DISENTANGLING SOURCES OF RISK FOR DISTRIBUTIONAL MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

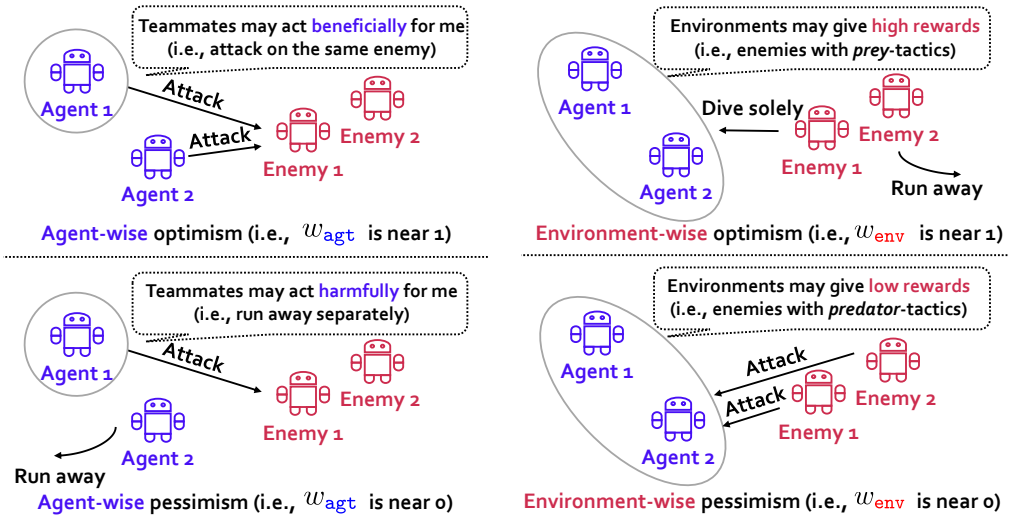
In cooperative multi-agent reinforcement learning, state transitions, rewards, and actions can all induce randomness (or uncertainty) in the observed long-term returns. These randomnesses are reflected from two risk sources: (a) *agent-wise* risk (i.e., how cooperative our teammates act for a given agent) and (b) *environment-wise* risk (i.e., transition stochasticity). Although these two sources are both important factors for learning robust policies of agents, prior works do not separate them or deal with only a single risk source, which could lead to suboptimal equilibria. In this paper, we propose **D**isentangled **RI**sk-sensitive **M**ulti-**A**gent reinforcement learning (DRIMA), a novel framework being capable of disentangling risk sources. Our main idea is to separate risk-level leverages (i.e., quantiles) in both centralized training and decentralized execution with a hierarchical quantile structure and quantile regression. Our experiments demonstrate that DRIMA significantly outperforms prior-arts across various scenarios in StarCraft Multi-agent Challenge. Notably, DRIMA shows robust performance regardless of reward shaping, exploration schedule, where prior methods learn only a suboptimal policy.

1 INTRODUCTION

In multi-agent reinforcement learning (MARL), centralized training with decentralized execution (CTDE) is one of the popular paradigms since it tackles non-stationarity issues caused by the changing behaviors of the agents (Oliehoek et al., 2016). Under this paradigm, value-based one has gained impressive attention, consisting of (i) training a joint action-value estimator with an access to global information in a centralized way and (ii) decompose the estimator into agent-wise utility functions, which can be executed in a decentralized manner without communication. Recently, several value-based CTDE methods have made training efficient and stable by designing the action-value estimator as flexible as possible while maintaining the execution constraint on decentralizability (Sunehag et al., 2018; Rashid et al., 2018; Son et al., 2019; Rashid et al., 2020a; Wang et al., 2020a).

However, despite the progress on value-based CTDE, agents still often fail to cooperate due to the environment randomness, limited observation of agents, and time-varying policies of other agents, especially in a highly stochastic environment. For such environment in single-agent settings, risk-sensitive reinforcement learning (Chow & Ghavamzadeh, 2014) has shown remarkable results by using policies that consider *risk* rather than simple expectations for return distribution caused by state transitions, rewards, and actions. Here, risk refers to the uncertainty over possible outcomes, and risk-sensitive policies act with a risk measure, such as variance or conditional value at risk (CVaR) (Chow & Ghavamzadeh, 2014). The main goal of this paper is to apply the risk-sensitive technique to value-based CTDE for learning more robust policies of agents against various factors of uncertainty.

To this end, our motivating observation in multi-agent settings is that the randomnesses to handle are reflected from two risk sources, (a) *agent-wise* and (b) *environment-wise*, which do not emerge in the single-agent setting. Here, the agent-wise risk is about how other agents behave cooperatively for a given agent, and the environment-wise risk is about how the environment reacts randomly (i.e., stochasticity of environmental dynamics) as illustrated in Figure 1. Unfortunately, existing value-based CTDE methods do not disentangle risk sources explicitly. For instance, recent approaches combining value-based CTDE with distributional RL (Qiu et al., 2021; Sun et al., 2021) learn the distribution of each agent-wise utility function but cannot separate different sources of risk. This can cause suboptimal policies under risk-sensitive RL; they cannot represent agent-wise risk-seeking



(a) Agent-wise risk

(b) Environment-wise risk

Figure 1: Two types of risk in MARL: (a) agent-wise risk and (b) environment-wise risk. Note that friendly agents learn policy over time, while enemies in the environment act through a stationary distribution. Current value-based CTDE methods do not consider risks explicitly or take them in an entangled way, which may lead to a suboptimal solution.

(cooperative) yet environment-wise risk-averse policies, which are likely to be of favorable choice in most practical scenarios (i.e., surviving long as a team with expecting cooperation of teammates).

Contribution. In this paper, we present **D**isentangled **RI**sk-sensitive **M**ulti-**A**gent reinforcement learning (DRIMA), a novel framework with disentangling sources of risk for distributional MARL. The main idea is to separate risk-level leverages (i.e., quantiles) in both centralized training and decentralized execution with a hierarchical quantile structure and quantile regression. To be specific, instead of entangled risk levels as prior works (Qiu et al., 2021; Sun et al., 2021), DRIMA takes agent-wise risk into each utility and environment-wise risk into a joint action-value estimator, respectively. Therefore, each utility is encouraged to learn the distribution with respect to another agents' cooperation, and the joint-action value estimator aims to learn distribution with respect to stochastic environments. Thanks to the disentanglement, DRIMA is able to obtain desirable policies more efficiently, while, as aforementioned, it is difficult to set agent-wise risk-seeking yet environment-wise risk-averse in prior-arts.

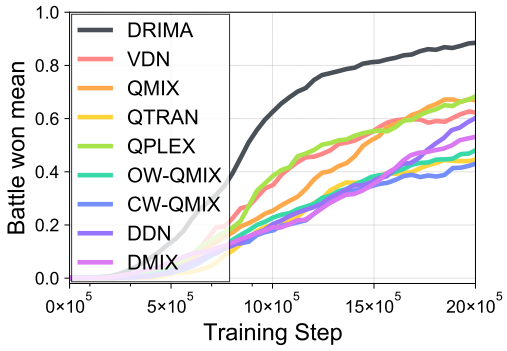


Figure 2: The median test win rate for DRIMA and eight value-based CTDE baselines, averaged across all 12 scenarios in our experiments. Results for each scenario are reported in Figure 4.

We demonstrate the effectiveness of DRIMA on various scenarios in StarCraft[1] Multi-Agent Challenge (SMAC), which is widely used in MARL literature (Samvelyan et al., 2019). As summarized in Figure 2, DRIMA significantly outperforms state-of-the-art methods, including distributional MARL methods (i.e., DDN and DMIX (Sun et al., 2021)) and non-distributional ones such as OW-QMIX, CW-QMIX, and QPLEX (Rashid et al., 2020a; Wang et al., 2020a). Notably, by disentangling sources of risk, DRIMA shows impressive results regardless of reward shaping and exploration schedule, whereas existing works learn only a suboptimal policy (see Figure 4). We expect that our idea of disentangling risks may motivate new interesting directions in the future (e.g., safe-MARL).

[1]StarCraft is a trademark of Blizzard Entertainment™.

2 PRELIMINARIES AND RELATED WORK

2.1 CENTRALIZED TRAINING WITH DECENTRALIZED EXECUTION

In this paper, we consider a decentralized partially observable Markov decision process (Oliehoek et al., 2016) represented by a tuple $\mathcal{G} = \langle \mathcal{S}, \mathcal{U}, P, r, O, N, \gamma \rangle$. To be specific, we let $s \in \mathcal{S}$ denote the true state of the environment. At each time step, an agent $i \in \mathcal{N} := \{1, ..., N\}$ selects an action $u_i \in \mathcal{U}$ as an element of the joint action vector $\boldsymbol{u} := [u_1, \cdots, u_N]$. It then goes through a stochastic transition dynamic described by the probability $P(s'|s, \boldsymbol{u})$. All agents share the same reward $r(s, \boldsymbol{u})$ discounted by a factor of γ. Each agent i is associated with a partial observation $o \in \mathcal{O}$, according to some observation function $O(s, i) : \mathcal{S} \times \mathcal{N} \mapsto \mathcal{O}$ and an action-observation history τ_i. Concatenation of the agent-wise action-observation histories is denoted as the overall action-observation history $\boldsymbol{\tau}$.

We consider value-based policies under the paradigm of centralized training with decentralized execution (CTDE). To this end, we train a joint action-value estimator $Q_{\texttt{jt}}(s, \boldsymbol{\tau}, \boldsymbol{u})$ with access to the overall action-observation history $\boldsymbol{\tau}$ and the underlying state s. Each agent i follows the policy of maximizing an agent-wise utility function $q_i(\tau_i, u_i)$ which can be executed in parallel without access to the state s. Finally, we denote the joint action-value estimator $Q_{\texttt{jt}}$ to be *decentralized* into agent-wise utility functions $q_1, \ldots, q_N$ when the following condition is satisfied:

$$\arg\max_{\boldsymbol{u}} Q_{\texttt{jt}}(s, \boldsymbol{\tau}, \boldsymbol{u}) = \big[\arg\max_{u_1} q_1(\tau_1, u_1), \ldots, \arg\max_{u_N} q_N(\tau_N, u_N) \big]. \tag{1}$$

Namely, the joint action-value estimator is decentralizable when there exist agent-wise policies maximizing the estimated true (joint) action-value. Under this paradigm, the two relevant works to DRIMA are QTRAN (Son et al., 2019) and Weighted QMIX (Rashid et al., 2020a), which consider agent-wise optimism for centralized training.

QTRAN. QTRAN (Son et al., 2019) is one of the popular value-based CTDE methods, which has strong theoretical capability of learning the largest class of true action-value functions up to date. QTRAN introduces a decentralizable *transformed* action-value estimator $Q_{\texttt{tran}}$ expressed as a linear summation over the utility functions and an additional value estimator. Then for each sample $(\boldsymbol{\tau}, \boldsymbol{u})$ collected from a replay buffer, QTRAN trains the transformed action-value estimator to approximate the behavior of "true" action-value estimator $Q_{\texttt{jt}}$ using two loss functions $\mathcal{L}_{\texttt{opt}}$ and $\mathcal{L}_{\texttt{nopt}}$:

$$\begin{aligned} \mathcal{L}_{\texttt{opt}}(\boldsymbol{\tau}) &= \big(Q_{\texttt{tran}}(\boldsymbol{\tau}, \boldsymbol{u}_{\texttt{opt}}) - Q_{\texttt{jt}}(\boldsymbol{\tau}, \boldsymbol{u}_{\texttt{opt}})\big)^2, \\ \mathcal{L}_{\texttt{nopt}}(\boldsymbol{\tau}, \boldsymbol{u}) &= \begin{cases} (Q_{\texttt{tran}}(\boldsymbol{\tau}, \boldsymbol{u}) - Q_{\texttt{jt}}(\boldsymbol{\tau}, \boldsymbol{u}))^2, & \text{if } Q_{\texttt{tran}}(\boldsymbol{\tau}, \boldsymbol{u}) \le Q_{\texttt{jt}}(\boldsymbol{\tau}, \boldsymbol{u}), \\ 0, & \text{otherwise}, \end{cases} \end{aligned} \tag{2}$$

where $\boldsymbol{u}_{\texttt{opt}}$ is the "optimal" action maximizing the utility functions $q_i(\tau_i, u_i)$ for $i \in \mathcal{N}$. This loss functions aim to learn agent-wise risk-seeking optimistic agents because it softly ignores learning of $Q_{\texttt{jt}}$ for non-optimal actions, and learns $Q_{\texttt{jt}}$ accurately only for optimal actions that agents are fully cooperative. However, they perform poorly in practical scenarios because QTRAN uses too small weights for non-optimal actions (Rashid et al., 2020a; Samvelyan et al., 2019).

Weighted QMIX. Rashid et al. (2020a) proposes CW-QMIX and OW-QMIX, which use a weighted projection that allows more emphasis to be placed on better joint actions. OW-QMIX applies weights to loss according to the sign of the TD-error for each sample:

$$\begin{aligned} \mathcal{L}(s, \boldsymbol{\tau}, \boldsymbol{u}) &= w(s, \boldsymbol{u})(Q_{\texttt{tran}}(s, \boldsymbol{\tau}, \boldsymbol{u}) - y)^2, \\ w(s, \boldsymbol{u}) &= \begin{cases} 1, & \text{if } Q_{\texttt{tran}}(s, \boldsymbol{\tau}, \boldsymbol{u}) \le y, \\ \alpha, & \text{otherwise}, \end{cases} \end{aligned} \tag{3}$$

where α is a hyperparameter and y is the true target action-value by unconstrained true action-value estimator $Q_{\texttt{jt}}$. Note that Rashid et al. (2020a) use $Q_{\texttt{tot}}$ instead of $Q_{\texttt{tran}}$ in their paper. Similar to $\mathcal{L}_{\texttt{nopt}}$ in QTRAN, the weighting mechanism according to the sign of this TD-error assigns a higher weighting for optimistic returns and learns risk-seeking policies to the cooperative. By optimistic training, they prove that Weighted QMIX recovers the correct maximal action for any true-action value. However, in a practical scenario, it is impossible to separate whether this true target action-value is high by the actions of the agents or high by the randomness of the reward. Due to this integrated risk-seeking policy, Weighted QMIX cannot isolate only agent-wise risk and shows lower performance than QMIX in many scenarios (Rashid et al., 2020a). We provide a more detailed description of other CTDE algorithms in Appendix C.

2.2 DISTRIBUTIONAL REINFORCEMENT LEARNING

Instead of training a scalar state-action estimator $Q^\pi(s,u)$, distributional reinforcement learning (RL) represents an action-value as a random variable, denoted $Z^\pi(s,u)$. Then the distributional Bellman operator for policy evaluation in single-agent RL can be expressed as follows:

$$Z^\pi(s,u) \overset{D}{=} R(s,u) + \gamma Z^\pi(S',U'), \tag{4}$$

where $S' \sim P(s'|s,u)$, $U' \sim \pi(u|s)$, and $A \overset{D}{=} B$ denotes that two random variables follow the same probability distribution.

IQN. Implicit Quantile Networks (IQN) (Dabney et al., 2018a) is one of the single-agent widely-used distributional RL methods. IQN approximates the true action-value function using a quantile-based representation of distributions. To be specific, the trained quantiles represent a random variable $Z(s,u,\omega)$ which is a reparameterization of risk-level samples ω drawn from a uniform distribution $\mathcal{U}(0,1)$. To train the IQN, one can use the following Huber quantile regression loss (Huber, 1964):

$$\begin{aligned}
\mathcal{L}_{\texttt{td-IQN}} &= \begin{cases} |\omega - \mathbb{1}_{\delta>0}| \times \frac{1}{2}\delta^2, & \text{if } |\delta| \leq 1, \\ |\omega - \mathbb{1}_{\delta>0}| \times (|\delta| - \frac{1}{2}), & \text{otherwise,} \end{cases} \\
\delta &= Z(s,u,\omega) - (r + \gamma Z^{\texttt{target}}(s', u'_{\texttt{opt}}, \omega')), \\
u'_{\texttt{opt}} &= \arg\max_{u'} \mathbb{E}_w[Z^{\texttt{target}}(s,u',w)],
\end{aligned}$$

where $Z^{\texttt{target}}$ is the target network, which periodically updates parameters from the original action-value estimator Z. Furthermore, (s,u,r,s') is a tuple of objects collected from consecutive time steps in the Markov decision process.

Distributional MARL. Recently, distributional RL has also been applied to the CTDE regime (Qiu et al., 2021; Sun et al., 2021), but they lack the disentanglement of risk sources. RMIX (Qiu et al., 2021) and DFAC (Sun et al., 2021) showed promising results by extending the agent-wise utility function from deterministic variables to random variables. RMIX demonstrates the effectiveness of risk-sensitive MARL via distributional RL, but it is limited since they cannot represent policies, which have different risk levels relying on sources (i.e., agent-wise risk-seeking yet environment-wise risk-averse policies). DFAC is another distributional MARL framework with proposed mean-shape decomposition while employing IQN network for agent-wise utility function, but they only showcase risk-neutral policies. Da Silva et al. (2019) and Lyu & Amato (2018) have applied distributional learning in fully decentralized cases, not CTDE methods.

3 DRIMA: DISENTANGLED RISK-SENSITIVE MARL

In this section, we propose a new method called, DRIMA, aiming at decentralizing the true action-value estimator with disentangled risk. Our key idea is two types of action-value estimators with different roles: (i) true action-value estimator $Z_{\texttt{jt}}(w_{\texttt{env}})$ for estimating the true action-value distribution while capturing environment-wise randomness with environment-wise risk level $w_{\texttt{env}}$, (ii) transformed action-value estimator $Z_{\texttt{tran}}(w_{\texttt{agt}})$ to transform the true action-value estimator into the space of estimators that are decentralized with respect to the utility functions while considering agent-wise randomness with agent-wise risk level $w_{\texttt{agt}}$. In section 3.1, we propose a training objective of DRIMA, which trains our action-value estimators. Next, in section 3.2, we introduce a new architecture for the action-value estimators. The overall architectural sketch is given in Figure 3.

3.1 TRAINING OBJECTIVES

The training objective of DRIMA is twofold. First, DRIMA trains the true action-value estimator for approximating the distributional true action-value estimator $Z_{\texttt{jt}}$ based on the distributional Bellman operator (Equation 4). Next, the transformed action-value estimators attempt to imitate the behavior of the true action-value estimator with weightings. Notably, the utility functions decentralize the transformed action-value estimator by construction, and they can approximately decentralize the true action-value estimator when the second objective is sufficiently optimized.

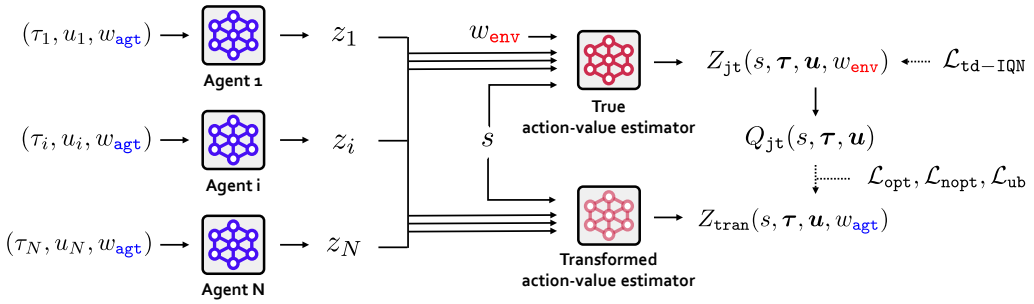


Figure 3: Architecture of DRIMA. Each agent network and true action-value estimator are built based on IQN and input a quantile of agent-wise risk w_{agt} and environment-wise risk w_{env}, respectively. Along with the hierarchical quantile structure and quantile regression, the architecture is capable of disentangling risk sources explicitly.

Loss for the true action-value estimator. To be specific, true action-value estimator Z_{jt} is trained with the distributional Bellman operator (Equation 4) and the Huber quantile loss (Huber, 1964):

$$\mathcal{L}_{\text{td-IQN}} = \begin{cases} |w_{\text{env}} - \mathbb{1}_{\delta>0}| \times \frac{1}{2}\delta^2, & \text{if } |\delta| \leq 1, \\ |w_{\text{env}} - \mathbb{1}_{\delta>0}| \times (|\delta| - \frac{1}{2}), & \text{otherwise,} \end{cases}$$
$$\delta = Z_{\text{jt}}(s, \boldsymbol{\tau}, \boldsymbol{u}, w_{\text{env}}) - (r + \gamma Z_{\text{jt}}^{\text{target}}(s', \boldsymbol{\tau}', \boldsymbol{u}'_{\text{opt}}, w'_{\text{env}})),$$

where $Z_{\text{jt}}^{\text{target}}$ is the target network that periodically updates parameters from the original estimator Z_{jt}. Furthermore, $\boldsymbol{u}'_{\text{opt}}$ is the set of actions maximizing the utility functions $z_i(\tau'_i, u'_i)$ for $i \in \mathcal{N}$. For the $\mathcal{L}_{\text{td-IQN}}$, risk-level samples $w_{\text{env}}, w'_{\text{env}}$ drawn from a uniform distribution $\mathcal{U}(0, 1)$.

Loss for the transformed action-value estimator. Next, we demonstrate the training objectives to decentralize the true action-value estimator. To this end, for each sample $(s, \boldsymbol{\tau}, \boldsymbol{u})$ from the replay buffer, we train transformed action-value estimator using loss functions $\mathcal{L}_{\text{opt}}$, $\mathcal{L}_{\text{opt}}$, and $\mathcal{L}_{\text{ub}}$ expressed as follows, where we omit their common function arguments $(s, \boldsymbol{\tau})$ for presentational simplicity:

$$\mathcal{L}_{\text{opt}} = \left(Z_{\text{tran}}(\boldsymbol{u}_{\text{opt}}, w_{\text{agt}}) - Q_{\text{jt}}(\boldsymbol{u}_{\text{opt}})\right)^2,$$
$$\mathcal{L}_{\text{nopt}} = \begin{cases} (Z_{\text{tran}}(\boldsymbol{u}, w_{\text{agt}}) - Q_{\text{jt}}(\boldsymbol{u}))^2, & \text{if } Z_{\text{tran}}(\boldsymbol{u}, w_{\text{agt}}) \leq Q_{\text{jt}}(\boldsymbol{u}), \\ 2(1 - w_{\text{agt}}) \times (Z_{\text{tran}}(\boldsymbol{u}, w_{\text{agt}}) - Q_{\text{jt}}(\boldsymbol{u}))^2, & \text{otherwise,} \end{cases}$$
$$\mathcal{L}_{\text{ub}} = \begin{cases} (Z_{\text{tran}}(\boldsymbol{u}, w_{\text{agt}}) - Q_{\text{ub}}(\boldsymbol{u}))^2, & \text{if } Z_{\text{tran}}(\boldsymbol{u}, w_{\text{agt}}) \geq Q_{\text{ub}}(\boldsymbol{u}), \\ 0, & \text{otherwise,} \end{cases}$$
$$Q_{\text{jt}}(\boldsymbol{u}) = \mathbb{E}_{w_{\text{env}}}[Z_{\text{jt}}(\boldsymbol{u}, w_{\text{env}})], \quad Q_{\text{ub}}(\boldsymbol{u}) = \max(Q_{\text{jt}}(\boldsymbol{u}), Q_{\text{jt}}(\boldsymbol{u}_{\text{opt}})),$$

where $\boldsymbol{u}_{\text{opt}}$ is an "optimal" action maximizing the utility functions $z(w_{\text{agt}})_i$ for $i \in \mathcal{N}$.

Conceptually, the agent-wise risk level w_{agt} determines how optimistic agents will behave. First, $\mathcal{L}_{\text{opt}}$ means that the transformed action-value estimator exactly follows the true action-value estimator for the most optimistic sample, in which all agents choose their optimal action.

Then, for non-optimal actions, the transformed action-value estimator, which has lower representational power, efficiently tracks the true action-value estimator depending on optimism determined by weights. If the value of true action-value estimator Q_{jt} is greater than the value of the transformed action-value estimator Z_{tran}, then this is an optimistic sample where the corresponding action is likely to be the optimal action, so the transformed action-value estimator exactly follows the true action-value estimator. Conversely, for a relatively small true action-value, we softly ignore and follow the true action-value estimator through a small weight depending on the agent-wise risk level w_{agt}. In the most extreme case, $w_{\text{agt}} = 1$, our loss functions $\mathcal{L}_{\text{opt}}$, and $\mathcal{L}_{\text{nopt}}$ become similar to QTRAN (Son et al., 2019). If the agent-wise risk level w_{agt} decreases to 0.5, both $\mathcal{L}_{\text{nopt}}$ and $\mathcal{L}_{\text{opt}}$ follow the true action-value without weighting, which is agent-wise risk-neutral.

As above, changing the weight according to the sign of the error shares a key idea with distributional reinforcement learning (Dabney et al., 2018a;b) and Weighted QMIX (Rashid et al., 2020a), but DRIMA has stronger a capacity. In distributional reinforcement learning, when Bellman TD-error

is positive, it is treated as an optimistic sample for *environment-wise* risk. On the other hand, in Weighted QMIX, when TD-error is positive, the sample is used as an optimistic sample for *agent-wise* risk. However, since reward and next state in TD-target include both agent-wise randomness and environment-wise randomness, agent-wise risk cannot be extracted enough from the sign of TD-error. Therefore, instead of TD-target, we use the expected value of the true action-value estimator $Z_{\texttt{jt}}$ with agent-wise randomness removed. Thanks to the disentanglement, DRIMA is able to obtain desirable policies more efficiently, while, as aforementioned, it is difficult to set agent-wise risk-seeking yet environment-wise risk-averse in prior-arts.

We can adjust the environment-wise risk sensitivity according to how we calculate expected value $Q_{\texttt{jt}}$ from the distributional true action-value estimator $Z_{\texttt{jt}}$. If the expectation is calculated by sampling $w_{\texttt{env}}$ through a uniform distribution $\mathcal{U}(0,1)$, the agents learn a risk-neutral policy for the environment-wise randomness. By changing this sampling distribution, we can learn the optimal policy for the desired environment-wise risk-sensitivity. For example, if we learn $Q_{\texttt{tran}}$ for $Q_{\texttt{jt}}$ calculated by sampling $w_{\texttt{env}}$ for uniform distribution $\mathcal{U}(0,0.25)$, agents take environment-wise risk-averse behaviors.

Finally, we add a loss function $\mathcal{L}_{\texttt{ub}}$, which makes an upper bound condition of $Z_{\texttt{tran}}$ for numerical stability because only the lower bound of $Z_{\texttt{tran}}$ exists with the $\mathcal{L}_{\texttt{nopt}}$ when $w_{\texttt{agt}} = 1$. To combine our loss functions, we obtain the following objective, which is minimized in an end-to-end manner to train the true action-value estimator and the transformed action-value estimator:

$$\mathcal{L} = \mathcal{L}_{\texttt{td-IQN}} + \lambda_{\texttt{opt}}\mathcal{L}_{\texttt{opt}} + \lambda_{\texttt{nopt}}\mathcal{L}_{\texttt{nopt}} + \lambda_{\texttt{ub}}\mathcal{L}_{\texttt{ub}}$$

where $\lambda_{\texttt{opt}}, \lambda_{\texttt{nopt}}, \lambda_{\texttt{ub}} > 0$ are hyperparameters controlling the importance of each loss function. Note that agent-wise utility functions z and true action-value estimator $Q_{\texttt{jt}}$ are learned through all loss functions. We also tried not to train the true action-value estimator for the losses for the transformed action-value estimator, but learning it together had high learning stability in practical scenarios. The training algorithm of DRIMA is provided in Appendix A.

3.2 Network architectures

Here, we introduce our architectures for the action-value estimators. First, we construct the estimators using the utility functions $z_1(w_{\texttt{agt}}), \ldots, z_N(w_{\texttt{agt}})$ with agent-wise risk level $w_{\texttt{agt}}$. Our main contribution in designing the estimators is twofold: (i) using IQN (Dabney et al., 2018a) for the true action-value estimator $Z_{\texttt{jt}}(w_{\texttt{env}})$ with environment-wise risk level $w_{\texttt{env}}$ and (ii) using a monotonic mixing network for the transformed action-value estimators $Z_{\texttt{tran}}$ with agent-wise risk level $w_{\texttt{agt}}$.

Agent-wise utility function z_i. In a partially observable setting, agents can learn better policies by using action-observation history τ_i instead of current observation. We represent agent-wise utility function as a deep recurrent Q-network (Hausknecht & Stone, 2015) that receives action-observation history as input and outputs z_i for each action and $w_{\texttt{agt}}$.

True action-value estimator $Z_{\texttt{jt}}$. From the monotonic mixing network with a limited representation of QMIX (Rashid et al., 2018), we utilize an additional network so that a true action-value estimator can express distributions with additional representation power. The mixing network for true action-value estimator utilizes a feed-forward network that takes the states s and set of utility functions z_i for $i \in \mathcal{N}$. Since we use z averaged over multiple $w_{\texttt{agt}}$ samples, the feed forward network is not conditioned on $w_{\texttt{agt}}$. To apply IQN for the true action-value estimator, we use an additional network that computes an embedding $\phi(w_{\texttt{env}})$ for the sample point $w_{\texttt{env}}$. We calculate the embedding of $w_{\texttt{env}}$ with cosine basis functions and utilize element-wise (Hamard) product, as in IQN paper.

Transformed action-value estimator $Z_{\texttt{tran}}$. Now we introduce the monotonic mixing network used for the transformed action-value estimator $Z_{\texttt{tran}}(w_{\texttt{agt}})$. The architecture of the transformed action-value estimator is largely the same as the mixing network of Rashid et al. (2018). The monotonic mixing function is parameterized by non-negative parameters obtained from state-dependent hypernetwork (Ha et al., 2017), and the transformed action-value estimator is expressed as follows:

$$Z_{\texttt{tran}}(s, \boldsymbol{\tau}, \boldsymbol{u}, w_{\texttt{agt}}) = f_{\texttt{mix}}\big(z_1(\tau_1, u_1, w_{\texttt{agt}}), \ldots, z_N(\tau_N, u_N, w_{\texttt{agt}}); \theta_{\texttt{tran}}(s, w_{\texttt{agt}})\big),$$

where $\theta_{\texttt{tran}}(s, w_{\texttt{agt}})$ is non-negative parameters obtained from state-dependent hypernetworks. A more detailed discussion is available in Appendix D.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Environments. We mainly evaluate our method on the Starcraft Multi-Agent Challenge (SMAC) environment (Samvelyan et al., 2019). In this environment, each agent is a unit participating in combat against enemy units controlled by handcrafted policies. In the environment, agents receive individual local observation containing distance, relative location, health, shield, and unit type of other allied and enemy units within their sight range. The SMAC environment additionally assumes that a global state is available during the training of the agents. To be specific, the global state contains information on all agents participating in the scenario. We select four representative scenarios which are diversified with respect to (i) desirable tactics and (ii) level of difficulty: 5m_vs_6m, 3s_vs_5z, 8m_vs_9m, and MMM2. For (i), in 5m_vs_6m and 8m_vs_9m, focusing fire is a plausible strategy; meanwhile, kiting is fruitful for 3s_vs_5z. For (ii), we select MMM2 since it has heterogeneous agents, which are more difficult to learn handling them than homogeneous ones. Appendix D contains additional experimental details.

Increased exploration and rewarding mechanisms. To further verify the effectiveness of disentangling risks, we additionally consider more difficult types of scenarios with differing exploration schedules and distinct mechanisms of reward functions. We refer to these three different types of scenarios as v1, v2 (increased exploration), and v3 (increased exploration and negative rewards), respectively. Details are as follows:

- v1: This is the same environment, which is used in Rashid et al. (2018). To be specific, ϵ-greedy policy is annealed from 1 to 0.05 over $50k$ time steps. The reward is determined only by the enemy unit's health condition, i.e., units receive reward only when the enemy unit is damaged.
- v2: This is a modified version from v1, which changes the exploration annealing schedule to "$500k$" time steps following setups in Rashid et al. (2020a). By increasing exploration, one can expect that agents are less likely to behave optimal actions in training time, where handling randomness becomes more important.
- v3: This is a modified version from v2, which changes reward shaping; v3 contains "negative" rewards as well as positive ones. To be specific, the reward is determined not only by the enemy units' health condition but also by damages dealt by our agents. In this environment, it is critical to consider agent-wise risks explicitly since it is likely to take "locally optimal" policy (i.e., selfish behavior such as running away) otherwise.

Interestingly, in increased exploration (v2 and v3), it is likely to obtain more non-cooperative behaviors from teammates, which represents the importance of considering agent-wise risk explicitly. In the existence of negative rewards (v3), if risks are not disentangled well, units are induced to have a "selfish" behavior (e.g., running away) in order to avoid being damaged.

Evaluation. We run 32 test episodes without an exploration factor for every $4 * 10^4$-th time step. The percentage of episodes where the agents defeat all enemy units, i.e., *test win rate*, is reported as the performance metric of the algorithms. We report the median performance with shaded 25-75% confidence intervals with four random seeds. For visual clarify, we smooth all the curves by moving average filter with a window size of 4.

4.2 RESULTS

Comparisons to baselines. We evaluate DRIMA compared to the following eight baselines, including non-distributional (i.e., VDN, QMIX, QTRAN, QPLEX, OW-QMIX, and CW-QMIX) and distributional CTDE methods (i.e., DDN and DMIX). We provide brief descriptions of the baselines as follows. VDN (Sunehag et al., 2018) factorizes joint action-value estimator into a linear summation over the utility functions. QMIX (Rashid et al., 2018) extends VDN by employing a *mixing network* to express a non-linear monotonic relationship among individual agent-wise utility functions in the joint action-value estimator. QTRAN (Son et al., 2019) trains a decentralizable *transformed* action-value estimator to follow the optimal behavior of a *true* action-value estimator, using soft regularizations. QPLEX (Wang et al., 2020a) takes a duplex dueling network architecture to factorize the joint value function. OW-QMIX and CW-QMIX (Rashid et al., 2020a) use a weighted projection that allows

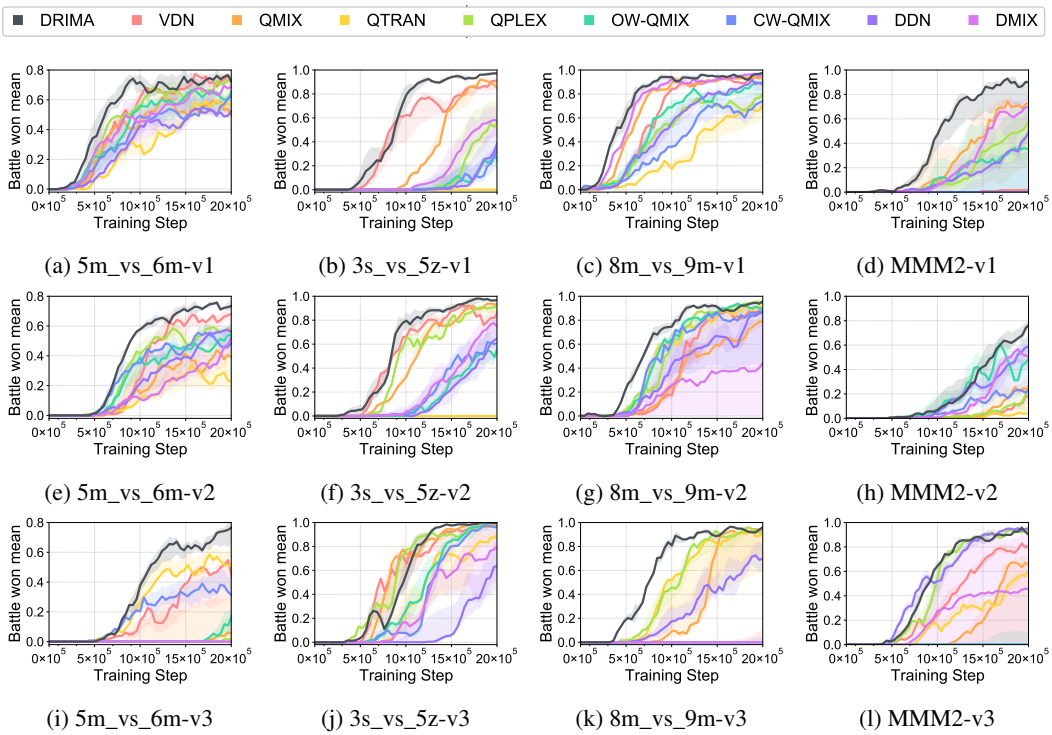


Figure 4: Median test win rate with 25%-75% percentile over four random seeds, comparing DRIMA with eight baselines.

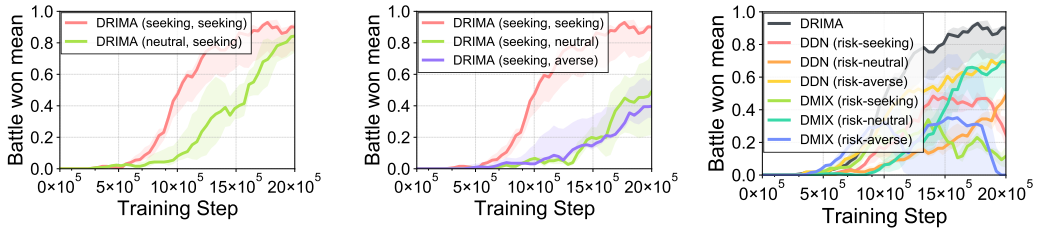
more emphasis to be placed on better joint actions with built on QMIX. DDN and DMIX (Sun et al., 2021) extend VDN and QMIX by employing distributional RL with *mean-shape decomposition*, respectively. We provide further details for each baseline in Appendix C.

In Figure 2, we present the aggregated result, which is the median test win rate averaged across all 12 scenarios (i.e., 4 Maps with 3 different versions) in our experiments. Along with Figure 4, which shows performance on each scenario, one can find that DRIMA achieves the state-of-the-art performance both sample-efficiently and asymptotically, while the second-best method varies for each scenario. For relatively easy tasks such as 5m_vs_6m, 3s_vs_5z, and 8m_vs_9m, we set (agent-wise risk-sensitivity, environment-wise risk-sensitivity) to (seeking, neutral). For a hard task like MMM2, we set to (seeking, seeking). The rationale of this setting is that for easy tasks, considering environment-wise risks do not crucially affect to find optimal strategy, but for a hard task, it is much beneficial to take environmental risks, which makes extensive exploration; environment-wise risk-averse behavior is likely to avoid dangerous actions, which makes less explorative.

Notably, as shown in Figure 4e-4l, DRIMA obtains significant gain in v2 (increased exploration) and v3 (increased exploration and negative rewards), where separating agent-wise risk and environment-wise risk is critical; it is likely to reach suboptimal equilibria (i.e., selecting actions of running away) if risk sources are not disentangled in such environments. These results indeed highlight empirical effectiveness via our proposed scheme of disentangling risk sources.

Moreover, DRIMA shows consistent strength regardless of difficulty and desirable tactics. Intriguingly, in relatively easier tasks (i.e., 5m_vs_6m), simple methods such as VDN show powerful results than more complex ones (i.e., DDN and DMIX[2]). We understand that a simple structure is more efficient to learn plausible strategies in such easy tasks. However, the reason why DRIMA is able to learn relatively easy tasks despite its sophisticated structure comes from that disentanglement of risk sources makes useful learning signals for conquering easy tasks efficiently.

[2]In the original paper of DDN and DMIX (Sun et al., 2021), the results of 5m_vs_6m, 3s_vs_5z, and 8m_vs_9m were not reported.

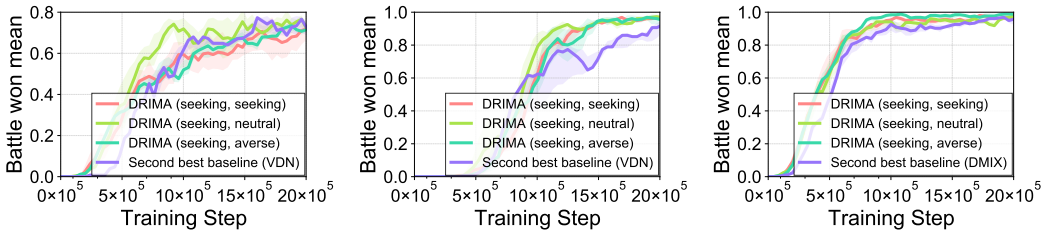


(a) agent-wise risk

(b) environment-wise risk

Figure 5: Ablation studies on varying risk-levels (agent-wise risk $w_{\texttt{agt}}$, environment-wise risk $w_{\texttt{env}}$) of DRIMA in MMM2-v1, one of *SUPER HARD* tasks.

Figure 6: Comparison of DRIMA to risk-sensitive variants of DDN and DMIX in MMM2-v1.



(a) 5m_vs_6m-v1

(b) 3s_vs_5z-v1

(c) 8m_vs_9m-v1

Figure 7: Robustness of DRIMA in relatively easy tasks. Differing risk-levels (agent-wise risk $w_{\texttt{agt}}$, environment-wise risk $w_{\texttt{env}}$) in DRIMA do not degrade below the second-best baseline.

DRIMA with varying risk-levels. To investigate the effect of different risk levels in DRIMA, we conduct experiments with adjusting agent-wise risk and environment-wise risk, respectively. As shown in Figure 5, we find that leveraging risk levels with disentanglement indeed affect performance in a hard task (i.e., MMM2). Interestingly, one can note that agent-wise risk-seeking and environment-wise risk-seeking shows the strongest performance. We understand that it is critical to strongly believe teammates act cooperatively (i.e., agent-wise risk-seeking), and enhance exploration through an environment-wise risk-seeking objective for finding the optimal tactic in such a hard task.

Comparisons to risk-senstive variants of DDN and DMIX. To further understand the effectiveness of disentangling risk sources, we compare DRIMA and risk-sensitive variants of DDN and DMIX in Figure 6. Although DFAC proposes only risk-neutral policies in their original paper, we additionally implement their risk-sensitive variants by adjusting a quantile-sampling range, i.e., sampling quantiles in $[0, 0.25]$ for risk-averse policy and $[0.75, 1]$ for risk-seeking policy. As shown in Figure 6, DRIMA also achieves superior performance over risk-sensitive variants of DFAC. We understand that the gain mainly comes from the ability to explicitly separating the two risk sources.

Robustness of DRIMA. In Figure 7, in order to show the robustness of DRIMA, we report performance over the second-best baseline across relatively easy tasks: 5m_vs_6m-v1, 3s_vs_5z-v1, and 8m_vs_9m-v1. We note that changing environmental-wise risks do not affect to beat second-best baseline. It shows not only an empirical strength of DRIMA, but also one interesting finding; environmental-risk does not matter to find desirable tactic in such easy tasks. However, note that in 8m_vs_9m-v1, only environment-wise risk-averse agents reached a 100% win rate, although other agents also achieve high win rates. This means that a safe policy can be beneficial for easy tasks.

5 Conclusion

In this paper, we present DRIMA, a new distributional MARL framework that disentangles agent-wise risks and environment-wise risks in order to obtain optimal policies efficiently. Our main idea is to make divided risk-level leverages (i.e., quantiles) along with the proposed hierarchical quantile structure and quantile regression. Through extensive experiments, DRIMA achieves state-of-the-art performance thanks to the disentanglement of risk sources. We believe that DRIMA would guide a new interesting direction in the MARL community.

ETHICS STATEMENT

Cooperative MARL has potential real-world applications such as robot swarms (Huttenrauch et al., 2017), autonomous driving (Shalev-Shwartz et al., 2016), and sensor networks (Zhang & Lesser, 2011). However, there could be possible negative impacts if a malicious user designs a reward shaping that harms humans and society. For this reason, it is important to deeply consider the consequences of the agents' behaviors before releasing developed algorithms in the real world.

We have proposed a method that applies distributional reinforcement learning to CTDE algorithms. Our methods require additional research in terms of security. When a malicious external agent is added, there is a risk that our system is easily broken. Therefore, learning robust multi-agent systems against adversarial agents is an interesting future direction of research.

REPRODUCIBILITY STATEMENT

One can find our reproducible code in the supplementary materials. All the used packages are along with the code. We remark that codebases of DRIMA and baselines including VDN, QMIX, QTRAN, OW-QMIX, CW-QMIX, DDN, and DMIX are based on the GitHub code of PyMARL[3]. As an evaluation benchmark, we employ StarCraft Multi-Agent Challenge (SMAC)[4] of which StarCraft version is SC2.4.6.2.69232. In our experiments, we use a single GPU (NVIDIA TITAN Xp) and 8 CPU cores (Intel Xeon E5-2630 v4).

REFERENCES

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of ICML*, pp. 449–458. PMLR, 2017.

Yinlam Chow and Mohammad Ghavamzadeh. Algorithms for cvar optimization in mdps. In *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, pp. 3509–3517, 2014.

Felipe Leno Da Silva, Anna Helena Reali Costa, and Peter Stone. Distributional reinforcement learning applied to robot soccer simulation. In *Adapative and Learning Agents Workshop, AAMAS*, 2019.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of ICML*, pp. 1096–1105. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

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

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

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

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

Peter J Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, pp. 73–101, 1964.

Maximilian Huttenrauch, Adrian Sosic, and Gerhard Neumann. Learning complex swarm behaviors by exploiting local communication protocols with deep reinforcement learning. *arxiv preprint arXiv:1709.07224*, 2017.

[3]https://github.com/oxwhirl/pymarl

[4]https://github.com/oxwhirl/smac

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

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

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Abbeel Pieter, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of NeurIPS*, pp. 6379–6390, 2017.

Xueguang Lyu and Christopher Amato. Likelihood quantile networks for coordinating multi-agent reinforcement learning. *arXiv preprint arXiv:1812.06319*, 2018.

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

Wei Qiu, Xinrun Wang, Runsheng Yu, Xu He, Rundong Wang, Bo An, Svetlana Obraztsova, and Zinovi Rabinovich. Rmix: Learning risk-sensitive policies for cooperative reinforcement learning agents. *arXiv preprint arXiv:2102.08159*, 2021.

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

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation. *arXiv preprint arXiv:2006.10800*, 2020a.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:2003.08839*, 2020b.

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

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

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

Wei-Fang Sun, Cheng-Kuang Lee, and Chun-Yi Lee. Dfac framework: Factorizing the value function via quantile mixture for multi-agent distributional q-learning. In *Proceedings of ICML*, 2021.

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 based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2085–2087, 2018.

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. 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*, 2021.

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.

Chongjie Zhang and Victor Lesser. Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

A DRIMA TRAINING ALGORITHM

The training algorithm for DRIMA is provided in Algorithm 1

Algorithm 1 DRIMA algorithm

Initialize replay memory $\mathcal{B} \leftarrow \emptyset$ and target parameters $\theta^- \leftarrow \theta$
for episode = 1 to M **do**
  Observe initial state $\boldsymbol{s}^0$ and observation $\boldsymbol{o}^0 = [O(\boldsymbol{s}^0, i)]_{i=1}^N$ for each agent i
  for $t = 1$ to T **do**
    With probability ϵ, each agent i select an random action u_i^t
    Otherwise, set $u_i^t = \arg\max_{u_i^t} z_i(\tau_i^t, u_i^t, w_{\texttt{agt}})$ for each agent i based on our agent-wise risk-sensitivity $w_{\texttt{agt}}$.
    Take action $\boldsymbol{u}^t$, and retrieve next state, observation and reward $(s^{t+1}, \boldsymbol{o}^{t+1}, r^t)$
    Store transition $(s^t, \boldsymbol{\tau}^{t+1}, \boldsymbol{u}^t, r^t, s^{t+1}, \boldsymbol{\tau}^{t+1})$ in $\mathcal{B}$
    Sample a transition $(s, \boldsymbol{\tau}, \boldsymbol{u}, r, s', \boldsymbol{\tau}')$ from $\mathcal{B}$
    Sample environment-wise risk levels $w_{\texttt{env}}, w'_{\texttt{env}}$ uniformly from $\mathcal{U}(0, 1)$
    Compute loss $\mathcal{L}_{\texttt{td-IQN}}$ for true action-value estimator from Section 3.1
    Sample environment-wise risk levels $w_{\texttt{env}}$ based on our environment-wise risk-sensitivity
    Sample agent-wise risk levels $w_{\texttt{agt}}$ uniformly
    Compute loss $\mathcal{L}_{\texttt{opt}}, \mathcal{L}_{\texttt{nopt}}$, and $\mathcal{L}_{\texttt{ub}}$ for transformed action-value estimator from Section 3.1
    Update θ by minimizing the losses
    Update target network parameters $\theta^- \leftarrow \theta$ with period I
  end for
end for

B DETAILED ARCHITECTURAL ILLUSTRATION

In this section, we provide detailed illustration for our architectural components: (i) agent-wise utility function, (ii) true action-value estimator, and (iii) transformed action-value estimator.

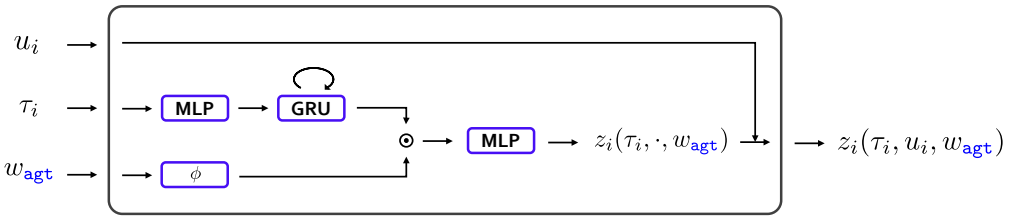


Figure 8: Agent-wise utility function. $\phi : [0, 1] \to \mathbb{R}^d$ denotes a quantile embedding function (Dabney et al., 2018a)

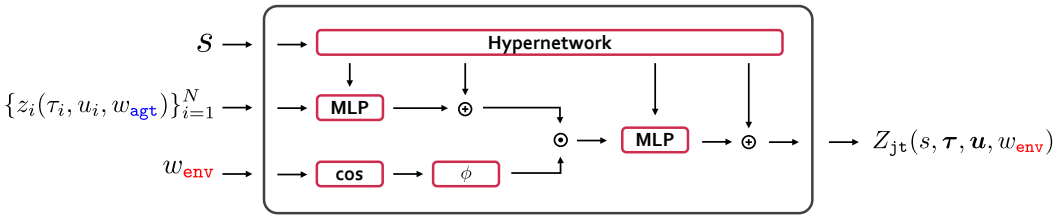


Figure 9: True action-value estimator. $\phi : [0, 1] \to \mathbb{R}^d$ denotes a quantile embedding function (Dabney et al., 2018a).

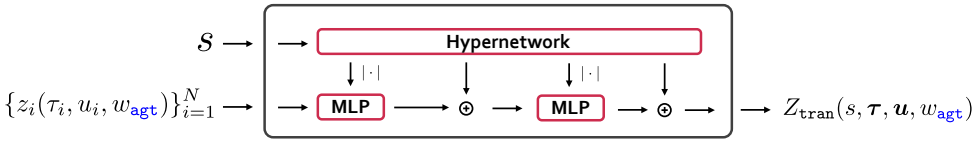


Figure 10: Transformed action-value estimator. $|\cdot|$ is employed to enforce monotonicity constraint (Rashid et al., 2018).

C RELATED WORK

C.1 CENTRALIZED TRAINING WITH DECENTRALIZED EXECUTION

Centralized training with decentralized execution (CTDE) has emerged as a popular paradigm under the multi-agent reinforcement learning framework. It assumes the complete state information to be fully accessible during training, while individual policies allow decentralization during execution. To train agents under the CTDE paradigm, both policy-based (Foerster et al., 2018; Lowe et al., 2017; Du et al., 2019; Iqbal & Sha, 2019; Wang et al., 2020b; 2021) and value-based methods (Sunehag et al., 2018; Rashid et al., 2018; Son et al., 2019; Yang et al., 2020; Sun et al., 2021) have been proposed. At a high level, the policy-based methods rely on the actor-critic framework with independent actors to achieve decentralized execution. On the other hand, the value-based methods attempt to learn a joint action-value estimator, which can be cleverly decomposed into individual agent-wise utility functions.

Among the policy-based methods, COMA (Foerster et al., 2018) trains individual policies with a joint critic and solves the credit assignment problem by estimating a counterfactual baseline. MADDPG (Lowe et al., 2017) extends the DDPG (Lillicrap et al., 2015) algorithm to learn individual policies in a centralized manner on both cooperative and competitive games. MAAC (Iqbal & Sha, 2019) includes an attention mechanism in critics to improve scalability. LIIR (Du et al., 2019) introduces a meta-gradient algorithm to learn individual intrinsic rewards to solve the credit assignment problem. ROMA (Wang et al., 2020b) proposes a role-oriented framework to learn roles via deep RL with regularizers and role-conditioned policies. Finally, DOP (Wang et al., 2021) proposes factorized critic for multi-agent policy gradients.

Among the value-based methods, value-decomposition networks (VDN, Sunehag et al. 2018) learns a centralized yet factored joint action-value estimator by representing the joint action-value estimator as a sum of individual agent-wise utility functions. QMIX (Rashid et al., 2018) extends VDN by employing a *mixing network* to express a non-linear monotonic relationship among individual agent-wise utility functions in the joint action-value estimator. Qatten (Yang et al., 2020) introduces a multi-head attention mechanism for approximating the decomposition of the joint action-value estimator, which is based on theoretical findings. However, our method satisfies both theoretical guarantees and practical performance.

QTRAN (Son et al., 2019) has been proposed to eliminate the monotonic assumption on the joint action-value estimator in QMIX (Rashid et al., 2018). Instead of directly decomposing the joint action-value estimator into utility functions, QTRAN proposes a training objective that enforces the decentralization of the joint action-value estimator into the summation of individual utility functions. However, recent studies have found that despite its promise, QTRAN performs empirically worse than QMIX in complex MARL environments (Samvelyan et al., 2019; Rashid et al., 2020b).

Recently, several other methods have been proposed to solve the limitations of QMIX. QPLEX (Wang et al., 2020a) takes a duplex dueling network architecture to factorize the joint value function. Unlike QTRAN, QPLEX learns using only one joint action-value network and a single loss function. Since they learn only a single estimator, it is impossible to learn agent-wise randomness and environment-wise randomness separately by applying distributional reinforcement learning as in our method. Also, Rashid et al. (2020a) proposed CW-QMIX and OW-QMIX, which use a weighted projection that allows more emphasis to be placed on better joint actions. Their methods apply weights to loss according to the sign of the TD-error for each sample, and it is theoretically guaranteed that the optimal policy can be learned for decentralizable tasks. However, whereas DRIMA learns the randomness of the environment using the weights from the TD-error, CW-QMIX and OW-QMIX only aim to consider the randomness of chosen actions. For this reason, they do not consider the distribution of the TD-error caused by the transition probability or the randomness of the reward in training, and it is not guaranteed that they can learn the optimal policy in stochastic environments. On the other hand, the true action-value estimator of DRIMA learns only the randomness of the environment, and the transformed action-value estimator learns through the estimated true action-value, not the TD-target. Therefore, DRIMA completely separates the randomness of the environment and the randomness of the action in training.

C.2 DISTRIBUTIONAL REINFORCEMENT LEARNING

Instead of training a scalar state-action estimator, distributional reinforcement learning focuses on representing a distribution over returns. Empirically, distributional reinforcement learning improves sample efficiency and performance.

A number of distributional reinforcement learning methods are proposed in a single-agent domain. Bellemare et al. (2017) proposed C51, which parameterizes the return distribution as a categorical distribution over a fixed set of equidistant points. In Dabney et al. (2018b), QR-DQN is proposed to estimate the distributions with a uniform mixture of N diracs and quantile regression. By estimating the quantile function with quantile regression, which has been shown to converge to the true quantile function, QR-DQN minimizes the Wasserstein distance to the distributional target. This estimation has been shown to converge to the true quantile function.

Implicit quantile Networks (IQN) (Dabney et al., 2018b) provides a way to learn an implicit representation of distributions by reparameterizing samples from base samples, typically $\omega \sim \mathcal{U}(0, 1)$. It learns a quantile function that maps from embeddings of sample probabilities to the corresponding quantiles, called implicit quantile networks. The quantiles are trained by Huber quantile regression loss (Huber, 1964).

Recently, distributional RL has also been applied to CTDE regime (Qiu et al., 2021; Sun et al., 2021), but they lack disentanglement of risk sources. RMIX (Qiu et al., 2021) and DFAC (Sun et al., 2021) showed promising results by extending agent-wise utility functions from deterministic variables to random variables. RMIX demonstrates the effectiveness of risk-sensitive MARL via distributional RL, but it is limited since they cannot represent policies, which have different risk levels relying on sources (i.e., agent-wise risk-seeking yet environment-wise risk-averse policies). DFAC is another distributional MARL framework with proposed mean-shape decomposition while employing IQN network for agent-wise utility function, but they only showcase risk-neutral policies. Da Silva et al. (2019) and Lyu & Amato (2018) have applied distributional learning in fully decentralized cases, not CTDE methods.

D EXPERIMENTAL DETAILS

The hyperparameters of training and testing configurations for VDN, QMIX, and QTRAN are the same as in the recent GitHub code of SMAC [5] (Samvelyan et al., 2019) and PyMARL [6] with StarCraft version SC2.4.6.2.69232 The architecture of all agents' policy networks is a deep recurrent Q-network (Hausknecht & Stone, 2015) consisting of two 64-dimensional fully connected layers and one 64-dimensional GRU. The mixing networks consist of a single hidden layer with 32 hidden widths and ELU activation functions. Hypernetworks consists of two layers with 64 hidden widths and ReLU activation functions. Also, the hyperparameters of WQMIX [7] (Rashid et al., 2020a) and QPLEX [8] (Wang et al., 2020a) are the same as in their GitHub codes. However, unlike they used a specific configuration for some environments, such as adding non-linearity for MMM2, we used the same setting for all environments. Finally, the hyperparameters of DFAC [9] (Sun et al., 2021) are the same as in their GitHub code. As in their paper, we use 256 hidden layer sizes for DMIX (Sun et al., 2021) in the MMM2 environment and 512 hidden layer sizes for DDN (Sun et al., 2021) in the MMM2 environment. For other scenarios, we fix all the hidden layer sizes as 64 for a fair comparison. There may be slight differences compared to their paper due to differences in StarCraft version.

Like the DFAC paper (Sun et al., 2021), following the optimizer of the IQN (Dabney et al., 2018a), we used the Adam optimizer. For other methods except for DRIMA and DFAC, according to their papers, all neural networks are trained using the RMSProp optimizer with a 0.0005 learning rate. We use ϵ-greedy action selection with decreasing ϵ from 1 to 0.05 for exploration, following Samvelyan et al. (2019). For the discount factor, we set $\gamma = 0.99$. The replay buffer stores 5000 episodes at most, and the mini-batch is 32. Using a Nvidia Titan Xp graphic card, the training time varies from 8 hours to 24 hours for different scenarios.

To apply IQN for the true action-value estimator, we use an additional network, which computes an embedding $\phi(w_{\texttt{env}})$ for the sample point $w_{\texttt{env}}$. We calculate the embedding of $w_{\texttt{env}}$ with cosine basis functions and utilize element-wise (Hamard) product, instead of a simple vector concatenation, for merging function as in IQN paper. The element-wise product forces a sufficiently early interaction between the two representations. The only difference is that because we use mixing networks rather than DQN, the hidden layer of the mixing network, not the convolution features, is multiplied by the embedded sample point to force interaction between the features and the embedded sample point.

For the loss function $\mathcal{L}_{\texttt{ub}}$, we fixed the value only for the threshold $\widehat{Q}_{\texttt{jt}}(s, \boldsymbol{\tau}, \boldsymbol{u}_{\texttt{opt}})$ of the clipping function that receives optimal actions as input. To combine our loss functions, we obtain the following objective, which is minimized in an end-to-end manner to train the true action-value estimator and the transformed action-value estimator:

$$\mathcal{L} = \mathcal{L}_{\texttt{td}} + \lambda_{\texttt{opt}}\mathcal{L}_{\texttt{opt}} + \lambda_{\texttt{nopt}}\mathcal{L}_{\texttt{nopt}} + \lambda_{\texttt{ub}}\mathcal{L}_{\texttt{ub}}$$

where $\lambda_{\texttt{opt}}, \lambda_{\texttt{nopt}} > 0$ are hyperparameters controlling the importance of each loss function. We set $\lambda_{\texttt{opt}} = 3$ and $\lambda_{\texttt{nopt}}, \lambda_{\texttt{ub}} = 1$.

As for the loss, the calculation of $\mathcal{L}_{\texttt{td}}$ was performed for multiple samples at the same time as in IQN. We simply set the number of samples $N_{\texttt{env}}$ for $w_{\texttt{env}}$ and $N'_{\texttt{env}} = 8$ for $w'_{\texttt{env}}$. Also, for the losses for the transformed action-value estimator, the expected value of true action-value estimator $Q_{\texttt{jt}}(\boldsymbol{u}) = \mathbb{E}_{w_{\texttt{env}}}[Z_{\texttt{jt}}(\boldsymbol{u}, w_{\texttt{env}})]$ was obtained with $K_{\texttt{env}} = 8$ samples. For the IQN (Dabney et al., 2018a) architecture, we use cosine basis functions with an embedding size of 64, ReLU nonlinearity, and multiplicative interaction according to their paper. For environment-wise risk-sensitivity, we used sampled $w_{\texttt{env}}$ from $\mathcal{U}(0, 0.25)$ as risk-averse, and $\mathcal{U}(0.75, 1.0)$ as risk-seeking,

For agent-wise risk, we don't need to learn $Z_{\texttt{tran}}$ values for all possible $w_{\texttt{agt}}$ because, unlike $Z_{\texttt{jt}}^{\texttt{target}}$, the value distribution of $Z_{\texttt{tran}}$ is not used for the target in loss functions. Therefore, for simplicity of implementation, we sample a fixed set of $w_{\texttt{agt}}$ with intervals of 0.1 from 0.1 to 1. In the execution phase, we select optimal action based on our agent-wise risk-sensitivity $w_{\texttt{agt}}$. For the agent-wise risk-sensitivity, we used $w_{\texttt{agt}} = 0.5$ as risk-neutral, $w_{\texttt{agt}} = 1.0$ as risk-seeking, and $w_{\texttt{agt}} = 0.1$ as risk-averse.

[5] https://github.com/oxwhirl/smac
[6] https://github.com/oxwhirl/pymarl
[7] https://github.com/oxwhirl/wqmix
[8] https://github.com/wjh720/QPLEX
[9] https://github.com/j3soon/dfac