

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

Chuming Li^{1, 2*}, Jie Liu^{2*}, Yinmin Zhang^{1, 2*}, Yuhong Wei³, 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.

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 a^t based on a state s^t is expanded to multiple intermediate states $[s_{a_1}^t, \dots, s_{a_{1:n}}^t]$, named SE-state. The SE-state $s_{a_{1:i}}^t$ 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. 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.

*Equal contribution

†Corresponding author

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

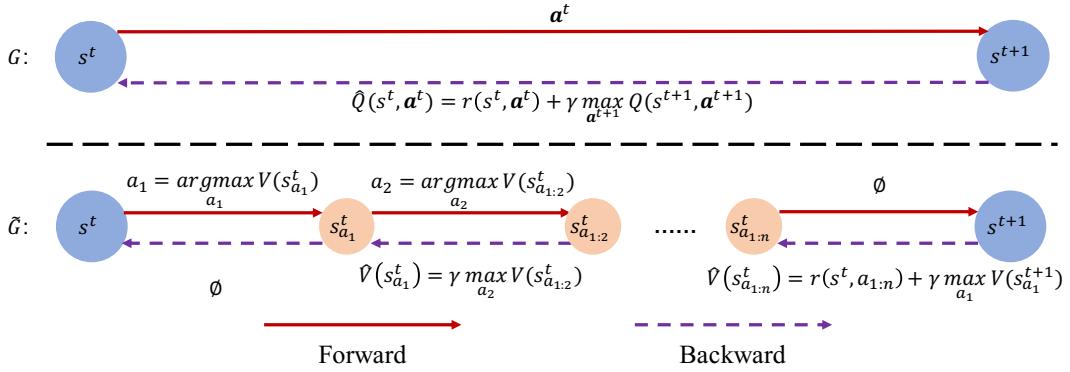


Figure 1: Comparison between the original MMDP (above) and the transformed SE-MDP (below). A single transition in MMDP is expanded to n sequentially expanded states in SE-MDP.

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 embeddings 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 consider 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 2018; 2019; 2020; 2020. To avoid the non-stationary problem, they usually develop different factorized value functions following the IGM principle 2019, which requires 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

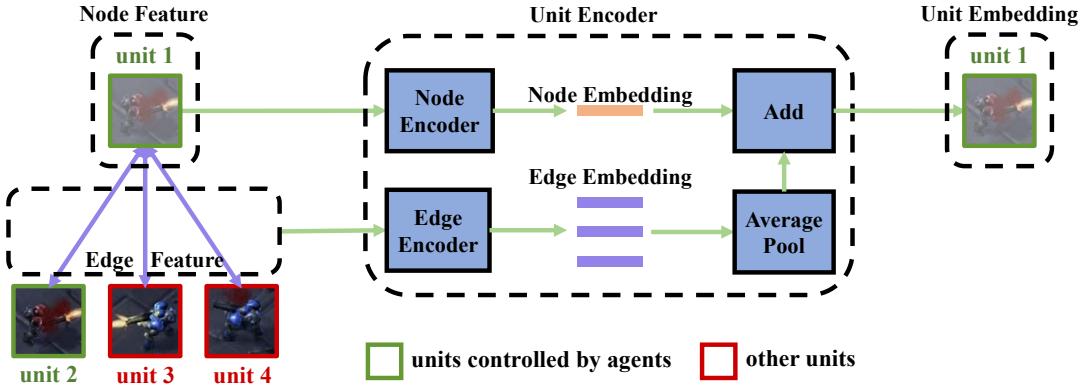


Figure 2: The schematic of the unit encoder. The node embedding is obtained by the node encoder, and the edge embedding (for the unit and its interacted units) is obtained from the edge encoder. The average-pooled edge embedding is added to the node embedding to provide unit embedding.

method that achieves remarkable performance following the action-dependent learning scheme.

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} is the space of global state and \mathcal{N} is the set of n agents. $\mathcal{A} \equiv \mathcal{A}_1 \times \dots \times \mathcal{A}_n$ 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 $a = [a_1, \dots, a_n]$ and the transition function $P(s'|s, a)$, the process transits to the next state s' and returns a reward $r(s, a)$. The target we consider is to learn an optimal policy $\pi(a | s)$ which maximizes the expected return $\mathcal{R} = \mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t r(s^t, a^t)]$.

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 $\tilde{\mathcal{G}}$. In $\tilde{\mathcal{G}}$, the state transits along the decision-making sequence. Specifically, an 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 $s_{a_{1:i}}^t$. At each intermediate transition, an agent i receives its intermediate state $s_{a_{1:i-1}}^t$ and produces its action a_i , then the intermediate state intermediately transits to $s_{a_{1:i}}^t$ with a reward 0. After the last agent n makes decision and the intermediate state intermediately transits

to $s_{a_{1:n}}^t$, a psuedo agent produces an empty action and the intermediate state transits from $s_{a_{1:n}}^t$ to s^{t+1} , with the reward $r(s^t, a^t)$ defined in the original MMDP \mathcal{G} . With the above definition, a transition $(s^t, a^t, r(s^t, a^t), s^{t+1})$ of \mathcal{G} is expanded into a sequence of intermediate transitions $(s^t, a_1^t, 0, s_{a_1}^t), (s_{a_1}^t, a_2^t, 0, s_{a_{1:2}}^t), \dots, (s_{a_{1:n-1}}^t, a_n, r(s^t, a^t), s^{t+1})$ in $\tilde{\mathcal{G}}$. We define $\tilde{\mathcal{G}}$ as the sequential expansion of \mathcal{G} and name this MDP as sequentially expanded MMDP (SE-MDP). Similarly, we define the intermediate state $s_{a_{1:i}}$ as sequentially expanded state (SE-state), of which the space is represented by $\tilde{\mathcal{S}}$.

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 $\tilde{\mathcal{G}}$ circumvents the non-stationary problem, which enables us to easily adopt different single-agent algorithms to solve $\tilde{\mathcal{G}}$. 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 $Q : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{A}|}$ 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 $s_{a_{1:i}}^t$, we can direct intermediate transition to the next

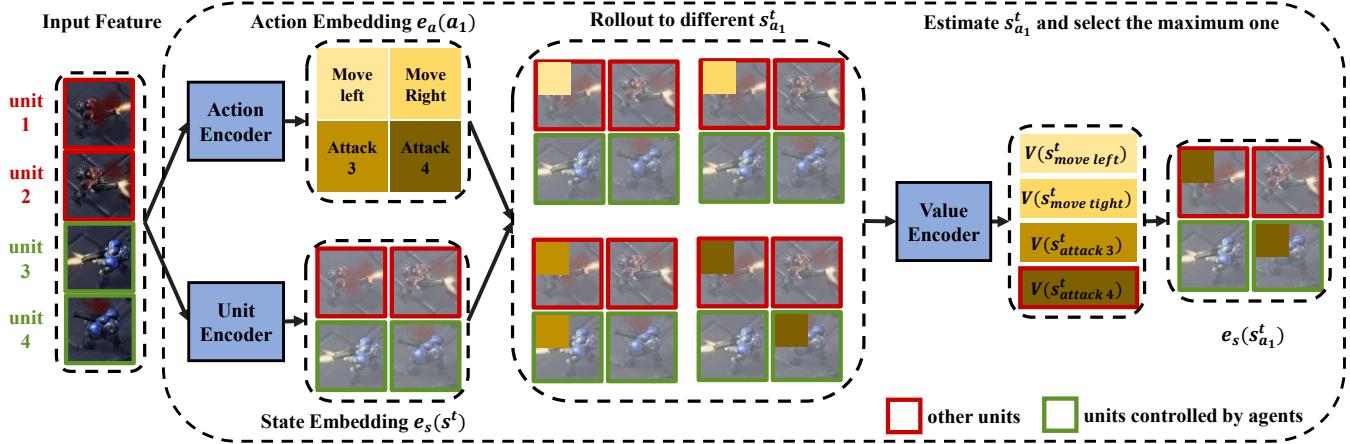


Figure 3: Schematic of the pipeline of ACE, which takes SMAC as an instance. There are four units in the map. Units 1 and 2 are controlled by the RL agent, and units 3 and 4 are enemies controlled by the environment. At first, the initial state embedding is generated, consisting of the initial embedding 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 $e(s_{a_1}^t)$ (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.

SE-state $s_{a_{1:i+1}}^t$ without interacting with the environment. Hence, we take a step forward and use the value function $V : \mathcal{S} \rightarrow \mathbb{R}$ to estimate the return of the SE-state rather than the action, and use the values $V(s_{a_{1:i+1}}^t)$ 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 $s_{a_{1:i}}^t$, we use agent $i+1$'s action space A_{i+1} to roll out to all possible next SE-states $s_{a_{1:i+1}}^t$, and select the action $a_{i+1}^t = \arg \max_{a_{i+1}} V(s_{a_{1:i}, a_{i+1}}^t)$, which leads to the next SE-state with the optimal value $V(s_{a_{1:i+1}}^t)$.

Update with Rollout Our value function V is updated by the standard Bellman backup operator in single agent RL. At an SE-state $s_{a_{1:i}}^t$, to obtain the target value to update the value $V(s_{a_{1:i}}^t)$, we also rollout to all possible next SE-states $s_{a_{1:i+1}}^t$, estimate their values $V(s_{a_{1:i+1}}^t)$ and select the maximum value as the target value. For the final SE-state $s_{a_{1:n}}^t$ in a decision sequence, we roll out at the first SE-state $s_{a_1}^{t+1}$ 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 $\hat{V}(s_{a_{1:i}}^t)$ denoting the Bellman target of $V(s_{a_{1:i}}^t)$.

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

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 $s_{a_{1:i}}^t$ with DNN.

Decomposed State Embedding **Firstly**, a transition $(s^t, a^t, r(s^t, a^t), s^{t+1})$ in the original MMDP \mathcal{G} corresponds to n intermediate transitions in the sequential expansion $\tilde{\mathcal{G}}$ 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 $e_s(s_{a_{1:i}}^t)$ along the sequential decision-making are decomposed into a shared embedding $e_s(s^t)$ of the initial state s^t , as well as a shared set of embeddings $e_a(a_1), \dots, e_a(a_n)$ of available actions a_1, \dots, a_n , all generated by the same action encoder. Then, the state embedding $e_s(s_{a_{1:i}}^t)$ is obtained by combining the initial state embedding $e_s(s^t)$ and the corresponding action embeddings $e_a(a_1), \dots, e_a(a_i)$. 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.

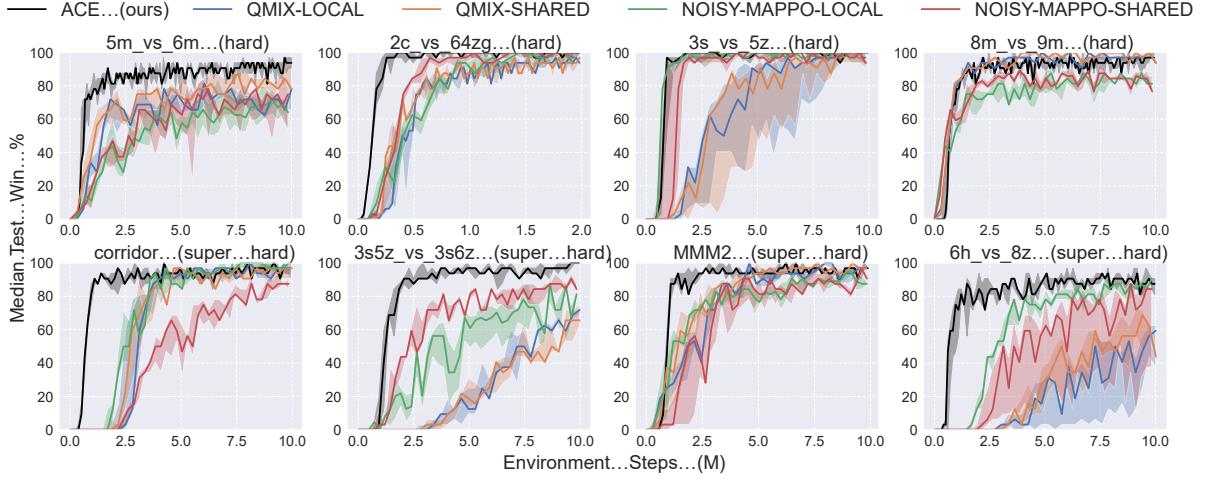


Figure 4: Comparison of ACE against baselines on four super hard and four hard SMAC maps.

Unit-wise State Embedding For state embedding, we use a **unit encoder** to generate the unit-wise embedding $e_u(u_i)$ of each unit u_i in the environment, which forms the initial state embedding $e_s(s^t) = [e_u(u_1), \dots, e_u(u_m)]$. 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 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 $e_s(s_{a_{1:i}}^t)$ 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 $e_a(a_i) = [e_a^a(a_i), e_a^p(a_i)]$. The active embedding $e_a^a(a_i)$ encodes the effect of action a_i on the unit u_i itself, and the passive embedding $e_a^p(a_i)$ encodes the effect of action a_i on the target units. For actions without interaction, it only has an active embedding $e_a^a(a_i)$, formalized by $e_a(a_i) = [e_a^a(a_i)]$.

After the generation of the original state embedding $e_s(s^t) = [e_u(u_1), \dots, e_u(u_m)]$ and the action embedding $e_a(a_1), \dots, e_a(a_i)$, we use an additive combination of the unit and action embedding to construct the state embedding $e(s_{a_{1:i}}^t)$ of the intermediate SE-states $s_{a_{1:i}}^t$, formalized by $e_s(s_{a_{1:i}}^t) = [e_u(u_{1,a_{1:i}}), \dots, e_u(u_{m,a_{1:i}})]$. The element

$e_u(u_{j,a_{1:i}})$ denotes the combination of the initial unit embedding $e_u(u_j)$ 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 $e_a^a(a_i)$ is added on the unit embedding $e_u(u_j)$ of its executor u_i ; if a_i involves an interaction with some target unit, its passive action embedding $e_a^p(a_i)$ is added on $e_u(u_j)$ to describe the interaction. The definition of $e_u(u_{j,a_{1:i}})$ is formalized by:

$$e_u(u_{j,a_{1:i}}) = \begin{cases} e_u(u_j) + \sum_{e_a^p(a_k) \in P(a_{1:i})_j} e_a^p(a_k), & \text{if } i < j \\ e_u(u_j) + e_a^a(a_j) + \sum_{e_a^p(a_k) \in P(a_{1:i})_j} e_a^p(a_k), & \text{if } i \geq j \end{cases} \quad (2)$$

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

In this paper, the passive embedding $e_a^p(a_i)$ 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 $e_a^a(a_i)$ is defined as a learnable parameterized vector, because it is added to the embedding $e_u(u_i)$ 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 $e_s(s_{a_{1:i}}^t) = [e_u(u_{1,a_{1:i}}), \dots, e_u(u_{m,a_{1:i}})]$ 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 $\tilde{\mathcal{G}}$.

Metric	Map	VDN	QMIX	QTRAN	ACE
Steps	5×5	0.78	0.77	0.60	0.04
	7×7	0.90	0.87	1.02	0.07
Samples (M)	5×5	0.19	0.19	0.17	0.09
	7×7	1.97	1.81	1.68	1.01

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.

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 fairness in this section.

Spiders-and-Fly. The Spiders-and-Fly problem is first proposed in 2019, where two RL-controlled spiders cooperate to catch a fly in a 2D grid and the fly avoids spiders’ neighboring locations. 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, RL-controlled ally units in StarCraft play against enemy units with built-in rules. They use cooperative micro-tricks to win. 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). A harder environment than SMAC, with a large action space and sparse reward. Agents coordinate to organize attacks; only scoring leads to rewards. We control the left team players, excluding the goalkeeper. We evaluate our method on two challenging scenarios, using standard 19 actions and similar observations to CDS. The final reward is +100 for winning, -1 for ball/player returning to 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.

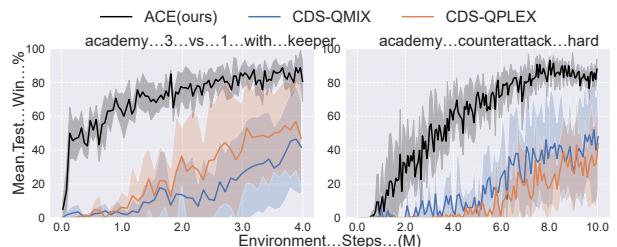


Figure 5: Comparison of ACE against baseline on GRF.

Performance

Spiders-and-Fly We compare ACE with three value factorization methods: QTRAN 2019, QMIX 2018 and VDN 2017, on 5x5 and 7x7 grids. 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 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. NOISY-MAPPO 2021 serves 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 the two baseline algorithms are originally designed for partially observable scenarios, where each agent only uses 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.

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.

Further Analysis

Ablation: What Matters in the Components of ACE?

To understand ACE’s performance, we make ablations and modifications. We compare ACE-w/o-IA (without interaction-

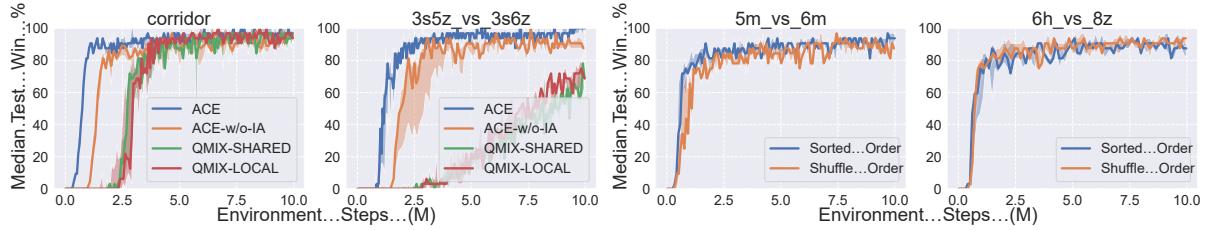


Figure 6: (a): Comparison of ACE and ACE-w/o-IA against QMIX on corridor and 3s5z_vs_3s6z. (b): Comparison of sorted order against shuffle order of ACE.

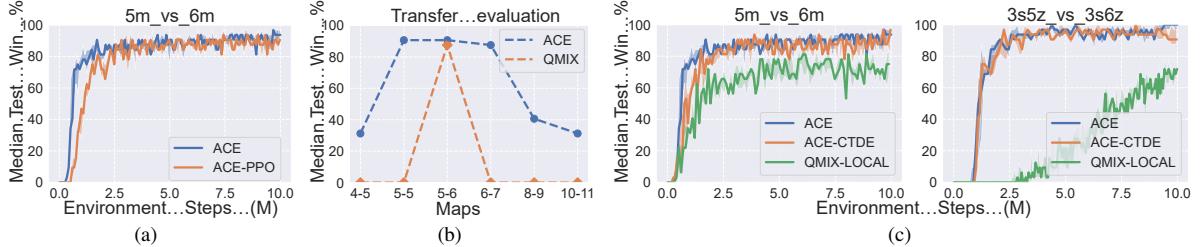


Figure 7: (a): Comparision of ACE and ACE-PPO on 5m_vs_6m. (b): Transfer from 5m_vs_6m to maps of different numbers of agents. 'x-y' represents xm_vs_ym. ACE can achieve remarkable performance in the zero-shot setting, regardless of whether agent number increases or decreases. (c): Comparison of ACE-CTDE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

aware action embedding) to ACE and fine-tuned QMIX. ACE-w/o-IA still outperforms QMIX but is worse than ACE. We also study decision-making order, comparing Shuffle Order (random) and Sorted Order (by unit types and locations). As shown in Figure 6, the two settings have little difference in performance in two SMAC maps, which validates that 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 co-operation 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. 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 an 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. Typically, CTDE methods employ individual value functions based on local observations and a joint value function using 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, to distill the optimal joint action generated via the sequential rollout in ACE, into an additional individual value function $Q(o_i^t, a_i)$. The counterfactual distillation is formalized by $\hat{Q}(o_i^t, a_i) = V(s_{a_i, a_{i-}}^t)$. $\hat{Q}(o_i^t, a_i)$ is the target to update $Q(o_i^t, a_i)$ 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 gratefully 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*.

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.; Zajac, 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.; 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*.

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.

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

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, 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*.