Abstract

In fully cooperative multi-agent reinforcement learning (MARL) settings, the environments are highly stochastic due to the partial observability of each agent and the continuously changing policies of the other agents. To address the above issues, we integrate distributional RL and value function factorization methods by proposing a Distributional Value Function Factorization (DFAC) framework to generalize expected value function factorization methods to their DFAC variants. DFAC extends the individual utility functions from deterministic variables to random variables, and models the quantile function of the total return as a quantile mixture. To validate DFAC, we demonstrate DFAC's ability to factorize a simple two-step matrix game with stochastic rewards and perform experiments on all Super Hard tasks of StarCraft Multi-Agent Challenge, showing that DFAC is able to outperform expected value function factorization baselines.

1. Introduction

In multi-agent reinforcement learning (MARL), one of the popular research directions is to enhance the training procedure of fully cooperative and decentralized agents. Examples of such agents include a fleet of unmanned aerial vehicles (UAVs), a group of autonomous cars, etc. This research direction aims to develop a decentralized and cooperative behavior policy for each agent, and is especially difficult for MARL settings without an explicit communication channel. The most straightforward approach is independent Q-learning (IQL) (Tan, 1993), where each agent is trained independently, with their behavior policies aimed to optimize the overall rewards in each episode. Nevertheless, each agent’s policy may not converge owing to two main difficulties: (1) non-stationary environments caused by the changing behaviors of the agents, and (2) spurious reward signals originated from the actions of the other agents. The agent’s partial observability of the environment further exacerbates the above issues. Therefore, in the past few years, a number of MARL researchers turned their attention to centralized training with decentralized execution (CTDE) approaches, with an objective to stabilize the training procedure while maintaining the agents’ abilities for decentralized execution (Oliehoek et al., 2016). Among these CTDE approaches, value function factorization methods (Sunehag et al., 2018; Rashid et al., 2018; Son et al., 2019) are especially promising in terms of their superior performances and data efficiency (Samvelyan et al., 2019).

Value function factorization methods introduce the assumption of individual-global-max (IGM) (Son et al., 2019), which assumes that each agent’s optimal actions result in the optimal joint actions of the entire group. Based on IGM, the total return of a group of agents can be factorized into separate utility functions (Guestrin et al., 2001) (or simply ‘utility’ hereafter) for each agent. The utilities allow the agents to independently derive their own optimal actions during execution, and deliver promising performance in StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al., 2019). Unfortunately, current value function factorization methods only concentrate on estimating the expectations of the utilities, overlooking the additional information contained in the full return distributions. Such information, nevertheless, has been demonstrated beneficial for policy learning in the recent literature (Lyle et al., 2019).

In the past few years, distributional RL has been empirically shown to enhance value function estimation in various single-agent RL (SARL) domains (Bellemare et al., 2017; Dabney et al., 2018a;b; Rowland et al., 2019; Yang et al., 2019). Instead of estimating a single scalar Q-value, it approximates the probability distribution of the return by either a categorical distribution (Bellemare et al., 2017) or a quantile function (Dabney et al., 2018a;b). Even though the above methods may be beneficial to the MARL domain due to the ability to capture uncertainty, it is inherently incompati-
able to expected value function factorization methods (e.g.,
value decomposition network (VDN) (Sunehag et al., 2018)
and QMIX (Rashid et al., 2018)). The incompatibility arises
from two aspects: (1) maintaining IGM in a distributional
form, and (2) factorizing the probability distribution of the
total return into individual utilities. As a result, an effective
and efficient approach that is able to solve the incompati-
bility is crucial and necessary for bridging the gap between
value function factorization methods and distributional RL.

In this paper, we propose a Distributional Value Function
Factorization (DFAC) framework, to efficiently integrate
value function factorization methods with distributional RL.
DFAC solves the incompatibility by two techniques: (1)
Mean-Shape Decomposition and (2) Quantile Mixture. The
former allows the generalization of expected value function
factorization methods (e.g., VDN and QMIX) to their DFAC
variants without violating IGM. The latter allows the total
return distribution to be factorized into individual utility dis-
tributions in a computationally efficient manner. To validate
the effectiveness of DFAC, we first demonstrate the abil-
ity of distribution factorization on a two-step matrix game
with stochastic rewards. Then, we perform experiments on
all Super Hard maps in SMAC. The experimental results
show that DFAC offers beneficial impacts on the baseline
methods in all Super Hard maps. In summary, the primary
contribution is the introduction of DFAC for bridging the
gap between distributional RL and value function factoriza-
tion methods efficiently by mean-shape decomposition and
quantile mixture.

2. Background and Related Works

In this section, we introduce the essential background ma-
terial for understanding the contents of this paper. We first
define the problem formulation of cooperative MARL and
CTDE. Next, we describe the conventional formulation of
IGM and the value function factorization methods. Then,
we walk through the concepts of distributional RL, quan-
tile function, as well as quantile regression, which are the
fundamental concepts frequently mentioned in this paper.
After that, we explain the implicit quantile network, a key
approach adopted in this paper for approximating quantiles.
Finally, we bring out the concept of quantile mixture, which
is leveraged by DFAC for factorizing the return distribution.

2.1. Cooperative MARL and CTDE

In this work, we consider a fully cooperative MARL envi-
ronment modeled as a decentralized and partially observable
Markov Decision Process (Dec-POMDP) (oliehoek &
Amato, 2016) with stochastic rewards, which is described
as a tuple \( \langle S, K, O, P, O, R, \gamma \rangle \) and is defined as
follows:

- \( S \) is the finite set of global states in the environment,
  where \( s' \in S \) denotes the next state of the current state
  \( s \in S \). The state information is optionally available
during training, but not available to the agents during
execution.
- \( K = \{ 1, ..., K \} \) is the set of K agents. We use \( k \in K \)
to denote the index of the agent.
- \( O_k = \prod_{k \in K} O_k \) is the set of joint observations. At
each timestep, a joint observation \( o = (o_1, ... o_K) \in O_k \)
is received. Each agent \( k \) is only able to observe its
individual observation \( o_k \in O_k \).
- \( H_{jt} = \prod_{k \in K} H_k \) is the set of joint action-observation
histories. The joint history \( h = (h_1, ... h_K) \in H_{jt} \)
concatenates all received observations and performed
actions before a certain timestep, where \( h_k \in H_k \)
represents the action-observation history from agent \( k \).
- \( U_{jt} = \prod_{k \in K} U_k \) is the set of joint actions. At each
timestep, the entire group of the agents take a joint action
\( u, \) where \( u = (u_1, ..., u_K) \in U_{jt} \). The individual
action \( u_k \in U_k \) of each agent \( k \) is determined based
on its stochastic policy \( \pi_k(u_k|h_k): H_k \times U_k \rightarrow [0,1] \),
expressed as \( u_k \sim \pi_k(\cdot|h_k) \). Similarly, in single agent
scenarios, we use \( u \) and \( u' \) to denote the actions of the
agent at state \( s \) and \( s' \) under policy \( \pi \), respectively.
- \( T = \{ 1, ..., T \} \) represents the set of timesteps with
horizon \( T \), where the index of the current timestep is
denoted as \( t \in T \). \( s', o', h', \) and \( u' \) correspond to the
environment information at timestep \( t \).
- The transition function \( P(s'|s,u) : S \times U_{jt} \times S \rightarrow
[0,1] \) specifies the state transition probabilities. Given
\( s \) and \( u \), the next state is denoted as \( s' \sim P(\cdot|s,u) \).
- The observation function \( O(s,u) : U_{jt} \times S \rightarrow [0,1] \)
specifies the joint observation probabilities. Given \( s \),
the joint observation is represented as \( o \sim O(\cdot|s) \).
- \( R(s,u) : S \times U_{jt} \times \mathbb{R} \rightarrow [0,1] \) is the joint reward
function shared among all agents. Given \( s \), the team
reward is expressed as \( r \sim R(\cdot|s,u) \).
- \( \gamma \in \mathbb{R} \) is the discount factor with value within \( (0,1] \).

Under such an MARL formulation, this work concentrates
on CTDE value function factorization methods, where the
agents are trained in a centralized fashion and executed in
a decentralized manner. In other words, the joint observa-
tion history \( h \) is available during the learning processes of
individual policies \( \{\pi_k\}_{k \in K} \). During execution, each agent’s
policy \( \pi_k \) only conditions on its observation history \( h_k \).

2.2. IGM and Factorizable Tasks

IGM is necessary for value function factorization (Son
et al., 2019). For a joint action-value function \( Q_{jt}(h,u) : H_{jt} \times
U_{jt} \rightarrow \mathbb{R}, \) if there exist K individual utility functions
\( Q_k(h_k, u_k) : H_k \times U_k \rightarrow \mathbb{R} \) such that the following
condition holds:
\[
\arg \max_u Q_{h}(h, u) = \left( \arg \max_{u_1} Q_1(h_1, u_1), \ldots, \arg \max_{u_K} Q_K(h_K, u_K) \right),
\]
(1)
then \([Q_k]_{k \in K}\) are said to satisfy IGM for \(Q_h\) under \(h\). In this case, we also say that \(Q_h(h, u)\) is factorized by \([Q_k(h_k, u_k)]_{k \in K}\) (Son et al., 2019). If \(Q_h\) in a given task is factorizable under all \(h \in H_j\), we say that the task is factorizable. Intuitively, factorizable tasks indicate that there exists a factorization such that each agent can select the greedy action according to their individual utilities \([Q_k]_{k \in K}\) independently in a decentralized fashion. This enables the optimal individual actions to implicitly achieve the optimal joint action across the \(K\) agents. Since there is no individual reward, the factorized utilities do not estimate expected returns on their own (Guestrin et al., 2001) and are different from the value function definition commonly used in SARL.

2.3. Value Function Factorization Methods

Based on IGM, value function factorization methods enable centralized training for factorizable tasks, while maintaining the ability for decentralized execution. In this work, we consider two such methods, VDN and QMIX, which can solve a subset of factorizable tasks that satisfies Additivity (Eq. (2)) and Monotonicity (Eq. (3)), respectively, given by:
\[
Q_{h}(h, u) = \sum_{k=1}^{K} Q_k(h_k, u_k),
\]
(2)
\[
Q_{h}(h, u) = M(Q_1(h_1, u_1), \ldots, Q_K(h_K, u_K)|s),
\]
(3)
where \(M\) is a monotonic function that satisfies \(\frac{\partial M}{\partial x_k} \geq 0, \forall \ k \in K\), and conditions on the state \(s\) if the information is available during training. Either of these two equation is a sufficient condition for IGM (Son et al., 2019).

2.4. Distributional RL

For notational simplicity, we consider a degenerated case with only a single agent, and the environment is fully observable until the end of Section 2.6. Distributional RL generalizes classic expected RL methods by capturing the full return distribution \(Z(s, u)\) instead of the expected return \(Q(s, u)\), and outperforms expected RL methods in various single-agent RL domains (Bellemare et al., 2017; 2019; Dabney et al., 2018b; Rowland et al., 2019; Yang et al., 2019). Moreover, distributional RL enables improvements (Nikolov et al., 2019; Zhang & Yao, 2019; Mavrin et al., 2019) that require the information of the full return distribution. We define the distributional Bellman optimality operator \(\mathcal{T}^\pi\) as follows:
\[
\mathcal{T}^\pi Z(s, u) \overset{D}{=} R(s, u) + \gamma Z(s', u'),
\]
(4)
and the distributional Bellman optimality operator \(\mathcal{T}^*\) as:
\[
\mathcal{T}^* Z(s, u) \overset{D}{=} R(s, u) + \gamma Z(s', u^*),
\]
(5)
where \(u^* = \arg \max_u \mathbb{E}[Z(s', u)]\) is the optimal action at state \(s'\), and the expression \(X \overset{D}{=} Y\) denotes that random variable \(X\) and \(Y\) follow the same distribution. Given some initial distribution \(Z_0\), \(Z\) converges to the return distribution \(Z^\pi\) under \(\pi\), contracting in terms of \(p\)-Wasserstein distance for all \(p \in [1, \infty)\) by applying \(\mathcal{T}^\pi\) repeatedly; while \(Z\) alternates between the optimal return distributions in the set \(Z^\pi := \{Z^\pi : \pi^* \in \Pi^*\}\), under the set of optimal policies \(\Pi^*\) by repeatedly applying \(\mathcal{T}^*\) (Bellemare et al., 2017). The \(p\)-Wasserstein distance \(W_p\) between the probability distributions of random variables \(X, Y\) is given by:
\[
W_p(X, Y) = \left( \int_0^1 |F_X^{-1}(\omega) - F_Y^{-1}(\omega)|^p d\omega \right)^{1/p},
\]
(6)
where \((F_X^{-1}, F_Y^{-1})\) are quantile functions of \((X, Y)\).

2.5. Quantile Function and Quantile Regression

The relationship between the cumulative distribution function (CDF) \(F_X\) and the quantile function \(F_X^{-1}\) (the generalized inverse CDF) of random variable \(X\) is formulated as:
\[
F_X^{-1}(\omega) = \inf \{x \in \mathbb{R} : \omega \leq F_X(x)\}, \forall \omega \in [0, 1],
\]
(7)
The expectation of \(X\) expressed in terms of \(F_X^{-1}(\omega)\) is:
\[
E[X] = \int_0^1 F_X^{-1}(\omega) \ d\omega.
\]
(8)
In (Dabney et al., 2018b), the authors model the value function as a quantile function \(F^{-1}(s, u|\omega)\). During optimization, a pair-wise sampled temporal difference (TD) error \(\delta\) for two quantile samples \(\omega, \omega'\) is defined as:
\[
\delta'_{s'} = r + \gamma F^{-1}(s', u' | \omega') - F^{-1}(s, u | \omega).
\]
(9)
The pair-wise loss \(\rho_{\kappa}^{\delta'}\) is then defined based on the Huber quantile regression loss \(\mathcal{L}_\kappa\) (Dabney et al., 2018b) with threshold \(\kappa = 1\), and is formulated as follows:
\[
\rho_{\kappa}^{\delta'}(\delta'_{s'}|\omega) = |\omega - I\{\delta'_{s'}|\omega < 0\}| \frac{\mathcal{L}_\kappa(\delta'_{s'}|\omega)}{\kappa},
\]
(10)
\[
\mathcal{L}_\kappa(\delta'_{s'}|\omega) = \begin{cases} \frac{1}{2} (\delta'_{s'}|\omega)^2, & \text{if } |\delta'_{s'}| \leq \kappa, \\ \kappa(|\delta'_{s'}| - \frac{1}{2}\kappa), & \text{otherwise}. \end{cases}
\]
(11)
Given \(N\) quantile samples \(\omega_j, j = 1, \ldots, N\) to be optimized with regard to \(N'\) target quantile samples \(\omega'_j, j = 1, \ldots, N'\), the total loss \(\mathcal{L}(s, u, r, s')\) is defined as the sum of the pair-wise losses, and is expressed as the following:
\[
\mathcal{L}(s, u, r, s') = \frac{1}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N'} \rho_{\kappa}^{\delta'}(\delta'_i|\omega_j). \]
2.6. Implicit Quantile Network

Implicit quantile network (IQN) (Dabney et al., 2018a) is relatively light-weight when it is compared to other distributional RL methods. It approximates the return distribution \( Z(s, u) \) by an implicit quantile function \( F^{-1}(s, u|\omega) = g(\psi(s), \phi(\omega))u \) for some differentiable functions \( g, \psi, \) and \( \phi \). Such an architecture is a type of universal value function approximator (UVFA) (Schaul et al., 2015), which generalizes its predictions across states \( s \in \mathcal{S} \) and goals \( \omega \in [0, 1] \), with the goals defined as different quantiles of the return distribution. In practice, \( \phi \) first expands the scalar \( \omega \) to an \( n \)-dimensional vector by \( \hat{\omega} = \cos(\pi \omega)_{i=0}^{n-1} \), followed by a single hidden layer with weights \( \omega_{ij} \) and biases \( b_j \) to produce a quantile embedding \( \phi(\omega)_j \) can be represented as the following:

\[
\phi(\omega)_j := \text{ReLU}(\sum_{i=0}^{n-1} \cos(\pi \omega)w_{ij} + b_j),
\]

where \( n = 64 \) and \( \dim(\phi(\omega)) = \dim(\psi(s)) \). Then, \( \phi(\omega) \) is combined with the state embedding \( \psi(s) \) by the element-wise (Hadamard) product \( \odot \), expressed as \( g := \psi \odot \phi \). The loss of IQN is defined as Eq. (12) by sampling a batch of \( N \) and \( N' \) quantiles from the policy network and the target network respectively. During execution, the action with the largest expected return \( Q(s, u) \) is chosen. Since IQN does not model the expected return explicitly, \( Q(s, u) \) is approximated by calculating the mean of the sampled return through \( N \) quantile samples \( \hat{\omega}_i \sim U([0, 1]), \forall i \in [1, N] \) based on Eq. (8), expressed as follows:

\[
Q(s, u) = \int_0^1 F^{-1}(s, u|\omega) \, d\omega \approx \frac{1}{N} \sum_{i=1}^{N} F^{-1}(s, u|\hat{\omega}_i).
\]

(14)

2.7. Quantile Mixture

Multiple quantile functions (e.g., IQNs) sharing the same quantile \( \omega \) may be combined into a single quantile function \( F^{-1}(\omega) \), in a form of quantile mixture expressed as follows:

\[
F^{-1}(\omega) = \sum_{k=1}^{K} \beta_k F^{-1}_k(\omega),
\]

where \( F^{-1}_k(\omega) \) are quantile functions, and \( \beta_k \) are model parameters (Karvanen, 2006). The condition for \( \beta_k \) is that \( F^{-1}(\omega) \) must satisfy the properties of a quantile function. The concept of quantile mixture is analogous to the mixture of multiple probability density functions (PDFs), expressed as follows:

\[
f(x) = \sum_{k=1}^{K} \alpha_k f_k(x),
\]

where \( f_k(x) \) are PDFs, \( \sum_{k=1}^{K} \alpha_k = 1 \), and \( \alpha_k \geq 0 \).

3. Methodology

In this section, we walk through the proposed DFAC framework and its derivation procedure. We first discuss a naive distributional factorization and its limitation in Section 3.1. Then, we introduce the DFAC framework to address the limitation, and show that DFAC is able to generalize distributional RL to all factorizable tasks in Section 3.2. After that, DDN and DMIX are introduced as the DFAC variants of VDN and QMIX, respectively, in Section 3.4. Finally, a practical implementation of DFAC based on quantile mixture is presented in Section 3.3. All proofs of the theorems in this section are provided in the supplementary material.

3.1. Distributional IGM

Since IGM is necessary for value function factorization, a distributional factorization that satisfies IGM is required for factoring stochastic value functions. We first discuss a naive distributional factorization that simply replaces deterministic utilities \( Q \) with stochastic utilities \( Z \). Then, we provide a theorem to show that the naive distributional factorization is insufficient to guarantee the IGM condition.

Definition 1 (Distributional IGM). A finite number of individual stochastic utilities \( [Z_k(h_k, u_k)]_{k \in \mathcal{K}} \), are said to satisfy Distributional IGM (DIGM) for a stochastic joint action-value function \( Z_{jt}(h, u) \), if \( E[Z_{jt}(h_k, u_k)]_{k \in \mathcal{K}} \) satisfy IGM for \( E[Z_{jt}(h, u)] \), represented as:

\[
\arg \max_u \mathbb{E}[Z_{jt}(h, u)] = \left( \begin{array}{c}
\arg \max_u \mathbb{E}[Z_1(h_1, u_1)] \\
\vdots \\
\arg \max_u \mathbb{E}[Z_K(h_K, u_K)]
\end{array} \right).
\]

Theorem 1. Given a deterministic joint action-value function \( Q_{jt} \), a stochastic joint action-value function \( Z_{jt} \), and a factorization function \( \Psi \) for deterministic utilities:

\[
Q_{jt}(h, u) = \Psi(Q_1(h_1, u_1), ..., Q_k(h_k, u_k)|s),
\]

such that \( [Q_k]_{k \in \mathcal{K}} \) satisfy IGM for \( Q_{jt} \) under \( h \), the following distributional factorization:

\[
Z_{jt}(h, u) = \Psi(Z_1(h_1, u_1), ..., Z_K(h_K, u_K)|s).
\]

is insufficient to guarantee that \( [Z_k]_{k \in \mathcal{K}} \) satisfy DIGM for \( Z_{jt} \) under \( h \).

In order to satisfy DIGM for stochastic utilities, an alternative factorization strategy is necessary.

3.2. The Proposed DFAC Framework

We propose Mean-Shape Decomposition and the DFAC framework to ensure that DIGM is satisfied for stochastic utilities.

Definition 2 (Mean-Shape Decomposition). A given ran-
We propose DFAC to decompose a joint return distribution $Z_{jt}$ into its deterministic part $Z_{\text{mean}}$ (i.e., expected value) and stochastic part $Z_{\text{shape}}$ (i.e., higher moments), which are approximated by two different functions $\Psi$ and $\Phi$, respectively. The factorization function $\Psi$ is required to precisely factorize the expectation of $Z_{jt}$ in order to satisfy DIGM. On the other hand, the shape function $\Phi$ is allowed to roughly factorize the shape of $Z_{jt}$, since the main objective of modeling the return distribution is to assist non-linear approximation of the expectation of $Z_{jt}$ (Lyle et al., 2019), rather than accurately model the shape of $Z_{jt}$.

**Theorem 2 (DFAC Theorem).** Given a deterministic joint action-value function $Q_{jt}$, a stochastic joint action-value function $Z_{jt}$, and a factorization function $\Psi$ for deterministic utilities:

$$Q_{jt}(h, u) = \Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s),$$

such that $[Q_k]_{k \in K}$ satisfy IGM for $Q_{jt}$ under $h$, by Mean-Shape Decomposition, the following distributional factorization:

$$Z_{jt}(h, u) = \mathbb{E}[Z_{jt}(h, u)] + (Z_{jt}(h, u) - \mathbb{E}[Z_{jt}(h, u)])$$

$$= Z_{\text{mean}}(h, u) + Z_{\text{shape}}(h, u)$$

$$= \Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s)$$

$$+ \Phi(Z_1(h_1, u_1), ..., Z_K(h_K, u_K)|s).$$

is sufficient to guarantee that $[Z_k]_{k \in K}$ satisfy DIGM for $Z_{jt}$ under $h$, where $\text{Var}(\Psi) = 0$ and $\mathbb{E}[\Phi] = 0$.

Theorem 2 reveals that the choice of $\Psi$ determines whether IGM holds, regardless of the choice of $\Phi$, as long as $\mathbb{E}[\Phi] = 0$. Under this setting, any differentiable factorization function of deterministic variables can be extended to a factorization function of random variables. Such a decomposition enables approximation of joint distributions for all factorizable tasks under appropriate choices of $\Psi$ and $\Phi$.

### 3.3. A Practical Implementation of DFAC

In this section, we provide a practical implementation of the shape function $\Phi$ in DFAC, effectively extending any differentiable factorization function $\Psi$ (e.g., the additive function of VDN, the monotonic mixing network of QMIX, etc.) that satisfies the IGM condition into its DFAC variant.

Theoretically, the sum of random variables appeared in $DDN$ and $DMIX$ can be described precisely by a joint CDF. However, the exact derivation of this joint CDF is usually computationally expensive and impractical (Lin et al., 2019). As a result, DFAC utilizes the property of quantile mixture to approximate the shape function $\Phi$ in $O(KN)$ time.

**Theorem 3.** Given a quantile mixture:

$$F^{-1}(\omega) = \sum_{k=1}^{K} \beta_k F_k^{-1}(\omega)$$

with $K$ components $[F_k^{-1}]_{k \in K}$ and non-negative model parameters $[\beta_k]_{k \in K}$. There exist a set of random variables $Z$ and $[Z_k]_{k \in K}$ corresponding to the quantile functions $F_k^{-1}$ and $[F_k^{-1}]_{k \in K}$, respectively, with the following relationship:

$$Z = \sum_{k \in K} \beta_k Z_k.$$
4. A Stochastic Two-Step Game

In the previous expected value function factorization methods (e.g., VDN, QMIX, etc.), the factorization is achieved by modeling $Q_{jk}$ and $[Q_k]_{k \in K}$ as deterministic variables, overlooking the information of higher moments in the full return distributions $Z_{jk}$ and $[Z_k]_{k \in K}$. In order to demonstrate DFAC’s ability of factorization, we begin with a toy example modified from (Rashid et al., 2018) to show that DFAC is able to approximate the true return distributions, and factorize the mean and variance of the approximated return distributions (i.e., $Z_{jk}$ and $[Z_k]_{k \in K}$).

Table 1 illustrates the flow of a two-step game consisting of two agents and three states $1, 2A,$ and $2B$, where $State 1$ serves as the initial state, and each agent is able to perform an action from $\{A, B\}$ in each step. In the first step (i.e., $State 1$), the action of agent 1 (i.e., actions $A_1$ or $B_1$) determines which of the two matrix games ($State 2A$ or $State 2B$) to play in the next step, regardless of the action performed by agent 2 (i.e., actions $A_2$ or $B_2$). For all joint actions performed in the first step, no reward is provided to the agents.

In the second step, both agents choose an action and receive a global reward according to the payoff matrices depicted in Table 1, where the global rewards are sampled from a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with mean $\mu$ and standard deviation $\sigma$. The hyperparameters of the two-step game are offered in the supplementary material in detail.

Table 2 presents the learned factorization of $DMIX$ for each state after convergence, where the first rows and the first columns of the tables correspond to the factorized distributions of the individual utilities (i.e., $Z_1$ and $Z_2$), and the main content cells of them correspond to the joint return distributions (i.e., $Z_{jk}$). From Tables 2(b) and 2(c), it is observed that no matter the true returns are deterministic (i.e., $State 2A$) or stochastic (i.e., $State 2B$), $DMIX$ is able to approximate the true returns in Table 1 properly, which are not achievable by expected value function factorization methods. The results demonstrate DFAC’s ability to factorize the joint return distribution rather than expected returns. $DMIX$’s ability to reconstruct the optimal joint policy in the two-step game further shows that $DMIX$ can represent the same set of tasks as QMIX.

To further illustrate DFAC’s capability of factorization, Figs. 2(a)-2(b) visualize the factorization of the joint action $(B_1, B_2)$ in $State 2A$ and $(B_1, B_2)$ in $State 2B$, respectively. As IQN approximates the utilities $Z_1$ and $Z_2$ implicitly, $Z_1$, $Z_2$, and $Z_{jk}$ can only be plotted in terms of samples. $Z_{jk}$ in Fig. 2(a) shows that $DMIX$ degenerates to QMIX when approximating deterministic returns (i.e., $\mathcal{N}(7, 0)$), while $Z_{jk}$ in Fig. 2(b) exhibits $DMIX$’s ability to capture the uncertainty in stochastic returns (i.e., $\mathcal{N}(8, 29)$).

5. Experiment Results on SMAC

In this section, we present the experimental results and discuss their implications. We start with a brief introduction to our experimental setup in Section 5.1. Then, we demon-
Table 1: An illustration of the flow of the stochastic two-step game. Each agent is able to perform an action from \{A, B\} in each step, with a subscript denoting the agent index. In the first step, action A \(_1\) takes the agents from the initial State 1 to State 2A, while action B \(_1\) takes them to State 2B instead. The transitions from State 1 to State 2A or State 2B yield zero reward. In the second step, the global rewards are sampled from the normal distributions defined in the payoff matrices.

| Agent 1 | State 2A | Agent 2 |
|---------|----------|---------|
| A \(_1\) | \(N(7, 0)\) | \(N(7, 0)\) |
| B \(_1\) | \(N(7, 0)\) | \(N(7, 0)\) |

Table 2: The learned factorization of DMIX. All of the cells show the sampled mean \(\mu\) and the sampled variance \(\sigma^2\) with Bessel’s correction. The main content cells correspond to the joint return distributions for different combinations of states and actions. The first columns and first rows of these tables correspond to the distributions of the utilities for agents 1 and 2, respectively. The top-left cells of these tables are the state-dependent utility \(V\). DFAC enables the approximation of the true joint return distributions in Table 1, and allows them to be factorized into the distributions of the utilities for the agents.

| State 1 | A \(_2\) | B \(_2\) | State 2A | A \(_2\) | B \(_2\) | State 2B | A \(_2\) | B \(_2\) |
|---------|----------|----------|---------|----------|----------|---------|----------|----------|
| \(\mu = 0.32\) | \(\mu = 0.49\) | \(\mu = 0.49\) | \(\mu = 1.76\) | \(\mu = 1.75\) | \(\mu = 1.75\) | \(\mu = 0.38\) | \(\mu = 0.38\) | \(\mu = 0.38\) |
| \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) | \(\sigma^2 = 0.00\) |

(a) Learned utilities of State 1 (b) Learned utilities of State 2A (c) Learned utilities of State 2B

Figure 2: (a) and (b) plot the value function factorization of the joint action \((B_1, B_2)\) in State 2A and State 2B. The black line/curve shows the true return CDFs. The blue circles and the orange cross marks depict agent 1’s and agent 2’s learned utility, respectively, while the green squares indicate the estimated joint return.

5.1. Experimental Setup

Environment. We verify the DFAC framework in the SMAC benchmark environments (Samvelyan et al., 2019) built on the popular real-time strategy game StarCraft II. Instead of playing the full game, SMAC is developed for evaluating the effectiveness of MARL micro-management algorithms. Each environment in SMAC contains two teams. One team is controlled by a decentralized MARL algorithm, with the policies of the agents conditioned on their local observation histories. The other team consists of enemy units controlled by the built-in game artificial intelligence based on carefully handcrafted heuristics, which is set to its highest difficulty equal to seven. The overall objective is to maximize the win rate for each battle scenario, where the rewards employed in our experiments follow the default settings of SMAC. The default settings use shaped rewards based on the damage dealt, enemy units killed, and whether the RL agents win the battle. If there is no healing unit in the enemy team, the maximum return of an episode (i.e., the score) is 20; otherwise, it may exceed 20, since enemies may receive more damages after healing or being healed.

The environments in SMAC are categorized into three different levels of difficulties: Easy, Hard, and Super Hard scenarios (Samvelyan et al., 2019). In this paper, we focus on all Super Hard scenarios including (a) 6h_vs_8z, (b) 3s5z_vs_3s6z, (c) MMM2, (d) 27m_vs_30m, and (e) corridor, since these scenarios have not been properly addressed in the previous literature without the use of additional assumptions such as intrinsic reward signals (Du et al., 2019), explicit communication channels (Zhang et al., 2019; Wang et al., 2019), common knowledge shared among the
agents (de Witt et al., 2019; Wang et al., 2020), and so on. Three of these scenarios have their maximum scores higher than 20. In 3s5z_vs_3s6z, the enemy Stalkers have the ability to regenerate shields; in MMM2, the enemy Medivacs can heal other units; in corridor, the enemy Zerglings slowly regenerate their own health.

**Hyperparameters.** For all of our experimental results, the training length is set to 8M timesteps, where the agents are evaluated every 40k timesteps with 32 independent runs. The curves presented in this section are generated based on five different random seeds. The solid lines represent the median win rate, while the shaded areas correspond to the 25th to 75th percentiles. For a better visualization, the presented curves are smoothed by a moving average filter with its window size set to 11. The detailed hyperparameter setups are provided in the supplementary material.

**Baselines.** We select IQL, VDN, and QMIX as our baseline methods, and compare them with their distributional variants in our experiments. The configurations are optimized so as to provide the best performance for each of the methods considered. Since we tuned the hyperparameters of the baselines, their performances are better than those reported in (Samvelyan et al., 2019). The hyperparameter searching process is detailed in the supplementary material.

**5.2. Independent Learners**

In order to validate our assumption that distributional RL is beneficial to the MARL domain, we first employ the simplest training algorithm, IQL, and extend it to its distributional variant, called DIQL. DIQL is simply a modified IQL that uses IQN as its underlying RL algorithm without any additional modification or enhancements (Matignon et al., 2007; Lyu & Amato, 2020).

From Figs. 3(a)-3(e) and Tables 3-4, it is observed that DIQL is superior to IQL even without utilizing any value function factorization methods. This validates that distributional RL has beneficial influences on MARL, when it is compared to RL approaches based only on expected values.

**5.3. Value Function Factorization Methods**

In order to inspect the effectiveness and impacts of DFAC on learning curves, win rates, and scores, we next summarize the results of the baselines as well as their DFAC variants on the Super Hard scenarios in Fig. 3(a)-(e) and Table 3-4. Fig. 3(a)-(e) plot the learning curves of the baselines and their DFAC variants, with the final win rates presented in Table 3, and their final scores reported in Table 4. The win rates indicate how often do the player’s team wins, while the scores represent how well do the player’s team performs. Despite the fact that SMAC’s objective is to maximize the win rate, the true optimization goal of MARL algorithms is the averaged score. In fact, these two metrics are not always positively correlated (e.g., VDN and QMIX in 6h_vs_8z and 3s5z_vs_3s6z, and QMIX and DMIX in 3s5z_vs_3s6z).

It can be observed that the learning curves of DDN and DMIX grow faster and achieve higher final win rates than their corresponding baselines. In the most difficult map: 6h_vs_8z, most of the methods fail to learn an effective policy except for DDN and DMIX. The evaluation results
also show that DDN and DMIX are capable of performing consistently well across all Super Hard maps with high win rates. In addition to the win rates, Table 4 further presents the final averaged scores achieved by each method, and provides deeper insights into the advantages of the DFAC framework by quantifying the performances of the learned policies of different methods.

The improvements in win rates and scores are due to the benefits offered by distributional RL (Lyle et al., 2019), which enables the distributional variants to work more effectively in MARL environments. Moreover, the evaluation results reveal that DDN performs especially well in most environments despite its simplicity. Further validations of DDN and DMIX on our self-designed Ultra Hard scenarios that are more difficult than Super Hard scenarios can be found in our GitHub repository (https://github.com/j3soon/dfac), along with the gameplay recording videos.

6. Conclusion

In this paper, we provided a distributional perspective on value function factorization methods, and introduced a framework, called DFAC, for integrating distributional RL with MARL domains. We first proposed DFAC based on a mean-shape decomposition procedure to ensure the Distributional IGM condition holds for all factorizable tasks. Then, we proposed the use of quantile mixture to implement the mean-shape decomposition in a computationally friendly manner. DFAC’s ability to factorize the joint return distribution into individual utility distributions was demonstrated by a toy example. In order to validate the effectiveness of DFAC, we presented experimental results performed on all Super Hard scenarios in SMAC for a number of MARL baseline methods as well as their DFAC variants. The results show that DDN and DMIX outperform VDN and QMIX. DFAC can be extended to more value function factorization methods and offers an interesting research direction for future endeavors.

7. Acknowledgements

The authors acknowledge the support from NVIDIA Corporation and NVIDIA AI Technology Center (NVAITC). The authors thank Kuan-Yu Chang for his helpful critiques of this research work. The last author would like to thank the funding support from Ministry of Science and Technology (MOST) in Taiwan under grant nos. MOST 110-2636-E-007-010 and MOST 110-2634-F-007-019.

References

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In Proc. Int. Conf. on Machine Learning (ICML), pp. 449–458, Jul. 2017.

Bellemare, M. G., Roux, N. L., Castro, P. S., and Moitra, S. Distributional reinforcement learning with linear function approximation. arXiv preprint arXiv:1902.03149, 2019.

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. Implicit quantile networks for distributional reinforcement learning. In Proc. Int. Conf. on Machine Learning (ICML), pp. 1096–1105, Jul. 2018a.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. Distributional reinforcement learning with quantile regression. In Proc. AAAI Conf. on Artificial Intelligence (AAAI), pp. 2892–2901, Feb. 2018b.

de Witt, C. S., Foerster, J., Farquhar, G., Torr, P., Böhmer, W., and Whiteson, S. Multi-agent common knowledge reinforcement learning. In Advances in Neural Information Processing Systems, pp. 9924–9935, 2019.

Du, Y., Han, L., Fang, M., Liu, J., Dai, T., and Tao, D. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. In Advances in Neural Information Processing Systems, pp. 4405–4416, 2019.

Guestrin, C., Koller, D., and Parr, R. Multiagent planning with factored mdps. In NIPS, 2001.

Karvanen, J. Estimation of quantile mixtures via l-moments and trimmed l-moments. Computational Statistics & Data Analysis, 51:947–959, 11 2006. doi: 10.1016/j.csda.2005.09.014.

Lin, Z., Zhao, L., Yang, D., Qin, T., Liu, T.-Y., and Yang, G. Distributional reward decomposition for reinforcement learning. In Advances in Neural Information Processing Systems, pp. 6212–6221, 2019.

Lyle, C., Bellemare, M. G., and Castro, P. S. A comparative analysis of expected and distributional reinforcement learning. In Proc. AAAI Conf. on Artificial Intelligence (AAAI), pp. 4504–4511, Feb. 2019.

Lyu, X. and Amato, C. Likelihood quantile networks for coordinating multi-agent reinforcement learning. In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, pp. 798–806, 2020.

Matignon, L., Laurent, G., and Fort-Piat, N. Hysteretic q-learning : an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. pp. 64 – 69, 12 2007. doi: 10.1109/IROS.2007.4399095.

Mavrin, B., Yao, H., Kong, L., Wu, K., and Yu, Y. Distributional reinforcement learning for efficient exploration.
In Proc. Int. Conf. on Machine Learning (ICML), pp. 4424–4434, Jul. 2019.

Nikolov, N., Kirschner, J., Berkenkamp, F., and Krause, A. Information-directed exploration for deep reinforcement learning. In Proc. Int. Conf. on Learning Representations (ICLR), May 2019. URL https://openreview.net/forum?id=Byx83s09Km.

Oliehoek, F. A. and Amato, C. A Concise Introduction to Decentralized POMDPs. Springer, 2016. ISBN 3319289276.

Rashid, T. et al. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Proc. Int. Conf. on Machine Learning (ICML), pp. 4295–4304, Jul. 2018.

Rowland, M. et al. Statistics and samples in distributional reinforcement learning. In Proc. Int. Conf. on Machine Learning (ICML), pp. 5528–5536, Jul. 2019.

Samvelyan, M. et al. The starcraft multi-agent challenge. In Proc. Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS), pp. 2186–2188, May 2019.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In International conference on machine learning, pp. 1312–1320, 2015.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In Proc. Int. Conf. on Machine Learning (ICML), pp. 5887–5896, Jul. 2019.

Sunehag, P. et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. In Proc. Int. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS), pp. 2085–2087, May 2018.

Tan, M. Multi-agent reinforcement learning: Independent versus cooperative agents. In Proc. Int. Conf. on Machine Learning (ICML), pp. 330–337, Jun. 1993. ISBN 1558603077.

Wang, T., Wang, J., Zheng, C., and Zhang, C. Learning nearly decomposable value functions via communication minimization. arXiv preprint arXiv:1910.05366, 2019.

Wang, T., Dong, H., Lesser, V., and Zhang, C. Multi-agent reinforcement learning with emergent roles. arXiv preprint arXiv:2003.08039, 2020.

Yang, D. et al. Fully parameterized quantile function for distributional reinforcement learning. In Proc. Conf. Advances in Neural Information Processing Systems (NeurIPS), pp. 6190–6199, Dec. 2019.

Zhang, S. and Yao, H. Quota: The quantile option architecture for reinforcement learning. In Proc. AAAI Conf. on Artificial Intelligence (AAAI), pp. 5797–5804, Feb. 2019.

Zhang, S. Q., Zhang, Q., and Lin, J. Efficient communication in multi-agent reinforcement learning via variance based control. In Advances in Neural Information Processing Systems, pp. 3230–3239, 2019.
Supplementary Materials

Wei-Fang Sun1,2 Cheng-Kuang Lee2 Chun-Yi Lee1

1Department of Computer Science, National Tsing Hua University, Taiwan
2NVIDIA AI Technology Center, NVIDIA Corporation

S1 Theorems and Proofs

In this section, we elaborate on the definitions, and provide the proofs of the theorems discussed in the main manuscript.

Proposition 1. Monotonicity for utility distributions:

\[ Z_{jt}(h, u) = \Psi(Z_1(h_1, u_1), ..., Z_K(h_K, u_K)|s) = M(Z_1(h_1, u_1), ..., Z_K(h_K, u_K)|s), \]

where \( M \) is a monotonic transformation that satisfies \( \frac{\partial M}{\partial Q_k} \geq 0, \forall k \in K \), is not a sufficient condition for DIGM, although the equality may hold for special cases of \( M \) and \([Z_k(h_k, u_k)]_{k \in K} \).

Proof. We consider a degenerated case and prove the theorem by contradiction. Consider a case where there is only a single agent (\( K = 1 \)), with a single fully observable state and an exponential transformation \( M(Z_1(h_1, u_1)|s) = \exp(Z_1(h_1, u_1)) \). The (joint) action space of this case consists of two (joint) actions: \( U_{jt} = U_1 = \{u_1^*, u_1'\} \), where \( u_1^* \) is the optimal action (with expected return 2) and \( u_1' \) is the suboptimal action (with expected return 1.5). We define the probability mass function (PMF) of \( Z_1(h_1, u_1^*) \) to be:

\[
p(z) = \begin{cases} 
1 & \text{if } z = 2 \\
0 & \text{otherwise,}
\end{cases}
\]

and the PMF of \( Z_1(h_1, u_1') \) to be:

\[
p(z) = \begin{cases} 
0.5 & \text{if } z = 0 \\
0.5 & \text{if } z = 3 \\
0 & \text{otherwise.}
\end{cases}
\]

By the definition above, we can calculate the followings:

\[
\begin{align*}
\mathbb{E}[Z_1(h_1, u_1^*)] &= 1 \cdot 2 = 2 \\
\mathbb{E}[Z_1(h_1, u_1')] &= 0.5 \cdot 0 + 0.5 \cdot 3 = 1.5 \\
\arg\max_{u_1} \mathbb{E}[Z_1(h_1, u_1)] &= u_1^* \\
\mathbb{E}[M(Z_1(h_1, u_1)|s)] &= \mathbb{E}[\exp(Z_1(h_1, u_1^*))] = e^2 \approx 7.39 \\
\mathbb{E}[M(Z_1(h_1, u_1')|s)] &= \mathbb{E}[\exp(Z_1(h_1, u_1'))] = 0.5 \cdot e^0 + 0.5 \cdot e^3 \approx 10.54 \\
\arg\max_{u_1} \mathbb{E}[M(Z_1(h_1, u_1)|s)] &= u_1'
\end{align*}
\]
Assume, to the contrary, that *Monotonicity* for utility distributions is a sufficient condition for DIGM. By the definition of DIGM:

\[
\arg\max_u \mathbb{E}[Z_{jt}(h, u)] = \left( \arg\max_{u_1} \mathbb{E}[Z_1(h_1, u_1)] \right)
\]

\[
\Rightarrow \arg\max_u \mathbb{E}[\mathcal{M}(Z_1(h_1, u_1)|s)] = \arg\max_{u_1} \mathbb{E}[Z_1(h_1, u_1)]
\]

\[
\Rightarrow u'_1 = u^*_1 \quad (\Rightarrow \text{ contradiction}).
\]

A contradiction occurs since \( u'_1 \neq u^*_1 \), showing that *Monotonicity* is not a sufficient condition for DIGM.

Since there exist a case where DIGM does not hold for \( K = 1 \), it certainly does not hold for all \( K \in \mathbb{Z} \).

\[\square\]

**Theorem 1.** Given a deterministic joint action-value function \( Q_{jt} \), a stochastic joint action-value function \( Z_{jt} \), and a factorization function \( \Psi \) for deterministic utilities:

\[
Q_{jt}(h, u) = \Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s),
\]

such that \( [Q_k]_{k \in K} \) satisfy IGM for \( Q_{jt} \) under \( h \), the following distributional factorization:

\[
Z_{jt}(h, u) = \Psi(Z_1(h_1, u_1), ..., Z_K(h_K, u_K)|s).
\]

is insufficient to guarantee that \( [Z_k]_{k \in K} \) satisfy DIGM for \( Z_{jt} \) under \( h \).

**Proof.** A contradiction is provided by Proposition 1.

\[\square\]

**Theorem 2 (DFAC Theorem).** Given a deterministic joint action-value function \( Q_{jt} \), a stochastic joint action-value function \( Z_{jt} \), and a factorization function \( \Psi \) for deterministic utilities:

\[
Q_{jt}(h, u) = \Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s),
\]

such that \( [Q_k]_{k \in K} \) satisfy IGM for \( Q_{jt} \) under \( h \), by Mean-Shape Decomposition, the following distributional factorization:

\[
Z_{jt}(h, u) = \mathbb{E}[Z_{jt}(h, u)] + (Z_{jt}(h, u) - \mathbb{E}[Z_{jt}(h, u)])
\]

\[
= Z_{\text{mean}}(h, u) + Z_{\text{shape}}(h, u)
\]

\[
= \Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s)
\]

\[
+ \Phi(Z_1(h_1, u_1), ..., Z_K(h_K, u_K)|s).
\]

is sufficient to guarantee that \( [Z_k]_{k \in K} \) satisfy DIGM for \( Z_{jt} \) under \( h \), where \( \text{Var}(\Psi) = 0 \) and \( \mathbb{E}[\Phi] = 0 \).
Proof. By mean-shape decomposition:

\[ \arg \max_u \{ \mathbb{E}[Z_{jt}(h, u)] \} \]

\[ = \arg \max_u \{ \mathbb{E}[Z_{\text{mean}}(h, u)] + \mathbb{E}[Z_{\text{shape}}(h, u)] \} \]

\[ = \arg \max_u \{ \mathbb{E}[\Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s)] + \mathbb{E}[\Phi(Z_1(h_1, u_1), ..., Z_K(h_K, u_K)|s)] \} \]

\[ = \arg \max_u \{ \Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s) + 0 \} \]

\[ = \arg \max_u \{ \Psi(Q_1(h_1, u_1), ..., Q_K(h_K, u_K)|s) \} \]

\[ = \left( \begin{array}{c}
  \arg \max_{u_1} Q_1(h_1, u_1) \\
  \vdots \\
  \arg \max_{u_K} Q_K(h_K, u_K)
\end{array} \right) \]

\[ \Rightarrow \arg \max_u \mathbb{E}[Z_{jt}(h, u)] = \left( \begin{array}{c}
  \arg \max_{u_1} \mathbb{E}[Z_1(h_1, u_1)] \\
  \vdots \\
  \arg \max_{u_K} \mathbb{E}[Z_K(h_K, u_K)]
\end{array} \right). \]

The equations above show that \([Z_k]_{k \in K}\) satisfy \(\text{DIGM}\) for \(Z_{jt}\) under \(h\).

\[ \square \]

**Theorem 3.** Given a quantile mixture:

\[ F^{-1}(\omega) = \sum_{k=1}^{K} \beta_k F_k^{-1}(\omega) \]

with \(K\) components \([F_k^{-1}]_{k \in K}\) and non-negative model parameters \([\beta_k]_{k \in K}\). There exist a set of random variables \(Z = F^{-1}(\tau)\) and \([Z_k = F_k^{-1}(\tau)]_{k \in K}\) corresponding to the quantile functions \(F^{-1}\) and \([F_k^{-1}]_{k \in K}\), respectively, where \(\tau\) is a random variable uniformly distributed on \([0, 1]\), with the following relationship:

\[ Z \equiv \sum_{k \in K} \beta_k Z_k. \]

**Proof.** We first prove the case for a quantile mixture with \(K = 2\) components, and then generalize it to all \(K \in \mathbb{Z}\). For \(K = 2\), the quantile mixture is simplified as follows:

\[ F^{-1}(\tau) = \beta_1 F_1^{-1}(\tau) + \beta_2 F_2^{-1}(\tau) \]

For notational simplicity, let \(X = \beta_1 Z_1, Y = \beta_2 Z_2,\) and \(\tau\) is a latent variable shared among the random variables \(X, Y,\) and \(Z\). The corresponding CDFs of the random variables \(X, Y,\) and \(Z\) are \(F_X, F_Y,\) and \(F_Z,\) respectively, with \(X(\tau) = F_X^{-1}(\tau), Y(\tau) = F_Y^{-1}(\tau),\) and \(Z(\tau) = F_Z^{-1}(\tau).\) Under this notation, the above equation can be re-written as:

\[ F_Z^{-1}(\tau) = F_X^{-1}(\tau) + F_Y^{-1}(\tau). \]
The goal is to prove that there exist random variables \((X,Y,Z)\) such that the following holds:

\[
Z \overset{D}{=} X + Y
\]

By the definition of the CDF of \(X + Y\), the following holds:

\[
F_{X+Y}(z), \forall z \in \mathbb{R} = \Pr(X + Y \leq z), \forall z \in \mathbb{R}
\]

The proof for quantile mixtures with two components can be iteratively applied to quantile mixtures with \(K \in \mathbb{Z}\) components.

\[\square\]

### S2 Hyperparameters and Settings

#### S2.1 Stochastic Two Step Game

In the stochastic two step game described in Section 4, each agent is implemented as an IQN with two hidden layers comprised of 64 units and 512 units, respectively, with a ReLU nonlinearity at the end of each layer. We optimize the IQNs with \(N = N' = 32\) quantile samples, where each of them is encoded into a 64-dimensional intermediate embedding and projected to a 512-dimensional quantile embedding by a single hidden layer. Each agent performs independent \(\epsilon\)-greedy action selection, with full exploration (i.e., \(\epsilon = 1\)). We set the discount factor \(\gamma\) to 0.99. The replay buffer contains the state-action pairs of the latest \(2k\) episodes, from which we uniformly sample a batch of size 512 for training. The target network is updated every 100 episodes. The optimizer is set to Adam, in which its learning rate is set to \(1 \times 10^{-4}\). We train for \(20k\) timesteps \((10k\) episodes\). All agent networks share parameters, and the one-hot encoded agent id \(([1 0]^T\) for agent 1 and \([0 1]^T\) for agent 2) is concatenated to each agent’s observation. We do not pass the previous actions taken by the agents as their inputs. Each agent receives the full state as its input. For DMIX, we use a mixing network with 8 units.

Each state is one-hot encoded. The starting state for the first timestep is State 1 (one-hot: \([1 0 0]^T\)). At State 1, if Agent 1 selects Action A, the agents transit to State 2A (one-hot: \([0 1 0]^T\)). On the other hand, if agent 1 selects Action B, the agents transit to State 2B (one-hot: \([0 0 1]^T\)).

#### S2.2 SMAC

We tuned the hyperparameters of both the baselines and their distributional variants by selecting their hidden layer sizes from \{32, 64, 128, 256, 512\} and choose the best ones. The quantile samples of DIQL and DDN
Table S1: A summary of the optimal hidden state sizes of the baseline methods and their distributional variants.

| Maps          | IQL | VDN | QMIX | QR-MIX | DIQL | DDN | DMIX |
|---------------|-----|-----|------|--------|------|-----|------|
| 3s5z_vs_3s6z  | 512 | 128 | 128  | 128    | 256  | 512 | 256  |
| 6h_vs_8z      | 128 | 128 | 256  | 256    | 512  | 512 | 256  |
| MMM2          | 256 | 64  | 64   | 64     | 512  | 512 | 256  |
| 27m_vs_30m    | 256 | 64  | 64   | 64     | 512  | 128 | 128  |
| corridor      | 256 | 128 | 256  | 64     | 512  | 128 | 64   |

Table S2: The detailed settings of the Super Hard maps.

| Difficulty | Map       | Player's Team                      | Enemy's Team                          |
|------------|-----------|------------------------------------|---------------------------------------|
|            | 6h_vs_8z  | 6 Hydralisks                        | 8 Zealots                             |
| Super Hard | 3s5z_vs_3s6z | 3 Stalkers & 5 Zealots             | 3 Stalkers & 6 Zealots                |
|            | MMM2      | 7 Marines, 2 Marauders & 1 Medivac  | 8 Marines, 3 Marauders & 1 Medivac    |
|            | 27m_vs_30m | 27 Marines                          | 30 Marines                            |
|            | corridor  | 6 Zealots                           | 24 Zerglings                          |

are simply set to $N = N' = 1$, since they do not require the calculation of the expected value during the optimization process. As for DMIX, the numbers of quantile samples are set to $N = N' = 8$ as in [1]. The optimizers follow those used in DQN and IQN. All of the other hyperparameters follow those used in SMAC. Table S1 lists the hyperparameters adopted for the baselines and their distributional variants. The StarCraft version we used is 4.10.

References

[1] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit quantile networks for distributional reinforcement learning. In Proc. Int. Conf. on Machine Learning (ICML), pages 1096–1105, Jul. 2018.