QOPT: Optimistic Value Function Decentralization for Cooperative Multi-Agent Reinforcement Learning

Kyunghwan Son
KAIST
kevinson9473@kaist.ac.kr

Sungsoo Ahn
KAIST
sungsoo.ahn@kaist.ac.kr

Roben Delos Reyes
KAIST
rddelosreyes@kaist.ac.kr

Jinwoo Shin
KAIST
jinwoos@kaist.ac.kr

Yung Yi
KAIST
yiyung@kaist.edu

Abstract

We propose a novel value-based algorithm for cooperative multi-agent reinforcement learning, under the paradigm of centralized training with decentralized execution. The proposed algorithm, coined QOPT, is based on the "optimistic" training scheme using two action-value estimators with separate roles: (i) true action-value estimation and (ii) decentralization of optimal action. By construction, our framework allows the latter action-value estimator to achieve (ii) while representing a richer class of joint action-value estimators than that of the state-of-the-art algorithm, i.e., QMIX. Our experiments demonstrate that QOPT newly achieves state-of-the-art performance in the StarCraft Multi-Agent Challenge environment. In particular, ours significantly outperform the baselines for the case where non-cooperative behaviors are penalized more aggressively.

1 Introduction

Recent advances in reinforcement learning (RL) have integrated deep learning techniques successfully; deep RL has already been applied to various tasks such as Atari games [1], robotic control [2], and Go [3, 4]. However, most of these successes are focused on controlling a single agent. The progress of multi-agent reinforcement learning (MARL) has been arguably slow despite its importance in many applications such as controlling robot swarms [5], packet routing [6], and autonomous driving [7]. One reason is that single-agent RL algorithms perform poorly when naïvely applied to multi-agent scenarios, *e.g.*, training agents using independent Q-learning [8–10]. The main challenge comes from the non-stationarity problem, where a small change on one agent's policy may cause another agent's policy to be sub-optimal.

Centralized training with decentralized execution (CTDE) has emerged as a popular approach to tackle this issue. It still considers the decentralization of the individual agents during execution but reasonably assumes them to be trained under a centralized scheme. To train agents under the CTDE paradigm, both policy-based [11–15] and value-based methods [16–20] have been proposed in the literature. Standard examples of policy-based methods are MADDPG [12] and COMA [11]. MADDPG learns individual policies in a centralized manner on both cooperative and competitive games with continuous action spaces. COMA trains individual policies with a joint critic and solves the credit assignment problem by estimating a counterfactual baseline.

On the other hand, value-based methods train a centralized joint action-value estimator which can be factorized into individual agent-wise utility functions. Value decomposition network (VDN) [16] represents the joint action-value estimator as a summation of the utility functions. QMIX [17] extends

Preprint. Under review.

VDN by employing a *mixing network* to express a non-linear monotonic relationship among the utility functions. The monotonic relationship allows the agents to jointly perform the optimal action while using only their locally optimal action. However, the assumption of monotonicity limits the representation complexity of QMIX, which may restrict the learning of optimal policies. QTRAN [18] has been proposed recently to eliminate the monotonic assumption in QMIX by using additional linear constraints between the utility functions and the joint action-value estimator. They prove that these linear constraints allow QTRAN to represent a richer class of joint action-value estimators than QMIX, while enabling computationally tractable maximization of the joint action-value estimator.

Comparing the methods on value function factorization in MARL, the class of joint action-value estimators that they can represent strictly increases, in the order of VDN, QMIX, and QTRAN. Intriguingly, however, several works have empirically shown that QTRAN actually performs worse than QMIX in complex MARL environments like the StarCraft Multi-Agent Challenge (SMAC) environment [20, 21]. This is unexpected since the monotonic relationship assumed by QMIX limits the space of action-value estimators able to be approximated without errors, while QTRAN is free of such an assumption. Our diagnosis tells us that the joint action-value estimator and loss functions of QTRAN are not scalable as the number of agents and the size of the action space increase. Moreover, there still exist rooms for (a) identifying the type of environments where QMIX underperforms and (b) developing an algorithm that can perform well in those environments.

Contribution. In this paper, we propose QOPT, a novel value-based MARL algorithm that achieves state-of-the-art performance, while being guaranteed to represent the largest class of decentralizable joint action-value estimators. We break down the problem of decentralized action-value estimation into two parts: (1) centralized action-value estimation and (2) decentralization of optimal action. To this end, we train two action-value networks to fulfill the role of each part. The role of the first network, referred to as *true action-value estimator,* is to estimate the action-value as accurately as possible with standard DQN training. The second network, which we call *optimistic action-value estimator,* is trained for the role of decentralizing the optimal action using an optimistic loss function under the monotonic constraint towards the true action-value estimator. In particular, we train the optimistic action-value estimator with an optimistic training scheme to bridge the gap between the true and the decentralized action-value estimators with tight relaxation. By splitting the roles of action-value estimation and action decentralization in two joint action-value estimators, we solve the structural limitations of QMIX while maintaining its tractable and efficient maximization. We demonstrate the performance of QOPT by comparing it against QMIX and QTRAN in the SMAC environment [21] which provides a variety of complex scenarios with partial observability.

In summary, the major contributions of this paper are as follows:

- We propose QOPT that handles a richer class of MARL tasks than the state-of-the art algorithm, *i.e.*, QMIX under the CTDE paradigm.
- We prove that our method can achieve decentralization of optimal actions for any decentralizable tasks, by introducing the true and the optimistic action-value estimators and training them using an "optimistic" loss function.
- Our experiments demonstrate that QOPT outperforms prior works in the SMAC environment. In particular, our approach significantly outperforms the baselines on tasks with negative rewards for receiving damage.

2 QOPT: Optimistic Value Function Decentralization

2.1 Problem statement

In this paper, we consider a decentralized partially observable Markov decision process [22] represented by a tuple $\mathcal{G} = \langle \mathcal{S}, \mathcal{U}, P, r, O, N, \gamma \rangle$. We let $s \in \mathcal{S}$ denote the true state of the environment. At each time step, an agent $i \in \mathcal{N} := \{1, ..., N\}$ selects an action $u_i \in \mathcal{U}$ as an element of the joint action vector $\boldsymbol{u} := [u_1, \cdots, u_N]$. It then goes through a stochastic transition dynamics described by the probability $P(s'|s, \boldsymbol{u})$. All agents share the same reward $r(s, \boldsymbol{u})$ which is discounted by the factor of γ. Each agent is associated with an individual partial observation $O(s, i)$ and an action-observation history τ_i. Finally, the concatenation of the individual action-observation histories is the overall action-observation history $\boldsymbol{\tau}$.

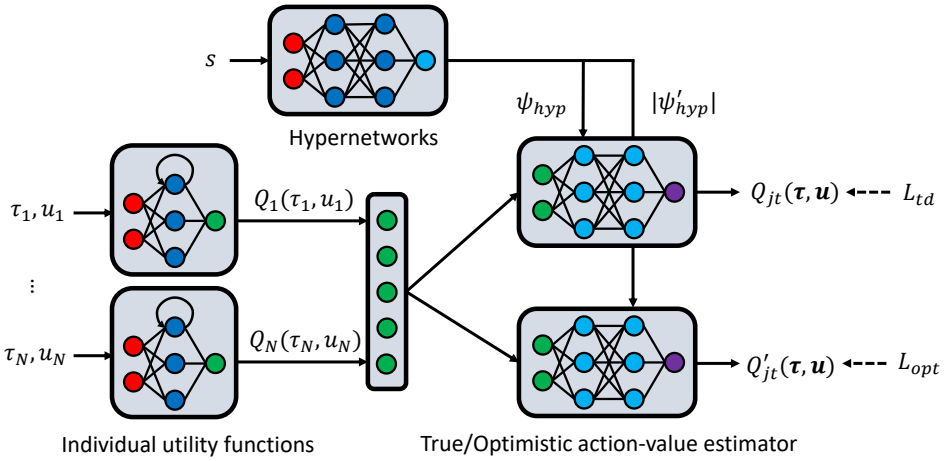


Figure 1: Architecture of QOPT

We aim to train agents individually under the paradigm of centralized training with decentralized execution, *i.e.*, we require that the information of the overall action-observation history is fully accessible during training, but we train the agents such that they can operate distributedly during execution. To this end, *decentralization* of the joint action-value estimator Q_{jt} is the key challenge. To be specific, we say that the joint action-value estimator Q_{jt} is decentralized into agent-wise utility functions $q_1, \ldots, q_N$ when the following condition is satisfied:

$$\arg\max_{\boldsymbol{u}} Q_{jt}(s, \boldsymbol{u}) = \left[\arg\max_{u_1} q_1(\tau_1, u_1), \ldots, \arg\max_{u_N} q_N(\tau_n, u_N)\right]. \tag{1}$$

In other words, once the optimal action that maximizes the joint action-value estimator is decentralized, the agents are able to get the optimal action-value during execution by simultaneously maximizing the agent-wise value functions without communicating with each other.

2.2 Overview of QOPT

In the rest of this section, we introduce our framework, coined QOPT, for the decentralization of the joint action-value estimator. Our approach is based on the idea of "optimistically" training two action-value estimators, parameterized by separate neural networks, with different roles: (i) true action-value estimation and (ii) decentralization of optimal action. We coin the first network as the *true action-value estimator* Q_{jt} to reflect how it is trained under the standard single-agent reinforcement learning algorithms without any compromise for achieving decentralization. Since the true action-value estimator is unrestricted, it can represent the widest class of functions possible for neural networks. The second network, coined *optimistic action-value estimator* Q'_{jt}, plays the role of being decentralized for training the individual utility functions.

The optimistic action-value estimator is trained to match its greedy action to that of the true action-value estimator so that it can predict the optimal action as accurately as possible. The challenge is that the optimistic action-value estimator with monotonic constraints has a limited capacity for following the true action-value estimator. Therefore, we propose an "optimistic" training objective that softly follows the true action-value estimator when the action-value of the optimistic action-value estimator is lower than a certain threshold. This allows the optimistic action-value estimator to follow the optimal action accurately while avoiding the waste of capacity for estimating the non-optimal action-values.

2.3 True and optimistic action-value estimators

We parameterize the true action-value estimator Q_{jt} and the optimistic action-value estimator Q'_{jt} using mixing networks $f_{mix}(\cdot; w)$, *i.e.*, a feed-forward network with parameter w that takes the individual utility functions $q_1, \ldots, q_N$ as inputs. Following Ha et al. [23] and Rashid et al. [17], we further introduce hypernetworks ψ_{hyp} and ψ'_{hyp} to generate the weights for the mixing networks corresponding to Q_{jt} and Q'_{jt}, respectively. Overall, the action-value estimators are expressed as

Algorithm 1 QOPT

Initialize replay memory $\mathcal{B} \leftarrow \emptyset$ and target parameters $\theta^- \leftarrow \theta$
for episode = 1 to M **do**
  Observe initial state $\boldsymbol{s}^0$ and observation $\boldsymbol{o}^0 = [O(\boldsymbol{s}^0, i)]_{i=1}^N$ for each agent i
  for $t = 1$ to T **do**
    With probability ϵ, select an action u_i^t
    Otherwise, set $u_i^t = \arg\max_{u_i^t} q_i(\tau_i^t, u_i^t)$ for each agent i
    Take action $\boldsymbol{u}^t$, and retrieve next state and reward (s^{t+1}, r^t)
    Store transition $(s^t, \boldsymbol{u}^t, r^t, s^{t+1})$ in $\mathcal{B}$
    Sample a transition $(s, \boldsymbol{u}, r, s')$ from $\mathcal{B}$
    Update θ by minimizing the losses L_{td} and L_{opt} from Equation (2) and (3), respectively:

$$L(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta}) = L_{td}(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta}) + \lambda_{opt} L_{\text{opt}}(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta})$$

    Update target network parameters $\theta^- \leftarrow \theta$ with period I
  end for
end for

follows:

$$Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) = f_{mix}(q_1(\tau_1, u_1), \ldots, q_N(\tau_N, u_N); \psi_{hyp}(s))$$
$$Q'_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) = f_{mix}(q_1(\tau_1, u_1), \ldots, q_N(\tau_N, u_N); |\psi'_{hyp}(s)|),$$

where $\psi_{hyp}(s)$ and $|\psi'_{hyp}(s)|$ are weights of the mixing network for Q_{jt} and Q'_{jt}, respectively. Note that the weights of the mixing network for Q'_{jt} are constrained to be non-negative to satisfy the following *monotonicity* condition:

$$\frac{\partial Q'_{jt}(s, \boldsymbol{u})}{\partial q_i(\tau_i, u_i)} \geq 0, \qquad \forall i \in \mathcal{N}.$$

Such a monotonicity condition allows the optimistic action-value estimator to be decentralized into utility functions [17]. The individual utility functions $q_1, \ldots, q_N$ are each represented by a deep recurrent Q-network (DRQN) [24]. At each step, agents receive their local observations as inputs and compute the individual utility values. We provide an illustration of the architecture in Figure 1.

2.4 Optimistic loss function

In QOPT, we set two goals for training. The first goal is to train the true action-value estimator to approximate the true action-value with standard temporal difference learning. The second one is to train the optimistic action-value estimator such that the optimistic and true action-value estimators have similar optimal actions. Following existing works, we use a replay buffer $\mathcal{B}$ to store and recycle the samples observed during training.

To be specific, we first update the true action-value estimator by using the following loss function:

$$\begin{aligned} L_{td}(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta}) &= \left(Q_{jt}(s, \boldsymbol{u}) - y_{dqn}(r, s'; \boldsymbol{\theta}^-)\right)^2, \\ y_{dqn}(r, s'; \boldsymbol{\theta}^-) &= r + \gamma Q_{jt}(s', \bar{\boldsymbol{u}}'; \boldsymbol{\theta}^-), \end{aligned} \tag{2}$$

where $\bar{\boldsymbol{u}}' = [\arg\max_{u_i} q_i(\tau_i', u_i; \boldsymbol{\theta}^-)]_{i \in \mathcal{N}}$ is the set of local optimal actions with respect to the individual utility functions, and $\boldsymbol{\theta}^-$ is the target network parameter periodically being updated by $\boldsymbol{\theta}$, as done by Mnih et al. [1].

Next, for training the optimistic action-value estimator Q'_{jt}, we use the following objective function:

$$\begin{aligned} L_{opt}(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta}) &= \begin{cases} (Q_{jt}(s, \boldsymbol{u}) - Q'_{jt}(s, \boldsymbol{u}))^2, & Q_{jt}(s, \boldsymbol{u}) \geq Q_{jt}(s, \bar{\boldsymbol{u}}), \\ (Q_{clip}(s, \boldsymbol{u}) - Q'_{jt}(s, \boldsymbol{u}))^2, & Q_{jt}(s, \boldsymbol{u}) < Q_{jt}(s, \bar{\boldsymbol{u}}), \end{cases} \\ Q_{clip}(s, \boldsymbol{u}) &= \texttt{clip}(Q'_{jt}(s, \boldsymbol{u}), Q_{jt}(s, \bar{\boldsymbol{u}}), Q_{jt}(s, \boldsymbol{u})), \end{aligned} \tag{3}$$

where local optimal actions $\bar{\boldsymbol{u}} = [\arg\max_{u_i} q_i(\tau_i, u_i; \boldsymbol{\theta})]_{i \in \mathcal{N}}$ and the function $\texttt{clip}(\cdot, \ell_1, \ell_2)$ bounds the output of the optimistic action-value estimator to be within the interval $[\ell_1, \ell_2]$.

Conceptually, the optimistic action-value estimator only follows the true action-value estimator exactly when the joint action-value of the true action-value estimator is larger than the threshold $Q_{jt}(s, \bar{\boldsymbol{u}})$. Since small joint action-values are softly ignored, problems with limited representation complexity in the monotonic network are alleviated while allowing the tractable maximization of the optimistic action-value estimator. This is similar to an optimistic agent proposed by Lauer and Riedmiller [9] that ignores the low target action-value when learning the optimal action with independent action-value estimators, which have representational limitations in a fully distributed setting. Now, we formally prove how minimization of the above losses leads to decentralization.

Theorem 1. *There exists a set of utility functions $\{q_i\}_{i\in\mathcal{N}}$ decentralizing the action-value estimator $Q_{jt}(s, \boldsymbol{u})$ if and only if there exists a function $Q'_{jt}(s, \boldsymbol{u})$ satisfying the following conditions:*

$$Q_{jt}(s, \bar{\boldsymbol{u}}) = Q'_{jt}(s, \bar{\boldsymbol{u}}), \qquad Q_{jt}(s, \bar{\boldsymbol{u}}) \geq Q'_{jt}(s, \boldsymbol{u}) \geq Q_{jt}(s, \boldsymbol{u}),$$
$$\frac{\partial Q_{jt}(s, \boldsymbol{u})}{\partial q_i(\tau_i, u_i)} \geq 0, \qquad \forall i \in \mathcal{N}, \qquad \bar{\boldsymbol{u}} = [\arg\max_{u_i} q_i(\tau_i, u_i; \boldsymbol{\theta})]_{i\in\mathcal{N}}.$$

We provide proof of Theorem 1 in Appendix A . Theorem 1 shows that our method can decentralize the optimal action of the true action-value estimator for any decentralizable tasks, assuming the estimators are powerful enough. Core of the proof for each sufficient and necessary condition is as follows: (i) We prove that the local optimal actions $\bar{\boldsymbol{u}}$ also maximize the true action-value estimator by utilizing the optimistic action-value estimator as the upper bound of the true action-value estimator. (ii) We define the parameterized monotonic function Q'_{jt} of the individual utility functions and prove there always exists parameters that satisfy the conditions for all decentralizable tasks.

Combining the two loss functions, we obtain the following objective, which is minimized in an end-to-end manner to train the true action-value estimator and the optimistic action-value estimator:

$$L(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta}) = L_{td}(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta}) + \lambda_{opt} L_{\text{opt}}(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta})$$

where r is the reward for action $\boldsymbol{u}$ at the state s transitioning to s', and $\lambda_{opt} > 0$ is hyperparameter controlling the importance of each loss function. We present the overall scheme in Algorithm 1.

3 Comparison to prior works: VDN, QMIX, and QTRAN

A considerable amount of works has been proposed to achieve the decentralization of the joint action-value estimator. In this section, we describe them one by one. We also provide a summary of the discussed frameworks, i.e., VDN, QMIX, and QTRAN, and our QOPT in Table 1. For a more detailed description that includes a comparison to other related works, see Appendix B.

VDN. Sunehag et al. [16] decomposes the joint action-value estimator into the summation of individual utility functions, i.e.,

$$Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{i\in\mathcal{N}} q_i(\tau_i).$$

Such a decomposition achieves the decentralization of the joint action-value estimator in the most straightforward way. However, VDN relies on the strong assumption that the joint action-value estimator is accurately approximated by the summation, leading to sub-optimal results.

QMIX. Following VDN, Rashid et al. [17] proposed QMIX which approximates the joint action-value estimator using an approximation monotonic with respect to the individual utility functions, expressed as follows:

$$\frac{\partial Q_{jt}(s, \boldsymbol{u})}{\partial q_i(\tau_i, u_i)} \geq 0, \qquad \forall i \in \mathcal{N}.$$

Once the monotonic relationship between the joint action-value estimator and the individual utility functions is satisfied, one can show that the decentralization of the joint action-value estimator is indeed achieved. QMIX learns a mixing network f_{mix} with non-negative weights from hypernetworks ψ_{hyp} [23] to satisfy the monotonocity condition. The hypernetworks allow the utilization of additional state information which is only observed during training.

QTRAN. Finally, Son et al. [18] looks at the value function factorization problem from a different angle. Instead of directly decomposing the joint action-value estimator into utility functions, QTRAN proposes a training objective which enforces the decentralization of the joint action-value estimator into the summation of individual utility functions as in Equation (1).

Table 1: Comparison with previous works with respect to the choice of parameterization on the true action-value estimation, optimal action decentralization, and loss functions. Here, q_i denotes the utility functions, f denotes an unconstrained neural network, $f_{mix}(q_1, \ldots, q_N; \psi_{hyp})$ denotes a mixing network with parameters generated from hypernetworks ψ_{hyp}, L_{td} denotes the loss function for the TD-error, and L_{qtran}, L_{opt} denote the action decentralization loss of QTRAN and QOPT, respectively.

Method	**True action-value estimation**	**Optimal action decentralization**	**Loss functions**
VDN	$\sum_i q_i$	$\sum_i q_i$	L_{td}
QMIX	$f_{mix}(q_1, \ldots, q_N; \lvert\psi_{hyp}\rvert)$	$f_{mix}(q_1, \ldots, q_N; \lvert\psi_{hyp}\rvert)$	L_{td}
QTRAN	f	$\sum_i q_i$	$L_{td} + \lambda L_{qtran}$
QOPT	$f_{mix}(q_1, \ldots, q_N; \psi_{hyp})$	$f_{mix}(q_1, \ldots, q_N; \lvert\psi'_{hyp}\rvert)$	$L_{td} + \lambda L_{opt}$

Among the existing works, QTRAN is arguably the most similar to ours since both frameworks attempt to propose a training objective for enforcing the decentralization of the joint action-value estimator instead of limiting the class of games that the joint action-value estimator can represent, as in VDN and QMIX. However, our QOPT further improves on QTRAN in several aspects. First, QOPT attempts to decentralize the joint action-value estimator into a mixing network consisting of individual utility functions, which is far more expressive than the summation of individual utility functions (as used in QTRAN). Second, our optimistic training scheme allows more stable training compared to that of QTRAN. Finally, we prove a refined version of the theorem for stating the necessary and sufficient condition for decentralizability of the joint action-value estimator.

4 Experiments

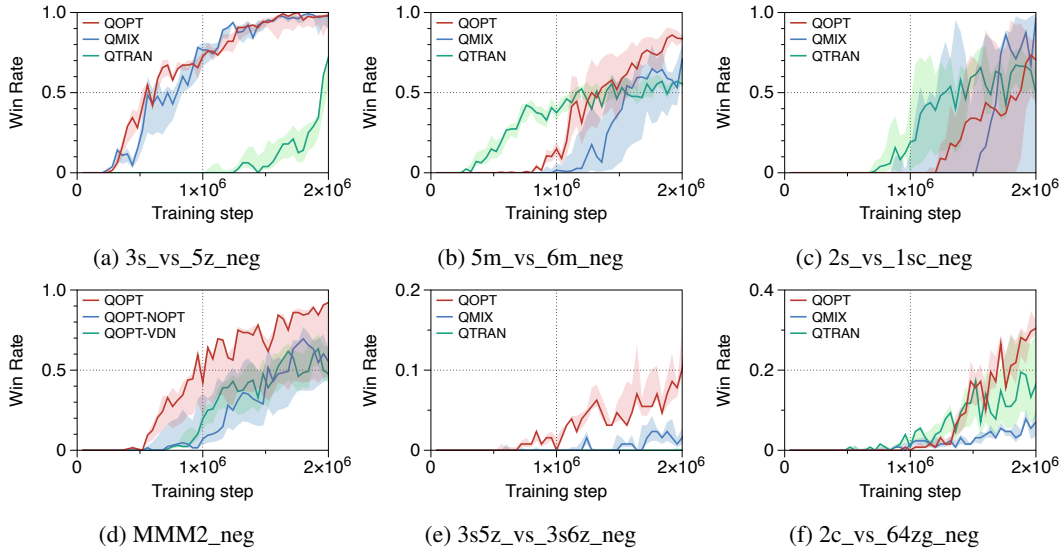
4.1 Experimental setup

To show the performance of QOPT, we use the StarCraft Multi-Agent Challenge (SMAC) environment [21] as our testbed. SMAC is a complex multi-agent environment used in many recent works, *e.g.*, Rashid et al. [17], Mahajan et al. [20] for evaluating state-of-the-art MARL methods. In this environment, multiple agents have their local observation and do not communicate in the execution phase. They are trained to solve combat scenarios against built-in scripted AI. The individual local observation contains distance, relative location, health, shield, and unit type of other allied and enemy units within their sight range, and there is a global state which is only available during the training phase. The global state vector contains the information on all agents in the map and the centralized trainer can use it during the centralized training.

Reward settings. For the reward, existing works run their algorithm on environments with only positive rewards that are based on the hit-point damage dealt. In our setting, we additionally test algorithms with a configured reward setting based on both hit-point damages dealt and received by agents. This makes the problem more challenging and practical than in original settings since each agent should learn more complex strategies to maximize the damage dealt while minimizing the damage received. We experimented with two *negative reward scale* P settings, where $P = 0, 0.5$. $P = 0$ produces a reward based only on the damage dealt to enemy units. When $P = 0.5$, a reward for the damage received from the opponents is added to the existing reward in a weighted manner. We represent the case of $P = 0.5$ by putting *_neg* in the map name. Appendix C contains additional experimental details.

We compare QOPT, QTRAN, and QMIX on several SMAC maps with 6 different scenarios which include easy, hard, and super hard levels classified by SMAC. Our evaluation procedure is also similar to SMAC. For every 10,000 time steps, we paused the training and run 32 test episodes without exploration for evaluation. After the evaluation, the percentage of episodes where the agents defeat all enemy units is referred to as the test win rate. All the results are averaged over at least 3 independent runs. Results are presented with median performance with shaded 25-75% confidence intervals.

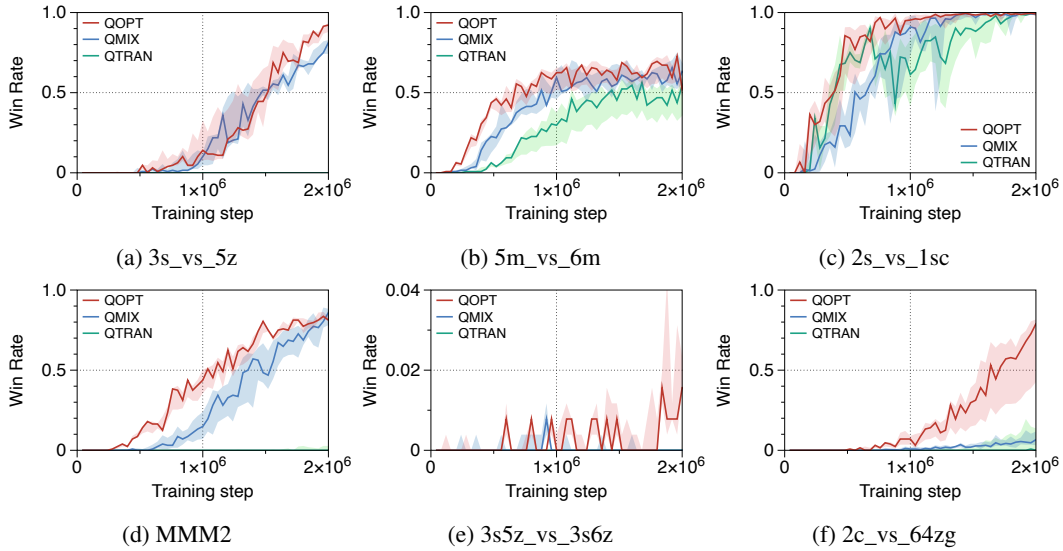
Ablation setup. In order to show the contributions of our double mixing networks and loss functions, we consider two ablation studies. First, we consider a comparison with an optimistic action-value estimator with a VDN-factored structure, which we call QOPT-VDN. It is designed to investigate whether the more expressive architecture is responsible for the performance. Next, we analyze the influence of optimistic training by comparing against QOPT-NOPT, which modifies QOPT by ensur-



(a) 3s_vs_5z_neg (b) 5m_vs_6m_neg (c) 2s_vs_1sc_neg

(d) MMM2_neg (e) 3s5z_vs_3s6z_neg (f) 2c_vs_64zg_neg

Figure 2: Average win rate with 25%-75% percentile for QOPT, QMIX, and QTRAN, where $P = 0.5$



(a) 3s_vs_5z (b) 5m_vs_6m (c) 2s_vs_1sc

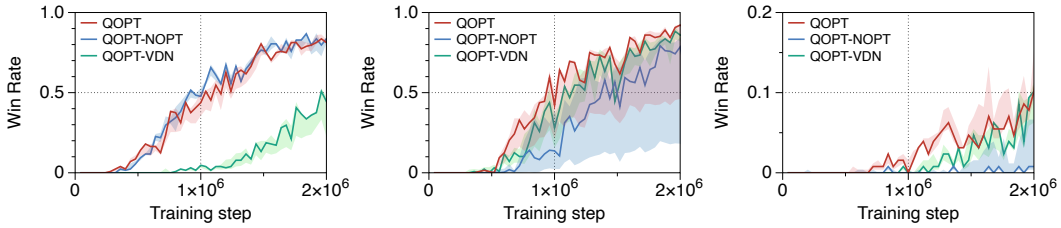
(d) MMM2 (e) 3s5z_vs_3s6z (f) 2c_vs_64zg

Figure 3: Average win rate with 25%-75% percentile for QOPT, QMIX, and QTRAN, where $P = 0$

ing the optimistic action-value estimators to accurately follow the true action-value estimators for all cases in Equation (3).

4.2 Results

Comparison with QMIX and QTRAN. Overall, QOPT achieves the highest win rate during training as shown in Figure 2 and Figure 3. First, when negative rewards exist, QOPT shows considerable gains on the tasks. Figure 2b shows QTRAN achieving higher performance than QOPT in the early stages of learning, but in the latter part, QOPT shows the best performance during training. In Figure 2d and 2e, QOPT shows significant performance improvements on the scenarios *MMM2_neg* and *2c_vs_64zg_neg*. However, one concern is that, in the case of *MMM2_neg*, depending on the initial coincidence, the agents may not learn meaningful policies at all. Especially in the most difficult scenario *3s5z_vs_3s6z_neg*, only QOPT learns meaningful policies. As shown in Figure 2a, QOPT shows the same performance as QMIX in the relatively easy scenario *3s_vs_5z_neg*. In Figure 2c, our algorithm has relatively poor performance. This is because the scenario *2s_vs_1sc_neg* has a hard exploration problem and the variance of the results is large.



(a) MMM2 (b) MMM2_neg (c) 3s5z_vs_3s6z_neg

Figure 4: Average win rate with 25%-75% percentile for QOPT, QMIX, and QTRAN

Compared to other baseline algorithms, QOPT shows fast learning speed and high performance, even if only positive rewards exist. Figure 3b, 3c, and 3d show that QOPT offers slightly faster learning than QMIX on the scenarios *5m_vs_6m*, *2s_vs_1sc* and *MMM2*. When learning difficulty increases, as shown in Figure 3f, only QOPT achieves a high win rate, and Figure 3e also shows that only QOPT achieves a non-zero win rate. On the other hand, QTRAN, which shows theoretical guarantees such as QOPT, shows that it does not learn meaningful policies for some scenarios such as *3s_vs_5z* and *MMM2*. These results show the gap in the process of converting theoretical proofs to loss functions of the neural networks.

Another interesting result is that negative rewards can help improve win rates. For example, as shown in Fig 2a and 3a, the presence of negative rewards can speed up learning. Furthermore, Figure 2b and 3b show QOPT learns a new kind of policy that has not been found in previous studies. This is because scenarios with negative rewards create more dense and direct rewards than conventional ones. However, negative rewards have the disadvantage of creating a new local optimal policy. If there are only positive rewards, the agents would have benefited from unconditionally fighting against the enemy. However, if there are negative rewards, running away without fighting might be an alternative way to maximize the rewards. The presence of these local alternative policies is likely to reduce performance by not satisfying the monotonic condition for QMIX, but our algorithm QOPT can solve these shortcomings of QMIX.

Ablation study. As shown in Figure 4, experiments in the three scenarios show how each element of QOPT affects its performance. First, Figure 4a shows that QOPT-QPT and QOPT have similar performance. Since MMM2 scenario only has positive rewards, QOPT-NOPT can learn optimal policies even though it is assuming a monotonic condition. The performance of QOPT-VDN, on the other hand, shows the importance of using the mixing network in the optimistic action-value estimators.

In contrast, Figure 4b and 4c show the need for the optimistic training due to the performance degradation of QOPT-NOPT where $P = 0.5$. As shown in Figure 4b, one interesting thing is that QOPT-NOPT performs better than QMIX, although it performs lower than QOPT. This is because the optimistic action-value estimator of QOPT-NOPT follows the centralized action-value estimator, which solves the overestimation problem. QMIX tends to overestimate the maximum Q-value for tasks that do not satisfy the monotonicity condition and the overestimation bias is accumulated during training. On the other hand, QOPT-NOPT has a limited representation complexity, such as QMIX, but there is no overestimation problem because it targets the centralized action-value estimator.

5 Conclusion

In this paper, we present QOPT, a novel value-based method for cooperative multi-agent reinforcement learning under the centralized training with decentralized execution paradigm. Unlike previous value function factorization methods, QOPT optimistically trains two separate action-value estimators, one for the role of true action-value estimation and the other for the role of optimal action decentralization. We theoretically and empirically demonstrate that our method handles a richer class of multi-agent reinforcement learning tasks.

To evaluate QOPT and other baselines, we use the StarCraft Multi-Agent Challenge (SMAC) environment, a standard benchmark for cooperative multi-agent reinforcement learning. Our results on SMAC show that our method performs well on most maps, and ablation studies demonstrate that both algorithmic and architectural advances of QOPT are crucial to its performance.

Broader Impact

It has been demonstrated that cooperative multi-agent reinforcement learning is an efficient framework for training agents to learn cooperative policies in multi-agent systems. As our algorithm falls under this framework, it can also be used by those who study these problems. In particular, it can be used in scenarios which cannot be solved by previous methods because our algorithm considers a wider range of practical cases.

Cooperative multi-agent reinforcement learning has mostly been tested on simulated environments. We have similarly demonstrated the strengths of our algorithm through simulations only. But when using these algorithms for real-world applications, such as network system optimization [6] and autonomous driving [7], it is more important to consider their robustness and safety than their capability to always maximize returns. An in-depth analysis on the safety and uncertainty of our algorithm must be additionally done before it can be applied to solve real-world problems.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[2] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arxiv preprint arXiv:1509.02971*, 2015.

[3] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[4] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[5] M. Yogeswaran, S. G. Ponnambalam, and G. Kanagaraj. Reinforcement learning in swarm-robotics for multi-agent foraging-task domain. In *Proceeding of IEEE Symposium on Swarm Intelligence (SIS)*, pages 15–21, 2013.

[6] Dayon Ye, Minji Zhang, and YU Yang. A multi-agent framework for packet routing in wireless sensor networks. *Sensors*, 15(5):10026-47, 2015.

[7] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[8] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of ICML*, pages 330–337, 1993.

[9] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of ICML*, 2000.

[10] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[11] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of AAAI*, 2018.

[12] Ryan Lowe, YI WU, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of NIPS*, pages 6379–6390, 2017.

[13] Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In *Proceedings of NeurIPS*, 2019.

[14] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of ICML*, 2019.

[15] Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. Multi-agent reinforcement learning with emergent roles. *arXiv preprint arXiv:2003.08039*, 2020.

[16] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinícius Flores Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of AAMAS*, pages 2085–2087, 2018.

[17] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of ICML*, 2018.

[18] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *Proceedings of ICML*, 2019.

[19] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv preprint arXiv:2002.03939*, 2020.

[20] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. In *Proceedings of NeurIPS*, 2019.

[21] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[22] Frans A Oliehoek, Christopher Amato, et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

[23] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. In *Proceedings of ICLR*, 2017.

[24] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *Proceedings of AAAI Fall Symposium Series*, 2015.

A Proofs

Theorem 1. *There exists a set of utility functions $\{q_i\}_{i\in\mathcal{N}}$ decentralizing the action-value estimator $Q_{jt}(s,\boldsymbol{u})$ if and only if there exists a function $Q'_{jt}(s,\boldsymbol{u})$ satisfying the following conditions:*

$$Q_{jt}(s,\bar{\boldsymbol{u}}) = Q'_{jt}(s,\bar{\boldsymbol{u}}), \tag{4}$$

$$Q_{jt}(s,\bar{\boldsymbol{u}}) \geq Q'_{jt}(s,\boldsymbol{u}) \geq Q_{jt}(s,\boldsymbol{u}), \tag{5}$$

$$\frac{\partial Q'_{jt}(s,\boldsymbol{u})}{\partial q_i(\tau_i,u_i)} \geq 0, \qquad \forall i \in \mathcal{N}, \tag{6}$$

$$\bar{\boldsymbol{u}} = [\arg\max_{u_i} q_i(\tau_i,u_i;\boldsymbol{\theta})]_{i\in\mathcal{N}}. \tag{7}$$

Proof. $\Longleftarrow$ We prove first the sufficiency of the theorem by showing that if the conditions hold, then $q_i(\tau_i,u_i)$ satisfies optimal decentralization $\arg\max_{\boldsymbol{u}} Q_{jt}(s,\boldsymbol{u}) = \bar{\boldsymbol{u}}$.

$$\begin{aligned} Q_{jt}(s,\bar{\boldsymbol{u}}) &= Q'_{jt}(s,\bar{\boldsymbol{u}}) \quad \text{(From (4))} \\ &\geq Q'_{jt}(s,\boldsymbol{u}) \quad \text{(From Monotonicity of } Q'_{jt}(s,\boldsymbol{u})\text{)} \\ &\geq Q_{jt}(s,\boldsymbol{u}). \quad \text{(From (5))} \end{aligned}$$

It means that the set of local optimal actions $\bar{\boldsymbol{u}}$ maximizes Q_{jt}, showing that q_i satisfies decentralizability.

$\Longrightarrow$ We turn now to the necessity. First, we define $Q'_{jt}(s,\bar{\boldsymbol{u}}) = \sum_{i=1}^{N} \alpha_i(q_i(\tau_i,u_i) - q_i(\tau_i,\bar{u}_i)) + Q_{jt}(s,\bar{\boldsymbol{u}})$, which satisfies condition (6), where constant $\alpha_i \geq 0$. By definition, (4) and the upper bound condition of (5) naturally hold, and proof for lower bound condition of (5) follows from the fact that there exists $[\alpha_i]$ small enough such that

$$Q'_{jt}(s,\boldsymbol{u}) - Q_{jt}(s,\boldsymbol{u}) = \sum_{i=1}^{N} \alpha_i(q_i(\tau_i,u_i) - q_i(\tau_i,\bar{u}_i)) - (Q_{jt}(s,\boldsymbol{u}) - Q_{jt}(s,\bar{\boldsymbol{u}})) \geq 0,$$

since $(q_i(\tau_i,u_i) - q_i(\tau_i,\bar{u}_i)), (Q_{jt}(s,\boldsymbol{u}) - Q_{jt}(s,\bar{\boldsymbol{u}})) < 0$ if $\boldsymbol{u} \neq \bar{\boldsymbol{u}}$. □

B Related Work

Centralized training with decentralized execution (CTDE) has emerged as a popular paradigm under the multi-agent reinforcement learning framework. It assumes the complete state information to be fully accessible during training, while individual policies allow decentralization during execution. To train agents under the CTDE paradigm, both policy-based [11–15] and value-based methods [16–19] have been proposed. At a high-level, the policy-based methods rely on the actor-critic framework with independent actors to achieve decentralized execution. On the other hand, the value-based methods attempt to learn a joint action-value estimator which can be cleverly decomposed into individual agent-wise utility functions.

For examples of the policy-based methods, COMA [11] trains individual policies with a joint critic and solves the credit assignment problem by estimating a counterfactual baseline. MADDPG [12] extends the DDPG [2] algorithm to learn individual policies in a centralized manner on both cooperative and competitive games. MAAC [14] includes an attention mechanism in critics to improve scalability. LIIR [13] introduces a meta-gradient algorithm to learn individual intrinsic rewards to solve the credit assignment problem. Recently, ROMA [15] proposes a role-oriented framework to learn roles via deep RL with regularizers and role-conditioned policies.

Among the value-based methods, value decomposition network (VDN) [16] learns a centralized, yet factored joint action-value estimator by representing the joint action-value estimator as a sum of individual agent-wise utility functions. QMIX [17] extends VDN by employing a *mixing network* to express a non-linear monotonic relationship among individual agent-wise utility functions in the joint action-value estimator. Qatten [19] introduces a multi-head attention mechanism for approximating the decomposition of the joint action-value estimator, which is based on theoretical findings. MAVEN [20] proposes a committed exploration algorithm to address the limitations of QMIX with regards to exploration.

QTRAN. Finally, QTRAN [18] has been proposed recently to eliminate the monotonic assumption on the joint action-value estimator in QMIX. Instead of directly decomposing the joint action-value estimator into utility functions, QTRAN proposes a training objective which enforces the decentralization of the joint action-value estimator into the summation of individual utility functions as in Equation (1).

Namely, they propose to minimize the combination of the following loss functions:

$$
\begin{aligned}
L_{\text{td}}(s, \boldsymbol{u}, r, s'; \boldsymbol{\theta}) &= \big(Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) - y_{\text{dqn}}(r, \boldsymbol{\tau}'; \boldsymbol{\theta}^-)\big)^2, \\
L_{\text{opt}}(s, \boldsymbol{u}, r, s'; \theta) &= \big(Q'_{jt}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) - \hat{Q}_{jt}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) + V_{jt}(\boldsymbol{\tau})\big)^2, \\
L_{\text{nopt}}(s, \boldsymbol{u}, r, s'; \theta) &= \big(\min[Q'_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) - \hat{Q}_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{jt}(\boldsymbol{\tau}), 0]\big)^2, \\
V_{jt}(\boldsymbol{\tau}) &= \max_{\boldsymbol{u}} Q_{jt}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i \in \mathcal{N}} q_i(\tau_i, \bar{u}_i).
\end{aligned}
$$

Here, the value $V_{\text{jt}}(\boldsymbol{\tau})$ corrects for the discrepancy between the centralized joint action-value function Q_{jt} and the sum of individual joint action-value functions $[Q_i]$. There are some differences between the loss functions of QTRAN and QOPT. QTRAN uses separate loss functions to learn the optimal actions and the non-optimal actions. For the non-optimal actions, Q'_{jt} is only learned through a lower bound condition that Q'_{jt} should be greater than the true action-value estimator Q_{jt}. Furthermore, it only follows Q_{jt} exactly for the optimal actions. However, with our loss functions, the optimistic action-value estimator learns to follow the true action-value estimator equally, even for non-optimal actions, if its values are greater than a certain threshold. In addition, our optimistic loss function has both lower bound and upper bound conditions for non-optimal actions, which is a tighter condition making learning more efficient. In the QTRAN paper, a new algorithm called QTRAN-alt is proposed to solve the problem that occurs when only the lower bound condition of the QTRAN algorithm exists. However, this QTRAN-alt algorithm has high computational complexity because it requires counterfactual action-value estimation when other actions are selected. In addition, Mahajan et al. [20] experimentally shows that QTRAN-alt does not work as well as QTRAN-base in SMAC [21] environments. We effectively solve this problem with an upper bound condition that does not require the counterfactual action-value estimation.

C Experimental details

The hyperparameters of training and testing configurations are the same as in the GitHub code of SMAC [21]. The architecture of all agents' policy networks is a DRQN consisting of two 64-dimensional fully connected layers and 64-dimensional GRU. The mixing networks consist of a single hidden layer with 32 hidden widths and ELU activation functions. Hypernetworks consist of two layers with 64 hidden widths and ReLU activation functions.

All neural networks are trained using the RMSProp optimizer with 0.0005 learning rates, and we use ϵ-greedy action selection with decreasing ϵ from 1 to 0.05 over 50000 time steps for exploration. For the discount factor, we set $\gamma = 0.99$. The replay buffer size is 5000 episodes and the minibatch size is 32. Using Nvidia Titan Xp graphic cards, the training time is about 8 hours to 24 hours, depending on the scenario.

We use the double Q-learning algorithm for practical implementation, following the SMAC paper, and our method additionally applies this idea with an optimistic loss. We redefine it as local optimal actions $\bar{\boldsymbol{u}} = [\arg\max_{u_i} q_i(\tau_i, u_i; \boldsymbol{\theta}^-)]_{i \in \mathcal{N}}$ with target network that parameterized by $\boldsymbol{\theta}^-$, and we see that it practically increases stability for training. As another idea, we configure the true action-value estimator as the summation of two mixing networks, where one mixing network is monotonic and the other is non-monotonic. This true action-value estimator accelerates training for tasks that satisfy the monotonic assumption while still having the full representation complexity through the non-monotonic mixing network.