i}\right)_j$ is the set of all passive action embeddings whose target unit is u_j. When $i >= j$, which means u_j is an agent-controlled unit and has made its decision a_j, the active embedding $e^a_a\left(a_j\right)$ will also be added to $e_u\left(u_j\right)$.

In this paper, the passive embedding $e^p_a\left(a_i\right)$ of a unit u_i is generated from an action encoder whose input is the node feature of the unit u_i, because the effect of action a_i may rely on the executor's state. For instance, in GRF the effect on the ball is affected by the speed of the controller. However, the active embedding $e^a_a\left(a_i\right)$ is defined as a learnable parameterized vector, because it is added to the embedding $e_u\left(u_i\right)$ of u_i which has already encoded the state of u_i. Both the two kinds of embeddings are learnable. Like the encoders of node and edge features, we also take a fully connected layer along with a ReLU activation as the action encoder in this paper.

At last, we use a encoder to estimate the value of each SE-state. The state embedding $e_s\left(s^t_{a_{1:i}}\right) = \left[e_u\left(u_{1,a_{1:i}}\right), ..., e_u\left(u_{m,a_{1:i}}\right)\right]$ is fed into a 'fc-relu' structure to encode the interaction-aware unit embedding, followed by a 'pooling-fc' structure to output the estimated value. Figure 3 demonstrates the pipeline of embeddings generation and how to use them to represent the transition in $\widetilde{\mathcal{G}}$.

Experiment

To study the advantages of ACE, we consider three tasks: (1) Spiders-and-Fly, (2) StarCraft Multi-Agent Challenge and (3) Google Research Football. Since the baselines we compare with are designed for partial observation settings, we also introduce our efforts to guarantee the fairness in this section. More details on these tasks are included in Appendix.

Spiders-and-Fly The Spiders-and-Fly problem is first proposed in 2019, where multiple spiders are controlled to catch a fly in a two 2D grid. At each time step, each spider moves to a neighboring location or stays put, while the fly moves randomly to a neighboring location. In this paper, we modify it to a much harder problem where only two spiders are controlled by the RL agent, and the fly will avoid moving to the neighboring locations of the spiders, otherwise stay still. Each episode starts with a state where the Manhattan distance between the fly and each spider is larger than 4. With such modifications, the two spiders must perform perfect cooperation to encircle the fly at the corner. The reward is defined as 10 if the fly is caught otherwise 0.

StarCraft Multi-Agent Challenge (SMAC) In SMAC 2019, the ally units controlled by the RL agent play

Metric	Map	VDN	QMIX	QTRAN	ACE
Steps	5×5	0.78±0.10	0.77±0.10	0.60±0.09	**0.04±0.03**
	7×7	0.90±0.12	0.87±0.11	1.02±0.09	**0.07±0.02**
Samples (M)	5×5	0.19±0.02	0.19±0.02	0.17±0.02	**0.09±0.01**
	7×7	1.97±0.10	1.81±0.09	1.68±0.09	**1.01±0.06**

Table 1: Comparison ACE against baselines on Spiders-and-Fly. Steps represent the gap between the average steps of the methods and the oracle policy. Samples represent the number of samples to achieve a 100% success rate within 10 steps.

against the enemy units controlled by the built-in rules. To win the competition, allies learn to perform cooperative micro-tricks, such as positioning, kiting and focusing fire. This benchmark consists of various maps classified as Easy, Hard, and Super Hard. Since the Easy maps solved well by existing methods 2021, we focus on four super hard maps: corridor, MMM2, 6h_vs_8z, and 3s5z_vs_3s6z, and four hard maps: 5m_vs_6m, 2c_vs_64zg, 8m_vs_9m and 3s_vs_5z.

Google Research Football (GRF) Compared with SMAC, GRF 2020 provides a harder environment with large action space and sparse reward. In the GRF, agents coordinate timing and location to organize attacks and only scoring leads to rewards. In our experiments, we control the left team players except for the goalkeeper while the built-in engine controls the right team players. We evaluate our method on two challenging scenarios: academy_3_vs_1_with_keeper and academy_counterattack_hard. For a fair comparison, we use the standard 19 actions (*i.e.*, moving, sliding, shooting and passing), and use the same observation in CDS 2021 to construct our input feature. Following the settings in CDS, we also make a reasonable change to the two half-court offensive scenarios: we will lose if our players or the ball returns to our half-court. All experiments are tested with this modification. The final reward is +100 when our team wins, -1 when our player or the ball returns to our half-court, and 0 otherwise.

Evaluation Metric For Spiders-and-Fly, we derive an analytical optimal solution as the oracle policy and introduce two metrics: (1) the samples required to achieve 100% success rate in ten steps, and (2) the gap between the average steps required by the RL policy and the oracle policy to catch the fly. For SMAC, we follow the official evaluation metric in 2019, *i.e.,* we run 32 test episodes without exploration to record the test win rate and report the median performance as well as the 25-75% percentiles across 5 seeds. For GRF, we similarly run 32 test episodes to obtain win rate and report the average win rate as well as the variance across 5 seeds.

Performance

Spiders-and-Fly We compare ACE with three value factorization methods: QTRAN 2019, QMIX 2018 and VDN 2017, on two grids with the sizes 5x5 and 7x7. As shown in Table 1, ACE is the only one that can approximate the performance of the oracle policy, while the baselines, although also find the best behavior in some cases, cannot consistently converge to the optimal policy. Moreover, ACE takes up to 50% fewer samples to achieve the 100% success rate in ten steps.

SMAC We compare ACE with both the state-of-the-art (SOTA) value-based and actor-critic methods on SMAC. First, our value-based baseline is the fine-tuned QMIX 2021 combining QMIX 2018 with bags of code-level optimizations and outperforming QPLEX 2020, QTRAN 2019, vanilla QMIX and Weighted QMIX 2020. Secondly, we choose the SOTA actor-critic method, NOISY-MAPPO 2021, as the actor-critic baseline. Although the two methods are proposed for CTDE pipeline, they are also important baselines to solve the exponentially large action space in multi-agent tasks. To this end, the comparison with them is fair. Note that both baseline algorithms are designed for partially observable scenarios, where each agent only use its local observation to generate the action, while ACE uses the observation of all units to make decisions. Thus, to make a fair comparison, in the two baselines, we make each agent share the union of all units' observations at the input, denoted as SHARED. We also evaluate the baselines with the original local observation, denoted as LOCAL, because in some cases the shared observation has worse performance. For example, NOISY-MAPPO-LOCAL achieves better performance than NOISY-MAPPO-SHARED in 6h_vs_5z. As shown in Figure 4, ACE surpasses fine-tuned QMIX and NOISY-MAPPO by a large margin in the final win rate and the sample efficiency. Remarkably, it achieves 100% test win rates in almost all maps, including 5m_vs_6m and 3s5z_vs_3s6z, which have not been solved well by existing methods even with shared observation. Therefore, ACE achieves a new SOTA on SMAC.

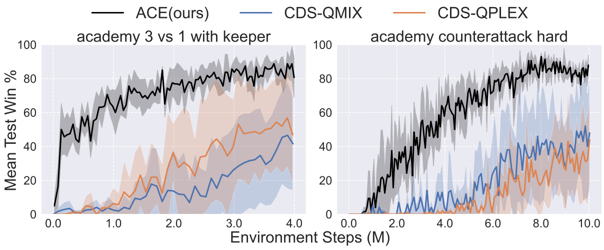


Figure 5: Comparison of ACE against baseline on GRF.

GRF We show the performance comparison against the baselines in Figure 5. ACE outperforms the SOTA methods CDS-QMIX 2021 and CDS-QPLEX 2021 by a large margin in both two scenarios. The gap between ACE and the baselines is even larger than that on SMAC, possibly due to that the football game requires more complex cooperation skills.

Ablation: What matters in the components of ACE?

To better understand why ACE outperforms the baseline algorithms, we further make ablations and modifications to it. First, we remove the interaction-aware action embedding by only using the active embedding, denoted by ACE-w/o-IA, and compare it with ACE and the fine-tuned QMIX. As shown in Figure 6a, the gap between ACE-w/o-IA and the QMIX is still large, which validates that the bidirectional action-dependency itself retains much of the benefit of ACE. Moreover, ACE-w/o-IA is worse than ACE due to the effective interaction-aware embedding. Secondly, we study how the order of decision-making influences the performance of ACE. We compare two settings: 1) Shuffle Order: random orders are generated for both data collection and training. 2) Sorted Order: agents are sorted by unit types and locations. As shown in Figure 6b, the two settings have little difference in performance in two SMAC maps, which validates that

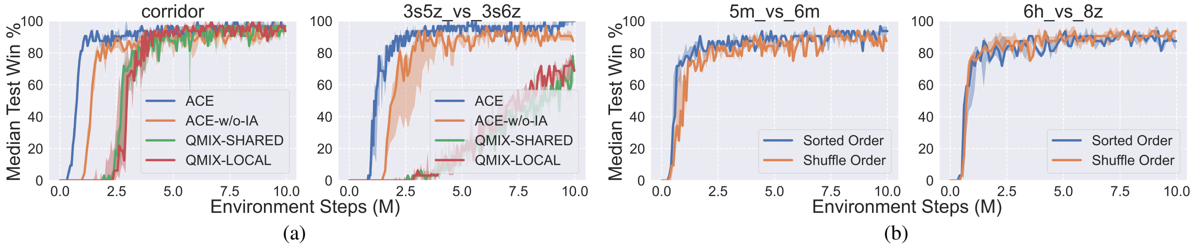


Figure 6: **(a):** Comparison of ACE and ACE-w/o-IA against QMIX on corridor and 3s5z_vs_3s6z. **(b):** Comparison of sorted order against shuffle order of ACE.

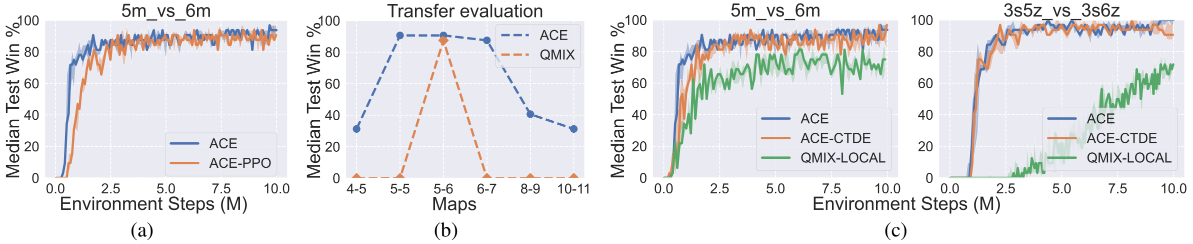


Figure 7: **(a):** Comparision of ACE and ACE-PPO on 5m_vs_6m. **(b):** Transfer from 5m_vs_6m to maps of different numbers of agents. 'x-y' represents xm_vs_ym. ACE can achieve remarkable performance in the zero-shot setting, regardless of whether agent number increases or decreases. **(c):** Comparison of ACE-CTDE against ACE.

ACE is quite robust to the order of agents.

Extension: Extend ACE to the actor-critic method

Our approach, transforming a MMDP into a MDP, is general and can be combined with a more extensive range of single-agent RL algorithms. In this section, we combine ACE with an actor-critic method, PPO, denoted by ACE-PPO. To generate the logit of each action in PPO, we roll out each action to the corresponding next SE-states, and use the same way how our value encoder evaluates these states to obtain the logit. As shown in Figure 7a, ACE-PPO achieves a comparable performance with ACE on the 5m_vs_6m map, which validates that ACE is applicable to wider types of algorithms.

Generalization: Does ACE generalize to a new task with a different number of agents?

An interesting advantage of ACE is its surprising generalization. Compared with prior methods where agents make decisions individually, ACE explicitly models the cooperation between agents. As a result, when the preceding agents take sub-optimal actions due to the change of the task, the subsequent agents compensate it through the learned cooperative skills. We train ACE and the fine-tuned QMIX-SHARED on 5m_vs_6m and test them on 4m_vs_5m, 5m_vs_5m, 6m_vs_7m, 8m_vs_9m and 10m_vs_11m. Note that the action and observation space change with the agent number, we address this problem as described in Appendix. As in Figure 7b, although without any fine-tuning on the test maps, ACE still achieves considerable win rates, which reveals an excellent generalization to the change of the agent number.

Practicability: Apply ACE to the CTDE scheme.

We develop a simple adaptation of ACE, denoted by ACE-CTDE, to apply it in the Centralized training and decentralized execution (CTDE) scheme, a popular scheme for multi-agent tasks with limited communication. In most CTDE methods, an individual value function uses the local observation to estimate the individual value, and a joint value function estimates the value of the joint action with the global state. The optimal actions of the two functions are aligned via well-designed constraints to guarantee the IGM property. Similarly, we use a counterfactual distillation technique, to distill the optimal joint action directly generated via the sequential rollout in ACE, into an additional individual value function $Q\left(o_i^t, a_i\right)$ of which the input is the local observation. The counterfactual distillation is formalized by $\hat{Q}\left(o_i^t, a_i\right) = V\left(s^t_{a_i, a^*_{i\_}}\right)$. $\hat{Q}\left(o_i^t, a_i\right)$ is the target to update $Q\left(o_i^t, a_i\right)$ and o_i^t is the local observation of agent i. $a^*_{i\_}$ denotes the optimal joint action generated by the sequential rollout excluding the action of agent i. This distillation estimates each individual action value of an agent with other agents' actions fixed jointly optimal, thus it follows the IGM principle. In Figure 7c, ACE-CTDE is evaluated with the individual value function Q in a decentralized way. We can see that ACE-CTDE performs nearly as well as ACE due to the IGM property of the proposed distillation.

Conclusion

In this paper, we introduce bidirectional action-dependency to solve the non-stationary problem in cooperative multi-agent tasks. The proposed ACE algorithm significantly improves the sample efficiency and the converged performance against the state-of-the-art algorithm. Comprehensive experiments validate the advantages of ACE in many aspects.

Acknowledgments

Wanli Ouyang was supported by the Australian Research Council Grant DP200103223, Australian Medical Research Future Fund MRFAI000085, CRC-P Smart Material Recovery Facility (SMRF) – Curby Soft Plastics, and CRC-P ARIA - Bionic Visual-Spatial Prosthesis for the Blind. This work is partially supported by the Shanghai Committee of Science and Technology (Grant No. 21DZ1100100). We acknowledge Yining Fang for the strong support and in depth discussion.

References

Agarap, A. F. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

Bertsekas, D. 2019. Multiagent rollout algorithms and reinforcement learning. *arXiv preprint arXiv:1910.00120*.

Böhmer, W.; Kurin, V.; and Whiteson, S. 2020. Deep coordination graphs. In *International Conference on Machine Learning*, 980–991. PMLR.

Canese, L.; Cardarilli, G. C.; Di Nunzio, L.; Fazzolari, R.; Giardino, D.; Re, M.; and Spanò, S. 2021. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11): 4948.

Chenghao, L.; Wang, T.; Wu, C.; Zhao, Q.; Yang, J.; and Zhang, C. 2021. Celebrating diversity in shared multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34.

Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2018. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Fu, W.; Yu, C.; Xu, Z.; Yang, J.; and Wu, Y. 2022. Revisiting Some Common Practices in Cooperative Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*.

Gronauer, S.; and Diepold, K. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2): 895–943.

Hu, J.; Hu, S.; and Liao, S.-w. 2021. Policy Perturbation via Noisy Advantage Values for Cooperative Multi-agent Actor-Critic methods. *arXiv preprint arXiv:2106.14334*.

Hu, J.; Wu, H.; Harding, S. A.; Jiang, S.; and Liao, S.-w. 2021. RIIT: Rethinking the Importance of Implementation Tricks in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2102.03479*.

Hüttenrauch, M.; Šošić, A.; and Neumann, G. 2017. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*.

Kuba, J. G.; Chen, R.; Wen, M.; Wen, Y.; Sun, F.; Wang, J.; and Yang, Y. 2021a. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*.

Kuba, J. G.; Feng, X.; Ding, S.; Dong, H.; Wang, J.; and Yang, Y. 2022. Heterogeneous-agent mirror learning: A continuum of solutions to cooperative marl. *arXiv preprint arXiv:2208.01682*.

Kuba, J. G.; Wen, M.; Meng, L.; Zhang, H.; Mguni, D.; Wang, J.; Yang, Y.; et al. 2021b. Settling the variance of multi-agent policy gradients. *Advances in Neural Information Processing Systems*.

Kurach, K.; Raichuk, A.; Stańczyk, P.; Zając, M.; Bachem, O.; Espeholt, L.; Riquelme, C.; Vincent, D.; Michalski, M.; Bousquet, O.; et al. 2020. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 4501–4510.

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

Lowe, R.; Wu, Y. I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30.

Papoudakis, G.; Christianos, F.; Rahman, A.; and Albrecht, S. V. 2019. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1906.04737*.

Rashid, T.; Farquhar, G.; Peng, B.; and Whiteson, S. 2020. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33: 10199–10210.

Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 4295–4304. PMLR.

Ruan, J.; Du, Y.; Xiong, X.; Xing, D.; Li, X.; Meng, L.; Zhang, H.; Wang, J.; and Xu, B. 2022. GCS: Graph-based Coordination Strategy for Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2201.06257*.

Samvelyan, M.; Rashid, T.; De Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G.; Hung, C.-M.; Torr, P. H.; Foerster, J.; and Whiteson, S. 2019. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

Shalev-Shwartz, S.; Shammah, S.; and Shashua, A. 2016. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*.

Son, K.; Ahn, S.; Reyes, R. D.; Shin, J.; and Yi, Y. 2020. QTRAN++: Improved Value Transformation for Cooperative Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2006.12010*.

Son, K.; Kim, D.; Kang, W. J.; Hostallero, D. E.; and Yi, Y. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, 5887–5896. PMLR.

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; et al. 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*.

Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, 330–337.

Terry, J.; Black, B.; Grammel, N.; Jayakumar, M.; Hari, A.; Sullivan, R.; Santos, L. S.; Dieffendahl, C.; Horsch, C.; Perez-Vicente, R.; et al. 2021. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 15032–15043.

Usunier, N.; Synnaeve, G.; Lin, Z.; and Chintala, S. 2016. Episodic exploration for deep deterministic policies for StarCraft micromanagement.

Wang, J.; Ren, Z.; Liu, T.; Yu, Y.; and Zhang, C. 2020. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*.

Wang, W.; Yang, T.; Liu, Y.; Hao, J.; Hao, X.; Hu, Y.; Chen, Y.; Fan, C.; and Gao, Y. 2019. Action Semantics Network: Considering the Effects of Actions in Multiagent Systems. In *International Conference on Learning Representations*.

Ye, J.; Li, C.; Wang, J.; and Zhang, C. 2022. Towards Global Optimality in Cooperative MARL with Sequential Transformation. *arXiv preprint arXiv:2207.11143*.

Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A.; and Wu, Y. 2021. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. *arXiv preprint arXiv:2103.01955*.

Zhang, C.; and Lesser, V. 2011. Coordinated multi-agent reinforcement learning in networked distributed POMDPs. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Zhang, C.; and Lesser, V. 2013. Coordinating multi-agent reinforcement learning with limited communication. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 1101–1108.

Zhang, T.; Xu, H.; Wang, X.; Wu, Y.; Keutzer, K.; Gonzalez, J. E.; and Tian, Y. 2020. Multi-agent collaboration via reward attribution decomposition. *arXiv preprint arXiv:2010.08531*.

Zhang, Z.; Li, H.; Zhang, L.; Zheng, T.; Zhang, T.; Hao, X.; Chen, X.; Chen, M.; Xiao, F.; and Zhou, W. 2019. Hierarchical reinforcement learning for multi-agent moba game. *arXiv preprint arXiv:1901.08004*.

Zhou, M.; Luo, J.; Villella, J.; Yang, Y.; Rusu, D.; Miao, J.; Zhang, W.; Alban, M.; Fadakar, I.; Chen, Z.; et al. 2020. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *arXiv preprint arXiv:2010.09776*.

Appendix

In this appendix, we first discuss the difference between this paper and related works. Then, we provide more elaboration on the implementation details, experiment results and qualitative results. Specifically, we present the discussion of the related works in Further Discussion of Related Work, the implementation details of ACE and the baselines on the three environments in Implementation Details, implementation details of ablation studies in Ablation Details, and additional experimental results in Additional Experiments.

Further Discussion of Related Work

Many MARL algorithms have been proposed to achieve improvements in cooperation aspects. The two tracks that are most relative with ACE are coordination graph 2011; 2013; 2020 and advanced network representation 2019.

To mitigate the non-stationary problem in MARL, the coordination Graph-based algorithms introduce action dependency into the formulation and utilize the higher-order state-action functions instead of fully individual state-action functions to represent the joint Q-function. However, coordination graph methods, still based on the value factorization paradigm, only introduce pairwise action dependency in the value estimation for MMDP. While ACE is based on the transformation from MMDP to MDP, which tackles the non-stationary problem by introducing full action dependency.

To boost the action representation in MARL, ASN 2019 also classifies the actions of an agent into A_{in} and A_{out} according to whether the actions affect other units. However, ACE is fundamentally different with ASN. First, ASN is designed for individual value network, while ACE for the joint value network. Secondly, ASN can only model the action of one agent, and ACE describe all actions that has been selected. Moreover, ASN uses the information of the target unit at the input to model the action that interacts with the target unit, while ACE use the action embedding to update the output embedding of the target unit.

ZO 2016 is another relative work. It also introduces a sequential inference scheme/sequential expansion (SE) for structured output prediction. However, the potential of SE has long been overlooked by the community. In this work, we rethink this topic and fully releases the potential of SE and further validate the compatibility of SE with other single agent algorithms (*e.g.,* PPO). Additionally, ACE directly learns the value function by $V(s, a_1, a_2, ..., a_i)$, which has a unified representation of agents that have and haven't made decisions. While the definition of value function in ZO is $Q(a_i|s, a_1, a_2, ..., a_{i-1})$, where the model must additionally learn which agent to choose act for in each step, which struggles to achieve high sample efficiency.

Implementation Details

Baselines

(1) Fine-tuned QMIX2021 and Noisy-MAPPO2021 We use the official code of fine-tuned QMIX [1] and Noisy-MAPPO [2] provided by their authors. These two implementations incorporate fine-tuned hyper-parameters and bags of code-level optimizations, which fully release the potential of the two baselines.

(2) The Difference between SHARED and LOCAL The aim of the SHARED setting is to enable each individual agent in our baseline algorithms to make the decision based on the observation of all agents. In the SHARED setting, we make the state of an enemy observable to all agents if it is observed by at least one agent. Also, the state of each agent is observed by all other agents. As a result, the SHARED setting enables all agents to have access to a shared global observation, which is the union of the observation of all agents.

(3) CDS2021 We use the official code of CDS [3].

(4) QTRAN, QMIX, and VDN on Spider and Fly 2021 We use the same hyper-parameters as ACE for QTRAN, QMIX, and VDN, as listed in Table 2. Also, the input feature of them is based on the same information as ACE, including the unit ID, the position, and the distance among units, as listed in Table 3. For the algorithm-specific hyper-parameters of these baselines, like the weight of the extra loss in QTRAN, we use the original choices provided in their original papers.

Parameter	Value
Exploration	
action_selector	epsilon_greedy
epsilon_type	linear
epsilon_start	1
epsilon_end	0.05
epsilon_decay	150k
Sample	
collector_env_num	8
sample_per_collect	1024
replay_buffer_size	1M
Training	
update_per_collect	10
batch_size	256
weight_decay	0
learning_rate	0.0005
target_update_theta	0.02
discount_factor	0.99
optimizer	adam
Model	
hidden_len	128

Table 2: Hyper-parameter Settings of ACE and baselines on Spiders and Fly.

ACE

We implemented our ACE based on the DI-engine [4], which is a generalized decision intelligence engine and supports

[1] https://github.com/hijkzzz/pymarl2

[2] https://github.com/hijkzzz/noisy-mappo

[3] https://github.com/lich14/CDS

[4] https://github.com/opendilab/DI-engine

Feature Name	Components
ACE	
node_feature	unit id (0, 1 for spiders, 2 for fly) unit position
edge_feature	distance on two axes
Baselines	
input_feature	unit id (0, 1 for spiders, 2 for fly) unit position, distance to all other units on two axes

Table 3: Input Feature on Spiders and Fly.

various deep reinforcement learning algorithms. All codes are licensed under the Apache License or MIT License. The hyper-parameter settings of ACE on Spiders and Fly, SMAC and GRF are respectively listed in Table 2, Table 4 and 9. The definition of the input feature on the three environments are provided in Table 3, Table 5 and 6. Different from pymarl2 [5], in DI-engine, the batch_size represents the number of transitions, not the episode.

Parameter	Value
Exploration	
action_selector	epsilon_greedy
epsilon_type	linear
epsilon_start	1
epsilon_end	0.05
epsilon_decay	50k
Sampler	
collector_env_num	8
episode_per_collect	32
replay_buffer_size	300k (100k for 2c_vs_64zg)
Training	
update_per_collect	50
batch_size	320
weight_decay	1e-5
learning_rate	3-4
target_update_theta	0.008
discount_factor	0.99
optimizer	rmsprop
Model	
hidden_len	256

Table 4: Hyperparameter Settings of ACE on SMAC.

Feature Name	Components
node_feature	unit id, unit type, unit position, health, shield, cool down
edge_feature	distance on two axes

Table 5: Input Feature of ACE on SMAC.

In Table 7, we show in detail how the two-fold interaction-aware action embedding was constructed.

Feature Name	Components
node_feature	unit id, unit position, speed, whether own ball
edge_feature	distance on two axes

Table 6: Input Feature of ACE on GRF.

Environments

Spiders and Fly Here, we visualize the initial positions of the two spiders and fly in a 7×7 map in the Figure 8. Each episode starts with a state where the Manhattan distance between the fly and each spider is larger than 4. In each time step, the optimal strategy is one of the following two types: (1) both of the spiders move to drive the fly to the corner, and when it is at the corner, (2) one spider stays still to restrict the possible movement of the fly and another one approaches the fly. Thus, optimal cooperation is required in each time step. It is observed in our experiments that, the proposed ACE is the only one that can approximate the performance of the oracle policy in the 5×5 and 7×7 map. This result benefits from that the introduced bidirectional action-dependency enables the explicit learning of how to cooperate with the preceding agents and how the successors react.

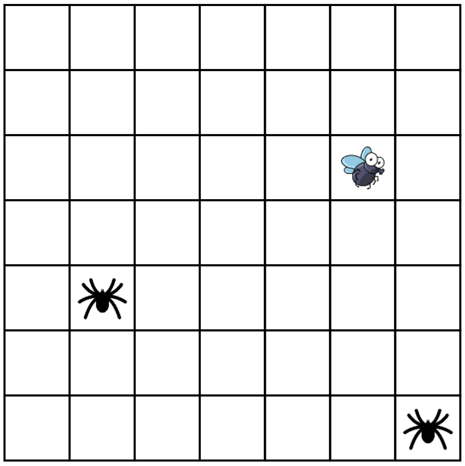


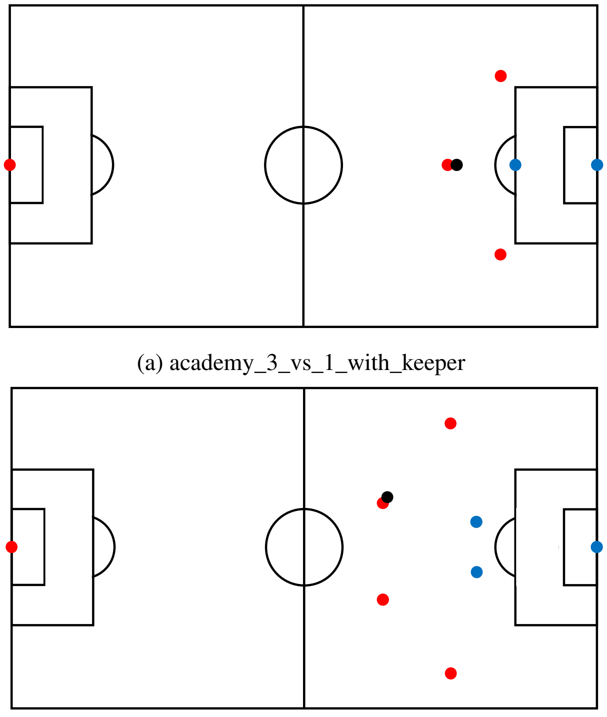
Figure 8: Visualization of spiders and the fly in a 7×7 map, where the two spiders are controlled by the RL agent. The reward is defined as 10 if the fly is caught otherwise 0.

GRF Academy_3_vs_1_with_keeper and academy_counter- attack_hard are two of the hardest official scenarios in GRF. To provide more clear details about the two scenarios in GRF, we visualize the initial positions of all players and the ball in the two scenarios in the Figure 9, and provide an RGB screenshot of academy_3_vs_1_with_keeper in Figure 10. In both two scenarios, the target is to control the left team players (red points in Figure 9) to get a score with the learned cooperative skills. The right team players are controlled by a built-in AI to defend. The proposed ACE achieves a new SOTA in both two scenarios.

[5] https://github.com/hijkzzz/pymarl2

Env	Action	Passive Embedding		Active Embedding	
		Whether Use?	Target Unit	Whether Use?	Target Unit
Spiders-and-Fly	move	No	N/A	Yes	Itself
	stay	No	N/A	Yes	Itself
SMAC	move	No	N/A	Yes	Itself
	attack	Yes	The unit it attack	Yes	Itself
	heal	Yes	The unit it heal	Yes	Itself
GRF	action of the ball owner	Yes	No	Yes	Itself
	action of other players	No	N/A	Yes	Itself

Table 7: Interaction-aware action embedding.



(a) academy_3_vs_1_with_keeper

(b) academy_counterattack_hard

Figure 9: Visualization of the initial positions of all players and the ball, where red points are our players, blue points are opponents and the black point represents the ball. All our players except for our goalkeeper are controlled by an RL agent while others are controlled by the built-in engine.

Ablation Details

ACE-PPO

In ACE-PPO, we derive the probability distribution over the action space of each agent via a softmax operation over a logit vector, formalized by:

$$p\left(a_{i+1}^t \mid s_{a_{1:i}}^t\right) = \frac{e^{l\left(a_{i+1}^t \mid s_{a_{1:i}}^t\right)}}{\sum_{\tilde{a}_{i+1}^t \in \mathcal{A}_{i+1}} e^{l\left(\tilde{a}_{i+1}^t \mid s_{a_{1:i}}^t\right)}}. \tag{3}$$

Here $l\left(a_{i+1}^t \mid s_{a_{1:i}}^t\right)$ is the logit generated in the same way as $V\left(s_{a_{1:i+1}}^t\right)$. Specifically, we use an additional logit encoder with the same structure as the value encoder to produce $l\left(a_{i+1}^t \mid s_{a_{1:i}}^t\right)$.

We use GAE(λ) as the advantage estimator, which is a common practice of PPO in most tasks. The temporal difference is calculated and summed over all adjacent SE-states, formalized by:

$$A\left(a_{i+1}^t \mid s_{a_{1:i}}^t\right) = \tag{4}$$

$$(\gamma\lambda)^0\left(\gamma V\left(s_{a_{1:i+1}}^t\right) - V\left(s_{a_{1:i}}^t\right)\right) \tag{5}$$

$$+(\gamma\lambda)^1\left(\gamma V\left(s_{a_{1:i+2}}^t\right) - V\left(s_{a_{1:i+1}}^t\right)\right) \tag{6}$$

$$+\dots \tag{7}$$

$$+(\gamma\lambda)^{(n-i)}\left(r\left(s^t, a_{1:n}^t\right) + \gamma V\left(s_{a_1}^{t+1}\right) - V\left(s_{a_{1:n}}^t\right)\right) \tag{8}$$

$$+(\gamma\lambda)^{(n-i+1)}\left(\gamma V\left(s_{a_2}^{t+1}\right) - V\left(s_{a_1}^{t+1}\right)\right) \tag{9}$$

$$+\dots. \tag{10}$$

In summary, our implementation of ACE-PPO on the SE-MDP is equivalent to the standard PPO on a single agent MDP. The hyper-parameters are listed in Table 8.

Generalization

To handle the change of the observation dimension and the action space among maps with different agent numbers, we develop extended observation and action space. In all the maps, we assume that there exist the largest possible numbers of ally units and enemy units, which are 10 and 11 in the largest map 10m_vs_11m. To realize any smaller map xm_vs_ym(with x<=10 and y<=11), we initialize each episode with only x ally units and y enemy units alive, which are randomly chosen. This modification enables us to run ACE and fine-tuned QMIX on all maps with the same observation dimension and action space.

ACE-CTDE

In this subsection we provide the detail of the implementation of $Q\left(o_i^t, a_i\right)$ in ACE-CTDE. For each agent i, we generate $Q\left(o_i^t, a_i\right)$ in the same way as that of $V\left(s_{a_{1:i}}^t\right)$, with the same unit encoder, action encoder and value encoder, except two differences. First, due to the partial observation, we only calculate the unit embedding of the units observed by agent i. Also, the edge feature of these units only includes the relation with the units observed by agent i. Second, due to the limited communication, only the action embedding $e_a\left(a_i\right)$ of agent i is added on the corresponding unit embeddings, rather than all the preceding actions. Finally, the set of unit embeddings

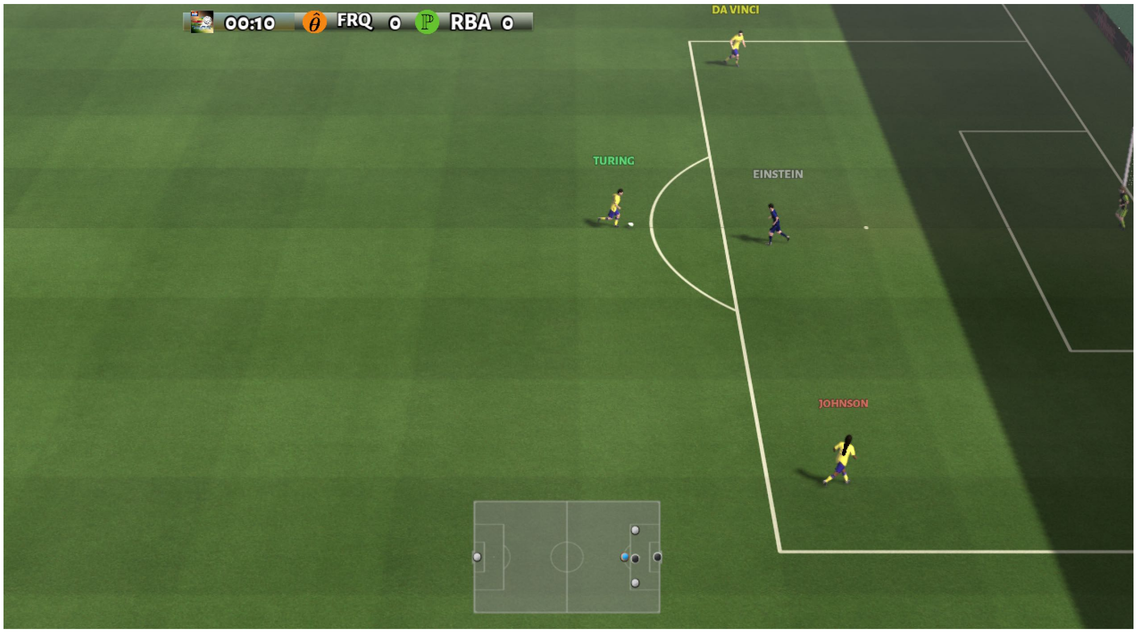


Figure 10: The academy_3_vs_1_with_keeper scenario in real game.

Parameter	Value
Exploration	
action_selector	softmax
Sampler	
collector_env_num	8
n_episode	32
gae_lambda	0.95
Training	
update_per_collect	50
batch_size	3200
learning_rate	5e-4
discount_factor	0.99
optimizer	adam
value_weight	1
entropy_weight	0.01
clip_ratio	0.05
use_value_clip	True
value_clip_ratio	0.3
recompute_adv	True
adv_norm	True
value_norm	True
Model	
hidden_len	256

Table 8: Hyperparameter Settings of ACE-PPO on SMAC.

(only of the units observed by agent i) incorporated with only $e_a\left(a_i\right)$ is encoded by a value encoder to generate $Q\left(o_i^t, a_i\right)$, with the same structure as that for V.

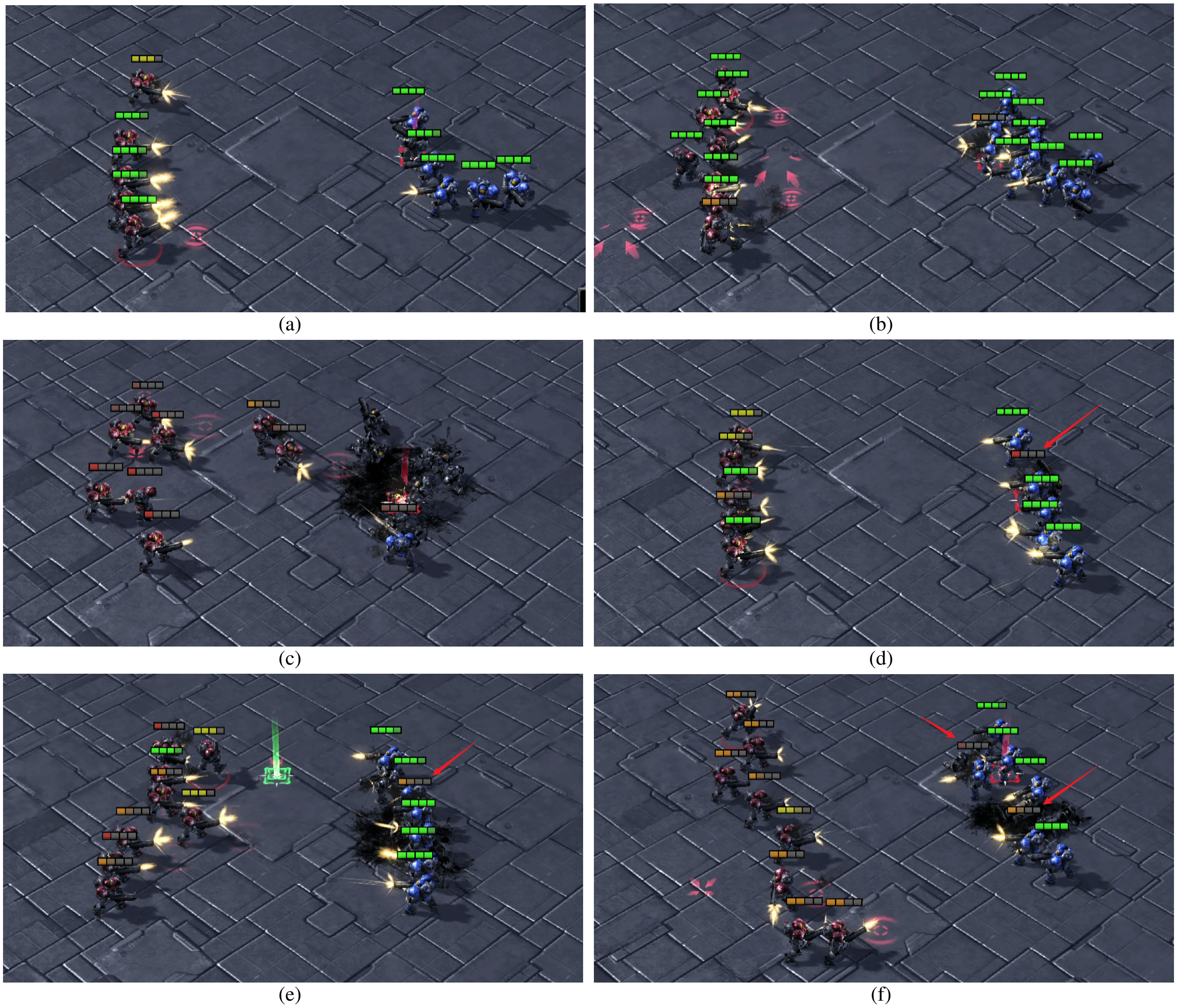
Additional Experiments

The behavior of transferred agents

In this section, we carefully analyze the behavior of transferred agents. As shown in Figure 11, ACE perfectly transfers the station position, careful positioning, alternating fire, and focusing fire of 5m_vs_6m to 8m_vs_9m and then achieves 40% win rate in 8m_vs_9m. However, the optimal choice of focusing fire in 8m_vs_9m is different from that in 5m_vs_6m, which is largely responsible for ACE not achieving a 100% win rate on the 8m_vs_9m map. Inspired with ACE, in future work, we will design an algorithm to use only one policy to solve all maps in SMAC consisting of different unit types and numbers.

Parameter	Value
Exploration	
action_selector	epsilon_greedy
epsilon_type	linear
epsilon_start	1
epsilon_end	0.05
epsilon_decay	50k for academy_3_vs_1_with_keeper 300k for academy_counterattack_hard
Sampler	
collector_env_num	8
episode_per_collect	32
replay_buffer_size	300k
Training	
update_per_collect	50
batch_size	320
weight_decay	1e-5
learning_rate	0.002 for academy_3_vs_1_with_keeper 0.0009 for academy_counterattack_hard
target_update_theta	0.08
discount_factor	0.99
optimizer	rmsprop
Model	
hidden_len	128

Table 9: Hyperparameter Settings of ACE on GRF.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 11: We denote agents trained on the Am_vs_Bm map and tested on the Cm_vs_Dm map as Train (A-B)-Test (C-D). For example, Train (5-6)-Test (8-9) means agents are tested on the 8m_vs_9m map while trained on the 5m_vs_6m map, corresponding to the settings of transfer learning. Train (5-6)-Test (5-6) means agents are trained and tested on the 5m_vs_6m map. **(a):** Agents' positions of Train (5-6)-Test (5-6). **(b):** Agents' positions of Train (5-6)-Test (8-9). As shown in the left half side of (a) and (b), ACE perfectly transfers the station position of 5m_vs_6m to 8m_vs_9m, that is, making the agents stand up and down in a line. **(c):** Careful positioning and alternating fire of Train (5-6)-Test (8-9). ACE perfectly transfers the operation of careful positioning and alternating fire of 5m_vs_6m to 8m_vs_9m. Each ally has very little health but kills all enemies. **(d):** Focusing fire in Train (5-6)-Test (5-6). **(e):** Focusing fire in Train (5-6)-Test (8-9). **(f):** Focusing fire in Train (8-9)-Test (8-9). The optimal choice (d) in 5m_vs_6m is all allies focusing fire on one enemy while the optimal choice (f) in 8m_vs_9m is all allies focusing fire on two enemies, since 5 allies can kill an enemy instantly and having 8 agents attacking one agent (e) at the same time is wasteful.