Uncertainty-Based Transformation for Monotonic Value Function Factorization in Multi-Agent Reinforcement Learning

Anonymous Authors[1]

Abstract

Monotonic value function factorization enforces that the joint-action value is monotonic in the per-agent values, which guarantees consistency between centralized and decentralized policies. However, this factorization suffers from the representation limitation because it can only represent values in the restricted monotonic space. Placing more importance on better joint actions is a promising way to rectify this issue. However, it introduces incorrect weights on the suboptimal with stochasticity, leading to learning instability and even convergence to the suboptimal. To solve this problem, we propose Uncertainty-Based Transformation (UTRAN), which projects the original action-value target into the space that monotonic value function factorization can represent. First, we apply a world model to predict the rewards and the future state for the current state-action pair, where the predicted standard deviations quantify the uncertainty of the state-action pair. Then we model the best per-agent value and use this value to replace the suboptimal target with considerable uncertainty. This transformation remains the optimal policy unchanged and eliminates the representation limitation. Empirical results demonstrate that UTRAN outperforms existing value factorization baselines on many multi-agent cooperative benchmarks, especially in games with non-monotonic and stochastic targets.

1. Introduction

Recent progress in cooperative multi-agent reinforcement learning (MARL) has shown attractive prospects for real-world applications, such as the smart grid management (Aladdin et al., 2020) and autonomous vehicles (Zhou et al., 2021). Due to practical communication constraints and intractably large joint action space, decentralized policies are often used in MARL. It is possible to use extra information from the environment and other agents in a simulated or laboratory setting. Exploiting this information can significantly benefit policy optimization and improve learning performance (Foerster et al., 2016; 2018; Rashid et al., 2020). In the paradigm of centralized training with decentralized execution (CTDE), agents' policies are trained with access to global information in a centralized way and executed only based on local histories in a decentralized way (Oliehoek et al., 2008; Kraemer & Banerjee, 2016). One of the most significant challenges is to guarantee the consistency between the individual policies and the centralized policy, which is also known as Individual-Global Max (Son et al., 2019). QMIX (Rashid et al., 2018) provides a promising class of algorithms to address this issue. It applies a monotonic mixing network to enforce the joint Q-value is monotonic in the per-agent values.

One fundamental problem for QMIX is the limited representational capacity, namely, a gap between the approximated joint Q-values by QMIX and the target Q-values. This representation limitation will result in convergence to suboptimum if the task requires significant coordination, e.g., *relative overgeneralization* (Son et al., 2019; Mahajan et al., 2019; Rashid et al., 2020). To decrease this gap, QPLEX (Wang et al., 2020a) and Tesseract (Mahajan et al., 2021) introduce the joint action or pairwise interactions into value factorization, which directly improves the representation capacity value function. Despite this advantage, these methods are less scalable because of the intractable computational complexity of centralized learning. In contrast, WQMIX introduces a weighting function into the projection from the target Q-value functions to the joint Q-values and uses it to down-weight every suboptimal action whose target value is less than the current estimate. However, it may be impossible to find an appropriate weight to recover the optimal policy when the target Q-value is characterized as non-monotonic and stochastic. This limitation stems from the stochasticity of the target at each update step. In WQMIX, the weight for a suboptimal should be small enough to prioritize the potential optimal action. However, the weights for all actions should be equal to avoid overestimating the

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

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

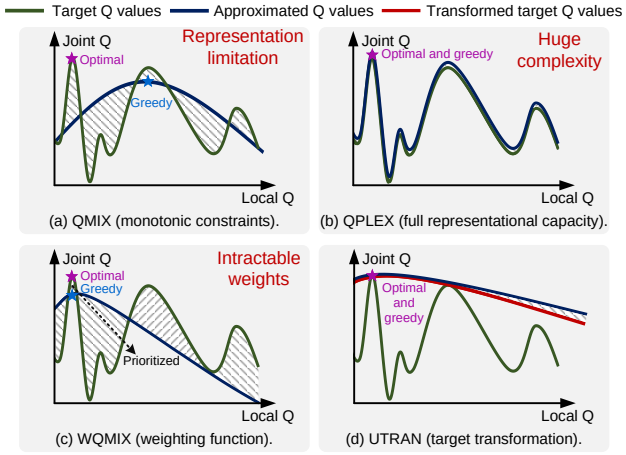

Figure 1. The relations between the target and the approximated joint Q-values in different value factorization methods.

suboptimal with stochastic targets, e.g., the reward is large with a low probability. A detailed explanation is provided in Section 3. An open research question in MARL raises:

Without representing all target Q-values from the environment, how to eliminate the representation limitation in tasks with stochasticity?

To tackle this problem, we propose a novel value function factorization method named Uncertainty-Based Transformation (UTRAN), which eliminates the representation limitation of monotonic value function factorization by projecting the joint Q-value target to the space that monotonic value function factorization can represent. First, we employ a world model to predict the reward and the future state for the current state-action pair, where the prediction error quantifies the stochasticity of the target. Then we model the best per-agent value function, which is the greatest Q-value of each action when other agents coordinate optimally. Finally, we use the minimal best per-agent values over all agents to replace the suboptimal target characterized as deterministic. We prove that this transformation remains the optimal policy unchanged and guarantees that all transformed targets are tractable for monotonic value function factorization. Fig. 1 shows the relations between the target and the approximated joint Q-values in QMIX, QPLEX, WQMIX, and UTRAN. The objective is to minimize the shaded area between the target and the approximated values. QPLEX tries to realize the full expressiveness power of the original targets. WQMIX prioritizes the area where the target is greater than the current estimate. In contrast, UTRAN achieves the full representational capacity of the transformed targets and converges to the original optimal.

We evaluate UTRAN on various cooperative MARL benchmarks, including matrix games, Predator-Prey (Son et al., 2019), and StarCraft Multi-Agent Challenge (Samvelyan et al., 2019). Empirical results show that UTRAN achieves superior performance over value factorization baselines, such as QMIX (Rashid et al., 2018), WQMIX (Rashid et al., 2020), and QTRAN (Son et al., 2019), especially on tasks which are characterized as non-monotonic and stochastic.

2. Background

A fully cooperative multi-agent task in the partially observable setting can be formulated as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (Oliehoek & Amato, 2016), consisting of a tuple $G = \langle A, S, \Omega, O, U, P, R, n, \gamma \rangle$, where $a \in A \equiv \{1, \ldots, n\}$ describes the set of agents, S denotes the set of states, Ω denotes the set of joint observations, and R denotes the set of rewards. At each time step, an agent obtains its observation $o \in \Omega$ based on the observation function $O(s, a) : S \times A \to \Omega$, and an action-observation history $\tau_a \in T \equiv (\Omega \times U)^*$. Each agent a chooses an action $u_a \in U$ by a stochastic policy $\pi_a(u_a|\tau_a) : T \times U \to [0, 1]$, forming a joint action $\mathbf{u} \in \mathbf{U}$, which leads to a transition on the environment through the transition function $P(s', r|s, \mathbf{u}) : S \times \mathbf{U} \times S \times R \to [0, 1]$, where $r \in R$ is the team reward. The goal of the task is to find the joint policy π which can maximize the joint Q-value function $Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}[G_t|s_t, \mathbf{u}]$, where $G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return. In this paper, we call the joint Q-value function $Q^\pi(s_t, \mathbf{u}_t)$ as the **true Q-value**, and the joint action $\mathbf{u}^* = \arg\max_{\mathbf{u}} Q^\pi(s_t, \mathbf{u})$ as the **real optimal joint action** at a given state s_t.

VDN (Sunehag et al., 2018), QMIX (Rashid et al., 2018), QPLEX (Wang et al., 2020a), and WQMIX (Rashid et al., 2020) are Q-learning algorithms for the fully cooperative multi-agent tasks, which estimate the joint Q-value function $Q(s, \mathbf{u})$ as Q_{tot} with specific forms. We consider a fully-observable setting for ease of representation. VDN factorizes Q_{tot} into a sum of individual Q-value functions. In contrast, QMIX applies a state-dependent monotonic mixing network to combine per-agent Q-value functions with the joint Q-value function. The **restricted spaces** of all $Q_{tot}(s, \mathbf{u})$ that VDN and QMIX can represent are

$$\mathcal{Q}^l := \{Q_{tot} | Q_{tot} = \sum_{a=1}^{n} Q_a(s, u_a)\}, \quad (1)$$

$$\mathcal{Q}^m := \{Q_{tot} | Q_{tot} = f_s(Q_1(s, u_1), ..., Q_n(s, u_n))\}, \quad (2)$$

where $Q_a(s, u_a) \in \mathbb{R}$, $\frac{\partial f_s}{\partial Q_a} \geqslant 0, \forall a \in A$. The monotonic mixing function f_s is parametrized as a feedforward network, whose non-negative weights are generated by hyper-networks that take the state as input. WQMIX views QMIX as an operator which first computes the **target**

$y(s,\mathbf{u}) = r + \gamma \max_{\mathbf{u}'} Q_{tot}(s',\mathbf{u}')$ at each update step, and then projects it into $\mathcal{Q}^m$. It introduces a weighting function into the projection to prioritize the optimal Q-value:

$$\Pi_{\mathrm{q}} Q := \arg\min_{Q_{tot}\in\mathcal{Q}^m} \sum_{\mathbf{u}\in\mathbf{U}} w(s,\mathbf{u})[y(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u})]^2. \quad (3)$$

Since it is computationally infeasible to obtain the optimal joint Q-value, WQMIX proposes Centrally-Weighted QMIX (CW-QMIX) and Optimistically-Weighted QMIX (OW-QMIX) to place more importance on better joint actions rather than the optimal. The weighting functions in CW-QMIX and OW-QMIX are defined as:

$$w^{cw}(s,\mathbf{u}) = \begin{cases} 1 & y(s,\mathbf{u}) > \hat{Q}^*(s,\hat{\mathbf{u}}^*) \text{ or } \mathbf{u} = \hat{\mathbf{u}}^* \\ \alpha & \text{otherwise} \end{cases}, \quad (4)$$

$$w^{ow}(s,\mathbf{u}) = \begin{cases} 1 & y(s,\mathbf{u}) > Q_{tot}(s,\mathbf{u}) \\ \alpha & \text{otherwise} \end{cases}, \quad (5)$$

where $\alpha \in (0,1]$, $\hat{\mathbf{u}}^* = \arg\max_{\mathbf{u}} Q_{tot}(s,\mathbf{u})$ is the **greedy joint action**, $\hat{Q}^*(s,\hat{\mathbf{u}}^*)$ is an approximation of the optimal Q-value which is not constrained to be monotonic.

QPLEX achieves the full representation capacity through a duelling mixing network, where the weights are produced through joint actions. Despite the complexity, it is easy to be stuck at the suboptimal because its mixing structure hinders the update of the optimal Q-values. More details can be found in Appendix B.2. In this paper, we mainly focus on the value function factorization methods without fully representing all target Q-values from the environment.

This paper focuses on the representation limitation that stems from the mixing structure of the value factorization method rather than the model capacity of the neural network.

3. Case Studies

WQMIX (Rashid et al., 2020) is the state-of-the-art value function factorization method to overcome the representational limitation without fully representing all target Q-values. In this section, we examine its weighting functions and prove it cannot guarantee convergence to the optimal in the task that the target joint Q-values are characterized as non-monotonic and stochastic. Since the non-linear mixing network in WQMIX makes many analysis methods inapplicable, we use WVDN instead of WQMIX to draw theoretical results. We then empirically show that WQMIX suffers from the same problem as WVDN in Appendix B.1.

The WVDN operator is:

$$\Pi_{\mathrm{v}} Q := \arg\min_{Q_{tot}\in\mathcal{Q}^l} \sum_{\mathbf{u}\in\mathbf{U}} w(s,\mathbf{u})[y(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u})]^2, \quad (6)$$

where $w(s,\mathbf{u})$ is the optimistic weighting from Eq. (5).

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**a**	b	b
u_2^2	b	c	c
u_2^3	b	c	c

(a) A non-monotonic game.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**8**	6	6
u_2^2	6	12/0	6
u_2^3	6	6	6

(b) A stochastic game.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**7.34**	7.29	6.06
u_2^2	7.29	7.23	6.05
u_2^3	6.05	6.03	5.92

(c) OW-QMIX(α=0.5).

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**7.33**	7.25	6.33
u_2^2	7.25	7.2	6.28
u_2^3	6.33	6.28	5.45

(d) CW-QMIX(α=0.5).

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	7.29	7.67	6.42
u_2^2	7.67	**10.9**	6.62
u_2^3	6.43	6.63	5.73

(e) OW-QMIX (α=0.1).

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**7.98**	**7.98**	6.33
u_2^2	**7.98**	**7.98**	6.33
u_2^3	6.33	6.33	5.51

(f) CW-QMIX (α=0.1).

Figure 2. (a) The payoff matrix for a one-step non-monotonic game, where $a > c > b$. (b) The payoff matrix for a one-step stochastic game. (c-f) The approximated Q_{tot} returned from WQMIX. Boldface means the real optimal joint action or the greedy joint action from the Q_{tot}.

To cope with non-monotonic target Q-values, WQMIX places the weight $\alpha \in (0,1]$ on every suboptimal action, and thus all potential optimal Q-values are prioritized. To recover the optimal joint policy, the weight α has an upper bound which is related to the reward function and the exploration rate. Consider a matrix game in Fig. 2-a. Suppose the training dataset is fixed and is collected by ϵ-greedy exploration, where the greedy action is set to (u_1^2, u_2^2). We prove that the weight α for WVDN should be smaller than $\frac{3\epsilon(a-c)+\epsilon^2(c-b)}{(\epsilon-3)^2(c-b)}$ to ensure convergence to the optimal. The proof can be found in Appendix A.2.

However, the weight α shapes the original data distribution and leads to potential learning risks in stochastic environments. This is because the weighting function prioritizes the action whose target Q-value is greater than the estimated joint Q-value from the current mixing network, which could be suboptimal. We show a numerical example as follows. Consider a matrix game in Fig. 2-b, where the optimal is $\mathbf{u}^* = (u_1^1, u_2^1)$, and the suboptimal $\mathbf{u}^s = (u_1^2, u_2^2)$ receives 12 with probability 0.5 and 0 with 0.5. We adopt uniform data distribution to ensure sufficient data collection.

Since this matrix game only involves one state, we omit the state s in the value function for simplicity. Fig. 2(c-f) demonstrate the joint Q-values $Q_{tot}(\mathbf{u})$ returned from WQMIX with different α. WQMIX with a small weight is drawn to the suboptimal $\mathbf{u}^s$ and has an incorrect argmax. OW-QMIX returns an overestimated $Q_{tot}(\mathbf{u}^s) = 10.9 > Q_{tot}(\mathbf{u}^*)$ and thus converge to the suboptimal $\mathbf{u}^s$. In contrast, CW-QMIX places more importance on the non-greedy action $\mathbf{u}^s$ when it receives the reward of 12, leading to an overestimated value $Q_{tot}(\mathbf{u}^s) > Q(\mathbf{u}^s) = 6$. With the increase of $Q_{tot}(\mathbf{u}^s)$,

$\mathbf{u}^s$ becomes the greedy action once $Q_{tot}(\mathbf{u}^s) > Q_{tot}(\mathbf{u}^*)$. In this situation, CW-QMIX does not place the α for $\mathbf{u}^s$ when it receives the reward of 0, leading to the decrease of $Q_{tot}(\mathbf{u}^s)$. As a result, CW-QMIX is stuck in this loop and cannot converge to any policy.

The weights for different actions should be uniform to anchor down the overestimated value of suboptimal. However, the weight for each suboptimal action should be small enough to solve non-monotonic targets. This contradiction may result in convergence to suboptimal.

Theorem 3.1. *Let* $\Pi_{\mathrm{v}}Q$ *be the operator defined in* Eq. (6). *Then* $\exists Q$ *such that* $\arg\max \Pi_{\mathrm{v}}Q \neq \arg\max Q$ *for any* $\alpha \in (0, 1]$.

The proof is provided in Appendix A.2. In Appendix B.1, we empirically show that WQMIX suffers from the same contradiction and cannot solve the task where the targets are characterized as non-monotonic and stochastic.

4. Method

In this section, we introduce a novel value function factorization method, Uncertainty-based Transformation (UTRAN), which eliminates the representation limitation of monotonic value function factorization without representing all target Q-values from the stochastic environment.

4.1. Uncertainty Estimation

UTRAN uses a world model that consists of three components to estimate stochasticity. A state representation model encodes the state to continuous vector-valued embeddings. Inspired by RND (Burda et al., 2018), the state representation model is a fixed and randomly initialized network, which provides the target embeddings for the transition model. The transition model predicts the embedding of the future state and its standard deviation. The reward model predicts the reward and its standard deviation,

$$\text{State representation model:} \quad h(z_t|s_t) \tag{7}$$

$$\text{Transition model:} \quad g_\theta(\hat{z}_{t+1}, \hat{\sigma}_t^z|s_t, \mathbf{u}_t) \tag{8}$$

$$\text{Reward model:} \quad v_\varphi(\hat{r}_t, \hat{\sigma}_t^r|s_t, \mathbf{u}_t). \tag{9}$$

Based on Bayesian deep learning method (Kendall & Gal, 2017), we train the transition and the reward model by:

$$L(\theta, \varphi) = \frac{1}{T}\sum_{t=0}^{T-1}\left[\frac{1}{2(\hat{\sigma}_t^r)^2}\delta_r^2 + \frac{1}{2}\log(\hat{\sigma}_t^r)^2 + \frac{1}{d}\sum_{i=1}^{d}\left(\frac{1}{2(\hat{\sigma}_t^{z,i})^2}\delta_z^2 + \frac{1}{2}\log(\hat{\sigma}_t^{z,i})^2\right)\right], \tag{10}$$

where T denotes the episode length, d denotes the dimension of the state embedding z_t, $\delta_z = z_t^i - \hat{z}_t^i$, and $\delta_r = r_t - \hat{r}_t$.

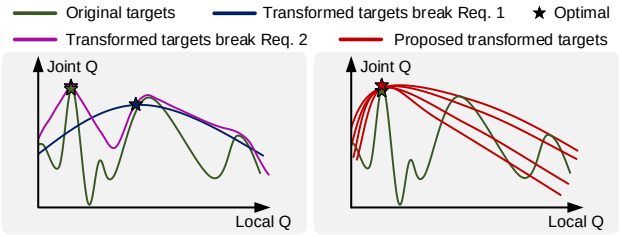

Figure 3. The relations between the original target and different transformed targets.

When the state-action pair results in stochastic transitions, the standard deviations are expected to be large. Under this interpretation, we define an indicator function $I(s, \mathbf{u}) = \mathbb{I}_{(\forall\sigma\in\{\hat{\sigma}_t^r, \hat{\sigma}_t^z\}<\alpha)}$, where α is a hyperparameter. $I(s, \mathbf{u}) = 0$ if the given state-action pair has considerable stochasticity.

4.2. Uncertainty-based Transformation

The basic idea of UTRAN is that we should ease the representation for monotonic value factorization due to its limited representational capacity. To solve this limitation, we can project the original target for joint Q-values to the restricted space that monotonic value factorization can represent by only showing the suboptimality for uncooperative actions rather than approximating their exact values.

First, we calculate the original target $y(s_t, \boldsymbol{\tau}_t, \mathbf{u}_t) = r(s_t, \mathbf{u}_t) + \gamma \max_{\mathbf{u}} Q_{tot}(s_{t+1}, \boldsymbol{\tau}_{t+1}, \mathbf{u}; \theta'_Q)$, where $Q_{tot}(s_t, \boldsymbol{\tau}_t, \mathbf{u}_t; \theta_Q) = f_s(Q_1(\tau_1, u_1), ..., Q_n(\tau_n, u_n))$ is the joint Q-value function, $\frac{\partial f_s}{\partial Q_a} \geqslant 0, \forall a \in A$, $Q_i(\tau_i, u_i)$ is the per-agent value function, θ'_Q denotes the parameters of a target network that are periodically copied from θ_Q.

Then, we introduce the transformation function:

$$Q^f(s, \boldsymbol{\tau}, \mathbf{u}) = \begin{cases} y^f(s, \boldsymbol{\tau}, \mathbf{u}) & \Delta(s, \boldsymbol{\tau}, \mathbf{u})I(s, \mathbf{u}) > 0 \\ y(s, \boldsymbol{\tau}, \mathbf{u}) & \text{otherwise} \end{cases}, \tag{11}$$

where $\Delta(s, \boldsymbol{\tau}, \mathbf{u}) = \max_{\mathbf{u}} Q_{tot}(s, \boldsymbol{\tau}, \mathbf{u}) - y(s, \boldsymbol{\tau}, \mathbf{u})$, and $y^f(s, \boldsymbol{\tau}, \mathbf{u})$ is a class of modified targets. Namely, this transformation is a general approach - we can find different $y^f(s, \boldsymbol{\tau}, \mathbf{u})$ to represent the suboptimality of the joint action. This transformation should satisfy the following two requirements: **Req. 1**, the optimal policy should not change after the transformation, and **Req. 2**, we can find a monotonic mixing function to represent all transformed targets $Q^f(s, \boldsymbol{\tau}, \mathbf{u})$. Fig. 3 illustrates that the transformation cannot guarantee convergence to the optimal policy if it breaks one of the requirements.

We only replace the suboptimal and deterministic target with $y^f(s, \boldsymbol{\tau}, \mathbf{u})$ to guarantee that we can find a mono-

tonic mixing function f_s to represent all transformed targets $Q^f(s, \boldsymbol{\tau}, \mathbf{u})$. In contrast, we remain the target unchanged if it is characterized as stochastic or greater than the current estimate of the greedy action value. The agent network is trained by minimizing the mean square error between $Q_{tot}(s_t, \boldsymbol{\tau}_t, \mathbf{u}_t; \theta_Q)$ and $Q^f(s, \boldsymbol{\tau}, \mathbf{u})$.

We introduce a practical implementation to achieve such transformation. We define the best per-agent value function to quantify the suboptimality of the joint action:

$$q_a(s, u_a; \theta_{q_a}) = \max_{\mathbf{u}_{-a}} Q(s, u_a, \mathbf{u}_{-a}), \tag{12}$$

where $q_a(s, u_a)$ is less than the optimal Q value if u_a is not part of the real optimal joint action. BAIL (Chen et al., 2020) is applied to approximate this value, and the detailed implementation can be found in Appendix C. Note that $Q_i(\tau_i, u_i)$ is the local agent Q-value for decision-making, while $q_a(s, u_a)$ is only used in training.

Then, we use the minimal value of these best per-agent value functions over all agents as the transformed target:

$$y^f(s, \boldsymbol{\tau}, \mathbf{u}) = \min\{q_a(s, u_a; \theta_{q_a})\}. \tag{13}$$

To better understand the transformation, we analyze its property and visualize the transformed targets. We consider a fully-observable setting, and our notations do not distinguish the concepts of states s and observation-action histories $\boldsymbol{\tau}$.

Theorem 4.1. *Let $Q(s, \mathbf{u})$ and $Q^f(s, \mathbf{u})$ be the original and the transformed target Q-values from* Eq. (11)*, where $y^f(s, \boldsymbol{\tau}, \mathbf{u})$ is defined in* Eq. (13)*. Suppose $\arg\max_{\mathbf{u}} Q(s, \mathbf{u})$ is unique. Then $\arg\max_{\mathbf{u}} Q^f(s, \mathbf{u}) = \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$ and $Q^f(s, \mathbf{u}) \in \mathcal{Q}^m$.*

The proof can be found in Appendix A.3.

After transformation, all targets $Q^f(s, \mathbf{u})$ lie in the restricted monotonic space $\mathcal{Q}^m$. Namely, the gap between the target and the approximated values by monotonic value factorization, i.e., the representational limitation, is eliminated. In addition, this transformation does not change the optimal action. As a result, monotonic value factorization can converge to the optimal for any original target Q-value function.

In Fig. 4, we show the performance comparison of monotonic value function factorization with original and transformed target values. Fig. 4-a shows the rewards of a 10×10 matrix game, where the suboptimal is filled with random numbers generated uniformly between -10 and 9, and the unique optimal value is +10. These rewards denote the original targets. Fig. 4-b demonstrates the transformed targets. Fig. 4-c and d demonstrate that monotonic value factorization with the original targets cannot converge due to the representational limitation. In contrast, it can perfectly represent all transformed targets by showing that the loss is almost zero and converges to the optimal after 400 iterations.

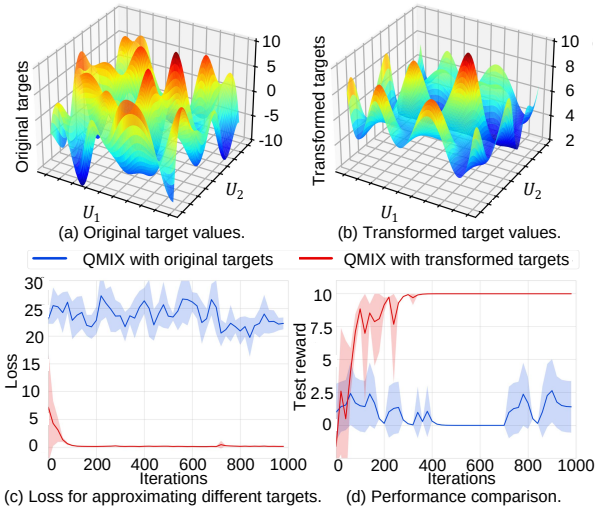

Figure 4. Comparison between the original and the transformed target joint Q-values.

5. Related work

This section briefly introduces recent related work on cooperative multi-agent reinforcement learning (MARL) in the paradigm of centralized training with decentralized execution (CTDE). One of the most significant challenges in CTDE is to ensure the correspondence between the individual Q-value functions and the joint Q-value function Q_{tot}, i.e., the Individual-Global Max (IGM) principle (Son et al., 2019). VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018) learn the joint Q-values and factorize them into individual Q-value functions in an additive and a monotonic fashion, respectively. Qatten (Yang et al., 2020b) is a variant of QMIX, which applies a multi-head attention structure to the mixing network. QPD (Yang et al., 2020a) utilizes integrated gradients to decompose Q_{tot} along trajectory paths. SMIX(λ) (Yao et al., 2021) changes the one-step Q-learning target with a SARSA(λ) target for QMIX. RODE (Wang et al., 2020b) decomposes joint action spaces into restricted role action spaces to boost learning efficiency and policy generalization. These methods apply the same monotonic mixing network and thus can only represent the same class as QMIX. However, as many previous studies pointed out, monotonic value function factorization limits the representational capacity of Q_{tot}, and fails to learn the optimal policy when the target Q-value functions are non-monotonic (Mahajan et al., 2019; Son et al., 2019; Rashid et al., 2020).

Some recent works try to achieve the full representational capacity of Q_{tot} to solve this problem. QPLEX (Wang et al., 2020a) proposes a duelling mixing network, in which the weights are produced through joint actions. Deep coordination graph (Böhmer et al., 2020) decomposes the Q_{tot} into individual utilities and payoff contributions based on

the actions of the agents connected by the (hyper-)edges. Tesseract (Mahajan et al., 2021) decomposes the Q-tensor across agents and utilises low-rank tensor approximations to model agent interactions relevant to the task, and thus learns a compact approximation of the target Q-value function. However, since the dimension of the state-action space increases exponentially as the number of agents grows, it is not easy to achieve the full representational capacity in complex MARL tasks.

Another solution is to learn a biased Q_{tot} by prioritizing the optimal joint action. QTRAN (Son et al., 2019) uses two soft regularisations to align the greedy action selections between the joint Q-value and the individual values. WQMIX (Rashid et al., 2020) introduces a weighting mechanism to place more importance on better joint actions. QTRAN can be viewed as a variant of WQMIX, which additionally uses the weight 0 for the joint Q-value whose target is smaller than the current estimate and the chosen action is not greedy. QTRAN approximates $\hat{Q}^*$ as the target instead of the original target y and thus can deal with stochasticity. However, due to the 0 weight for overestimated Q-values, QTRAN is empirically hard to scale to more complex tasks (Samvelyan et al., 2019). We also prove that there may not exist an appropriate weight when the targets are non-monotonic and stochastic.

In addition, there have been many developments in policy-based methods under CTDE settings. MAPPO (Yu et al., 2021) applies PPO (Schulman et al., 2017) into MARL and shows strong empirical performance. However, Kuba et al. (2021) points out MAPPO suffers from instability arising from the non-stationarity induced by simultaneously learning and exploring agents. Therefore, they introduce the sequential policy update scheme to achieve monotonic improvement on the joint policy. However, this method can only guarantee convergence to one of the Nash Equilibriums and falls into the suboptimal if this solution cannot be improved by coordinate descent (Bertsekas, 2019). This problem can be interpreted as the non-monotonic target problem in value function factorization methods.

Relationship To Reward Shaping. Reward shaping is a common technique for improving single-agent learners' performance by integrating expert knowledge into MDP (Gullapalli & Barto, 1992). Since it is not always possible to find an expert with complete domain knowledge, setting rewards for complex MARL tasks in advance is difficult. Therefore, shaping the reward adaptively is a more attractive way. Some reward randomisation methods (Gupta et al., 2021; Tang et al., 2021) are proposed to convert the original MDP to a new MDP with random rewards through universal successor features. However, these methods require a massive number of randomisation to find a desirable reward function. GVR (Wan et al., 2022) proposes inferior target reshaping and superior experience replay to eliminate the non-optimal self-transition nodes, which is similar to WQMIX and ignores the stochasticity of the transition. Compared with these methods, UTRAN is more efficient because it can perfectly represent the expected value for stochastic targets and guarantee policy invariance during target shaping.

Relationship To Independent Learning Algorithms. Some independent learning algorithms have proven robust to solve relative overgeneralization in the low-dimensional setting. Distributed Q-learning (Lauer, 2000) and Hysteretic Q-learning (Matignon et al., 2007) place more importance on positive updates that increase a Q-value estimate, which is similar to the weighting function in WQMIX. However, Wei & Luke (2016) prove that these methods are vulnerable towards misleading stochasticity and propose LMRL2 (Wei & Luke, 2016), where agents forgive the other's miscoordination in the initial exploration phase but become less lenient when the visitation of state-action pair increases. However, it requires carefully tuned hyperparameters that rarely translate across domains. Best possible Q-learning (Jiang & Lu, 2023) computes the expected values of all possible transition probabilities and updates the state-action value to be the maximal one, which is similar to the best per-agent value function we used for transformation. However, this method achieves lower performance than UTRAN in the team-reward task because it cannot guarantee consistency between decentralized policies and ignores credit assignment, which is verified in our ablation study (UBEST in Sec. 6.3).

6. Experiments

We conduct empirical experiments to answer the following questions: 1) Is UTRAN better than the existing methods target Q-values are seriously non-monotonic and stochastic? 2) Can UTRAN perform efficient coordination in challenging multi-agent tasks? 3) What is the contribution of each component in UTRAN? We compare UTRAN with VDN (Sunehag et al., 2018), QMIX (Rashid et al., 2018), QTRAN (Son et al., 2019), QPLEX (Wang et al., 2020a), WQMIX (Rashid et al., 2020), and MAPPO (Yu et al., 2021) on matrix games, Predator-Prey (Son et al., 2019), and StarCraft II Multi-agent Challenge (SMAC) (Samvelyan et al., 2019). We also compare UTRAN with GVR (Wan et al., 2022) and WQMIX with different weights in Appendix D. Detailed experimental settings are provided in Appendix G.

6.1. Toy Examples

We consider a matrix game in Fig. 5-a, where the local optimal is difficult to jump out due to the considerable miscoordination penalties and stochastic reward. We adopt a uniform exploration strategy to approximate the uniform data distribution and eliminate the challenge of exploration

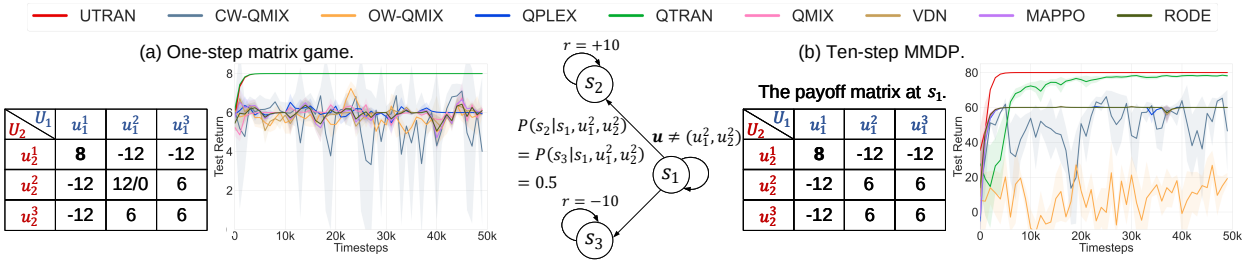

Figure 5. Performance comparisons on a one-step matrix game and a ten-step MMDP.

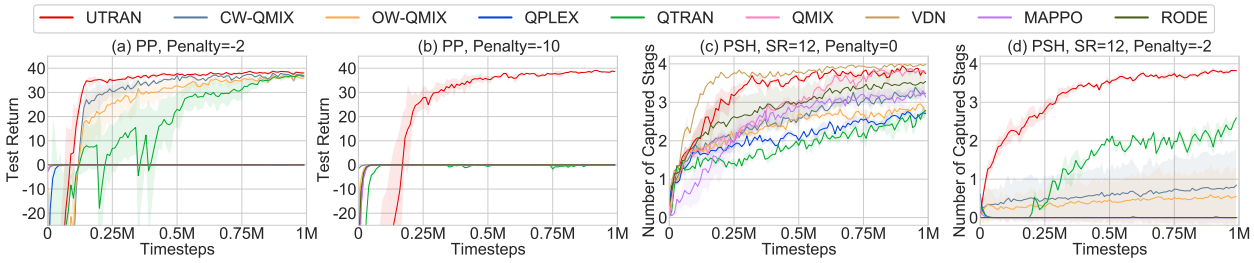

Figure 6. Performance comparisons on Predator-Prey (PP) and Predator-Stag-Hare (PSH).

and sample complexity. It is observed that only UTRAN and QTRAN can achieve optimal performance. CW-QMIX fluctuates dramatically and cannot converge to any policy, which is consistent with our analysis in Section 3.

We also consider a Multi-Agent Markov Decision Process (MMDP) with non-monotonic and stochastic targets in Fig. 5-b. This MMDP involves two agents, three actions and a team reward. Two agents start at state s_1 and explore extrinsic rewards for ten environment steps. The state transits from s_1 to s_2 with probability 50% and s_3 with 50% if agents perform $\mathbf{u} = (u_1^2, u_2^2)$ at s_1, and remains unchanged otherwise. The team continually obtains a deterministic reward of +10 and -10 at s_2 and s_3, respectively. Since QMIX, QPLEX, VDN, and MAPPO estimate the stochastic targets well but cannot deal with the non-monotonicity, they converge to the suboptimal. The results show that CW-QMIX and OW-QMIX converge to the stochastic suboptimal due to the intractable weights. In contrast, UTRAN and QTRAN achieve optimal performance, where UTRAN learns faster than QTRAN by a considerable margin.

6.2. Performance Comparisons

In this section, we compare UTRAN with our baselines on Predator-Prey and SMAC benchmarks.

Predator-Prey raises challenges of partial observability and *relative overgeneralization* pathology, namely, the target is non-monotonic. Since the reward observed by an agent is highly related to the actions of others, the learning of decentralized policies is unstable due to the exploration of other agents. Fig. 6-a shows that WQMIX, QTRAN, and UTRAN can solve the task when the penalty for miscoordination is 2, while others fail to return positive returns. However, only UTRAN can solve the task when the penalty is -10, suggesting difficulties in choosing the weight and the network architecture in advance for WQMIX and QTRAN.

We extend Predator-Prey to a stochastic environment named Predator-Stag-Hare, which consists of eight predators, four stags, and four hares. Despite the miscoordination penalty for stags, a catch of hare results in a stochastic reward, which is equal to SR$\in \mathbb{R}^+$ and -SR with the same chance 50%. Fig. 6-c and d show that VDN and QMIX can quickly recover the optimal policy when the penalty is 0 because they can accurately evaluate the stochastic reward. When the penalty is -2, VDN, QMIX, QPLEX, and WQMIX fail to solve this task. QTRAN also struggles with the suboptimal in this setting. In contrast, UTRAN learns quickly and reliably in these tasks because it avoids overestimating stochastic values and eliminates the adverse impact of the miscoordination penalty by target transformation.

We also compare UTRAN with our baselines on hard and super-hard maps in SMAC. Fig. 7 shows the improved performance of UTRAN, which indicates that the target transformation can be applied to more general benchmarks. More results on SMAC can be found in Appendix F.

6.3. Ablation Study

In this section, we conduct an ablation study to demonstrate the contribution of each component in UTRAN. We compare UTRAN with its variants: 1) UTRAN without target transformation (UBEST), where agents choose ac-

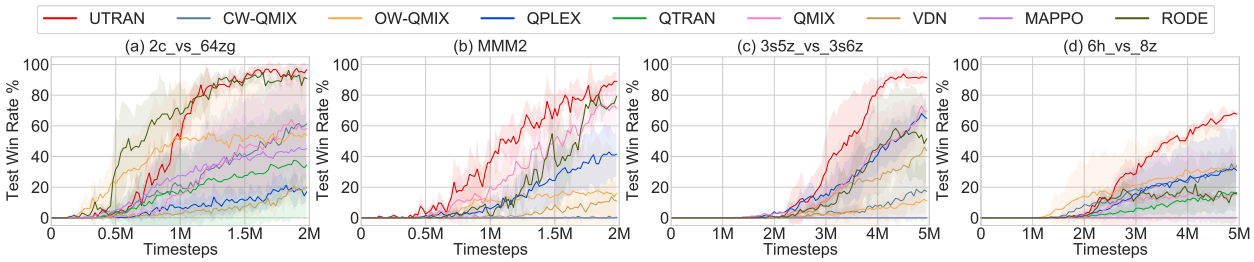

Figure 7. Performance comparisons on SMAC.

tions based on the best per-agent value function, 2) UTRAN without the transition model (UTRAN-wo-T), 3) UTRAN without the reward model (UTRAN-wo-R), and 4) QMIX, the natural ablation of UTRAN.

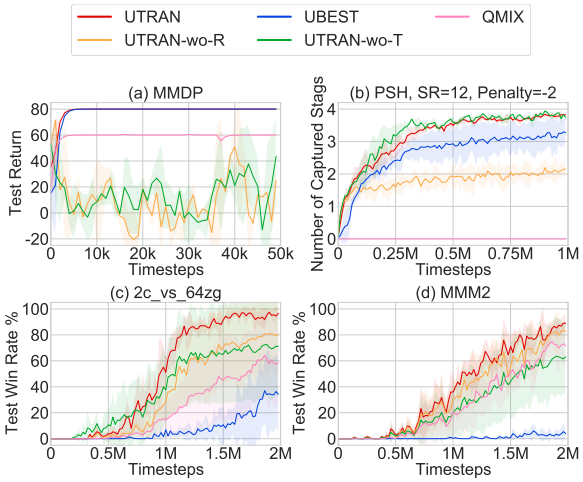

Figure 8. Ablation results on MMDP, PSH, and SMAC.

Fig. 8 shows that UTRAN-wo-R cannot solve MMDP and PSH because these tasks involve stochastic rewards. The negative results in MMDP for UTRAN-wo-S demonstrate that it cannot identify stochastic state transitions. However, UTRAN-wo-S converge to the optimal quickly on PSH because this task only involves miscoordination penalty and stochastic rewards. Despite learning steadily on MMDP and PSH, UBEST fails to achieve good performance on SMAC and takes far longer to reach UTRAN and other variants. In contrast, UTRAN outperforms its variants across all tasks, demonstrating the effectiveness of the uncertainty-based target transformation for monotonic value factorization.

Here we emphasize the contribution of target transformation by analyzing why UTRAN outperforms UBEST. The first reason is that UTRAN realizes full representational capacity for the transformed targets, while UBEST does not have this advantage. The second one is that UBEST does not apply monotonic value function factorization in policy learning. Value factorization methods guarantee consistency between decentralized policies and implicitly achieve credit assignment using counterfactual baselines. In contrast, UBEST introduces additional estimation errors that may cause significant variances, leading to slow convergence when the most and the second greatest target value is close.

To verify our analysis, we consider a one-step 10×10 matrix game, where the suboptimal is filled with random numbers generated uniformly between -20 and 19, and the unique optimal value is +20. Fig. 9 shows that UTRAN has lower standard deviations than UBEST and QMIX, implying that monotonic value function factorization with transformed targets can reduce variances.

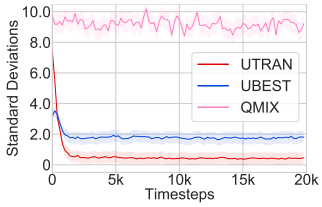

Figure 9. Standard deviations of per-agent action values.

7. Conclusion and Future Work

This paper presents Uncertainty-Based Transformation (UTRAN), which was inspired by analyzing the representation limitation of monotonic value factorization and the weighting function in Weighted QMIX (WQMIX). We first prove that there may not exist an appropriate weight for WQMIX to recover the optimal policy if the targets are stochastic and non-monotonic. We introduce an alternative solution to eliminating the representational limitation, which projects the joint Q-value target to the space that monotonic value factorization can represent. UTRAN approximates the reward and the further state for a state-action pair and quantifies the uncertainty of the target Q-value function by the predicted standard deviations. Then UTRAN replaces the suboptimal targets with large uncertainty by the best per-agent value. Empirical results demonstrate the improved ability of UTRAN to cope with the task that the target is characterized as non-monotonic and stochastic. If the task involves multiple optimal policies, a more efficient exploration method could be considered, as opposed to the simple ϵ-greedy exploration scheme we used in this paper.

References

Aladdin, S., El-Tantawy, S., Fouda, M. M., and Eldien, A. S. T. MARLA-SG: multi-agent reinforcement learning algorithm for efficient demand response in smart grid. *IEEE Access*, 8:210626–210639, 2020.

Bertsekas, D. Multiagent rollout algorithms and reinforcement learning. *arXiv preprint arXiv:1910.00120*, 2019.

Böhmer, W., Kurin, V., and Whiteson, S. Deep coordination graphs. In *International Conference on Machine Learning*, pp. 980–991. PMLR, 2020.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Chen, X., Zhou, Z., Wang, Z., Wang, C., Wu, Y., and Ross, K. Bail: Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:18353–18363, 2020.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.1, 2018.

Foerster, J. N., Assael, Y. M., de Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2145–2153, 2016.

Gullapalli, V. and Barto, A. G. Shaping as a method for accelerating reinforcement learning. In *Proceedings of the 1992 IEEE international symposium on intelligent control*, pp. 554–559. IEEE, 1992.

Gupta, T., Mahajan, A., Peng, B., Böhmer, W., and Whiteson, S. Uneven: Universal value exploration for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 3930–3941. PMLR, 2021.

Jiang, J. and Lu, Z. Best possible q-learning, 2023.

Kendall, A. and Gal, Y. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

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

Kuba, J. G., Chen, R., Wen, M., Wen, Y., Sun, F., Wang, J., and Yang, Y. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

Lauer, M. An algorithm for distributed reinforcement learning in cooperative multiagent systems. In *Proc. 17th International Conf. on Machine Learning*, 2000.

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

Mahajan, A., Samvelyan, M., Mao, L., Makoviychuk, V., Garg, A., Kossaifi, J., Whiteson, S., Zhu, Y., and Anandkumar, A. Tesseract: Tensorised actors for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 7301–7312. PMLR, 2021.

Matignon, L., Laurent, G. J., and Le Fort-Piat, N. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 64–69. IEEE, 2007.

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

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

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

Rashid, T., Farquhar, G., Peng, B., and Whiteson, S. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33: 10199–10210, 2020.

Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

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

Simchowitz, M. and Jamieson, K. G. Non-asymptotic gap-dependent regret bounds for tabular mdps. *Advances in Neural Information Processing Systems*, 32, 2019.

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

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2085–2087, 2018.

Tang, Z., Yu, C., Chen, B., Xu, H., Wang, X., Fang, F., Du, S. S., Wang, Y., and Wu, Y. Discovering diverse multi-agent strategic behavior via reward randomization. In *International Conference on Learning Representations*, 2021.

Wan, L., Liu, Z., Chen, X., Lan, X., and Zheng, N. Greedy based value representation for optimal coordination in multi-agent reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 22512–22535. PMLR, 17–23 Jul 2022.

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

Wang, T., Gupta, T., Mahajan, A., Peng, B., Whiteson, S., and Zhang, C. Rode: Learning roles to decompose multi-agent tasks. *arXiv preprint arXiv:2010.01523*, 2020b.

Wei, E. and Luke, S. Lenient learning in independent-learner stochastic cooperative games. *The Journal of Machine Learning Research*, 17(1):2914–2955, 2016.

Yang, Y., Hao, J., Chen, G., Tang, H., Chen, Y., Hu, Y., Fan, C., and Wei, Z. Q-value path decomposition for deep multiagent reinforcement learning. In *International Conference on Machine Learning*, pp. 10706–10715. PMLR, 2020a.

Yang, Y., Hao, J., Liao, B., Shao, K., Chen, G., Liu, W., and Tang, H. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv preprint arXiv:2002.03939*, 2020b.

Yao, X., Wen, C., Wang, Y., and Tan, X. Smix (λ): Enhancing centralized value functions for cooperative multiagent reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., and Wu, Y. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

Zhou, M., Luo, J., Villella, J., Yang, Y., Rusu, D., Miao, J., Zhang, W., Alban, M., FADAKAR, I., Chen, Z., et al. Smarts: An open-source scalable multi-agent rl training school for autonomous driving. In *Conference on Robot Learning*, pp. 264–285. PMLR, 2021.

A. Proofs

The operator of weighted linear value function factorization (WVDN) is defined as

$$\Pi_{\mathrm{v}} Q := \arg\min_{Q_{tot} \in \mathcal{Q}^l} \sum_{\mathbf{u} \in \mathbf{U}} w(s, \mathbf{u})[y(s, \mathbf{u}) - Q_{tot}(s, \mathbf{u})]^2, \tag{14}$$

where $\mathcal{Q}^l := \{Q_{tot} | Q_{tot}(s, \mathbf{u}) = \sum_{a=1}^{n} Q_a(s, u_a), Q_a(s, u) \in \mathbb{R}\}$, $w(s, \mathbf{u})$ is the optimistic weighting:

$$w^{ow}(s, \mathbf{u}) = \begin{cases} 1 & y(s, \mathbf{u}) > Q_{tot}(s, \mathbf{u}) \\ \alpha & \text{otherwise} \end{cases}. \tag{15}$$

The optimizing of WVDN can be constructed as a weighted linear regression problem:

$$\min_{\mathbf{x}} \|\sqrt{\mathbf{w}^\top} \cdot \sqrt{\mathbf{p}^\top} \cdot (\mathbf{A}\mathbf{x} - \mathbf{b})\|, \tag{16}$$

where $\mathbf{A} \in \mathbb{R}^{m^n \times mn}$, $\mathbf{x} \in \mathbb{R}^{mn}$, $\mathbf{w}$,$\mathbf{p}$,$\mathbf{b} \in \mathbb{R}^{m^n}$, m,$n \in Z^+$. In particular, m dentoes the action space size for each agent, n denotes the number of agents.

Denote $\mathbf{A}^w = \sqrt{\mathbf{w}^\top} \cdot \sqrt{\mathbf{p}^\top} \cdot \mathbf{A}$, $\mathbf{b}^w = \sqrt{\mathbf{w}^\top} \cdot \sqrt{\mathbf{p}^\top} \cdot \mathbf{b}$. The optimal solution is:

$$\mathbf{x}^* = \mathbf{A}^{w,\dagger} \mathbf{b}^w, \tag{17}$$

where $\mathbf{A}^{w,\dagger} = V S^\dagger U^\top$ is the pseudo-inverse of $\mathbf{A}^w$ according to the Singular Value Decomposition. $U \in \mathbb{R}^{m^n \times m^n}$ and $V \in \mathbb{R}^{mn \times mn}$ are real orthogonal matrices, $S^\dagger \in \mathbb{R}^{m^n \times mn}$ is a diagonal matric with non-negative real numbers on the diagonal.

The proof is trivial.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**a**	b	b
u_2^2	b	c	c
u_2^3	b	c	c

(a) A non-monotonic matrix game.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**a**	b	b
u_2^2	b	c/d	e
u_2^3	b	e	e

(b) A stochastic matrix game.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**8**	6	6
u_2^2	6	12/0	6
u_2^3	6	6	6

(c) Payoff of a stochastic matrix game.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**8**	-12	-12
u_2^2	-12	12/0	6
u_2^3	-12	6	6

(d) Payoff of a stochastic and non-monotonic game.

Figure 10. Payoffs of matrix games. (a) is a non-monotonic matrix game, where $a > c > b$; (b) is a stochastic matrix game, where (u_1^2, u_2^2) receives c with probability p and d with $(1-p)$, $c > a > \max\{b, d, e\}$ and $a > pc + (1-p)d$; (c-d) are two numerical examples for stochastic matrix games, where the reward in (c) is monotonic and in (d) is non-monotonic.

A.1. Linear Value Function Factorization

Consider the full exploration scheme (i.e., ϵ-greedy exploration with $\epsilon = 1$), and the non-monotonic game in Fig. 10a, where $a = 8, b = -12, c = 6$. Denote the joint action $\mathbf{u} = [u_1, ..., u_a, ..., u_n]$, where n is the number of agents. Denote u_a^i as the i-th action of u_a.

For VDN (a special case of WVDN where $\alpha = 1$), we have

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} Q_1(s, u_1^1) \\ Q_1(s, u_1^2) \\ Q_1(s, u_1^3) \\ Q_2(s, u_2^1) \\ Q_2(s, u_2^2) \\ Q_2(s, u_2^3) \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 8 \\ -12 \\ -12 \\ -12 \\ 6 \\ 6 \\ -12 \\ 6 \\ 6 \end{pmatrix}, \mathbf{p} = \mathbf{w} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{18}$$

According to Eq. (17), the optimal solution is

$$\mathbf{x}^* = \begin{pmatrix} -4.44 & 0.89 & 0.89 & -4.44 & 0.89 & 0.89 \end{pmatrix}^\top. \tag{19}$$

Clearly, $Q_i(u_i^1) < \max\{Q_i(u_i^2), Q_1(u_i^3)\}, \forall i \in \{1, 2\}$, which indicates that VDN fails to converge to the optimal.

A.2. Weighted Linear Value Function Factorization

This section shows the omitted proofs in Section 3.

Consider matrix games in Fig. 10. Suppose the training dataset is fixed and the data has a ϵ-greedy distribution, where the greedy action is (u_1^2, u_2^2) and $\epsilon \in (0, 1]$. The probability of each action is:

$$P(u_1^i, u_2^j) = \begin{cases} \dfrac{\epsilon^2}{9}, & (i,j) \in \{(1,1),(1,3),(3,1),(3,3)\}, \\ \dfrac{(3-2\epsilon)^2}{9}, & (i,j) = (2,2), \\ \dfrac{(3-2\epsilon)\epsilon}{9}, & \text{otherwise.} \end{cases} \tag{20}$$

To recover the optimal policy, the bound of the weight α in WVDN is related to the exploration rate ϵ and the reward space. Since these matrix games only involve one state, we omit s for the input of value functions in the following proofs.

Proposition 1. The weight α in WVDN should be smaller than $\frac{3\epsilon(a-c)+\epsilon^2(c-b)}{(\epsilon-3)^2(c-b)}$ to converge to the optimal on the non-monotonic matrix game in Fig. 10-a.

Proof. First, we compute the gradient for $Q(u_1^1), Q(u_1^2), Q(u_1^3)$:

$$\begin{aligned} \nabla Q(u_1^1) = \frac{\epsilon}{9}[&(\epsilon + (3-\epsilon)\alpha)Q(u_1^1) + \epsilon Q(u_2^1) + (3-2\epsilon)\alpha Q(u_2^2) + \epsilon\alpha Q(u_2^3) \\ &- \epsilon a - \alpha(3-\epsilon)b] \end{aligned} \tag{21}$$

$$\nabla Q(u_1^2) = \frac{(3-2\epsilon)\alpha}{9}[3Q(u_1^2) + \epsilon(Q(u_2^1) + Q(u_2^3)) + (3-2\epsilon)Q(u_2^2) - \epsilon b - (3-\epsilon)c] \tag{22}$$

$$\nabla Q(u_1^3) = \frac{\epsilon\alpha}{9}[3Q(u_1^3) + \epsilon(Q(u_2^1) + Q(u_2^3)) + (3-2\epsilon)Q(u_2^2) - \epsilon b - (3-\epsilon)c] \tag{23}$$

Let $\nabla Q(u_1^1) = \nabla Q(u_1^2) = \nabla Q(u_1^3) = 0$:

$$Q(u_1^1) = \frac{\epsilon a + \alpha(3-\epsilon)b - \epsilon Q(u_2^1) - (3-2\epsilon)\alpha Q(u_2^2) - \epsilon\alpha Q(u_2^3)}{\epsilon + 3\alpha - \epsilon\alpha} \tag{24}$$

$$Q(u_1^2) = Q(u_1^3) = \frac{\epsilon b + (3-\epsilon)c - \epsilon(Q(u_2^1) + Q(u_2^3)) - (3-2\epsilon)Q(u_2^2)}{3} \tag{25}$$

To recover the optimal joint policy, the following conditions should be satisfied:

$$Q(u_1^1) - \max\{Q(u_1^2), Q(u_1^3)\} > 0, Q(u_2^1) - \max\{Q(u_2^2), Q(u_2^3)\} > 0 \tag{26}$$

Denote $t = \epsilon + 3\alpha - \epsilon\alpha$. Then

$$\begin{aligned} &Q(u_1^1) - \max\{Q(u_1^2), Q(u_1^3)\} \\ &= \frac{\epsilon a + \alpha(3-\epsilon)b - \epsilon Q(u_2^1) - (3-2\epsilon)\alpha Q(u_2^2) - \epsilon\alpha Q(u_2^3)}{t} \\ &- \frac{\epsilon b + (3-\epsilon)c - \epsilon(Q(u_2^1) + Q(u_2^3)) - (3-2\epsilon)Q(u_2^2)}{3} \\ &< \underbrace{(\frac{t + \epsilon\alpha - \epsilon - 3\alpha}{t})}_{=0} Q(u_2^1) + \frac{\epsilon}{t}a + \frac{(9-3\epsilon)\alpha - \epsilon t}{3t}b + \frac{\epsilon - 3}{3}c. \end{aligned} \tag{27}$$

Let $Q(u_1^1) - \max\{Q(u_1^2), Q(u_1^3)\} > 0$ and $Q(u_2^1) - \max\{Q(u_2^2), Q(u_2^3)\} > 0$.

Then $\alpha < \frac{3\epsilon(a-c)+\epsilon^2(c-b)}{(\epsilon-3)^2(c-b)}$. □

Theorem 1. Let $\Pi_{\text{v}}Q$ be the operator defined in Eq. (6). Then $\exists Q$ such that $\arg\max \Pi_{\text{v}}Q \neq \arg\max Q$ for any $\alpha \in (0, 1]$.

Proof. Consider the stochastic matrix game in Fig. 10-b.

First, we compute the gradient for $Q(u_1^1), Q(u_1^2), Q(u_1^3)$:

$$\begin{aligned}\nabla Q(u_1^1) = \frac{\epsilon}{9}[&(\epsilon + 3\alpha - \epsilon\alpha)Q(u_1^1) + \epsilon Q(u_2^1) + (3 - 2\epsilon)\alpha Q(u_2^2) + \epsilon\alpha Q(u_2^3) - \\ &\epsilon a - \alpha(3 - \epsilon)b] \end{aligned} \tag{28}$$

$$\begin{aligned}\nabla Q(u_1^2) = \frac{3 - 2\epsilon}{9}[&((3 - 2\epsilon)(1 - \alpha)p + 3\alpha)Q(u_1^2) + \epsilon\alpha(Q(u_2^1) + Q(u_2^3)) \\ &+ (3 - 2\epsilon)(p + \alpha - p\alpha)Q(u_2^2) - \epsilon\alpha b - (3 - 2\epsilon)(pc + \alpha d - \alpha pd) - \epsilon\alpha e] \end{aligned} \tag{29}$$

$$\nabla Q(u_1^3) = \frac{\epsilon\alpha}{9}[3Q(u_1^3) + \epsilon(Q(u_2^1) + Q(u_2^3)) + (3 - 2\epsilon)Q(u_2^2) - \epsilon b - (3 - \epsilon)e] \tag{30}$$

To recover the optimal joint policy, the following conditions should be satisfied:

$$Q(u_1^1) - \max\{Q(u_1^2), Q(u_1^3)\} > 0, Q(u_2^1) - \max\{Q(u_2^2), Q(u_2^3)\} > 0$$

Let $\nabla Q(u_1^1) = \nabla Q(u_1^2) = \nabla Q(u_1^3) = 0$. Denote $t = \epsilon + 3\alpha - \epsilon\alpha$, $m = (3 - 2\epsilon)(1 - \alpha)p + 3\alpha$. Then

$$Q(u_1^1) - Q(u_1^2) < \frac{\epsilon}{t}a + \frac{(3 - \epsilon)\alpha}{t}b - \frac{\epsilon\alpha}{m}(b + e) - \frac{(3 - 2\epsilon)p}{m}c - \frac{(3 - 2\epsilon)(1 - p)\alpha}{m}d, \tag{31}$$

$$Q(u_1^1) - Q(u_1^3) < \frac{\epsilon}{t}a + \frac{(3 - \epsilon)\alpha}{t}b - \frac{\epsilon}{3}b - \frac{(3 - \epsilon)p}{3}e. \tag{32}$$

Consider the following situations.

(1) The target is monotonic but stochastic. For example, in Fig. 10c, $a = 8$, $b = e = 6$, $c = 12$, $d = 0$, and $p = 0.5$. To recover the optimal joint action, the weight α should be greater than $(0.434, 0.667, 0.827, 0.932)$ when $\epsilon = (1, 0.75, 0.5, 0.25)$, respectively.

(2) The target is non-monotonic and stochastic. For example, in Fig. 10d, $a = 8, b = -12, c = 12, d = 0, e = 6$, and $p = 0.5$. When $\epsilon = (1, 0.75, 0.5, 0.25)$, there does not exist the weight $\alpha \in (0, 1]$ making $Q(u_1^1) - \max\{Q(u_1^2), Q(u_1^3)\} > 0$, i.e., WVDN cannot converge to the optimal.

Thus, $\exists Q$ such that $\arg\max \Pi_{\text{v}}Q \neq \arg\max Q$ for any $\alpha \in (0, 1]$. □

A.3. Target Transformation

Theorem 4.1 *Let $Q(s, \mathbf{u})$ and $Q^f(s, \mathbf{u})$ be the orignal and the transformed target Q-values from* Eq. (11)*, where $y^f(s, \boldsymbol{\tau}, \mathbf{u})$ is defined in* Eq. (13)*. Suppose $\arg\max_{\mathbf{u}} Q(s, \mathbf{u})$ is unique. Then $\arg\max_{\mathbf{u}} Q^f(s, \mathbf{u}) = \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$ and $Q^f(s, \mathbf{u}) \in \mathcal{Q}^m$.*

Proof. We consider only a fully-observable setting for ease of representation. Thus our notations do not distinguish the concepts of states and observation-action histories. In addition, when more than one optimal policy exists, most Q-learning algorithms fail to converge to a stable point (Simchowitz & Jamieson, 2019; Wang et al., 2020a; Rashid et al., 2020). Thus, we consider an additional assumption in Assumption 1.

Assumption 1 (Unique Optimal Solution)**.** For each state $s \in S$, the optimal policy $\mathbf{u}_Q^* = \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$ is unique.

Under this assumption, assume $\max_{\mathbf{u}_{-a}} Q(s, u_a, \mathbf{u}_{-a})$ for all u_a are perfectly represented.

Then $Q(s, \mathbf{u}) \leqslant \min_{a\in\mathbb{A}}\{\max_{\mathbf{u}_{-a}} Q(s, u_a, \mathbf{u}_{-a})\}$.

According to the transformation defined in Eq. (11), we have:

$$\begin{cases} Q^f(s, \mathbf{u}_Q^*) = Q(s, \mathbf{u}_Q^*) \\ Q^f(s, \mathbf{u}) = \min\{\max_{\mathbf{u}_{-i}} Q(s, u_i, \mathbf{u}_{-i})\}, \forall u_i \in \mathbf{u} \neq \mathbf{u}_Q^* \text{ and } u_i \notin \mathbf{u}_Q^* \end{cases}. \tag{33}$$

Then, $Q^f(s, \mathbf{u}^*_Q) > Q^f(s, \mathbf{u})$ for all $\mathbf{u} \neq \mathbf{u}^*_Q$. Therefore, $\arg\max_{\mathbf{u}} Q^f(s, \mathbf{u}) = \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$.

Suppose $\exists f_s(s, Q_1(s, u_1), ..., Q_n(s, u_n)) = Q^f(s, \mathbf{u}), \forall \mathbf{u} = (u_1, ..., u_n) \in \mathbf{u}$. Then

$$\begin{cases} f_s(s, Q_a(u_a), \boldsymbol{Q}_{-a}(\mathbf{u}_{-a})) = \min\{\max_{\mathbf{u}_{-a}} Q(s, u_a, \mathbf{u}_{-a}), \max_{\mathbf{u}_{-i}} Q(s, u_i, \mathbf{u}_{-i})\}, \forall i \in -a \\ f_s(s, Q_a(u'_a), \boldsymbol{Q}_{-a}(\mathbf{u}_{-a})) = \min\{\max_{\mathbf{u}_{-a}} Q(s, u'_a, \mathbf{u}_{-a}), \max_{\mathbf{u}_{-i}} Q(s, u_i, \mathbf{u}_{-i})\}, \forall i \in -a \end{cases}, \tag{34}$$

where $\boldsymbol{Q}_{-a}(\mathbf{u}_{-a}) = \{Q_i(s, u_i)\}_{i \in -a}$.

Let $Q_a(s, \mathbf{u}_a) \geqslant Q_a(s, \mathbf{u}'_a)$ if $\max_{\mathbf{u}_{-a}} Q(s, u_a, \mathbf{u}_{-a}) \geqslant \max_{\mathbf{u}_{-a}} Q(s, u'_a, \mathbf{u}_{-a}), \forall a \in A$.

Then, according to Eq. (34), $f_s(s, Q_a(u_a), \boldsymbol{Q}_{-a}(\mathbf{u}_{-a})) \geqslant f_s(s, Q_a(u'_a), \boldsymbol{Q}_{-a}(\mathbf{u}_{-a}))$.

Thus we can find $f_s(s, Q_1(s, u_1), ..., Q_n(s, u_n))$ to represent $Q^f(s, \mathbf{u})$, $\forall \mathbf{u} = (u_1, ..., u_n) \in \mathbf{u}$, where $\frac{\partial f_s}{\partial Q_a} \geqslant 0$. And so $Q^f(s, \mathbf{u}) \in \mathcal{Q}^m, \forall s \in S, \forall \mathbf{u} \in \mathbf{u}$. □

We discuss the case that the Unique Optimal Solution defined in Assumption 1 does not hold in Appendix B.3, which brings a limitation for all value factorization methods.

B. More Results on Matrix Games

B.1. WQMIX with Different Weights

In this section, we empirically verify that WVDN and WQMIX suffer from the contradiction of the choice of α and converge to the suboptimal whose target is characterized as non-monotonic and stochastic.

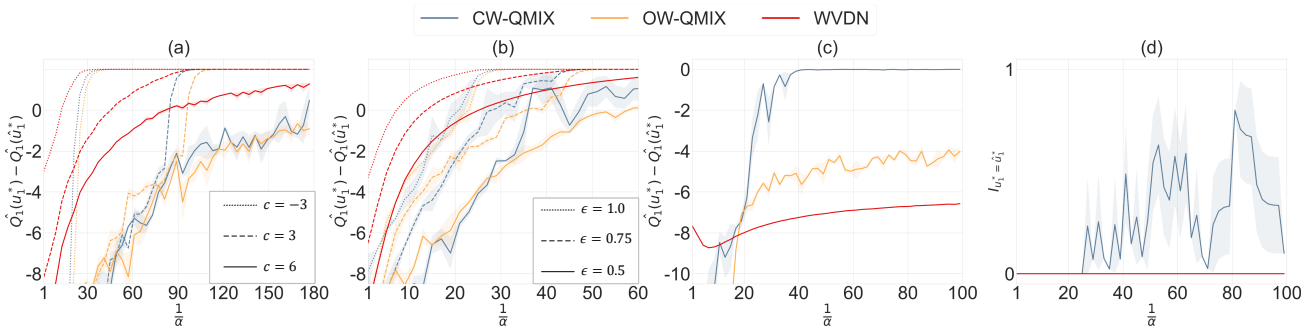

Figure 11. (a-b) Performance comparisons between WVDN and WQMIX on the game in Figure 10-a. (c-d) Performance comparisons between WVDN and WQMIX on the game in Figure 10-b, where $a = 8, b = -12, c = 20, d = -20, e = 6$. (a) The performance with different c, where $a = 8, b = -12, \epsilon = 0.25$; (b) The performance with different ϵ, where $a = 8, b = -12, c = 6$; (c) The gap between the real optimal action value and the greedy action value; (4) The indicator function that shows whether the greedy action is the real optimal action.

For the non-monotonic game in Fig. 10-a, we set $a = 8$, $b = -12$. Suppose the dataset is collected by the ϵ-greedy exploration, where the greedy action is set to $\mathbf{u}^s = (u_1^2, u_2^2)$. Then we choose (1) three reward assignments, $c \in \{-3, 3, 6\}$, $\epsilon = 0.25$, as well as (2) three exploration rate assignments, $\epsilon \in \{0.5, 0.75, 1\}$, $c = 6$, and run WVDN and WQMIX with five random seeds on a fixed dataset. We show the gap between the approximated value of the real optimal action $\hat{Q}_1(u_1^*)$ and the greedy action value $\hat{Q}_1(\hat{u}_1^*)$ for agent 1, which should be zero if the algorithm converges to the optimal. Fig. 11-a and b show that the gap reduces with the increase of $\frac{1}{\alpha}$, indicating that the weight α should be small enough to deal with non-monotonicity. In addition, we can see that α should be small enough to converge to the optimal when the suboptimal value is large, or the exploration rate is low, which is consistent with our analysis in Section 3.

For the stochastic game in Fig. 10-b, we set $a = 8, b = -12, c = 20, d = -20, e = 6$, which indicates that this game also has non-monotonic target values. Suppose the dataset is collected by the ϵ-greedy exploration, where the greedy action is set to $\mathbf{u}^s = (u_1^2, u_2^2)$ and $\epsilon = 0.5$. In Fig. 11-c, we show the gap between $\hat{Q}_1(u_1^*)$ and $\hat{Q}_1(\hat{u}_1^*)$ and it is clear that WVDN and OW-QMIX cannot converge to the optimal.

Since the gap for CW-QMIX is close to zero, we also show whether the greedy action is optimal by an indicator function $\mathbb{I}(\hat{\mathrm{Q}}_1(\mathrm{u}_1^*) = \hat{\mathrm{Q}}_1(\hat{\mathrm{u}}_1^*))$. We empirically verify that CW-QMIX cannot converge to any policy by showing that the output of the indicator function keeps fluctuating between 0 and 1, which is consistent with our analysis in Section 3. CW-QMIX does not place α on the current greedy joint action, it can get rid of the stochastic suboptimal $\mathbf{u}^s$ when $\mathbf{u}^s$ is the current greedy action, i.e., $\hat{\mathbf{u}}^* = \mathbf{u}^s$. However, with the decrease of $Q_{tot}(\mathbf{u}^s)$, CW-QMIX falls into this suboptimal again once $\hat{\mathbf{u}}^* \neq \mathbf{u}^s$. As a result, although the gap between $\hat{Q}_1(u_1^*)$ and $\hat{Q}_1(\hat{u}_1^*)$ is tiny, CW-QMIX cannot converge to the optimal.

B.2. The Approximated Joint Action-Value Functions

Since we have discussed WQMIX in Section B.1, this section mainly analyzes the limitation of QPLEX and GVR.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**8**	-12	-12
u_2^2	-12	20 / -20	**6**
u_2^3	-12	**6**	**6**

(a) A game with non-monotonic and stochastic rewards.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**8**	6	6
u_2^2	6	6	6
u_2^3	6	6	6

(b) UTRAN.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	1.68	1.68	1.68
u_2^2	1.68	**16.8**	6.33
u_2^3	1.68	6.33	5.93

(c) OW-QMIX (α=0.1).

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	2.72	2.72	2.72
u_2^2	2.72	**5.97**	**5.97**
u_2^3	2.72	**5.97**	**5.97**

(d) CW-QMIX (α=0.1).

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	-8.29	-8.29	-8.29
u_2^2	-8.29	1.49	4.64
u_2^3	-8.29	5.00	**8.15**

(e) QMIX (α=1).

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	5.31	5.33	5.33
u_2^2	5.35	**6.03**	**6.03**
u_2^3	5.35	**6.03**	**6.03**

(f) GVR.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**8.09**	-12.32	-12.24
u_2^2	-12.32	-2.27	5.76
u_2^3	-12.24	5.76	5.62

(g) QTRAN.

$U_2 \backslash U_1$	u_1^1	u_1^2	u_1^3
u_2^1	**4.37**	-8.06	-8.45
u_2^2	-6.93	1.61	**4.37**
u_2^3	-8.09	**4.37**	**4.37**

(h) QPLEX.

Figure 12. The results on a matrix game with non-monotonic and stochastic rewards. Boldface means the optimal joint action from the payoff matrix, or the greedy joint action from the Q_{tot}.

We show the joint Q-value Q_{tot} returned from our baselines on a matrix game with non-monotonic and stochastic rewards in Fig. 12-a. We adopt a uniform exploration strategy to approximate the uniform data distribution (i.e., ϵ-greedy exploration with $\epsilon = 1$). As shown in Fig. 12, UTRAN and QTRAN can recover the optimal policy, while other methods fail to solve this simple task.

In QPLEX, the joint Q-value is formulated as:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = V_{tot}(\boldsymbol{\tau}) + A_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} V_i(\boldsymbol{\tau}) + \sum_{i=1}^{n} \lambda_i(\boldsymbol{\tau}, \mathbf{u}) A_i(\boldsymbol{\tau}, u_i)$$
$$= \sum_{i=1}^{n} [w_i(\boldsymbol{\tau}) V_i(\tau_i) + b_i(\boldsymbol{\tau})] + \sum_{i=1}^{n} \lambda_i(\boldsymbol{\tau}, \mathbf{u}) [Q_i(\boldsymbol{\tau}, u_i) - V_i(\boldsymbol{\tau})] \tag{35}$$

where $V_i(\tau_i) = \max_{u_i} Q_i(\tau_i, u_i), w_i(\boldsymbol{\tau}) \geq 0, \lambda_i(\boldsymbol{\tau}, \mathbf{u}) > 0$.

Let $\mathbf{u}^* = [u_1^*, ..., u_i^*, ..., u_n^*]$, $\mathbf{u}^s = [u_1^s, ..., u_i^s, ..., u_n^s]$ denote the optimal and one of the suboptimal joint action. If QPLEX is initialized to $\mathbf{u}^s$, then

$$V_{tot}(\boldsymbol{\tau}) = \sum_{i=1}^{n} Q_i(\tau_i, u_i^s). \tag{36}$$

It is difficult to jump out from this suboptimal for QPLEX. Consider $Q(s, \mathbf{u}^s)$ is the real second greatest action value at state s, where $u_i^s \neq u_i^*, \forall i \in A$. All targets which are smaller than $Q_{tot}(\boldsymbol{\tau}, \mathbf{u}^s)$ can be perfectly represented by QPLEX. However, when $(\boldsymbol{\tau}, \mathbf{u}^*)$ is sampled, QPLEX tries to minimizing the following loss:

$$(\sum_{i=1}^{n} \lambda_i(\boldsymbol{\tau}, \mathbf{u}^*)[Q_i(\boldsymbol{\tau}, u_i^*) - V_i(\boldsymbol{\tau})] - [Q(\boldsymbol{\tau}, \mathbf{u}^*) - V_{tot}(\boldsymbol{\tau})])^2, \tag{37}$$

where $Q(\boldsymbol{\tau}, \mathbf{u}^*) - V_{tot}(\boldsymbol{\tau}) > 0$, $Q_i(\boldsymbol{\tau}, u_i^*) - V_i(\boldsymbol{\tau}) \leq 0$, and $\lambda_i(\boldsymbol{\tau}, \mathbf{u}^*) > 0$. As a result, $\lambda_i(\boldsymbol{\tau}, \mathbf{u}^*)$ decreases to almost zero, and $Q_i(\boldsymbol{\tau}, u_i^*)$ gradually reaches to $V_i(\boldsymbol{\tau})$.

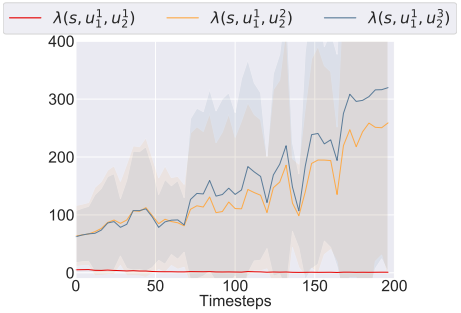

Figure 13. The weights for different joint actions in QPLEX.

The inequities of the weights hinder the update of the optimal Q-values. Namely, it is difficult for $Q_{tot}(\mathbf{u}^*)$ to exceed the suboptimal $V_i(\boldsymbol{\tau})$. In addition, based on the analysis of WQMIX, the weight $\lambda_i(\boldsymbol{\tau}, \mathbf{u}^*)$ should be greater than the weight for suboptimal to prioritize the potential optimal action, with QPLEX the opposite in the case.

To verify our analysis, we show the weights for different actions in QPLEX in Fig. 13. We can see that the weight for the real optimal action (u_1^1, u_2^1) is extremely small (almost 0), and the weights for suboptimal (u_1^1, u_2^2) and (u_1^1, u_2^3) are large (near 300). These imbalanced weights seriously undermine the learning of the real optimal. Consequently, we can see that $Q_{tot}(\mathbf{u}^*) = Q_{tot}(u_1^2, u_2^3) = Q_{tot}(u_1^3, u_2^2) = Q_{tot}(u_1^3, u_2^3) = 4.37$,

GVR uses inferior target shaping and superior experience replay to eliminate the non-optimal self-transition nodes. However, it ignores the stochasticity as WQMIX and thus behaves as CW-QMIX and cannot converge to any joint action. More comparisons among GVR, WQMIX, and UTRAN is provided in Appendix D.

B.3. Limitation: Multiple Optimal Solutions

In this section, we analyze the case that the Unique Optimal Solution defined in Assumption 1 does not hold.

Although many value-based algorithms fail to converge on a stable point when more than one optimal policy exists in the Markov Decision Process, multiple optimal is common in a multi-agent cooperative task. Consider a case that two agents arrive at a crossroads. The optimal policy is that one chooses to move, and another chooses to make way. Since these two agents are identical, two optimal joint policies exist in this simple case.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	**1**	0.5
u_2^2	0.5	**1**

(a) Payoff matrix with multiple optimal policies.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	0.75	0.75
u_2^2	**0.76**	**0.76**

(b) QMIX, $\epsilon = 1$.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	0.74	**0.75**
u_2^2	0.74	**0.75**

(c) VDN, $\epsilon = 1$.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	0.95	0.95
u_2^2	**0.97**	**0.97**

(d) UTRAN, $\epsilon = 1$.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	0.95	0.96
u_2^2	0.96	**1.02**

(e) UTRAN, $\epsilon = 0.75$.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	0.92	0.93
u_2^2	0.92	**1.02**

(f) UTRAN, $\epsilon = 0.5$.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	0.73	0.76
u_2^2	0.76	**1.01**

(g) UTRAN, $\epsilon = 0.25$.

$U_2 \backslash U_1$	u_1^1	u_1^2
u_2^1	0.61	0.65
u_2^2	0.66	**1.00**

(h) UTRAN, $\epsilon = 0.05$.

Figure 14. (a) The payoff matrix with multiple optimal. (b-h) The estimated joint Q-values Q_{tot} returned from QMIX, VDN, and UTRAN with different exploration rates. Boldface means the optimal joint action from the payoff matrix or the greedy joint action from the Q_{tot}.

We take an example of a simple one-step matrix game in Fig. 14-a with multiple optimal. When the full exploration scheme (i.e., ϵ-greedy with $\epsilon = 1$) is applied, QMIX and VDN cannot converge to the optimal. In addition, even if a value factorization method, which satisfies the Individual-Global-Max principle (Son et al., 2019):

$$\arg\max_{\mathbf{u}} Q_{tot}(s_t, \mathbf{u}) = (\arg\max_{u_1} Q_1(s_t, u_1), ..., \arg\max_{u_n} Q_n(s_t, u_n)), \tag{38}$$

can realize full representational capacity, it still cannot converge to the optimal. This is because the joint action formed by the actions sampled from two different optimal may not be the global optimal.

We can solve this problem by ϵ-greedy with decayed ϵ. In particular, we show the Q_{tot} returned from UTRAN to illustrate the effect of the ϵ. As shown in Figure 14-e to h, we can see the gap between the suboptimal and the optimal increases with the decrease of the ϵ, improving learning stability.

In addition, UTRAN inherits the characteristic of QMIX and thus is not a contraction, i.e., it would return two distinct Q_{tot} and converge to any one of the optimal.

C. Training Details for Best Per-agent Value Functions

Inspired by BAIL (Chen et al., 2020), $q_a(s, u_a; \theta_{q_a})$ is trained by minimizing the following loss:

$$L(\theta_{q_a}) = \sum_{t=0}^{T-1} K(s_t, \mathbf{u}_t)\delta(s_t, \mathbf{u}_t)^2 + \lambda\|\theta_{q_a}\|, \tag{39}$$

$$\text{where } K(s_t, \mathbf{u}_t) = \begin{cases} k & \delta(s_t, \mathbf{u}_t) > 0 \\ 1 & \text{otherwise} \end{cases}, \tag{40}$$

where $\delta(s_t, \mathbf{u}_t) = y_a(s_t, u_t^a) - q_a(s, u_t^a)$, $y_a(s_t, u_t^a) = \hat{r}(s_t, \mathbf{u}_t) + \gamma \max_{u^a} q_a(s_{t+1}, u^a)$, λ is the regularisation coefficients, and $k \gg 1$. Note that $\hat{r}(s_t, \mathbf{u}_t)$ is the expected reward return by the reward model. The best per-agent value function prioritizes the action whose target is greater than the current estimate.

D. WQMIX with different weights and GVR

In this section, we further show that WQMIX with fine-tuned weight α and GVR can solve non-monotonic target Q-value functions, but they are particularly brittle with stochasticity. We compare QTRAN, WQMIX with different weights and GVR on Predator-Prey and Predator-Stag-Hare with different levels of miscoordination penalties and stochasticity.

Fig. 15 demonstrates that WQMIX with a very low α and GVR can obtain the positive test return in all penalty settings on Predator-Prey. However, they cannot recover the optimal joint policy on stochastic Predator-Stag-Hare. This is because WQMIX overestimates the expected value of the suboptimal whose target is large with a low probability, which is consistent with our analysis in Section 3. GVR also ignores stochasticity and cannot accurately identify whether a given action is inferior. In contrast, UTRAN outperforms WQMIX and GVR in these tasks.

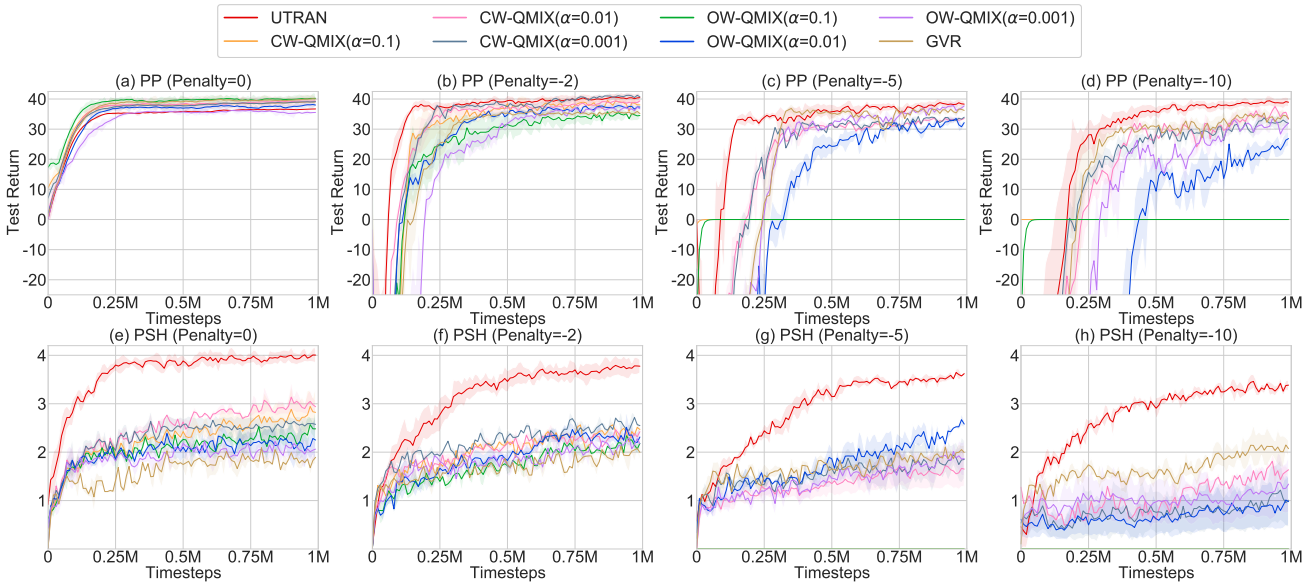

Figure 15. Performance comparisons among UTRAN, GVR, and WQMIX with different weights.

E. Effect of α on Performance

Fig. 16 shows the effect of varying α in the indicator function. We can see if α is too low, performance degrades considerably because UTRAN degenerates to QMIX (the indicator function mistakes all targets for stochastic samples, and thus no target is transformed). In contrast, UTRAN replaces all targets with the best per-agent value if α is too high. Its poor performance in Fig. 16-a and c suggest difficulties in solving the stochastic state-action pairs.

Therefore, α is a task-specific hyperparameter related to the reward and the state scale. To lighten the burden of tuning, our recommendation is to normalize the state, observation, and reward when we create an environment.

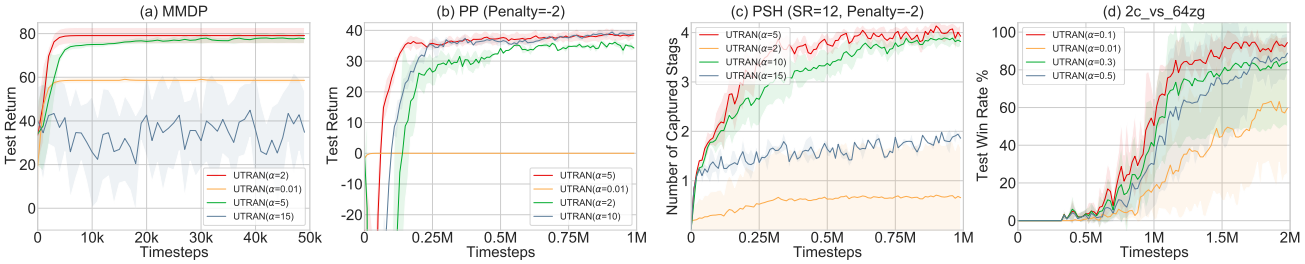

Figure 16. The results of UTRAN with different α on MMDP, PP, PSH, and SMAC.

F. More Results on SMAC

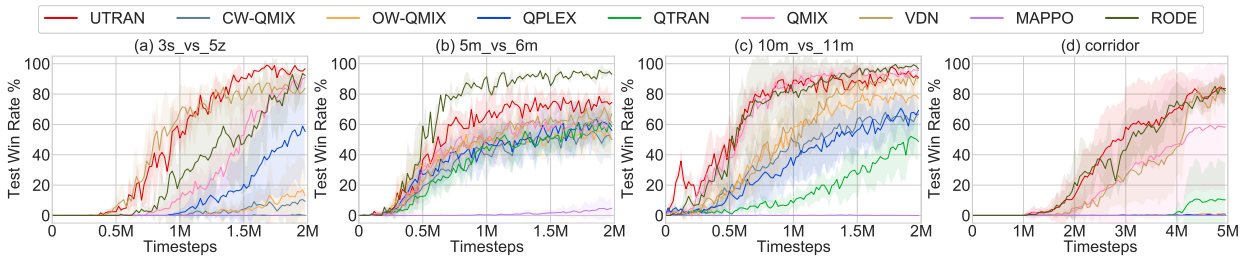

Figure 17. More results on the StarCraft Multi-Agent Challenge benchmark, where UTRAN achieves comparable performance with baseline methods. These results indicate that many tasks in SMAC may not have problems stemming from non-monotonic and stochastic targets. However, UTRAN can still learn quickly and steadily, demonstrating its adaptability to more general tasks.

G. Experimental Setup

G.1. Environments

Predator-Prey (PP) is a partially observable environment containing eight predators (agents) and eight prey in a 10×10 grid world. Each agent observes a 5×5 sub-grid around it and can perform five actions, i.e., up, down, left, right, and catch. The team receives a miscoordination penalty when two agents surround a prey, and only one tries to catch it. In contrast, they earn a bonus of 10 if they catch the prey simultaneously. After a successful catch, the catching predators and prey will be removed from the grid.

StarCraft Multi-Agent Challenge (SMAC) is a partially observable reinforcement learning benchmark. An individual agent with parameter sharing controls each allied unit, and a hand-coded built-in StarCraft II AI controls enemy units. The difficulty of the game AI is set to the "very difficult" level. At each time step, agents have access to their local observations within the field of view. The feature vector contains attributes of both allied and enemy units: `distance`, `relative x`, `relative y`, `health`, `shield`, and `unit_type`. In addition, agents can observe the last actions of allied units and the terrain features surrounding them. The global state vector includes the coordinates of all agents relative to the centre of the map and other features present in the local observation of agents. In addition, the state stores the energy of Medivacs, the cooldown of the rest of the allied units and the last actions of all agents. Note that the global state information is only available to agents during centralized training. All features in state and local observations are normalized by their maximum values. After receiving the observations, each agent is allowed to take action from a discrete set which consists of `move[direction]`, `attack[enemy_id]`, `stop` and `no-op`. Move direction includes north, south, east, and west. Note that the dead agents can only take `no-op` action while live agents cannot. For health units, Medivacs use `heal[agent_id]` actions instead of `attack[enemy_id]`. Depending on different scenarios, the maximum number of actions varies between 7 and 70. Note that agents can only perform the `attack[enemy_id]` action when the enemy is

within its shooting range. At each time step, agents take joint action and receive a positive global reward based on the total damage dealt to the enemy units. In addition, they can receive an extra reward of 10 points after killing each enemy unit and 200 points after killing all enemy units. The rewards are scaled to around 20, so the maximum cumulative reward is achievable in each scenario.

Not that we use the version of `SC2.4.6.2.69232` rather than `SC2.4.10` for SMAC, which is the same as our baselines expect MAPPO. Performance is not always comparable between versions.

G.2. Network Structures and Hyperparameters

We adopt the same architecture for VDN, QMIX, QTRAN, QPLEX, RODE and WQMIX as (Samvelyan et al., 2019; Rashid et al., 2020; Wang et al., 2020b). Each agent independently learns a policy with fully shared parameters between all policies. We used RMSProp with a learning rate of 5×10^{-4} and $\gamma = 0.99$, buffer size 5000, mini-batch size 32 for all algorithms. We use an ϵ-greedy strategy in which ϵ decreases from 1 to 0.05 over 1M timesteps for `3s5z_vs_3s6z`, `6h_vs_8z` and `corridor`, and over 50k timesteps for other tasks. The dimension of each agent's GRU hidden state is set to 64.

For WQMIX, we use $\alpha = 0.1$ on Predator-Prey and Predator-Stag-Hare (unless specified otherwise), and $\alpha = 0.5$ in SMAC.

The implementation of MAPPO is consistent with their official repositories. As shown in Tab. 1, all hyper-parameters are left unchanged at the origin best-performing status.

Hyperparameter	Value	Hyper-parameter	Value
critic lr	5e-4	actor lr	5e-4
ppo epoch	5	ppo-clip	0.2
optimizer	Adam	batch size	3200
optim eps	1e-5	hidden layer	1
gain	0.01	training threads	32
rollout threads	8	γ	0.99
hidden layer dim	64	activation	ReLU

Table 1. Hyper-parameters in MAPPO.

For UTRAN, we use the same agent and mixing network in QMIX as (Rashid et al., 2018). For the best Q-value function, we use feed-forward networks with three hidden layers of dim $\{64, 64\}$ and ReLU non-linearities, where the optimizer is RMSProp with a learning rate of 5×10^{-4}. The regularization coefficient λ is set to 1×10^{-3}, and the weight k is set to 1×10^{3}. For the reward and the state predictors, we use a multilayer perceptron (MLP) with ReLU non-linearities and a gated recurrent unit (GRU) to extract features, where the hidden dim is 128. Then we use another MLP to output the predicted reward, the standard deviation and the embedding of the next state. The first and the GRU parameters are shared. The dimension of the target embedding for the state m is set to 16. We use Adam with a learning rate of 1×10^{-3} to train the predictors.

We conduct experiments on an NVIDIA RTX 3090 GPU. Each task needs to train for about 5 hours on Predator-Prey, and about 8 to 16 hours on SMAC, depending on the number of agents and episode length limit of each map. We evaluate 32 episodes with decentralized greedy action selection every $10k$ timesteps for each algorithm.