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

Anonymous Authors[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 for each agent based on the degree of required exploration. In order to handle instability arising from updating multiple entropy temperature parameters for multiple agents, we disentangle the soft value function into two types: one for pure reward and the other for entropy. By applying multi-agent value factorization to the disentangled value function of pure reward, we obtain a relevant metric to assess the necessary degree of exploration for 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

The goal of RL is to find the 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 (Watkins & Dayan, 1992; Sutton & Barto, 2018), but this is practically impossible due to large state and/or action spaces in real-world problems. Thus, effective exploration, which aims to visit uncharted parts of the environment, has been a core problem in RL, and various 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 investigated. In practical real-world problems, however, the given time for learning is limited and thus the learner should exploit its own policy based on its experiences so far. Therefore, the learner should balance exploration and exploitation in the dimension of time and this is typically called *exploration-exploitation trade-off* in RL.

The problem of exploration-exploitation trade-off becomes more challenging in multi-agent RL (MARL) because the state-action space grows exponentially as the number of agents increases. In addition, the degree of necessary exploration can be different across agents and moreover one agent's exploration can hinder other agents' exploitation. Thus, the balance of exploration and exploitation *across multiple agents* should also be considered for MARL in addition to along the time dimension. We refer to this problem as *multi-agent exploration-exploitation trade-off*. Although there exist many algorithms for better exploration in MARL (Liu et al., 2021; Zhang et al., 2021; Kim et al., 2020; Mahajan et al., 2019), the research on multi-agent exploration-exploitation trade-off has not been investigated much yet.

In this paper, we propose a new framework based on entropy regularization for adaptive exploration in MARL to handle the multi-agent exploration-exploitation trade-off. The proposed framework allocates different target entropy across agents and across time based on our newly-proposed metric for the degree of necessary exploration for each agent. In order to implement the proposed framework, we adopt the method of disentanglement between exploration and exploitation (Han & Sung, 2021; Beyer et al., 2019) to decompose the joint soft value function into two types: one for the return and the other for the entropy sum. This disentanglement alleviates instability which can occur due to the updates of multiple entropy temperature parameters and enables applying the multi-agent value factorization technique to return and entropy separately. To derive a metric for the level of required exploration for each agent, we exploit this value factorization on the disentangled value function of pure return and use the partial derivative of the joint value function of pure return with respect to in-

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

dividual value function. The intuition behind this choice is that the agents having high contributions to the return should focus more on exploitation, whereas the agents having low contributions to the return should explore more to search better actions. Various experiments demonstrate the effectiveness of the proposed framework for multi-agent exploration-exploitation trade-off.

2. Background

Basic setup We consider a decentralized partially observable MDP (Dec-POMDP), which describes a fully cooperative multi-agent task (Oliehoek & Amato, 2016). Dec-POMDP is defined by a tuple $< \mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}, \mathcal{P}, \{\Omega_i\}, \mathcal{O}, r, \gamma >$, where $\mathcal{N} = \{1, 2, \cdots, N\}$ is the set of agents. 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 $a_t^i \in \mathcal{A}_i$, forming 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 $\mathcal{P}(\cdot|s_t, a_t)$ and a joint reward $r(s_t, a_t)$ according to the reward function $r(\cdot, \cdot)$. Each agent i has an observation-action history $\tau_t^i \in (\Omega_i \times \mathcal{A}_i)^*$ and trains its decentralized policy $\pi^i(a^i|\tau^i)$ to maximize the expected cumulative 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 via a centralized way during the training phase (Oliehoek et al., 2008).

Value Factorization In MARL, it is difficult to learn the joint action-value function, which is defined as $Q_{JT}(s, \boldsymbol{\tau}, \boldsymbol{a}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s, \boldsymbol{\tau}, \boldsymbol{a}]$ due to the problem of the curse of dimensionality as the number of agents increases. For efficient learning of the joint action-value function, *value factorization* techniques have been proposed to factorize it into individual action-value functions $Q_i(\tau^i, a^i)$, $i = 1, \cdots, N$. One representative example is value decomposition network (VDN), which factorizes the joint action-value function into the sum of individual action-value functions as $Q_{JT}(\boldsymbol{\tau}, \boldsymbol{a}) = \sum_{i=1}^{N} Q_i(\tau^i, a^i)$. Another representative example is QMIX, which introduces a monotonic constraint between the joint action-value function and the individual action-value function. The joint action-value function in QMIX is expressed as

$$Q_{JT}(s, \boldsymbol{\tau}, \boldsymbol{a}) = f_{mix}(s, Q_1(\tau^i, a^i), \cdots, Q_N(\tau^N, a^N)), \quad \frac{\partial Q_{JT}(s, \boldsymbol{\tau}, \boldsymbol{a})}{\partial Q_i(\tau^i, a^i)} \geq 0, \quad \forall i \in \mathcal{N}, \tag{1}$$

where f_{mix} is a mixing network which combines the individual action-values into the joint action-value based on the global state. In order to satisfy the monotonic constraint $\partial Q_{JT} / \partial Q_i \geq 0$, the mixing network is restricted to have positive weights. There exist other value-based MARL algorithms with value factorization (Son et al., 2019; Wang et al., 2020a). Actor-critic based MARL algorithms also considered value factorization to learn the centralized critic (Peng et al., 2021; Su et al., 2021).

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

$$J_{MaxEnt}(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t (r_t + \alpha \mathcal{H}(\pi(\cdot|s_t)))\right], \tag{2}$$

where $\mathcal{H}(\cdot)$ is the entropy function and α is the temperature parameter which determines the importance of the entropy compared to the reward. Soft actor-critic (SAC) is an off-policy actor-critic algorithm which efficiently solves the maximum entropy RL problem (2) based on soft policy iteration. For this, SAC defines the soft Q function as the sum of the total reward and the future entropy and the corresponding soft Bellman backup operator. The soft Q function for given policy is estimated by repeatedly applying the soft Bellman backup operator based on the fixed-point theorem, and this step is called the soft policy evaluation. Then, the policy is updated based on the evaluated soft Q function and this step is called the soft policy improvement. By iterating the soft policy evaluation and soft policy improvement, called the soft policy iteration, SAC converges to an optimal policy that maximizes (2) within the considered policy class in the case of finite MDPs. SAC also works effectively for large MDPs with function approximation (Haarnoja et al., 2018a).

One issue with SAC is the adjustment of the hyperparameter α in (2), which control the relative importance of the entropy with respect to 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. Because the optimal entropy depends on this magnitude, this dependence makes the temperature adjustment difficult (Haarnoja et al., 2018b). 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 based on approximate dynamic programming. For this, they reformulated the maximum entropy RL as the following entropy-regularized optimization:

$$J_{ER}(\pi_{0:T}) = \mathbb{E}_{\pi_{0:T}}\left[\sum_{t=0}^{T} r_t\right] \quad \text{s.t. } \mathbb{E}_{(s_t, a_t) \sim \pi_t}\left[-\log(\pi_t(a_t|s_t))\right] \geq \mathcal{H}_0 \tag{3}$$

where $\mathcal{H}_0$ is the target entropy. Exploiting the fact that π_t affects only the present and future, the technique of dy-

namic programming is used, i.e., $\max_{\pi_{t:T}} \mathbb{E}[\sum_{i=t}^{T} r_i] = \max_{\pi_t} \left\{ \mathbb{E}[r_t] + \max_{\pi_{t+1:T}} \mathbb{E}[\sum_{i=t+1}^{T} r_i] \right\}$. Then, the backward recursion can be applied to obtain optimal α at time step t based on the technique of Lagrange multiplier by the dual optimization:

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

where π_t^* is the maximum entropy policy at time step t. In the infinite-horizon case, the discount factor γ is included and π_t^* is replaced with the current approximate maxent solution by SAC. Thus, 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 Here, we mainly focus on the entropy-based MARL. 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 based on a simple technique. The entropy regularization technique proposed in (Zhou et al., 2020) dynamically controls the magnitude of the gradient regarding entropy to address the high sensitivity of the temperature parameter caused by the curvature of derivative of entropy. It was shown that LICA allows multiple agents to perform consistent level of exploration. However, LICA does not maximize the cumulative sum of entropy but regularize the entropy of policy. 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 and to factorize the joint soft Q-function. 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 on exploration at the beginning of training, which considers exploration-exploitation only in time dimension in a heuristic way.

A key point is that the aforementioned algorithms maximize or regularize the entropy of the policies of multiple agents to encourage *the same level of exploration across the agents*. Such exploration is still useful for several benchmarks 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 on one agent, however, the contribution to the global reward and that to the sum 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

In order to address the aforementioned problems, we propose an **AD**aptive **E**ntropy-**R**egularization framework (ADER) for multi-agent reinforcement learning, which can balance *exploration and exploitation across multiple agents* by learning and controlling the target entropy for each agent. We first provide an example to motivate the learning problem of exploitation-and-exploration trade-off for multi-agent RL and then describe our framework to address this problem, including a novel method determining the target entropy based on the degree of necessity of exploration.

3.1. Motivation

The convergence of model-free RL requires the assumption that all state-action pairs should be visited infinitely often (Watkins & Dayan, 1992; Sutton & Barto, 2018), and this necessitates exploration. In practice, however, the number of time steps during which an agent can interact with the environment is limited. Thus, a balance between exploration and exploitation in the dimension of time is crucial for high performance in RL. Furthermore, in the case of MARL, a balance between exploration and exploitation in the dimension of agents should be considered. This is because 1) the degree of necessity of exploration can be different across multiple agents and 2) one agent's exploration can hinder other agents' exploitation, resulting in the situation that simultaneous exploration of multiple agents can make learning unstable. We refer to this problem as *multi-agent exploration-exploitation trade-off*. 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). We should allocate higher target entropies to the agents who need more exploration and lower target entropies to the agents who need more exploitation. To make such adaptive target entropy control possible, we need a metric to capture the degree of required exploration, which changes during the learning process. In order to see the necessity of such adaptive exploitation-exploration trade-off control in MARL, let us consider a modified continuous cooperative matrix game (Peng et al., 2021). The considered game consists of two agents: each agent has an one-dimensional continuous action a^i which is bounded in $[-1, 1]$. The state of the environment is the position, described by the $\boldsymbol{s} = (x, y)$, in the 2-dimensional plane in Fig. 1, and the state is determined by

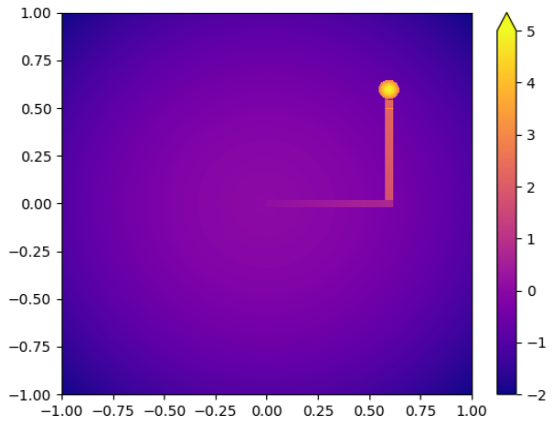


Figure 1. Reward surface in the considered continuous cooperative matrix game. a_1 and a_2 correspond to x-axis and y-axis, respectively.

the joint action as $\boldsymbol{s} = (x, y) = \boldsymbol{a} = (a^1, a^2)$. The shared reward is determined by the state/joint action, and the reward surface is given 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 reward gradually increases only along the path as the position approaches the center of the circle and the maximum reward is 5. There is a penalty if the joint action yields the position outside the path or the circle, and the penalty value increases as the outside position is farther from the origin $(0, 0)$. The agents start from the origin with initial action pair $\boldsymbol{a} = (0, 0)$ and want to learn to reach the circle along the path. 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 upto 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 upto 0.6 because of the penalty. Once the joint action is trained to $(0.6, 0)$, on the other hand, the necessity of exploration is changed. In this phase, a_1 should keep its action at 0.6, whereas a_2 should be trained to increase upto 0.55. As seen in this example, it is important to control the trade-off between exploitation and exploration across multiple agents. In addition, we should update the trade-off over time because the required trade-off can change during the learning process. As we will see in Section 4, a method that retains the same or different-but-constant level of exploration across all agents fails to learn in this continuous cooperative matrix game. Thus, we need a framework that can adaptively learn appropriate levels of exploration for all agents over time, considering the time-varying multi-agent exploration-exploitation trade-off.

3.2. Adaptive Entropy-Regularized MARL

We now propose our framework named ADER enabling adaptive exploration capturing the multi-agent exploration-exploitation trade-off. One can adopt the entropy constrained objective defined in (3) and extend it to multi-agent systems. A simple extension is to maximize the team reward while keeping the average 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 different but fixed target entropies for multiple agents. However, this case cannot handle the time-varying behavior of multi-agent exploitation-exploration trade-off, discussed in the previous subsection with Fig. 1. To incorporate the multi-agent exploration-exploitation trade-off, we consider the following optimization problem:

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

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 $\mathcal{H}_0$. Then, this total entropy budget $\mathcal{H}_0$ is shared by all agents. When some agents' target entropies are high for more exploration, the target entropies 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 is how to learn individual target entropy values $\mathcal{H}_1, \cdots, \mathcal{H}_N$ (such that $\sum_{j=1}^{N}\mathcal{H}_j = \mathcal{H}_0$) over time (i.e., time dimension) as the learning progresses.

We postpone the presentation of our method of learning the individual target entropy values to Section 3.4. Here, we consider how to solve the problem (5) when $\mathcal{H}_1, \cdots, \mathcal{H}_N$ are determined. In order to solve the problem (5) for determined $\mathcal{H}_1, \cdots, \mathcal{H}_N$, 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 technique of Lagrange multiplier, 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 this, the joint soft Q-function $Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ can be defined as $Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) :=$

$$r_t + \mathbb{E}_{\tau_{t+1}\sim\pi}\Big[\sum_{l=t+1}^{\infty}\gamma^{l-t}(r_l + \sum_{i=1}^{N}\alpha^i\mathcal{H}(\pi^i(\cdot|\tau_l^i)))\Big], \quad (6)$$

and then this joint soft Q-function is estimated based on the following Bellman backup operator: $\mathcal{T}^{\pi}Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) :=$

$$r_t + \gamma \mathbb{E}_{\tau_{t+1}}\left[V(s_{t+1}, \boldsymbol{\tau_{t+1}})\right] \text{ where } V_{JT}(s_t, \boldsymbol{\tau_t}) = \mathbb{E}_{a_t \sim \boldsymbol{\pi}}\left[Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) - \sum_{i=1}^{N} \alpha^i \log \pi(a_t^i|\tau_t^i)\right]. \quad (7)$$

However, optimizing the objective (5) based on the joint soft Q-function in (6) and the corresponding Bellman operator (7) has several limitations. First, the estimation of the joint soft Q-function can be unstable due to the changing $\{\alpha^i\}_{i=1}^N$ in (7) as the determined target entropy values are updated over time. Second, we cannot apply value factorization to return and entropy separately because the joint soft Q-function defined in (6) estimates only the sum of return and entropy. For a single agent, the contribution to the global reward may be different from that to the total entropy. Thus, learning to decompose the entropy can prevent the mixing network from learning to decompose the global reward. Furthermore, due to the inseparability of reward and entropy, it is difficult to pinpoint each agent's contribution to the global reward itself, which actually provides the information about the goodness of each agent's current policy to assess the necessity for more exploration.

3.3. Disentangled Exploration and Exploitation

To address the aforementioned problems and facilitate the acquisition of a metric for the degree of required exploration for each agent in MARL, we disentangle exploration from exploitation by decomposing the joint soft Q-function into two types of Q-function: One for reward and the other for entropy. That is, the joint soft Q-function is decomposed as $Q_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) + \sum_{i=1}^N \alpha^i Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$, where $Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ and $Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ are the joint action value function for reward and the joint action value function for the entropy of Agent i's policy, respectively, given by

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

$$Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = \mathbb{E}_{\boldsymbol{\tau_{t+1}} \sim \boldsymbol{\pi}}\left[\sum_{l=t+1}^{\infty} \gamma^{l-t} \mathcal{H}(\pi^i(\cdot|\tau_t^i))\right], \quad (9)$$

for all $i \in \mathcal{N}$. The action value functions $Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ and $Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})$ can be estimated based on their corresponding Bellman backup operators, defined by

$$\mathcal{T}^{\pi}_R Q^R_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) := r_t + \gamma \mathbb{E}\left[V^R_{JT}(s_t, \boldsymbol{\tau_{t+1}})\right],$$
$$\mathcal{T}^{\pi}_{H,i} Q^{H,i}_{JT}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) := \gamma \mathbb{E}\left[V^{H,i}_{JT}(s_t, \boldsymbol{\tau_{t+1}})\right] \quad (10)$$

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

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

Proof: See Appendix A.

Now we apply value decomposition with a mixing network (Rashid et al., 2018) to represent each of all joint action value and value functions as a mixture of individual value functions. For instance, the joint value function for reward $V^R_{JT}(s, \boldsymbol{\tau})$ is decomposed as $V^R_{JT}(s, \boldsymbol{\tau}) = f^{V,R}_{mix}(s, V^R_1(\tau^1), \cdots, V^R_N(\tau^N))$, where $V^R_i(\tau^i)$ is the individual value function of Agent i and $f^{V,R}_{mix}$ is the mixing network for the joint value function for reward. Similarly, we apply value decomposition and mixing networks to $Q^R_{JT}(\boldsymbol{\tau_t}, \boldsymbol{a_t})$ and $Q^{H,i}_{JT}(\boldsymbol{\tau_t}, \boldsymbol{a_t})$, $i \in \mathcal{N}$.

Based on the decomposed joint soft Q-functions, the optimal policy 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: $\max_{\boldsymbol{\pi}_{t:T}} \mathbb{E}\left[\sum_{i=t}^T r_i\right] =$

$$\max_{\pi_t}\left(\mathbb{E}[r_t] + \max_{\pi_{t+1:T}}\left(\mathbb{E}[\sum_{i=t+1}^{T} r_i],\right)\right) \text{ s.t. } \mathbb{E}_{(s_t, \boldsymbol{a_t}) \sim \boldsymbol{\pi}_t}\left[-\log(\pi_t^i(a_t^i|\tau_t^i))\right] \geq \mathcal{H}_i, \ \ \forall t, i. \quad (11)$$

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:

$$\boldsymbol{\pi}_t^* = \arg\max_{\boldsymbol{\pi}_t} \mathbb{E}_{\boldsymbol{a_t} \sim \boldsymbol{\pi}_t}\Big[\underbrace{Q^{R*}_{JT}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}_t)}_{(a)} + \sum_{i=1}^{N} \alpha_t^i \underbrace{(Q^{H*,i}_{JT}(s_t, \boldsymbol{\tau}_t, \boldsymbol{a}_t) - \log \boldsymbol{\pi}_t(a_t^i|\tau_t^i))}_{(b)}\Big] \quad (12)$$

$$\alpha_t^{i*} = \arg\min_{\alpha_t^i} \mathbb{E}_{\boldsymbol{a_t} \sim \boldsymbol{\pi}_t^*}\left[-\alpha_t^i \log \boldsymbol{\pi}_t^*(a_t^i|\tau_t^i) - \alpha_t^i \mathcal{H}_i\right], \quad (13)$$

for all $i \in \mathcal{N}$. In the infinite-horizon case, (12) and (13) provide the update formulae at time step t, and the optimal policy is replaced with the current approximate multi-agent

maximum-entropy solution, which can be obtained by extending SAC to MARL. Note that maximizing the term (a) in (12) corresponds to the ultimate goal of MARL, i.e., the expected return. On the other hand, maximizing the term (b) in (12) corresponds to enhancing exploration of Agent i.

3.4. Learning Individual Target Entropies

In the ADER formulation (5), the amount of exploration for Agent i can be controlled by the target entropy $\mathcal{H}_i$ under the sum constraint $\sum_{j=1}^{N} \mathcal{H}_j = \mathcal{H}_0$. In this subsection, we describe the proposed method to determine the target entropy for each agent. First, we represent the target entropy of Agent i as $\mathcal{H}_i = \beta_i \times \mathcal{H}_0$ with $\sum_{i=1}^{N} \beta_i = 1$ to satisfy the entropy sum constraint. To properly assign an individual target entropy to each agent, i.e., determine $\beta_i,\ i = 1, \cdots, N$, we need a metric that measures the necessity of exploration for each agent. For this, we exploit our value factorization and mixing network for the disentangled joint value function for pure reward V_{JT}^R of which factorization is given by $V_{JT}^R(s,\tau) = f_{mix}^{V,R}(s, V_1^R(\tau^1), \cdots, V_N^R(\tau^N))$ with the mixing constraint $\partial V_{JT}^R / \partial V_i^R \geq 0$ (Su et al., 2021; Rashid et al., 2018). Note that the partial derivative $\partial V_{JT}^R / \partial V_i^R$ denotes the change in the joint (pure reward) value with respect to the change in the local value of Agent i. When this quantity is large, the contribution of Agent i to the joint (reward) value is large and the policy of Agent i can be considered to work effectively with more exploitation preferred. When this quantity is small, on the other hand, the contribution of Agent i to the joint (reward) value is small and the policy of Agent i can be considered to operate poorly and need more exploration. Hence, we propose using the negative of this partial derivative as a metric to assess the necessity for exploration. Thus, for $\mathcal{H}_0 \geq 0$, we set the coefficients β_i for determining the individual target entropy $\mathcal{H}_i$ as $\boldsymbol{\beta} = \left[\beta_1, \cdots, \beta_i, \cdots, \beta_N\right] =$

$$\text{Softmax}\Bigg[-\mathbb{E}\Big[\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_1^R(\tau^1)}\Big], \cdots, -\mathbb{E}\Big[\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_i^R(\tau^i)}\Big], \cdots, -\mathbb{E}\Big[\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_N^R(\tau^N)}\Big]\Bigg]. \quad (14)$$

The relative required level of exploration across agents can change as the learning process and this is captured in these partial derivatives. During the training phase, we continuously compute (B.7) from the samples in the replay buffer and set the target entropies. Instead of using the computed value directly, we apply exponential moving average (EMA) filtering for smoothing. The exponential moving average filter prevents the target entropy from changing abruptly. A rapid change in the target entropy can cause instability in policy learning by perturbing the temperature parameter $\{\alpha^i\}_{i=1}^N$ too much. The output of EMA filter $\boldsymbol{\beta}^{EMA} = \left[\beta_1^{EMA}, \cdots, \beta_N^{EMA}\right]$ is computed recursively as

$$\boldsymbol{\beta}^{EMA} \leftarrow (1-\xi)\boldsymbol{\beta}^{EMA} + \xi\boldsymbol{\beta} \quad (15)$$

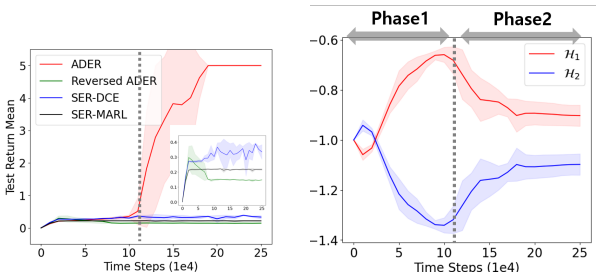
where $\boldsymbol{\beta}$ is given in (B.7) and $\xi \in [0,1]$. Thus, the target entropy is given by $\mathcal{H}_i = \beta_i^{EMA} \times \mathcal{H}_0$.

Finally, the procedure of ADER at time step t is composed of the policy evaluation based on the Bellman operators in (16) and Proposition 1, the policy update for policy and temperature parameters in (12) and (13), and the target entropy update in (B.7) and (B.8). The detailed implementation is provided in Appendix B.

4. Experiments

In this section, we provide numerical results and ablation studies to evaluate ADER. We first present the result on the continuous cooperative matrix game described in Sec. 3.1, showing multi-agent exploration-exploitation trade-off, and then results including sparse StarCraft II micromanagement (SMAC) tasks (Samvelyan et al., 2019).

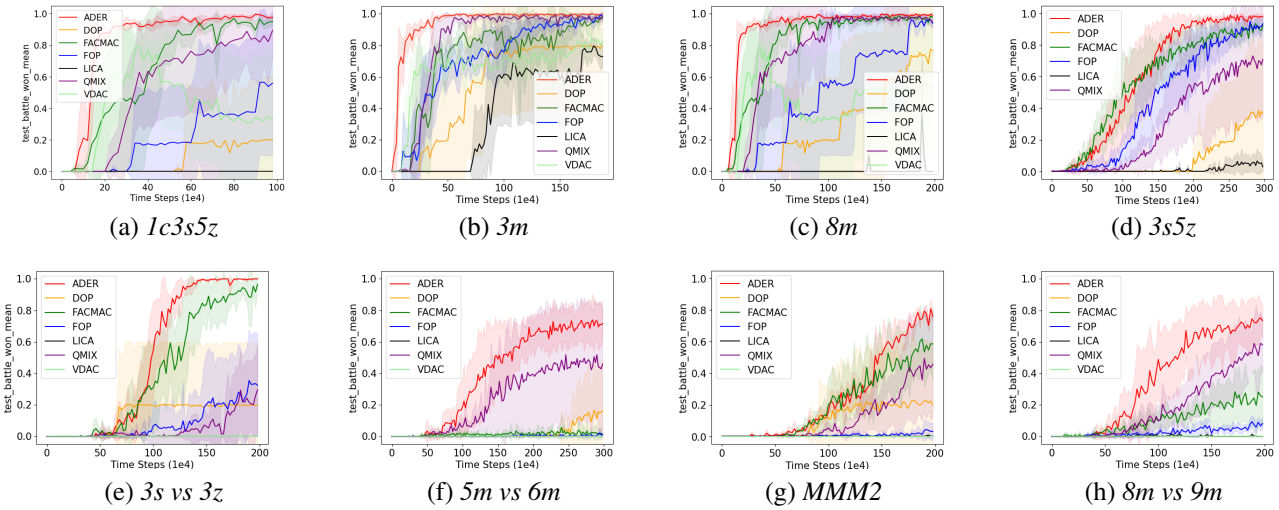
Continuous Cooperative Matrix Game As mentioned in Sec.3.1, the goal of this environment is to learn two actions a_1 and a_2 so that the position (a_1, a_2) starting from $(0,0)$ to reach the target circle along a narrow path, as shown in Fig. 1. The maximum reward 5 is obtained if the position reaches the center of the circle. We compare ADER with three baselines. One is SER-MARL with the same target entropy for all agents. Another is SER-MARL with different but constant target entropies of two agents (SER-DCE). Here, we set a higher target entropy for a_1 than a_2. The other is Reversed ADER, which reversely uses the proposed metric $-\partial V_{JT}^R / \partial V_i^R$ for the level of required exploration. That is, Reversed ADER assigns a high target entropy to the agent whose contribution to the joint value is large.



(a) Averaged test return (b) Target entropy of ADER

Figure 2. Comparison of ADER with the baselines on the continuous cooperative matrix game.

Fig. 2(a) shows the performances 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 entropies of SER-DCE are



(a) *1c3s5z* (b) *3m* (c) *8m* (d) *3s5z*

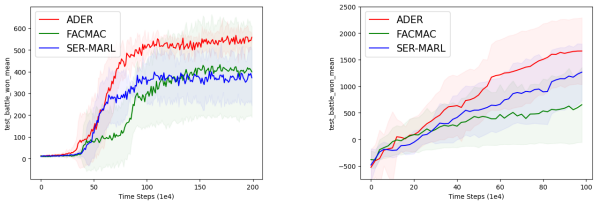
(e) *3s vs 3z* (f) *5m vs 6m* (g) *MMM2* (h) *8m vs 9m*

Figure 3. Average test win rate on eight different SMAC maps

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 entropies $\mathcal{H}_1$ 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 line in Figs. 2(a) and (b) denotes 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 entropies 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 this continuous cooperative matrix game, ADER successfully learns the time-varying trade-off of multi-agent exploration-exploitation by learning appropriate target entropies for all agents.

Continuous Action Tasks We evaluated ADER on two complex continuous action tasks: multi-agent HalfCheetah (Peng et al., 2021) and heterogeneous predator-prey. The multi-agent HalfCheetah divides the body into disjoint subgraphs and each sub-graph corresponds to an agent. We used 6×1-HalfCheetah, which consists of six agents with one action dimension. Next, the heterogeneous predator-prey 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 tradeoff should be considered. Here, we used two baselines: SER-MARL and FACMAC (Peng et al., 2021). As seen in Fig. 4 showing the performances of ADER and the baselines averaged over 9 random seeds, ADER outperforms the considered baselines.



(a) H-PP (b) HalfCheetah(6×1)

Figure 4. Comparison of ADER with SER-MARL and FACMAC on multi-agent HalfCheetah and hetrogenuous predator prey (H-PP)

Starcraft II We also 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 dead reward and time-penalty reward. The dead reward is given only when an ally or an enemy dies. Unlike the original reward in SMAC which gives the hit-point damage dealt as a reward, multiple agents do not receive a reward for damaging the enemy immediately in our sparse reward setting. We compare ADER with six state-of-the-art baselines: DOP (Wang et al., 2020b), FACMAC (Peng et al., 2021),

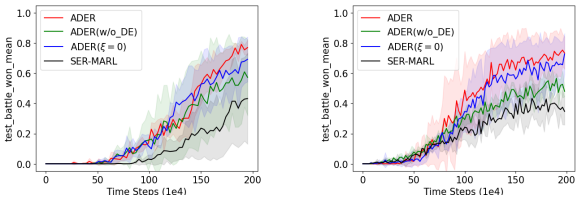
FOP (Zhang et al., 2021), LICA (Zhou et al., 2020), QMIX (Rashid et al., 2018) and VDAC(Su et al., 2021). For evaluation, we conduct experiments on eight different SMAC maps with 5 different random seeds. Fig. 4 shows the performance of ADER and the considered six baselines on the modified SMAC environment. It is seen that ADER outperforms all the considered baselines. Especially, on the hard tasks shown in Figs. 4(e)-(h), 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 *5m vs 6m*, *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.

Ablation Study We provide an analysis of learning target entropy in the continuous cooperative matrix game. Through the analysis, we can see how the changing target entropy affects the learning as seen in Fig. 2. In addition, 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 without the EMA filter. As seen in Fig. 5, it seems that the EMA filter enhances the stability of ADER. Lastly, 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.

We provided the training details for all considered environments in Appendix C.

5. Conclusion

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 entropies across agents over time by using a newly-proposed metric to measure the necessary degree of exploration for each agent. Numerical results on various tasks including the sparse SMAC environment show that ADER can properly 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 exploration methods for MARL such as intrinsic motivation.



(a) *MMM2* (b) *8m vs 9m*

Figure 5. Ablation study

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.

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.

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.

Kim, W., Jung, W., Cho, M., and Sung, Y. A maximum mutual information framework for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.02732*, 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, 2021.

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

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.

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

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, J., Ren, Z., Liu, T., Yu, Y., and Zhang, C. Qplex: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2020a.

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*, 2020b.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3):279–292, 1992.

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.

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 2. *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

$$\begin{aligned} \mathcal{T}_R^\pi Q_{JT}^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) &:= r_t + \gamma \mathbb{E}\left[V_{JT}^R(s_{t+1}, \boldsymbol{\tau_{t+1}})\right], \quad \text{where} \\ V_{JT}^R(s_t, \boldsymbol{\tau_t}) &= \mathbb{E}\left[Q_{JT}^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t})\right] \end{aligned} \tag{16}$$

$$\begin{aligned} \mathcal{T}_{H,i}^\pi Q_{JT}^{H,i}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) &:= \gamma \mathbb{E}\left[V_{JT}^{H,i}(s_{t+1}, \boldsymbol{\tau_{t+1}})\right], \quad \text{where} \\ V_{JT}^{H,i}(s_t, \boldsymbol{\tau_t}) &= \mathbb{E}\left[Q_{JT}^{H,i}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) - \alpha^i \log \pi(a_t^i|\tau_t^i)\right]. \end{aligned} \tag{17}$$

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 (16), we have

$$\mathcal{T}_R^\pi Q^R(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = r_t + \gamma \mathbb{E}_{s_{t+1}, \boldsymbol{\tau_{t+1}}, \boldsymbol{a_{t+1}}}\left[Q^R(s_{t+1}, \boldsymbol{\tau_{t+1}}, \boldsymbol{a_{t+1}})\right]. \tag{18}$$

Then, we have

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

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 $\| \sum_{\substack{s_{t+1}, \boldsymbol{\tau_{t+1}} \\ \boldsymbol{a_{t+1}}}} \boldsymbol{\pi}(\boldsymbol{a_{t+1}}|\boldsymbol{\tau_{t+1}}) p(s_{t+1}, \boldsymbol{\tau_{t+1}}|s_t, \boldsymbol{\tau_t}, \boldsymbol{a}_t)\|_\infty \leq 1$. 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 (17), we have

$$\mathcal{T}_{H,i}^\pi Q^{H,i}(s_t, \boldsymbol{\tau_t}, \boldsymbol{a_t}) = \gamma \mathbb{E}\left[Q^{H,i}(s_{t+1}, \boldsymbol{\tau_{t+1}}, \boldsymbol{a_{t+1}}) - \alpha^i \log \pi(a_{t+1}^i|\tau_{t+1}^i))\right]. \tag{19}$$

Then, we have

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

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

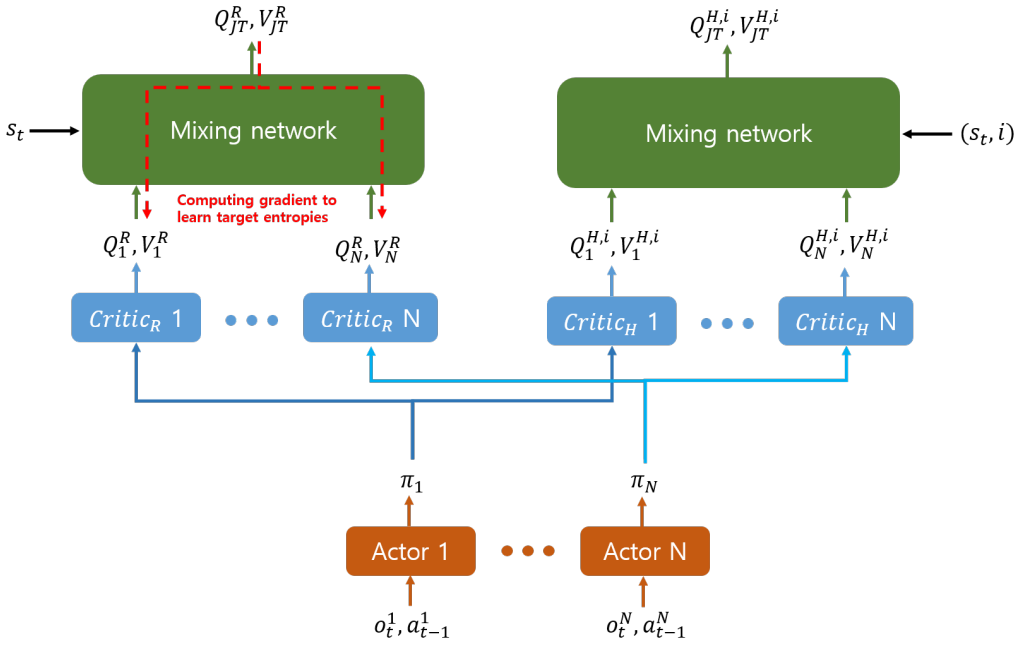


Figure 6. Overall architecture of the proposed ADER

Appendix B: Detailed Implementation for ADER

Here, we describe the implementation of ADER for discrete action tasks based on (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 $\{\pi_{\phi_i}^i\}_{i=1}^N$, the joint action value functions Q_{JT,θ_R}^R and $Q_{JT,\theta_{H,i}}^{H,i}$ 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

$$L(\boldsymbol{\phi}) = \mathbb{E}_{(s_t,\boldsymbol{\tau_t})\sim\mathcal{D},\{a_t^i\sim\pi^i(\cdot|\tau_t^i)\}_{i=1}^N}\left[\sum_{i=1}^N \alpha^i(\log\pi_{\phi_i}^i(a_t^i|\tau_t^i) - Q_{JT,\theta_{H,i}}^{H,i}(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t})) - Q_{JT,\theta_R}^R(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t})\right], \tag{B.1}$$

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

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

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

where $V_{JT,\bar{\theta}_R}^R$ and $V_{JT,\bar{\theta}_{H,i}}^{H,i}$ are implicitly parameterized via the parameters of joint action value functions, and directly computed as follows:

$$V_{JT,\bar{\theta}_R}^R(s_t,\boldsymbol{\tau_t}) = \mathbb{E}\left[Q_{JT,\bar{\theta}_R}^R(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t})\right] \tag{B.4}$$

$$V_{JT,\bar{\theta}_{H,i}}^{H,i}(s_t,\boldsymbol{\tau_t}) = \mathbb{E}\left[Q_{JT,\bar{\theta}_{H,i}}^{H,i}(s_t,\boldsymbol{\tau_t},\boldsymbol{a_t}) - \alpha^i\log\pi(a_t^i|\tau_t^i)\right]. \tag{B.5}$$

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.

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

Initialize parameters $\{\phi_i\}_{i=1}^N$, θ_R, $\{\theta_{H,i}\}_{i=1}^N$, $\bar{\theta}_R$, $\{\bar{\theta}_{H,i}\}_{i=1}^N$
Generate a trajectory τ by interacting with the environment by using the joint policy $\boldsymbol{\pi}$ and store τ in the replay memory
for $episode = 1, 2, \cdots$ **do**
  Generate a trajectory τ by using the joint policy $\boldsymbol{\pi}$ and store τ 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 θ_R, $\{\theta_{H,i}\}_{i=1}^N$ by minimizing the loss functions Eq. (B.2) and Eq. (B.3)
    Update α^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 θ_R and $\{\theta_{H,i}\}_{i=1}^N$
  end for
end for

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

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

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 $\boldsymbol{\beta} = \left[\beta_1, \cdots, \beta_i, \cdots, \beta_N\right] =$

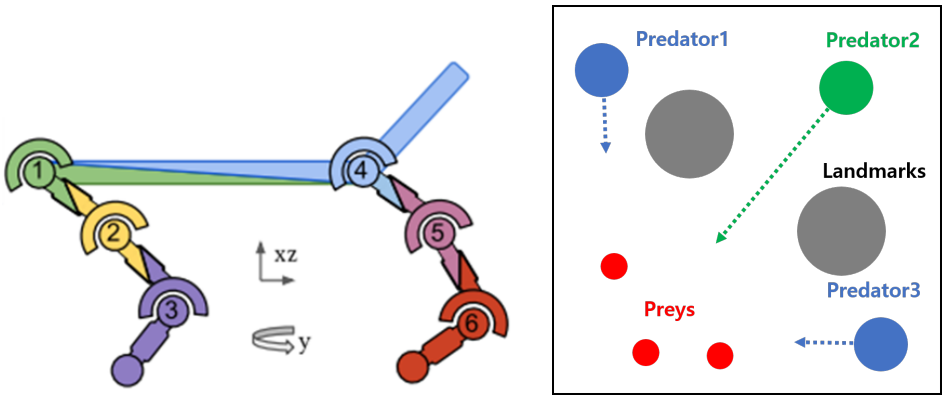
$$\text{Softmax}\left[-\mathbb{E}\Big[\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_1^R(\tau^1)}\Big], \cdots, -\mathbb{E}\Big[\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_i^R(\tau^i)}\Big], \cdots, -\mathbb{E}\Big[\frac{\partial V_{JT}^R(s,\boldsymbol{\tau})}{\partial V_N^R(\tau^N)}\Big]\right]. \tag{B.7}$$

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 contribution 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

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

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

Appendix C: Training details



(a) 6×1-HalfCheetah (b) H-PP

Figure 7. Considered continuous action tasks

C1. Environment Details

Multi-agent HalfCheetah We considered the multi-agent HalfCheetah introduced in (Peng et al., 2021). As illustrated in Fig. 7 (a), 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. As illustrated in Fig. 7 (b), the considered heterogeneous predator-prey consists of three predator agents, where the maximum speeds of an agent ($v^1_{max} = 1.0$) and other agents ($v^2_{max} = 0.75$) are different, three preys with the maximum speed ($v^3_{max} = 1.25$) 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 $\alpha^i_{init} = e^{-2}$ 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}
8m	0.1	0.9	e^{-2}
3s5z	0.05	0.9	e^{-3}
3s vs 3z	0.1	0.9	e^{-3}
5m vs 6m	0.15	0.5	e^{-3}
MMM2	0.1	0.9	$e^{-2.5}$
8m vs 9m	0.1	0.9	e^{-2}

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.