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

Chao Wen*, **Xinghu Yao***, **Yuhui Wang, Xiaoyang Tan**
College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
MIIT Key Laboratory of Pattern Analysis and Machine Intelligence
Collaborative Innovation Center of Novel Software Technology and Industrialization
Nanjing 211106, China
{chaowen, xinghuyao, y.wang, x.tan}@nuaa.edu.cn

Abstract

This work presents a sample efficient and effective value-based method, named SMIX(λ), for reinforcement learning in multi-agent environments (MARL) within the paradigm of centralized training with decentralized execution (CTDE), in which learning a stable and generalizable centralized value function (CVF) is crucial. To achieve this, our method carefully combines different elements, including 1) removing the unrealistic centralized greedy assumption during the learning phase, 2) using the λ-return to balance the trade-off between bias and variance and to deal with the environment's non-Markovian property, and 3) adopting an experience-replay style off-policy training. Interestingly, it is revealed that there exists inherent connection between SMIX(λ) and previous off-policy $Q(\lambda)$ approach for single-agent learning. Experiments on the StarCraft Multi-Agent Challenge (SMAC) benchmark show that the proposed SMIX(λ) algorithm outperforms several state-of-the-art MARL methods by a large margin, and that it can be used as a general tool to improve the overall performance of a CTDE-type method by enhancing the evaluation quality of its CVF. We open-source our code at: https://github.com/chaovven/SMIX.

1 Introduction

Recently, reinforcement learning (RL) has made great success in a variety of domains, from game playing (Mnih et al. 2015; Silver et al. 2017) to complex continuous control tasks (Schulman et al. 2017). However, many real-world problems are inherently multi-agent in nature, such as network packet routing (Ye, Zhang, and Yang 2015), traffic light control (Van der Pol and Oliehoek 2016), and multi-player games (Jaderberg et al. 2019), 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 (Foerster et al. 2018; Rashid et al. 2018), unstable environments caused by the interaction of individual agents (Laurent et al. 2011; Lowe et al. 2017), and multi-agent credit assignment in cooperative scenarios with global rewards (Foerster et al. 2018; Rashid et al. 2018). 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 (Tan 1993; Foerster et al. 2017; 2018; Rashid et al. 2018).

Recently the paradigm of centralized training with decentralized execution (CTDE) has become popular for multi-agent learning (Oliehoek, Spaan, and Vlassis 2008; Kraemer and Banerjee 2016; Foerster et al. 2018; Rashid et al. 2018) 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 learnt 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 (Lowe et al. 2017; Rashid et al. 2018; Foerster et al. 2018; Sunehag et al. 2018), 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 (Sunehag et al. 2018), QMIX (Rashid et al. 2018), and QTRAN (Son et al. 2019) 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" (Bellman 1957) of the joint action space results in the sparsity of experiences; 2) the challenges of non-Markovian property (Laurent et al. 2011) 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,

*These authors contributed equally to this work.

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 (Sutton and Barto 2018) 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 is implemented with experience replay, which is shown to have 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) (Samvelyan et al. 2019), SMIX(λ) is demonstrated to achieve state-of-the-art performance in selected scenarios. Furthermore, significant performance improvements by existing CTDE-type MARL algorithms are observed by replacing their CVF estimator with the newly proposed SMIX(λ).

In what follows, after a brief discussion on the characteristics of the hypothesis space of CVF in Section 2, the proposed SMIX(λ) method and its convergence analysis are respectively described in Section 3 and Section 4. Experimental results are given in Section 5 and we conclude the paper in Section 6.

2 Background

We consider a cooperative multi-agent task which can be described as a Dec-POMDP (Oliehoek and Amato 2016). A Dec-POMDP is 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 time step, each agent $i \in \{1, 2, \cdots, N\}$ chooses an action $a^i \in \mathcal{A}$, forming a joint action $\mathbf{a} = \left\{a^1, a^2, \cdots, a^N\right\} \in \mathcal{A}^N$. 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 in which each agent draws partial observation $o \in \mathcal{O}$ from the observation function $\mathcal{Z}(s, i) : \mathcal{S} \times N \mapsto \mathcal{O}$. [1] 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 learnt 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. 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.

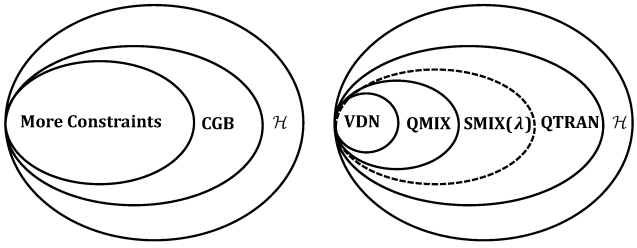
[1] Here we condsider a observation function that differs from the standard Dec-POMDP (Oliehoek and Amato 2016).

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 (Shalev-Shwartz and Ben-David 2014; Russell and Norvig 2016). 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 learnt 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:

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

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.



(a) (b)

Figure 1: (a) The size of hypothesis space corresponding to different constraints. (b) The relationship of hypothesis spaces of different algorithms.

VDN, QMIX and QTRAN

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

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

where Q_{tot} is the centralized Q function and Q^i is the Q function for each agent i. In VDN (Sunehag et al. 2018), all the combination coefficients $\alpha_i, i = 1, 2, \cdots, N$ are set to 1. QMIX (Rashid et al. 2018) 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*

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

The proof of this theorem is provided in the Supplementary. Note that QMIX also relies on this result to simplify its optimization procedure.

QTRAN (Son et al. 2019) 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. Although a coordinate decent-type method is proposed in QTRAN to address the issue, the method's scalability and the range of practical use can be limited by this. Figure 1b gives an illustration of the relationship among hypothesis spaces for VDN, QMIX, SMIX(λ), and QTRAN. The hypothesis space of SMIX(λ) is further discussed in Section 3.

3 Methods

In this section, we give the details of the proposed SMIX(λ) method, which is a SARSA(λ) (Sutton et al. 2014) style off-policy method that aims at learning a centralized value function within the framework of CTDE for better MARL.

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 agents to guide the training process of each agent. Typical implementations of this idea are VDN (Sunehag et al. 2018) and QMIX (Rashid et al. 2018), in which the centralized Q_{tot} function is learnt 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 update 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 (Watkins 1989) as well. Hence to remove our exact relying on this condition in the learning phase, it is necessary to abandon such updating rule at all when learning the centralized value function network.

Alternatively, one can use a SARSA (Sutton and Barto 2018)-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 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.

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 function for $\boldsymbol{\pi}$ using data $\boldsymbol{\tau}$ generated by following $\boldsymbol{\mu}$ can be expressed as follows (Munos et al. 2016),

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

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 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 (Liu et al. 2018), 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, Foerster et al. (2017) proposed a method which effectively addresses the first issue by avoiding calculating the product over the trajectories but how to solve the second one remains open.

The above analysis highlights the need of exploring alternative approaches that are able to perform off-policy learning without importance sampling in multi-agent settings.

The SMIX(λ) Method

To achieve the above goal, our key idea 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) and use an experience replay memory to store the most recent off-policy data. Actually, previous work (Fujimoto, Meger, and Precup 2019) shows that the off-policy experiences generated during the interaction with the environment tend to be heavily correlated to the current policy. Their experimental results also reveal that the distribution of off-policy data during the training procedure is very close to the current policy.

Then, we use the λ-return (Sutton and Barto 2018) as the TD target estimator, which is defined as follows:

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

[2]The policy evaluation strategy of many popular methods can be expressed as (4), including SARSA(λ) (Sutton et al. 2014), off-policy importance sampling methods (Precup, Sutton, and Dasgupta 2001), off-policy $Q(\lambda)$ method (Harutyunyan et al. 2016), tree-backup method, TB(λ) (Precup, Sutton, and Singh 2000) and Retrace(λ) (Munos et al. 2016). These methods differ in the definition of the coefficient ρ_i and error term $\delta_t^{\boldsymbol{\pi}}$ (Harutyunyan et al. 2016; Munos et al. 2016).

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 n-step return and θ^- are parameters of the target network. Plugging this into (4) and setting $\rho_i = 1.0$ for all i, we have,

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

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:

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

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

Note that in the training phase, we remove the explicit structural constraints between the CVF and decentralized policies by abandoning the Q-learning updating rules, which makes SMIX(λ) optimize in a larger hypothesis space than QMIX (Rashid et al. 2018). While in implementation, the deep network architecture for $Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \mathbf{a}; \theta)$ in SMIX(λ) is the same as that of QMIX except the training objective. This indicates that 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 SMIX(λ) is much better than QTRAN, 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 the Supplementary. It is worth noting that our method of training a CVF is a general method and can be easily applied to other CTDE methods, such as VDN (Sunehag et al. 2018), COMA (Foerster et al. 2018), and even fully decentralized methods, such as IQL (Tan 1993) (cf., Figure 2).

4 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 $Q(\lambda)$ Harutyunyan et al. (2016), 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 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,

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

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, in which the importance sampling ratio ρ_i plays the role for off-policy correction. In SMIX(λ), it is further relaxed as: $\rho_i = \frac{\boldsymbol{\pi}(\mathbf{a}_i|\boldsymbol{\tau}_i)}{\boldsymbol{\mu}(\mathbf{a}_i|\boldsymbol{\tau}_i)} = 1$, then corresponding to (4), the update rule of SMIX(λ) can be expressed as,

$$Q^{\text{SMIX}}(\boldsymbol{\tau}_t, \mathbf{a}_t) \leftarrow Q^{\text{SMIX}}(\boldsymbol{\tau}_t, \mathbf{a}_t) + \mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\lambda\gamma\right)\delta_k^{\boldsymbol{\pi}}\right],$$
$$\delta_k^{\boldsymbol{\pi}} = \left(r_{k+1} + \gamma \mathbb{E}_{\boldsymbol{\mu}} Q^{\text{SMIX}}(\boldsymbol{\tau}_{k+1}, \cdot) - Q^{\text{SMIX}}(\boldsymbol{\tau}_k, \mathbf{a}_k)\right). \quad (10)$$

In contrast with the multiplicative operation for off-policy correction, an additive-type operation can also be used (Harutyunyan et al. 2016). In particular, an additive correction term, derived from (9), is added to each reward when calculating $G^{\boldsymbol{\mu},\boldsymbol{\pi}}$. The major advantage of this additive 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 [3]. Specifically, the updating rule of $Q(\lambda)$ method is (Harutyunyan et al. 2016):

$$Q^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t) \leftarrow Q^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t) + \mathbb{E}_{\boldsymbol{\mu}}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\lambda\gamma\right)\hat{\delta}_k^{\boldsymbol{\pi}}\right],$$
$$\hat{\delta}_k^{\boldsymbol{\pi}} = \left(r_{k+1} + \gamma \mathbb{E}_{\boldsymbol{\pi}} Q^{Q(\lambda)}(\boldsymbol{\tau}_{k+1}, \cdot) - Q^{Q(\lambda)}(\boldsymbol{\tau}_k, \mathbf{a}_k)\right). \quad (11)$$

By comparing (10) and (11), 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:*

$$|Q_{n+1}^{SMIX}(\boldsymbol{\tau}_t, \mathbf{a}_t) - Q_{n+1}^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t)| \leq \frac{\epsilon\gamma}{1-\lambda\gamma} M. \quad (12)$$

The proof of this theorem is provided in the Supplementary. 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 (Harutyunyan et al. 2016).

Theorem 3. *(Harutyunyan et al. 2016) Consider the sequence of Q-functions computed according to (11) 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 same condition required for the convergence of TD(λ) we have, almost surely:*

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

[3] 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., Fujimoto, Meger, and Precup (2019)).

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 MARL 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 MARL, 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 (11), 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 (10), 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 MARL, compared to the $Q(\lambda)$ and QTRAN.

Finally, before ending this section, we summarize some of the key characteristics of QMIX and SMIX(λ) in Table 1.

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^*	$Q^{\boldsymbol{\pi}}$

Table 1: The comparison of QMIX and SMIX(λ).

5 Experiments

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

Environmental Setup and Implementation Details

We evaluate the algorithms on the StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al. 2019) benchmark, which provides a set of rich and challenging cooperative scenarios. We focus on the *decentralized micromanagement* problem in the combat scenarios, in which each of the learning agents controls an individual army unit. At the beginning of each episode, two groups of units are placed on the map with random initial positions within groups. The units of the two groups are controlled by decentralized agents and built-in heuristic game AI bot respectively. The difficulty of the built-in AI bot is set to *very difficult* in experiments. Each agent can only receive local observations within its *sight range*. Global state information is available during training. 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 Samvelyan et al. (2019) for full details of the environment.

We consider the following 3 types of scenarios: (A) *homogeneous and symmetric units*: 3 Marines (3m), 8 Marines (8m); (B) *heterogeneous and symmetric units*: 2 Stalkers and 3 Zealots (2s3z), 3 Stalkers and 5 Zealots (3s5z); (C) *asymmetric units*: 3 Stalkers vs. 3 Zealots (3s_vs_3z), 2 Stalkers vs. 1 Spine Crawler (2s_vs_1sc).

We use *test win rate* as the evaluation metric, which is proposed in Samvelyan et al. (2019) and evaluated in the following procedure: we pause training after every 20000 timesteps and run 24 independent test episodes 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.

SMIX(λ) adopts the same architecture as QMIX (Rashid et al. 2018), except that 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. See Supplementary for more details.

Comparative Evaluation

We compare our SMIX(λ) with state-of-the-art algorithms QMIX (Rashid et al. 2018) and COMA (Foerster et al. 2018), which currently perform the best on the SMAC benchmark. VDN (Sunehag et al. 2018) and IQL (Tan 1993) are chosen as baselines for comparisons.

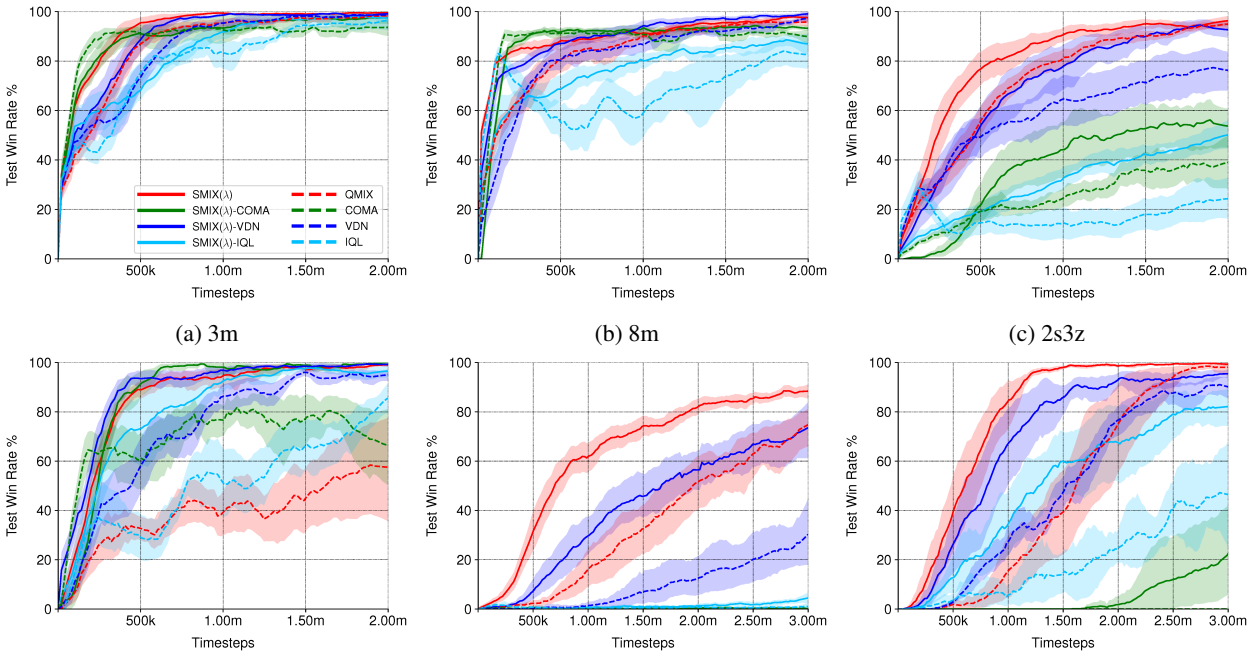
The results of all methods are plotted in Figure 2. Overall, SMIX(λ) significantly outperforms all the comparison methods in heterogeneous or asymmetric scenarios (i.e., scenarios except 3m and 8m), while performing comparably to them in homogeneous and symmetric scenarios (i.e., 3m and 8m) both in terms of the learning speed and final performance. Especially in 3s5z, SMIX(λ) (solid red line) achieves nearly 90% win rate, while the best comparison method QMIX (dotted red line) achieves about only 70% test win rate. In 2s_vs_1sc and 3s_vs_3z, SMIX(λ) also requires less than half the number of samples of QMIX and other comparison methods to reach the asymptotic performance. The superior performance of SMIX(λ) using λ-return with off-policy episodes presents a clear benefit over the one-step estimation of QMIX.

Generalizing SMIX(λ) to Other MARL Algorithms

SMIX(λ) focuses on CVF estimation with λ-return using off-policy data. This method could ideally be generalized to other MARL algorithms incorporating critic estimation.

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 3). Then we get three new algorithms called SMIX(λ)-COMA, SMIX(λ)-VDN and SMIX(λ)-IQL respectively. Figure 2 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 in difficult scenarios such as 3s5z, SMIX(λ)-VDN achieves about 75% final win rate, which



(a) 3m (b) 8m (c) 2s3z

(d) 2s_vs_1sc (e) 3s5z (f) 3s_vs_3z

Figure 2: 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 six 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.

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 learning. Furthermore, we find that SMIX(λ)-VDN performs even better than QMIX in most scenarios. Note that VDN uses linear combination of decentralized Q-values (and so does our SMIX(λ)-VDN), whereas QMIX extends VDN by combining decentralized Q-values in a non-linear way, which allows it to represent a richer class of centralized value functions. 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.

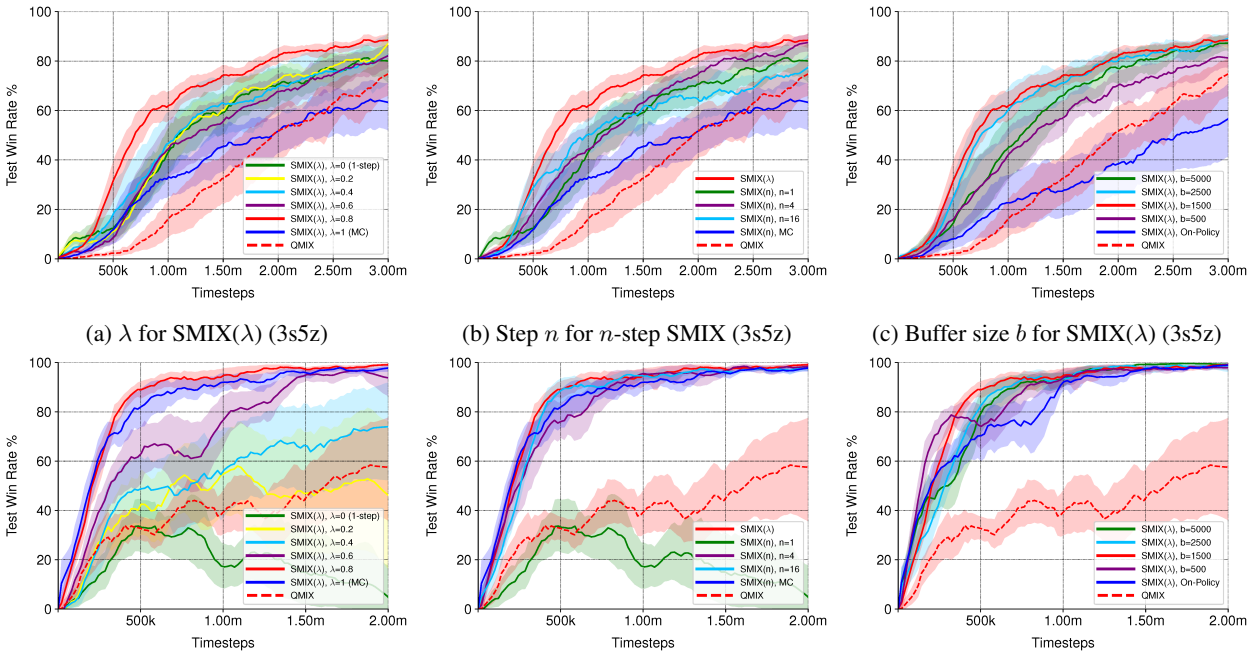
It is worth mentioning that the extended methods may not make improvements if the original methods do not work, e.g., COMA, IQL, and their counterparts do not work in 3s5z (Figure 2e). The reason may be that the main limitations of COMA and IQL on 3s5z do not lie in the inaccurate value function estimation, but rather in other problems, e.g., scaling not well to large number of agents and multi-agent credit assignment problem.

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.

λ-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 3a and 3d show, SMIX(λ) with $\lambda = 0.8$ consistently achieves the best performance in selected maps. The method with $\lambda = 1$ (MC) performs the worst in 3s5z, while $\lambda = 0$ (one-step) 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 3b and 3e), 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



(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)

Figure 3: Sensitivity of SMIX(λ) to selected hyperparameters in two different scenarios. The mean and 95% confidence interval are 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.

different maps. In summary, it seems necessary to balance the bias and variance in multi-agent problems, and λ-return could serve as a convenient tool to achieve such a balance.

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 3c and 3f, 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, and the performance may even degrade once the buffer size exceeds a threshold value. That is because the buffer size is corresponding to the ϵ in Theorem 2 which measures the mismatch between the target policy $\boldsymbol{\pi}$ and the behavior policy $\boldsymbol{\mu}$. 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. In practice, a moderate buffer size of 1500 could be a good candidate for a range of tasks.

6 Conclusions

One of the central challenges in MARL with CTDE settings is to estimate the CVF. To address this issue, we present the SMIX(λ) approach. Experimental results show that our approach significantly improves the state-of-the-art performance by enhancing the quality of CVF through three contributions: (1) removing the greedy assumption 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. Our results also show that the proposed method is beneficial to other MARL methods by replacing their CVF estimator with ours. Last but not least, our analysis shows that SMIX(λ) has nice convergence guarantee through off-policy learning without importance sampling, which brings potential advantages in multi-agent settings.

7 Acknowledgements

This work is partially supported by National Science Foundation of China (61976115, 61672280, 61732006), AI+ Project of NUAA(56XZA18009), Graduate Innovation Foundation of NUAA (Kfjj20191608).

References

Bellman. 1957. *Dynamic Programming*. Rand Corporation research study. Princeton University Press.

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

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

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

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2052–2062.

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

Jaderberg, M.; Czarnecki, W. M.; Dunning, I.; Marris, L.; Lever, G.; Castaneda, A. G.; Beattie, C.; Rabinowitz, N. C.; Morcos, A. S.; Ruderman, A.; et al. 2019. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science* 364(6443):859–865.

Kraemer, L., and Banerjee, B. 2016. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing* 190:82–94.

Laurent, G. J.; Matignon, L.; Fort-Piat, L.; et al. 2011. The world of independent learners is not markovian. *International Journal of Knowledge-based and Intelligent Engineering Systems* 15(1):55–64.

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

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

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

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

Oliehoek, F. A., and Amato. 2016. *A concise introduction to decentralized POMDPs*, volume 1. Springer.

Oliehoek, F. A.; Spaan, M. T.; and Vlassis, N. 2008. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research* 32:289–353.

Precup, D.; Sutton, R. S.; and Dasgupta, S. 2001. Off-policy temporal-difference learning with function approximation. In *International Conference on Machine Learning*, 417–424.

Precup, D.; Sutton, R. S.; and Singh, S. 2000. Eligibility traces for off-policy policy evaluation. In *International Conference on Machine Learning*.

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

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

Samvelyan, M.; Rashid, T.; Schroeder de Witt, C.; Farquhar, G.; Nardelli, N.; Rudner, T. G.; Hung, C.-M.; Torr, P. H.; Foerster, J.; and Whiteson, S. 2019. The starcraft multi-agent challenge. In *International Conference on Autonomous Agents and MultiAgent Systems*, 2186–2188.

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

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

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

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

Sunehag, P.; Lever, G.; Gruslys, A.; et al. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *International Conference on Autonomous Agents and MultiAgent Systems*, 2085–2087.

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

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

Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *International Conference on Machine Learning*, 330–337.

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

Watkins, C. J. C. H. 1989. Learning from delayed rewards.

Ye, D.; Zhang, M.; and Yang, Y. 2015. A multi-agent framework for packet routing in wireless sensor networks. *sensors* 15(5):10026–10047.

Supplementary

A. Proof of Theorem 1

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

$$\max_{\mathbf{a}} Q_{tot}(\boldsymbol{\tau}, \mathbf{a}) = Q_{tot}\left(\tau, \text{argmax}_{a^1} Q^1(\tau^1, a^1), \cdots, \text{argmax}_{a^N} Q^N(\tau^N, a^N)\right) \tag{13}$$

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

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

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

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

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

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

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

$$\begin{aligned}
& \hat{Q}\left(Q^1(\tau^1, a^1), \ldots, Q^N(\tau^N, a^N)\right) \\
\leq\ & \hat{Q}\left(\max_{a^1} Q_1(\tau^1, a^1), \max_{a^2} Q^2(\tau^2, a^2), Q^3(\tau^3, a^3), \ldots, Q^N(\tau^N, a^N)\right) \\
\leq\ & \hat{Q}\left(\max_{a^1} Q^1(\tau^1, a^1), \max_{a^2} Q^2(\tau^2, a^2), \ldots, \max_{a^N} Q^N(\tau^N, a^N)\right).
\end{aligned}$$

Therefore, we obtain

$$\max_{a^1, \ldots, a^N} \hat{Q}\left(Q^1(\tau^1, a^1), \ldots, Q^N(\tau^n, a^n)\right) = Q\left(\max_{a^1} Q^1(\tau^1, a^1), \max_{a^2} Q^2(\tau^2, a^2), \ldots, \max_{a^N} Q_n(\tau^N, a^N)\right),$$

which is the specific form of (13) for QMIX. □

B. Proof of Theorem 2

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:*

$$|Q_{n+1}^{SMIX}(\boldsymbol{\tau}_t, \mathbf{a}_t) - Q_{n+1}^{Q(\lambda)}(\boldsymbol{\tau}_t, \mathbf{a}_t)| \leq \frac{\epsilon\gamma}{1 - \lambda\gamma} M. \tag{14}$$

Proof. First, we have,

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

Thus,

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

Therefore, the expression (14) holds. □

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 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 b episodes $\sim$ Uniform($\mathcal{D}$)
  Calculate λ-return targets $y_i^{tot} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$, where $G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^n \mathbb{E}_{\boldsymbol{\pi}} Q\left(\boldsymbol{\tau}_{t+n}, \mathbf{a}_{t+n}; \theta^-\right)$
  Update θ by minimizing $\sum_{t=1}^{T-1} \sum_{i=1}^{b} \left[(y_i^{tot} - Q_{tot}^{\boldsymbol{\pi}}(\boldsymbol{\tau}, \mathbf{a}; \theta))^2\right]$
  Replace target parameters $\theta^- \leftarrow \theta$ every C episodes
end for

C. Algorithm

D. Implementation Details

The agent network consists of a 64-dimensional GRU (Chung et al. 2014). 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 $Q^i(\tau^i, \cdot)$ 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. Agent's previous action is also concatenated to the input. The target network is updated after every 200 training episodes. We use RMSprop optimizer with learning rate 0.0005 and $\alpha = 0.99$ without weight decay or momentum during training. We perform independent ϵ-greedy for exploration. ϵ is annealed linearly from 1.0 to 0.05 across the first 50k timesteps for all experiments. The discount factor is set to $\gamma = 0.99$. The architecture of the mixing network is the same as in Rashid et al. (2018). Actually, all the settings of SMIX(λ)'s parameters are kept the same as those of QMIX for fair comparison, except that SMIX(λ) has an extra parameter λ and uses smaller buffer size.

E. Additional Results

We provide quantitative comparisons of our methods and their counterparts in Table 2.

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

Table 2: Mean, standard deviation, and median of test win rate percentages after training for 1 million timesteps in six different scenarios. The highest results are in bold. Our methods (SMIX(λ), SMIX(λ)-COMA, SMIX(λ)-VDN, SMIX(λ)-IQL) perform on par or significantly better than their counterparts in terms of the learning speed across all scenarios.