ReMIX: Regret Minimization for Monotonic Value Function Factorization in Multi-Agent Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Value function factorization methods have become a dominant approach for cooperative multi-agent reinforcement learning under a centralized training and decentralized execution paradigm. By factorizing the optimal joint action-value function using a monotonic mixing function of agents' utilities, these algorithms ensure the consistency between joint and local action selections for decentralized decision-making. Nevertheless, the use of monotonic mixing functions also induces representational limitations. Finding the optimal projection of an unrestricted mixing function onto monotonic function classes is still an open problem. To this end, we propose ReMIX, formulating this optimal projection problem for value function factorization as a regret minimization over the projection weights of different state-action values. Such an optimization problem can be relaxed and solved using the Lagrangian multiplier method to obtain the close-form optimal projection weights. By minimizing the resulting policy regret, we can narrow the gap between the optimal and the restricted monotonic mixing functions, thus obtaining an improved monotonic value function factorization. Our experimental results on Predator-Prey and StarCraft Multi-Agent Challenge environments demonstrate the effectiveness of our method, indicating the better capabilities of handling environments with non-monotonic value functions.

1 Introduction

Reinforcement learning has demonstrated great potential in solving challenging real-world problems, from autonomous driving (Cao et al., 2012; Hu et al., 2019) to robotics and planning (Matignon et al., 2012; Levine et al., 2016; Hüttenrauch et al., 2017). In many scenarios, these tasks involve multiple agents within the same environment and thus require multi-agent reinforcement learning (MARL) (Vinyals et al., 2019; Jaques et al., 2019; Baker et al., 2019; Wang et al., 2020b) to coordinate agents and learn desired behaviors from their experiences. Due to practical communication constraints and the need to cope with vast joint action space, MARL algorithms often leverage fully decentralized policies but learn them in a centralized fashion with access to additional information during training. Value function factorization methods, e.g., QMIX (Rashid et al., 2018), QPLEX (Wang et al., 2020a), Qatten (Yang et al., 2020), FOP (Zhang et al., 2021), and DOP (Wang et al., 2020c), have been a dominant approach for such centralized training and decentralized execution (CTDE) MARL (Kraemer & Banerjee, 2016). By factorizing the optimal joint action value function using a monotonic mixing function of per-agent utilities, these algorithms ensure the consistency between joint and local action selections for decentralized decision-making. Superior performance has been reported in many MARL tasks, such as the StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al., 2019).

It is known that value function factorization can be viewed as an operator (Dugas et al., 2009), which first computes the optimal joint action value functions as targets and then projects them onto the space representable by monotonic function classes. The projected monotonic mixing functions enable efficient maximization yet allow decentralized decision-making. However, it also poses representational limitations. For instance, QMIX leverages a universal approximator for non-linear monotonic mixing functions. It prevents QMIX from efficiently representing joint action value functions where agents' orderings of their action choices depend on each other (Mahajan et al., 2019). Later, the authors in the paper (Rashid et al., 2020)

proposed an improved projection using Weighted QMIX (WQMIX). It assigns higher weights to the values of optimal joint actions than the suboptimal ones, resulting in a better projection that more accurately represents these optimal values. However, WQMIX relies purely on a heuristic design – such as Centrally-Weighted (CW) and Optimistically-Weighted (OW) – where such weight term is a constant. Finding an optimal projection onto the monotonic function class is still an open problem.

To this end, we propose ReMIX, formulating the optimal projection problem for value function factorization as a regret minimization over the projection weights of different state-action values. Specifically, we construct an optimal policy following the optimal joint action-value function and a restricted policy using its projection onto monotonic mixing functions. A policy regret is then defined as the difference between the expected discounted reward of the optimal policy and that of the restricted policy. By minimizing such policy regret through an upper bound, we can narrow the gap between the optimal and restricted policies and thus force the projected monotonic value function to approach the optimal one during learning, leading to an optimal monotonic factorization with minimum regret. We note that while policy regret minimization has been employed to formulate various optimizations in reinforcement learning, such as optimal prioritized experience replay (Liu et al., 2021) and loss function design (Jin et al., 2018), to the best of our knowledge, this is the first proposal for optimizing value function factorization in MARL through policy regret minimization.

We show that the proposed regret minimization can be solved via the Lagrangian method (Bertsekas, 2014) considering an upper bound. By examining a weighted Bellman equation involving monotonic mixing functions and per-agent critics, we leverage the implicit function theorem (Krantz & Parks, 2002) and derive Karush–Kuhn–Tucker (KKT) (Ghojogh et al., 2021) conditions to find the optimal projection weights in closed form. Our results highlight the key principles contributing to optimal monotonic value function factorization. The optimal projection weights can be interpreted to consist of four components: Bellman error, value underestimates, the gradient of the monotonic mixing function, and the on-policiness of available transitions. We note that the first two terms relating to Bellman error and value underestimates are consistent with the weighting heuristics proposed in WQMIX, thus providing a quantitative justification and recovering WQMIX as a special case. More importantly, our analysis reveals that an optimal value function factorization should also depend on the gradient of the monotonic mixing function and the positive impact of more current transitions.

Following the theoretical results, we provide a tractable approximation of the optimal projection weights and propose a MARL algorithm of ReMIX with regret-minimizing monotonic value function factorization. We validate the effectiveness of ReMIX in Predator-Prey (Böhmer et al., 2020) and SMAC. Compared with state-of-the-art factorization-based MARL algorithms (e.g., WQMIX, QPlex, FOP, DOP), ReMIX is shown to better cope with environments with non-monotonic value functions, resulting in improved convergence and superior empirical performance.

The main contributions of our work are as follows:

- We propose a novel method, ReMIX, formulating the optimal value function factorization as a policy regret minimization and solving the weights of the optimal projection in closed form.
- The theoretical results and tractable weight approximations of ReMIX enable cooperative MARL algorithms with improved value function factorization.
- Experiment results of ReMIX in Predator-Prey and SMAC environments demonstrate superior convergence and empirical performance over state-of-the-art factorization-based methods. We further perform ablation studies to demonstrate the contribution of each component in our design.

2 Background

2.1 Partially Observable Markov Decision Process

We describe a fully cooperative multi-agent sequential decision-making task as a decentralized partially observable Markov decision process (Dec-POMDP) (Oliehoek & Amato, 2016) consisting of a tuple $G = \langle S, U, P, R, Z, O, n, \gamma \rangle$, where $s \in S$ describes the global state of the environment. At each time step, each

agent $a \in A \equiv \{1, \ldots, n\}$ selects an action $u_a \in U$, and all selected actions are combined to form a joint action $\mathbf{u} \in \mathbf{U} \equiv U_a$. This process leads to a transition in the environment based on the state transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$. All agents share the same reward function $r(s, \mathbf{u}) : S \times \mathbf{U} \to \mathbb{R}$ with a discount factor $\gamma \in [0, 1)$.

In the partially observable environment, the agents' individual observations $z \in Z$ are generated by the observation function $O(s, u) : S \times A \to Z$. Each agent has an action-observation history $\tau_a \in T \equiv (Z \times U)^*$. Conditioning on the history, the policy becomes $\pi^a(u_a|\tau_a) : T \times U \to [0, 1]$. The joint policy $\boldsymbol{\pi}$ has a joint action-value function: $Q^{\boldsymbol{\pi}}(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}}[R_t|s_t, \mathbf{u}_t]$, where t is the timestep and $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return. In this paper, we adopt the centralized training and decentralized execution paradigm: the learning algorithm has access to all local action-observation histories $\boldsymbol{\tau}$ and global state s during training, yet every agent can only access its individual history in execution. Although we compute individual policy based on histories in practice, following the existing work (Su & Lu, 2022) where the state solely consists of state and history, we will use $\boldsymbol{\pi}(\mathbf{u}|s)$ and $\pi^a(u_a|s)$ in analysis and proofs for simplicity.

2.2 Value Function Decomposition

In MARL, value function decomposition methods (Sunehag et al., 2017; Rashid et al., 2018; Son et al., 2019; Wang et al., 2020c) learn a joint action value functions $Q_{tot}(s, \mathbf{u})$ as a function of combined individual action value functions, conditioning individual local observation history, then these local action values are combined with a learnable mixing neural network to produce joint action values, given by:

$$Q_{tot}(s, \mathbf{u}) = \text{Mixer}\left(s, Q^a(\tau_a, u_a)\right) = f_s\left(Q^a(\tau_a, u_a)\right).$$

Under the principle of guaranteed consistency between global optimal joint actions and local optimal actions, a global argmax performed on Q_{tot} yields the same result as a set of individual arg max operations performed on each local value, also known as Individual Global Maximum (IGM):

$$\arg\max_{\mathbf{u}} Q_{tot} = \left(\arg\max_{u_1} Q^1, \cdots, \arg\max_{u_n} Q^n\right).$$

VDN (Sunehag et al., 2017) takes the joint action value function as a summation of local action value:

$$Q_{tot}(s, \mathbf{u}) = \sum_{i=1}^{n} Q^i(\tau_i, u_i),$$

while QMIX (Rashid et al., 2018) proposed a more general case of VDN by approximating a broader class of monotonic functions to represent joint action value functions rather than summation of the local action values, such that:

$$\frac{\partial Q_{tot}(s, \mathbf{u})}{\partial Q^a\left(\tau_a, u_a\right)} > 0, \forall a \in A \equiv 1, \ldots, n, \tag{1}$$

which restricts the joint action value function to be a monotonic mixing of agents' utilities, preventing it from projecting non-monotonic joint action representation. Later proposed WQMIX (Rashid et al., 2020) solved the limitation by introducing the weights into the projection to retrieve the optimal policy. The WQMIX algorithms – OW and CW QMIXs – can place more importance on the better Q_{tot} in minimizing the loss: $\sum_{i=1}^{b} w(s, \mathbf{u})(Q_{tot}(s, \mathbf{u}; \theta) - \bar{y}_i)^2$, where $\bar{y}_i = r + \gamma \hat{Q}^*(s', \arg\max_{\mathbf{u}'} Q_{tot}(s', \mathbf{u}'; \theta^-))$ is the fixed target, $\hat{Q}^*$ is the unrestricted joint action-value function, b is the batch size and w is the weighting function. For example, in OW, the weight w is given by:

$$w(s, \mathbf{u}) = \begin{cases} 1 & Q_{tot}(s, \mathbf{u}) < \bar{y}_i \\ \alpha & \text{otherwise.} \end{cases} \tag{2}$$

When a transition is overestimated in the OW paradigm, it will be assigned with a constant weight $\alpha \in (0, 1]$. Compared to OW, CW has a similar mechanism but assigns weights to a transition whose joint action $\mathbf{u}$ is not the best. We note that while insightful, these methods are based on heuristic designs of projection weights. Finding optimal projection weights for monotonic value function factorization is still an open problem. In this paper, we reformulate the problem as a policy regret minimization and solve the optimal projection weights in closed form by relaxing the objective and the Lagrangian method.

2.3 Policy Regret

The object of MARL is to find a joint policy $\boldsymbol{\pi}$ that can maximize the expected return: $\eta(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}}[\sum_{i=0}^{\infty} \gamma^i r_{t+i}]$. For a fixed policy, the Markov decision process becomes a Markov reward process, where the discounted state distribution is defined as $d^{\pi}(s)$. Similarly, the discounted state-action distribution is defined as $d^{\boldsymbol{\pi}}(s, \mathbf{u}) = d^{\boldsymbol{\pi}}(s)\boldsymbol{\pi}(\mathbf{u}|s)$. Thus, we will have the expected return rewritten as $\eta(\boldsymbol{\pi}) = \frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}}(s,\mathbf{u})}[r(s, \mathbf{u})]$.

We assume there exists an optimal joint policy $\boldsymbol{\pi}^*$ such that $\boldsymbol{\pi}^* = \arg\max_{\boldsymbol{\pi}} \eta(\boldsymbol{\pi})$. The regret of the joint policy $\boldsymbol{\pi}$ is defined as $\text{regret}(\boldsymbol{\pi}) = \eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi})$. The policy regret measures the expected loss when following the current policy $\boldsymbol{\pi}$ instead of optimal policy $\boldsymbol{\pi}^*$. Since $\eta(\boldsymbol{\pi}^*)$ is a constant, minimizing the regret is consistent with maximizing of expected return $\eta(\boldsymbol{\pi})$. In this paper, we use regret as an alternative optimization objective for finding the optimal projection in MARL, along with multiple constraints, e.g., the Bellman equation and the sum of projection weights. By minimizing the regret, the current policy $\boldsymbol{\pi}_k$ following a monotonic value factorization will approach the optimum $\boldsymbol{\pi}^*$ following an unrestricted value function.

3 Related Work

3.1 Multi-Agent Reinforcement Learning

MARL algorithms have developed into neural-network-based methods that can cope with high-dimensional state and action spaces. Early methods practice finding policies for a multi-agent system by directly learning decentralized value functions or policies. For example, independent Q-learning (Tan, 1993) trains independent action-value functions for each agent via Q-learning. (Tampuu et al., 2017) extends this technique to DQN (Mnih et al., 2015). Recently, approaches for CTDE have come up as centralized learning of joint actions that can conveniently solve coordination problems without introducing non-stationary. COMA (Foerster et al., 2018) uses a centralized critic to train decentralized actors to estimate a counterfactual advantage function for every agent. Similar works (Gupta et al., 2017; Lowe et al., 2017) are also proposed based on such analysis. Under the CTDE manner, value decomposition methods, such as QMIX (Rashid et al., 2018), perform well in solving cooperative problems. Besides, other mechanisms can also solve competitive problems or mixed problems. For instance, MADDPG (Lowe et al., 2017) utilizes the ensemble of policies for each agent that leads to more robust multi-agent policies, showing strength in cooperative and competitive scenarios, and the extension (Iqbal & Sha, 2019) of MADDPG has been proposed to realize further optimization towards the original algorithm. This paper focuses on the cooperative setting in MARL and aims to find the optimal weighting scheme to retrieve the best projection onto the monotonic function class.

3.2 Value Decomposition Approaches

Value decomposition approaches (Guestrin et al., 2002; Castellini et al., 2019) are widely used in value-based MARL. Such methods integrate each agent's local action-value functions through a learnable mixing function to generate global action values. For instance, VDN (Sunehag et al., 2017) and QMIX estimate the optimal joint action-value function Q^* as Q_{tot} with different formations. VDN aims to learn a joint action-value function Q_{tot} of the sum of individual utilities for each agent. QMIX calculates Q_{tot} by combining mentioned utilities via a continuous state-dependent monotonic function, generated by a feed-forward mixing network with non-negative weights. QTRAN (Son et al., 2019) and QPLEX further extend the class of value functions that can be represented. Besides value-based factorization algorithms, some works extend the value decomposition method to policy-based actor-critic algorithms. In VDAC (Su et al., 2021), a factorized actor-critic framework compatible with A2C can obtain a reasonable trade-off between training efficiency and algorithm performance. Recently proposed FOP (Zhang et al., 2021) provides a new way to factorize the optimal joint policy induced by maximum-entropy MARL into individual policies. DOP (Wang et al., 2020c) addresses the issue of centralized-decentralized mismatch and credit assignment in both discrete and continuous action spaces in the multi-agent actor-critic framework. In this paper, we recast the problem of projecting an unrestricted value function onto monotonic function classes as a policy regret minimization, whose solution allows us to find the optimal projection weights to obtain an improved value function factorization.

4 Optimal Projection onto Monotonic Value Functions

4.1 Problem Formulation as Regret Minimization

Let Q^* be the unrestricted joint action value function and $Q_{tot} = f_s(Q^1(\tau_1, u_1), \ldots, Q^n(\tau_n, u_n))$ be its estimation obtained through a monotonic mixing function $f_s(\cdot)$ of per-agent utilities $Q^a(\tau_i, u_i)$ for $a = 1, \ldots, n$. For simplicity of notations, we use Q_k to denote Q_{tot} at step k. Adopting $\mathcal{B}^* Q^*_{k-1}$ as the target with a Bellman operator $\mathcal{B}^*$, we update Q_k in tandem using a weighted Bellman equation: $Q_k = \arg\min_{Q \in \mathcal{Q}} \mathbb{E}_\mu[w_k(s, \mathbf{u})(Q - \mathcal{B}^* Q^*_{k-1})^2(s, \mathbf{u})]$, where $w_k(s, \mathbf{u})$ are non-negative projection weights for different transitions that need to be optimized. This projects the unrestricted value function onto a monotonic function class $Q \in \mathcal{Q}$.

To formulate the policy regret with respect to this projection, we consider a Boltzmann policy $\boldsymbol{\pi}_k$ following the agent's individual utilities Q^a_k obtained from such monotonic value factorization, i.e., $\boldsymbol{\pi}_k = [\pi^1_k, ..., \pi^n_k]^{\mathrm{T}}$ and $\pi^a_k = e^{Q^a_k(\tau_a, u_a)} / [\sum_{\tau_a, u'_a} e^{Q^a_k(\tau_a, u'_a)}]$, as well as a similar policy $\boldsymbol{\pi}^*$ following the unrestricted value function Q^* that is defined over joint actions in the Boltzmann manner. Our objective is to minimize the policy regret $\eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi})$ over non-negative projection weights under relevant constraints, i.e.,

$$
\begin{aligned}
\min_{w_k} \quad & \eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \\
\text{s.t.} \quad & Q_k = \arg\min_{Q \in \mathcal{Q}} \mathbb{E}_\mu[w_k(s, \mathbf{u})(Q - \mathcal{B}^* Q^*_{k-1})^2(s, \mathbf{u})], \\
& \mathbb{E}_\mu[w_k(s, \mathbf{u})] = 1, \quad w_k(s, \mathbf{u}) \geq 0, \\
& Q_k(s, \mathbf{u}) = f_s(Q^1(\tau_1, u_1), \ldots, Q^n(\tau_n, u_n)),
\end{aligned} \tag{3}
$$

where $\boldsymbol{\pi}^*$ and $\boldsymbol{\pi}_k$ are policies following the unrestricted and monotonic value functions, respectively. The projection weights must sum up to 1, and μ is the uniform distribution that we sample data from the replay buffer. An additional table to summarize and explain the given notations is provided in Appendix A.

4.2 Solving Optimal Projection Weights

The solution to this optimization problem relies on the monotonic function $f_s(\cdot)$ represented by a mixing network, which takes the state and agent networks' output Q^a_k as inputs and generates an estimate of joint value function Q_{tot}. Solving the regret minimization problem through the Lagrangian method requires analyzing the KKT conditions. Thus, we first find the first-order derivative of the monotonic mixing network, which will also be leveraged to find an optimal solution. The mixing network is a universal approximator consisting of a two-layer network of non-negative weight (Dugas et al., 2009). We compute its first-order derivative in the following lemma.

Lemma 1. *Considering a two-layer mixing network of the weight matrix W_1, W_2, bias b_1, b_2 and activation function $h(\cdot)$, the derivative of Q_{tot} over one of the local utilities Q^a is:*

$$
f'_{s,Q^a} = \frac{\partial Q_{tot}}{\partial Q^a} = h'_{Q^a}(\vec{Q}^{\mathrm{T}} W_1 + b_1) \sum_{j=1}^{m} w^1_{aj} w^2_j,
$$

where $\vec{Q} = [Q^1, \ldots, Q^n]^{\mathrm{T}}$. W_1, W_2 are the $n \times m$ and $1 \times m$ matrix correspondingly, with the respective elements w^1_{ij} and w^2_j in each matrix. n is the agent number, and m is the width of the mixing network.

Proof. See Appendix B. □

Given that the monotonic mixing function is smooth and differentiable, we consider an upper bound of the regret objective (obtained using a relaxation and Jensen's inequality) and formulate its Lagrangian by introducing Lagrangian multipliers with respect to the constraints. It allows us to solve the proposed regret-minimization problem and obtain optimal projection weights in closed form (albeit with a normalization factor Z^*).

Theorem 1 (Optimal weighting scheme)**.** *Under mild conditions, the optimal weight $w_k(s, \mathbf{u})$ to a relaxation of the regret minimization problem in equation 3 with discrete action space is given by:*

$$w_k(s, \mathbf{u}) = \frac{1}{Z^*}(E_k(s, \mathbf{u}) + \epsilon_k(s, \mathbf{u})), \tag{4}$$

where when $Q_k \leq \mathcal{B}^* Q^*_{k-1}$, we have

$$E_k(s, \mathbf{u}) = \frac{d^{\boldsymbol{\pi}_k}(s, \mathbf{u})}{\mu(s, \mathbf{u})}(\mathcal{B}^* Q^*_{k-1} - Q_k)\exp(Q^*_{k-1} - Q_k)\left(\sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} - 1\right),$$

and otherwise (i.e., when $Q_k > \mathcal{B}^* Q^*_{k-1}$), we have

$$E_k(s, \mathbf{u}) = 0,$$

where Z^* is the normalization factor, and $\epsilon_k(s, \mathbf{u}))$ is a negligible term when the probability of reversing back to the visited state is small, or the number of steps agents take to revisit a previous state is large.

Proof. We give a sketch of the proof below and provide the complete proof in Appendix C. The derivation of optimal weights consists of the following major steps: (i) Use a relaxation and Jensen's inequality to obtain a more tractable upper bound of the regret objective for minimization. (ii) Formulate the Lagrangian for the new optimization problem and analyze its KKT conditions. (iii) Compute various terms in the KKT condition and, in particular, analyze the gradient of Q_k with respect to weights p_k (defined through the weighted Bellman equation) by leveraging the implicit function theorem (IFT). (iv) Derive the optimal projection weights in closed form by setting the Lagrangian gradient to zero and applying KKT and its slackness conditions.

Step 1: Relaxing the objective and adopting Jensen's inequality. To begin with, we replace the original optimization objective function, the policy regret, with a relaxed upper bound. This replacement can be achieved through the following inequality:

$$\eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \leq \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[(Q^*_{k-1} - Q_k)(s, \mathbf{u}^*)] + \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[(Q_k - Q^*_{k-1})(s, \mathbf{u})]. \tag{5}$$

The proof of this result is given in the appendix. The key idea is to rewrite the regret using the expectation of the action-value functions with respect to discounted distribution $d^{\boldsymbol{\pi}_k}$. After that, we adopt Jensen's inequality (McShane, 1937) to continue relaxing the intermediate objective function based on a convex function $g(x) = \exp(-x)$. Thus, a new optimization objective generated from equation 5 becomes:

$$\min_{w_k} \quad -\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s, \mathbf{u}^*)] - \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[\exp(Q^*_{k-1} - Q_k)(s, \mathbf{u})], \tag{6}$$

where the constraints still hold for the new optimization objective.

Step 2: Computing the Lagrangian. In this step, we leverage the Lagrangian multiplier method to solve the new optimization problem in equation 6. For simplicity, we use p_k that absorbs the data distribution μ into w_k. The constructed Lagrangian is:

$$\mathcal{L}(p_k; \lambda, \nu) = -\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s, \mathbf{u}^*)] - \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[\exp(Q^*_{k-1} - Q_k)(s, \mathbf{u})] + \lambda(\sum_{s,\mathbf{u}} p_k - 1) - \nu^{\mathrm{T}} p_k,$$

where p_k is the weight w_k multiplied by the data distribution μ, and λ, ν are the Lagrange multipliers.

Step 3: Computing the Gradients Required in the Lagrangian. According to the first constraint in equation 3, the gradient $\frac{\partial Q_k}{\partial p_k}$ can be computed via IFT given by:

$$\frac{\partial Q_k}{\partial p_k} = -[\mathrm{diag}(p_k)]^{-1}[\mathrm{diag}(Q_k - \mathcal{B}^* Q^*_{k-1})].$$

We also derive the gradient $\frac{\partial d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}{\partial p_k}$ for solving the Lagrangian. The derivation details are given in the appendix.

Step 4: Deriving the Optimal Weight. After having the equation for two gradients and an expression of the Lagrangian, we can compute the optimal p_k via an application of the KKT conditions, which needs to set the partial derivative of the Lagrangian equaling to zero, as $\frac{\partial \mathcal{L}(p_k;\lambda,\nu)}{\partial p_k} = 0$, where the optimal weight w_k can be acquired from the p_k.

□

The theoretical results shed light on the key factors determining an optimal projection onto monotonic mixing functions. Specifically, the optimal projection weights consist of four components relating to Bellman error, value underestimation, the gradient of the monotonic mixing function, and the on-policiness of available transitions. We will interpret these four components next and develop a deep MARL algorithm through approximations of the optimal projection weights.

Bellman error $\mathcal{B}^*Q^*_{k-1} - Q_k$: Q_k is the estimation of the action-value function after the Bellman update. This term measures the distance between the estimation and the Bellman target. A large difference in this term means higher hindsight Bellman error. Due to the KKT slackness condition, our analysis indicates that the optimal projection weight is zero when $Q_k > \mathcal{B}^*Q^*_{k-1}$ is an overestimate of the target value, and otherwise, a higher weight should be assigned when Q_k is more underestimated.

Value underestimation $\exp(Q^*_{k-1} - Q_k)$: If Q_{tot} after the Bellman update at current step k is smaller than optimal Q^*_{k-1}, it results in an underestimate. In this case, we will assign a higher weight (always larger than 1) to this transition, which is proportional to the exponential of this underestimation gap. In contrast, when overestimating (with a negative gap), the assigned weight becomes lower and always smaller than 1. This is important because an underestimate of function approximation may lead to a sub-optimal Q_k estimation and thus non-optimal action selections.

Gradient of the mixing network $\sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} - 1$: It turns out that the optimal projection weights also depend on the inverse of the gradient of the monotonic mixing function $f_s(\cdot)$, which is a new result. Intuitively, the optimal projection weights would become higher when the monotonic mixing function is insensitive to underlying per-agent utility values (i.e., having a small, positive gradient). We view this result as a form of normalization with respect to different shapes of monotonic mixing function $f_s(\cdot)$. In practical algorithms, we often use the two-layer mixing network with non-negative weights to approximate the monotonic function $f_s(\cdot)$ to produce Q_k. The parameters of the mixing network are updated every step, and the gradient value can be readily computed from these parameters. We have provided an instance regarding calculating the gradient of a two-layer mixing network in Lemma 1. It is worth noting that similar gradients can also be obtained for other value function factorization methods.

Measurement of on-policy transitions $\frac{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}{\mu(s,\mathbf{u})}$: The efficient update of the joint action value function can be achieved by focusing on transitions that are more possibly to be visited by the current policy, i.e., with a higher $d^{\boldsymbol{\pi}_k}(s,\mathbf{u})$. Adding this term can speed up the search for the optimal Q_k close to Q^*_{k-1}.

4.3 Proposed Algorithm

Our analytical results in Theorem 1 identify four key factors determining the optimal projection weights. Interestingly, the first two terms, relating to Bellman error and value underestimation, recover the heuristic designs in WQMIX. Specifically, when the Bellman error of a particular transition is high, which indicates a wide gap between Q_k and Q^*_{k-1}, we may consider assigning a larger weight to this transition. Similarly, value underestimation works as a correction term for incoming transitions: based on the difference of current Q_k and ideal Q^*_{k-1}, it will compensate the underestimated Q_k with larger importance while penalizing overestimated Q_k with a smaller weighting modifier, consistent with OW scheme in equation 2.

Additionally, our analysis identifies two new terms: the gradient of the monotonic mixing function and measurement of on-policy transitions, which are crucial in obtaining an optimal projection onto monotonic value function factorization. As discussed, we interpret the gradient term in optimal weights as a form

Algorithm 1 ReMIX

Initialize step, the parameters of mixing network, agent networks, and hyper-network.
Set the learning rate α and replay buffer $\mathcal{D}$
let $\theta^- = \theta$
for step $= 1 :$ step$_{max}$ **do**
  $k = 0, s_0 =$ initial state
  while $s_k \neq$ terminal and $k <$ episode limit **do**
    for each agent a **do**
      $\tau_k^a = \tau_{k-1}^a \cup (o_k, u_{k-1})$
      $u_k^a = \begin{cases} \arg\max_{u_k^a} Q(\tau_k^a, u_k^a) & \text{with probability } 1-\epsilon \\ \text{randint}(1, |U|) & \text{with probability } \epsilon \end{cases}$
    end for
    Obtain the reward r_k and next state s_{k+1}
    Store the current trajectory into replay buffer $\mathcal{D} = \mathcal{D} \cup (s_k, \mathbf{u}_k, r_k, s_{k+1})$
    $k = k+1, \text{step} = \text{step} + 1$
  end while
  Collect b samples from the replay buffer $\mathcal{D}$ following uniform distribution μ.
  for each timestep k in each episode in batch b **do**
    Evaluate Q_k, Q^* and target values
    Obtain the utilities Q_a from agents' local networks, and compute the individual policy π_k^a
    Compute the weight:
    $w_k \propto \begin{cases} (\mathcal{B}^* Q_{k-1}^* - Q_k)\exp(Q_{k-1}^* - Q_k)\left(\sum_{j=1}^n \frac{1-\pi^j}{f'_{s,Q^j}} - 1\right) & \text{when } Q_k \leq \mathcal{B}^* Q_{k-1}^* \\ \epsilon & \text{when } Q_k > \mathcal{B}^* Q_{k-1}^* \end{cases}$
  end for
  Minimize the Bellman error for Q_k weighted by w_k, update the network parameter θ:
  $\theta = \theta - \alpha(\nabla_\theta \frac{1}{b}\sum_i^b w_k(Q_k - y_i)^2)$.
  if update-interval steps have passed **then**
    $\theta^- = \theta$
  end if
end for

of normalization – by increasing the weights for transitions, where the monotonic mixing function is less sensitive to the underlying per-agent utility, and decreasing the weights otherwise. The measurement of on-policy transitions in the weighting expression emphasizes the useful information carried by more current, on-policy transitions.

Following these theoretical results, we provide a tractable approximation of the optimal projection weights and propose a MARL algorithm, ReMIX, with regret-minimizing projections onto monotonic value function factorizations. The procedure of ReMIX can be found in Algorithm 1. We consider a new loss function with respect to the optimal projection weights w_k applied to the Bellman equation of Q_k (considering Q_{tot} at step k), i.e.,

$$L_{\text{ReMIX}} = \sum_{i=1}^{b} \left[w_i(s, \mathbf{u})(Q_k - y_i)^2(s, \mathbf{u}) \right], \tag{7}$$

where b is the batch size, and $y_i = \mathcal{B}^* Q_{k-1}^*$ is a fixed target using an unrestricted joint action-value function that can be approximated using a separate network similar to WQMIX.

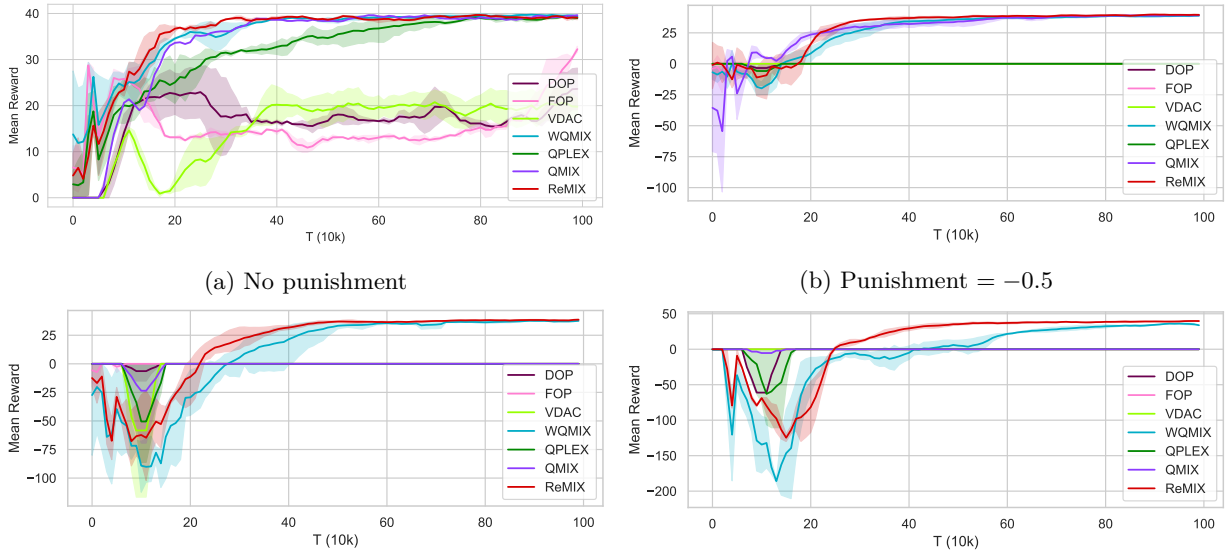
To compute the projection weights for Bellman error and value underestimation terms, we again leverage the unrestricted joint action-value function Q^* to compute them quantitatively. We note that the Bellman error term also works as the condition in Theorem 1 for deciding whether the weight should be zero. The gradient of the monotonic mixing network can be directly computed using Lemma 1. Since the distribution $d^{\boldsymbol{\pi}_k}(s, \mathbf{u})$ in the numerator of the measurement of on-policy transitions term is not readily available, we

approximated this term via an additional replay buffer storing the recently generated transitions and sample from it to obtain trajectories that are more possibly to be visited by current policy. To account for the unknown normalization factor Z^* and improve the stability of the training process, we map the projection weights to a given range, which is modeled as a hyperparameter of our algorithm. We provide numerical results adjusting it in the experiment section.

5 Experiments

In this section, we present our experimental results on Predator-Prey and SMAC and demonstrate the effectiveness of ReMIX by comparing the results with several state-of-the-art MARL baselines. Besides, we visualize the optimal weight pattern in heat maps to show the step-wise weight assignment for each transition. Additionally, we conduct the ablation experiments by disabling each term in Theorem 1, and deliver the sensitivity experiments regarding the normalization factor. More details about the environment and hyper-parameter setting are provided in Appendix D. The code of this work is available on GitHub (see supplementary files during the review period).

5.1 Predator-Prey



(a) No punishment

(b) Punishment $= -0.5$

(c) Punishment $= -1.5$

(d) Punishment $= -2$

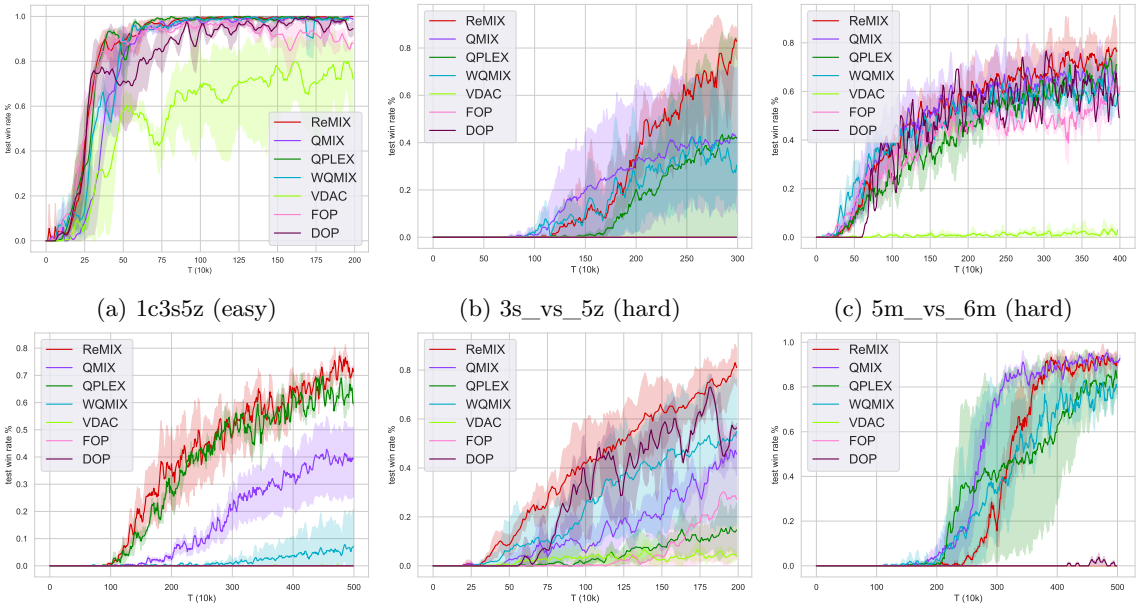
Figure 1: Average reward per episode on the Predator-Prey tasks for ReMIX and other baseline algorithms of 4 settings.

To start with, we consider a complex partially-observable multi-agent cooperative environment, Predator-Prey, that involves 8 agents in cooperation as predators to catch 8 prey on a 10×10 grid. In this task, a successful capture with the positive reward of 1 must include two or more predator agents surrounding and catching the same prey simultaneously, requiring a high level of cooperation. A failed coordination between agents to capture the prey, which happens when only one predator catches the prey, will receive a negative punishment reward. The greater punishment determines the degree of monotonicity. Algorithms that suffer from relative overgeneralization issues or make poor trade-offs in joint action-value function projection will fail to solve this task.

We select multiple state-of-the-art MARL approaches as baseline algorithms for comparison, which include value-based factorization algorithm (i.e., QMIX, WQMIX, and QPLEX), decomposed policy gradient method (i.e., VDAC), and decomposed actor-critic approaches (i.e., FOP and DOP). All mentioned baseline algorithms have shown strength in handling MARL tasks in existing works.

Figure 1 shows the performance of seven algorithms with different punishments, where all results demonstrate the superiority of ReMIX over others. Besides, regarding efficiency, we can spot that ReMIX has the fastest convergence speed in seeking the best policy. In Figure 1c and 1d, ReMIX significantly outperforms other state-of-the-art algorithms in a hard setting requiring a higher level of coordination among agents as learning the best policy with improved joint action representation is required in this setting. Most algorithms, such as QMIX, FOP, and DOP, end up learning a sub-optimal policy where agents learn to work together with limited coordination. Although ReMIX and WQMIX acquired good results eventually, compared to the latter, ReMIX achieves better performance and converges to the optimal policy profoundly faster than WQMIX, demonstrating that our optimal weighting approach can generate a better joint action-value projection.

5.2 SMAC



(a) 1c3s5z (easy) (b) 3s_vs_5z (hard) (c) 5m_vs_6m (hard)

(d) 6h_vs_8z (super hard) (e) MMM2 (super hard) (f) corridor (super hard)

Figure 2: Results of 6 maps (from easy to super hard) on the SMAC benchmark.

Next, we evaluate ReMIX on the SMAC benchmark. We report the experiments on six maps consisting of one easy map, two hard maps, and three super-hard maps. The selected state-of-the-art baseline algorithms for this experiment are consistent with those in the Predator-Prey environment. The empirical results are provided in Figure 2, demonstrating that ReMIX can effectively generate optimal weight projection for joint actions on SMAC for achieving a higher win rate, especially when the environment becomes substantially complicated and harder, such as *MMM2*. We can see that several state-of-the-art policy-based factorization algorithms are brittle when significant exploration is undergone since joint action representations generated by them are sub-optimal.

Specifically, ReMIX performs well on an easy map *1c3s5z* in Figure 2a, albeit holding the comparable performance among algorithms. On hard maps, such as *3s_vs_5z*, the best policy found by our optimal weighting approach significantly outperforms the remaining baseline algorithms regarding winning rate. For super-hard map *6h_vs_8z*, *MMM2*, and *corridor*, ReMIX, along with QMIX, WQMIX, and QPLEX, can learn a better policy than VDAC, DOP, and FOP. We achieve the highest winning rate by adopting our algorithm on *6h_vs_8z* and *MMM2*. Compared to our method, QMIX and WQMIX suffer from this map as their joint action representations are oblivious to some latent factors, such as the shape of the monotonic mixing network, and therefore fail to generate an accurate joint action representation. On *corridor*, ReMIX manages to learn the model with better performance than WQMIX, QPLEX, and other policy-based algorithms, though standard QMIX has the fastest convergence rate among all baseline algorithms.

5.3 Optimal Weight Pattern

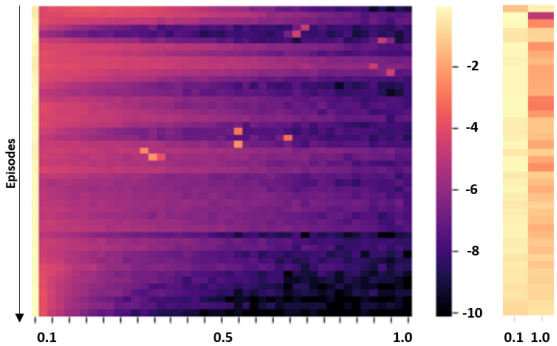


Figure 3: Heatmap pattern of generated optimal weights (left) and WQMIX weights (right) used in the Predator-Prey environment. The training episodes range from 0 to 1M.

In this part, we draw heat maps of the projecting weight probability distributions of ReMIX and WQMIX as the training proceeds to better visualize and compare the weight evolution pattern of transitions sampled as in a minibatch, shown in Figure 3. Adopted weights are generated from the Predator-Prey task with a punishment of -2. We re-scale the absolute value of the transition number to logarithmic probability for scale normalization. As shown in the figure, the probability value of a certain weight is represented by colors, decreasing from 0 in light yellow to -10 in black. The vertical axis represents the training steps, and the horizontal axis represents the normalized weight value, where ours ranges from 0.1 to 1 and WQMIX is either 0.1 or 1.

The heat map effectively shows the general trend of the weight evolution pattern at different steps. For WQMIX on the Figure 3 right, with the training of the algorithm, the transitions with the smaller weight (0.1) will become more, and those with the larger weight (1) will become fewer. Evolution like this happens since the transitions will approach optimal as the training goes on, while the algorithm will still take all transitions as potential overestimations and assign smaller weights to them as adjustments. A similar evolution pattern can be found in our weight pattern. On the left of Figure 3, during the training, the transitions with higher weights become less, and most transitions will migrate to the bottom right with lower weights, which empirically recovers the heuristic in WQMIX.

Moreover, as an optimal weight projection is used in ReMIX, we will assign different weights to transitions based on evaluating every one of them. We notice that some transitions are assigned with medium weight during the training, given by the light yellow spots on the left of Figure 3. Such a phenomenon demonstrates that the binary-weighted projections in WQMIX are not always accurate. Hence, ReMIX considers all transitions by applying optimal weights to their projections, leading to better results, which also illustrates the performance gap with other algorithms like WQMIX in previous experiments.

5.4 Sensitivity Experiment regarding Normalization

We run the experiment in the Predator-Prey environment with a punishment of -1.5 to report the sensitivity with respect to the different normalization of weight ranges. We keep the maximum normalized weight as 1 but test the effects of using different minimums, which are 0.1, 0.5, and 0.8.

As shown in Figure 4, the experiment results are sensitive to the range of the normalized weight. When we map the weight to a minimum of 0.5, the agents in this task can only find a sub-optimal solution. It may be because there exist many overestimations in this task. The joint action representation generated at the is not accurate. Higher minimum weight normalization damages the capability of ReMIX to adjust the projection to retrieve a precise representation rapidly. Therefore, ReMIX performs well under 0.1 to 1 normalization of the weight in this scenario. Note in WQMIX weight is used as $\alpha = 0.1$ for Predator-Prey and $\alpha = 0.5$ for SMAC according to their experiment settings.

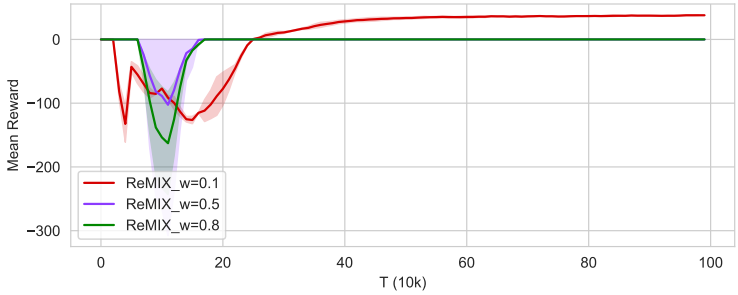


Figure 4: Sensitivity of normalizing the minimum weight to 0.1, 0.5, and 0.8.

5.5 Ablation Experiment

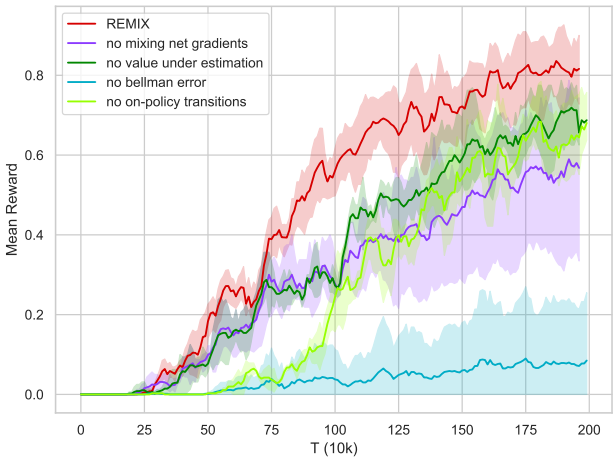


Figure 5: Ablation by disabling one term each for ReMIX on MMM2 (super hard)

For ablations, we conduct experiments by disabling one single term (in Theorem 1) each at a time to investigate their contribution to finding optimal projection weights, respectively. The ablation results are given in Figure 5 showing the results on *MMM2*. The terms considered in these experiments are the Bellman error, value underestimation, gradient of the mixing network, and measurement of on-policy transitions. Compared to the original result, missing any terms will harm the performance. The tests without Bellman error have the lowest final winning rate, which is less than 10%, and the test result without on-policy transitions shows a relatively slower convergence speed with less optimal performance. Furthermore, when we disable the gradient of the mixing network term, the result is only around 60%, demonstrating that providing a quantitative weight factorization for the value projection is the critical factor in value-factorization-based MARL tasks. The designing of an optimal weighting scheme without considering the mixing network's influence will be less capable of achieving the ideal final results.

6 Conclusion

In this paper, we formulate the optimal value function factorization as a policy regret minimization and solve the optimal projection weights for the cooperative multi-agent reinforcement learning problems in closed form. The theoretical results shed light on key factors for an optimal projection. Therefore, we propose ReMIX as a tractable weight approximation approach to enable MARL algorithms with improved value function factorization. Our experiment results in multiple MARL environments show the effectiveness of ReMIX by demonstrating superior convergence and empirical performance over state-of-the-art factorization-based methods.

References

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *International Conference on Machine Learning*, pp. 980–991. PMLR, 2020.

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.

Jacopo Castellini, Frans A Oliehoek, Rahul Savani, and Shimon Whiteson. The representational capacity of action-value networks for multi-agent reinforcement learning. *arXiv preprint arXiv:1902.07497*, 2019.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating functional knowledge in neural networks. *Journal of Machine Learning Research*, 10(6), 2009.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Kkt conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey. *arXiv preprint arXiv:2110.01858*, 2021.

Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pp. 227–234. Citeseer, 2002.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International conference on autonomous agents and multiagent systems*, pp. 66–83. Springer, 2017.

Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih-wei Liao. Riit: Rethinking the importance of implementation tricks in multi-agent reinforcement learning. *arXiv preprint arXiv:2102.03479*, 2021.

Yeping Hu, Alireza Nakhaei, Masayoshi Tomizuka, and Kikuo Fujimura. Interaction-aware decision making with adaptive strategies under merging scenarios. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 151–158. IEEE, 2019.

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

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*, pp. 2961–2970. PMLR, 2019.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*, pp. 3040–3049. PMLR, 2019.

Peter Jin, Kurt Keutzer, and Sergey Levine. Regret minimization for partially observable deep reinforcement learning. In *International conference on machine learning*, pp. 2342–2351. PMLR, 2018.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

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

Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Xu-Hui Liu, Zhenghai Xue, Jingcheng Pang, Shengyi Jiang, Feng Xu, and Yang Yu. Regret minimization experience replay in off-policy reinforcement learning. *Advances in Neural Information Processing Systems*, 34:17604–17615, 2021.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *arXiv preprint arXiv:1910.07483*, 2019.

Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Twenty-sixth AAAI conference on artificial intelligence*, 2012.

Edward James McShane. Jensen's inequality. *Bulletin of the American Mathematical Society*, 43(8):521–527, 1937.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning, 2020.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 5887–5896. PMLR, 2019.

Jianyu Su, Stephen Adams, and Peter Beling. Value-decomposition multi-agent actor-critics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11352–11360, 2021.

Kefan Su and Zongqing Lu. Divergence-regularized multi-agent actor-critic. In *International Conference on Machine Learning*, pp. 20580–20603. PMLR, 2022.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

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

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020a.

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

Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Dop: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*, 2020c.

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.

Tianhao Zhang, Yueheng Li, Chen Wang, Guangming Xie, and Zongqing Lu. Fop: Factorizing optimal joint policy of maximum-entropy multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 12491–12500. PMLR, 2021.

A Nomenclature

Table 1 summarizes the common notations in this paper.

Table 1: Definitions of notations.

Notation	Definition
s	State of the environment
a	Agent
$\mathbf{u}$	Agents' joint action
r	Reward
γ	Discount factor
π	Individual policy
$\boldsymbol{\pi}$	Joint policy
$\boldsymbol{\pi}^*$	Expected optimal joint policy
$\eta(\boldsymbol{\pi})$	Expected return under the joint policy $\boldsymbol{\pi}$
$d^{\boldsymbol{\pi}}(s)$	Discounted state distribution
$Q(\cdot)$	Action value function
$Q_{tot}(\cdot)$	Monotonic mixing of per-agent action value function
$Q^*(\cdot)$	Unrestricted joint action value function
$V(\cdot)$	Value function
$A(\cdot)$	Advantage function
$f_s(\cdot)$	Monotonic function with input state s
$\mathcal{B}^*$	Bellman operator, where $\mathcal{B}^*Q(s,\mathbf{u}) \stackrel{\text{def}}{=} r(s,\mathbf{u}) + \gamma \arg\max_{\mathbf{u}'} \mathbb{E}_{s'} Q(s',\mathbf{u}')$
w	Projection weights of transitions

B Proof of Lemma 1

Considering a two-layer mixing network of the non-negative weight matrix W_1, W_2, bias b_1, b_2 and activation function $h(\cdot)$. The input $\vec{Q}$ is the vector of all the agents' utilities. Assume there are n agents, $\vec{Q}$ is:

$$\vec{Q} = [Q^1, \ldots, Q^n]^{\mathrm{T}}$$

We assume the mixing network has the width of m, based on the input/output dimension, W_1 should be a $n \times m$ matrix as:

$$W_1 = \begin{bmatrix} w^1_{11} & \ldots & w^1_{1m} \\ \vdots & \ddots & \vdots \\ w^1_{n1} & \ldots & w^1_{nm} \end{bmatrix},$$

and W_2 is a m-dimension vector given by:

$$W_2 = [w^2_1, \ldots, w^2_m]^{\mathrm{T}}.$$

Therefore, Q_{tot} calculated from the utility vector $\vec{Q}$ becomes:

$$f_s(\vec{Q}) = h(\vec{Q}^{\mathrm{T}} W_1 + b_1) W_2^{\mathrm{T}} + b_2. \tag{8}$$

Considering one of the utilities Q^a, as long as the derivative of activation $h(\cdot)$ exists ($h(\cdot)$ is smooth and differentiable), based on equation 8, the result is:

$$f'_{s,Q^a} = \frac{\partial Q_{tot}}{\partial Q^a} = h'_{Q^a}(\vec{Q}^{\mathrm{T}} W_1 + b_1) \sum_{j=1}^{m} w^1_{aj} w^2_j. \tag{9}$$

This concludes the proof.

C Proof of Theorem 1

We have provided the outline of the proof containing four key steps. In this section, we present detailed proof of the theorem. Following the existing work (Su & Lu, 2022), we will use state-based policies for simplicity. Since we focus on the regret bound of the action value and define the Boltzmann policy in the regret via action value, i.e., Q_{tot} and Q^*, we aim to find the return gap between two action values characterized by Boltzmann policy function under centralized training.

The original optimization problem needs solving is:

$$\begin{aligned}
\min_{w_k} \quad & \eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \\
\text{s.t.} \quad & Q_k = \arg\min_{Q \in \mathcal{Q}} \mathbb{E}_\mu[w_k(s,\mathbf{u})(Q - \mathcal{B}^* Q_{k-1}^*)^2(s,\mathbf{u})], \\
& \mathbb{E}_\mu[w_k(s,\mathbf{u})] = 1, \quad w_k(s,\mathbf{u}) \geq 0, \\
& Q_k(s,\mathbf{u}) = f_s(Q^1(\tau_1,u_1),\ldots,Q^n(\tau_n,u_n)),
\end{aligned}$$

and this problem is equivalent to:

$$\begin{aligned}
\min_{p_k} \quad & \eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \\
\text{s.t.} \quad & Q_k = \arg\min_{Q \in \mathcal{Q}} \mathbb{E}_{p_k}[(Q - \mathcal{B}^* Q_{k-1}^*)^2(s,\mathbf{u})], \\
& \sum_{s,\mathbf{u}} p_k(s,\mathbf{u}) = 1, \quad p_k(s,\mathbf{u}) \geq 0, \\
& Q_k(s,\mathbf{u}) = f_s(Q^1(\tau_1,u_1),\ldots,Q^n(\tau_n,u_n)),
\end{aligned} \tag{10}$$

where $p_k = w_k(s,\mathbf{u})\mu(s,\mathbf{u})$ is the solution to problem equation 10.

To solve the optimization problem in equation 10, we needed to provide some definitions, which are *total variation distance*, *Wasserstein metric*, *the diameter of a set*, and *universal approximator*.

Definition 1 (Total variation distance)**.** *The total variation distance of the distribution P and Q is defined as $D(P,Q) = \frac{1}{2}\|P - Q\|$.*

Definition 2 (Wasserstein metric)**.** *For F, G two cumulative distribution functions over the reals, the Wasserstein metric is defined as $d_p(F,G) \stackrel{\text{def}}{=} \inf_{U,V} \|U - V\|_p$, where the infimum is taken over all pairs of random variables (U,V) with cumulative distributions F and G, respectively.*

Definition 3 (Diameter of a set)**.** *The diameter of a set A is defined as $\operatorname{diam}(A) = \sup_{x,y \in A} m(x,y)$, where m is the metric on A.*

Definition 4 (Universal approximator)**.** *A class of function $\hat{\mathcal{F}}$ from $\mathbb{R}^n$ to $\mathbb{R}$ is a universal approximator for a class of functions $\mathcal{F}$ from $\mathbb{R}^n$ to $\mathbb{R}$ if for any $f \in \mathcal{F}$, any compact domain $D \subset \mathbb{R}^n$, and any positive ϵ, one can find a $\hat{f} \in \hat{\mathcal{F}}$ with $\sup_{x \in D} |f(x) - \hat{f}(x)| \leq \epsilon$.*

Furthermore, we introduce some mild assumptions as follows:

Assumption 1. *The state space S, action space U, and observation space Z are compact metric spaces.*

Assumption 2. *The action-value and observation functions are continuous on $S \times U$ and Z, respectively.*

Assumption 3. *The transition function T is continuous with respect to $S \times U$ in the sense of Wasserstein metric, which is $\lim_{(s,\mathbf{u}) \to (s_0,\mathbf{u}_0)} d_p(T(\cdot|s,\mathbf{u}), T(\cdot|s_0,\mathbf{u}_0))$.*

Assumption 4. *The joint policy $\boldsymbol{\pi}$ is the product of each agent's individual policy $\pi^a(u_a|s_a)$.*

Assumption 5. *The monotonic mixing function $f_s(\cdot)$ regarding per-agent action-value function Q^a for $\forall a \in A$ is smooth and differentiable.*

These assumptions are not strict and can be satisfied in most MARL environments.

Let $d^{\pi^a}(s)$ denote the discounted state distribution of agent a, and $d_i^{\pi^a}(s)$ denote the distribution where the state is visited by the agent for the i-th time. Thus, we have:

$$d^{\pi^a}(s) = \sum_{i=1}^{\infty} d_i^{\pi^a}(s), \tag{11}$$

where each $d_i^{\pi^a}(s)$ is given by:

$$d_i^{\pi^a}(s) = (1-\gamma) \sum_{t_i=0}^{\infty} \gamma^{t_i} \Pr(s_{t_i} = s, s_{t_k} = s, \forall k = 1, ..., i-1), \tag{12}$$

where the $\Pr(s_{t_i} = s, s_{t_k} = s, \forall k = 1, ..., i-1)$ in this equation contains the probability of visiting state s for the i-th time at t_i and a sequence of times t_k, for $k = 1, ..., i$, such that state s is visited at each t_k. Thus, state s will be visited for i times at time t_i in total.

The following lemmas are proposed by Liu et al. (2021), where Lemma 2 support the derivation of the Lemma 3, and the latter demonstrates that $\left|\frac{\partial d^{\pi^a}(s)}{\partial \pi^a(s)}\right|$ is a small quantity.

Lemma 2. *Let f be an Lebesgue integrable function. P and Q are two probability distributions, $f \leq C$, then:*

$$|\mathbb{E}_{P(x)} f(x) - \mathbb{E}_{Q(x)} f(x)| \leq C \cdot D(P, Q). \tag{13}$$

Lemma 3. *Let ρ be the probability of the agent a starting from (s, u^a) and coming back to s at time step t under policy π^a, i.e. $\Pr(s_0 = s, u_0^a = u^a, s_t = s, s_{1:t-1} \neq s; \pi^a)$, and $\epsilon = \sup_{s,u^a} \sum_{t=1}^{\infty} \gamma^t \rho^{\pi^a}(s, u^a, t)$. We have:*

$$\left|\frac{\partial d^{\pi^a}(s)}{\partial \pi^a(s)}\right| \leq \epsilon d_1^{\pi^a}(s), \tag{14}$$

where $d_1^{\pi^a}(s) = (1-\gamma) \sum_{t_1=0}^{\infty} \gamma^{t_1} \Pr(s_{t_1} = s)$ and $\epsilon \leq 1$.

Lemma 1 and 2 can be extended to suit the multi-agent scenario. Besides, we have the following lemma holds in MARL:

Lemma 4. *Given two policy $\boldsymbol{\pi}$ and $\bar{\boldsymbol{\pi}}$, where $\boldsymbol{\pi} = \frac{\exp(Q(s,\mathbf{u}))}{\sum_{\mathbf{u}'} \exp(Q(s,\mathbf{u}'))}$ is defined by Boltzmann policy, we have:*

$$\mathbb{E}_{\mathbf{u}\sim\bar{\boldsymbol{\pi}}}[Q(s, \mathbf{u})] - \mathbb{E}_{\mathbf{u}\sim\boldsymbol{\pi}}[Q(s, \mathbf{u})] \leq 1. \tag{15}$$

Proof. Suppose there are two joint actions $\mathbf{u}$ and $\bar{\mathbf{u}}$. Let $Q(s, \mathbf{u}) = p$, $Q(s, \bar{\mathbf{u}}) = q$ and let $p \leq q$.

$$\begin{aligned}
\mathbb{E}_{\mathbf{u}\sim\bar{\boldsymbol{\pi}}}[Q(s, \mathbf{u})] - \mathbb{E}_{\mathbf{u}\sim\boldsymbol{\pi}}[Q(s, \mathbf{u})] &\leq q - \frac{pe^p + qe^q}{e^p + e^q} \\
&= q - \frac{p + qe^{q-p}}{1 + e^{q-p}} \\
&= q - p - \frac{(q-p)e^{q-p}}{1 + e^{q-p}}.
\end{aligned}$$

Let $f(z) = z - \frac{ze^z}{1+e^z}$, the maximum point z_0 satisfies $f'(z) = 0$, from which we further have $1 + e^{z_0} = z_0 e^{z_0}$ where $z_0 \in (1, 2)$. Therefore, we have

$$\mathbb{E}_{\mathbf{u}\sim\bar{\boldsymbol{\pi}}}[Q(s, \mathbf{u})] - \mathbb{E}_{\mathbf{u}\sim\boldsymbol{\pi}}[Q(s, \mathbf{u})] \leq f(q-p) \leq z_0 - 1 \leq 1.$$

It is worth noting that the derived inequality can also be applied to the situation where we have joint action more than two or we consider the situation regarding per-agent action. □

The following lemma is introduced by Kakade & Langford (2002). It was originally proposed for the finite MDP, while it will also hold for the continuous scenario that is given by Assumption 1 and 2.

Lemma 5. *For any policy $\boldsymbol{\pi}$ and $\tilde{\boldsymbol{\pi}}$, we have*

$$\eta(\tilde{\boldsymbol{\pi}}) - \eta(\boldsymbol{\pi}) = \frac{1}{1-\gamma}\mathbb{E}_{d^{\tilde{\boldsymbol{\pi}}}(s,\mathbf{u})}[A^{\boldsymbol{\pi}}(s,\mathbf{u})], \tag{16}$$

where $A^{\boldsymbol{\pi}}(s,\mathbf{u})$ is the advantage function given by $A^{\boldsymbol{\pi}}(s,\mathbf{u}) = Q^{\boldsymbol{\pi}}(s,\mathbf{u}) - V^{\boldsymbol{\pi}}(s)$.

Lemma 6. *Let $\epsilon_{\boldsymbol{\pi}_k} = \sup_{s,\mathbf{u}} \sum_{t=1}^{\infty} \gamma^t \rho^{\boldsymbol{\pi}}(s,\mathbf{u},t)$, the optimal solution p_k to a relaxation of optimization problem in equation 10 satisfies relationship as follows:*

$$p_k(s,\mathbf{u}) = \frac{1}{Z^*}(D_k(s,\mathbf{u}) + \epsilon_k(s,\mathbf{u})), \tag{17}$$

where when $Q_k \leq \mathcal{B}^* Q^*_{k-1}$, we have $D_k(s,\mathbf{u}) = d^{\boldsymbol{\pi}_k}(s,\mathbf{u})(\mathcal{B}^* Q^*_{k-1} - Q_k)\exp(Q^*_{k-1} - Q_k)(\sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} - 1)$, and when $Q_k > \mathcal{B}^* Q^*_{k-1}$, we have $D_k(s,\mathbf{u}) = 0$. Z^* is the normalization constant.

Proof. Suppose $\mathbf{u}^* \sim \boldsymbol{\pi}^*$. Let $\boldsymbol{\pi} = \boldsymbol{\pi}^*$ and $\tilde{\boldsymbol{\pi}} = \boldsymbol{\pi}_k$ in Lemma 5, we have

$$\begin{aligned}
&\eta(\boldsymbol{\pi}^*) - \eta(\boldsymbol{\pi}_k) \\
&= -\frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[A^{\boldsymbol{\pi}^*}(s,\mathbf{u})] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[V^*(s) - Q^*(s,\mathbf{u})] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[V^*(s) - Q_k(s,\mathbf{u}^*) + Q_k(s,\mathbf{u}^*) - Q_k(s,\mathbf{u}) + Q_k(s,\mathbf{u}) - Q^*(s,\mathbf{u})] \\
&\overset{(a)}{\leq} \frac{1}{1-\gamma}\left[\mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}(Q^*(s,\mathbf{u}^*) - Q_k(s,\mathbf{u}^*)) + \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}(Q_k(s,\mathbf{u}) - Q^*(s,\mathbf{u})) + 1\right],
\end{aligned} \tag{18}$$

where (a) uses Lemma 4. □

Since the original optimization is non-tractable, we consider this upper bound to obtain a closed-form solution. Therefore, we replace the objective in equation 10 with the upper bound in equation 18 and solve the relaxed optimization problem, given by

$$\begin{aligned}
\min_{p_k} \quad & \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[(Q^*_{k-1} - Q_k)(s,\mathbf{u}^*)] + \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[(Q_k - Q^*_{k-1})(s,\mathbf{u})] \\
\text{s.t.} \quad & Q_k = \arg\min_{Q\in\mathcal{Q}} \mathbb{E}_{p_k}[(Q - \mathcal{B}^* Q^*_{k-1})^2(s,\mathbf{u})], \\
& \sum_{s,\mathbf{u}} p_k(s,\mathbf{u}) = 1, \quad p_k(s,\mathbf{u}) \geq 0, \\
& Q_k(s,\mathbf{u}) = f_s(Q^1(\tau_1,u_1),\ldots,Q^n(\tau_n,u_n)),
\end{aligned} \tag{19}$$

The derived objective in equation 19 can be further relaxed with Jensen's inequality, given by:

$$\mathbb{E}[g(X)] \geq g(\mathbb{E}[X]), \tag{20}$$

when $g(x)$ is a convex function on real space $\mathbb{R}$.

According to equation 20, we select the convex function $g(x) = \exp(-x)$, and the objective can be further relaxed as:

$$\begin{aligned}
\min_{p_k} \quad & -\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s,\mathbf{u}^*)] - \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[\exp(Q^*_{k-1} - Q_k)(s,\mathbf{u})] \\
\text{s.t.} \quad & Q_k = \arg\min_{Q\in\mathcal{Q}} \mathbb{E}_{p_k}[(Q - \mathcal{B}^* Q^*_{k-1})^2(s,\mathbf{u})], \\
& \sum_{s,\mathbf{u}} p_k(s,\mathbf{u}) = 1, \quad p_k(s,\mathbf{u}) \geq 0, \\
& Q_k(s,\mathbf{u}) = f_s(Q^1(\tau_1,u_1),\ldots,Q^n(\tau_n,u_n)),
\end{aligned} \tag{21}$$

In order to handle the optimization problem in equation 21, we follow the standard procedures of Lagrangian multiplier method, which is:

$$\mathcal{L}(p_k;\lambda,\nu) = -\log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s,\mathbf{u}^*)] - \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[\exp(Q^*_{k-1} - Q_k)(s,\mathbf{u})] + \lambda(\sum_{s,\mathbf{u}} p_k - 1) - \nu^{\mathrm{T}} p_k, \tag{22}$$

After constructing the Lagrangian, we further compute some gradients that will be used in calculating the optimal solution. We first calculate the $\frac{\partial Q_k}{\partial p_k}$ according to the implicit function theorem (IFT). Based on the first constraint in equation 21, we aim to find the minimum Q_k to satisfy the $\arg\min(\cdot)$, and therefore we need to ensure the derivative of the term inside $\arg\min(\cdot)$ (we use $f(p_k, Q_k)$ to denote this term) to be zero, which is:

$$f'_{Q_k} = 2\sum_{s,\mathbf{u}} p_k(Q_k - \mathcal{B}^* Q_{k-1}) = 0 \tag{23}$$

We can notice that $F(p_k, Q_k) : f'_{Q_k} = 0$ is an implicit function regarding Q_k and p_k. Hence, we apply the IFT on the $F(p_k, Q_k)$ considering the Hessian matrices of p_k and Q_k in $f(p_k, Q_k)$ as follows:

$$\frac{\partial Q_k}{\partial p_k} = -\frac{F'_{p_k}}{F'_{Q_k}} = -\left[\mathrm{diag}(p_k)\right]^{-1}\left[\mathrm{diag}(Q_k - \mathcal{B}^* Q^*_{k-1})\right]. \tag{24}$$

Next, we derive the expression for $\frac{\partial d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}{\partial p_k}$ in the following equation:

$$\begin{aligned}
\frac{\partial d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}{\partial p_k} &= \frac{\partial d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}{\partial \boldsymbol{\pi}_k}\frac{\partial \boldsymbol{\pi}_k}{\partial Q^a}\frac{\partial Q^a}{\partial Q_k}\frac{\partial Q_k}{\partial p_k} \\
&= \mathrm{diag}(d^{\boldsymbol{\pi}_k}(s) + \epsilon_0(s))\frac{\partial \boldsymbol{\pi}_k}{\partial Q^a}\frac{\partial Q^a}{\partial Q_k}\frac{\partial Q_k}{\partial p_k} \\
&\overset{(b)}{=} \mathrm{diag}(d^{\boldsymbol{\pi}_k}(s) + \epsilon_0(s))\mathrm{diag}(\boldsymbol{\pi}_k(1-\boldsymbol{\pi}_k))\frac{\partial Q^a}{\partial Q_k}\frac{\partial Q_k}{\partial p_k} \\
&\overset{(c)}{=} d^{\boldsymbol{\pi}_k}(s,\mathbf{u})(1-\boldsymbol{\pi}_k)\frac{1}{f'_{s,Q_k}}\frac{\partial Q_k}{\partial p_k} + \epsilon_0(s)\boldsymbol{\pi}_k(1-\boldsymbol{\pi}_k)\frac{1}{f'_{s,Q_k}}\frac{\partial Q_k}{\partial p_k},
\end{aligned} \tag{25}$$

where $\epsilon_0(s) = \frac{\partial d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}{\partial \boldsymbol{\pi}_k(s)}$ is a small quantity provided by Lemma 3. Besides, (b) is based on the the definition of the Boltzmann policy and Assumption 4, and (c) is based on Assumption 5 the gradient of the monotonic mixing function in Lemma 1.

Since we have all the preparations ready, we now compute the Lagrangian by applying the Karush–Kuhn–Tucker (KKT) condition. We let the Lagrangian gradient to be zero, i.e.,

$$\frac{\partial \mathcal{L}(p_k;\lambda,\nu)}{\partial p_k} = 0 \tag{26}$$

Besides, the partial derivative of the Lagrangian can be computed as:

$$\begin{aligned}
\frac{\partial \mathcal{L}(p_k;\lambda,\nu)}{\partial p_k} &= -\frac{\partial \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s)}[\exp(Q_k - Q^*_{k-1})(s,\mathbf{u}^*)]}{\partial p_k} - \frac{\partial \log \mathbb{E}_{d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}[\exp(Q^*_{k-1} - Q_k)(s,\mathbf{u})]}{\partial p_k} + \lambda - \nu_{s,\mathbf{u}} \\
&= -\frac{1}{Z}\exp(Q^*_{k-1} - Q_k)\left(\frac{\partial d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}{\partial p_k} - d^{\boldsymbol{\pi}_k}(s,\mathbf{u})\frac{\partial Q_k}{\partial p_k}\right) + \lambda - \nu_{s,\mathbf{u}},
\end{aligned} \tag{27}$$

where $Z = \mathbb{E}_{s',\mathbf{u}'\sim d^{\boldsymbol{\pi}_k}(s,\mathbf{u})}\exp(Q^* - Q_k)(s',\mathbf{u}')$.

Based on equation 26 and equation 27, and substituting the expression of $\frac{\partial Q_k}{\partial p_k}$ and $\frac{\partial d^{\boldsymbol{\pi}_k}(s,a)}{\partial p_k}$ with the derived results in equation 24 and equation 25, we obtain:

$$\begin{aligned} p_k(s,\mathbf{u}) = &\frac{1}{Z(\nu^*_{s,\mathbf{u}} - \lambda^*)} \Bigg[d^{\boldsymbol{\pi}_k}(s,\mathbf{u})(Q_k - \mathcal{B}^* Q^*_{k-1}) \exp(Q^*_{k-1} - Q_k) \left(\sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} - 1 \right) \\ &+ \epsilon_0 \boldsymbol{\pi}_k (Q_k - \mathcal{B}^* Q^*_{k-1}) \exp(Q^*_{k-1} - Q_k) \sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} \Bigg], \end{aligned} \tag{28}$$

According to Lemma 3, the value of ϵ_0 is smaller than $d^{\boldsymbol{\pi}_k}(s)$ so the second term will not influence the sign of the equation. Equation 28 will always be larger or equal to zero. By KKT condition, when the $Q_k - \mathcal{B}^* Q^*_{k-1} < 0$, we have $\nu^*_{s,\mathbf{u}} = 0$. When equation 28 equal to zero, we let $\nu^*_{s,\mathbf{u}} = 0$ because the value of $\nu^*_{s,\mathbf{u}}$ will not affect p_k. In the contrast, when the $Q_k - \mathcal{B}^* Q^*_{k-1} > 0$, the p_k should equal to zero. Therefore, by introducing a normalization factor Z^*, equation 28 can be simplify as follows:

$$p_k(s,\mathbf{u}) = \frac{1}{Z^*}(D_k(s,\mathbf{u}) + \epsilon_k(s,\mathbf{u})), \tag{29}$$

where when $Q_k \leq \mathcal{B}^* Q^*_{k-1}$, we have

$$\begin{aligned} D_k(s,\mathbf{u}) &= d^{\boldsymbol{\pi}_k}(s,\mathbf{u})(\mathcal{B}^* Q^*_{k-1} - Q_k) \exp(Q^*_{k-1} - Q_k) \left(\sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} - 1 \right) \\ \epsilon_k &= \epsilon_0 \boldsymbol{\pi}_k (Q_k - \mathcal{B}^* Q^*_{k-1}) \exp(Q^*_{k-1} - Q_k) \sum_{j=1}^{n} \frac{1-\pi^j}{f'_{s,Q^j}} \end{aligned} \tag{30}$$

and when $Q_k > \mathcal{B}^* Q^*_{k-1}$, we have

$$\begin{aligned} D_k(s,\mathbf{u}) &= 0 \\ \epsilon_k &= 0 \end{aligned} \tag{31}$$

This concludes the proof.

D Environment Details

We use more recent baselines (i.e., FOP and DOP) that are known to outperform QTRAN (Son et al., 2019) and QPLEX (Wang et al., 2020a) in the evaluation. In general, we tend to choose baselines that are more closely related to our work and most recent. This motivated the choice of QMIX (baseline for value-based factorization methods), WQMIX (close to our work that uses weighted projections so better joint actions can be emphasized), VDAC (Su et al., 2021), FOP (Zhang et al., 2021), DOP (Wang et al., 2020c) (SOTA actor-critic based methods). We acquired the results of QMIX, WQMIX based on their hyper-parameter tuned versions from pymarl2(Hu et al., 2021) and implemented our algorithm based on it.

D.1 Predator-Prey

A partially observable environment on a grid-world predator-prey task is used to model relative overgeneralization problem (Böhmer et al., 2020) where 8 agents have to catch 8 prey in a 10×10 grid. Each agent can either move in one of the 4 compass directions, remain still, or try to catch any adjacent prey. Impossible actions, i.e., moving into an occupied target position or catching when there is no adjacent prey, are treated as unavailable. If two adjacent agents execute the catch action, a prey is caught and both the prey and the catching agents are removed from the grid. An agent's observation is a 5×5 sub-grid centered around it, with one channel showing agents and another indicating prey. An episode ends if all agents have been removed or after 200 steps. Capturing a prey is rewarded with r = 10, but unsuccessful attempts by single agents are punished by a negative reward p. In this paper, we consider two sets of experiments with $p =$ (0, -0.5, -1.5, -2). The task is similar to the matrix game proposed by Son et al. (2019) but significantly more complex, both in terms of the optimal policy and in the number of agents.

Table 2: Hyperparameter value settings.

Hyperparameter	Value
Batch size	128
Replay buffer size	10000
Target network update interval	Every 200 episodes
Learning rate	0.001
TD-lambda	0.6

D.2 SMAC

For the experiments on StarCraft II micromanagement, we follow the setup of SMAC (Samvelyan et al., 2019) with open-source implementation including QMIX (Rashid et al., 2018), WQMIX (Rashid et al., 2020), QPLEX (Wang et al., 2020a), FOP (Zhang et al., 2021), DOP (Wang et al., 2020c) and VDAC (Su et al., 2021). We consider combat scenarios where the enemy units are controlled by the StarCraft II built-in AI and the friendly units are controlled by the algorithm-trained agent. The possible options for built-in AI difficulties are Very Easy, Easy, Medium, Hard, Very Hard, and Insane, ranging from 0 to 7. We carry out the experiments with ally units controlled by a learning agent while built-in AI controls the enemy units with difficulty = 7 (Insane). Depending on the specific scenarios(maps), the units of the enemy and friendly can be symmetric or asymmetric. At each time step each agent chooses one action from discrete action space, including noop, move[direction], attack[enemy_id], and stop. Dead units can only choose noop action. Killing an enemy unit will result in a reward of 10 while winning by eliminating all enemy units will result in a reward of 200. The global state information is only available in the centralized critic. Each baseline algorithm is trained with 4 random seeds and evaluated every 10k training steps with 32 testing episodes for main results, and with 3 random seeds for ablation results and additional results.

D.3 Implementation details and Hyperparameters

In this section, we introduce the implementation details and hyperparameters we used in the experiment. We carried out the experiments on NVIDIA 2080Ti with fixed hyperparameter settings. Recently Hu et al. (2021) demonstrated that MARL algorithms are significantly influenced by code-level optimization and other tricks, e.g. using TD-lambda, Adam optimizer, and grid-searched hyperparameters (where many state-of-the-art are already adopted), and proposed fine-tuned QMIX and WQMIX, which is demonstrated with significant improvements from their original implementation. We implemented our algorithm based on its open-sourced codebase and acquired the results of QMIX and WQMIX from it.

We use one set of hyperparameters for each environment, i.e., no tuned hyperparameters for individual maps. We use epsilon greedy for action selection with annealing from $\epsilon = 0.995$ decreasing to $\epsilon = 0.05$ in 100000 training steps in a linear way. The performance for each algorithm is evaluated for 32 episodes every 1000 training steps. More hyperparameter values are given in Table 2.