DQMIX: A DISTRIBUTIONAL PERSPECTIVE ON MULTI-AGENT REINFORCEMENT LEARNING

Jian Zhao[1], Mingyu Yang[1], Xunhan Hu[1], Wengang Zhou[1], and Houqiang Li[1]

[1]University of Science and Technology of China

February 22, 2022

ABSTRACT

In cooperative multi-agent tasks, a team of agents jointly interact with an environment by taking actions, receiving a team reward and observing the next state. During the interactions, the uncertainty of environment and reward will inevitably induce stochasticity in the long-term returns and the randomness can be exacerbated with the increasing number of agents. However, most of the existing value-based multi-agent reinforcement learning (MARL) methods only model the expectations of individual Q-values and global Q-value, ignoring such randomness. Compared to the expectations of the long-term returns, it is more preferable to directly model the stochasticity by estimating the returns through distributions. With this motivation, this work proposes DQMIX, a novel value-based MARL method, from a distributional perspective. Specifically, we model each individual Q-value with a categorical distribution. To integrate these individual Q-value distributions into the global Q-value distribution, we design a distribution mixing network, based on five basic operations on the distribution. We further prove that DQMIX satisfies the *Distributional-Individual-Global-Max* (DIGM) principle with respect to the expectation of distribution, which guarantees the consistency between joint and individual greedy action selections in the global Q-value and individual Q-values. To validate DQMIX, we demonstrate its ability to factorize a matrix game with stochastic rewards. Furthermore, the experimental results on a challenging set of StarCraft II micromanagement tasks show that DQMIX consistently outperforms the value-based multi-agent reinforcement learning baselines.

1 Introduction

Reinforcement learning (RL) has been widely adopted in a variety of cooperative multi-agent problems, such as multiplayer games [1, 2, 3, 4], sensor networks [5, 6, 7] and traffic light control [8, 9, 10]. In the setting of partial observability and communication constraints, multiple agents are required to take actions to interact with the environment in a decentralized manner. During the interaction process, these partial observations, changing policies of all the agents, rewards and state transitions can all bring about stochasticity in the observed long-term return. Moreover, the stochasticity caused by actions will be intensified with the increasing number of agents.

Considering the partial observability and communication constraints, recent advanced works in MARL have decomposed the global Q-value into multiple individual Q-values with a centralized training and decentralized execution (CTDE) paradigm [11, 12]. Despite their effectiveness, these value function factorization methods focus only on the value expectation, which is insufficient to reflect the stochasticity and might lead to sub-optimal policy in the risk-aware scenario. Due to the randomness in the long-term returns, it is more preferable to model the distributions rather than the expectations over returns. Since the distribution intrinsically captures all the stochasticity of a random variable, a proper modeling of it will benefit the value function estimation.

In the past few years, distributional RL has achieved considerable advance in various single-agent domains [13, 14, 15, 16, 17]. Generally, it predicts the distribution over returns instead of a scalar mean value by leveraging either a categorical distribution [13] or a quantile function [15]. Although these works provide some insights for MARL in

terms of stochasticity modeling, they cannot be directly applied to the value-based MARL. Specifically, different from distributional single-agent reinforcement learning (SARL), the key challenge in value-based distributional MARL is how to integrate the individual distributional Q-values into global distributional Q-value under the *Distributional-Individual-Global-Max* (DIGM) principle [18].

To our best knowledge, there exist few works focusing on distributional MARL [18, 19]. RMIX [19] only models the individual Q-values with distributions, which mainly aims to trade off the expectation and variation (risk) in action selection. Only one recent work, DMIX [18], parameterizes both the individual and global Q-values with quantile function. However, DMIX still approximates the expectation of the global distribution with the expectation of the individual ones, suffering the loss of rich information in individual Q-value distributions.

In this work, we propose a novel distributional MARL method, named DQMIX. Our method models both individual Q-value distributions and global Q-value distribution by categorical distribution. In this way, the distributions of individual Q-values capture the uncertainty of the environment from each agent's perspective while the distribution of global Q-value directly approximates the randomness of the total return. To integrate the individual distributions into the global distribution, we define five basic operations, namely Weighting, Bias, Convolution, Projection and Function, which can realize several transformations of the distribution and the combination of multiple distributions. Given these fundamental operations, we design the distribution mixing network, where the parameters of the Weighting and Bias operations are generated by the hypernetwork conditioned on the global state. Furthermore, we provide a theoretical analysis that DQMIX satisfies DIGM principle with respect to the expectation of distribution when the parameters of Weighting operation are non-negative and the Function operation is monotonically increasing.

To evaluate the capability of DQMIX in distribution factorization, we test it on a simple stochastic matrix game, in which the true return distributions are known. The results reveal that the distributions estimated by our method are very close to the true return distributions. Beyond that, we perform experiments on a range of unit micromanagement benchmark tasks in StarCraft II [20]. The results on StarCraft II micromanagement benchmark tasks show that DQMIX significantly outperforms the value-based MARL baselines. Furthermore, we analyze the impact of the hyperparameter—the size of the support set of categorical distribution, and figure out that a size of 51 is sufficient to obtain considerable performance.

2 Background and Related Work

In this section, we introduce some background knowledge for convenience of understanding the content of this paper. First, we discuss the problem formulation of a fully cooperative MARL task. Next, we introduce the concept of deep multi-agent Q-learning. After that, we present the CTDE paradigm and recent representative value function factorization methods in this field. Finally, we describe the concept of distributional RL and summarize the related studies.

2.1 Decentralized Partially Observable Markov Decision Process

A fully cooperative multi-agent task is usually modeled as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [21], following most of recent works in cooperative MARL domain. Dec-POMDP can be described as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{Z}, \mathcal{A}, r, P, O, \mathcal{N}, \gamma \rangle$, where $\mathcal{S}$ is a finite set of global states, $\mathcal{Z}$ is the set of individual observations and $\mathcal{A}$ is the set of individual actions. At each time step, each agent $g_i \in \mathcal{N} := \{g_1, \cdots, g_N\}$ selects an action $a_i \in \mathcal{A}$, forming a joint action $\boldsymbol{a} := [a_i]_{i=1}^N \in \mathcal{A}^N$. This results in a transition on the environment according to the state transition function $P(s'|s, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A}^N \times \mathcal{S} \to [0, 1]$ and the environment returns a joint reward (*i.e.*, team reward) $r(s, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A}^N \to \mathbb{R}$ shared among all agents. $\gamma \in [0, 1)$ is the discount factor. Each agent g_i can only receive an individual and partial observation $o_i \in \mathcal{Z}$, according to the observation function $O(s, i) : \mathcal{S} \times \mathcal{N} \to \mathcal{Z}$. And each agent g_i has an action-observation history $\tau_i \in \mathcal{T} := (\mathcal{Z} \times \mathcal{A})^*$, which can be used to construct its individual policy $\pi_i(a_i|\tau_i) : \mathcal{T} \times \mathcal{A} \to [0, 1]$.

The objective of a fully cooperative multi-agent task is to learn a joint policy $\boldsymbol{\pi} := [\pi_i]_{i=1}^N$ so as to maximize the expected cumulative team reward.

2.2 Deep Multi-Agent Q-Learning

As one of the most naive multi-agent Q-learning algorithms, independent Q-learning (IQL) [22], learns decentralized policy for each agent independently. IQL is simple to implement but suffers from the non-stationarity of environment and may lead to non-convergence of the policy. To this end, many multi-agent Q-learning algorithms [23, 24, 25, 26, 27] are dedicated to learning a global Q-value function:

$$Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) = \mathbb{E}_{s_{1:\infty}, \boldsymbol{a}_{1:\infty}} \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \boldsymbol{a}_0 = \boldsymbol{a} \right], \tag{1}$$

where $\boldsymbol{\tau}$ is the joint action-observation history and r_t is the team reward at time t. Similar to DQN [28], deep multi-agent Q-learning algorithms [23, 24] represent the global Q-value function with a deep neural network parameterized by θ, and then use a replay memory to store the transition tuple $(\boldsymbol{\tau}, \boldsymbol{a}, r, \boldsymbol{\tau}')$. Parameters θ are learnt by sampling a batch of transitions $\{(\boldsymbol{\tau}^{(i)}, \boldsymbol{a}^{(i)}, r^{(i)}, \boldsymbol{\tau}'^{(i)})\}_{i=1}^{n}$ to minimize the following TD error:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \left[\left(y_{tot}^{(i)} - Q_{tot}\left(\boldsymbol{\tau}^{(i)}, \boldsymbol{a}^{(i)}; \theta\right) \right)^2 \right], \tag{2}$$

where $y_{tot}^{(i)} = r^{(i)} + \gamma \max_{\boldsymbol{a}'} Q_{tot}\left(\boldsymbol{\tau}'^{(i)}, \boldsymbol{a}'; \theta^-\right)$. θ^- are the parameters of the target network that are copied every C steps from θ. The joint policy can be derived as: $\boldsymbol{\pi}(\boldsymbol{\tau}) = \arg\max_{\boldsymbol{a}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}; \theta)$.

2.3 CTDE and Value Function Factorization

In cooperative MARL, fully decentralized methods [22, 29] are scalable but suffer from non-stationarity issue. On the contrary, fully centralized methods [30, 31] mitigate the non-stationarity issue but encounter the challenge of scalability, as the joint state-action space grows exponentially with the number of agents. To combine the best of both worlds, a popular paradigm called centralized training with decentralized execution (CTDE) has drawn substantial attention recently. In CTDE, agents take actions based on local observations and are trained to coordinate their actions in a centralized way. During execution, the policy of each agent only relies on its local action-observation history, which guarantees the decentralization.

Recent value-based MARL methods realize CTDE mainly by factorizing the global Q-value function into individual Q-value functions [23, 24, 25]. To ensure the collection of individual optimal actions of each agent during execution are equivalent to the optimal actions selected from global Q-value, value function factorization methods have to satisfy the following IGM [25] condition:

$$\arg\max_{\boldsymbol{a}} Q_{tot}(\boldsymbol{\tau}, \boldsymbol{a}) = \begin{pmatrix} \arg\max_{a_1} Q_1(\tau_1, a_1) \\ \vdots \\ \arg\max_{a_N} Q_N(\tau_N, a_N) \end{pmatrix}. \tag{3}$$

As the first attempt of this stream, VDN [23] represents the global Q-value function as a sum of individual Q-value functions. Considering that VDN ignores the global information during training, QMIX [24] assigns the non-negative weights to individual Q-values with a non-linear function of the global state. These two factorization methods are sufficient to satisfy Eq. (3) but inevitably limit the global Q-value function family they can represent due to their structural constraint. To address the representation limitation, QTRAN proposes to learn a state-value function and transform the original global Q-value function Q_{tot} into a easily factorizable one Q'_{tot} that shares the same optimal actions with Q_{tot} [25]. However, the computationally intractable constraint imposed by QTRAN may lead to poor performance in complex multi-agent tasks.

2.4 Distributional RL

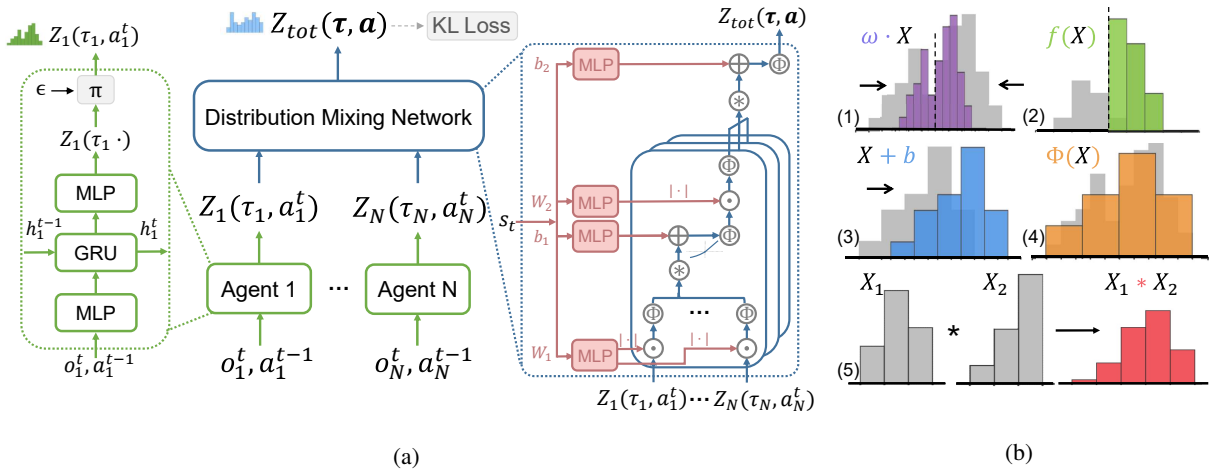
Distributional RL aims to approximate the distribution of returns (*i.e.*, the discounted cumulative rewards) denoted by a random variable $Z(s, a)$, whose expectation is the scalar value function $Q(s, a)$. Similar to the Bellman equation of Q-value function, the distributional Bellman equation can be defined by

$$Z(s, a) \overset{D}{:=} R(s, a) + \gamma Z(s', a'), \tag{4}$$

where $s' \sim P(\cdot \mid s,\ a)$, $a' \sim \pi(\cdot \mid s')$, and $A \overset{D}{=} B$ denotes that random variables A and B have the same distribution. As revealed in Eq. (4), $Z(s, a)$ involves three sources of randomness : the reward $R(s, a)$, the transition $P(\cdot \mid s,\ a)$, and the next-state value distribution $Z(s', a')$ [13]. Then, we have the distributional Bellman optimality operator T^* as follows:

$$T^* Z(s, a) \overset{D}{:=} R(s, a) + \gamma Z(s', \arg\max_{a'} \mathbb{E}\left[Z(s', a')\right]). \tag{5}$$

Based on the distributional Bellman optimality operator, the objective of distributional RL is to reduce the distance between the distribution $Z(s, a)$ and the target distribution $T^* Z(s, a)$. Therefore, a distributional RL algorithm must address two issues: how to parameterize the return distribution and how to choose an appropriate metric to measure the distance between two distributions. To model the return distribution, many RL methods in SARL domain are proposed with promising results [13, 14, 15, 16, 17]. In this paper, we employ the categorical distribution [13] , which represents the distribution with probability masses placed on a discrete set of possible returns, and then minimize the Kullback–Leibler (KL) divergence between the Bellman target and the current estimated return distribution.



(a) (b)

Figure 1: (a) The overall DQMIX architecture. (b) The five basic operations on distribution of random variables. Operations (1)-(5) represent Weighting, Function, Bias, Projection and Convolution, respectively.

3 Method

In this section, we first define five basic operations on the distribution of random variables. Next, we introduce the DQMIX framework, including modeling the distributions of individual Q values, and integrating the individual distributions into the global distribution through a distribution mixing network. Then, we prove that our method satisfies the DIGM principle. Finally, we present the training and execution strategy.

3.1 Basic Operations on Distribution

Let X be a discrete random variable, following a categorical distribution, denoted by $X \sim (M, V_{min}, V_{max}, \mathbf{P})$, where $M \in \mathbb{N}, V_{min}, V_{max} \in \mathbb{R}$; its support is a set of atoms $\{x_j = V_{min} + j\triangle x : 0 \leq j < M\}$, $\triangle x := \frac{V_{max} - V_{min}}{M-1}$ and $\mathbf{P}$ represents the atom probability, *i.e.,*

$$X = x_j \quad w.p. \quad p_j. \tag{6}$$

To apply transformation and combination on random variables with categorical distribution, we define five basic operations as illustrated in Figure 1(b) and elaborate each of them in the following.

Operation 1. [Weighting] Analogous to the scaling operation to a scalar variable, the weighting operation W_w to scale up a discrete random variable by $w \in \mathbb{R}$ is defined as follows:

$$W_w X := w x_j \quad w.p. \quad p_j. \tag{7}$$

The Weighting operation over a distribution can be abbreviated as $w \cdot X$.

Operation 2. [Bias] Analogous to the panning operation to a scalar variable, the bias operation B_b to pan a discrete random variable by $b \in \mathbb{R}$ is defined as follows:

$$B_b X := x_j + b \quad w.p. \quad p_j, \tag{8}$$

which is abbreviated as $X + b$.

Operation 3. [Convolution] To combine the two random variables $X_1 \sim (M_1, V_{1,min}, V_{1,max}, \mathbf{P}_1)$ and $X_2 \sim (M_2, V_{2,min}, V_{2,max}, \mathbf{P}_2)$ with the same atom interval $\triangle x$, we define the convolution operation $Conv(\cdot, \cdot)$ as follows:

$$Conv(X_1, X_2) := x_j^* \quad w.p. \quad p_j^*, \tag{9}$$

$$\text{where } x_j^* = V_{1,min} + V_{2,min} + \triangle x \cdot j, \tag{10}$$

and $0 \leq j < M_1 + M_2 - 1$. Let $\hat{M} := M_1 + M_2 - 1$. If $M_1 \geq M_2$, then

$$p_j^* = \begin{cases} \sum_{k=0}^{j} p_{1,k} p_{2,j-k} & 0 \leq j < M_2 \\ \sum_{k=j-M_2+1}^{j} p_{1,k} p_{2,j-k} & M_2 \leq j < M_1 \\ \sum_{k=j-M_2+1}^{M_1-1} p_{1,k} p_{2,j-k} & M_1 \leq j < \hat{M} \end{cases}. \tag{11}$$

If $M_1 < M_2$, then

$$p_j^* = \begin{cases} \sum_{k=0}^{j} p_{1,j-k} p_{2,j} & 0 \le j < M_1 \\ \sum_{k=j-M_1+1}^{j} p_{1,j-k} p_{2,j} & M_1 \le j < M_2 \\ \sum_{k=j-M_1+1}^{M_2-1} p_{1,j-k} p_{2,j} & M_2 \le j < \hat{M} \end{cases}. \tag{12}$$

$Conv(X_1, X_2)$ is abbreviated as $X_1 * X_2$.

Operation 4. [Projection] Considering projecting random variable distribution of $[x_j]$ to atoms $[\hat{x}_k]$ where $\{\hat{x}_k = \hat{V}_{min} + k\triangle\hat{x} : 0 \le k < K\}$, $\triangle\hat{x} := \frac{\hat{V}_{max} - \hat{V}_{min}}{K-1}$, we define the projection operation $\Phi_{[\hat{x}_k]}$ as follows :

$$\Phi_{[\hat{x}_k]} X \coloneqq \hat{x}_k \quad w.p. \quad \sum_j \left[1 - \frac{|[x_j]_{\hat{V}_{min}}^{\hat{V}_{max}} - \hat{x}_k|}{\triangle\hat{x}} \right]_0^1 p_j, \tag{13}$$

where $[\cdot]_b^a$ bounds its argument in the range $[a, b]$.

Operation 5. [Function] To apply non-linear operation over a random variable, we define the function operation $F_f, f : \mathbb{R} \to \mathbb{R}$ as follows:

$$F_f X \coloneqq f(x_j) \quad w.p. \quad p_j, \tag{14}$$

which is abbreviated as $f(X)$.

3.2 Framework of DQMIX

In this section, we give a detailed illustration of our method, DQMIX. The overview of our proposed network architecture is shown in Figure 1(a).

For each agent, we parameterize its individual Q-value distribution with categorical distribution, which has high flexibility to approximate any shape of the distribution. Assume the support set of the agent's distribution, denoted as $[z] = \{z_1, z_2, \cdots, z_M | z_1 \le \cdots \le z_M, M \in \mathbb{N}\}$, is uniformly distributed over the predefined range $[V_{min}, V_{max}]$, where $V_{min}, V_{max} \in \mathbb{R}$ are the minimum and maximum returns, respectively. Note that all the individual Q-value distributions share the same support set.

Based on the above assumption, learning individual distributions is equivalent to learning the atom probabilities. For each agent, there is one agent network $\theta : \mathcal{Z} \times \mathcal{A} \to \mathbb{R}^M$, that estimates the probabilities of atoms in the support set. Take agent g_i as an example, at each time step, its agent network receives the current individual observation o_i^t and the last action a_i^{t-1} as input, and generates the atom probabilities as follows:

$$Z_i(\tau_i, a_i) = z_j \quad w.p. \quad p_{i,j}(\tau_i, a_i) \coloneqq \frac{e^{\theta_j(\tau_i, a_i)}}{\sum_k e^{\theta_k(\tau_i, a_i)}},$$

where $j \in \{1, \cdots, M\}$ and τ_i denotes the agent g_i's action-observation history. For scalability, parameters θ are shared among all agents.

To integrate the individual Q-value distributions $\{Z_1(\tau_1, a_1), \cdots, Z_N(\tau_N, a_N)\}$ into the global Q-value distribution $Z_{tot}(\boldsymbol{\tau}, \boldsymbol{a})$, we propose the distribution mixing network, which is a multi-layer feed-forward neural network. Within each layer, we firstly apply Weighting operation in Eq. (7) over individual distributions. Given the transformed distributions, we adopt the Projection operation in Eq. (13) to ensure that the individual random variables share the same support set and then leverage the Convolution operation in Eq. (9) to integrate the transformed individual distributions. After that, the Bias operation in Eq. (8) is utilized to shift the integrated distribution. Inspired by the activation function in MLP, we employ the Function operation in Eq. (14) as the non-linear transformation to enhance the network representation capability (except for the last layer). Finally, the Projection operation in Eq. (13) is applied to control the size of the support set.

The sequence of operations of k^{th} layer is formulated as follows:

1. $A_{i,j}^k := w_{i,j}^k \cdot Z_i^k$, (Weighting)
2. $B_{i,j}^k := \Phi_{[z]}(A_{i,j}^k)$, (Projection)
3. $C_j^k := B_{1,j}^k * B_{2,j}^k * \cdots * B_{N_k,j}^k$, (Convolution)
4. $D_j^k := C_j^k + b_j^k$, (Bias)
5. $E_j^k := f\left(D_j^k\right)$, (Function)
6. $Z_j^{k+1} := \Phi_{[z]}(E_j^k)$, (Projection)

a_1 \ a_2	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$
$\mathcal{A}^{(1)}$	$\mathcal{N}(0,1)$	$\mathcal{N}(2,2)$
$\mathcal{A}^{(2)}$	$\mathcal{N}(2,2)$	$\frac{\mathcal{N}(1,2)+\mathcal{N}(8,2)}{2}$

(a)

a_1 \ a_2	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$
$\mathcal{A}^{(1)}$	$\mu = 0.10$ $\sigma^2 = 1.05$	$\mu = 1.88$ $\sigma^2 = 2.40$
$\mathcal{A}^{(2)}$	$\mu = 2.06$ $\sigma^2 = 2.06$	$\mu = 4.70$ $\sigma^2 = 14.03$

(b)

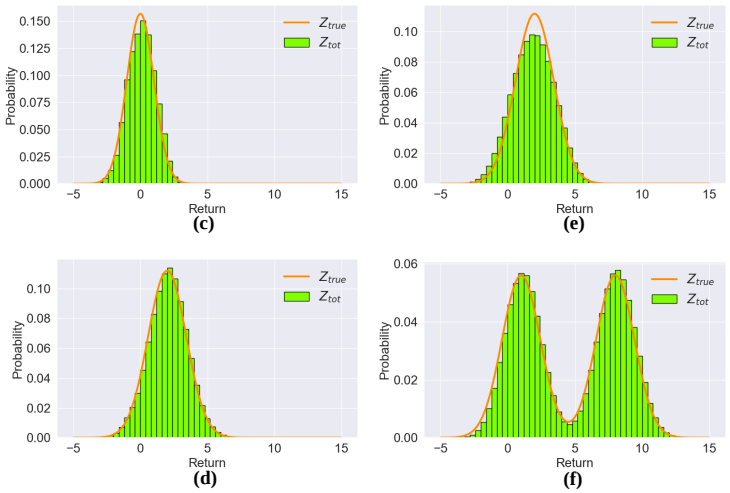


Figure 2: (a) The stochastic matrix game. Each agent $g_i \in \{g_1, g_2\}$ takes an action $a_i \in \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}\}$ and then receives a joint reward that is sampled from the corresponding distribution in the matrix. $\mathcal{N}(\mu, \sigma^2)$ represents the gaussian distribution with mean μ and variance σ^2. The joint action $\langle \mathcal{A}^{(2)}, \mathcal{A}^{(2)} \rangle$ results in a bimodal distribution $\frac{\mathcal{N}(1,2)+\mathcal{N}(8,2)}{2}$, which means that the joint reward is sampled from $\mathcal{N}(1,2)$ or $\mathcal{N}(8,2)$ with equal probability. (b) The learned global Q-value distribution of DQMIX. μ and σ^2 indicate the sampled mean and the sampled variance, respectively. (c)-(f) The total return distributions of joint action $\langle \mathcal{A}^{(1)}, \mathcal{A}^{(1)} \rangle$, $\langle \mathcal{A}^{(2)}, \mathcal{A}^{(1)} \rangle$, $\langle \mathcal{A}^{(1)}, \mathcal{A}^{(2)} \rangle$ and $\langle \mathcal{A}^{(2)}, \mathcal{A}^{(2)} \rangle$, respectively. The orange line represents the true return distribution (*i.e.*, the sampled joint reward distribution). The green histogram shows the global Q-value distribution learned by DQMIX.

where $i \in \{1, \cdots, N_k\}, j \in \{1, \cdots, N_{k+1}\}$, and N_k is the number of input distributions of k^{th} layer. The parameters $w_{i,j}^k \in \mathbb{R}$ and $b_j^k \in \mathbb{R}$ are generated by the k^{th} hypernetwork conditioned on the global state. Note that each hypernetwork that generates $w_{i,j}^k$ is followed by an absolute activation function, which guarantees that the parameters of the Weighting operations are non-negative.

3.3 DIGM Proof

To ensure the consistency between joint and individual greedy action selections, the distribution mixing network must satisfy the DIGM [18] condition, which is formulated as follows:

$$\arg\max_{\boldsymbol{a}} \mathbb{E}\left[Z_{tot}(\boldsymbol{\tau}, \boldsymbol{a})\right] = \begin{pmatrix} \arg\max_{a_1} \mathbb{E}\left[Z_1(\tau_1, a_1)\right] \\ \vdots \\ \arg\max_{a_N} \mathbb{E}\left[Z_N(\tau_N, a_N)\right] \end{pmatrix}.$$

One sufficient condition for the distribution mixing network that meets the DIGM condition is that all operations meet the DIGM condition. The following propositions demonstrate that, under certain conditions, the five basic distribution operations, *i.e.*, weighting, bias, convolution, projection and active function, satisfy the DIGM condition.

Proposition 1. If $w \geq 0$, then

$$\arg\max_a \mathbb{E}[X(a))] = \arg\max_a \mathbb{E}[w \cdot X(a)].$$

Proposition 2.

$$\arg\max_a \mathbb{E}[X(a)] = \arg\max_a \mathbb{E}[X(a) + b].$$

Proposition 3.

$$\arg\max_{a_1, a_2} \mathbb{E}[X_1(a_1) * X_2(a_2)] = \begin{pmatrix} \arg\max_{a_1} \mathbb{E}[X_1(a_1)] \\ \arg\max_{a_2} \mathbb{E}[X_2(a_2)] \end{pmatrix}.$$

Proposition 4. For any atoms $[x]$,

$$\arg\max_a \mathbb{E}[X(a)] = \arg\max_a \mathbb{E}[\Phi_{[x]}(X(a))].$$

Proposition 5. For any monotone increasing function $f : \mathbb{R} \to \mathbb{R}$,

$$\arg\max_a \mathbb{E}[X(a))] = \arg\max_a \mathbb{E}[f(X(a))].$$

The proofs of Prop. 1, 2, and 5 are trivial. Here, we briefly explain the proofs of Prop. 3 and 4. For Prop. 3, we firstly prove that the expectation of the convolution result of the two distributions is equal to the sum of the expectations of two distributions, *i.e.*,

$$\mathbb{E}[X_1 * X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2]. \tag{15}$$

Given Eq. (15), it can be easily inferred that Prop. 3 holds. For Prop. 4, we prove that the expectation of the distribution before and after projection remains unchanged. The detailed proofs are available in Appendix A.

3.4 Training and Inference

In the training phase, each agent g_i interacts with the environment using the ϵ-greedy policy over the expectation of individual Q-value distribution, *i.e.*, $\mathbb{E}[Z_i] = \sum_j p_{i,j} z_j$. The transition tuple $(\boldsymbol{\tau}, \boldsymbol{a}, r, \boldsymbol{\tau}')$ is stored into a replay memory. Then, the learner randomly fetches a batch of samples from the replay memory. The network is optimized by minimizing the sample loss, *i.e.*, the cross-entropy term of KL divergence:

$$D_{\text{KL}}(\Phi T^* Z_{tot}(\boldsymbol{\tau}, \boldsymbol{a}; \theta^-) \;||\; Z_{tot}(\boldsymbol{\tau}, \boldsymbol{a}; \theta)), \tag{16}$$

where $T^* Z_{tot}(\boldsymbol{\tau}, \boldsymbol{a}; \theta^-)$ is the Bellman target according to Eq. (5), θ^- are the parameters of a target network that are periodically copied from θ, and Φ is the projection of Bellman target onto the support of $Z_{tot}(\boldsymbol{\tau}, \boldsymbol{a}; \theta)$.

Since our method satisfies the DIGM condition, the policy learnt during centralized training can be directly applied to execution. During the inference phase, each agent chooses a greedy action a_i at each time step with respect to $\mathbb{E}[Z_i]$.

4 Experiments

In this section, we first present our method on a simple stochastic matrix game to show DQMIX's ability to approximate the true return distribution and the benefits of modeling the value distribution. Then, we carry out experiments on StarCraft Multi-Agent Challenge (SMAC) benchmark environment [20] to compare DQMIX with both the expected value based MARL methods [22, 23, 24, 25] and a distributional MARL model [18]. Finally, we study our method's performance on SMAC in relation to the number of atoms. All of our experiments are conducted on GeForce RTX 2080Ti GPU. The implementation code is available at https://github.com/wudiymy/DQMIX.

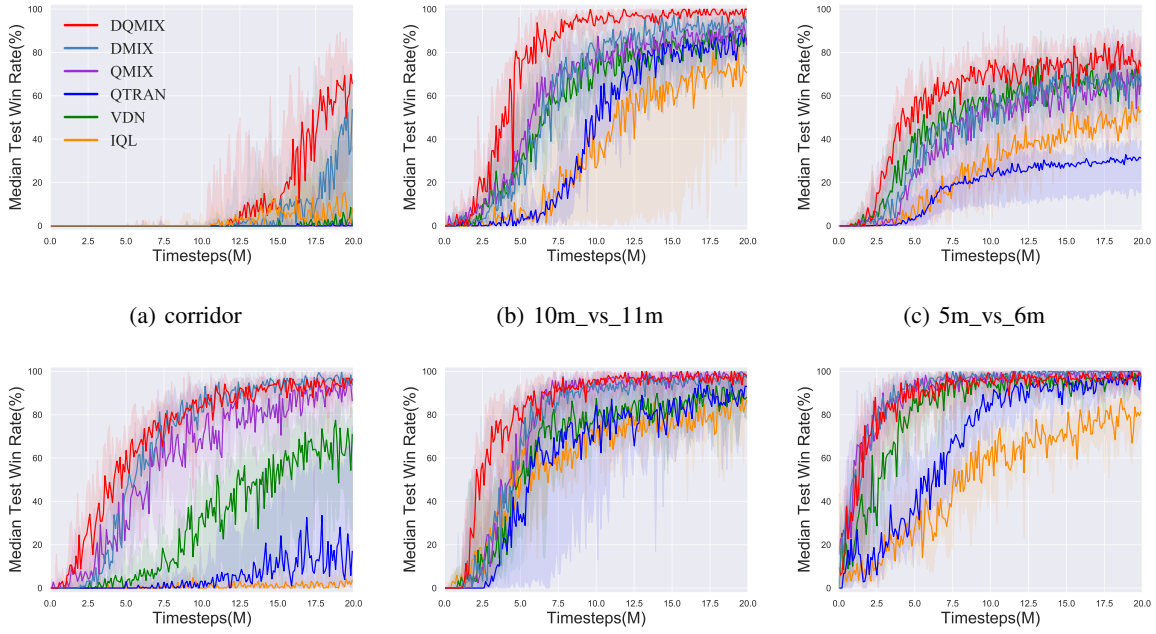
4.1 Stochastic Matrix Game

Matrix game is widely adopted to test the effectiveness of the methods [24, 25, 18]. To demonstrate the capacity of DQMIX to approximate the true return distribution, we design a two-agent stochastic matrix game. Specifically, two agents jointly take actions and will receive a joint reward, which follows a distribution rather than a deterministic value. Here, we set the joint reward to follow a normal distribution or a mixture of normal distributions, as illustrated in Figure 2(a).

We train DQMIX on the matrix game for 2 million steps with full exploration (*i.e.*, ϵ-greedy exploration with $\epsilon = 1$). Full exploration ensures that DQMIX can explore all available game states, such that the representational capacity of the state-action value distribution approximation remains the only limitation [24]. As shown in Figure 2(b), the learned global Q-value distributions are close to the true return distributions in terms of the mean and variance. Moreover, we visualize the true return distribution and the learned distribution of joint action in Figure 2(c)-(f). It can be observed that the estimated distributions are extremely close to the true ones, which can not be achievable by expected value function factorization methods. One advantage of learning the distribution of joint reward is that the agents can flexibly take the optimal actions under the different risk aversion rates.

4.2 StarCraft II Micromanagement Benchmark

We further evaluate DQMIX on the SMAC [20] benchmark, which is based on the popular real-time strategy game StarCraft II. The overall objective is to maximize the test win rate for each battle scenario. There are multiple units in



(a) corridor (b) 10m_vs_11m (c) 5m_vs_6m

(d) 3s5z (e) 2c_vs_64zg (f) 2s3z

Figure 3: The test win rate curves of DQMIX and the comparison algrithms IQL [22], VDN [23], QTRAN [25], QMIX [24], DMIX [18] on the SMAC benchmark. The solid line shows the median win rate and the shadow area represents the min to max win rate on 5 random seeds.

Scenario	Median(%)						Min(%)						Max(%)					
	IQL	VDN	QTRAN	QMIX	DMIX	DQMIX	IQL	VDN	QTRAN	QMIX	DMIX	DQMIX	IQL	VDN	QTRAN	QMIX	DMIX	DQMIX
corridor	7.7	0.2	0.0	0.0	53.7	**65.8**	0.0	0.0	0.0	0.0	0.0	**34.1**	28.5	40.6	36.8	1.6	68.0	**74.8**
10m_vs_11m	70.7	84.3	85.2	83.5	93.8	**100.0**	64.2	72.6	76.2	82.3	78.8	**94.9**	81.3	93.8	93.3	87.5	**100.0**	**100.0**
5m_vs_6m	53.1	64.4	31.3	68.9	65.3	**73.7**	36.9	62.5	15.7	58.3	58.8	**67.7**	58.6	68.9	39.0	86.5	**87.6**	81.2
3s5z	3.6	70.9	16.9	86.5	93.8	**96.5**	0.0	26.7	2.8	85.3	92.1	**92.7**	6.0	82.3	74.6	**100.0**	96.2	96.9
2c_vs_64zg	86.5	88.1	93.1	97.1	97.4	**98.0**	78.1	77.9	84.4	**95.6**	88.8	92.8	88.9	92.6	94.6	**100.0**	**100.0**	**100.0**
2s3z	81.2	97.8	97.8	97.3	97.4	**98.7**	76.2	90.5	95.1	**96.9**	92.7	85.2	87.5	**100.0**	99.7	**100.0**	**100.0**	**100.0**
Average	50.5	67.6	54.0	72.2	83.6	**88.8**	42.6	55.0	45.7	69.7	68.5	**77.9**	58.5	79.7	73.0	79.3	92.0	**92.1**

Table 1: The final median, min and max test win rate comparison with the baselines IQL [22], VDN [23], QTRAN [25], QMIX [24], DMIX [18] of 5 independent runs after 20 million training timesteps. Boldface means the highest median, min and max test win rate across DQMIX and the baselines.

the SMAC environment. Each ally unit is controlled by a decentralized RL agent and all enemy units are controlled by the built-in game AI. We use the default environment settings in SMAC except that the SC2 version is 4.10. We adopt open-source implementations on PyMARL [20] of five value-based MARL baselines: IQL [22], VDN [23], QTRAN [25], QMIX [24] and DMIX [18], where DMIX is the distributional MARL baseline and others are expected MARL baselines. Our method is also based on PyMARL. The common hyperparameters of all methods are set to be the same as that in the default implementation of PyMARL. Similar to C51 [13], we set $M = 51$ and choose $V_{min} = -10$, $V_{max} = 20$ from preliminary experiments on SMAC. We conduct experiments on six SMAC benchmark scenarios: corridor, 10m_vs_11m, 5m_vs_6m, 3s5z, 2c_vs_64zg and 2s3z.

To speed up the data collection, we use parallel runners to generate a total of 20 million timesteps data for each scenario and train network with a batch of 32 episodes after collecting every 8 episodes. Performance is evaluated every 10000 timesteps with 32 test episodes. We present the median, min, and max win rates on 5 random seeds for every method in every scenario.

Figure 3 shows the learning curves of DQMIX and the baselines on six SMAC scenarios, and their final median, min, max performance are presented in Table 1. We can observe that DQMIX consistently outperforms the baselines with faster convergence. As illustrated in Table 1, DQMIX is able to achieve the highest final min and max test win rate on most scenarios and the best median performance across all scenarios. Specifically, DQMIX obtains considerable

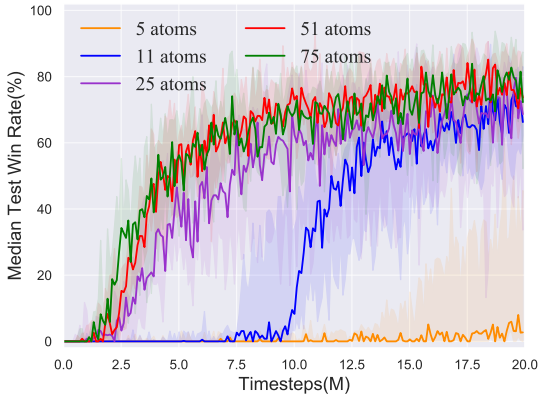


Figure 4: The test win rate curves of DQMIX on 5m_vs_6m with different number of atoms.

improvements on the super hard scenario corridor, where the expected baselines perform very poorly. The leading performance of DQMIX on the SMAC benchmark confirms the effectiveness and importance of the value distribution in approximating multi-agent reinforcement learning.

4.3 Impact of the Atom Number

In this subsection, we conduct an ablation experiment to study DQMIX's performance with respect to the number of atoms, which is the core hyperparameter in our framework. Figure 4 reports the test win rate curves of DQMIX on 5m_vs_6m by varying the number of atoms with the value of $\{5, 11, 25, 51, 75\}$. It can be observed that the performance is extremely poor when the number of atoms is 5 and the test win rate reaches a better performance when the number is greater than 11. Besides, the increasing number of atoms brings about faster convergence. The results indicate that more atoms contribute to better performance and the marginal effect decreases as the number of atoms increases. This is consistent with the fact that, given the fixed value range, a support with more atoms has better expressive power and is more likely to be close to the true distribution. However, the richer support set leads to higher computational cost. Considering that the performance is approaching saturation when the support size is greater than 51, we set the number of atoms to be 51 throughout this work, so as to balance the effectiveness and efficiency.

5 Conclusion

In this paper, we propose DQMIX, a novel distributional value-based MARL method, which explicitly models the stochasticity in long-term returns by categorical distribution. To integrate the individual Q-value distributions into the global one, we design a distribution mixing network, of which the parameters are generated by hypernetwork conditioned on the global state. Theoretical analysis shows that DQMIX satisfies DIGM condition, which ensures the feasibility of decentralized execution. Empirical experiments on the stochastic matrix game and SMAC benchmark demonstrate the efficacy of DQMIX. Beyond that, the ablation study on the number of atoms further indicates that parametrizing the distribution with categorical distribution can balance effectiveness and efficiency.

References

[1] Ruizhuo Song, Frank L Lewis, and Qinglai Wei. Off-policy integral reinforcement learning method to solve nonlinear continuous-time multiplayer nonzero-sum games. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):704–713, 2016.

[2] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[3] Mushuang Liu, Yan Wan, Frank L Lewis, and Victor G Lopez. Adaptive optimal control for stochastic multiplayer differential games using on-policy and off-policy reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(12):5522–5533, 2020.

[4] Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, Yinyuting Yin, Bei Shi, Liang Wang, Tengfei Shi, Qiang Fu, Wei Yang, Lanxiao Huang, and Wei Liu. Towards playing full moba games with deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 621–632, 2020.

[5] Chongjie Zhang and Victor Lesser. Coordinated multi-agent reinforcement learning in networked distributed pomdps. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 25, 2011.

[6] A Pravin Renold and S Chandrakala. MRL-SCSO: multi-agent reinforcement learning-based self-configuration and self-optimization protocol for unattended wireless sensor networks. *Wireless Personal Communications*, 96(4):5061–5079, 2017.

[7] Quan Zhou, Yonggui Li, and Yingtao Niu. Intelligent anti-jamming communication for wireless sensor networks: A multi-agent reinforcement learning approach. *IEEE Open Journal of the Communications Society*, 2:775–784, 2021.

[8] Pengyuan Zhou, Xianfu Chen, Zhi Liu, Tristan Braud, Pan Hui, and Jussi Kangasharju. DRLE: Decentralized reinforcement learning at the edge for traffic light control in the iov. *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2262–2273, 2021.

[9] Yanan Wang, Tong Xu, Xin Niu, Chang Tan, Enhong Chen, and Hui Xiong. STMARL: A spatio-temporal multi-agent reinforcement learning approach for cooperative traffic light control. *IEEE Transactions on Mobile Computing*, 2020.

[10] Tong Wu, Pan Zhou, Kai Liu, Yali Yuan, Xiumin Wang, Huawei Huang, and Dapeng Oliver Wu. Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(8):8243–8256, 2020.

[11] Jakob N Foerster, Yannis M Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676*, 2016.

[12] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2974–2982, 2018.

[13] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 70, pages 449–458, 2017.

[14] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2892–2901, 2017.

[15] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1096–1105, 2018.

[16] Derek Yang, Li Zhao, Zichuan Lin, Tao Qin, Jiang Bian, and Tie-Yan Liu. Fully parameterized quantile function for distributional reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 6190–6199, 2019.

[17] Jingliang Duan, Yang Guan, Shengbo Eben Li, Yangang Ren, Qi Sun, and Bo Cheng. Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[18] 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 the International Conference on Machine Learning (ICML)*, pages 9945–9954, 2021.

[19] 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.

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

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

[22] M. Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1993.

[23] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 2085–2087, 2018.

[24] 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 the International Conference on Machine Learning (ICML)*, pages 4292–4301, 2018.

[25] 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 the International Conference on Machine Learning (ICML)*, pages 5887–5896, 2019.

[26] 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.

[27] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: Duplex dueling multi-agent q-learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[29] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PLOS ONE*, 12(4), 2017.

[30] Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 227–234, 2002.

[31] Jelle R. Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(65):1789–1828, 2006.

A Proofs

In this section, we provide the proofs of the propositions discussed in this paper.

Proposition 1. If $w \geq 0$, then

$$\arg\max_a \mathbb{E}[X(a))] = \arg\max_a \mathbb{E}[w \cdot X(a)].$$

Proof.

$$\begin{aligned}
&\arg\max_a \mathbb{E}[w \cdot X(a)] \\
=&\arg\max_a \sum_j p_j(wx_j) \\
\overset{(*)}{=}&\arg\max_a \sum_j p_j x_j \\
=&\arg\max_a \mathbb{E}[X(a))],
\end{aligned}$$

where (*) is satisfied because $w \geq 0$. □

Proposition 2.

$$\arg\max_a \mathbb{E}[X(a)] = \arg\max_a \mathbb{E}[X(a) + b].$$

Proof.

$$\begin{aligned}
&\arg\max_a \mathbb{E}[X(a) + b] \\
=&\arg\max_a \sum_j p_j(x_j + b) \\
=&\arg\max_a \sum_j p_j x_j \\
=&\arg\max_a \mathbb{E}[X(a))].
\end{aligned}$$

□

Proposition 3.

$$\arg\max_{a_1,a_2} \mathbb{E}[X_1(a_1) * X_2(a_2)] = \begin{pmatrix} \arg\max_{a_1} \mathbb{E}[X_1(a_1)] \\ \arg\max_{a_2} \mathbb{E}[X_2(a_2)] \end{pmatrix}.$$

Proof.

$$\begin{aligned}
&\mathbb{E}[X_1(a_1) * X_2(a_2)] \\
=&\sum_{i=0}^{M_1-1}\sum_{j=0}^{M_2-1} p_{1,i}p_{2,j}(x_{1,i} + x_{2,j}) \\
=&\sum_{i=0}^{M_1-1} p_{1,i}\left(\sum_{j=0}^{M_2-1} p_{2,j}x_{1,i} + \sum_{j=0}^{M_2-1} p_{2,j}x_{2,j}\right) \\
=&\sum_{i=0}^{M_1-1} p_{1,i}\Big(x_{1,i} + \mathbb{E}[X_2(a_2)]\Big) \\
=&\sum_{i=0}^{M_1-1} p_{1,i}x_{1,i} + \mathbb{E}[X_2(a_2)] \\
=&\mathbb{E}[X_1(a_1)] + \mathbb{E}[X_2(a_2)].
\end{aligned}$$

Given the above equation,

$$\begin{aligned}&\arg\max_{a_1,a_2}\mathbb{E}[X_1(a_1)*X_2(a_2)]\\=&\arg\max_{a_1,a_2}(\mathbb{E}[X_1(a_1)]+\mathbb{E}[X_2(a_2)])\\=&\begin{pmatrix}\arg\max_{a_1}\mathbb{E}[X_1(a_1)]\\\arg\max_{a_2}\mathbb{E}[X_2(a_2)]\end{pmatrix}.\end{aligned}$$

□

Proposition 4. For any atoms $[x]$,

$$\arg\max_a \mathbb{E}[X(a)] = \arg\max_a \mathbb{E}[\Phi_{[x]}(X(a))].$$

Proof. Considering projecting random variable distribution of $[x_j]$ to atoms $[\hat{x}_k]$, let's assume that the atom range after projection $[\hat{V}_{min}, \hat{V}_{max}]$ is enough to cover all atoms before projection, *i.e.,* $\forall\, 0 \le j < M$, $x_j \in [\hat{V}_{min}, \hat{V}_{max}]$.

$\forall\, x_j$, $\exists\, 1 \le k_j < K$, *s.t.* $\hat{x}_{k_j-1} \le x_j < \hat{x}_{k_j}$, *i.e.,* the immediate neighbours of x_j is $\hat{x}_{k_j-1}$ and $\hat{x}_{k_j}$. According to the definition of projection operation, the probability p_j in x_j is disassembled as $\frac{\hat{x}_{k_j}-x_j}{\triangle\hat{x}}p_j$ in $\hat{x}_{k_j-1}$ and $\frac{x_j-\hat{x}_{k_j-1}}{\triangle\hat{x}}p_j$ in $\hat{x}_{k_j}$.

$$\begin{aligned}&\mathbb{E}[\Phi_{[x]}(X(a))]\\=&\sum_k(\sum_j\left[1-\frac{|[x_j]_{\hat{V}_{min}}^{\hat{V}_{max}}-\hat{x}_k|}{\triangle\hat{x}}\right]_0^1 p_j)\hat{x}_k\\=&\sum_j(\sum_k\left[1-\frac{|[x_j]_{\hat{V}_{min}}^{\hat{V}_{max}}-\hat{x}_k|}{\triangle\hat{x}}\right]_0^1 p_j\hat{x}_k)\\=&\sum_j(\frac{\hat{x}_{k_j}-x_j}{\triangle\hat{x}}p_j\cdot\hat{x}_{k_j-1}+\frac{x_j-\hat{x}_{k_j-1}}{\triangle\hat{x}}p_j\cdot\hat{x}_{k_j})\\=&\sum_j\Big(\frac{p_j}{\triangle\hat{x}}[(\hat{x}_{k_j}-x_j)\cdot(\hat{x}_{k_j}-\triangle\hat{x})\\&\qquad+(x_j-\hat{x}_{k_j}+\triangle\hat{x})\cdot\hat{x}_{k_j}]\Big)\\=&\sum_j p_jx_j\\=&\mathbb{E}[X(a)].\end{aligned}$$

Therefore,

$$\arg\max_a \mathbb{E}[\Phi_{[x]}(X(a))] = \arg\max_a \mathbb{E}[X(a)].$$

□

Proposition 5. For any monotone increasing function $f : \mathbb{R} \to \mathbb{R}$,

$$\arg\max_a \mathbb{E}[X(a))] = \arg\max_a \mathbb{E}[f(X(a))].$$

Proof.

$$\begin{aligned}&\arg\max_a\mathbb{E}[f(X(a))]\\=&\arg\max_a\sum_j p_j(f(x_j))\\\overset{(*)}{=}&\arg\max_a\sum_j p_jx_j\\=&\arg\max_a\mathbb{E}[X(a)],\end{aligned}$$

where (*) is satisfied because of the monotonicity of function f. □