SMIX(λ): Enhancing Centralized Value Functions for Cooperative Multi-Agent Reinforcement Learning

Xinghu Yao*, Chao Wen*, Yuhui Wang and Xiaoyang Tan
Nanjing University of Aeronautics and Astronautics, China
{xinghuyao, chaowen, y.wang, x.tan}@nuaa.edu.cn

***Abstract*—Learning a stable and generalizable centralized value function (CVF) is a crucial but challenging task in multi-agent reinforcement learning (MARL), as it has to deal with the issue that the joint action space increases exponentially with the number of agents in such scenarios. This paper proposes an approach, named SMIX(λ), that uses an off-policy training to achieve this by avoiding the greedy assumption commonly made in CVF learning. As importance sampling for such off-policy training is both computationally costly and numerically unstable, we proposed to use the λ-return as a proxy to compute the TD error. With this new loss function objective, we adopt a modified QMIX network structure as the base to train our model. By further connecting it with the $Q(\lambda)$ approach from an unified expectation correction viewpoint, we show that the proposed SMIX(λ) is equivalent to $Q(\lambda)$ and hence shares its convergence properties, while without being suffered from the aforementioned curse of dimensionality problem inherent in MARL. Experiments on the StarCraft Multi-Agent Challenge (SMAC) benchmark demonstrate that our approach not only outperforms several state-of-the-art MARL methods by a large margin, but also can be used as a general tool to improve the overall performance of other CTDE-type algorithms by enhancing their CVFs.**

***Index Terms*—Deep reinforcement learning (DRL), multi-agent reinforcement learning (MARL), multi-agent systems, StarCraft Multi-Agent Challenge (SMAC).**

I. Introduction

Recently, reinforcement learning (RL) has made great success in a variety of domains, from game playing [1], [2] to complex continuous control tasks [3]–[5]. However, many real-world problems are inherently multi-agent in nature, such as network packet routing [6], automatic control [7], [8], social dilemmas [9], consensus in multi-agent systems [10]–[12] and multi-player video games [13], which raises great challenges that are never encountered in single-agent settings.

In particular, the main challenges in multi-agent environments include the dimension of joint action space that grows exponentially with the number of agents [14], [15], unstable environments caused by the interaction of individual agents [16], [17], and multi-agent credit assignment in cooperative scenarios with global rewards [14], [15]. These challenges make it troublesome for both fully centralized methods which consider all agents as a single meta agent and fully decentralized methods which individually train each agent by treating other agents as part of the environment .

Recently the paradigm of centralized training with decentralized execution (CTDE) has become popular for multi-agent reinforcement learning [14], [15], [18], [19] due to its conceptual simplicity and practical effectiveness. Its key idea is to learn a centralized value function (CVF) shared by all the agents during training, while each agent acts in a decentralized manner during the execution phase. The CVF works as a proxy to the environment for each agent, through which individual value/advantage functions for each agent can be conveniently learned by incorporating appropriate credit assignment mechanism.

Unfortunately, the central role played by the CVF in the CTDE approach seems to receive inadequate attention in current practice - it is commonly treated in the same way as in single-agent settings [14], [15], [17], [20], leading to larger estimation error in multi-agent environments. Furthermore, to reduce the difficulty of decomposing the centralized value function to individual value functions, many algorithms impose extra structural assumptions onto the hypothesis space of the CVF during training. For example, VDN [20], QMIX [15], and QTRAN [21] assume that the optimal joint action is equivalent to the collection of each agent's optimal action.

On the other hand, performing an accurate estimation of CVF in multi-agent environments is inherently difficult due to the following reasons: 1) the "curse of dimensionality" [22] of the joint action space results in the sparsity of experiences; 2) the challenges of non-Markovian property [16] and partial observability in multi-agent environments become even more severe than in single-agent settings; 3) the dynamics of multi-agent environments are complex and hard to model, partially due to the complicated interactions among agents. In practice, these factors usually contribute to an unreliable and unstable CVF with high bias and variance.

To tackle these difficulties, this work proposes a new sample efficient multi-agent reinforcement learning method, named SMIX(λ), under the framework of CTDE. The SMIX(λ) improves the centralized value function estimation with an off policy-based CVF learning method which removes the need of explicitly relying on the centralized greedy behavior (CGB) assumption (see (1)) during training, and incorporates the λ-return [23] to better balance the bias and variance trade-off and to better account for the environment's non-Markovian property. The particular off-policy learning mechanism of SMIX(λ) is motivated by importance sampling but

*Equal contribution.
Code is available at: https://github.com/chaovven/SMIX

is implemented with experience replay, which is shown to have a close connection with a previous off-policy $Q(\lambda)$ approach for single-agent learning. With all these elements, the SMIX(λ) method effectively improves the sample efficiency and stabilizes the training procedure - on the benchmark of the StarCraft Multi-Agent Challenge (SMAC) [24], SMIX(λ) is demonstrated to achieve state-of-the-art performance in all scenarios considered in our experiments. Furthermore, significant performance improvements by existing CTDE-type MARL algorithms are observed by replacing their CVF estimator with the newly proposed SMIX(λ).

A preliminary version of this work appears in [25]. However, due to page limits, [25] fails to cover all important information about SMIX(λ). This expanded version aims to help readers gain a more comprehensive understanding of SMIX(λ). Specifically, with the addition of the related work section (Section II) and more discussions in subsection IV-C, this release provides a more intuitive motivation to adopt the off-policy methods and to relax the importance sampling ratio in SMIX(λ). Moreover, the detailed proofs of theorems and more derivation details are included in order to provide a more detailed description of the theoretical properties of QMIX, SMIX(λ) and $Q(\lambda)$ [41]. Besides, Figure 2 illustrates that the estimation method of the CVF in SMIX(λ) can be easily applied to other popular value-based and actor-critic-based CTDE methods. And more experimental results have been added to show the effectiveness and generality of this estimation method. Last but not least, more implementation details are presented in Algorithm 1 and Section VI, which can improve the reproducibility of SMIX(λ).

In what follows, we first introduce the related work in Section II, then after a brief discussion on the characteristics of the hypothesis space of CVF in Section III, the proposed SMIX(λ) method and its theoretical analysis are respectively described in Section IV and Section V. Main experimental results and ablation studies are given in Section VI and we conclude the paper in Section VII.

II. Related Work

Deep reinforcement learning (DRL) has made significant progress in recent years with the powerful representation capabilities of deep neural networks [1], [2], [26]. However, challenges in multi-agent scenarios such as the unstable environments and curse of dimensionality make it hard to apply classic DRL methods to multi-agent environments [14], [15], [27], [28].

The CTDE paradigm provides a simple solution to the above issue by separating the agent learning and execution, under the greedy assumption that the optimal actions for individual agents lead to optimal joint action. It has gradually become the de facto standard in cooperative multi-agent scenarios due to its conceptual simplicity and practical effectiveness. Representative methods include COMA [14], VDN [20], QMIX [15] and QTRAN [21] – COMA is an on-policy actor-critic method that uses a carefully designed counterfactual baseline to perform credit assignment, while VDN, QMIX, and QTRAN are typical value-based CTDE methods by learning individual agents through learning a centralised value function (CVF) first.

Our SMIX(λ) belongs to the CTDE framework as well, but we focus more on how to learn efficiently in the non-Markovian environments and how to perform an accurate estimation of the CVF. Our key idea is to use off-policy training to achieve these goals, while relaxing the greedy assumption at least in the learning stage. Although off-policy methods are known to improve the sample efficiency [1], [29], the popular importance sampling methods for off-policy training is problematic as it often involves calculating a product of a series of importance sampling ratios, which is not only computationally costly but has high variance [30] as well. There exist some partial solutions to this issue in literature but only under the single-agent setting [27].

The estimation of the CVF plays a central role in the CTDE framework, as its bias and variance directly affect the performance of the whole system. Some authors propose to use one-step TD to stabilize the training process [27], but it has larger estimation variance than multi-step methods [31]. Foerster et al. adopt a variant of TD(λ) [14] to balance the bias and variance in CVF estimation, but they use an on-policy training method which could be sample inefficient. Precup et al. propose an importance sampling-based TD(λ) method in the single-agent setting and prove the convergence property with linear function approximation [32].

It is worth mentioning that in practice, however, off-policy correction is not always needed in off-policy learning, especially when the behavior policy and target policy are close to each other. For example, Hernandez et al. find that it is possible to ignore off-policy correction over off-policy Sarsa [23] and $Q(\sigma)$ [33] without seeing an adverse effect on the overall performance [34]. Unfortunately, the authors fail to analyze the theoretical property behind this phenomenon. Fujimoto et al. show that the off-policy experiences generated during the interaction with the environment tend to be heavily correlated to the current policy, and their experimental results also reveal that the distribution of off-policy data during the training procedure is very close to that of the current policy [35]. Their analysis provides an intuitive explanation for why performance can be improved even without off-policy correction. In Section V, we will provide further theoretical analyze on this interesting issue.

Finally, there have been several attempts on the StarCraft Multi-Agent Challenge (SMAC) [24], including [14], [15], [27]. The results of [14] are the published state-of-the-art in value-based methods and [14] in actor-critic-based methods.

III. Background

A. Problem Formulation

The cooperative multi-agent task we considered can be described as a variant of Dec-POMDP [36]. Specifically, this task can be defined as a tuple: $\mathcal{G} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \mathcal{Z}, \mathcal{O}, N, \gamma \rangle$, where $s \in \mathcal{S}$ denotes the true state of the environment, $\mathcal{A}$ is the action set for each of N agents, and $\gamma \in [0, 1]$ is the discount factor. At each timestep, each agent $i \in \{1, 2, \cdots, N\}$ chooses an action $a^i \in \mathcal{A}$, forming a joint action $\mathbf{a} = \{a^1, a^2, \cdots, a^N\} \in \mathcal{A}^N$. Then the environment gets into next state s' through a dynamic transition function $\mathcal{P}(s'|s,\mathbf{a}) : \mathcal{S} \times \mathcal{A}^N \times \mathcal{S} \mapsto [0,1]$. All agents share the same reward function $r(s,\mathbf{a}) : \mathcal{S} \times \mathcal{A}^N \mapsto \mathbb{R}$. We consider a partial observable scenario[1] in which each agent draws partial observation $o \in \mathcal{O}$ from the observation function $\mathcal{Z}(s,i) : \mathcal{S} \times N \mapsto \mathcal{O}$. Each agent i also has an observation-action history $\tau^i \in \mathcal{T} \equiv (\mathcal{O} \times \mathcal{A})^*$, on which it conditions a stochastic policy. A stochastic policy is a mapping defined as $\pi(a|\tau) : \mathcal{T} \times \mathcal{A} \mapsto [0,1]$.

In the training phase of the CTDE paradigm, a centralized action-value function $Q([s,\boldsymbol{\tau}],\mathbf{a})$ (or simply expressed as $Q(\boldsymbol{\tau},\mathbf{a})$) is learned from the local observation history of all agents (denoted as $\boldsymbol{\tau} = \{\tau^1, \tau^2, \cdots, \tau^N\}$) and the global state (denoted as s), while during the execution phase, each agent's policy π^i only relies on its own observation-action history τ^i. The agents aim to learn a policy that maximize their expected discounted returns $\mathbb{E}_{\mathbf{a}\in\boldsymbol{\pi}, s\in\mathcal{S}}\left[\sum_{t=0}^{\infty} \gamma^t r(s,\mathbf{a})\right]$. To simplify notation, we denote joint quantities over agents in bold. We also omit the index i of each agent when there is no ambiguity in the following sections.

B. Hypothesis Space for Centralized Value Functions

The hypothesis space (or hypothesis set) $\mathcal{H}$ is a space of all possible hypotheses for mapping inputs to outputs that can be searched [37], [38]. To learn a stable and generalizable CVF, choosing a suitable hypothesis space is of importance, which is not only related to the characteristic of the problem domain but related to how the learned system is deployed as well. In particular, in multi-agent systems, the joint action space of all agents increases exponentially with the increase of the number of agents, implying that the hypothesis space of CVF should be large enough to account for such complexity. Furthermore, to facilitate the freedom of each agent to make decision based on its local observations without consulting the CVF, the following centralized greedy behavior (CGB) assumption is generally adopted:

$$\underset{\mathbf{a}}{\operatorname{argmax}}\ Q_{tot}(\boldsymbol{\tau},\mathbf{a}) = \begin{pmatrix} \underset{a^1}{\operatorname{argmax}}\, Q^1\left(\tau^1, a^1\right) \\ \vdots \\ \underset{a^N}{\operatorname{argmax}}\, Q^N\left(\tau^N, a^N\right) \end{pmatrix}. \tag{1}$$

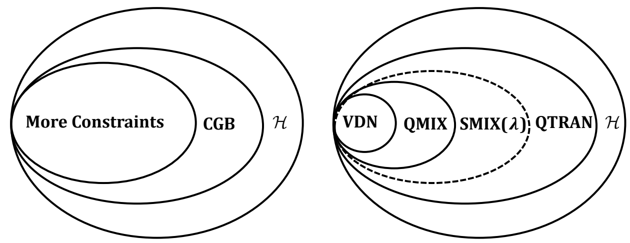
This property establishes a structural constraint between the centralized value function and the decentralized value functions, which can be thought of as a simplified credit assignment mechanism during the execution phase. Figure 1a illustrates how the structural constraints reduce the effective size of the hypothesis space.

C. VDN, QMIX and QTRAN

One sufficient condition for (1) is the following non-negative linear combination:

$$Q_{tot}(\boldsymbol{\tau},\mathbf{a};\theta) = \sum_{i=1}^{N} \alpha_i Q^i(\tau^i, a^i; \theta^i), \alpha_i \geq 0, \tag{2}$$

[1]In standard Dec-POMDP, the observation function $\mathcal{Z}(\mathbf{o}|\mathbf{a}, s')$ denotes the probability of the observing joint observation $\mathbf{o}$ given that joint action $\mathbf{a}$ was taken and led to state s' (cf., [36]).



(a) (b)

Fig. 1. (a) The size of hypothesis space corresponding to different constraints. (b) The relationship of hypothesis spaces in several different algorithms. See subsection IV-C for more discussions about the hypothesis space of the proposed SMIX(λ).

where Q_{tot} is the centralized Q function and Q^i is the Q function for each agent i. In VDN [20], all the combination coefficients $\alpha_i, i = 1, 2, \cdots, N$ are set to 1. QMIX [15] extends this additive value factorization to a more general case by enforcing $\frac{\partial Q_{tot}}{\partial Q^i} \geq 0, i \in \{1, \cdots, N\}$. The following theorem explicitly gives the consequential structural constraints imposed on the CVF hypothesis space due to QMIX,

Theorem 1. *For QMIX, if $\frac{\partial Q_{tot}}{\partial Q^i} \geq 0$ for $i \in \{1, 2, \cdots, N\}$, then we have*

$$\max_{\mathbf{a}} Q_{tot}(\boldsymbol{\tau},\mathbf{a}) = Q_{tot}\left(\boldsymbol{\tau}, \underset{a^1}{\operatorname{argmax}}\, Q^1(\tau^1, a^1), \cdots, \underset{a^N}{\operatorname{argmax}}\, Q^N(\tau^N, a^N)\right). \tag{3}$$

Proof:
For QMIX, we have

$$Q_{tot}(\boldsymbol{\tau},\mathbf{a}) := \hat{Q}\left(Q_1(\tau^1, a^1), \ldots, Q^N(\tau^N, a^N)\right),$$

where $\hat{Q}$ is the *mixing network* and $\mathbf{a} = (a_1, \ldots, a_N)$. Similarly, we have $\frac{\partial \hat{Q}}{\partial Q^i} \geq 0$ and

$$Q_{tot}\left(\tau, \operatorname{argmax}_{a^1} Q^1(\tau^1, a^1), \cdots, \operatorname{argmax}_{a^N} Q^N(\tau^N, a^N)\right) := \hat{Q}\left(\max_{a^1} Q^1(\tau^1, a^1), \ldots, \max_{a^n} Q^N(\tau^N, a^N)\right).$$

Since $\frac{\partial \hat{Q}}{\partial Q^i} \geq 0$, given $(\bar{a}^2, \ldots, \bar{a}^n)$, we have

$$\hat{Q}\left(Q^1(\tau^1, a^1), Q^2(\tau^2, \bar{a}^2), \ldots, Q^N(\tau^N, \bar{a}^N)\right) \leq \hat{Q}\left(\max_{a^1} Q^1(\tau^1, a^1), Q^2(\tau^2, \bar{a}^2), \ldots, Q^N(\tau^N, \bar{a}^N)\right)$$

for any a^1. Similarly, given $(\bar{a}^1, \ldots, \bar{a}^{k-1}, \bar{a}^{k+1}, \ldots, \bar{a}^N)$, we have

$$\hat{Q}\left(Q_1(\tau^1, \bar{a}^1), \ldots, Q^k(\tau^k, a^k), \ldots, Q^N(\tau^N, \bar{a}^N)\right) \leq \hat{Q}\left(Q^1(\tau^1, \bar{a}^1), \ldots, \max_{a^k} Q^k(\tau^k, a^k), \ldots, Q^N(\tau^N, \bar{a}^N)\right)$$

for any a^k. Finally, for any $(a^1, \ldots, a^N)$, we have

$$\begin{aligned} &\hat{Q}\left(Q^1(\tau^1, a^1), \ldots, Q^N(\tau^N, a^N)\right) \\ \leq &\hat{Q}\left(\max_{a^1} Q_1(\tau^1, a^1), \ldots, Q^N(\tau^N, a^N)\right) \\ \leq &\hat{Q}\left(\max_{a^1} Q^1(\tau^1, a^1), \ldots, \max_{a^N} Q^N(\tau^N, a^N)\right). \end{aligned}$$

Therefore, we obtain

$$\max_{a^1,\ldots,a^N} \hat{Q}\left(Q^1(\tau^1,a^1),\ldots,Q^N(\tau^n,a^n)\right)$$
$$= Q\left(\max_{a^1} Q^1(\tau^1,a^1),\ldots,\max_{a^N} Q_n(\tau^N,a^N)\right),$$

which is the specific form of (3) for QMIX. ■

Note that the QMIX algorithm relies on this result to simplify its optimization procedure. So does VDN, as it can be considered as a simplification of the QMIX algorithm. QTRAN [21] further relaxes the constraints of VDN and QMIX and works in a larger hypothesis space structured by a sufficient and necessary condition of (1), but at the cost of having to optimize the joint value function in the whole action space formed by all the agents, which is computationally challenging even in the case of small number of agents. Although a coordinate decent-type method is proposed in QTRAN to address the issue, the method's scalability and range of practical use can be limited by this.

Finally, before ending this section, we give an illustration of the relationship among hypothesis spaces for VDN, QMIX, QTRAN and the proposed SMIX(λ) method in Figure 1b.

IV. Methods

In this section, we give the details of the proposed SMIX(λ) method, which is a SARSA(λ) [39] style off-policy method that aims at learning a centralized value function within the framework of CTDE for better MARL.

A. Relaxing the CGB Assumption in Learning

Recall that in a standard CTDE approach, a centralized Q_{tot} function or critic function for all agents is first trained, whose value is then assigned to the individual agent to guide the training process of each agent. A typical implementation of this idea is the QMIX [15], in which the centralized Q_{tot} function is learned through a traditional Q learning algorithm. However, due to the high dimensionality of the joint action space, taking the $\max$ of $Q_{tot}(\boldsymbol{\tau},\mathbf{a})$ w.r.t. $\mathbf{a}$ required by Q learning updates could be untractable. To address this issue, the aforementioned CGB assumption is explicitly followed although it is seemingly unrealistic.

Note that this greedy assumption is not only followed by the Q-learning algorithm in its updating rule, but followed by the classic n-step Q-learning method and Watkins's $Q(\lambda)$ method [40] as well. Hence to remove our exact relying on this condition in the learning phase, it is necessary to abandon such updating methods at all when learning the centralized value function network.

Alternatively, one can use a SARSA [23]-based method instead of Q-learning, where a Bellman expectation backup operator is applied to learn the Q_{tot} function. However, it is an on-policy method and only considers the one-step return. In what follows, we will extend this method to an off-policy setting and integrate it with multi-step returns to handle the non-Markovian environments.

B. Off-Policy Learning without Importance Sampling

One way to alleviate the curse of dimensionality issue of joint action space and to improve exploration is the off-policy learning. Denoting the behavior policy as $\boldsymbol{\mu}$ and the target policy as $\boldsymbol{\pi}$, a general off-policy strategy to evaluate the Q value function for $\boldsymbol{\pi}$ using data $\boldsymbol{\tau}$ generated by following $\boldsymbol{\mu}$ can be expressed as follows [30],

$$Q(\boldsymbol{\tau},\mathbf{a}) \leftarrow Q(\boldsymbol{\tau},\mathbf{a}) + \mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{t\geq 0}\gamma^t\left(\prod_{i=1}^{t}\rho_i\right)\delta_t^{\boldsymbol{\pi}}\right], \tag{4}$$

where each ρ_i is a non-negative coefficient and satisfies $\prod_{i=1}^{t}\rho_i = 1$ when $t=0$. The error term $\delta_t^{\boldsymbol{\pi}}$ is generally written as the following expected TD-error,

$$\delta_t^{\boldsymbol{\pi}} = r_{t+1} + \gamma\mathbb{E}_{\boldsymbol{\pi}}Q\left(\boldsymbol{\tau}_{t+1},\cdot\right) - Q\left(\boldsymbol{\tau}_t,\mathbf{a}_t\right), \tag{5}$$

where $\mathbb{E}_{\boldsymbol{\pi}}Q(\boldsymbol{\tau},\cdot) = \sum_{\mathbf{a}}\boldsymbol{\pi}(\mathbf{a}|\boldsymbol{\tau})Q(\boldsymbol{\tau},\mathbf{a})$.[2] In particular, for the importance sampling (IS) method, each ρ_i in (4) is defined as the relative probability of their trajectories occurring under the target policy $\boldsymbol{\pi}$ and behavior policy $\boldsymbol{\mu}$, also called importance sampling ratio, i.e., $\rho_i = \frac{\boldsymbol{\pi}(\mathbf{a}_i|\boldsymbol{\tau}_i)}{\boldsymbol{\mu}(\mathbf{a}_i|\boldsymbol{\tau}_i)}$.

Despite its theoretical soundness, the importance sampling (IS) method faces great challenges under the setting of multi-agent environments: 1) it suffers from large variance due to the product of the ratio [43], and 2) the "curse of dimensionality" issue of the joint action space makes it impractical to calculate the $\boldsymbol{\pi}(\mathbf{a}_i|\boldsymbol{\tau}_i)$ even for a single timestep i, when the number of agents is large. Previously, Foerester et al. proposed a method that effectively addresses the first issue by avoiding calculating the product over the trajectories [27], but how to solve the second one remains open.

The above analysis highlights the need for exploring alternative approaches that can perform off-policy learning without importance sampling in multi-agent settings.

C. The SMIX(λ) Method

To achieve the above goal, the key idea of SMIX(λ) is to further simplify the coefficient ρ_i in (4), so as to reduce the variance of the importance sampling estimator and to potentially bypass the curse of dimensionality involved in calculating $\boldsymbol{\pi}(\cdot|\boldsymbol{\tau})$.

Specifically, we relax each coefficient $\rho_i = 1.0$ in (4) based on the discussions in subsection IV-B and use an experience replay memory to store the most recent off-policy data.

Then, we use the λ-return [23] as the TD target estimator, which is defined as follows:

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}, \tag{6}$$

where $G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n\mathbb{E}_{\boldsymbol{\pi}}Q\left(\boldsymbol{\tau}_{t+n},\mathbf{a}_{t+n};\theta^-\right)$ is the n-step return and θ^- are the parameters of the target

[2]The policy evaluation strategy of many popular methods can be expressed as (4), including SARSA(λ) [39], off-policy importance sampling methods [32], off-policy $Q(\lambda)$ method [41], tree-backup method, TB(λ) [42] and Retrace(λ) [30]. These methods differ in the definition of the coefficient ρ_i and error term $\delta_t^{\boldsymbol{\pi}}$ [30], [41].

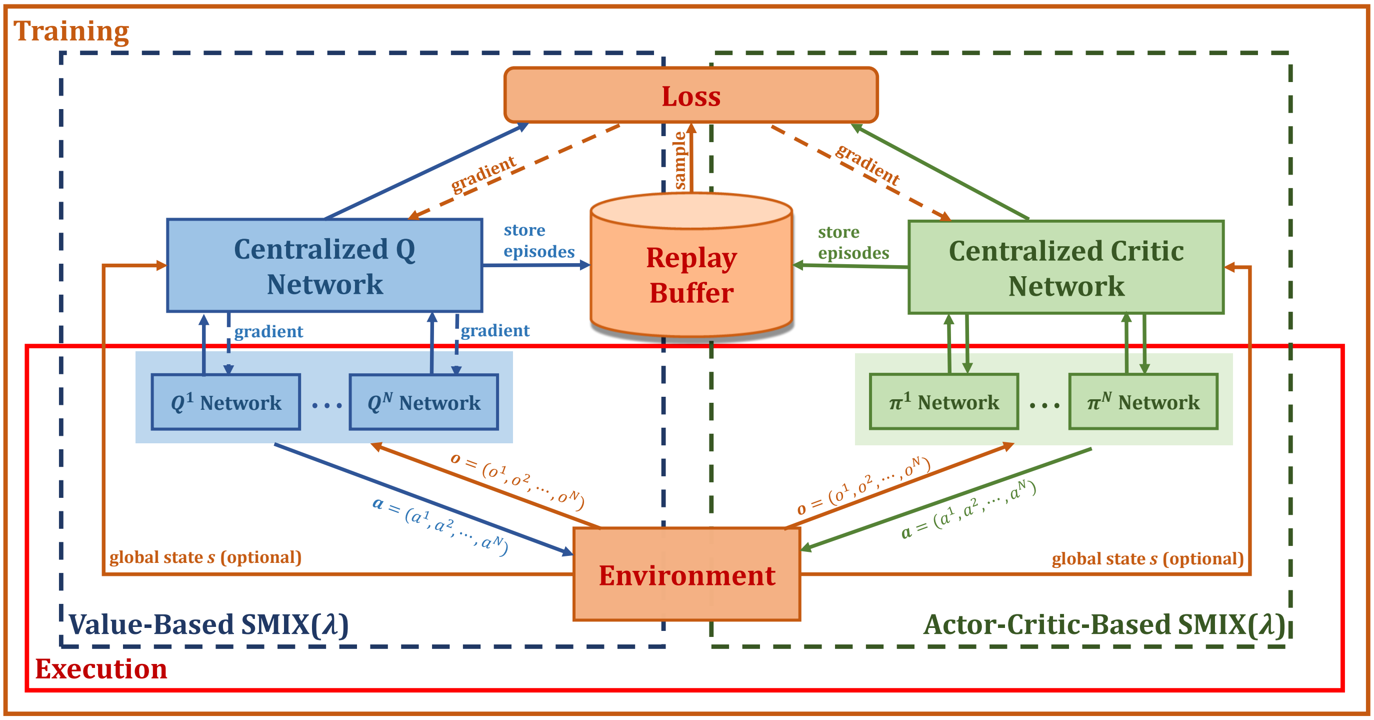


Fig. 2. Value-based SMIX(λ) and actor-critic-based SMIX(λ) architecture.

network. Plugging this into (4) and setting $\rho_i = 1.0$ for all i, we have (the update step-size α is omitted for simplification),

$$Q(\boldsymbol{\tau}, \mathbf{a}) \leftarrow Q(\boldsymbol{\tau}, \mathbf{a}) + \mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{t \geq 0} \gamma^t (G_t^\lambda - Q(\boldsymbol{\tau}_t, \mathbf{a}_t))\right]. \quad (7)$$

In implementation, SMIX(λ) is trained end-to-end and the loss function for the centralized value function $Q_{tot}^{\boldsymbol{\pi}}$ has the following form:

$$\mathcal{L}_t(\theta) = \sum_{i=1}^{N_b} \left[(y_i^{tot} - Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \mathbf{a}; \theta))^2\right], \quad (8)$$

where $y_i^{tot} = G_t^\lambda$ is defined in (6) and is estimated through experience replaying, N_b is the batch size.

The QMIX [15] structure is adopted as the basic deep network architecture for the proposed SMIX(λ). Each agent i has its own decentralized $Q^i(\tau^i, a^i)$ network composed of GRU [44] modules. Then all the individual Q^i values are passed into a mixing network to calculate the joint action value $Q_{\text{tot}}^{\boldsymbol{\pi}}$. The weight of the mixing network is generated by hypernetworks using the global state s. All the neural networks are trained end-to-end and the centralized value function $Q_{\text{tot}}^{\boldsymbol{\pi}}$ is updated by minimizing (8).

In the training phase, we remove the explicit structural constraints between centralized value function and decentralized policies by abandoning the Q-learning updating rules, which makes SMIX(λ) optimize in a larger hypothesis space than QMIX [15]. While in the implementation, SMIX(λ) requires that all the weights in the mixing network be non-negative, which is a sufficient condition of the CGB assumption. Thus, the hypothesis space of SMIX(λ) is smaller than QTRAN (cf. Figure 1b). However, the sample complexity of our method is much better than the QTRAN method, as the latter has to estimate an expectation over the high dimensional joint action space of all agents, which is computationally challenging even in the case of a small number of agents. The general training procedure for SMIX(λ) is provided in Algorithm 1.

It is worth noting that our method of training a centralized value function is a general method and can be easily applied to other CTDE methods, include value-based methods (such as VDN [20]), actor-critic-based methods (e.g. COMA [14]), and even fully decentralized methods (e.g. IQL [45]). The overall architecture of these generalized versions of SMIX(λ) is given in Figure 2. The left and right dashed boxes show the value-based SMIX(λ) and actor-critic based SMIX(λ) algorithms, and the two solid boxes represent the modules in the centralized training and decentralized execution respectively. Each agent's Q network or π network only has access to its own observation (and its own observation history), and the centralized Q network or centralized critic network aggregates all agents' decentralized information.

V. Analysis

In this section, we give the convergence analysis of the proposed SMIX(λ) algorithm, by first building the connection between SMIX(λ) and a previous method named $Q(\lambda)$ [41], originally proposed for off-policy value function evaluation under single-agent settings.

Denoting $G^{\boldsymbol{\pi}}$ as the λ-return estimator (cf., 6) for the action value of the target policy $\boldsymbol{\pi}$, the goal of an off-policy method

Algorithm 1 Training Procedure for SMIX(λ)

Initialize the behavior network with parameters θ, the target network with parameters θ^-, empty replay buffer $\mathcal{D}$ to capacity $N_{\mathcal{D}}$, training batch size N_b
for each training episode **do**
  for each episode **do**
    for $t = 1$ to $T-1$ **do**
      Obtain the partial observation $\mathbf{o}_t = \{o_t^1, \cdots, o_t^N\}$ for all agents and global state s_t
      Select action a_t^i according to ϵ-greedy policy w.r.t agent i's decentralized value function Q^i for $i = 1, \cdots, N$
      Execute joint action $\mathbf{a}_t = \{a^1, a^2, \cdots, a^N\}$ in the environment
      Obtain the global reward r_{t+1}, the next partial observation o_{t+1}^i for each agent i and next global state s_{t+1}
    end for
    Store the episode in $\mathcal{D}$, replacing the oldest episode if $|\mathcal{D}| \geq N_{\mathcal{D}}$
  end for
  Sample a batch of N_b episodes $\sim$ Uniform($\mathcal{D}$)
  Calculate λ-return targets y_i^{tot} according to (6) using θ^- for each timestep
  Update θ by minimizing $\sum_{t=1}^{T-1} \sum_{i=1}^{N_b} \left[(y_i^{tot} - Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \mathbf{a}; \theta))^2\right]$
  Replace target parameters $\theta^- \leftarrow \theta$ every C episodes
end for

is to use the data from the behavior policy $\boldsymbol{\mu}$ to correct $G^{\boldsymbol{\pi}}$, in a way such that the following criterion is met,

$$\mathbb{E}_{\boldsymbol{\pi}}\Big[G^{\boldsymbol{\pi}}\Big] = \mathbb{E}_{\boldsymbol{\mu}}\Big[G^{\boldsymbol{\mu},\boldsymbol{\pi}}\Big], \tag{9}$$

where $G^{\boldsymbol{\mu},\boldsymbol{\pi}}$ is the corrected return of off-policy data.

The most commonly used method for calculating the $G^{\boldsymbol{\mu},\boldsymbol{\pi}}$ is the importance sampling (IS) method which multiply each reward with a weighted term to satisfy (9). Indeed, the motivation behind SMIX(λ) is to simplify the IS method so that it can be used in multi-agent settings. If we define the IS ratio at timestep t as: $\rho_t = \frac{\boldsymbol{\pi}(\mathbf{a}_t|\boldsymbol{\tau}_t)}{\boldsymbol{\mu}(\mathbf{a}_t|\boldsymbol{\tau}_t)}$, then the n-step return using IS can be defined as:

$$\begin{aligned} G_t^{(n)} =& r_{t+1} + \gamma\rho_{t+1}r_{t+2} + \cdots \\ &+ \gamma^{n-1}\rho_{t+1}\cdots\rho_{t+n-1}r_{t+n} \\ &+ \gamma^{n}\rho_{t+1}\cdots\rho_{t+n}\mathbb{E}_{\boldsymbol{\pi}}Q^{\text{SMIX}}(\boldsymbol{\tau}_{t+n}, \cdot). \end{aligned} \tag{10}$$

Thus, we have the following form of $G^{\boldsymbol{\mu},\boldsymbol{\pi}}$:

$$G^{\boldsymbol{\mu},\boldsymbol{\pi}} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}. \tag{11}$$

Plugging (10) into (11), we have:

$$\begin{aligned} &G^{\boldsymbol{\mu},\boldsymbol{\pi}} \leftarrow Q^{\text{SMIX}}(\boldsymbol{\tau}_t, \mathbf{a}_t) + \sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\gamma\lambda\rho_i\right)\delta_k^{\boldsymbol{\pi}}, \\ &\delta_k^{\boldsymbol{\pi}} = \left(r_{k+1} + \gamma\rho_{k+1}Q^{\text{SMIX}}(\boldsymbol{\tau}_{k+1}, \mathbf{a}_{k+1}) - Q^{\text{SMIX}}(\boldsymbol{\tau}_k, \mathbf{a}_k)\right). \end{aligned} \tag{12}$$

In SMIX(λ), all the importance sampling factor are relaxed to 1.0, then corresponding to (4), the update rule of SMIX(λ) can be expressed as[3],

$$\begin{aligned} &Q^{\text{SMIX}}(\boldsymbol{\tau}_t, \mathbf{a}_t) \leftarrow Q^{\text{SMIX}}(\boldsymbol{\tau}_t, \mathbf{a}_t) + \mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\lambda\gamma\right)\delta_k^{\boldsymbol{\pi}}\right], \\ &\delta_k^{\boldsymbol{\pi}} = \left(r_{k+1} + \gamma\mathbb{E}_{\boldsymbol{\mu}}Q^{\text{SMIX}}(\boldsymbol{\tau}_{k+1}, \cdot) - Q^{\text{SMIX}}(\boldsymbol{\tau}_k, \mathbf{a}_k)\right). \end{aligned} \tag{13}$$

[3]We consider the expected form of $Q^{\text{SMIX}}(\boldsymbol{\tau}_{k+1}, \mathbf{a}_{k+1})$ in (12) and the training data is sampled from a replay buffer.

In contrast with the multiplicative operation for off-policy learning, an additive-type operation is used in $Q(\lambda)$ [41]. In particular, an additive correction term $\Delta_r^{\boldsymbol{\mu},\boldsymbol{\pi}}$, is added to each reward when calculating $G^{\boldsymbol{\mu},\boldsymbol{\pi}}$ in (9) and get:

$$\begin{aligned} G_t^{(n)} &= (r_{t+1} + \Delta_{r_{t+1}}^{\boldsymbol{\mu},\boldsymbol{\pi}}) + \cdots + \gamma^{n-1}(r_{t+n} + \Delta_{r_{t+n}}^{\boldsymbol{\mu},\boldsymbol{\pi}}) \\ &\quad + \gamma^n\mathbb{E}_{\boldsymbol{\pi}}Q^{Q(\lambda)}(\boldsymbol{\tau}_{t+n}, \cdot). \end{aligned} \tag{14}$$

The major advantage of this additive off-policy correction is that there is no product of the ratio and no the joint policy $\boldsymbol{\pi}(\mathbf{a}|\boldsymbol{\tau})$ involved, hence completely bypassing the limitations of the IS method[4]. Specifically, the updating rule of $Q(\lambda)$ method is [41]:

$$\begin{aligned} &Q^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t) \leftarrow Q^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t) + \mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\lambda\gamma\right)\hat{\delta}_k^{\boldsymbol{\pi}}\right], \\ &\hat{\delta}_k^{\boldsymbol{\pi}} = \left(r_{k+1} + \Delta^{\boldsymbol{\mu},\boldsymbol{\pi}}r_{k+1}\right), \\ &\Delta^{\boldsymbol{\mu},\boldsymbol{\pi}}r_{k+1} = \gamma\mathbb{E}_{\boldsymbol{\pi}}Q^{Q(\lambda)}(\boldsymbol{\tau}_{k+1}, \cdot) - Q^{Q(\lambda)}(\boldsymbol{\tau}_k, \mathbf{a}_k). \end{aligned} \tag{15}$$

By comparing (13) and (15), we see that our SMIX(λ) and off-policy $Q(\lambda)$ are essentially equivalent except that SMIX(λ) calculates $\mathbb{E}_{\boldsymbol{\mu}}Q^{\text{SMIX}}(\boldsymbol{\tau}_{k+1}, \cdot)$ in $\delta_k^{\boldsymbol{\pi}}$ while $Q(\lambda)$ calculate $\mathbb{E}_{\boldsymbol{\pi}}Q^{Q(\lambda)}(\boldsymbol{\tau}_{k+1}, \cdot)$ in $\hat{\delta}_k^{\boldsymbol{\pi}}$. Note that SMIX($\lambda$) is a biased estimation of $Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \mathbf{a})$, while $Q(\lambda)$ unbiased.

The following theorem states that when $\boldsymbol{\pi}$ and $\boldsymbol{\mu}$ are sufficiently close, the difference between the output of SMIX(λ) and $Q(\lambda)$ is bounded. This implies that SMIX(λ) is consistent with the $Q(\lambda)$ algorithm.

Theorem 2. *Suppose we update the value function from* $Q_n^{SMIX}(\boldsymbol{\tau}_t, \mathbf{a}_t) = Q_n^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t)$*, where* n *represents the* n*-th update. Let* $\epsilon = \max_{\boldsymbol{\tau}}\|\boldsymbol{\pi}(\cdot|\boldsymbol{\tau}) - \boldsymbol{\mu}(\cdot|\boldsymbol{\tau})\|_1$, $M = \max_{\boldsymbol{\tau},\mathbf{a}}|Q_n^{Q(\lambda)}(\boldsymbol{\tau}, \mathbf{a})|$*. Then, the error between* $Q_{n+1}^{SMIX}(\boldsymbol{\tau}_t, \mathbf{a}_t)$ *and* $Q_{n+1}^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t)$ *can be bounded by the expression:*

$$|Q_{n+1}^{SMIX}(\boldsymbol{\tau}_t, \mathbf{a}_t) - Q_{n+1}^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t)| \leq \frac{\epsilon\gamma}{1-\lambda\gamma}M. \tag{16}$$

[4]But under the condition that the behavior policy $\boldsymbol{\mu}$ should be close to the target policy $\boldsymbol{\pi}$, which under our experience replay setting should not be a problem (cf., [35]).

Proof: First, we have,

$$\begin{aligned}\left|\delta_t^{\boldsymbol{\pi}} - \hat{\delta}_t^{\boldsymbol{\pi}}\right| &= \left|\gamma \mathbb{E}_{\boldsymbol{\mu}} Q_n^{\text{SMIX}}\left(\boldsymbol{\tau}_{t+1}, \cdot\right) - \gamma \mathbb{E}_{\boldsymbol{\pi}} Q_n^{Q(\lambda)}\left(\boldsymbol{\tau}_{t+1}, \cdot\right)\right| \\ &= \gamma \left|\sum_{\mathbf{a}} \boldsymbol{\mu}(\mathbf{a}|\boldsymbol{\tau}_{t+1}) Q_n^{\text{SMIX}}\left(\boldsymbol{\tau}_{t+1}, \cdot\right) - \right. \\ &\qquad \left. \sum_{\mathbf{a}} \boldsymbol{\pi}(\mathbf{a}|\boldsymbol{\tau}_{t+1}) Q_n^{Q(\lambda)}\left(\boldsymbol{\tau}_{t+1}, \cdot\right)\right| \\ &\leq \gamma \epsilon M.\end{aligned}$$

Thus,

$$\begin{aligned}&\left|Q_{n+1}^{\text{SMIX}}\left(\boldsymbol{\tau}_t, \mathbf{a}_t\right) - Q_{n+1}^{Q(\lambda)}\left(\boldsymbol{\tau}_t, \mathbf{a}_t\right)\right| \\ &= \left|\mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k} \lambda\gamma\right) \delta_k^{\boldsymbol{\pi}}\right] - \mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k} \lambda\gamma\right) \hat{\delta}_k^{\boldsymbol{\pi}}\right]\right| \\ &= \left|\mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k} \lambda\gamma\right)\left(\delta_k^{\boldsymbol{\pi}} - \hat{\delta}_k^{\boldsymbol{\pi}}\right)\right]\right| \\ &\leq \left|\mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k} \lambda\gamma\right)(\gamma\epsilon M)\right]\right| \\ &\leq \mathbb{E}_{\boldsymbol{\mu}}\left[\frac{1}{1-\lambda\gamma}(\gamma\epsilon M)\right] = \frac{\epsilon\gamma}{1-\lambda\gamma} M.\end{aligned}$$

Therefore, the expression (16) holds. ■

This theorem indicates that SMIX(λ) has the similar convergence property to $Q(\lambda)$ under some mild conditions. The following theorem presents the convergence property of the $Q(\lambda)$ method [41].

Theorem 3. *[41] Consider the sequence of Q-functions computed according to (15) with fixed policy $\boldsymbol{\pi}$ and $\boldsymbol{\mu}$. Let $\epsilon = \max_{\boldsymbol{\tau}} \|\boldsymbol{\pi}(\cdot|\boldsymbol{\tau}) - \boldsymbol{\mu}(\cdot|\boldsymbol{\tau})\|_1$. If $\lambda\epsilon < \frac{1-\gamma}{\gamma}$, then under the following conditions:*

- $\sum_{t \geq 0} \mathbb{P}\{\boldsymbol{\tau}_t, \mathbf{a}_t = \boldsymbol{\tau}, \mathbf{a}\} \geq D > 0.$, *where $\mathbb{P}(\boldsymbol{\tau}, \mathbf{a})$ represents the visit frequency,*
- $\mathbb{E}_{\boldsymbol{\mu}_n} T_n^2 < \infty$, *where T_n is the length of $\boldsymbol{\tau}_n$,*
- $\sum_{n \geq 0} \alpha_n(\boldsymbol{\tau}, \mathbf{a}) = \infty, \sum_{n \geq 0} \alpha_n^2(\boldsymbol{\tau}, \mathbf{a}) < \infty$, *where α_n is the step-size of the n-th iteration,*

we have, almost surely:

$$\lim_{n \to \infty} Q_n^{Q(\lambda)}(\boldsymbol{\tau}, \mathbf{a}) = Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \mathbf{a}).$$

By Theorem 2 and Theorem 3, we know that SMIX(λ) has convergence guarantee to current policy's value function $Q^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \mathbf{a})$ if $\boldsymbol{\pi}$ and $\boldsymbol{\mu}$ are sufficiently close. This could mean a lot to a multi-agent reinforcement learning algorithm, as it bypasses major drawbacks of importance sampling.

The above analysis shows that SMIX(λ) and $Q(\lambda)$ have similarities both formally and analytically. However, when applying them to the problem of multi-agent reinforcement learning, their computational complexity is fundamentally different. This is because to calculate the additive error correction term, $Q(\lambda)$ has to estimate the expectation over target policy $\boldsymbol{\pi}$ in (15), but this is unrealistic in the multi-agent setting since the dimension of the joint action space grows exponentially with the number of agents. By contrast, the SMIX(λ) relies on the experience replay technique to compute the expectation in (13), whose computational complexity grows only linearly with the number of training samples, regardless of the size of joint action space and the number of agents involved. Such scalability makes our method more appropriate for the task of multi-agent reinforcement learning, compared to the $Q(\lambda)$ and QTRAN (which suffers from the same problem as $Q(\lambda)$).

Finally, before ending this section, we summarize some of the key characteristics of QMIX and SMIX(λ) in Table I.

TABLE I
THE COMPARISON OF QMIX AND SMIX(λ).

Property	**QMIX**	**SMIX(λ)**
Constraint in the learning phase	Centralized greedy assumption	No assumption
Uses experience replay	✓	✓
Handles non-Markovian domains	✗	✓
Uses λ-return	✗	✓
Stable point of convergence	Q^*	$Q^{\boldsymbol{\pi}}$

VI. EXPERIMENTS

In this section, we first describe the environmental setup and the implementation details of our method. Then we give the experimental results and ablation study.

A. Environmental Setup

We evaluate our SMIX(λ) in the StarCraft Multi-Agent Challenge (SMAC) [24] environment. The SMAC is chosen as our testbed mainly because of the following two reasons: (1) SMAC provides a set of rich cooperative scenarios that challenge algorithms to handle significant partial observability and credit assignment problem [14]. These problems bring a great challenge for centralized value function estimation. (2) SMAC also provides an open-source Python-based implementation of several key algorithms, which allows for fair comparisons between different methods.

SMAC is based on the popular real-time strategy (RTS) game StarCraft II[5]. Each unit can be seen as an individual agent which has a complex set of micro-actions. Different from the full StarCraft II game, SMAC focuses on fully cooperative, decentralized *micromanagement* multi-agent problems. *Micromanagement* means the task of controlling individual or grouped units to fight enemy units. While high-level strategies such as economy and resource management, known as *macromanagement*, are not considered in SMAC.

SMAC provides several challenging micro scenarios that aim to evaluate different aspects of cooperative behaviors of a group of agents. In each scenario, two groups of agents are placed on the map with random initial positions within groups at the beginning of each episode. Each agent can only receive local observations within its *sight range*, which introduces significant partial observability. Extra global state information is available during centralized training. The units of the first group (allied units) are controlled by decentralized agents, while the units of the other group (enemy units) are controlled by built-in heuristic game AI bot with difficulty

[5]StarCraft II is a trademark of Blizzard Entertainment™.

TABLE II
THE SCENARIOS CONSIDERED IN OUR EXPERIMENTS.

Name	Ally Units	Enemy Units	Type
3m	3 Marines	3 Marines	homogeneous & symmetric
8m	8 Marines	8 Marines	homogeneous & symmetric
2s3z	2 Stalkers & 3 Zealots	2 Stalkers & 3 Zealots	heterogeneous & symmetric
3s5z	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots	heterogeneous & symmetric
2m_vs_1z	2 Marines	1 Zealot	asymmetric
2s_vs_1sc	2 Stalkers	1 Spine Crawler	asymmetric
3s_vs_3z	3 Stalkers	3 Zealots	asymmetric
1c3s5z	1 Colossi, 5 Stalkers & 5 Zealots	1 Colossi, 5 Stalkers &5 Zealots	heterogeneous & symmetric
MMM	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 2 Marauders & 7 Marines	heterogeneous & symmetric

ranging from *very easy* to *cheat insane*. In our experiments, we set the difficulty of the game AI bot to *very difficult* for all experiments. Available actions for each agent contain: `move[direction]`, `attack[enemy id]`, `stop`, and `noop`. Agents receive a joint reward equal to the total damage dealt on the enemy units. We use the default setting for the reward. Refer to [24] for more details.

(a) 3 Staklers vs 5 Zealots

(b) 8 Marines vs 8 Marines

Fig. 3. Screenshots of two SMAC scenarios.

The following 3 types of scenarios are considered in our experiments: (1) *homogeneous and symmetric units*, (2) *heterogeneous and symmetric units*, (3) *asymmetric units*. The list of scenarios considered in our experiment is presented in Table II. Figure 3 shows the screenshots of two SMAC scenarios used in our experiments.

We use *test win rate* as the evaluation metric, which is proposed in [24] and is the default evaluation metric in the SMAC environment. The test win rate is evaluated in the following procedure: the training process is interrupted after every 20,000 timesteps, then 24 independent test episodes are run with each agent performing greedy action selection in a decentralized way. Test win rate refers to the percentage of episodes where the agents defeat all enemy units within the time limit.

B. Implementation Details

The agent network architecture of SMIX(λ) consists of a 64-dimensional GRU [44]. One 64-dimensional fully connected layer with ReLU activation function before GRU is applied for processing the input. The layer after GRU is a fully connected layer of 64 units, which outputs the decentralized state-action values $Q^i(\tau,\cdot)$ of agent i. All agent networks share parameters for reducing the number of parameters to be learned. Thus the agent's one-hot index i is concatenated onto each agent's observations. The agent's previous action is also concatenated to the input.

Based on the basic network architecture of QMIX [15], SMIX(λ) performs the centralized value function estimation with λ-return (λ = 0.8) calculated from a batch of 32 episodes. The batch is sampled uniformly from a replay buffer that stores the most recent 1500 episodes. We run 4 episodes simultaneously. Then we perform training on those fully unrolled episodes. The target network is updated after every 200 training episodes. λ is set to 0.8.

The ϵ-greedy method is used in the training procedure for exploration. ϵ is annealed linearly from 1.0 to 0.05 across the first 50k timesteps for all experiments. The discount factor γ is set to 0.99, and the RMSprop optimizer is used with learning rate $lr = 0.0005$ and $\alpha = 0.99$ without weight decay or momentum during training.

C. Comparative Evaluation

We compare our SMIX(λ) with state-of-the-art algorithms QMIX [15] and COMA [14], which currently perform the best on the SMAC benchmark. VDN [20] and IQL [45] are chosen as baselines for comparisons[6].

The results of all methods in the training process are plotted in Figure 4 and we also provide quantitative comparisons of our methods and their counterparts after training for 1 million steps in Table III. Overall, SMIX(λ) significantly outperforms all the comparison methods in heterogeneous or asymmetric scenarios (i.e., scenarios except 3m and 8m), while performing comparably to them in homogeneous and symmetric scenarios (i.e., 3m and 8m) both in terms of the learning speed and final performance.

In homogeneous and symmetric scenarios such as 3m and 8m, COMA is only slightly faster than SMIX(λ) but underperforms SMIX(λ) in terms of the final performance. In asymmetric (e.g., 3s_vs_3z, 2s_vs_1sc) or heterogeneous

[6] According to [46] and the preliminary experimental results of our own, QTRAN indeed performs poorly on most SMAC maps (nearly 0% win rate). Due to this reason, we currently do not list it for comparison in our experiments.

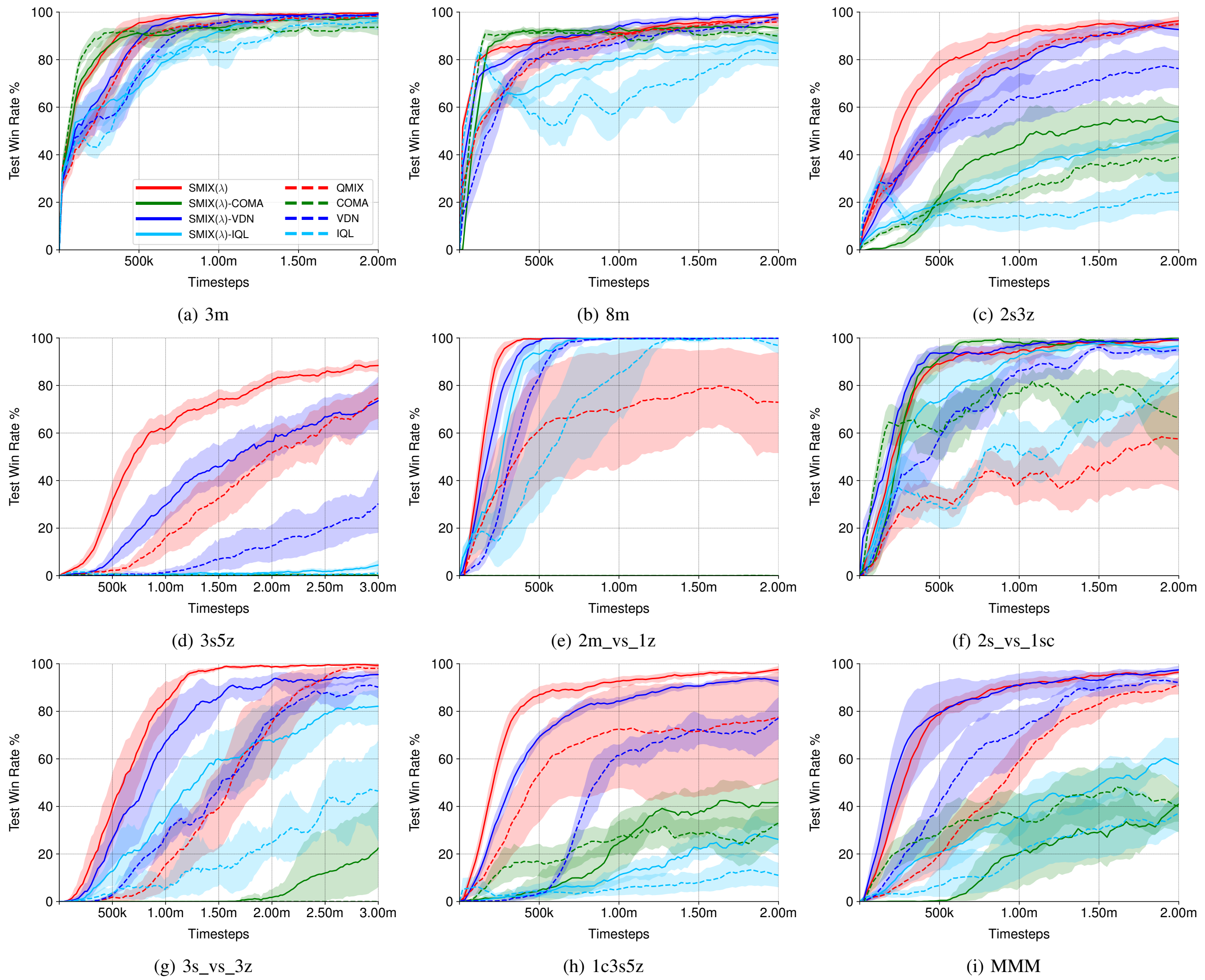


Fig. 4. Test win rates for our methods (SMIX(λ), SMIX(λ)-COMA, SMIX(λ)-VDN, SMIX(λ)-IQL) and comparison methods (QMIX, COMA, VDN, IQL) in nine different scenarios. The performance of our methods and their counterparts are shown with solid and dashed lines of the same color, respectively. The mean and 95% confidence interval are shown across 10 independent runs. The legend in (a) applies across all plots.

(e.g., 2s3z, 3s5z, 1c3s5z) maps, COMA fails to solve these scenarios effectively and the performance of QMIX can be seen as the state-of-the-art. However, the learning speed of SMIX(λ) is almost twice as fast as QMIX. In 3s5z, SMIX(λ) (solid red line) achieves a nearly 90% win rate, while the best comparison method QMIX (dotted red line) achieves about only 70% test win rate. In 2s_vs_1sc, SMIX(λ) also requires less than half the number of samples of QMIX and other comparison methods to reach the asymptotic performance. The largest performance gap can be seen in 3s_vs_3z map (Figure 4g). QMIX needs to be trained for nearly 3 million timesteps to achieve a 100% test win rate, while half of the timesteps are sufficient for SMIX(λ) to achieve the same win rate. In MMM, we can find an interesting result that the VDN can achieve better performance than QMIX (see Figure 4i). This indicates that a simpler network structure can also have enough representative capacity and the reason for VDN's superior performance is that a simpler network architecture only needs a relatively small number of samples for training. Furthermore, by incorporating the proposed centralized training method, VDN's performance can be further improved, which implies that the bottleneck of VDN and QMIX is the bias and variance of the CVF estimation. On the whole, the superior performance of SMIX(λ) using λ-return with off-policy episodes presents a clear benefit over the one-step estimation of QMIX.

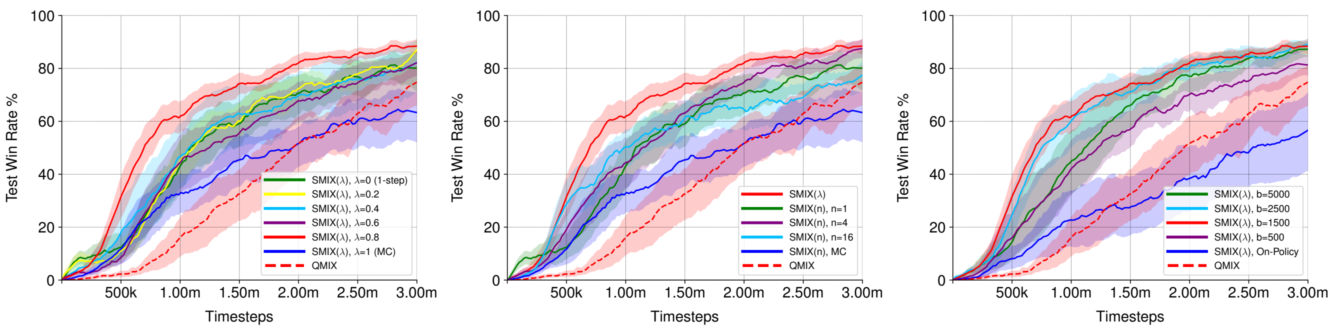
D. *Generalizing SMIX(λ) to Other MARL Algorithms*

SMIX(λ) focuses on centralized value function evaluation with λ-return calculated from off-policy episodes. This method could ideally be generalized to other MARL algorithms incorporating critic estimation, including critic-only and actor-critic algorithms.

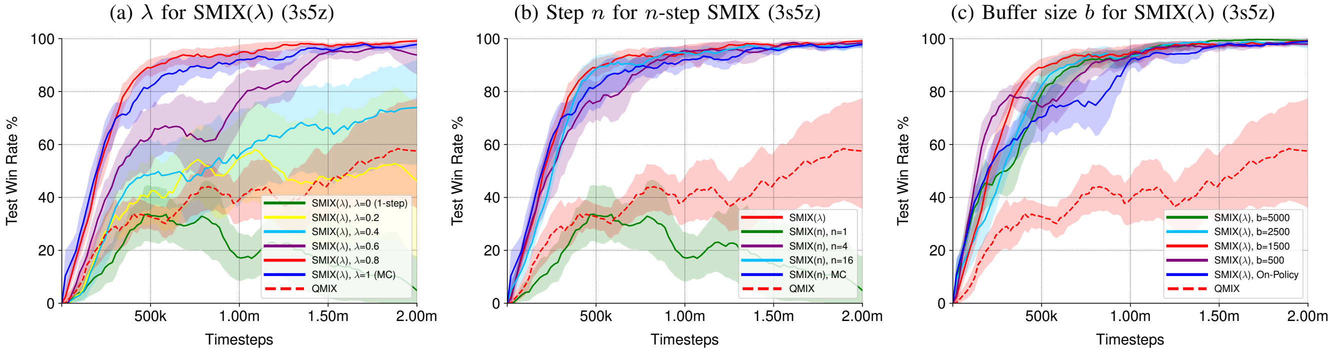
To demonstrate the benefits of our approach, we generalize SMIX(λ) to the following algorithms: COMA, VDN and IQL. We achieve these by replacing their original value function estimation procedure with ours (see Section IV). Then we

TABLE III
MEAN, STANDARD DEVIATION, AND MEDIAN OF TEST WIN RATE PERCENTAGES AFTER TRAINING FOR 1 MILLION TIMESTEPS IN NINE DIFFERENT SCENARIOS.

Algorithms		SMIX(λ)	QMIX	SMIX(λ)-COMA	COMA	SMIX(λ)-VDN	VDN	SMIX(λ)-IQL	IQL
3m	mean $\pm$ *std*	**99** ($\pm$0)	95 ($\pm$3)	**93** **($\pm$8)**	92 ($\pm$2)	**98** ($\pm$0)	95 ($\pm$2)	**91** ($\pm$4)	83 ($\pm$9)
	median	**99**	95	**97**	93	**98**	95	**94**	86
8m	mean $\pm$ *std*	**91** ($\pm$3)	90 ($\pm$3)	**92** ($\pm$2)	90 ($\pm$2)	**94** ($\pm$3)	86 ($\pm$5)	**80** ($\pm$5)	59 ($\pm$15)
	median	**90**	89	**93**	91	**93**	87	**79**	58
2s3z	mean $\pm$ *std*	**90** ($\pm$4)	81 ($\pm$7)	**44** ($\pm$18)	24 ($\pm$6)	**78** ($\pm$14)	64 ($\pm$16)	**32** ($\pm$8)	14 ($\pm$10)
	median	**91**	81	**47**	24	**79**	71	**31**	13
3s5z	mean $\pm$ *std*	**61** ($\pm$11)	16 ($\pm$12)	**0** ($\pm$0)	**0** ($\pm$0)	**29** ($\pm$12)	1 ($\pm$2)	**0** ($\pm$0)	**0** ($\pm$0)
	median	**62**	11	**0**	**0**	**26**	0	**0**	**0**
2m_vs_1z	mean $\pm$ *std*	**99** ($\pm$0)	69 ($\pm$38)	**0** ($\pm$0)	**0** ($\pm$0)	**99** ($\pm$0)	**99** ($\pm$0)	**99** ($\pm$0)	85 ($\pm$27)
	median	**100**	99	**0**	**0**	**99**	**99**	**100**	99
2s_vs_1sc	mean $\pm$ *std*	**94** ($\pm$5)	39 ($\pm$19)	**97** ($\pm$4)	77 ($\pm$11)	**96** ($\pm$2)	86 ($\pm$8)	**92** ($\pm$6)	51 ($\pm$22)
	median	**96**	45	**100**	78	**97**	88	**94**	54
3s_vs_3z	mean $\pm$ *std*	**84** ($\pm$14)	15 ($\pm$20)	**0** ($\pm$0)	**0** ($\pm$0)	**67** ($\pm$25)	27 ($\pm$9)	**35** ($\pm$21)	5 ($\pm$4)
	median	**88**	9	**0**	**0**	**83**	27	**31**	6
1c3s5z	mean $\pm$ *std*	**92** ($\pm$3)	72 ($\pm$32)	24 ($\pm$19)	**27** ($\pm$13)	**84** ($\pm$3)	61 ($\pm$5)	**11** ($\pm$5)	5 ($\pm$6)
	median	**93**	88	18	**28**	**85**	61	**11**	3
MMM	mean $\pm$ *std*	**91** ($\pm$4)	59 ($\pm$15)	20 ($\pm$17)	**34** ($\pm$18)	**91** ($\pm$9)	72 ($\pm$17)	**35** ($\pm$14)	20 ($\pm$19)
	median	**91**	61	22	**39**	**94**	78	**34**	15



(a) λ for SMIX(λ) (3s5z) (b) Step n for n-step SMIX (3s5z) (c) Buffer size b for SMIX(λ) (3s5z)



(d) λ for SMIX(λ) (2s_vs_1sc) (e) Step n for n-step SMIX (2s_vs_1sc) (f) Buffer size b for SMIX(λ) (2s_vs_1sc)

Fig. 5. Sensitivity of SMIX(λ) to selected hyperparameters in two different scenarios. The mean and 95% confidence interval is shown across 10 independent runs. The performance of the baseline (QMIX) is shown as a dashed red line. (a) and (d) show the sensitivity of SMIX(λ) to the value of λ; (b) and (e) show the results using n-step TD with different backup steps; (c) and (f) show the comparison between SMIX(λ) and its on-policy version.

get three new algorithms called SMIX(λ)-COMA, SMIX(λ)-VDN and SMIX(λ)-IQL respectively. Figure 4 gives the comparisons between our methods and their counterparts (we also provide quantitative results in Supplementary). Overall, most of the extended methods (solid line) perform on par or significantly better than their counterparts (in the same color but dashed line) in most scenarios both in terms of the final win rate and learning speed.

SMIX(λ)-VDN considerably improves the performance of VDN. Especially under challenging scenarios such as 3s5z, SMIX(λ)-VDN achieves about 75% final win rate, which is more than twice as that of VDN (nearly 30%). Such improvement may be contributed to λ-return and the independence of the unrealistic centralized greedy assumption during the learning phase. Furthermore, we find that SMIX(λ)-VDN performs even better than QMIX in most scenarios. Recall that VDN uses a linear combination of decentralized Q-values (and so does our SMIX(λ)-VDN) and QMIX extends VDN by combining decentralized Q-values in a non-linear way. Thus, QMIX can represent a richer class of CVF than VDN. However, our results indicate that the performance bottleneck of VDN may not be the limited representational capacity, but how to effectively balance the bias and variance in the estimation of CVF.

Similar performance improvements can also be seen in COMA, which can be considered as a success of utilizing the off-policy data, as COMA also adopts λ-return but uses only the on-policy data. Another observation is that our method also works for IQL, which is a fully decentralized MARL algorithm. This suggests that our method is not limited to *centralized* value function estimation but also applicable to *decentralized* cases.

It is worth mentioning that the extended methods may not make improvements if the original methods do not work, e.g., COMA, IQL, and their counterparts do not work in 3s5z scenario (see Figure 4d and Table III). The reason may be that the main limitations of COMA and IQL on 3s5z do not lie in the inaccurate value function estimation, but rather in other problems, e.g., scaling not well to a large number of agents and multi-agent credit assignment problem.

E. Ablation Study

We perform the ablation experiments to investigate the necessity of balancing the bias and variance and the influence of utilizing the off-policy data.

$\boldsymbol{\lambda}$-Return vs. n-Step Returns. To investigate the necessity of balancing the bias and variance in multi-agent problems, we adjust the parameter λ, where larger λ corresponds to smaller bias and larger variance whereas smaller λ indicates the opposite. Especially, $\lambda = 0$ is equivalent to *one-step return* (corresponding to the largest bias and the smallest variance); $\lambda = 1$ is equivalent to *Monte-Carlo (MC) return* (∞-step, corresponding to the smallest bias and the largest variance). We also evaluate a variant named SMIX(n), which uses n-step return in place of λ-return as the TD target, i.e., $y_t^{tot} = \sum_{i=1}^{n} \gamma^{i-1} r_{t+i} + \gamma^n Q(\boldsymbol{\tau}_{t+n}, \mathbf{a}_{t+n}; \theta^-)$.

As Figure 5a and 5d show, SMIX(λ) with $\lambda = 0.8$ consistently achieves the best performance in selected scenarios. The method with $\lambda = 1$ (MC return, blue line) performs the worst in 3s5z, while $\lambda = 0$ (one-step return, green line) performs the worst in 2s_vs_1sc. These results reveal that the large variance of MC return or large bias of one-step return may degrade the performance. Similar results could also be seen in SMIX(n) (Figure 5b and 5e), where SMIX(n) with $n = 4$ performs the best in 3s5z, while the one with $n = 16$ performs the best in 2s_vs_1sc. It is not easy to find the same n for SMIX(n) as SMIX(λ) which sets $\lambda = 0.8$ and performs consistently well across different maps. In summary, it seems necessary to balance the trade-off between bias and variance in multi-agent problems, and λ-return could serve as a convenient method to achieve such a trade-off.

Incorporating Off-Policy Data vs. Pure On-Policy Data. To investigate the influence of utilizing the off-policy data, we perform experiments to compare SMIX(λ) against its on-policy version by scaling the size of the replay buffer. The on-policy version of SMIX(λ) corresponds to SMIX(λ) with buffer size $b = 4$ (the most recent 4 episodes in the replay buffer are all on-policy data), while the off-policy SMIX(λ) are the ones with buffer size $b > 4$, where the percentage of off-policy data increases with the size of the replay buffer.

As shown in Figure 5c and 5f, all the variants of SMIX(λ) incorporating off-policy data ($b > 4$) perform better than the on-policy version ($b = 4$) in selected scenarios. Notably, the performance of SMIX(λ) with $b = 1500$ is almost twice that of the on-policy version both in terms of the final win rate and learning speed in 3s5z. Note that 3s5z (8 units) map is more complex than 2s_vs_1sc (2 units) in terms of the number of agents, and consequently, the joint action space of the former is much larger. However, more off-policy data does not always lead to better performance, as the method with $b = 5000$ (green line) performs worse than the one with $b = 1500$ (solid red line) in both scenarios. Actually, the buffer size is corresponding to the ϵ in Theorem 2 which measures the mismatch between the target policy π and the behavior policy μ. A smaller buffer size makes SMIX(λ) less sample efficient but a larger buffer size results in a looser error bound which biases the CVF estimation. This may explain why the performance degrades once the buffer size exceeds a threshold value. And our experimental results suggest that a moderate buffer size of 1500 could be a good candidate.

TABLE IV
THE SCALABILITY OF SMIX(λ) AND QMIX AFTER TRAINING FOR 1 MILLION TIMESTEPS.

Algorithms		SMIX(λ)	QMIX
3m	mean $\pm$ *std*	**99** ($\pm$0)	95 ($\pm$3)
	median	**99**	95
8m	mean $\pm$ *std*	**91** ($\pm$3)	90 ($\pm$3)
	median	**90**	89
25m	mean $\pm$ *std*	**75** ($\pm$26)	30 ($\pm$17)
	median	**93**	24

Scalability. The results in Table IV show the scalability of SMIX(λ) and QMIX after training for 1 million steps. Overall, the performance of both methods decreases along with the increasing number of agents. However, our SMIX(λ) still outperforms QMIX, especially in hard scenarios 25m. Specifically, with 3 agents (the 3m map), SMIX(λ) achieves the best performance among the compared methods with a 99% win rate. By increasing the number of agents to 8 (the 8m map), the performance of all the methods decreases due to the higher degree of challenging of the task, while our method still performs best among the compared ones[7]. Finally, when the number of agents been increased to 25 (the 25m map), the performance of QMIX decreases dramatically, which is not the case for SMIX(λ). These results show that the centralized value function estimation method used in SMIX(λ) method

[7]See Table III for more results on 3m and 8m.

has better scalability and performs more robust in challenging tasks than QMIX. Finally, it is worth mentioning that for our experiments with up to 25 agents, the joint action space would be as large as $|\mathcal{A}|^{25}$, which imposes a great challenge to any MARL method.

VII. Conclusions & Future work

One of the central challenges in multi-agent reinforcement learning with CTDE settings is to estimate the centralized value function. However, the sparse experiences and non-Markovian nature of the multi-agent environments make this become a challenging task. To address this issue, we present the SMIX(λ) approach. Experimental results show that our approach significantly improves the state-of-the-art performance for MARL by enhancing the quality of CVF through three contributions: (1) removing the greedy assumption to help to learn a more flexible functional structure, (2) using off-policy learning to alleviate the problem of sparse experiences and to improve exploration, and (3) using λ-return to handle the non-Markovian property of the environments and balance the bias and variance of the algorithm. Our results also show that the proposed method is beneficial to other MARL methods by replacing their CVF estimator with SMIX(λ). Last but not least, our analysis shows that SMIX(λ) has a nice convergence guarantee through off-policy learning without importance sampling, which brings potential advantages in multi-agent settings.

Our future work will focus on incorporating the communication and opponent modeling methods into SMIX(λ) to further tackle the non-stationarity issue during the execution. We also aim to make SMIX(λ) perform more efficiently in dealing with a large numbers of agents.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[3] D. Zhao and Y. Zhu, "Meca near-optimal online reinforcement learning algorithm for continuous deterministic systems," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 2, pp. 346–356, 2014.

[4] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2017.

[5] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5174–5184, 2018.

[6] D. Ye, M. Zhang, and Y. Yang, "A multi-agent framework for packet routing in wireless sensor networks," *sensors*, vol. 15, no. 5, pp. 10 026–10 047, 2015.

[7] E. Van der Pol and F. A. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," *Proceedings of the Learning, Inference and Control of Multi-Agent Systems*, 2016.

[8] W. Liu and J. Huang, "Cooperative adaptive output regulation for lower triangular nonlinear multi-agent systems subject to jointly connected switching networks," *IEEE transactions on neural networks and learning systems*, 2019.

[9] C. Yu, M. Zhang, F. Ren, and G. Tan, "Emotional multiagent reinforcement learning in spatial social dilemmas," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 12, pp. 3083–3096, 2015.

[10] G. Jing, Y. Zheng, and L. Wang, "Consensus of multiagent systems with distance-dependent communication networks," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 11, pp. 2712–2726, 2016.

[11] X. Bu, Z. Hou, and H. Zhang, "Data-driven multiagent systems consensus tracking using model free adaptive control," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 5, pp. 1514–1524, 2017.

[12] Y. Zheng, J. Ma, and L. Wang, "Consensus of hybrid multi-agent systems," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 4, pp. 1359–1365, 2017.

[13] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman *et al.*, "Human-level performance in 3d multiplayer games with population-based reinforcement learning," *Science*, vol. 364, no. 6443, pp. 859–865, 2019.

[14] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] T. Rashid, M. Samvelyan, C. S. Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, 2018, pp. 4292–4301.

[16] G. J. Laurent, L. Matignon, L. Fort-Piat *et al.*, "The world of independent learners is not markovian," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 15, no. 1, pp. 55–64, 2011.

[17] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6379–6390.

[18] F. A. Oliehoek, M. T. Spaan, and N. Vlassis, "Optimal and approximate q-value functions for decentralized pomdps," *Journal of Artificial Intelligence Research*, vol. 32, pp. 289–353, 2008.

[19] L. Kraemer and B. Banerjee, "Multi-agent reinforcement learning as a rehearsal for decentralized planning," *Neurocomputing*, vol. 190, pp. 82–94, 2016.

[20] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 2085–2087.

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

[22] Bellman, *Dynamic Programming*, ser. Rand Corporation research study. Princeton University Press, 1957.

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

[24] M. Samvelyan, T. Rashid, C. Schroeder de Witt, G. Farquhar, N. Nardelli, T. G. Rudner, C.-M. Hung, P. H. Torr, J. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," in *International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 2186–2188.

[25] C. Wen, X. Yao, Y. Wang, and X. Tan, "Smix(λ): Enhancing centralized value functions for cooperative multi-agent reinforcement learning," in *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] J. Foerster, N. Nardelli, G. Farquhar, T. Afouras, P. H. Torr, P. Kohli, and S. Whiteson, "Stabilising experience replay for deep multi-agent reinforcement learning," in *International Conference on Machine Learning*, vol. 70. JMLR. org, 2017, pp. 1146–1155.

[28] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "Is multiagent deep reinforcement learning the answer or the question? a brief survey," *arXiv preprint arXiv:1810.05587*, 2018.

[29] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[30] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 1054–1062.

[31] M. J. Kearns and S. P. Singh, "Bias-variance error bounds for temporal difference updates." in *Annual Conference on Computational Learning Theory*. Citeseer, 2000, pp. 142–147.

[32] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *International Conference on Machine Learning*, 2001, pp. 417–424.

[33] K. De Asis, J. F. Hernandez-Garcia, G. Z. Holland, and R. S. Sutton, "Multi-step reinforcement learning: A unifying algorithm," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[34] J. F. Hernandez-Garcia and R. S. Sutton, "Understanding multi-step deep reinforcement learning: A systematic study of the dqn target," *arXiv preprint arXiv:1901.07510*, 2019.

[35] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International Conference on Machine Learning*, 2019, pp. 2052–2062.

[36] F. A. Oliehoek and Amato, *A concise introduction to decentralized POMDPs*. Springer, 2016, vol. 1.

[37] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[38] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[39] R. Sutton, A. R. Mahmood, D. Precup, and H. Hasselt, "A new q (λ) with interim forward view and monte carlo equivalence," in *International Conference on Machine Learning*, 2014, pp. 568–576.

[40] C. J. C. H. Watkins, "Learning from delayed rewards," *Phd thesis*, 1989.

[41] A. Harutyunyan, M. G. Bellemare, T. Stepleton, and R. Munos, "Q (λ) with off-policy corrections," in *Proceedings of the International Conference on Algorithmic Learning Theory*. Springer, 2016, pp. 305–320.

[42] D. Precup, R. S. Sutton, and S. Singh, "Eligibility traces for off-policy policy evaluation," in *International Conference on Machine Learning*, 2000.

[43] Q. Liu, L. Li, Z. Tang, and D. Zhou, "Breaking the curse of horizon: Infinite-horizon off-policy estimation," in *Advances in Neural Information Processing Systems*, 2018, pp. 5356–5366.

[44] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Advances in Neural Information Processing Systems*, 2014.

[45] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *International Conference on Machine Learning*, 1993, pp. 330–337.

[46] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson, "Maven: Multi-agent variational exploration."