An Adaptive Entropy-Regularization Framework for Multi-Agent Reinforcement Learning

Woojun Kim [1] Youngchul Sung [1]

Abstract

In this paper, we propose an adaptive entropy-regularization framework (ADER) for multi-agent reinforcement learning (RL) to learn the adequate amount of exploration of each agent for entropy-based exploration. In order to derive a metric for the proper level of exploration entropy for each agent, we disentangle the soft value function into two types: one for pure return and the other for entropy. By applying multi-agent value factorization to the disentangled value function of pure return, we obtain a metric to determine the relevant level of exploration entropy for each agent, given by the partial derivative of the pure-return value function with respect to (w.r.t.) the policy entropy of each agent. Based on this metric, we propose the ADER algorithm based on maximum entropy RL, which controls the necessary level of exploration across agents over time by learning the proper target entropy for each agent. Experimental results show that the proposed scheme significantly outperforms current state-of-the-art multi-agent RL algorithms.

1. Introduction and Motivation

RL is an effective approach to solving decision-making problems such as robot control (Hester et al., 2012; Ebert et al., 2018), traffic light control (Wei et al., 2018; Wu et al., 2020) and games (Mnih et al., 2015b; Silver et al., 2017). The goal of RL is to find an optimal policy that maximizes expected return. To guarantee convergence of model-free RL, the assumption that each element in the joint state-action space should be visited infinitely often is required (Sutton & Barto, 2018), but this is impractical due to large state-action spaces in real-world problems. Thus, efficient exploration has been one of the core problems of RL. Furthermore, the given time for learning is limited in real-world problems. Hence, the learner cannot just do exploration only but needs to exploit its own policy based on its experiences so far. For better overall learning, the learner should balance exploration and exploitation over time. This is called *exploration-exploitation trade-off* in RL, which has been a challenging topic in single-agent RL.

The problem of exploration-exploitation trade-off becomes more challenging in multi-agent RL (MARL) because the volume of state-action space grows exponentially as the number of agents increases. Furthermore, the necessity and benefit of exploration at a gvien time can be different across agents and even one agent's exploration can hinder other agents' exploitation, resulting in a situation in which simultaneous exploration of multiple agents can make learning unstable. Thus, the balance of exploration and exploitation *across multiple agents* should also be considered for MARL in addition to that across the typical time dimension. We refer to this problem as *multi-agent exploration-exploitation trade-off*. In order to handle the problem of multi-agent exploration-exploitation trade-off, we need to control the amount of exploration of each agent adaptively and learn this amount across agents (i.e., agent dimension) and over time (i.e., time dimension).

To see the necessity of such adaptive exploration-exploitation trade-off control in MARL, let us consider a modified continuous cooperative matrix game (Peng et al., 2021), described in Fig. 1. As seen in Fig. 1, there is a connected narrow path from the origin $(0,0)$ to $(0.6,0.55)$, consisting of two subpaths: one from $(0,0)$ to $(0.6,0)$ and the other from $(0.6,0)$ to $(0.6,0.55)$. There is a circle with center at $(0.6,0.6)$ and radius 0.05. The goal of this game is to

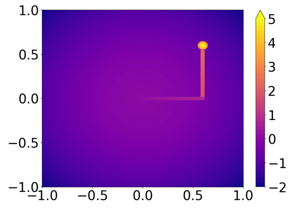


Figure 1. Reward surface in the considered matrix game.

[1] School of Electrical Engineering, KAIST, Daejeon 34141, Republic of Korea. Correspondence to: Youngchul Sung <ycsung@kaist.ac.kr>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

move the 2-D position $\boldsymbol{a} = (a^1, a^2)$ from $(0, 0)$ to the target circle at $(0.6, 0.6)$ along the narrow path. The 2-D position is controlled by two agents, where Agent i controls $a^i \in [-1, 1]$ as its action, $i = 1, 2$. The shared reward is determined by the joint action $\boldsymbol{a} = (a^1, a^2)$, and the reward surface is given in Fig. 1. The reward increases only along the path as the position (a^1, a^2) approaches the center of the target circle and the maximum reward is 5. There is a penalty if the joint action yields the position outside the path or the target circle, and the penalty value increases as the outside position is farther from the origin $(0, 0)$. Then, the agents should learn to reach the circle along the path, starting from the origin with initial action pair $\boldsymbol{a} = (0, 0)$. Even though this game is stateless, exploration in the action space is required to find the action (0.6, 0.6). One may think that one can find the optimal joint action all at once when the action near the circle is selected. However, the action starting with (0,0) cannot jump to (0.6, 0.6) since we use function approximators for the policies and train them based on stochastic gradient descent with a small step size. The action should be trained to reach the circle along the two subpaths sequentially. In the beginning, to go through the first subpath, a^2 (i.e., y-axis movement) should not fluctuate from 0, and a^1 should be trained to increase up to 0.6. In this phase, if a^2 explores too much, the positive reward is rarely obtained. Then, a^1 is not trained to increase up to 0.6 because of the penalty. Once the joint action is trained to $(0.6, 0)$, on the other hand, the necessity and benefit of exploration are changed. In this phase, a^1 should keep its action at 0.6, whereas a^2 should be trained to increase up to 0.6. Thus, it is necessary to control the trade-off between exploitation and exploration across agents. In addition, we need to update this trade-off over time because the required trade-off changes during the learning process. We will see that such adaptive control of the trade-off between exploitation and exploration is beneficial not only to this particular example but also to many general MARL tasks in Sec. 4.

Although there exist many algorithms for better exploration in MARL (Mahajan et al., 2019; Liu et al., 2021a; Zhang et al., 2021; Kim et al., 2023), the research on the aforementioned multi-agent exploration-exploitation trade-off has not been investigated much yet. In this paper, we propose a new framework that can adaptively learn appropriate levels of exploration for multiple agents over time, considering the time-varying multi-agent exploration-exploitation trade-off. To build a framework for such adaptive multi-agent exploration-exploitation trade-off, we employ the widely-used entropy-based exploration (Haarnoja et al., 2018a;b). Under the entropy-based exploration, we allocate higher target entropy values to the agents who have larger benefits from exploration and need more exploration, and allocate lower target entropy values to the agents who have less benefits from exploration, inclining to more exploitation. Then, under the constraint of fixed total target entropy value sum across all agents, such target entropy value assignment will realize multi-agent exploration-exploitation trade-off.

The key question of this approach is how to measure the benefit of more exploration for each agent. To devise a metric for this, one can exploit value functions. However, we believe that soft value functions associated with entropy-based RL are not appropriate since these functions estimate the sum of reward and policy entropy, and thus we cannot pinpoint the impact of the policy entropy on the pure reward sum which is the ultimate goal of RL. Hence, in order to devise a metric for the benefit of more exploration, we adopt the method of disentanglement between exploration and exploitation (Beyer et al., 2019; Han & Sung, 2021) to decompose the joint soft value function into two types: one for the return and the other for the entropy sum. Based on this disentanglement, we propose *the partial derivative of the joint value function of pure return w.r.t. policy action entropy* as the metric for the benefit of more exploration for each agent. The intuition behind this choice is clear for entropy-based exploration: Agents with higher gradient of joint pure-return value w.r.t. their action entropy should increase their target action entropy resulting in higher exploration level in order to contribute more to pure return. Then, under the constraint of total target entropy sum across all agents, which we will impose, the target entropy of agents with lower gradient of joint pure-return value w.r.t. their action entropy will be reduced, leading towards exploitation. Thus, multi-agent exploration-exploitation trade-off can be achieved. However, the computation of the partial derivative of the joint value function w.r.t. policy action entropy is not trivial in the discrete-action case. We circumvent this difficulty successfully by adopting an actor-critic architecture and value decomposition and exploiting the imposed architecture. The details will follow in the upcoming sections.

2. Background

Basic setup We consider a decentralized partially observable MDP (Dec-POMDP), which describes a fully cooperative multi-agent task (Oliehoek & Amato, 2016). A Dec-POMDP is defined by a tuple $< \mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}, \mathcal{P}, \{\Omega_i\}, \mathcal{O}, r, \gamma >$, where $\mathcal{N} = \{1, \cdots, N\}$ is the set of agents, $\mathcal{S}$ is the set of states, $\mathcal{A}_i$ is the set of actions of Agent i, $\mathcal{P}$ is the transition probability, Ω_i is the set of observations of Agent i, $\mathcal{O}$ is the observation function, r is the reward function. and γ is the discount factor. At time step t, Agent $i \in \mathcal{N}$ makes its own observation $o_t^i \in \Omega_i$ according to the observation function $\mathcal{O}(s, i) : \mathcal{S} \times \mathcal{N} \to \Omega_i : (s_t, i) \mapsto o_t^i$, where $s_t \in \mathcal{S}$ is the global state at time step t. Agent i selects action $a_t^i \in \mathcal{A}_i$

to form a joint action $\boldsymbol{a_t} = \{a_t^1, a_t^2, \cdots, a_t^N\}$. The joint action yields the next global state s_{t+1} according to the transition probability $\mathcal{P}(\cdot|s_t, \boldsymbol{a}_t)$ and a joint shared reward $r_t = r(s_t, \boldsymbol{a}_t)$. Each agent i has an observation-action history $\tau^i \in (\Omega_i \times \mathcal{A}_i)^*$ and trains its policy $\pi^i(a^i|\tau^i)$ to maximize the return $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$. We consider the framework of centralized training with decentralized execution (CTDE), where decentralized policies are trained with additional information including the global state in a centralized way during the training phase (Oliehoek et al., 2008). Thus, the joint policy $\boldsymbol{\pi}_t$ is given as the product form $\boldsymbol{\pi}_t = \prod_{i=1}^N \pi_t^i$.

Maximum Entropy RL and Entropy Regularization Maximum entropy RL aims to promote exploration by finding an optimal policy that maximizes the cumulative sum of reward and entropy (Haarnoja et al., 2017; 2018a;b). The objective function of maximum entropy RL is given by

$$J_{MaxEnt}(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t (r_t + \alpha \mathcal{H}(\pi(\cdot|s_t)))\right], \quad (1)$$

where $\mathcal{H}(\cdot)$ is the entropy function and α is a temperature parameter. Soft actor-critic (SAC, Haarnoja et al. (2018a)) is an off-policy actor-critic algorithm which efficiently solves the maximum entropy RL problem (1) based on soft policy iteration, composed of soft policy evaluation and soft policy improvement. For this, the soft Q-function is defined as the sum of the return and the total future entropy, i.e., $Q^{\pi}(s_t, a_t) := r_t + \mathbb{E}_{\tau_{t+1}\sim\pi}\left[\sum_{l=t+1}^{\infty} \gamma^{l-t}(r_l + \sum_{i=1}^N \alpha\mathcal{H}(\pi(\cdot|s_l)))\right]$. In the soft policy evaluation step, for a given policy π, the soft Q-function is estimated by repeatedly applying the soft Bellman backup operator $\mathcal{T}_{sac}^{\pi}$ to an estimate function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where the soft Bellman backup operator is given by $\mathcal{T}_{sac}^{\pi} Q(s_t, a_t) = r_t + \gamma\mathbb{E}_{s_{t+1}}[V(s_{t+1})]$ with $V(s_t) = \mathbb{E}_{a_t\sim\pi}[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t)]$. In the soft policy improvement step, the policy is updated with the evaluated soft Q-function as $\pi_{new} = \arg\max_{\pi} \mathbb{E}_{a_t\sim\pi}\left[Q^{\pi_{old}}(s_t, a_t) - \alpha \log \pi(a_t|s_t)\right]$. By iterating the soft policy evaluation and the soft policy improvement, SAC converges to an optimal policy that maximizes (1) within the considered policy class in the case of finite MDPs. SAC also works effectively for large MDPs with function approximation.

One issue with SAC is the adjustment of the hyperparameter α in (1), which controls the relative importance of the entropy w.r.t. the reward. The magnitude of the reward depends not only on tasks but also on the policy which improves over time during the training phase. Thus, Haarnoja et al. (2018b) proposed a method to adjust the temperature parameter α over time to guarantee the minimum average entropy at each time step. For this, they reformulated the maximum entropy RL as the following entropy-regularized optimization:

$$J_{ER}(\pi_{0:T}) = \mathbb{E}_{\pi_{0:T}}\left[\sum_{t=0}^{T} r_t\right] \quad (2)$$
$$\text{s.t. } \mathbb{E}_{(s_t,a_t)\sim\pi_t}\left[-\log(\pi_t(a_t|s_t))\right] \geq \mathcal{H}_0,$$

where $\mathcal{H}_0$ is the target entropy. In order to optimize the objective (2), the technique of dynamic programming can be used (Haarnoja et al., 2018b), i.e., $\max_{\pi_{t:T}} \mathbb{E}[\sum_{i=t}^T r_i] = \max_{\pi_t}\left\{\mathbb{E}[r_t] + \max_{\pi_{t+1:T}} \mathbb{E}[\sum_{i=t+1}^T r_i]\right\}$. Starting from time step T, we obtain the optimal policy $\pi_{0:T}^*$ and $\alpha_{0:T}^*$ by applying backward recursion. That is, we begin with the constrained optimization at time step T, given by

$$\max_{\pi_T} \mathbb{E}[r_T] \;\text{ s.t. } \mathbb{E}_{(s_T,a_T)\sim\pi_T}\left[-\log(\pi_T(a_T|s_T))\right] \geq \mathcal{H}_0$$

and convert this problem into the Lagrangian dual problem as

$$\min_{\alpha_T}\max_{\pi_T} \mathbb{E}[r_T - \alpha_T \log \pi_T(a_T|s_T)] - \alpha_T\mathcal{H}_0$$
$$= \min_{\alpha_T} \mathbb{E}[-\alpha_T \log \pi_T^*(a_T|s_T) - \alpha_T\mathcal{H}_0]. \quad (3)$$

Thus, the optimal temperature parameter α_T^* at time step T, which corresponds to the Lagrangian multiplier, can be obtained by solving the problem (3). Then, the backward recursion can be applied to obtain optimal α at time step t based on the Lagrange dual problem:

$$\alpha_t^* = \arg\min_{\alpha_t} \underbrace{\mathbb{E}_{a_t\sim\pi_t^*}[-\alpha_t \log \pi_t^*(a_t|s_t) - \alpha_t\mathcal{H}_0]}_{:=J(\alpha_t)}, \quad (4)$$

where π_t^* is the maximum entropy policy at time step t. Here, by minimizing the loss function $J(\alpha)$, α is updated to increase (or decrease) if the entropy of policy is lower (or higher) than the target entropy $\mathcal{H}_0$. In the infinite-horizon case, the discount factor γ is included and π_t^* is replaced with the current approximate maximum entropy solution by SAC. In this way, the soft policy iteration of SAC is combined with the α adjustment based on the loss function $J(\alpha)$ defined in (4). This algorithm effectively handles the reward magnitude change over time during training (Haarnoja et al., 2018b). Hence, one needs to set only the target entropy $\mathcal{H}_0$ for each task and then α is automatically adjusted over time for the target entropy.

Related Works We here focus on the entropy-based MARL. Other related works regarding multi-agent exploration are provided in Appendix E. There exist previous works on entropy-based MARL. Zhou et al. (2020) proposed an actor-critic algorithm, named LICA, which learns implicit credit assignment and regularizes the action entropy by dynamically controlling the magnitude of the gradient regarding entropy to address the high sensitivity of the temperature parameter caused by the curvature of derivative of entropy. LICA allows multiple agents to perform a consistent level of exploration. However, LICA does not maximize the cumulative sum of entropy but regularize the action entropy. Zhang et al. (2021) proposed an entropy-regularized MARL algorithm, named FOP, which introduces a constraint that the entropy-regularized optimal joint policy is decomposed into the product of the optimal individual policies. FOP introduced a weight network to determine individual temperature parameters. Zhang et al. (2021) considered individual temperature parameters for updating policy, but in practice, they used the same value (for all agents) which is annealed during training for the temperature parameters. This encourages multiple agents to focus more on exploration in the beginning of training, which considers exploration-exploitation only in time dimension in a heuristic way.

The key point is that the aforementioned algorithms maximize or regularize the entropy of the policies to encourage the same level of exploration across the agents. Such exploration is still useful for several benchmark tasks but cannot handle the multi-agent exploration-exploitation trade-off. Furthermore, in the previous methods, the joint soft Q-function defined as the total sum of return and entropy is directly factorized by value decomposition, and hence the return is not separated from the entropy in the Q-value. From the perspective of one agent, however, the contribution to the reward and that to the entropy can be different. What we actually need to assess the goodness of a policy is the return estimate, which is difficult to obtain by such unseparated factorization.

3. Methodology

To address the aforementioned problems, we propose an **AD**aptive **E**ntropy-**R**egularization framework (ADER), which can balance exploration and exploitation across multiple agents by learning the target entropy for each agent.

3.1. Adaptive Entropy-Regularized MARL

For entropy-based exploration in MARL, one can adopt the entropy-constrained objective defined in (2) and extend it to multi-agent systems. A simple extension is to maximize the team reward while keeping the policy entropy of each agent above the same target entropy. For the sake of convenience, we call this scheme simple entropy-regularization for MARL (SER-MARL). However, SER-MARL cannot handle the multi-agent exploration-exploitation trade-off because the amounts of exploration for all agents are the same. One can also consider the use of different but fixed target entropy values for multiple agents. However, this case cannot handle the time-varying behavior of multi-agent exploitation-exploration trade-off, discussed in Sec. 1 with Fig. 1. Hence, to realize the time-varying multi-agent exploration-exploitation trade-off, we consider the following optimization problem:

$$\begin{aligned}\max_{\boldsymbol{\pi}} \mathbb{E}_{\boldsymbol{\pi}} &\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \\ &\text{s.t. } \mathbb{E}_{(s_t, \boldsymbol{a_t}) \sim \boldsymbol{\pi}} \left[-\log(\pi_t^i(a_t^i|\tau_t^i))\right] \geq \mathcal{H}_i, \ \forall i \in \mathcal{N}, \\ &\quad \sum_{j=1}^{N} \mathcal{H}_j = \mathcal{H}_0, \end{aligned} \tag{5}$$

where $\boldsymbol{\pi} = (\pi^1, \cdots, \pi^N)$, $\mathcal{H}_i$ is the target entropy of Agent i, and $\mathcal{H}_0$ is the total sum of all target entropies. The key point here is that *we fix the target entropy sum as $\mathcal{H}_0$ but each $\mathcal{H}_i$ is adaptive and learned over time.* Note that the total entropy budget $\mathcal{H}_0$ is shared by all agents. Therefore, when some agents' target entropy values are high for more exploration, the target entropy values of other agents should be low, leading to more exploitation, due to the fixed total entropy budget. Thus, the exploitation-exploration trade-off across agents (i.e., agent dimension) can be captured. The main challenge in this approach is *how to learn individual target entropy values $\mathcal{H}_1, \cdots, \mathcal{H}_N$ over time (i.e., time dimension) as the learning progresses.*

The learning of $\mathcal{H}_1, \cdots, \mathcal{H}_N$ is closely related to the way how the problem (5) is solved. To solve (5), one can simply extend the method in (Haarnoja et al., 2018b) to the MARL case. That is, one can first consider a finite-horizon case with terminal time step T, apply approximate dynamic programming and the Lagrange multiplier method, obtain the update formula at time step t, and then relax to the infinite-horizon case by introducing the discount factor, as in (Haarnoja et al., 2018b). For soft policy iteration involved in this approach, one can define the joint soft Q-function as

$$\begin{aligned} &Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) := \\ &\quad r_t + \mathbb{E}_{\tau_{t+1} \sim \pi}\left[\sum_{l=t+1}^{\infty} \gamma^{l-t}(r_l + \sum_{i=1}^{N} \alpha^i \mathcal{H}(\pi^i(\cdot|\tau_l^i)))\right]. \end{aligned} \tag{6}$$

Then, one can estimate this joint soft Q-function based on the following Bellman backup operator:

$$\mathcal{T}_{JT}^{\pi} Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) := r_t + \gamma \mathbb{E}_{\tau_{t+1}}\left[V(s_{t+1}, \boldsymbol{\tau_{t+1}})\right],$$

where $V_{JT}(s_t, \boldsymbol{\tau_t}) = \mathbb{E}_{a_t \sim \boldsymbol{\pi}}\big[Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) - \sum_{i=1}^{N} \alpha^i \log \pi(a_t^i|\tau_t^i)\big]$. However, solving (5) directly based on the joint reward-entropy-aggregated soft Q-function in (6) and the corresponding Bellman operator $\mathcal{T}_{JT}^{\pi}$ does not suit our purpose. Note that the value function is an effective indicator of the goodness of a policy-generated action. When the reward-entropy-aggregated soft value function is used, it is difficult to pinpoint each agent's contribution, i.e., credit, to the return, i.e., the pure reward sum, which is the ultimate goal of RL, due to the inseparability of reward and entropy.

3.2. Disentangled Exploration and Exploitation

In order to measure the impact of each agent's policy on the joint global return and enable us to derive a metric for the benefit of more exploration for each agent, we disentangle the return from the entropy (Beyer et al., 2019; Han & Sung, 2021). Specifically, we decompose the joint soft Q-function into two types: One for the return and the other for the entropy. That is, the joint soft Q-function is decomposed as

$$Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) + \sum_{i=1}^N \alpha^i Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}), \quad (7)$$

where $Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ is the joint action value function for the return and $Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ is the joint action value function for the future entropy of Agent i's policy:

$$Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = r_t + \mathbb{E}_{\tau_{t+1}\sim\boldsymbol{\pi}}\left[\sum_{l=t+1}^{\infty} \gamma^{l-t} r_l\right], \quad (8)$$

$$Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = \mathbb{E}_{\boldsymbol{\tau_{t+1}}\sim\boldsymbol{\pi}}\left[\sum_{l=t+1}^{\infty} \gamma^{l-t}\mathcal{H}(\pi^i(\cdot|\tau^i_t))\right],$$

for $i = 1, \cdots, N$. Then, for $Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ and $Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$, we define their corresponding Bellman backup operators as

$$\mathcal{T}^\pi_R Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) := r_t + \gamma\mathbb{E}\left[V^R_{JT}(s_t, \boldsymbol{\tau_{t+1}})\right]$$
$$\mathcal{T}^\pi_{H,i} Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) := \gamma\mathbb{E}\left[V^{H,i}_{JT}(s_t, \boldsymbol{\tau_{t+1}})\right], \quad (9)$$

where $V^R_{JT}(s_t, \boldsymbol{\tau_t}) = \mathbb{E}_{\boldsymbol{a_t}}\left[Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\right]$ and $V^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}) = \mathbb{E}_{\boldsymbol{a_t}}\left[Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) - \alpha^i \log\pi(a^i_t|\tau^i_t)\right]$ are the joint value functions regarding return and entropy, respectively.

Proposition 3.1. *The disentangled Bellman operators $\mathcal{T}^\pi_R$ and $\mathcal{T}^\pi_{H,i}$ in (15) are contractions.*

Proof: See Appendix A.

Due to Proposition 3.1, the disentangled value functions can be estimated by repeatedly applying their corresponding Bellman operators.

Let us postpone the explanation of how to learn the target entropies $\mathcal{H}_1, \cdots, \mathcal{H}_N$ to the next subsection and explain how to obtain the policy and the temperature parameters for given target entropy values with the disentangled action value functions here.

Policy and Temperature Parameters for Given Target Entropies: With the disentangled joint action value functions and their estimates, the joint policy $\boldsymbol{\pi}_t = \prod_{i=1}^N \pi^i$ and the temperature parameters can be obtained as functions of $\mathcal{H}_1, \cdots, \mathcal{H}_N$ by using a similar technique to that in (Haarnoja et al., 2018b) based on dynamic programming and Lagrange multiplier. That is, we first consider the finite-horizon case and apply dynamic programming with backward recursion:

$$\max_{\boldsymbol{\pi}_{t:T}} \mathbb{E}\left[\sum_{i=t}^T r_i\right] = \max_{\pi_t}\left(\mathbb{E}[r_t] + \max_{\pi_{t+1:T}}\left(\mathbb{E}[\sum_{i=t+1}^T r_i],\right)\right)$$
$$\text{s.t. } \mathbb{E}_{(s_t,\boldsymbol{a_t})\sim\boldsymbol{\pi}_t}\left[-\log(\pi^i_t(a^i_t|\tau^i_t))\right] \geq \mathcal{H}_i, \ \ \forall t, \forall i.$$

We can obtain the optimal policy and the temperature parameters by recursively solving the dual problem from the last time step T by using the technique of Lagrange multiplier. At time step t, the optimal policy is obtained for given temperature parameters, and the optimal temperature parameters are computed based on the obtained optimal policy as follows:

$$\boldsymbol{\pi}^*_t = \underset{\boldsymbol{\pi}_t=\prod_i \pi^i_t}{\arg\max}\, \mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}_t}\Bigg[\underbrace{Q^{R*}_{JT}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}_t)}_{(a)} + \sum_{i=1}^N \alpha^i_t \underbrace{(Q^{H,i*}_{JT}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}_t) - \log\pi^i_t(a^i_t|\tau^i_t))}_{(b)}\Bigg] \quad (10)$$

$$\alpha^{i*}_t = \underset{\alpha^i_t}{\arg\min}\, \mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}^*_t}\left[-\alpha^i_t \log\pi^{i*}_t(a^i_t|\tau^i_t) - \alpha^i_t\mathcal{H}_i\right], \forall i. \quad (11)$$

where Q^{R*}_{JT} and $Q^{H,i*}_{JT}$ are Agent i's joint action value function for the return and the future entropy at time step t, respectively. In the infinite-horizon case, (10) and (11) provide the update formulae at time step t, and the optimal policy is replaced by the current approximate multi-agent maximum-entropy solution with estimated action value functions, which can be obtained by extending SAC to MARL. Note that maximizing the term (a) in (10) corresponds to the ultimate goal of MARL, i.e., the expected return. On the other hand, maximizing the term (b) in (10) corresponds to enhancing exploration of Agent i. Note that the optimization (10) is equivalent to $\boldsymbol{\pi}^*_t = \arg\max_{\boldsymbol{\pi}_t} \mathbb{E}_{\boldsymbol{a}_t\sim\boldsymbol{\pi}_t}[Q^*_{JT}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}_t) - \log\boldsymbol{\pi}_t(\boldsymbol{a}_t|\boldsymbol{\tau}_t)]$ because of (7). Thus, (10) and (11) are an extension of (Haarnoja et al., 2018a;b) to MARL with the joint action value function being disentangled.

3.3. Learning Individual Target Entropy Values

Now, we describe how to learn the target entropy for each agent over time, which is the main contribution of this paper.

In our formulation (5), the amount of exploration of Agent i is controlled by the target entropy value $\mathcal{H}_i$ under the sum constraint $\sum_{j=1}^N \mathcal{H}_j = \mathcal{H}_0$. Note that the ultimate control variables of our method are $\mathcal{H}_1, \cdots, \mathcal{H}_N$ since the policy and the temperature parameters are given once $\mathcal{H}_1, \cdots, \mathcal{H}_N$ are determined as seen in (10) and

(11). Here, we want higher targe entropy values to be assigned to the agents with higher benefits from more exploration. Then, the target entropy values of the agents with lower benefit from exploration are reduced due to the sum constraint. If $\mathcal{H}_i$ is reduced from the balance point $\mathbb{E}_{a_t \sim \pi_t^{i*}}[-\log \pi_t^{i*}] = \mathcal{H}_i$, then α_t^i will be reduced due to the form of $\arg\min_{\alpha_t^i} \alpha_t^i(\mathbb{E}_{a_t \sim \pi_t^i}[-\log \pi_t^{i*}] - \mathcal{H}_i)$ in (11) since $\mathbb{E}_{a_t \sim \pi_t^i}[-\log \pi_t^{i*}] - \mathcal{H}_i > 0$ for the reduced $\mathcal{H}_i$. Then, in turn, the policy objective (10) with reduced α_t^i will focus more on the pure return term $\mathbb{E}_{\boldsymbol{a}_t \sim \boldsymbol{\pi}_t}[Q_{JT}^{R*}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}_t)]$, i.e., exploitation.

For such control of multi-agent exploration-exploitation trade-off, we need to measure the benefit of higher target entropy value for each agent. Considering the fact that the ultimate goal of RL is to maximize the return and the entropy intrinsic reward is added to achieve this ultimate goal in the end, we propose the *partial derivative* $\partial V_{JT}^R / \partial \mathcal{H}(\pi_t^i)$ *at time* t to assess the benefit of increasing the target entropy $\mathcal{H}_i$ of Agent i for more exploration at time t, where the value function of return V_{JT}^R is readily obtained from Q_{JT}^R due to disentanglement. Note that $\partial V_{JT}^R / \partial \mathcal{H}(\pi_t^i)$ is a natural metric to assess the benefit of more exploration for entropy-based exploration. It denotes the change in the joint pure-return value w.r.t. the differential increase in Agent i's policy action entropy, and can be viewed as the *credit of the differential increase of Agent i's policy entropy to the joint return*. Suppose that $\partial V_{JT}^R / \partial \mathcal{H}(\pi_t^i) > \partial V_{JT}^R / \partial \mathcal{H}(\pi_t^j)$ for two agents i and j. Then, if we update two policies π_t^i and π_t^j to two new policies so that the entropy of each of the two policies is increased by the same amount $\Delta\mathcal{H}$, then Agent i contributes more to the pure return than Agent j. Then, under the total entropy sum constraint, the target entropy of Agent i should be assigned higher than that of Agent j for higher return. Furthermore, when this quantity for a certain agent is highly negative, increasing the target entropy for this agent can decrease the joint (return) value significantly, which implies that exploration of this agent can hinder other agents' exploitation. Therefore, we allocate higher target entropy to agents whose $\partial V_{JT}^R / \partial \mathcal{H}(\pi_t^i)$ is larger.

On top of this basic assignment principle, we add some countermeasure to prevent the case in which most of $\mathcal{H}_0$ is assigned to a few dominant agents, and the target entropy values of the remaining agents reduce to zero and they lose the opportunity of exploration in the early stage of learning. One way to prevent this is to set a lower bound on $\mathcal{H}_i$ so that $\mathcal{H}_i \geq \mathcal{H}_{min}, \forall i$ for some $\mathcal{H}_{min}$ throughout the update of $\mathcal{H}_i$. Another way is more direct. Note that $\mathcal{H}_i$ affects α_t^i, and α_t^i affects the policy through the α_t^i in the soft policy update formula (10). For $\alpha_t^i = 0$, there is no exploration by entropy for Agent i. Hence, we can lower bound α_t^i itself above zero, i.e., we clip α_t^i from below to guarantee a certain level of exploration and the softness of the policy for every agent. We choose the latter option in this paper.

With the above guidelines, we construct the adaptation method for $\mathcal{H}_i, \forall i$ as follows. At each time step t, we compute the following N-dimensional vector from the samples in the replay buffer:

$$\boldsymbol{\beta}_t = [\beta_t^1, \cdots, \beta_t^N] = \text{Softmax}\left[\mathbb{E}\Big[\frac{\partial V_{JT}^R(s, \boldsymbol{\tau})}{\partial \mathcal{H}(\pi_t^1(\cdot|\tau^1))}\Big], \cdots, \mathbb{E}\Big[\frac{\partial V_{JT}^R(s, \boldsymbol{\tau})}{\partial \mathcal{H}(\pi_t^N(\cdot|\tau^N))}\Big]\right]. \tag{12}$$

Note that $\sum_{i=1}^N \beta_t^i = 1$ due to the softmax operation, and the vector $\boldsymbol{\beta}_t$ captures the relative benefit of policy entropy increase for all agents. Then, to prevent the target entropy from changing abruptly, we apply exponential moving average (EMA) filtering for smoothing. The EMA filtered $\overline{\boldsymbol{\beta}}_t$ is given by

$$\overline{\boldsymbol{\beta}}_t \leftarrow (1 - \xi)\overline{\boldsymbol{\beta}}_t + \xi \boldsymbol{\beta}_t \tag{13}$$

where $\xi \in [0, 1]$. Finally, the target entropy for Agent i at time step t is given by $\mathcal{H}_i = \overline{\beta}_t^i \mathcal{H}_0$ in the case of $\mathcal{H}_0 \geq 0$, where $\overline{\beta}_t^i$ is the i-th element of $\overline{\boldsymbol{\beta}}_t$. Then, the entropy sum constraint $\sum_{i=1}^N \mathcal{H}_i = \mathcal{H}_0$ is satisfied due to $\sum_{i=1}^N \overline{\beta}_t^i = 1$.

Summarizing the above, the procedure of ADER is composed of the policy evaluation based on Proposition 1, the policy update and the temperature parameter update in (10) and (11), and the target entropy update $\mathcal{H}_i = \overline{\beta}_t^i \mathcal{H}_0$ (for $\mathcal{H}_0 > 0$) with (12) and (13). Throughout the update, the temperature parameter clipping is applied to guarantee a minimum level of exploration for every agent.

The implementation details and Pseudocode are provided in Appendix B. The source code is available at https://github.com/wjkim1202/ader.

Computation of The Partial Derivative: The final ingredient to complete our algorithm is the computation of the partial derivative w.r.t. individual policy entropy $\partial V_{JT}^R / \partial \mathcal{H}(\pi^i)$.

For this, we adopt an actor-critic structure and thus, for each agent we have a separate actor, i.e., policy, in both continuous-action and discrete-action cases. The computation of $\frac{\partial V_{JT}^R(s,\tau)}{\partial \mathcal{H}(\pi_t^i(\cdot|\tau^i))}$ depends on the overall structure, especially on the structure of the critic network. To facilitate the computation of $\frac{\partial V_{JT}^R(s,\tau)}{\partial \mathcal{H}(\pi_t^i(\cdot|\tau^i))}$, we further employ *value decomposition with a mixing network* (Rashid et al., 2018) to represent each of the disentangled joint action-value and value functions as a mixture of individual value functions. For example, the joint value function for pure return $V_{JT}^R(s, \boldsymbol{\tau})$ is decomposed as

$$V_{JT}^R(s, \boldsymbol{\tau}) = f_{mix}^{V,R}(s, V_1^R(\tau^1), \cdots, V_N^R(\tau^N)), \tag{14}$$

where $V_i^R(\tau^i)$ is the individual local value function of Agent i, and $f_{mix}^{V,R}$ is the mixing network generating the joint value V_{JT}^R from the individual local values $V_1^R, \cdots, V_N^R$. Similarly, we apply value decomposition and mixing network to $V_{JT}^{H,i}(s,\tau)$, $Q_{JT}^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ and $Q_{JT}^{H,i}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$, $\forall i \in \mathcal{N}$. The overall structures for continuous-action and discrete-action cases are shown in Appendix B.

The Continuous-Action Case: In this case, we adopt a Gaussian policy for each agent. Then, the policy neural network of Agent i with trainable parameter θ^i takes current trajectory τ^i as input and generates the mean μ^i and the log variance $\log \sigma^i$ as output. Based on these outputs and the reparameterization trick, the action of Agent i is generated as $a^i = \mu^i + \exp(\log \sigma^i) Z^i$, where $Z^i \sim N(0, I)$. The action a^i and trajectory τ^i are applied as input to the return critic network $Q_j^R(\tau^i, a^i)$ of Agent i, which generates the local Q-value $Q_i^R(\tau^i, a^i)$ due to our imposed value factorization structure. Then, all local Q-values $Q_1^R(\tau^1, a^1), \cdots, Q_N^R(\tau^N, a^N)$ from all agents are applied as input to the mixing network for global return value Q_{JT}^R. Due to the connected tensor structure of the overall network, at the time of learning, the gradient of Q_{JT}^R w.r.t. Gaussian policy entropy $\log \sigma^i$ can be computed by deep learning libraries such as Pytorch. V_{JT}^R can be obtained by sampling multiple a^i's from the same policy π_t^i, computing the corresponding multiple Q-values and taking the average over the multiple a^i samples. The gradient of V_{JT}^R can be obtained from the gradients of the corresponding multiple Q_{JT}^R.

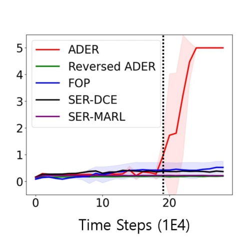
The Discrete-Action Case: In the discrete-action case, the critic network typically uses the DQN structure (Mnih et al., 2015a), which takes the trajectory τ^i as input and generates all $Q_i^R(\tau^i, a_1^i), \cdots, Q_i^R(\tau^i, a_{|\mathcal{A}|}^i)$ as output. In the discrete-action case, action is over a finite action set $\mathcal{A} = \{a_1, \cdots, a_{|\mathcal{A}|}\}$, and the policy π^i is described by a categorical distribution $\mathbf{p}^i = \left[p_1^i, \cdots, p_{|\mathcal{A}|}^i\right]$ over $\mathcal{A}$ for each trajectory. Hence, our actor, i.e, policy π^i for Agent i is a deep neural network which takes the observation τ^i as input and generates probability vector $\mathbf{p}^i = \left[p_1^i, \cdots, p_{|\mathcal{A}|}^i\right]$ as output. Let us denote the policy deep neural network parameter by θ^i and denote the policy π^i by $\pi_{\theta^i}^i$, showing the current parameter explicitly. Using the output $\mathbf{p}^i = \left[p_1^i, \cdots, p_{|\mathcal{A}|}^i\right]$ of the policy network and the output $Q_i^R(\tau^i, a_1^i), \cdots, Q_i^R(\tau^i, a_{|\mathcal{A}|}^i)$ of the critic network, we compute the local return value as $V_i^R(\tau^i) = \sum_{j=1}^{|\mathcal{A}|} p_j^i(\tau^i) Q_i^R(\tau^i, a_j^i)$. Then, all local return values $V_1^R(\tau^1), \cdots, V_N^R(\tau^N)$ are fed to the mixing network to yield the global return value V_{JT}^R as in (14).

In this discrete-action case, the policy entropy is given by $\mathcal{H}(\pi_{\theta^i}^i(\cdot|\tau^i)) = -\sum_{j=1}^N p_j^i \log p_j^i$. On the contrary to the continuous-action case in which the policy entropy $\log \sigma^i$ is an explicit node value in the overall architecture and hence the output V_{JT}^R gradient w.r.t. the node $\log \sigma^i$ is directly available, in the discrete-action case there is no node in the learing architecture corresponding to the value $\mathcal{H}(\pi_{\theta^i}^i(\cdot|\tau^i)) = -\sum_{j=1}^N p_j^i \log p_j^i$. Hence, the gradient $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ is not readily available from the architecture. Note that we only have nodes for $p_1^i, \cdots, p_{|\mathcal{A}|}^i$ in the architecture, but the gradient of V_{JT}^R w.r.t. p_j^i is not $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$. Furthermore, it is not easy to compute $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ from $\frac{\partial V_{JT}^R}{\partial p_j^i}$, $j = 1, \cdots, |\mathcal{A}|$ with $\sum_j p_j^i = 1$ for general cardinality $|\mathcal{A}|$. To circumvent this difficulty and compute the metric $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$, we exploit the policy network parameter θ^i and numerical computation. For the current policy network parameter θ^i, we have the corresponding policy network output $p_1^i, \cdots, p_{|\mathcal{A}|}^i$. Now, we consider the scalar objective function $\mathcal{H}(\pi_{\theta^i}^i)$ for the policy network, and compute the gradient of $\mathcal{H}(\pi_{\theta^i}^i)$ w.r.t. the policy parameter θ^i. Let us denote this gradient by $\frac{\partial \mathcal{H}(\pi_{\theta^i}^i)}{\partial \theta^i}$, which is the direction of θ^i for maximum policy entropy increase. With this gradient, we update the policy parameter as $\tilde{\theta}^i = \theta^i + \delta \frac{\partial \mathcal{H}(\pi_{\theta^i}^i)}{\partial \theta^i}$, where δ is a positive stepsize. Then, for the updated policy $\pi_{\tilde{\theta}^i}^i$, we compute the corresponding $p_1^i, \cdots, p_{|\mathcal{A}|}^i$ and the corresponding entropy. Using the local Q-values $Q_i^R(\tau_i, a_j^i)$ and the updated probability values, we compute the updated local value V_i^R. Using the values before and after the update, we compute $\frac{\Delta V_i^R(\tau^i)}{\Delta \mathcal{H}(\pi^i)} = \frac{V_i^R(\tau^i; \pi_{\tilde{\theta}^i}^i) - V_i^R(\tau^i; \pi_{\theta^i}^i)}{\mathcal{H}(\pi_{\tilde{\theta}^i}^i) - \mathcal{H}(\pi_{\theta^i}^i)}$. Finally, the metric $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ can be computed based on the chain rule exploiting the imposed value factorization structure. That is, $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)} = \frac{\partial V_{JT}^R(s, \boldsymbol{\tau})}{\partial V_i^R(\tau^i)} \cdot \frac{\partial V_i^R(\tau^i)}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$, where the first term $\frac{\partial V_{JT}^R(s, \boldsymbol{\tau})}{\partial V_i^R(\tau^i)}$ is available from deep learning libraries since V_{JT}^R and V_i^R are nodes of the learning architecture. The second term $\frac{\partial V_i^R(\tau^i)}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ can be approximated by $\frac{\Delta V_i^R(\tau^i)}{\Delta \mathcal{H}(\pi^i)}$ above. Note that the policy update $\tilde{\theta}^i = \theta^i + \delta \frac{\partial \mathcal{H}(\pi_{\theta^i}^i)}{\partial \theta^i}$ is only for computation of the metric, and is not done for actual learning update. Note that the role of value decomposition in addition to disentanglement is critical to compute the desired metric, i.e., credit of differential entropy increase of Agent i to the global return. This is another meaningful use case of value decomposition.

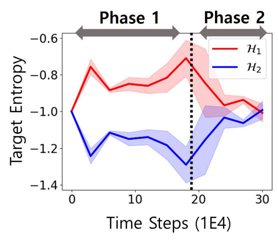
4. Experiments

Here, we provide numerical results and ablation studies.

Continuous Cooperative Matrix Game As mentioned in Sec.1, the goal of this environment is to learn two actions



(a) Averaged test return



(b) Target entropy

Figure 2. (a) The performance of ADER and the baselines on the considered matrix game and (b) The learned target entropy values during the training.

a_1 and a_2 so that the position (a_1, a_2) starting from $(0, 0)$ to reach the target circle along a narrow path, as shown in Fig. 1. The maximum reward 5 is obtained if the position reaches the center of the circle. We compare ADER with four baselines. One is SER-MARL with the same target entropy for all agents. The second is SER-MARL with different-but-constant target entropy values for two agents (SER-DCE). Here, we set a higher target entropy for a_1 than a_2. The third is Reversed ADER, which reversely uses the proposed metric $-\partial V_{JT}^{R}/\partial \mathcal{H}(\pi_t^i)$ for the level of required exploration. The fourth is FOP, which is an entropy-regularized MARL algorithm.

Fig. 2(a) shows the performance of ADER and the baselines averaged over 5 random seeds. It is seen that the considered baselines fail to learn to reach the target circle, whereas ADER successfully learns to reach the circle. Here, the different-but-constant target entropy values of SER-DCE are fixed as $(\mathcal{H}_1, \mathcal{H}_2) = (-0.7, -1.3)$, which are the maximum entropy values in ADER. It is observed that SER-DCE performs slightly better than SER-MARL but cannot learn the task with time-varying multi-agent exploration-exploitation trade-off. Fig. 2(b) shows the target entropy values $\mathcal{H}_1$ and $\mathcal{H}_2$ for a_1 and a_2, respectively, which are learned with the proposed metric during training, and shows how ADER learns to reach the target circle based on adaptive exploration. The black dotted lines in Figs. 2(a) and (b) denote the time when the position reaches the junction of the two subpaths. Before the dotted line (phase 1), ADER learns so that the target entropy of a_1 increases whereas the target entropy of a_2 decreases. So, Agent 1 and Agent 2 are trained so as to focus on exploration and exploitation, respectively. After the black dotted line (phase 2), the learning behaviors of target entropy values of a_1 and a_2 are reversed so that Agent 1 now does exploitation and Agent 2 does exploration. That is, the trade-off of exploitation and exploration is changed across the two agents. In the considered game, ADER successfully learns the time-varying trade-off of multi-agent exploration-exploitation by learning appropriate target entropies for the two agents.

Starcraft II We evaluated ADER on the StarcraftII micromanagement benchmark (SMAC) environment (Samvelyan et al., 2019). To make the problem more difficult, we modified the SMAC environment to be sparse. The considered sparse reward setting consisted of a dead reward and time-penalty reward. The dead reward was given only when an ally or an enemy died. Unlike the original reward in SMAC which gives the hit-point damage dealt as a reward, multiple agents did not receive a reward for damaging the enemy immediately in our sparse reward setting. We compared ADER with six state-of-the-art baselines: DOP (Wang et al., 2020), FACMAC (Peng et al., 2021), FOP (Zhang et al., 2021), LICA (Zhou et al., 2020), QMIX (Rashid et al., 2018), VDAC(Su et al., 2021) and MAPPO (Yu et al., 2021). For evaluation, we conducted experiments on the different SMAC maps with 5 different random seeds. Fig. 3 shows the performance of ADER and the considered seven baselines on the modified SMAC environment. It is seen that ADER significantly outperforms other baselines in terms of training speed and final performance. Especially in the hard tasks with imbalance between allies and enemies such as *MMM2*, and *8m vs 9m*, it is difficult to obtain a reward due to the simultaneous exploration of multiple agents. Thus, consideration of multi-agent exploration-exploitation trade-off is required to solve the task, and it seems that ADER effectively achieves this goal.

Continuous Action Tasks We evaluated ADER on two complex continuous action tasks: multi-agent HalfCheetah (Peng et al., 2021) and heterogeneous predator-prey (H-PP). The multi-agent HalfCheetah divides the body into disjoint sub-graphs and each sub-graph corresponds to an agent. We used 6×1-HalfCheetah, which consists of six agents with one action dimension. Next, the H-PP consists of three agents, where the maximum speeds of an agent and other agents are different. In both environments, each agent has a different role to achieve the common goal and thus the multi-agent exploration-exploitation trade-off should be considered. Here, we used two baselines: SER-MARL and FACMAC (Peng et al., 2021). In Fig. 4, showing the performance of ADER and the baselines averaged over 9 random seeds, ADER outperforms the considered baselines.

Additional experiment results on the original SMAC tasks, SMAC v2 (Ellis et al., 2022), and Google Research Football (GRF) task (Kurach et al., 2020) are provided in Appendix D, also showing significant improvement by our algorithm.

Ablation Study We conducted an ablation study on the key factors of ADER in the SMAC environment. First, we compared ADER with SER-MARL. As in the continuous action tasks, Fig. 5 shows that ADER outperforms SER-MARL. From the result, it is seen that consideration of the multi-agent exploration-exploitation trade-off yields better performance. Second, we compared ADER with and with-

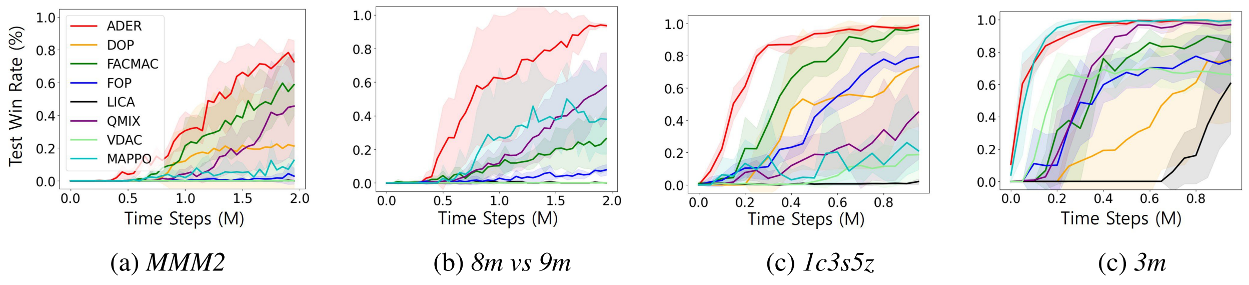


Figure 3. Average test win rate on the SMAC maps. More results are provided in Appendix D.

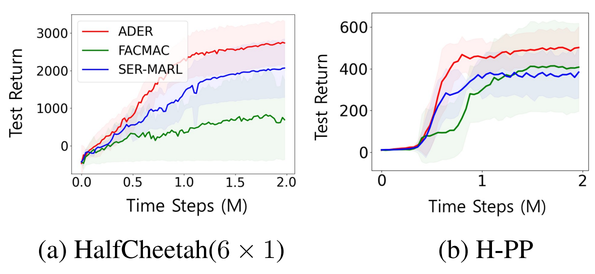


Figure 4. Performance on multi-agent HalfCheetah and H-PP

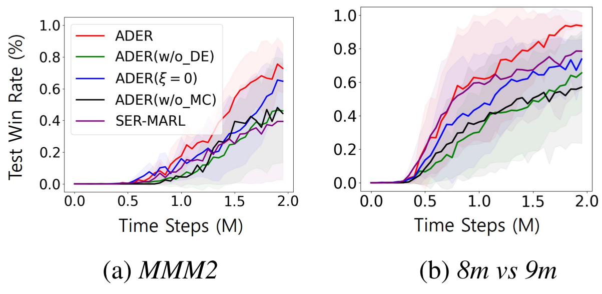


Figure 5. Ablation study: Disentangled exploration (DE), EMA filter ($\xi = 0$), SER-MARL (fixed target entropy) and the monotonic constraint (MC)

out the EMA filter. As seen in Fig. 5, it seems that the EMA filter enhances the stability of ADER. Third, we conducted an experiment to access the effectiveness of disentangling exploration and exploitation. We implemented ADER based on one critic which estimates the sum of return and entropy. As seen in Fig. 5, using two types of value functions yields better performance. Lastly, we compare ADER with and without the monotonic constraint on the value decomposition to see the necessity of the monotonic constraint. It is seen that enforcing the constraint improves performance.

The training details for all considered environments and further ablation studies are provided in Appendix C and D, respectively.

5. Conclusion and Dicussion

We have proposed the ADER framework for MARL to handle multi-agent exploration-exploitation trade-off. The proposed method is based on entropy regularization with learning proper target entropy values across agents over time by using a newly-proposed metric to measure the relative benefit of more exploration for each agent. Numerical results on various tasks show that ADER can handle time-varying multi-agent exploration-exploitation trade-off effectively and outperforms other state-of-the-art baselines. Furthermore, we expect the key ideas of ADER can be applied to other problems such as intrinsic reward design for MARL. In particular, our adaptive entropy control method can be applied to many conventionally ”single-agent” RL tasks with multiple action dimensions by viewing each action dimension as one agent and controling dimension-wise target entropy under the total entropy constraint on the overall multi-dimensional action. Such application can be helpful when each action dimension of single agent can significantly affect other action dimensions. Up to now, we assumed fixed $\mathcal{H}_0$, which is the main hyperparameter of our method. Learning of $\mathcal{H}_0$ remains as a future research work.

Acknowledgements

This work was supported in part by Institute of Information & communications Technology Planing & Evaluation (IITP) Grant funded by the Korea government (MSIT) (No.2022-0-00124, Development of Artificial Intelligence Technology for Self-Improving Competency-Aware Learning Capabilities) and in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea government (MSIT) (NRF-2021R1A2C2009143). We thank the anonymous reviewers for their valuable comments and discussion to improve the paper. A preliminary version of this work was presented in Decision Awareness in Reinforcement Learning (DARL) Workshop at ICML 2022.

References

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations*, 2019.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Beyer, L., Vincent, D., Teboul, O., Gelly, S., Geist, M., and Pietquin, O. Mulex: Disentangling exploitation from exploration in deep rl. *arXiv preprint arXiv:1907.00868*, 2019.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *International Conference on Learning Representations*, 2018.

Chentanez, N., Barto, A., and Singh, S. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17, 2004.

Christodoulou, P. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Ellis, B., Moalla, S., Samvelyan, M., Sun, M., Mahajan, A., Foerster, J. N., and Whiteson, S. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning, 2022. URL https://arxiv.org/abs/2212.07489.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., et al. Noisy networks for exploration. In *International Conference on Learning Representations*, 2018.

Gupta, T., Mahajan, A., Peng, B., Böhmer, W., and Whiteson, S. Uneven: Universal value exploration for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 3930–3941. PMLR, 2021.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pp. 1352–1361. PMLR, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Han, S. and Sung, Y. A max-min entropy framework for reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Hester, T., Quinlan, M., and Stone, P. Rtmba: A real-time model-based reinforcement learning architecture for robot control. In *2012 IEEE International Conference on Robotics and Automation*, pp. 85–90. IEEE, 2012.

Kim, W., Jung, W., Cho, M., and Sung, Y. A variational approach to mutual information-based coordination for multi-agent reinforcement learning. In *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2023.

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

Liu, I.-J., Jain, U., Yeh, R. A., and Schwing, A. Cooperative exploration for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pp. 6826–6836. PMLR, 2021a.

Liu, I.-J., Jain, U., Yeh, R. A., and Schwing, A. Cooperative exploration for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pp. 6826–6836. PMLR, 2021b.

Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. Maven: Multi-agent variational exploration. *Advances in Neural Information Processing Systems*, 32, 2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015a.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015b.

Oliehoek, F. A. and Amato, C. *A concise introduction to decentralized POMDPs*. Springer, 2016.

Oliehoek, F. A., Spaan, M. T., and Vlassis, N. Optimal and approximate q-value functions for decentralized pomdps.

Journal of Artificial Intelligence Research, 32:289–353, 2008.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. Count-based exploration with neural density models. In *International conference on machine learning*, pp. 2721–2730. PMLR, 2017.

Peng, B., Rashid, T., Schroeder de Witt, C., Kamienny, P.-A., Torr, P., Böhmer, W., and Whiteson, S. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34, 2021.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. In *International Conference on Learning Representations*, 2018.

Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

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. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Su, J., Adams, S., and Beling, P. A. Value-decomposition multi-agent actor-critics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11352–11360, 2021.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Wang, T., Wang, J., Wu, Y., and Zhang, C. Influence-based multi-agent exploration. In *International Conference on Learning Representations*, 2019.

Wang, Y., Han, B., Wang, T., Dong, H., and Zhang, C. Dop: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*, 2020.

Wei, H., Zheng, G., Yao, H., and Li, Z. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2496–2505, 2018.

Wu, T., Zhou, P., Liu, K., Yuan, Y., Wang, X., Huang, H., and Wu, D. O. Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(8):8243–8256, 2020.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., and Wu, Y. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

Zhang, T., Li, Y., Wang, C., Xie, G., and Lu, Z. Fop: Factorizing optimal joint policy of maximum-entropy multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 12491–12500. PMLR, 2021.

Zheng, L., Chen, J., Wang, J., He, J., Hu, Y., Chen, Y., Fan, C., Gao, Y., and Zhang, C. Episodic multi-agent reinforcement learning with curiosity-driven exploration. *Advances in Neural Information Processing Systems*, 34: 3757–3769, 2021.

Zhou, M., Liu, Z., Sui, P., Li, Y., and Chung, Y. Y. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 33:11853–11864, 2020.

Appendix A: Proofs

Proposition .1. *The decomposed soft Bellman operators $\mathcal{T}_R^\pi$ and $\mathcal{T}_{H,i}^\pi$ are contractions.*

Proof: The action value functions $Q_{JT}^R(\boldsymbol{\tau_t}, \boldsymbol{a_t})$ and $Q_{JT}^{H,i}(\boldsymbol{\tau_t}, \boldsymbol{a_t})$ can be estimated based on their corresponding Bellman backup operators, defined by

$$\begin{aligned} \mathcal{T}_R^\pi Q_{JT}^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) &:= r_t + \gamma \mathbb{E}\left[V_{JT}^R(s_{t+1}, \boldsymbol{\tau_{t+1}})\right], \text{ where} \qquad (15)\\ V_{JT}^R(s_t, \boldsymbol{\tau_t}) &= \mathbb{E}\left[Q_{JT}^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\right] \\ \mathcal{T}_{H,i}^\pi Q_{JT}^{H,i}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) &:= \gamma \mathbb{E}\left[V_{JT}^{H,i}(s_{t+1}, \boldsymbol{\tau_{t+1}})\right], \text{ where} \qquad (16)\\ V_{JT}^{H,i}(s_t, \boldsymbol{\tau_t}) &= \mathbb{E}\left[Q_{JT}^{H,i}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) - \alpha^i \log \pi(a_t^i|\tau_t^i)\right]. \end{aligned}$$

Here, $V_{JT}^R(s_t, \boldsymbol{\tau_t})$ and $V_{JT}^{H,i}(s_t, \boldsymbol{\tau_t})$ are the joint value functions regarding reward and entropy, respectively.

First, let us consider the decomposed Bellman operator regarding reward, $\mathcal{T}_R^\pi$. For the sake of simplicity, we abbreviate $(Q_{JT}^R, Q_{JT}^{H,i}, V_{JT}^R, V_{JT}^{H,i})$ as $(Q^R, Q^{H,i}, V^R, V^{H,i})$. From (15), we have

$$\mathcal{T}_R^\pi Q^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = r_t + \gamma \mathbb{E}_{s_{t+1}, \boldsymbol{\tau_{t+1}}, \boldsymbol{a_{t+1}}}\left[Q^R(s_{t+1}, \boldsymbol{\tau_{t+1}}, \boldsymbol{a_{t+1}})\right]. \qquad (17)$$

Then, we have

$$\begin{aligned} &\|\mathcal{T}_R^\pi(q_t^1) - \mathcal{T}_R^\pi(q_t^2)\|_\infty \\ &= \|(r_t + \gamma \sum_{\substack{s_{t+1}, \boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi}(\boldsymbol{a_{t+1}}|\boldsymbol{\tau_{t+1}}) p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t) \cdot q_{t+1}^1) \\ &\qquad - (r_t + \gamma \sum_{\substack{s_{t+1}, \boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi}(\boldsymbol{a_{t+1}}|\boldsymbol{\tau_{t+1}}) p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t) \cdot q_{t+1}^2)\|_\infty \\ &= \|\gamma \sum_{\substack{s_{t+1}, \boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi}(\boldsymbol{a_{t+1}}|\boldsymbol{\tau_{t+1}}) p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t) \cdot (q_{t+1}^1 - q_{t+1}^2))\|_\infty \\ &\le \|\gamma \sum_{\substack{s_{t+1}, \boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi}(\boldsymbol{a_{t+1}}|\boldsymbol{\tau_{t+1}}) p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\|_\infty \|q_{t+1}^1 - q_{t+1}^2\|_\infty \\ &\le \gamma \|q_{t+1}^1 - q_{t+1}^2\|_\infty \end{aligned}$$

for $q_t^1 = \left[Q_1^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\right]_{\substack{s_t \in \mathcal{S}, \boldsymbol{a_t} \in \mathcal{A} \\ \boldsymbol{\tau_t} \in (\Omega \times \mathcal{A})^*}}$ and $q_t^2 = \left[Q_2^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\right]_{\substack{s_t \in \mathcal{S}, \boldsymbol{a_t} \in \mathcal{A} \\ \boldsymbol{\tau_t} \in (\Omega \times \mathcal{A})^*}}$ since $\|\sum_{\substack{s_{t+1}, \boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi}(\boldsymbol{a_{t+1}}|\boldsymbol{\tau_{t+1}}) p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\|_\infty \le 1$. Thus, the operator $\mathcal{T}_R^\pi$ is a γ-contraction.

Next, let us consider the decomposed Bellman operator regarding entropy, $\mathcal{T}_{H,i}^\pi$. From (16), we have

$$\mathcal{T}_{H,i}^\pi Q^{H,i}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = \gamma \mathbb{E}\left[Q^{H,i}(s_{t+1}, \boldsymbol{\tau_{t+1}}, \boldsymbol{a_{t+1}}) - \alpha^i \log \pi(a_{t+1}^i|\tau_{t+1}^i))\right]. \qquad (18)$$

Then, we have

$$
\begin{aligned}
&\|\mathcal{T}^{\pi}_{H,i}(q^1_t) - \mathcal{T}^{\pi}_{H,i}(q^2_t)\|_\infty \\
&= \|(\gamma \sum_{\substack{s_{t+1},\boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi(a_{t+1}|\tau_{t+1})} p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t) \cdot (q^1_{t+1} - \alpha^i \log \pi(a^i_{t+1}|\tau^i_{t+1})) \\
&\qquad - (\gamma \sum_{\substack{s_{t+1},\boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi(a_{t+1}|\tau_{t+1})} p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t) \cdot (q^2_{t+1} - \alpha^i \log \pi(a^i_{t+1}|\tau^i_{t+1}))\|_\infty \\
&= \|\gamma \sum_{\substack{s_{t+1},\boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi(a_{t+1}|\tau_{t+1})} p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t) \cdot (q^1_{t+1} - q^2_{t+1}))\|_\infty \\
&\le \|\gamma \sum_{\substack{s_{t+1},\boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi(a_{t+1}|\tau_{t+1})} p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t)\|_\infty \|q^1_{t+1} - q^2_{t+1}\|_\infty \\
&\le \gamma \|q^1_{t+1} - q^2_{t+1}\|_\infty
\end{aligned}
$$

for $q^1_t = \Big[Q^R_1(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\Big]_{\substack{s_t \in \mathcal{S}, \boldsymbol{a_t} \in \mathcal{A} \\ \boldsymbol{\tau_t} \in (\Omega \times \mathcal{A})^*}}$ and $q^2_t = \Big[Q^R_2(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\Big]_{\substack{s_t \in \mathcal{S}, \boldsymbol{a_t} \in \mathcal{A} \\ \boldsymbol{\tau_t} \in (\Omega \times \mathcal{A})^*}}$ since $\|\sum_{\substack{s_{t+1},\boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi(a_{t+1}|\tau_{t+1})} p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t)\|_\infty \le 1$. Thus, the operator $\mathcal{T}^{\pi}_{H,i}$ is a γ-contraction.

Appendix B: Detailed Implementation

Here, we describe the implementation of ADER for discrete action tasks based on SAC-discrete (Christodoulou, 2019). The learning process consists of the update of both temperature parameters and target entropies and the approximation of multi-agent maximum entropy solution, which consists of the update of the joint policy and the critics. To do this, we first approximate the policies $\{\pi^i_{\phi_i}\}_{i=1}^N$, the joint action value functions Q^R_{JT,θ_R} and $Q^{H,i}_{JT,\theta_{H,i}}$ by using deep neural networks with parameters, $\{\phi_i\}_{i=1}^N$, θ_R and $\{\theta_{H,i}\}_{i=1}^N$.

First, the joint policy is updated based on Eq. (12) and the loss function is given by

$$L(\boldsymbol{\phi}) = \mathbb{E}_{(s_t,\boldsymbol{\tau_t})\sim\mathcal{D},\{a^i_t\sim\pi^i(\cdot|\tau^i_t)\}_{i=1}^N}\Bigg[\sum_{i=1}^N \alpha^i(\log\pi^i_{\phi_i}(a^i_t|\tau^i_t) - Q^{H,i}_{JT,\theta_{H,i}}(s_t,\boldsymbol{\tau}_t,\boldsymbol{a}_t)) - Q^R_{JT,\theta_R}(s_t,\boldsymbol{\tau}_t,\boldsymbol{a}_t)\Bigg], \tag{B.1}$$

where $\boldsymbol{\phi} = \{\phi_i\}_{i=1}^N$ is the parameter for the joint policy. Next, the joint action value functions are trained based on the disentangled Bellman operators defined in Eq. (10) and the loss functions are given by

$$L(\theta_R) = \mathbb{E}_{(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t},s_{t+1},\boldsymbol{\tau_{t+1}})\sim\mathcal{D}}\left[\frac{1}{2}(Q^R_{JT,\theta_R}(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t}) - (r_t + \gamma V^R_{JT,\bar\theta_R}(s_{t+1},\boldsymbol{\tau_{t+1}})))^2\right] \tag{B.2}$$

$$L(\theta_{H,i}) = \mathbb{E}_{(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t},s_{t+1},\boldsymbol{\tau_{t+1}})\sim\mathcal{D}}\left[\frac{1}{2}(Q^{H,i}_{JT,\theta_i}(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t}) - \gamma V^{H,i}_{JT,\bar\theta_{H,i}}(s_{t+1},\boldsymbol{\tau_{t+1}})))^2\right] \tag{B.3}$$

where $V^R_{JT,\bar\theta_R}$ and $V^{H,i}_{JT,\bar\theta_{H,i}}$ are defined as follows:

$$V^R_{JT,\bar\theta_R}(s_t,\boldsymbol{\tau_t}) = \mathbb{E}\left[Q^R_{JT,\bar\theta_R}(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t})\right] \tag{B.4}$$

$$V^{H,i}_{JT,\bar\theta_{H,i}}(s_t,\boldsymbol{\tau_t}) = \mathbb{E}\left[Q^{H,i}_{JT,\bar\theta_{H,i}}(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t}) - \alpha^i\log\pi(a^i_t|\tau^i_t)\right]. \tag{B.5}$$

Note that $\bar\theta_R$ and $\bar\theta_{H,i}$ are obtained based on the EMA of the parameters of the joint action-value functions.

Although the definitions of the state value functions are given by (B.4) and (B.5), we do not use this definition to compute the state value functions. This is because the marginalization over joint action becomes complex as the number of agents increases. For the practical computation of V^R_{JT} and $V^{H,i}_{JT}$, we do the following for reduced complexity. We first marginalize the individual Q-function based on individual action to get V^R_i. Then, we feed $V^R_1, \cdots, V^R_N$ of all agents to the mixing network $f^{V,R}_{mix}$ to obtain the joint state value function as $V^R_{JT}(s,\boldsymbol{\tau}) = f^{V,R}_{mix}(s, V^R_1(\tau^1),\cdots,V^R_N(\tau^N))$. Here, $f^{V,R}_{mix}$ is learned such that $f^{V,R}_{mix}$ follows the definition by the TD loss eq. (B.2) and the Bellman equation. In addition, we share the mixing network for $Q^{H,i}_{JT}$ for all $i \in \mathcal{N}$ and inject the one-hot vector which denotes the agent index i to handle the scalability.

We update the temperature parameters based on Eq. (13) and the loss function is given by

$$L(\alpha^i) = \mathbb{E}_{\boldsymbol{\tau_t}\sim\mathcal{D},\{a^i_t\sim\pi^i(\cdot|\tau^i_t)\}_{i=1}^N}\left[-\alpha^i\log\boldsymbol{\pi}_t(a^i_t|\tau^i_t) - \alpha^i\mathcal{H}_i\right], \quad \forall i\in\mathcal{N}. \tag{B.6}$$

Finally, we update the target entropy of each agent. For $\mathcal{H}_0 \geq 0$, we set the coefficients β_i for determining the individual target entropy $\mathcal{H}_i$ as $\boldsymbol{\beta} = \left[\beta_1,\cdots,\beta_i,\cdots,\beta_N\right] =$

$$\text{Softmax}\left[\mathbb{E}\Big[\frac{\partial V^R_{JT}(s,\boldsymbol{\tau})}{\partial\mathcal{H}(\pi^1_t(\cdot|\tau^1))}\Big],\cdots,\mathbb{E}\Big[\frac{\partial V^R_{JT}(s,\boldsymbol{\tau})}{\partial\mathcal{H}(\pi^i_t(\cdot|\tau^i))}\Big],\cdots,\mathbb{E}\Big[\frac{\partial V^R_{JT}(s,\boldsymbol{\tau})}{\partial\mathcal{H}(\pi^N_t(\cdot|\tau^N))}\Big]\right], \tag{B.7}$$

where the computation of $\frac{\partial V^R_{JT}(s,\tau)}{\partial\mathcal{H}(\pi^i_t(\cdot|\tau^i))}$ is explained in Sec. 5.

Note that we change the sign of the elements in Eq. (B.7) if $\mathcal{H}_0 < 0$ to satisfy the core idea of ADER, which assigns a high target entropy to the agent whose benefit to the joint value is small.

In addition, before the softmax layer, we normalize the elements in Eq. (B.7). Based on the coefficients, the target entropy is given by $\mathcal{H}_i = \beta_i^{EMA} \times \mathcal{H}_0$ where β_i^{EMA} is computed recursively as

$$\boldsymbol{\beta}^{EMA} \leftarrow (1-\xi)\boldsymbol{\beta}^{EMA} + \xi\boldsymbol{\beta} \tag{B.8}$$

B1. Computation of the metric $\frac{\partial V_{JT}^R(s,\tau)}{\partial \mathcal{H}(\pi_t^i(\cdot|\tau^i))}$

We adopted an actor-critic structure for our algorithm. Hence, for each agent we have a separate actor, i.e., policy in both continuous-action and discrete-action cases, as seen in Figures 6 and 7, which show the overall structure for continuous-action and discrete-action cases, respectively. The computation of the partial derivative $\frac{\partial V_{JT}^R(s,\tau)}{\partial \mathcal{H}(\pi_t^i(\cdot|\tau^i))}$ in Eq. (B.7) depends on the overall structure, especially on the structure of the individual critic network.

First, consider the continuous-action case. In this case, we used a Gaussian policy for each agent. Then, the policy neural network of Agent i with trainable parameter θ^i takes trajectory τ_t^i as input and generates the mean μ^i and the log variance $\log\sigma^i$ as output, as shown in Figure 6. Based on these outputs and the reparameterization trick, the action of Agent i is generated as $a^i = \mu^i + \exp(\log\sigma^i)Z^i$, where Z^i is Gaussian-distributed with zero mean and identity covariance matrix, i.e., $Z^i \sim N(0,I)$. The action a^i and trajectory τ^i are applied as input to both return and entropy critic networks for Agent i, as seen in Figure 6. Now, focus on the return critic network of Agent i, which is relevant to the computation of our metric. The return critic of Agent i generates the local Q-value $Q_i^R(\tau^i, a^i)$. All local Q-values $Q_1^R(\tau^1,a^1),\cdots,Q_N^R(\tau^N,a^N)$ from all agents are applied as input to the mixing network for global return value Q_{JT}^R, as seen in Figure 6. Due to the connected tensor structure in Figure 6, at the time of learning, the gradient of Q_{JT}^R with respect to $\log\sigma^i$ can be computed by deep learning libraries such as Pytorch. Note that $\log\sigma^i$ is simply a scaled version of the Gaussian policy entropy. So, we can just obtain this value $\partial Q_{JT}^R/\partial\log\sigma^i$ from deep learning libraries. Furthermore, V_{JT}^R can be obtained by sampling multiple a^i's from the same policy π_t^i, computing the corresponding multiple Q-values and taking the average over the multiple a^i samples. However, we simplify this step and just use $\partial Q_{JT}^R/\partial\log\sigma^i$ as our estimate for the metric $\frac{\partial V_{JT}^R(s,\tau)}{\partial \mathcal{H}(\pi_t^i(\cdot|\tau^i))}$. Indeed, many algorithms use single-sample average for obtaining expectations for algorithm simplicity.

Second, consider the discrete-action case. In this case, we again use an actor-critic structure for our algorithm. The structure of the critic network of Agent i in the discrete-action case is different from that in the continuous-action case. Whereas the critic network takes the trajectory τ^i and the action a^i as input, and generates $Q_i^R(\tau^i,a^i)$ in the continuous-action case, the critic network typically uses the DQN structure (Mnih et al. 2015), which takes the trajectory τ^i as input and generates all $Q_i^R(\tau^i,a_1^i),\cdots,Q_i^R(\tau^i,a_{|\mathcal{A}|}^i)$ as output in the discrete-action case. In the discrete-action case, action is over a finite action set $\mathcal{A}=\{a_1,\cdots,a_{|\mathcal{A}|}\}$, and the policy is described by a categorical distribution $\mathbf{p}^i = \left[p_1^i,\cdots,p_{|\mathcal{A}|}^i\right]$ over $\mathcal{A}$ for each state (or trajectory). Hence, our actor, i.e, policy π^i for Agent i is a deep neural network which takes the observation τ^i as input and generates probability vector $\mathbf{p}^i = \left[p_1^i,\cdots,p_{|\mathcal{A}|}^i\right]$ as output. Here, let us denote the policy deep neural network parameter by θ^i and denote the policy π_t^i by $\pi_{\theta^i}^i$, showing the current parameter explicitly. Then, using the output $\mathbf{p}^i = \left[p_1^i,\cdots,p_{|\mathcal{A}|}^i\right]$ of the policy network and the output $Q_i^R(\tau^i,a_1^i),\cdots,Q_i^R(\tau^i,a_{|\mathcal{A}|}^i)$ of the critic network, we compute the local return value as

$$V_i^R(\tau^i) = \sum_{j=1}^{|\mathcal{A}|} p_j^i(\tau^i) Q_i^R(\tau^i, a_j^i). \tag{19}$$

Then, all local return values $V_1^R(\tau^1),\cdots,V_N^R(\tau^N)$ are fed to the mixing network for global return value V_{JT}^R, as seen in Figure 7.

In this discrete-action case, the policy entropy is given by $\mathcal{H}(\pi_{\theta^i}^i(\cdot|\tau^i)) = -\sum_{j=1}^N p_j^i \log p_j^i$. On the contrary to the continuous-action case in which the policy entropy $\log\sigma^i$ is an explicit node value in the overall structure and hence the output V_{JT}^R gradient with respect to the node $\log\sigma^i$ is directly available, in the discrete-action case there is no node corresponding to the value $\mathcal{H}(\pi_{\theta^i}^i(\cdot|\tau^i)) = -\sum_{j=1}^N p_j^i \log p_j^i$. Hence, the gradient $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ is not readily available from the architecture. Note that we only have nodes for $p_1^i,\cdots,p_{|\mathcal{A}|}^i$ in the architecture, but the gradient of V_{JT}^R with respect to p_j^i is not $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$. Furthermore, it is not easy to compute $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ from $\frac{\partial V_{JT}^R}{\partial p_j^i}$, $j=1,\cdots,|\mathcal{A}|$ with $\sum_j p_j^i = 1$ for general cardinality $|\mathcal{A}|$.

To circumvent this difficulty and compute the metric $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$, we exploit the policy network parameter θ^i and numerical computation. When the current policy network parameter is θ^i, we have the corresponding policy network output $p_1^i, \cdots, p_{|\mathcal{A}|}^i$. Then, consider the temporary scalar objective function $\mathcal{H}(\pi_{\theta^i}^i)$ for the policy network. We can compute the gradient of $\mathcal{H}(\pi_{\theta^i}^i)$ with respect to the policy parameter θ^i. Let us denote this gradient by $\frac{\partial \mathcal{H}(\pi_{\theta^i}^i)}{\partial \theta^i}$, which is the direction of θ^i for maximum policy entropy increase. Then, we update the policy parameter as $\tilde{\theta}^i = \theta^i + \delta \frac{\partial \mathcal{H}(\pi_{\theta^i}^i)}{\partial \theta^i}$, where δ is a positive stepsize. Then, for the updated policy $\pi_{\tilde{\theta}^i}^i$, we compute the corresponding $p_1^i, \cdots, p_{|\mathcal{A}|}^i$. Using these updated probability values, we compute the local value V_i^R by using eq. (19). Using the values before and after the update, we compute $\frac{\Delta V_i^R(\tau^i)}{\Delta \mathcal{H}(\pi^i)} = \frac{V_i^R(\tau^i;\pi_{\tilde{\theta}^i}^i) - V_i^R(\tau^i;\pi_{\theta^i}^i)}{\mathcal{H}(\pi_{\tilde{\theta}^i}^i) - \mathcal{H}(\pi_{\theta^i}^i)}$.

Now, the metric $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ can be computed based on the chain rule. That is, we have $\frac{\partial V_{JT}^R}{\partial \mathcal{H}(\pi_{\theta^i}^i)} = \frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_i^R(\tau^i)} \times \frac{\partial V_i^R(\tau^i)}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$. Here, the first term $\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_i^R(\tau^i)}$ is available from deep learning libraries since V_{JT}^R and V_i^R are nodes of the learning architecture. The second term $\frac{\partial V_i^R(\tau^i)}{\partial \mathcal{H}(\pi_{\theta^i}^i)}$ can be approximated by $\frac{\Delta V_i^R(\tau^i)}{\Delta \mathcal{H}(\pi^i)}$ in the above.

Note that the policy update $\tilde{\theta}^i = \theta^i + \delta \frac{\partial \mathcal{H}(\pi_{\theta^i}^i)}{\partial \theta^i}$ is only for computation of the metric. It is not done for the actual learning update.

B2. Overall Architecture and Algorithm Pseudocode

We summarize the proposed algorithm in Algorithm 1 and illustrate the overall architecture of the proposed ADER in Figures 6 and 7.

Algorithm 1 ADaptive **E**ntropy-**R**egularization for multi-agent reinforcement learning (ADER)

```
Initialize parameters $\{\phi_i\}_{i=1}^N$, $\theta_R$, $\{\theta_{H,i}\}_{i=1}^N$, $\bar{\theta}_R$, $\{\bar{\theta}_{H,i}\}_{i=1}^N$
Generate a trajectory $\tau$ by interacting with the environment by using the joint policy $\boldsymbol{\pi}$ and store $\tau$ in the replay memory
for $episode = 1, 2, \cdots$ do
    Generate a trajectory $\tau$ by using the joint policy $\boldsymbol{\pi}$ and store $\tau$ in the replay memory $\mathcal{D}$
    for each gradient step do
        Sample a minibatch from $\mathcal{D}$
        Update $\{\phi_i\}_{i=1}^N$ by minimizing the loss function Eq. (B.1)
        Update $\theta_R$, $\{\theta_{H,i}\}_{i=1}^N$ by minimizing the loss functions Eq. (B.2) and Eq. (B.3)
        Update $\alpha^i$ by minimizing the loss function Eq. (B.6)
        Update $\{\mathcal{H}_i\}_{i=1}^N$ by computing Eq. (B.7) and Eq. (B.8)
        Update $\bar{\theta}_R$ and $\{\bar{\theta}_{H,i}\}_{i=1}^N$ by EMA based on $\theta_R$ and $\{\theta_{H,i}\}_{i=1}^N$
    end for
end for
```

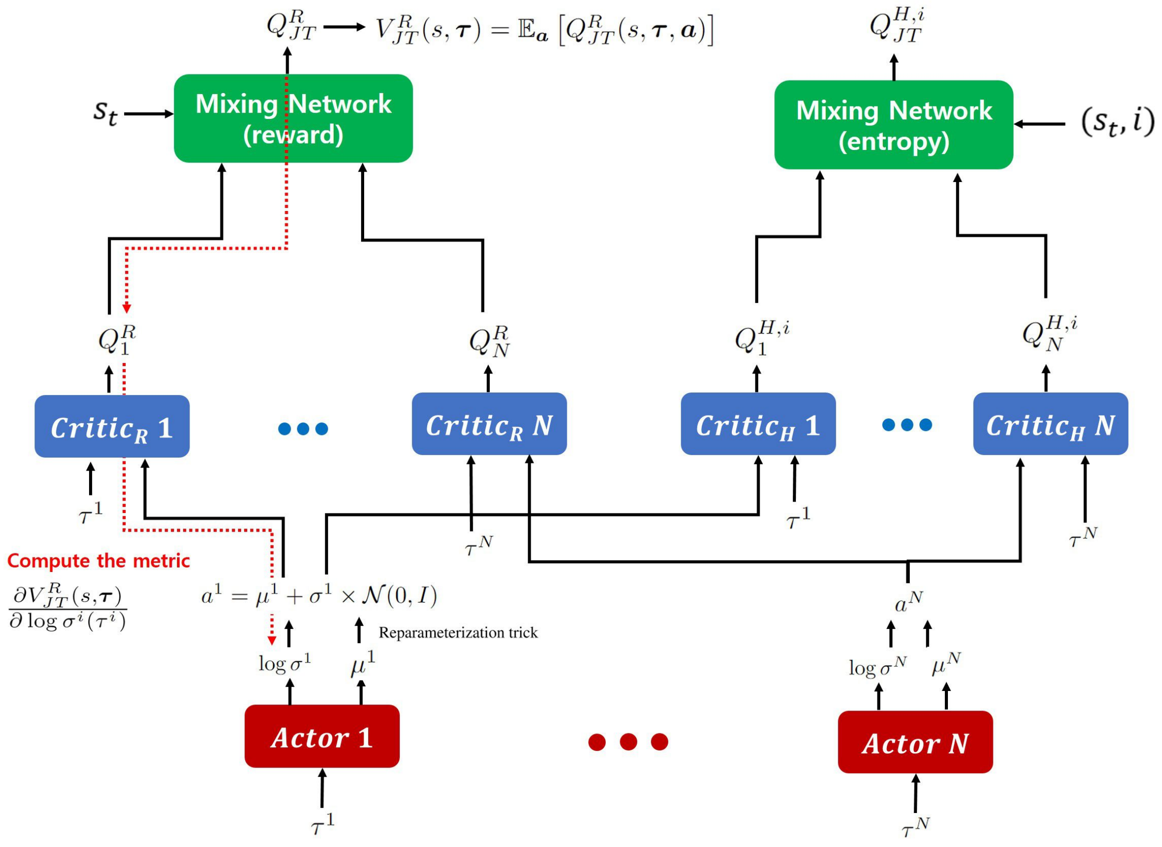


Figure 6. Overall architecture of ADER in continuous action cases

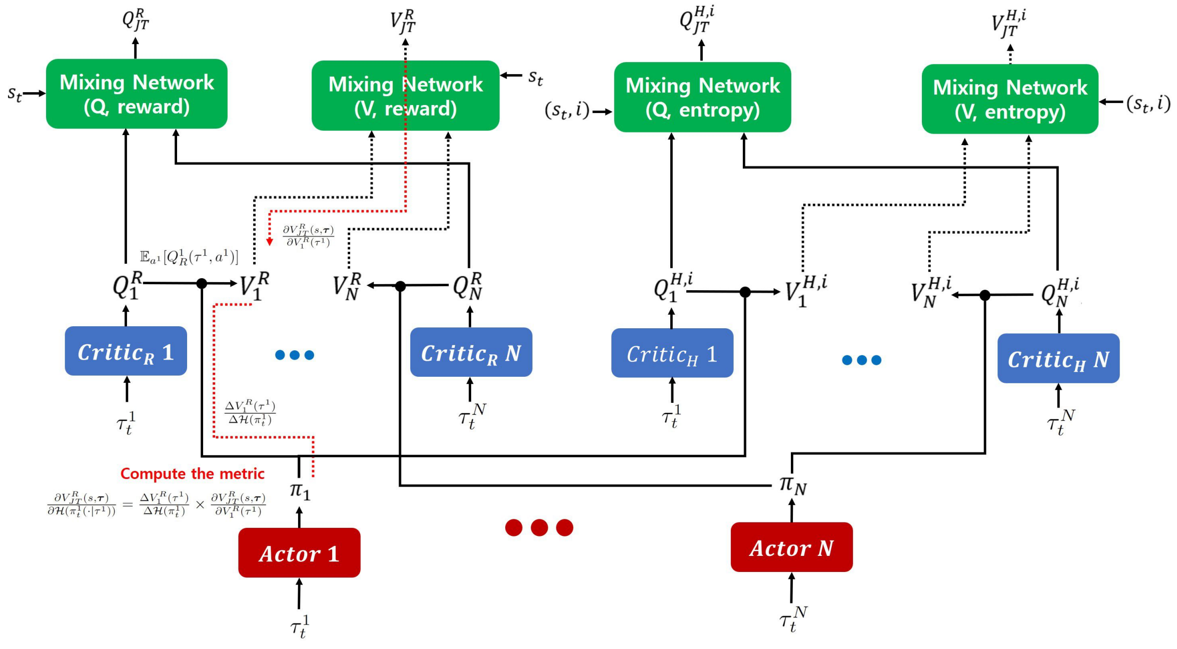

Q_{JT}^R
V_{JT}^R
$Q_{JT}^{H,i}$
$V_{JT}^{H,i}$
Mixing Network (Q, reward)
Mixing Network (V, reward)
Mixing Network (Q, entropy)
Mixing Network (V, entropy)
s_t
(s_t, i)
$\mathbb{E}_{a^1}[Q_R^1(\tau^1, a^1)]$
$\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_1^R(\tau^1)}$
Q_1^R
V_1^R
V_N^R
Q_N^R
$Q_1^{H,i}$
$V_1^{H,i}$
$V_N^{H,i}$
$Q_N^{H,i}$
$Critic_R\ 1$
$Critic_R\ N$
$Critic_H\ 1$
$Critic_H\ N$
τ_t^1
τ_t^N
$\frac{\Delta V_1^R(\tau^1)}{\Delta \mathcal{H}(\pi_t^1)}$
Compute the metric
$\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial \mathcal{H}(\pi_t^1(\cdot|\tau^1))} = \frac{\Delta V_1^R(\tau^1)}{\Delta \mathcal{H}(\pi_t^1)} \times \frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_1^R(\tau^1)}$
π_1
π_N
Actor 1
Actor N

Figure 7. Overall architecture of ADER in discrete action cases

Appendix C: Training details

We compute the joint value function as $V_{JT}^R(s,\tau) = f_{mix}^{V,R}(s, V_1^R(\tau^1), \cdots, V_N^R(\tau^N))$. To compute this, as similar in (Zhang et al., 2021), we first obtain the local value functions as $V_i^R(\tau^i) = \mathbb{E}_{a^i}[Q^R(\tau^i, a^i)]$ and then input the obtained local value functions into the mixing network. For discrete action environments, we share the mixing network for both V_{JT}^R and Q_{JT}^R, and thus the mixing network is trained to minimize the TD error of Q_{JT}^R. It works well as the reviewer can see in the experimental results. For continuous action environments, we use two mixing networks for V_{JT}^R and Q_{JT}^R which are trained separately as in SAC (Haarnoja et al., 2018a). In addition, we need N mixing networks for $Q_{JT}^{H,i}$. To handle the scalability, we share the mixing network for $Q_{JT}^{H,i}$ for all $i \in \mathcal{N}$ and inject the one-hot vector which denotes the agent index i as QMIX shares the local Q-functions with one parameterized neural network.

C1. Environment Details

Multi-agent HalfCheetah We considered the multi-agent HalfCheetah introduced in (Peng et al., 2021). The multi-agent HalfCheetah divides the body into disjoint sub-graphs and each sub-graph corresponds to an agent. We used 6×1-HalfCheetah, which consists of six agents with one action dimension. We set the maximum graph distance $k = 1$, where k denotes the distance each agent can observe. We set the maximum episode length as $T_{max} = 1000$.

Heterogeneous Predator-Prey (H-PP) We modified the continuous predator-prey environment considered in (Peng et al., 2021) to be heterogeneous. The considered heterogeneous predator-prey consists of three predator agents, where the maximum speeds of an agent ($v_{max}^1 = 1.0$) and other agents ($v_{max}^2 = 0.75$) are different, three preys with the maximum speed ($v_{max}^3 = 1.25$) is faster than all predators and the landmarks. The preys move away from the nearest predator implemented in (Peng et al., 2021) and thus the predators should be trained to pick one prey and catch the prey together. Each agent observes the relative positions of the other predators and the landmarks within view range and the relative positions and velocities of the prey within view range. The reward $+10$ is given when one of the predators collides with the prey. We set the maximum episode length as $T_{max} = 50$.

Starcraft II We evaluated ADER on the StarcraftII micromanagement benchmark (SMAC) environment (Samvelyan et al., 2019). To make the problem more difficult, we modified the SMAC environment to be sparse. The considered sparse reward setting consists of a death reward and time-penalty reward. The time-penalty reward is -0.1 and the death reward is given $+10$ and -1 when one enemy dies and one ally dies, respectively. Additionally, the dead reward is given $+200$ if all enemies die.

C2. Training Details and Hyperparameters

We implemented ADER based on (Samvelyan et al., 2019; Peng et al., 2021; Zhang et al., 2021) and conducted the experiments on a server with Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz and 8 Nvidia Titan xp GPUs. Each experiment took about 12 to 24 hours. We used the implementations of the considered baselines provided by the authors.

Multi-agent HalfCheetah In the multi-agent halfcheetah environment, the architecture of the policies and critics for ADER follows (Peng et al., 2021). We use an MLP with 2 hidden layers which have 400 and 300 hidden units and ReLU activation functions. The final layer uses tanh activation function to bound the action as in (Haarnoja et al., 2018a). We also use the reparameterization trick for the policy as in (Haarnoja et al., 2018a). The replay buffer stores up to 10^6 transitions and 100 transitions are uniformly sampled for training. As in (Haarnoja et al., 2018b), we set the sum of target entropy as

$$\mathcal{H}_0 = N \times (-dim(\mathcal{A})) = 6 \times (-1) = -6,$$

where N is the number of agents. We set the hyperparameter for EMA filter as $\xi = 0.9$ and initialize the temperature parameters as $\alpha_{init}^i = e^{-2}$ for all $i \in \mathcal{N}$.

Heterogeneous Predator-Prey In the heterogeneous predator-prey environment, the architecture of the policies and critics for ADER follows (Peng et al., 2021). To parameterize the policy, we use a deep neural network which consists of a fully-connected layer, GRU and a fully-connected layer which have 64 dimensional hidden units. The final layer uses tanh activation function to bound the action. Next, for the critic network, we use a MLP with 2 hidden layers which have 64 hidden units and ReLU activation function. The replay buffer stores up to 5000 episodes and 32 episodes are uniformly sampled for training. As in (Haarnoja et al., 2018b), we set the sum of target entropy as

$$\mathcal{H}_0 = N \times (-dim(\mathcal{A})) = 3 \times (-2) = -6.$$

We set the hyperparameter for EMA filter as $\xi = 0.9$ and initialize the temperature parameters as $\alpha^i_{init} = e^{-2}$ for all $i \in \mathcal{N}$.

Starcraft II For parameterization of the policy we use a deep neural network which consists of a fully-connected layer, GRU and a fully-connected layer which have 64 dimensional hidden units. For the critic networks we use a MLP with 2 hidden layers which have 64 hidden units and ReLU activation function. The replay buffer stores up to 5000 episodes and 32 episodes are uniformly sampled for training. For the considered maps in SMAC, we use different hyperparameters. We set the sum of target entropy based on the maximum entropy, which can be achieved if the policy is uniform distribution, as

$$\mathcal{H}_0 = N \times \mathcal{H}^* \times k_{ratio} = N \times \log(dim(\mathcal{A})) \times k_{ratio}.$$

The values of k_{ratio}, ξ, and initial temperature parameter for each map are summarized Table 1.

Table 1. Hyperparameters for the considered SMAC environment

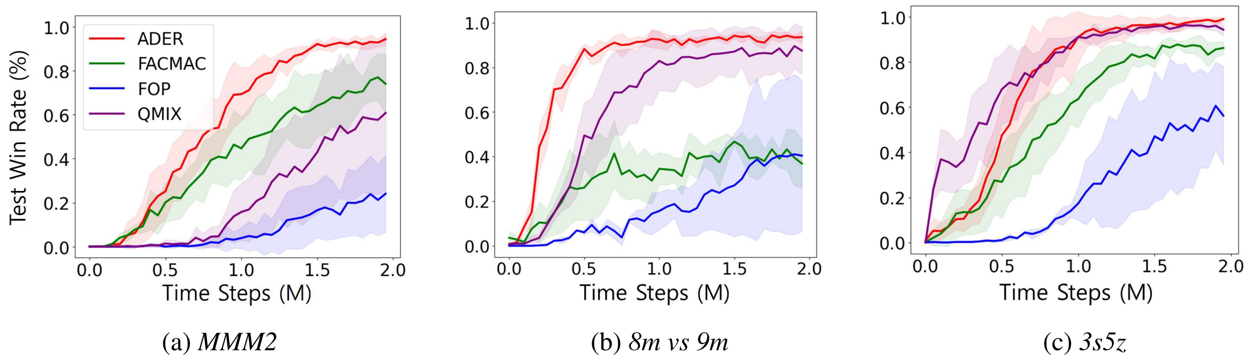
MAP	k_{ratio}	ξ	α^i_{init}
1c3s5z	0.05	0.9	e^{-3}
3m	0.1	0.9	e^{-2}
3s5z	0.05	0.9	e^{-3}
3s vs 3z	0.1	0.9	e^{-3}
MMM2	0.1	0.9	$e^{-2.5}$
8m vs 9m	0.1	0.9	e^{-3}

In all the considered environments, we apply the value factorization technique proposed in (Rashid et al., 2018). The architecture of the mixing network for ADER, which follows (Rashid et al., 2018), takes the output of individual critics as input and outputs the joint action value function. The weights of the mixing network are produced by the hypernetwork which takes the global state as input. The hypernetwork consists of a MLP with a single hidden layer and an ELU activation function. Due to the ELU activation function, the weights of the mixing network are non-negative and this achieves the monotonic constraint in (Rashid et al., 2018). We expect that ADER can use other value factorization technique to yield better performance.

Appendix D: Further experiments

D1. Experiments on the original SMAC environments

We here provide the experiments on the original SMAC environments. We compared ADER with three baselines including FACMAC (Peng et al., 2021), FOP (Zhang et al., 2021) and QMIX (Rashid et al., 2018). For all the considered maps, ADER outperforms the baselines, as shown in Fig. 8. Thus, the proposed adaptive entropy-regularization method performs well in both original and sparse SMAC environments.



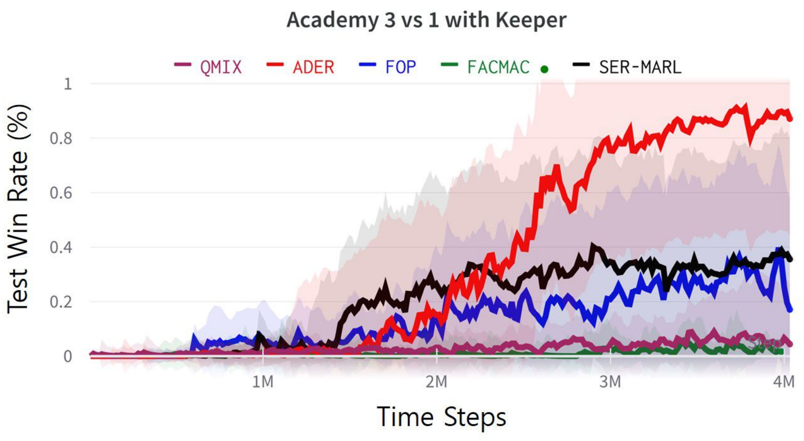
(a) *MMM2* (b) *8m vs 9m* (c) *3s5z*

Figure 8. Average test win rate on the original SMAC maps.

D2. Experiments on google research football environment

We evaluated ADER on the google research football (GRF) environment, which is known as hard exploration tasks. We consider one scenario in GRF named *Academy 3 vs 1 with keeper*. In this environment, the agents receive a reward only when they succeed in scoring, which requires hard exploration. Thus, it is difficult to obtain the reward if all agents focus on exploration simultaneously.

We compared ADER with four baselines: QMIX, FOP, FACMAC, and SER-MARL. Fig. 9 shows the performance of ADER and the baselines, and the y-axis in Fig. 9 denotes the median winning rate over 7 random seed. It is seen in Fig. 9 that ADER outperforms the baselines significantly. Since ADER handles multi-agent exploration-exploitation trade-off across multiple agents and over time, ADER performs better than SER-MARL, which keeps the same level of exploration across agents.

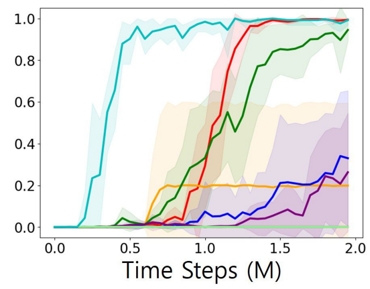


(a) *Academy 3 vs 1 with keeper*

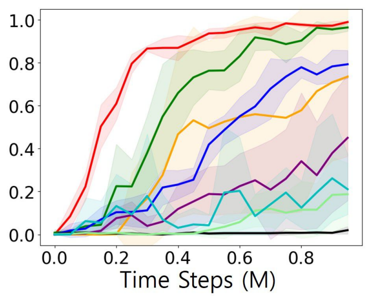
Figure 9. Median test winning rate on Academy 3 vs 1 with keeper

D3. Experiments on the modified SMAC environments

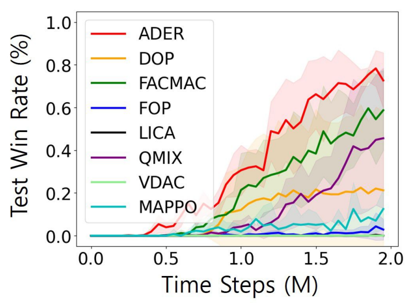
Fig. 10 shows the performance of ADER and the considered seven baselines on the modified SMAC environment. It is seen that ADER outperforms all the considered baselines. Especially, on the hard tasks shown in Fig. 10 , ADER significantly outperforms other baselines in terms of training speed and final performance. This is because those hard maps require high-quality adaptive exploration across agents over time. In the maps *3s vs 3z*, the stalkers (ally) should attack a zealot (enemy) many times and thus the considered reward is rarely obtained. In addition, since the stalker is a ranged attacker whereas the zealot is a melee attacker, the stalker should be trained to attack the zealot at a distance while avoiding the zealot. For this reason, if all stalkers focus on exploration simultaneously, they hardly remove the zealot, which leads to failure in solving the task. Similarly, in the hard tasks with imbalance between allies and enemies such as *MMM2*, and *8m vs 9m*, it is difficult to obtain a reward due to the simultaneous exploration of multiple agents. Thus, consideration of multi-agent exploration-exploitation trade-off is required to solve the task, and it seems that ADER effectively achieves this goal.



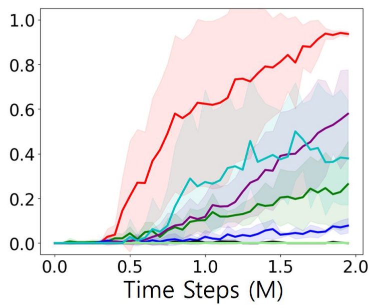
(a) *3s vs 3z*



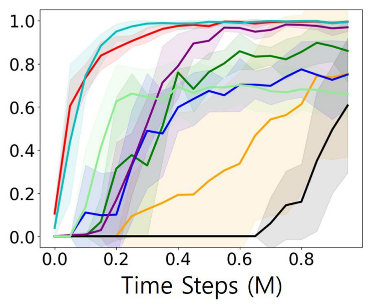
(b) *1c3s5z*



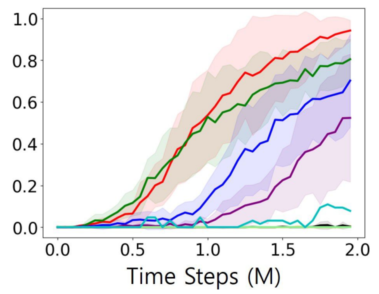
(c) *MMM2*



(d) *8m vs 9m*



(e) *3m*

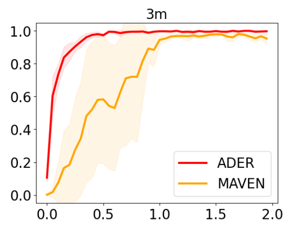


(f) *3s5z*

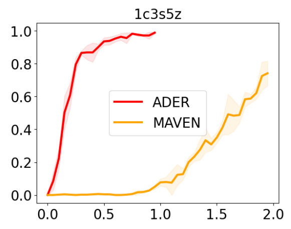
Figure 10. Average test win rate on the sparse SMAC maps.

D4. Comparison with MAVEN

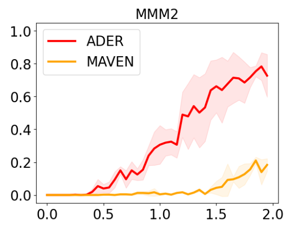
We additionally provide the comparsion of ADER with MAVEN (Mahajan et al., 2019), which is known as improved multi-agent exploration QMIX-based algorithm, on the sparse SMAC environment. It is observed that ADER significantly outperforms MAVEN on the considered environment, as shown in Fig. 11.



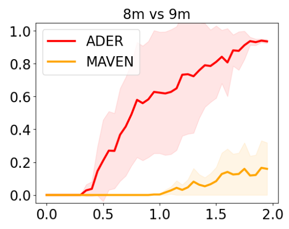
(a) *3m*



(b) *1c3s5z*



(c) *MMM2*

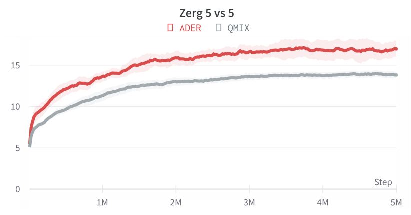


(d) *8m vs 9m*

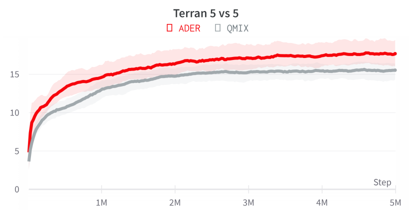
Figure 11. Comparison with MAVEN on the sparse SMAC maps.

D5. Experiments on SMACv2 environments

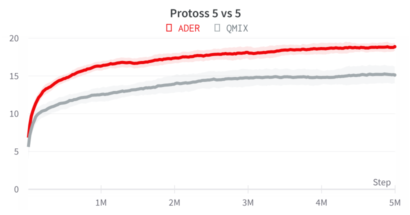
We evaluated ADER with QMIX in the three tasks in SMACv2 environments, which is newly introduced to tackle the lackness of stochasticity in the SMAC environment. As shown in Fig. 12, ADER outperforms QMIX on the considered tasks. In addition, ADER also outperforms MAPPO since it was shown that QMIX outperforms MAPPO on SMACv2 (Ellis et al., 2022).



(a) *Zerg 5 vs 5*



(b) *Terran 5 vs 5*



(c) *Protoss 5 vs 5*

Figure 12. Peformances of ADER (red) and QMIX (gray) on three SMACv2 tasks. y-axis denotes the test return.

Appendix E: Further Related Works

For effective exploration in single-agent RL, several approaches such as maximum entropy/entropy regularization (Haarnoja et al., 2017; 2018a), intrinsic motivation (Chentanez et al., 2004; Badia et al., 2019; Burda et al., 2018), parameter noise (Plappert et al., 2018; Fortunato et al., 2018) and count-based exploration (Ostrovski et al., 2017; Bellemare et al., 2016) have been considered. Also in MARL, exploration has been actively studied in various ways. MAVEN introduced a latent variable and maximized the mutual information between the latent variable and the trajectories to solve the poor exploration of QMIX caused by the representational constraint (Mahajan et al., 2019). Wang et al. (2019) proposes a coordinated exploration strategy by considering the interaction between agents. Liu et al. (2021b) proposes an efficient coordinated exploration method based on restricted space selection to encourage multiple agents to explore worthy state space. Zheng et al. (2021) extends the intrinsic motivation-based exploration method to MARL and utilizes the episodic memory which stores highly rewarded episodes to boost learning. Gupta et al. (2021) promotes joint exploration by learning different tasks simultaneously based on multi-agent universal successor features to address the problem of relative overgeneralization. The aforementioned methods successfully improve exploration in MARL. However, to the best of our knowledge, none of the works address the multi-agent exploration-exploitation tradeoff, which is the main motivation of this paper.

Appendix F: Limitation

In this paper, we only considered a fully cooperative setting where multiple agents share the global reward and showed that the proposed method successfully addresses the multi-agent exploration-exploitation tradeoff in such setting. However, the metric to measure the benefit of exploration can differ in other MARL settings such as mixed cooperative-competitive settings. Thus, we believe finding the metric in other MARL settings can be a good research direction.