REVISITING THE MONOTONICITY CONSTRAINT IN COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

QMIX, a popular MARL algorithm based on the monotonicity constraint, has been used as a baseline for the benchmark environments, such as Starcraft Multi-Agent Challenge (SMAC), Predator-Prey (PP). Recent variants of QMIX target relaxing the monotonicity constraint of QMIX to improve the expressive power of QMIX, allowing for performance improvement in SMAC. However, we find that such performance improvements of the variants are significantly affected by various implementation tricks. In this paper, we revisit the monotonicity constraint of QMIX, (1) we design a novel model RMC to further investigate the monotonicity constraint; the results show that monotonicity constraint can improve sample efficiency in some purely cooperative tasks; (2) we then re-evaluate the performance of QMIX and these variants by a grid hyperparameter search for the tricks; the results show QMIX achieves the best performance among them, achieving SOTA performance on SMAC and PP; (3) we analyze the monotonic mixing network from a theoretical perspective and show that it can represent any tasks which can be interpreted as purely cooperative. These analyses demonstrate that relaxing the monotonicity constraint of the mixing network will not always improve the performance of QMIX, which breaks our previous impressions of the monotonicity constraints.

1 INTRODUCTION

Multi-agent cooperative games have many complex real-world applications such as, robot swarm control [7; 34; 14], autonomous vehicle coordination [3; 38], and sensor networks [36], a complex task always requires multi-agents to accomplish together. Multi-Agent Reinforcement Learning (MARL), is used to solve the multi-agent systems tasks [34].

In multi-agent systems, a typical challenge is a limited scalability and inherent constraints on agent observability and communication. Therefore, decentralized policies that act only on their local observations are necessitated and widely used [37]. Learning decentralized policies is an intuitive approach for training agents independently. However, simultaneous exploration by multiple agents often results in non-stationary environments, which leads to unstable learning. Therefore, *Centralized Training and Decentralized Execution* (CTDE) [10] allows for independent agents to access additional state information that is unavailable during policy inference.

Many CTDE learning algorithms have been proposed for the better sample efficiency in cooperative tasks[33]. Among them, several value-based approaches achieve state-of-the-art (SOTA) performance [19; 30; 35; 20] on such benchmark environments, e.g., Starcraft Multi-Agent Challenge (SMAC) [21], Predator-Prey (PP) [2; 16]. To enable effective CTDE for multi-agent Q-learning, the Individual-Global-Max (IGM) principle [23] of equivalence of joint greedy action and individual greedy actions is critical. The primary advantage of the IGM principle is that it ensures consistency of policy with centralized training and decentralized execution. To ensure IGM principle, QMIX [19] was proposed for factorizing the joint action-value function with the *Monotonicity Constraint* [30], however, limiting the expressive power of the mixing network.

To improve the performance of QMIX, some variants of QMIX [1], including value-based approaches [35; 20; 30; 24] and a policy-based approach [37], have been proposed with the aim to relax the monotonicity constraint of QMIX. However, while investigating the codes of these variants, we find that their performance is significantly affected by their implementation tricks. Therefore, it is left unclear whether monotonicity constraint indeed impairs the QMIX's performance.

In this paper, we investigate the *monotonicity constraint* and *implementation tricks* (Appendix B) in cooperative MARL. (1) Firstly, we propose a novel method, RMC, for studying the impact of monotonicity constraints in the some purely cooperative tasks, i.e, SMAC and Predator-Prey. The experimental results show that monotonicity constraint significantly improves the performance of RMC in SMAC and PP, and RMC outperforms the previous policy-based algorithms. (2) Next, we re-test the performance of QMIX and its variants by a grid hyperparameter search for the tricks; and the results show that the Fine-tuned QMIX can solve almost all hard scenarios of SMAC, achieving SOTA performance. (3) Then, we discuss the properties of monotonicity constraints from a theoretical perspective; and we prove that QMIX can represent any purely cooperative tasks.

All these results show that relaxing the monotonicity constraint of the mixing network will not always improve the performance of QMIX; and **the monotonicity constraint works well in multi-agent tasks which can be interpreted as purely cooperative**, even if the task can also be interpreted as competitive.

2 Background

Dec-POMDP. We model a multi-agent cooperative task as decentralized partially observable Markov decision process (Dec-POMDP) [15], which composed of a tuple $G = \langle \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, N, \gamma \rangle$. $s \in S$ describes the true state of the environment. At each time step, each agent $i \in \mathcal{N} := \{1, \ldots, N\}$ chooses an action $u^i \in \mathcal{U}$, forming a joint action $\boldsymbol{u} \in \mathcal{U}^N$. All state transition dynamics are defined by function $P\left(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{u}\right) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \mapsto [0,1]$. Each agent has independent observation $z \in \mathcal{Z}$, determined by observation function $O(s,i) : \mathcal{S} \times \mathcal{N} \mapsto \mathcal{Z}$. All agents share the same reward function $r(s, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \to \mathbb{R}$ and $\gamma \in [0,1)$ is the discount factor. The objective function, shown in Eq. 1, is to maximize the joint value function to find a joint policy $\boldsymbol{\pi} = \langle \pi_1, ..., \pi_n \rangle$.

$$J(\pi) = \mathbb{E}_{u^1 \sim \pi^1, \ldots, u^N \sim \pi^N, s \sim T}\left[\sum_{t=0}^{\infty} \gamma_t r_t\left(s_t, u_t^1, \ldots, u_t^N\right)\right] \tag{1}$$

Centralized Training and Decentralized Execution (CTDE). To resolve the non-stationary problem for MARL, CTDE is a popular paradigm [30] which allows for the learning process to utilize additional state information [10]. Agents are trained in a centralized way, i.e., learning algorithms, to access all local action observation histograms, global states, and sharing gradients and parameters. In the execution stage, each individual agent can only access its local action observation history τ^i.

QMIX and Monotonicity Constraint. As a popular CTDE algorithm in cooperative MARL, QMIX [19] learns a joint action-value function Q_{tot} which can be represented in Eq. 2,

$$\begin{aligned} Q_{tot}(s, \boldsymbol{u}; \boldsymbol{\theta}, \phi) = & g_\phi\left(s, Q_1\left(\tau^1, u^1; \theta^1\right), \ldots, Q_N\left(\tau^N, u^N; \theta^N\right)\right) \\ & \frac{\partial Q_{tot}(s, \boldsymbol{u}; \boldsymbol{\theta}, \phi)}{\partial Q_i\left(\tau^i, u^i; \theta^i\right)} \geq 0, \quad \forall i \in \mathcal{N} \end{aligned} \tag{2}$$

where ϕ is the trainable parameter of the monotonic mixing network, which is a mixing network with monotonicity constraint, and θ^i is the parameter of the agent network i. Benefiting from the monotonicity constraint in Eq. 2, maximizing joint Q_{tot} is precisely the equivalent of maximizing individual Q_i, resulting in and allowing for optimal individual action to maintain consistency with optimal joint action. QMIX learns by sampling a multitude of transitions from the replay buffer and minimizing the mean squared temporal-difference (TD) error loss:

[1]These algorithms are based on the mixing network from QMIX, so we call the variants of QMIX.

$$\mathcal{L}(\theta)=\frac{1}{2}\sum_{i=1}^{b}\left[\left(y_i-Q_{tot}(s,u;\theta,\phi)\right)^2\right] \tag{3}$$

where the TD target value $y=r+\gamma\max_{u'}Q_{tot}\left(s',u';\theta^-,\phi^-\right)$ and θ^-, ϕ^- are the target network parameters copied periodically from the current network and kept constant for a number of iterations. However, the monotonicity constraint limits the mixing network's expressiveness, which may fail to

12	-12	-12
-12	0	0
-12	0	0

(a) Payoff matrix

-12	-12	-12
-12	0	**0**
-12	0	0

(b) QMIX: Q_{tot}

Table 1: A non-monotonic matrix game. Bold text indicates the reward of the argmax action.

learn in non-monotonic cases [12] [20]. Table 1a shows a non-monotonic matrix game that violates the monotonicity constraint. This game requires both robots to select the first action 0 (actions are indexed from top to bottom, left to right) in order to catch the reward 12; if only one robot selects action 0, the reward is -12. QMIX may learn an incorrect Q_{tot} which has an incorrect argmax action as shown in Table 1b.

3 Related Works

In this section, we introduce these variants of QMIX; and we provide the details of these algorithms in Appendix E.

Value-based Methods To enhance the expressive power of QMIX, Qatten [35] introduces an attention mechanism to enhance the expression of QMIX; QPLEX [30] transfers the monotonicity constraint from Q values to Advantage values [13]; QTRAN++ [24] and WQMIX [20] further relax the monotonicity constraint through a true value network and some theoretical constraints; however, Value-Decomposition Networks (VDNs) [28] only requires a linear decomposition where $Q_{tot}=\sum_i^N Q_i$, which can be seen as strengthening the monotonicity constraint.

Policy-based Methods LICA [37] completely removes the monotonicity constraint through a policy mixing critic. For other MARL policy-based methods, DOP [31] learns the policy networks using the Counterfactual Multi-Agent Policy Gradients (COMA) [6] with the Q_i decomposed by QMIX.

To improve the efficiency of QMIX under parallel training [2], VMIX [26] combines the Advantage Actor-Critic (A2C) [25] with QMIX to extend the monotonicity constraint to value networks, i.e., replacing the value network with the monotonic mixing network, as shown in Figure 1 and Eq. 4.

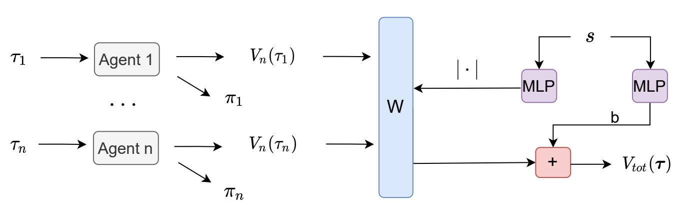


Figure 1: Architecture for VMIX: $|\cdot|$ denotes **absolute value operation**, decomposing V_{tot} into V_i.

$$\begin{gathered}V_{tot}(s;\boldsymbol{\theta},\phi)=g_\phi\left(s,V^1\left(\tau^1;\theta^1\right),\ldots,V^N\left(\tau^N;\theta^N\right)\right)\\ \frac{\partial V_{tot}}{\partial V^i}\geq 0,\quad \forall i\in\mathcal{N}\end{gathered} \tag{4}$$

[2]We find that this problem can be solved by training QMIX with Adam [8]

where ϕ is the parameter of value mixing network, and θ_i is the parameter of agent network. With the centralized value function V_{tot}, the policy networks can be trained by policy gradient (Eq. 5),

$$\hat{g}_i = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_{\theta^i}\left(u_t^i \mid \tau_t^i\right)\Big|_{\theta^i} \hat{A}_t \tag{5}$$

where $\hat{A}_t = r + V_{tot}(s_{t+1}) - V_{tot}(s_t)$ is the advantage value function [13], and $\mathcal{D}$ denotes sampled trajectories. At last, we briefly describe the properties of these algorithms in Table 2.

Algorithms	Type	Attention	Monotonic Constraint Strength	Off-policy
VDNs	Value-based	No	Very Strong	Yes
QMIX	Value-based	No	Strong	Yes
Qatten	Value-based	Yes	Strong	Yes
QPLEX	Value-based	Yes	Medium	Yes
WQMIX	Value-based	No	Weak	Yes
VMIX	Policy-based	No	Strong	No
LICA	Policy-based	No	No	No
RMC	Policy-based	No	Strong	Yes

Table 2: Properties of coopertive MARL algorithms. The analysis of the monotonicity constraint strength is in the Appendix E.3.

All these algorithms show that their performance exceeds QMIX in SMAC, yet we find that they do not consider the impact of various code-level optimizations (Appendix B) in the implementations. **Moreover, the performance of these algorithms is not even consistent in these papers.** For example, in papers [30] and [31], QPLEX and DOP outperform QMIX, while in paper [16], both QPLEX and DOP underperform QMIX.

4 RMC

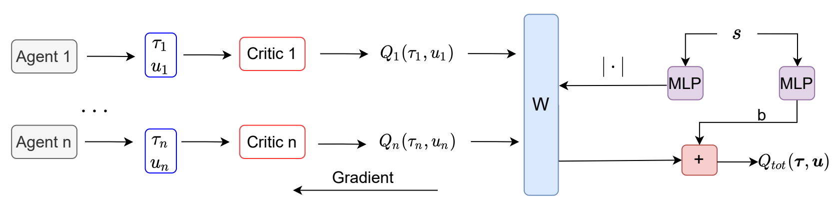


Figure 2: Architecture for RMC: $|\cdot|$ **denotes absolute value operation**, implementing the monotonicity constraint of QMIX. W denotes the non-negative mixing weights. Agent i denotes the policy network which can be **trained end-to-end by maximizing the** Q_{tot}

To study the impact of monotonicity constraint in pratical multi-agent tasks, we propose an novel end-to-end Actor-Critic method, called **RMC**. Specifically, we use the monotonic mixing network as a critic network, shown in Figure 2. Then, in Eq. 6, with a trained critic $Q^\pi_{\theta_c}$ estimate, the decentralized policy networks $\pi^i_{\theta_i}$ can then be optimized end-to-end simultaneously by maximizing $Q^\pi_{\theta_c}$ with the policies $\pi^i_{\theta_i}$ as inputs; and the $\mathbb{E}_i\left[\mathcal{H}\left(\pi^i_{\theta_i}\left(\cdot \mid z^i_t\right)\right)\right]$ is the Adaptive Entropy [37]. We use a novel two-stage approach to train the actor-critic network of RMC, as shown in Algo. 1.

$$\max_\theta \mathbb{E}_{t, s_t, u_t^1, \ldots, \tau_t^n}\left[Q^\pi_{\theta_c}\left(s_t, \pi^1_{\theta_1}\left(\cdot \mid \tau_t^1\right), \ldots, \pi^n_{\theta_n}\left(\cdot \mid \tau_t^n\right)\right) + \mathbb{E}_i\left[\mathcal{H}\left(\pi^i_{\theta_i}\left(\cdot \mid \tau_t^i\right)\right)\right]\right] \tag{6}$$

As the monotonicity constraint on the critic (Figure 2) is theoretically no longer required as the critic is not used for greedy action selection. RMC can swith to non-monotonic mode by removing the absolute value operation in the monotonic mixing network. In this way, RMC can also be easily extended to non-monotonic tasks. Beside, since RMC is trained end-to-end, it can also be used for continuous control tasks.

5 EXPERIMENTS SETUP

In this section we first introduce the environments and the evaluation criteria for our experiments.

5.1 BENCHMARK ENVIRONMENT

These environments include the purely cooperative tasks, i.e, SMAC and DEPP; and the non-monotonic matrix games.

StarCraft Multi-Agent Challenge (SMAC) is used as our main benchmark testing environment, which is a ubiquitously-used multi-agent cooperative control environment for MARL algorithms [30; 19; 24; 20]. SMAC consists of a set of StarCraft II micro battle scenarios, whose goals are for allied agents to defeat enemy agents, and it classifies micro scenarios into *Easy*, *Hard*, and *Super Hard* levels. QMIX and VDNs achieves a 0% win rate in Super Hard scenarios such as, $corridor$, $3s5z\_vs\_3s5z$, and $6h\_vs\_8z$ [21]. SMAC mainly uses a shaped reward signal calculated from the hit-point damage dealt, some positive reward after having enemy units killed and a positive bonus for winning the battle; Intuitively, these positive rewards can be interpreted as purely cooperative.

Difficulty-Enhanced Predator-Prey (DEPP) In vanilla Predator-Prey (PP) [11], three cooperating agents control three predators to chase a faster robot prey (the prey acts randomly). The goal is to capture the prey with the fewest steps possible. We leverage two difficulty-enhanced Predator-Prey variants to test the algorithms: (1) the first Discrete Predator-Prey (Discrete PP) [2] requires two predators to catch the prey at the same time to get a reward; (2) In the Continuous Predator-Prey (Continuous PP), the prey's policy is replaced by a hard-coded heuristic, that, at any time step, moves the prey to the sampled position with the largest distance to the closest predator. DEPPs only reward the predators when they catche preys, so the DEPPs can also be considered as purely cooperative tasks.

We explain in detail in Sec. 7.2 why SMAC and DEPP can be interpreted as purely cooperative tasks.

Non-monotonic Matrix Game We evaluate performance of the algorithm in competitive cases in two non-monotonic matrix games from [23] and (b) [12], shown in Sec. 6.4.

5.2 PARALLEL SAMPLING

To quickly sample from the complex environments, **8 rollout processes** for parallel sampling are used for SMAC and Discrete PP; and **4 rollout processes** are used for Continuous PP. Specifically, our experiments collect **10 million samples** within 9 hours with a Core i7-7820X CPU and a GTX 1080 Ti GPU in SMAC. This also ensures that we have enough samples to evaluate the convergence performance of the algorithms.

5.3 EVALUATION METRIC

Our primary evaluation metric is the function that maps the steps for the environment observed throughout the training to the median winning percentage (episode return for Predator-Prey) of the evaluation. Just as in QMIX [19], we repeat each experiment with several independent training runs (five independent random experiments).

6 EXPERIMENTS

In this section, we first study the effects of the monotonicity constraint in purely cooperative tasks with RMC and VMIX. Next, as the past studies evaluate the performance of QMIX's variants with inconsistent implementation tricks, we retested their performance based on the normalized tricks. Then, we also study the monotonicity constraint in two non-monotonic matrix games.

6.1 ABLATION STUDY OF MONOTONICITY CONSTRAINT

Since our proposed algorithm RMC can easily switch between monotonic and non-monotonic modes, we can evaluate the effects of monotonicity constraints in practical tasks effectively. The ablation experiments in Figure 3 demonstrates that the monotonicity constraint significantly improves the performance of RMC in SMAC and Continuous PP. To explore the generality of monotonicity constraints, we extend the ablation experiments to VMIX [27]. We already know that VMIX adds the monotonicity constraint to the value network of A2C; and it learns the decentralized policies by advantage-based policy gradient (Sec. 2). Therefore, the monotonicity constraint is not necessary for greedy action selection for VMIX either. We can evaluate the effects of the monotonicity constraint by removing the absolute value operation in Figure 1. The ablation experiment in Figure 4 shows that the monotonicity constraint also improves the sample efficiency in value networks.

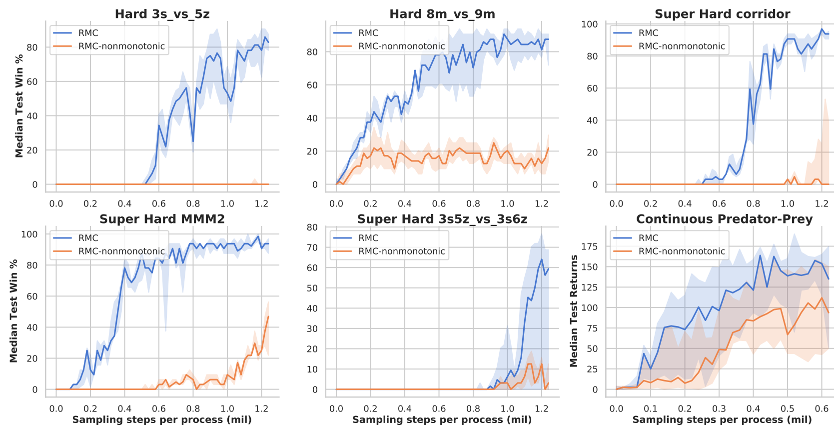


Figure 3: Comparing RMC w./ and w./o. monotonicity constraint (remove absolute value operation) on SMAC and Continuous Predator-Prey.

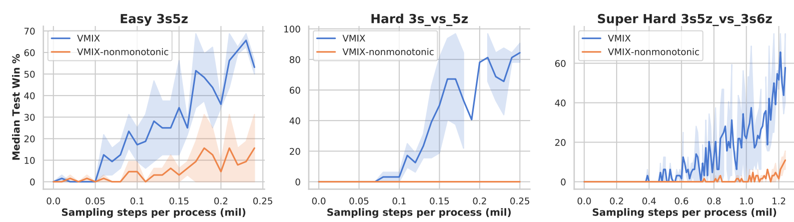


Figure 4: Comparing VMIX with and without monotonicity constraint on SMAC.

The above experimental results indicate that the monotonicity constraint can improve the sample efficiency in some purely cooperative tasks, such as SMAC and DEPP.

6.2 RE-EVALUATION

We then normalize the mainly tricks for all these algorithms for the re-evaluation, i.e, we perform grid search schemes on a typical hard environment (5m_vs_6m) and super hard environment (3s5z_vs_3s6z) to find **a general set of** hyperparameters for each algorithm (details in Appendix C). As shown in Table 3 [3], the test results on the hardest scenarios in SMAC and DEPP demonstrate that, (1) The performance of values-based methods and VMIX with normalized tricks exceeds the test results in the past literatures [21; 30; 16; 20; 27] (details in Appendix D.2). (2) **QMIX outpeforms all its variants**. (3) The linear VDNs is also relatively effective. (4) The performance of the algorithm becomes progressively worse as the monotonicity constraint decreases (QMIX > QPLEX > WQMIX and RMC, **VMIX** > LICA, details in Appendix E.3) in the benchmark environment.

The experimental results, specifically (2), (3) and (4), show that these variants of QMIX that relax the monotonicity constraint do not obtain better performance than QMIX in some purely cooperative tasks, either SMAC or DEPP.

[3]Note that our experimental results (we use StarCraft 2, SC2.4.10) are not always comparable with the previous works , which use SC2.4.6.

Scenarios	Difficulty	Value-based					Policy-based			
		QMIX	VDNs	Qatten	QPLEX	WQMIX	LICA	VMIX	DOP	RMC
2c_vs_64zg	Hard	**100%**	**100%**	**100%**	**100%**	93%	**100%**	98%	84%	**100%**
8m_vs_9m	Hard	**100%**	**100%**	**100%**	95%	90%	48%	75%	96%	95%
3s_vs_5z	Hard	**100%**	**100%**	**100 %**	**100%**	**100%**	3%	96%	**100%**	96%
5m_vs_6m	Hard	**90%**	**90%**	**90%**	**90%**	**90%**	53%	9%	63%	67%
3s5z_vs_3s6z	S-Hard	**75%**	43%	62%	68%	6%	0%	56%	0%	**75%**
corridor	S-Hard	**100%**	98%	**100%**	96%	96%	0%	0%	0%	**100%**
6h_vs_8z	S-Hard	84%	**87%**	82%	78%	78%	4%	80%	0%	19%
MMM2	S-Hard	**100%**	96%	**100%**	**100%**	23%	0%	70%	3%	**100%**
27m_vs_30m	S-Hard	**100%**	**100%**	**100%**	**100%**	0%	9%	93%	0%	93%
Discrete PP	-	**40**	39	-	39	39	30	39	38	38
Avg. Score	(Hard+)	**94.9%**	91.2%	92.7%	92.5%	67.4%	29.2%	67.4%	44.1%	84.0%

Table 3: Median test winning rate (episode return) of MARL algorithms with normalized tricks. S-Hard denotes Super Hard. We compare their performance in the most difficult scenarios of SMAC and the Discrete PP.

6.3 Finetuned-QMIX

Next, we perform a hyperparameter search for QMIX **for each scenario** of SMAC (Appendix. C). As shown in Table 4, the Finetuned-QMIX attains extraordinary high win rates in all hard and super hard SMAC scenarios, far exceeding vanilla QMIX.

Senarios	Difficulty	QMIX (batch size=128)	Finetuned-QMIX
2s_vs_1sc	Easy	**100%**	**100%**
2s3z	Easy	**100%**	**100%**
1c3s5z	Easy	**100%**	**100%**
3s5z	Easy	**100%**	**100%**
10m_vs_11m	Easy	98%	**100%**
8m_vs_9m	Hard	84%	**100%**
5m_vs_6m	Hard	84%	**90%**
3s_vs_5z	Hard	96%	**100%**
bane_vs_bane	Hard	**100%**	**100%**
2c_vs_64zg	Hard	**100%**	**100%**
corridor	Super Hard	0%	**100%**
MMM2	Super Hard	98%	**100%**
3s5z_vs_3s6z	Super Hard	3%	**85%** (envs = 4)
27m_vs_30m	Super Hard	56%	**100%**
6h_vs_8z	Super Hard	0%	**93%** ($\lambda = 0.3$)

Table 4: Best median test win rate of Finetuned-QMIX and QMIX in all scenarios.

6.4 Non-monotonic Matrix Games

In this section we first show the Q_{tot} learned by QMIX in two non-monotonic matrix games; then we propose a simple trick that may improve the performance of QMIX in such environments.

Table 5c and 5d show the Q_{tot} learned by QMIX for the two non-monotonic matrix games (Table 5a and 5b). Specifically, Table 5b shows that the finetuned QMIX can learn the correct optimal action for payoff matrix 5d, while the Q_{tot} is not consistent to that of payoff matrix 5d. However, Table 5c shows that QMIX learns incorrect argmax action for payoff matrix 5a.

Reward Shaping To resolve the incorrect argmax action in above non-monotonic matrix game (Table 5a), we investigate whether QMIX can learn a correct argmax action by reshaping the task's reward function without changing its goal. We find that the reward -12 in Table 5a does not assist the agents in finding the optimal solution. Then, as shown in Table 6, this non-monotonic matrix can be solved by simply replacing the insignificant reward -12 with -0.5. Because the reward function for reinforcement learning is usually set by the users. In practice, this tip hints that we can improve the performance of QMIX in some tasks by increasing the scale of the important rewards of the tasks; and reduce the scale of rewards that may cause disruption.

8	-12	-12
-12	0	0
-12	0	0

(a) Payoff matrix 1

12	0	10
0	10	10
10	10	10

(b) Payoff matrix 2

-12	-12	-12
-12	0	0
-12	0	**0**

(c) QMIX: Q_{tot} for Payoff matrix 1

12.0	0.3	9.9
0.2	-4.4	-1.1
10.0	-0.9	7.9

(d) QMIX: Q_{tot} for Payoff matrix 2

Table 5: Non-monotonic matrix games from (a) [23] and (b) [12]; and the learned Q_{tot} (c) and (d) for Table (a) and (b); Bold text indicates the reward of the argmax action.

The results of this experiment further demonstrate that some non-monotonic games may not be truly non-monotonic, but rather have poorly designed reward functions.

8.0	-0.5	-0.5
-0.5	0	0
-0.5	0	0

(a) Reshaped Payoff matrix 1

8.0	-0.3	-0.3
-0.3	-0.3	-0.3
-0.3	-0.3	-0.3

(b) QMIX: Q_{tot}

Table 6: We replace the insignificant reward -12 with reward -0.5 for Matrix Game 5a. QMIX learns a Q_{tot} which has a correct argmax. Bold text indicates argmax action's reward.

7 Discussion

7.1 Theory

To better understand the monotonicity constraint, we first make a theoretical analysis for it. Our core assumption is that the joint action-value function Q_{tot} can be represented by a non-linear mapping $f_\phi(s; Q_1, Q_2, ...Q_N)$, *but without the monotonicity constraint*.

Definition 1. ***Cooperative tasks.*** *For a task with N agents ($N > 1$), all agents have a common goal.*

Definition 2. ***Semi-cooperative Tasks.*** *Given a cooperative task with a set of agents $\mathbb{N}$. For all states s of the task, if there is a subset $\mathbb{K} \subseteq \mathbb{N}$, $\mathbb{K} \neq \varnothing$, where the $Q_i, i \in \mathbb{K}$ increases while the other $Q_j, j \notin \mathbb{K}$ are fixed, this will lead to an increase in Q_{tot}.*

As a counterexample, the collective action problem (social dilemma) is not Semi-cooperative task. i.e., since the Q value may not include future rewards when γ ¡ 1, the collective interest in the present may be detrimental to the future interest.

Definition 3. ***Competitive Cases.*** *Given two agents i and j, we say that agents i and j are competitive if either an increase in Q_i leads to a decrease in Q_j or an increase in Q_j leads to a decrease in Q_i.*

As an examples, the matrix game as in Table 1a is a cooperative task with competitive cases. As the random samples in reinforcement learning may lead to different behavioral preferences of agents. If one agent prefers action 0 (Like hunting) and the other agent prefers action 1 or 2 (Like sleeping or entertaining), they will have a conflict of interest (Those who like to entertaining will cause the hunter to fail to catch the prey).

Definition 4. ***Purely Cooperative Tasks.*** *Semi-cooperative tasks without competitive cases.*

Proposition 1. *Purely Cooperative Tasks can be represented by monotonic mixing networks.*

Proof. Since the monotonic mixing network is a universal function approximator of monotonic functions, for a Semi cooperative task, if there is a case (state s) that cannot be represented by a monotonic mixing network, i.e., $\frac{\partial Q_{tot}(s)}{\partial Q_i} < 0$, then an increase in Q_i must lead to a decrease in $Q_j, j \neq i$ (since there is no Q_j decrease, by Def. 2, the constraint $\frac{\partial Q_{tot}(s)}{\partial Q_i} < 0$ does not hold).

Therefore, by Def. 3 this cooperative task has a competitive case which means it is not a purely cooperative task. □

7.2 Why monotonicity constraints work well in SMAC and DEPP?

In this section, we future discuss why the monotonicity constraint works well in these purely cooperative tasks. First we explain in detail why SMAC and DEPP can be interpreted as purely cooperative tasks. In practice, (1) For the SMAC, we can decompose the hit-point damage dealt linearly, and divide the units killed rewards to the agents near the enemy evenly, the victory rewards to all agents. This approximate linear decomposition [4] also explains why the VDNs also work well in SMAC (Table. 3). (2) For the DEPP, we can divide the reward for catching prey evenly to the nearest predators. These simple positive rewards of SMAC and DEPP make these agents have only a shared goal, i.e, to kill all enemies or capture preys. Intuitively, **these linear and fairly assigned rewards allow the agents to work in a purely cooperative mode**. Therefore, QMIX can represent a optimal solution of SMAC, i.e., a purely cooperative decomposition of Q values.

Then, just as in RMC's implementation (Figure 2), the monotonicity constraint reduces the range of values of each mixing weight by half, the hypothesis space is assumed to decrease exponentially by $(\frac{1}{2})^N$ (N denotes the number of weights). By Proposition 1, the Q value decomposition mappings of the SMAC and DEPP are subsets of the hypothesis space of monotonic mixing network. Therefore, using the monotonicity constraint can allow for avoiding searching invalid parameters, leading to a significant improvement in sampling efficiency.

Our analysis shows that QMIX works well if a multi-agent task can be interpreted as purely cooperative, even if it can also be interpreted as competitive. **That is, QMIX will try to find a purely cooperative interpretation for a complex multi-agent task.**

8 Conclusion

In this paper, we investigate the influence monotonicity constraint and implementation tricks in cooperative MARL tasks. Our analyses show that relaxing the monotonicity constraint of the mixing network will not always improve the performance of QMIX. What's more critical is that monotonicity constraint can improve sample efficiency in some purely cooperative tasks, such as SMAC and DEPP. Benefiting from the monotonicity constraint, the fine-tuned QMIX achieves SOTA performance in SMAC. These facts imply that we can design reward functions in the real multi-agent task that can be interpreted as purely cooperative, improving the learning sample efficiency of the MARL.

9 Broader Impact

Many complex real-world multi-agent cooperative problems can be simulated as CTDE multi-agent tasks. Specifically, decentralized agents can be applied to robot swarm control, vehicle coordination, and network routing. Applying MARL to these scenarios often requires a large number of samples to train the model, which implies high implementation costs, such as thousands of CPUs, power resources, and expensive robotic equipment (damaged drones or autonomous cars). Therefore, there is an urgent need to avoid any and all waste of such resources. In this work, we shows the monotonicity constraint and implementation tricks can help to improve the sample efficiency in some purely cooperative tasks, thereby reducing the wasting of resources. In addition, we are hopeful that this paper will call on the community to be more fair in comparing the performance of algorithms.

[4] As $Q_\pi(s,u) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s, u]$, the reward is linearly assignable meaning that Q value is linearly assignable.

REFERENCES

[1] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *arXiv:2006.05990*, 2020.

[2] Wendelin Boehmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *ICML 2020, 13-18 July 2020, Virtual Event*, pp. 980–991, 2020.

[3] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9 (1):427–438, 2012.

[4] Karl Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. *arXiv preprint arXiv:2009.04416*, 2020.

[5] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO. *arXiv:2005.12729*, 2020.

[6] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI-18, New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 2974–2982. AAAI Press, 2018.

[7] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *arXiv preprint arXiv:1709.06011*, 2017.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR 2015, San Diego, CA, USA, May 7-9, 2015*, 2015.

[9] Tadashi Kozuno, Yunhao Tang, Mark Rowland, Rémi Munos, Steven Kapturowski, Will Dabney, Michal Valko, and David Abel. Revisiting peng's q (λ) for modern reinforcement learning. *arXiv preprint arXiv:2103.00107*, 2021.

[10] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016. ISSN 09252312.

[11] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 6379–6390, 2017.

[12] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: multi-agent variational exploration. In *NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 7611–7622, 2019.

[13] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1928–1937, 2016.

[14] Ofir Nachum, Michael Ahn, Hugo Ponte, Shixiang Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *arXiv preprint arXiv:1908.05224*, 2019.

[15] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Pomdps for robotic tasks with mixed observability. 5:4, 2009.

[16] Bei Peng, Tabish Rashid, Christian A Schroeder de Witt, Pierre-Alexandre Kamienny, Philip HS Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *arXiv e-prints*, pp. arXiv–2003, 2020.

[17] Jing Peng and Ronald J Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pp. 226–232. Elsevier, 1994.

[18] Doina Precup, Richard S. Sutton, and Satinder P. Singh. Eligibility traces for off-policy policy evaluation. In *(ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pp. 759–766. Morgan Kaufmann, 2000.

[19] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 4292–4301, 2018.

[20] Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted QMIX: Expanding Monotonic Value Function Factorisation. *arXiv preprint arXiv:2006.10800*, 2020.

[21] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[23] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Hostallero, and Yung Yi. QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 5887–5896, 2019.

[24] Kyunghwan Son, Sungsoo Ahn, Roben Delos Reyes, Jinwoo Shin, and Yung Yi. QTRAN++: Improved Value Transformation for Cooperative Multi-Agent Reinforcement Learning. *arXiv:2006.12010*, 2020.

[25] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

[26] Jianyu Su, Stephen Adams, and Peter A Beling. Value-decomposition multi-agent actor-critics. *arXiv preprint arXiv:2007.12306*, 2020.

[27] Jianyu Su, Stephen Adams, and Peter A. Beling. Value-Decomposition Multi-Agent Actor-Critics. *arXiv:2007.12306*, 2020.

[28] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv preprint arXiv:1706.05296*, 2017.

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[30] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: Duplex Dueling Multi-Agent Q-Learning. *arXiv:2008.01062*, 2020.

[31] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Off-Policy Multi-Agent Decomposed Policy Gradients. *arXiv:2007.12322*, 2020.

[32] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1995–2003.

[33] Ermo Wei, Drew Wicke, David Freelan, and Sean Luke. Multiagent Soft Q-Learning. *arXiv preprint arXiv:1804.09817*, 2018.

[34] Yuchen Xiao, Joshua Hoffman, and Christopher Amato. Macro-action-based deep multi-agent reinforcement learning. In *Conference on Robot Learning*, pp. 1146–1161. PMLR, 2020.

[35] 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.

[36] Chongjie Zhang and Victor R. Lesser. Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

[37] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning Implicit Credit Assignment for Multi-Agent Actor-Critic. *arXiv preprint arXiv:2007.02529*, 2020.

[38] Ming Zhou, Jun Luo, and Julian Villella et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving, 2020.

A Pseudo-code

In this section, we show the pseudo-code for the training procedure of RMC. (1) Training the critic network with offline samples and 1-step TD error loss improves the sample efficiency for critic networks; (2) Training policy networks end-to-end and critic with TD(λ) and online samples improves learning stability of RMC [5].

Algorithm 1 Optimization Procedure for RMC

Initialize offline replay memory D and online replay memory D'.
Randomly initialize θ and ϕ for the policy networks and the mixing critic respectively.
Set $\phi^- \leftarrow \phi$.
while not terminated **do**
  # Off-policy stage
  Sample b episodes $\tau_1, ..., \tau_b$ with $\tau_i = \{s_{0,i}, o_{0,i}, u_{0,i}, r_{0,i}, ..., s_{T,i}, o_{T,i}, u_{T,i}, r_{T,i}\}$ from offline replay memory D.
  Update the monotonic mixing network with $y_{t,i}$ calculated by 1-step bootstrap return ($y_{t,i} = r_{t,i} + \gamma Q^{\pi}_{\phi^-}(s_{t+1}, \vec{u}_{t+1})$):

$$\nabla_{\phi} \frac{1}{bT} \sum_{i=1}^{b} \sum_{t=1}^{T} \left(y_{t,i} - Q^{\pi}_{\phi}\left(s_{t,i}, u^1_{t,i}, ..., u^n_{t,i}\right)\right)^2 . \tag{7}$$

  # On-policy stage
  Sample b episodes $\tau_1, ..., \tau_b$ with $\tau_i = \{s_{0,i}, o_{0,i}, u_{0,i}, r_{0,i}, ..., s_{T,i}, o_{T,i}, u_{T,i}, r_{T,i}\}$ from online replay memory D'.
  Update the monotonic mixing network with $y^{TD(\lambda)}_{t,i}$ calculated by TD(λ) (Eq. 10):

$$\nabla_{\phi} \frac{1}{bT} \sum_{i=1}^{b} \sum_{t=1}^{T} \left(y^{TD(\lambda)}_{t,i} - Q^{\pi}_{\phi}\left(s_{t,i}, u^1_{t,i}, ..., u^n_{t,i}\right)\right)^2 . \tag{8}$$

  Update the decentralized policy networks end-to-end by maximizing the Q value with the Adaptive Entropy :

$$\nabla_{\theta} \frac{1}{bT} \sum_{i=1}^{b} \sum_{t=1}^{T} \left(-Q^{\pi}_{\phi}\left(s_{t,i}, \pi^1_{\theta_1}(\cdot|z^1_{t,i}), ..., \pi^n_{\theta_n}(\cdot|z^n_{t,i})\right) - \frac{1}{n} \sum_{a=1}^{n} \mathcal{H}\left(\pi^a_{\theta_a}(\cdot|z^a_{t,i})\right)\right) . \tag{9}$$

  if at target update interval **then**
    Update the target mixing network $\phi^- \leftarrow \phi$.
  end if
end while

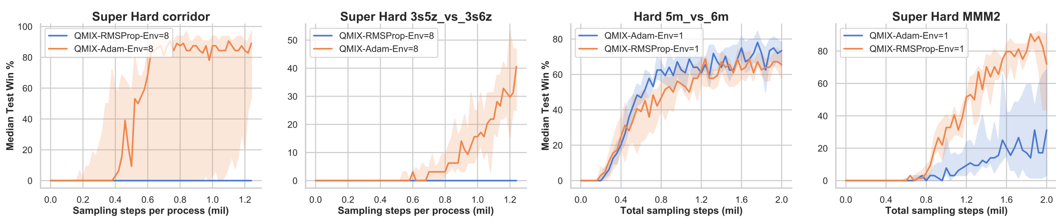
[5] [4] shows that actor-networks generally have a lower tolerance for sample reuse than critic networks; and for RMC, our empirical evidence shows that $TD(\lambda)$ is not stable in the offline samples.

B Code-level Optimizations

Engstrom *et.al* [5] and Andrychowicz *et. al* [1] investigate the influence of code-level optimizations on the performance of PPO [22] and provide tuning optimizations. These optimizations include: (1) Adam and Learning rate annealing. (2) Orthogonal initialization and Layer scaling. (3) Observation normalization. (4) Value normalization. (5) N-step returns (eligibility traces). (6) Reward scaling. (7) Reward clipping etc. In this section, we investigate the impact of a part of these optimizations in multi-agent scenarios and provide tuning optimizations. We use **8 rollout processes** for parallel sampling to obtain as many samples as possible from SMAC at a high rate.

B.1 Optimizer

Study description. QMIX and the majority of its variant algorithms use RMSProp to optimize neural networks as they prove stable in SMAC. We attempt to use Adam to optimize QMIX's neural network with quickly convergence benefiting from momentum:



(a) Eight rollout processes are used for sampling. (b) Only one rollout process is used for sampling.

Figure 5: (a) Adam significantly improves performance when samples are updated quickly; (b) The Q networks optimized by Adam is prone to overfitting when samples are updated slowly.

Interpretation. Adam boosts the network's convergence allowing for full utilization of the large quantity of samples sampled in parallel. Figure 5a shows that Adam [8] increases the win rate by 100% on the Super Hard map *corridor*. However, Figure 5b shows that when we use only one sampling process, samples are being updated slowly with the fixed size of the replay buffer; and the neural network becomes prone to overfitting. We find that the Adam optimizer solves the problem posed by VMIX[26] in which QMIX does not work well under parallel training.

Recommendation. Use Adam and quickly update the samples; or reducing the learning rate when the samples update slowly.

B.2 N-step Returns

Study description. N-step returns such as TD(λ) [29], Peng's Q(λ) [17], and TB(λ) [18] achieve a balance between return-based algorithms (where return refers to the sum of discounted rewards $\sum_t \gamma^t r_t$) and bootstrap algorithms (where return refers to $r_t + V(s_{t+1})$), speeding up the convergence of reinforcement learning algorithms. TD(λ) can be expressed as Eq. 10:

$$
\begin{aligned}
G_s^\lambda &\doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{s:s+n} \\
G_{s:s+n} &\doteq \sum_{t=s}^{s+n}\gamma^{t-s}r_t + \gamma^{n+1}V\left(s_{s+n+1}, u\right)
\end{aligned} \tag{10}
$$

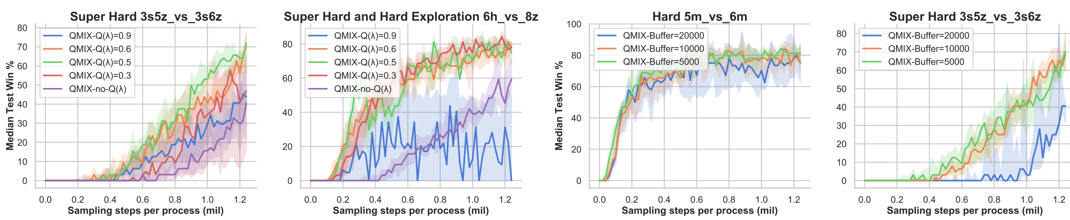
Peng's Q(λ) replaces the V value of the next state with the max Q value, as shown in Eq. 11:

$$
G_{s:s+n} \doteq \sum_{t=s}^{s+n}\gamma^{t-s}r_t + \gamma^{n+1}\max_u Q\left(s_{s+n+1}, u\right) \tag{11}
$$

where λ is the discount factor of the traces and $\left(\prod_{s=1}^{t} \lambda\right) = 1$ when $t = 0$. When λ is set to 0, it is equivalent to 1-step bootstrap returns. When λ is set to 1, it is equivalent to Monte Carlo [29] returns. [9] show that while Peng's Q(λ) does not learn optimal policies under arbitrary behavior policies, a convergence guarantee can be recovered if the behavior policy tracks the target policy, as is often the case in practice. Therefore, we study the application of Peng's Q(λ) for QMIX,

Interpretation. Q networks without sufficient training usually have a large bias that impacts bootstrap returns. Figure 6a shows that Q(λ) allows for faster convergence in our experiments by reducing this bias. However, large values of λ may lead to failed convergence due to large variance and off-policy bias. Figure 6a shows that when λ is set to 0.9, it has a detrimental impact on the performance of QMIX.

Recommendation. Use Q(λ) with a small value of λ.



(a) Experiments for Q(λ).

(b) Experiments for Replay Buffer Size.

Figure 6: (a) Q(λ) significantly improves performance of QMIX, but large values of λ lead to instability in the algorithm. (b) Setting the replay buffer size to 5000 episodes allows for QMIX's learning to be more stable than by setting it to 20000 episodes.

B.3 Replay Buffer Size

Study description. In single-agent Deep Q-networks (DQN), the replay buffer size is usually set to a large value. However, in multi-agent tasks, as the action space becomes larger than that of single-agent tasks, the distribution of samples changes more quickly. In this section, we study the impact of the replay buffer size on performance.

Interpretation. Figure 6b shows that a large replay buffer size causes instability in QMIX's learning. The causes of this phenomenon are as follows: (1) In multi-agent tasks, samples become obsolete more quickly than in single-agent tasks. (2) Echoing in Appendix. B.1, Adam performs better with samples with fast updates.

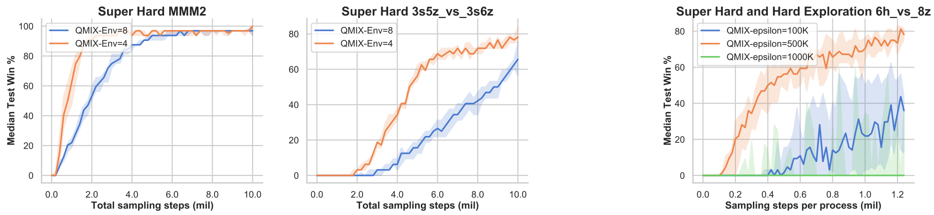
Recommendation. Use a small replay buffer size.

B.4 Rollout Process Number

Study description. When we collect samples in parallel as is done in A2C [25], it shows that when there is a defined total number of samples and an unspecified number of rollout processes, the median test performance becomes inconsistent. This study aims to perform analysis and provide insight on the impact of the number of processes on the final performance.

Interpretation. Under the A2C [13] training paradigm, the total number of samples can be calculated as $S = E \cdot P \cdot I$, where S is the total number of samples, E is the number of samples in each episode, P is the number of rollout processes, and I is the number of policy iterations. Figure 7a shows that we are given both S and E; the fewer the number of rollout processes, the greater the number of policy iterations [29]; a higher number of policy iterations leads to an increase in performance. However, it also causes both longer training time and decreased stability.

Recommendation. Use fewer rollout processes when samples are difficult to obtain, especially for real-world robot learning; otherwise, use more rollout processes.



(a) Experiments for Rollout Process Number. (b) Experiments for ϵ anneal period.

Figure 7: (a) Given the total number of samples, fewer processes achieve better performance. We set the replay buffer size to be proportional to the number of processes to ensure that the novelty of the samples is consistent. (b) On the hard-to-explore scenario $6h\_vs\_8z$, defining a proper length for ϵ anneal period significantly improves performance.

B.5 Exploration Steps

Study description. Some scenarios in SMAC are hard to explore, such as $6h\_vs\_8z$, so the settings of ϵ-greedy become critically important. In this study, we analyze the effect of ϵ anneal period on performance.

Interpretation. As shown in Figure 7b, increasing the length of the ϵ anneal period from 100K steps to 500K steps allows for a 38% increase in win rate in the Super Hard Exploration scenario $6h\_vs\_8z$. However, increasing this value to 1000K instead causes the model to collapse.

Recommendation. Increase the value of the ϵ anneal period to an appropriate length on hard-to-explore scenarios.

C Hyperparameters

Tricks	Value-based (VB)	Policy-bassed (PG)
Optimizer	Adam, RMSProp	Adam, RMSProp
Learning Rates	0.0005, 0.001	0.0005, 0.001, (and 0.0001 for DOP)
Batch Size(episodes)	32, 64, 128	32, 64
Replay Buffer Size	5000, 10000, 20000	2000, 5000, 10000, 20000
Q(λ), TD(λ)	0, 0.3, 0.6, 0.9	0, 0.3, 0.6, 0.9
(Adaptive) Entropy	-	0.01, 0.03, 0.06
ϵ Anneal Steps	50K, 100K, 500K, 1000K	100K, 500K for DOP

Table 7: Hyperparameters Search on SMAC.

In this section, we present our tuning process. We get the optimal hyperparameters for each algorithm by the grid search, shown in Table 7. Specifically,

1. For experiments in Sec. 6.2, we perform grid search schemes on a typical hard environment (5m_vs_6m) and super hard environment (3s5z_vs_3s6z) to find a **general** set of hyperparameters for each algorithm. In this way, we can evaluate the robustness of these MARL algorithms.
2. For experiments in Sec. 6.3, we perform hyperparameter search on each scenarios for QMIX to demonstrate the best performance of QMIX.

Table 8a and 8b shows our general settings for the these algorithms. The network size is calculated under $6h\_vs\_8z$, where adding *Our* denotes the new hyperparameter settings. Next, we describe in detail the setting of these hyperparameters,

Neural Network Size We first ensure the network size is the same order of magnitude, which means that we decrease the critic-net size of LICA from 29696K to 389K, and we use 4 attention heads leading the mixing-net size of QPLEX from 476K to 152K. All the agent networks are the same as those found in QMIX [19].

Algorithms	LICA	**OurLICA**	DOP	**OurDOP**	RMC
Optimizer	Adam	**Adam**	RMSProp	RMSProp	Adam
Batch Size(episodes)	32	32	Off=32, On=16	**Off=64, On=32**	Off=64, On=32
TD(λ)	0.8	**0.6**	0.8, TB(λ=0.93)	0.8, TB(λ=0.93)	0.6
Adaptive Entropy	0.06	0.06	-	-	0.03
ϵ Anneal Steps	-	-	500K (double)	500K (double)	-
Critic-Net Size	29696K	**389K**	122K	122K	69K
Rollout Processes	32	**8**	4	**8**	8

(a) Setting of Policy-based algorithms.

double: DOP first adds noise to the output of the policy network, then mask invalid actions and adds noise to the probabilities again.

Algorithms	QMIX	**OurQMIX**	Qatten	**OurQatten**	QPLEX	**OurQPLEX**
Optimizer	RMSProp	**Adam**	RMSProp	**Adam**	RMSProp	**Adam**
Batch Size (epi.)	128	**128**	32	**128**	32	**128**
Q(λ)	0	**0.6**	0	**0.6**	0	**0.6**
Attention Heads	-	-	4	4	10	**4**
Mixing-Net Size	41K	41K	58K	58K	476K	**152K**
ϵ Anneal Steps	50K $\rightarrow$ **500K for** $6h\_vs\_8z$**, 100 K for others**					
Rollout Processes	8	**8**	1	**8**	1	**8**

(b) Setting of Value-based algorithm.

Table 8: Hyperparameters Settings.

Optimizer & Learning Rate We use Adam to optimize all networks, except VMIX and DOP (works better with RMSProp), as it may accelerate the convergence of the algorithms. Furthermore, we use different learning rates for each algorithm: (1) For all value-based algorithms, neural networks are trained with 0.001 learning rate. (2) For LICA, we set the learning rate of the agent network to 0.0025 and the critic network's learning rate to 0.0005. (3) For RMC and VMIX, we set the learning rates to 0.001.

Batch Size We find that a large batch size helps to improve the stability of the algorithms. Therefore, for value-based algorithms, we set the batch size to 128. For the policy-based algorithms, we set the batch size to 64/32 (Offline/Online training) due to the fact that online update requires only the newest data.

Replay Buffer Size As discussed in Appendix. B.3, a small replay buffer size facilitates the convergence of the MARL algorithms. Therefore, for SMAC, the size of all replay buffers is set to 5000 episodes. For Predator-Prey, we set the buffer size to 1000 episodes.

Exploration As discussed in Appendix. B.5, we use ϵ-greedy action selection, decreasing ϵ from 1 to 0.05 over n-time steps (n can be found in Table 8b) for value-based algorithms. We use the Adaptive Entropy [37] (Appendix. E.2.1) for LICA and RMC, because it facilitates the automatic adjustment of the size of the entropy loss in different scenarios. DOP first adds noise to the output of the policy network, then mask invalid actions and adds noise to the probabilities again, called **double noise**. This double noise prevents DOP from collapsing during training.

N-step returns We find that the λ values of Q(λ) and TD(λ) are hevily depend on the scenario. We are using λ = 0.6 for all tasks as the value works stably in most scenarios. However, for the on-policy method VMIX, we set λ = 0.8.

Rollout Processes Number For SMAC and Discrete PP, 8 rollout processes for parallel sampling are used to obtain as many samples as possible from the environments at a high rate. This also ensures that all the algorithms share the same number of policy iterations and sample size (10 million). For the non-monotonic matrix games, we set the processes number to 32. At last, 4 rollout processesare used for Continuous PP.

Other Settings We set all discount factors γ = 0.99. We update the target network every 200 episodes. We find that the optimal hyperparameters of the value-based algorithms are similar due to the fact that they share the same basic architecture and training paradigm. Therefore, the settings for VDNs and WQMIX are the same as for QMIX. Specifically, we use OW-QMIX, detailed in E.1.4, in WQMIX as the baseline.

D OMITTED EXPERIMENTAL RESULTS

D.1 OMITTED FIGURES

In this section we provide the figures omitted in Sec. 6.2. Figure 8 shows that (1) QMIX achieves excellent performance on all hard scenarios in SMAC, and outperforms other algorithms; (2) QPLEX's policy collapses in some hard scenarios, such as $8m\_vs\_9m$, $6h\_vs\_8z$ and $corridor$ [6]; (3) each of the algorithms achieves good performance on DEPP. The median test winning rates in Figure 8 are lower than in Table 3 as we smoothed these curves.

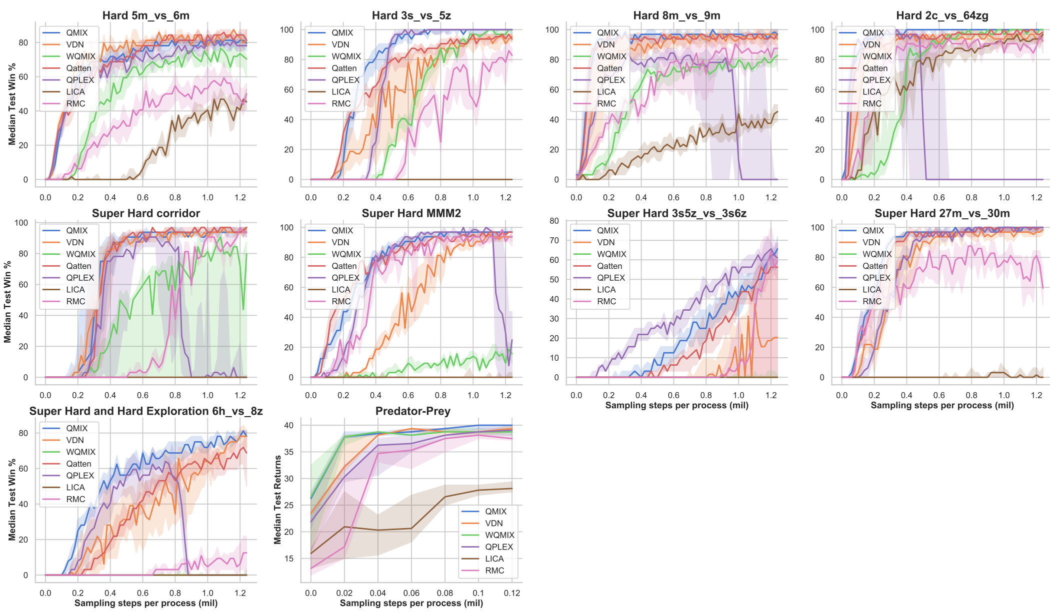


Figure 8: Median test winning rates of MARL algorithms on Hard scenarios of SMAC and DEPP.

D.2 THE PERFORMANCE OF ORIGINAL ALGORITHMS

In this section, we compare performance of the original algorithms with third-party experimental results, i.e. experimental results of the paper citing the algorithm.

For VDNs and QMIX, the original SMAC paper [21] shows that VDNs and QMIX do not perform well in hard and super hard scenarios. For Qatten, the experiments in [30] demonstrates that the performance of Qatten is worse than vanilla QMIX. [16] demonstrates that QPLEX and DOP does not work well in hard and super hard scenarios in SMAC, and the their performance is worse than vanilla QMIX. It is interesting that WQMIX [20] shows the poor performance of WQMIX in super hard scenarios $3s5z\_vs\_3s6z$ and $corridor$. The original test results in LICA are not considered as 64 million samples are used in their experiments. [27] shows that VMIX mainly works well in Easy scenarios.

However, after our hyperparameter tuning, all the value-based methods perform well in Hard and Super Hard scenarios; and VMIX works well in some Hard scenarios. This shows that our hyperparameters does improve their performance.

[6]It may be that QPLEX feeds both actions and states into the mixing network in its implementation. The mixing network can predict true Q_{tot} without correct Q_i, so that the Q_i becomes useless.

E CTDE ALGORITHMS

E.1 VALUE-BASED METHODS

E.1.1 VDNS

Value decomposition networks (VDNs) [7] [28] seek to learn a joint action-value function $Q_{tot}(\boldsymbol{\tau}, \mathbf{u})$, where $\boldsymbol{\tau} \in \mathbf{T} \equiv \mathcal{T}^n$ is a joint action- observation history and $\mathbf{u}$ is a joint action. It represents Q_{tot} as the sum of individual value functions $Q_a\left(\tau^i, u^i; \theta^i\right)$:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q_i\left(\tau^i, u^i; \theta^i\right).$$

E.1.2 QATTEN

Qatten [8] [35], introduces an attention mechanism into the monotonic mixing network of QMIX:

$$Q_{tot} \approx c(s) + \sum_{h=1}^{H} w_h \sum_{i=1}^{N} \lambda_{i,h} Q^i \tag{12}$$

$$\lambda_{i,h} \propto \exp\left(e_i^T W_{k,h}^T W_{q,h} e_s\right) \tag{13}$$

where $w_h = \left|f^{NN}(s)\right|_h$, $W_{q,h}$ transforms e_s into a global query, and $W_{k,h}$ transforms e_i into an individual key. The e_s and e_i may be obtained by an embedding transformation layer for the true global state s and the individual state s_i.

E.1.3 QPLEX

QPLEX [9] [30] decomposes Q values into advantages and values based on Qatten, similar to Dueling-DQN [32]:

$$\begin{aligned} \text{(Joint Dueling) } Q_{tot}(\tau, u) &= V_{tot}(\tau) + A_{tot}(\tau, u) \\ V_{tot}(\boldsymbol{\tau}) &= \max_{u'} Q_{tot}\left(\boldsymbol{\tau}, \boldsymbol{u}'\right) \end{aligned} \tag{14}$$

$$\begin{aligned} \text{(Individual Dueling) } Q_i\left(\tau_i, u_i\right) &= V_i\left(\tau_i\right) + A_i\left(\tau_i, u_i\right) \\ V_i\left(\tau_i\right) &= \max_{u'} Q_i\left(\tau_i, u_i'\right) \end{aligned} \tag{15}$$

$$\frac{\partial A_{tot}(s, \boldsymbol{u}; \boldsymbol{\theta}, \phi)}{\partial A_i\left(\tau^i, u^i; \theta^i\right)} \geq 0, \quad \forall i \in \mathcal{N} \tag{16}$$

In other words, Eq. 16 (advantage-based monotonicity) transfers the monotonicity constraint from Q values to advantage values. QPLEX thereby reduces limitations on the mixing network's expressiveness.

E.1.4 WQMIX

WQMIX [10] [20], just like Optimistically-Weighted QMIX (OW-QMIX), uses different weights for each sample to calculate the squared TD error of QMIX:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} w(s, \mathbf{u}) \left(Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) - y_i\right)^2 \tag{17}$$

[7] VDN code: https://github.com/oxwhirl/pymarl
[8] Qatten code: https://github.com/simsimiSION/pymarl-algorithm-extension-via-starcraft
[9] QPLEX code: https://github.com/wjh720/QPLEX
[10] WQMIX code: https://github.com/oxwhirl/wqmix

$$w(s, \mathbf{u}) = \begin{cases} 1 & Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) < y_i \\ \alpha & \text{otherwise.} \end{cases} \quad (18)$$

Where $\alpha \in (0, 1]$ is a hyperparameter and y_i is the true target Q value. WQMIX prefers those optimistic samples (true returns are larger than predicted), i.e., decreasing the weights of samples with non-optimistic returns. More critically, WQMIX uses an unconstrained true Q Network as a target network to guide the learning of QMIX. The authors prove that this approach can resolve the estimation errors of QMIX in the non-monotonic case.

E.2 Policy-based Methods

E.2.1 LICA

LICA [11] [37] completely removes the monotonicity constraint through a policy mixing critic, as shown in Figure 9:

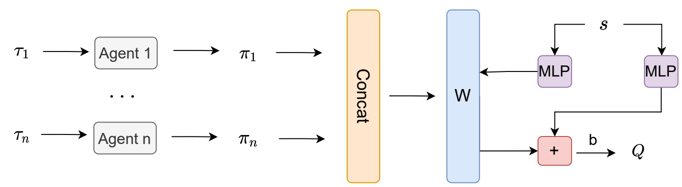


Figure 9: Architecture for LICA. LICA's mixing critic maps policy distribution to the Q value directly, in effect obviating the monotonicity constraint.

LICA's mixing critic is trained using squared TD error. With a trained critic estimate, decentralized policy networks may then be optimized end-to-end simultaneously by maximizing $Q^{\pi}_{\theta_c}$ with the stochastic policies $\pi^i_{\theta_i}$ as inputs:

$$\max_{\theta} \mathbb{E}_{t, s_t, u^1_t, \ldots, \tau^n_t} \left[Q^{\pi}_{\theta_c} \left(s_t, \pi^1_{\theta_1} \left(\cdot \mid \tau^1_t \right), \ldots, \pi^n_{\theta_n} \left(\cdot \mid \tau^n_t \right) \right) + \mathbb{E}_i \left[\mathcal{H} \left(\pi^i_{\theta_i} \left(\cdot \mid \tau^i_t \right) \right) \right] \right] \quad (19)$$

where the gradient of entropy item $\mathbb{E}_i \left[\mathcal{H} \left(\pi^i_{\theta_i} \left(\cdot \mid z^i_t \right) \right) \right]$ is normalized by taking the quotient of its own modulus length: Adaptive Entropy (Adapt Ent). Adaptive Entropy automatically adjusts the coefficient of entropy loss in different scenarios.

E.3 Summary

VDNs requires a linear decomposition of Q values, so it has the strongest monotonicity constraint. Since the weights calculated by softmax (attention mechanism) are greater than or equal to zero, the constraint strengths of Qatten and QMIX are approximately equal. QPLEX just shifts the constraint to advantage values without removing it. WQMIX relaxes the monotonicity constraint even further by a true Q value network and theoretical guarantees. LICA completely removes the monotonicity constraint by new network architecture. We rank the strength of the monotonicity constraints on these MARL algorithms:

$$\text{VDNs} > \text{QMIX} \approx \text{Qatten} > \text{QPLEX} > \text{WQMIX} > \text{LICA} \quad (20)$$

[11] LICA code: https://github.com/mzho7212/LICA