

Reducing Overestimation in Value Mixing for Cooperative Deep Multi-Agent Reinforcement Learning

Zipeng Fu¹^a, Qingqing Zhao², and Weinan Zhang³

¹*Department of Computer Science, University of California, Los Angeles, United States*

²*Department of Physics, The University of Hong Kong, Hong Kong, China*

³*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China*
fu-zipeng@engineering.ucla.edu, cyanzhao@hku.edu, wnzhang@sjtu.edu.cn

Keywords: Reinforcement Learning, Multi-Agent Systems, Q-Learning, Deep Q-Networks, Value Function Approximation, Centralized Training Decentralized Execution

Abstract: Since the debut of Deep Q-Network (DQN), numerous researches have been conducted to integrate Deep Neural Networks (DNN) with Reinforcement Learning (RL). The tremendous expressive power of DNNs empowers Reinforcement Learning, which was mostly only functional in simple discrete / tabular settings, to solve complex problems in continuous and high-dimensional settings. Recently, Deep RL is adapted to Multi-Agent Systems (MAS). In many real-world scenarios, a group of agents, each with generally different local observations, needs to cooperate to achieve a collective reward. Despite decentralized execution, global state information can be shared among agents in a laboratory setting during the rehearsal period. We propose double QMIX, an end-to-end multi-agent Q-learning method with reduction of value overestimation, that trains decentralized agents' policies in a centralized setting. The centralized Q-value is computed from each agent's utility in a non-linear and anti-overestimated fashion. We provide the theoretical analysis of the reason why traditional DQN training methods lead to significant value overestimation in multi-agent setting, and how double QMIX solves this problem is explained. We also evaluate double QMIX in StarCraft II micromanagement environment to show a better performance, compared with other state-of-the-art value-based multi-agent reinforcement learning methods.


1 INTRODUCTION

Reinforcement learning (RL) is the study of how to learn an (nearly) optimal policy given observed rewards in an environment (Russell and Norvig, 2016). Used to be largely restricted in environments with small-scale discrete spaces (Kaelbling et al., 1996; Tesauro, 1994; Singh and Bertsekas, 1997; Matarić, 1997), RL, paired with Deep Neural Networks as non-linear function approximators, levels up and is capable of solving problems with high-dimensional discrete or continuous state and action spaces (Mnih et al., 2015; Mnih et al., 2016; Levine et al., 2016; Li et al., 2016).

Recently, Deep Multi-Agent Reinforcement Learning (Deep MARL) emerges as a promising way to address complicated coordination problems, in which more than one autonomous agent is involved (Jaderberg et al., 2019; OpenAI, 2018).

In many such multi-agent coordination problems, the framework of centralized training and decentralized execution (CTDE) is commonly applicable (Oliehoek et al., 2016; Kraemer and Banerjee, 2016). Agents act in an environment separately, and each agent is only able to perceive its partial observation of the environment (eg. it may not observe where other agents are and what other agents can observe). The decentralized policies condition on ones' own observations, so there is no requirement of communicating with other agents to construct an aggregate observation. During training, agents can learn their decentralized policies in a simulated setting with full state information supplied and inter-agent communication bandwidth constraints removed.

This paradigm significantly alleviates the problem of exponential grow in observation and action space with the number of agents when the centralized execution is used. Additionally, unlike centralized execution methods, the problem of 'lazy agent' (Hausknecht, 2016) can be avoided.

^a <https://orcid.org/0000-0002-7462-7215>

Most of the state-of-the-art methods for CTDE include a centralized action-value function (Q -value function) as an indispensable module for centralized training. One approach is the extension of the single-agent RL method, policy gradient (Williams, 1992), and its variant for variance reduction, Actor-Critic (Sutton et al., 2000; Konda and Tsitsiklis, 2000). COMA (Foerster et al., 2018) and MADDPG (Lowe et al., 2017) are two notable examples, where the centralized critic(s) with all related information about state guides the learning process of decentralized policies of all agents.

Another approach is to equip multi-agent Q -learning with extra state information during training. VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018) implement such idea by mixing individual utility values to obtain the centralized action value Q_{MIX} . QTRAN (Son et al., 2019) extends the representation power of the network mixing individual utilities; nevertheless, QTRAN fails to utilize the full state information when it's available. This shortcoming makes it unsuitable for problems where the union of all agents' observation contains much less information than the full state. Our test bench, StarCraft II micromanagement (Samvelyan et al., 2019), is one of the examples.

Despite the superb performance of QMIX and VDN in complex multi-agent coordination tasks, it is well-known that the vanilla Q -learning methods (Watkins and Dayan, 1992; Mnih et al., 2015) yield overestimation of learned action values in both tabular settings (Thrun and Schwartz, 1993; Hasselt, 2010) and complex cases where generalized function approximators are used (Van Hasselt et al., 2016). Thrun and Schwartz give illustrative examples showing overestimations result in sub-optimal policies in single-agent tabular settings. Van Hasselt et al. provide a comprehensive study of overestimations in single-agent DQN (Mnih et al., 2015) and experimentally show the overestimations lead to sub-optimality. In Atari 2600 games, provided by Arcade Learning Environment (Bellemare et al., 2013), DQN is outperformed by double DQN, which significantly reduces overestimations by decorrelating greedy action selection and target action value evaluation.

In this work, we formally prove that overestimations exist in multi-agent value mixing Q -learning under certain conditions. We show that the idea presented in double DQN can be generalized to multi-agent settings. We present the adapted method, double QMIX, and illustrate its performance gains against other value mixing methods in StarCraft II micromanagement tasks.

Table 1: multi-agent coordination setting.

Notation	Description	Formulation
S	state space	$s \in S$
A	agent indices	$a \in A \triangleq \{1, \dots, n\}$
U	agent action space	$u^a \in U$
\mathbf{U}	joint action space	$\mathbf{u} \in \mathbf{U} \triangleq U^n$
P	transition function	$P(s_{t+1} s_t, \mathbf{u}) : S \times \mathbf{U} \rightarrow \Delta_S$
r	shared reward function	$r(s, \mathbf{u}) : S \times \mathbf{U} \rightarrow \mathbb{R}$
Z	observation space	$z \in Z$
O	observation function	$z_t = O(s_t, a)$
γ	discounted factor	$\gamma \in (0, 1]$
τ^a	agent observation-action history	$\tau^a \in T$ $T \triangleq (Z \times U)^*$
τ	joint observation-action history	$\tau = (\tau^1, \dots, \tau^n)$ $\tau \in \mathcal{T} \triangleq T^n$
π^a	agent decentralized policy	$\pi^a(u^a \tau^a) : T \rightarrow \Delta_U$
R_t	discounted return	$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$
Q^π	joint action-value function	$Q^\pi(s_t, \mathbf{u}_t) = \mathbb{E}_{\pi, P}[R_t s_t, \mathbf{u}_t]$
Q^*	optimal Q	$\max_{\pi} Q^\pi$
π^*	optimal π	$\arg\max_{\pi} Q^\pi$
θ^*	parameters of π^*	

2 BACKGROUND & RELATED WORKS

A multi-agent coordination setting can be modelled by a Dec-POMDP (Oliehoek, 2012), which is a tuple $G = \langle S, A, U, P, r, Z, O, n, \gamma \rangle$. Each agent has a different partial observation z_t about the current state s_t after action \mathbf{u}_{t-1} is chosen and a transition in the environment from previous state s_{t-1} leads to the current state. Every agent's policy conditions on its own observation-action history. This decentralized policy is used during decentralized execution, but the training is centralized. To realize the CTDE idea, the learning algorithm of double QMIX has access to s and all individual observation action trajectories τ^a .

Each agent's goal is to maximize its own discounted return by selecting actions based on its policy and observation-action history. To enforce the cooperation between agents, a shared reward function is used. Detailed notations and formulations refer to Table 1.

2.1 Deep Q -Learning

Q -learning (Watkins and Dayan, 1992) is a form of off-policy temporal difference learning (TD Learn-

ing) (Sutton, 1988). Through bootstrapping with external reward feedback from the environment, Q -learning intends to learn an optimal action value for every state-action pair. The expressive power of deep neural network (Raghu et al., 2017) makes it a suitable model for action value approximator. Deep Q -Network (Mnih et al., 2015), with parameters θ , is proposed to scale Q -learning to high-dimensional state and action space. Two important changes are made in the learning algorithm of deep Q -learning. Uniform sampling from the experience replay (Lin, 1992) reduces the correlation of experiences in a mini-batch, so stochastic gradient descent (SGD) optimizers can work properly. In addition to the online network constantly updating parameters, a target network, with parameters θ^- is used for stable training. The parameters of a target network are periodically copied from θ . The target value is

$$y_t^{\text{DQN}} = r_t + \gamma \max_{u_{t+1}} Q(s_{t+1}, u_{t+1}; \theta^-). \quad (1)$$

$$\mathcal{L}(\theta) = \mathbb{E} \left[(y_t^{\text{DQN}} - Q(s_t, u_t; \theta))^2 \right] \quad (2)$$

is the loss function to minimize.

2.2 Double Deep Q -Learning

Double deep Q -learning (Van Hasselt et al., 2016) addresses the overestimation problem in target value y_t^{DQN} in Equation 1. Double DQN uses the online network (θ) to evaluate the greedy policy (the max operator to select the best action for next time step in Equation 1), but target network (θ^-) is used for computing the selected action's Q -value. Formally, the target value is

$$y_t^{\text{DoubleDQN}} = r_{t+1} + \gamma Q(s_{t+1}, \arg \max_{u_{t+1}} Q(s_{t+1}, u_{t+1}; \theta); \theta^-). \quad (3)$$

The update process of parameters in the target network stays unchanged.

2.3 Deep Recurrent Q -Learning

DQN and double DQN are formulated in the fully observable Markov decision process (MDP). In partially observable settings, estimating action value solely based on current observation can be arbitrarily ineffective due to $Q(o_t, a_t) \neq Q(s_t, a_t)$. Deep recurrent Q -network (DRQN) (Hausknecht and Stone, 2015) uses recurrent neural networks (eg. LSTM (Hochreiter and Schmidhuber, 1997), GRU (Chung et al., 2015)) to model recurrency in observation-action history. Experiments show that this formulation provides a better estimation of underlying (latent) state.

2.4 Value Mixing (Vanilla QMIX Methods)

Unlike independent Q -learning (Tan, 1993), which decomposes multi-agent Q -learning into a collection of simultaneous single-agent Q -learning, value mixing methods aim to learn a joint action-value $Q_{\text{MIX}}(\tau, \mathbf{u})$ within linear complexity regarding the number of agents n . This approach is also different from joint action learners (JAL) (Claus and Boutilier, 1998), where a centralized agent control actions of all agents. This results in intractability of action space and observation space with exponential size in the number of agents n .

Since agents need to act in decentralized fashion after training, it is reasonable for Q_{MIX} to condition on each agent's action value function, on which greedy policy in decentralized execution is based.

$$Q_{\text{MIX}}(\tau, \mathbf{u}) = f \left(\begin{bmatrix} Q_1(\tau^1, u^1; \theta^1) \\ \vdots \\ Q_n(\tau^n, u^n; \theta^n) \end{bmatrix}; \theta^{\text{MIX}} \right), \quad (4)$$

where f is the *mixing function*, and Q_a denotes a utility function (Guestrin et al., 2002). Q_a is not action value function because it does not estimate an expected discounted return R_t . Instead, Q_{MIX} models R_t . However, suitable f and Q -learning algorithm will promote Q_a to be properly scaled expected discounted return. For notation simplicity, we refer both Q_{MIX} and Q_a as action value functions.

2.4.1 Value Decomposition Networks

The value decomposition network (VDN) (Sunehag et al., 2018) represents f in Equation 4 as a unweighted arithmetic mean function:

$$Q_{\text{MIX}}^{\text{VDN}}(\tau, \mathbf{u}) = \sum_{a=1}^n Q_a(\tau^a, u^a; \theta^a). \quad (5)$$

Then θ^{MIX} are fixed constants. Q_a is modelled by a recurrent neural network, analogous to DRQN. The loss function is equivalent to Equation 2, in which $\{\theta^a\}_{a=1, \dots, n}$ is trained. The 2nd term of target value of $Q_{\text{MIX}}^{\text{VDN}}$ (analogous to Equation 1) can simply be the summation of every $\max Q_a$, which means linear computation complexity in n .

In this structure, the centralized training setting is required for the simultaneous update of θ^a for every agent a . Nevertheless, neither constant θ^{MIX} are flexible representations for the mixing function f , nor available state information in training environment that is not present in any agent's observation (eg. the HP values of opponents) is efficiently used.

2.4.2 Monotonic Value Function Factorisation

The monotonic value function factorisation (mono QMIX) (Rashid et al., 2018) addresses the aforementioned problems of VDN by modeling f with a special neural network. A monotonic relationship is ensured:

$$\forall a \in A, \frac{\partial Q_{\text{MIX}}^{\text{MONO}}}{\partial Q_a} \geq g_a, \text{ and } g_a \geq 0, \quad (6)$$

so

$$\max_{\mathbf{u}} Q_{\text{MIX}}^{\text{MONO}}(\tau, \mathbf{u}; \Theta) = f_{\text{NN}} \left(\begin{bmatrix} \max_{u^1} Q_1(\tau^1, u^1; \theta^1) \\ \vdots \\ \max_{u^n} Q_n(\tau^n, u^n; \theta^n) \end{bmatrix}; \theta^{\text{MIX}} \right). \quad (7)$$

As a result, the maximization of $Q_{\text{MIX}}^{\text{MONO}}$ can be computed in linear time in the number of agents n .

The replay buffer and target network are used. The target value function and loss function are similar to Equation 1 and 2:

$$y_t^{\text{MIX}} = r_t + \gamma \max_{\mathbf{u}_{t+1}} Q_{\text{MIX}}^{\text{MONO}}(\tau_{t+1}, \mathbf{u}_{t+1}; \Theta^-), \quad (8)$$

$$\mathcal{L}(\Theta) = \mathbb{E} \left[(y_t^{\text{MIX}} - Q_{\text{MIX}}^{\text{MONO}}(\tau_t, \mathbf{u}_t; \Theta))^2 \right], \quad (9)$$

where

$$\max_{\mathbf{u}_{t+1}} Q_{\text{MIX}}^{\text{MONO}}(\tau_{t+1}, \mathbf{u}_{t+1}; \Theta^-) = f_{\text{NN}} \left(\begin{bmatrix} \max_{u_{t+1}^1} Q_1(\tau_{t+1}^1, u_{t+1}^1; \theta^{1-}) \\ \vdots \\ \max_{u_{t+1}^n} Q_n(\tau_{t+1}^n, u_{t+1}^n; \theta^{n-}) \end{bmatrix}; \theta^{\text{MIX}-} \right). \quad (10)$$

The *mixing network* f_{NN} is a feed-forward network that takes Q_a as inputs and outputs Q_{MIX} . The monotonicity constraints in Equation 6 is satisfied by restricting all weights in θ^{MIX} to be non-negative through linearly rectification (ReLU). Biases have no such constraint. θ^{MIX} are outputs of a set of hypernetworks (Ha et al., 2017), each of which takes state s as the input.

3 OVERESTIMATION IN MULTI-AGENT VALUE MIXING Q-LEARNING

In single-agent setting, if the estimated action values contain independent noise uniformly distributed

in $[-\epsilon, \epsilon]$, then we have the expected overestimation of target value (Thrun and Schwartz, 1993)¹

$$\mathbb{E}[Z] \triangleq \mathbb{E}[y_t - y_t^*], \quad (11)$$

$$0 \leq \mathbb{E}[Z] \leq \gamma \frac{m-1}{m+1}, \quad (12)$$

where m is the size of action space $|A|$, and y^* is the true optimal target value, satisfying the Bellman condition (Bellman, 1966):

$$y_t^* \triangleq Q^*(s_t, u_t; \theta^*) = r_t + \gamma \max_{u_{t+1}} Q^*(s_{t+1}, u_{t+1}; \theta^*). \quad (13)$$

In this section, we show that the overestimation of target action value also exists in value mixing methods for partially observable multi-agent Q -learning, regardless of the source of errors.

Theorem 1. Assume for each agent $Q_a^*(\tau^a, u) = V_a^*(\tau^a)$ and the errors of estimated action values form an independent uniform distribution in range $[-\epsilon, \epsilon]$. Then the expected overestimation is at least $\gamma g \epsilon n \frac{m-1}{m+1}$ under linear condition on f , where g is the lowest bounding gradient for g_a .

Proof of Theorem 1. We first define the expected overestimation:

$$\begin{aligned} \mathbb{E}[Z] &\triangleq \mathbb{E}[y_t^{\text{MIX}} - y_t^{\text{MIX}*}] \\ &= \mathbb{E} \left[\left(r_t + \gamma f \left(\begin{bmatrix} \max_{u_{t+1}^1} Q_1(\tau_{t+1}^1, u_{t+1}^1) \\ \vdots \\ \max_{u_{t+1}^n} Q_n(\tau_{t+1}^n, u_{t+1}^n) \end{bmatrix} \right) \right) \right. \\ &\quad \left. - \left(r_t + \gamma f \left(\begin{bmatrix} V_1^*(\tau_{t+1}^1) \\ \vdots \\ V_n^*(\tau_{t+1}^n) \end{bmatrix} \right) \right) \right] \end{aligned} \quad (14)$$

Define the normally distributed noise as a random variable $Y_a^{s,u}$ conditioning on states, actions and agent indices:

$$\begin{aligned} Y_a^{\tau,u} &\triangleq Q_a(\tau, u) - Q_a^*(\tau, u) \\ &= Q_a(\tau, u) - V_a^*(\tau). \end{aligned} \quad (15)$$

So the probability density of the error is

$$P_{Y_a^{\tau,u}}(y) = \begin{cases} \frac{1}{2\epsilon}, & \text{if } y \in [-\epsilon, \epsilon] \\ 0, & \text{otherwise.} \end{cases} \quad (16)$$

In theory, weights in f can be arbitrarily large. Here we assume that f is a linear model: $\frac{\partial Q_{\text{MIX}}}{\partial Q_a} = g_a$. Denote the lower bound gradient $\min_a g_a$ as g , where

¹As some readers may notice, we do not differentiate random variables, samples, and sometimes function of the same subject by separate symbols (eg. reward r), due to otherwise too many symbols to keep track. This is a common convention in RL literature.

$g \geq 0$. For VDN, $g = 1$. Continuing from Equation 14, we get:

$$\mathbb{E}[Z] = \gamma \mathbb{E} \left[\sum_{a \in A} g_a \left(\max_{u_{t+1}^a} Q_a(\tau_{t+1}^a, u_{t+1}^a) - V_a^*(\tau_{t+1}^a) \right) \right] \quad (17)$$

$$\geq \gamma g \sum_{a \in A} \mathbb{E} \left[\max_u Y_a^{\tau, u} \right]. \quad (18)$$

Equation 17 and 18 are based on the linearity of expectation. Notice that errors $Y_a^{\tau, u}$ are not necessarily independent for different agent a . Since $Y_a^{\tau, u}$ are independent for different u ,

$$P(\max_u Y_a^{\tau, u} \leq y) = P(Y_a^{\tau, 1} \leq y \wedge \dots \wedge Y_a^{\tau, m} \leq y) \quad (19)$$

$$= \prod_{u=1}^m P(Y_a^{\tau, u} \leq y) \quad (20)$$

By calculating cumulative distribution function (CDF) from Equation 16,

$$P(\max_u Y_a^{\tau, u} \leq y) = \begin{cases} 0, & \text{if } y \leq -\epsilon \\ \left(\frac{\epsilon+y}{2\epsilon}\right)^m, & \text{if } y \in (-\epsilon, \epsilon] \\ 1, & \text{otherwise.} \end{cases} \quad (21)$$

Differentiate the both sides of Equation 21. We get the probability density function

$$P_{\max_u Y_a^{\tau, u}}(y) = \begin{cases} \frac{m}{2\epsilon} \left(\frac{1}{2} + \frac{y}{2\epsilon}\right)^{m-1}, & \text{if } y \in [-\epsilon, \epsilon] \\ 0, & \text{otherwise.} \end{cases} \quad (22)$$

This yields the expected overestimation of each agent,

$$\mathbb{E} \left[\max_u Y_a^{\tau, u} \right] = \int_{-\epsilon}^{\epsilon} y \frac{m}{2\epsilon} \left(\frac{1}{2} + \frac{y}{2\epsilon}\right)^{m-1} dy \quad (23)$$

$$= \epsilon \frac{m-1}{m+1}. \quad (24)$$

Hence, from Equation 18 and 24,

$$\mathbb{E}[Z] \geq \gamma g \epsilon n \frac{m-1}{m+1}. \quad (25)$$

4 DOUBLE QMIX

Inspired by double DQN, we proposed a simple change to the target value to reduce the overestimation. Specifically,

$$y_t^{\text{MIX}} = r_t + \gamma Q_{\text{MIX}}(\tau_{t+1}, \mathbf{u}_{t+1}'; \Theta^-), \quad (26)$$

where

$$\mathbf{u}_{t+1}' = \underset{\mathbf{u}_{t+1}}{\operatorname{argmax}} Q_{\text{MIX}}(\tau_{t+1}, \mathbf{u}_{t+1}; \Theta). \quad (27)$$

For *mixing functions* with monotonic constraints,

$$\mathbf{u}_{t+1}' = \begin{bmatrix} \underset{u_{t+1}^1}{\operatorname{argmax}} Q_1(\tau_{t+1}^1, u_{t+1}^1; \theta^1) \\ \vdots \\ \underset{u_{t+1}^n}{\operatorname{argmax}} Q_n(\tau_{t+1}^n, u_{t+1}^n; \theta^n) \end{bmatrix}. \quad (28)$$

Compared with Equation 8, Equation 26 decomposes the greedy action selector (the max operator) into 2 parts. Double QMIX selects the next time step action based on the online network (Θ), but the evaluation of action values uses the target network (Θ^-). The selection and evaluation and action values are decorrelated due to the general discrepancy in parameters between online network and target network. This approach prevents the target values from being always evaluated based the largest next-step action value given an agent's observation-action trajectory.

Double QMIX requires no additional network structures or parameters, and the training pipeline, including uniform sampling from replay buffer, back-propagation to reduce sampled TD-errors and periodic updates of target network's parameters, is unchanged. In addition, the training time complexity and space complexity are equivalent to those of QMIX methods using the vanilla target values.

Since immediately after each copying of the parameters of online network to the target network, both networks are essentially identical, the decorrelation effect is elusive during that period. A common approach is reducing the update frequency (Van Hasselt et al., 2016).

5 EXPERIMENTS

We conduct two experiments: one for visualizing the overestimation in vanilla QMIX and the effectiveness of double QMIX in reducing the overestimation of target values, and the other focusing on the performance gain of using double QMIX, compared with vanilla QMIX methods, in StarCraft II micromanagement tasks (Samvelyan et al., 2019; Vinyals et al., 2017).

5.1 Overestimation Visualization

We plot two heatmaps within the same scale respectively for vanilla QMIX and double QMIX in Figure 1. We made the same assumption as in Section 3. Namely, $Q_a^*(\tau^a, u) = V_a^*(\tau^a)$, and the errors of estimated action values form an independent uniform distribution in range $[-\epsilon, \epsilon]$. We assume linear model f and g_a are all 1, so f is a VDN structure. Also due to the monotonicity constraint on QMIX,

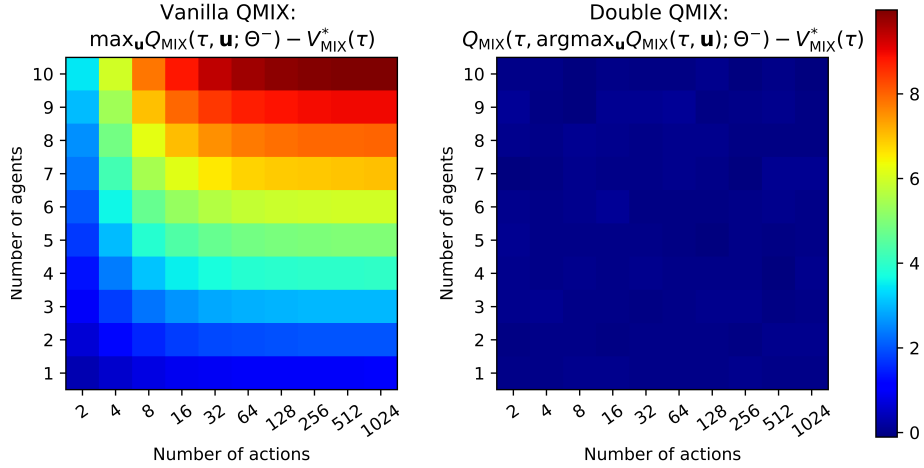


Figure 1: Heatmaps of action value overestimation in vanilla QMIX and double QMIX. The vanilla QMIX method results in large overestimation in settings with many agents or large action space, whereas double QMIX manages to decorrelate the action selection and action value evaluation, so the overestimation of action value is constantly low (< 0.128) regardless of the number of agents or actions.

$Q_{\text{MIX}}^* = V_{\text{MIX}}^* = \sum_a V_a^*$, and $Q_{\text{MIX}} = \sum_a Q_a$. Hence, the values that heatmaps visualize are averaged sample estimations of $\mathbb{E}[Z]$ without the γ factor in different settings (given number of agents, number of actions, and methods). In double QMIX, the online network and the target network Θ^- are two independent networks.

We sample $V_a(\tau^a)$ uniformly from a wide range $[-100, 100]$ to better simulate real-world scenarios where optimal action values are in a large range. For each setting, we generate 1000 random pairs of estimate action values and optimal action values. We set ϵ to be only 1, so the estimated action is in the close proximity of the optimal action value.

Despite accurate estimated action values for individual agents, vanilla QMIX shows a significant increase in the overestimation along an increase in the number of actions or in the number of agents or in both, whereas double QMIX manages to keep the overestimation constantly below 0.128.

The heatmap of vanilla QMIX also indicates that the number of agents n has a larger correlation with the overestimation than the number of actions m . This partially reflects the lower bound shown in Equation 25.

5.2 StarCraft II Micromanagement

We have shown that double QMIX method can reduce the overestimation of action target value. However, the proof in Section 4 and the visualization in 5.1 are based on the assumption of constant optimal agent action value for a given trajectory and the *mixing function* being linear. In a general multi-agent con-

trol problem and QMIX with monotonic value function factorisation (Section 2.4.2), these conditions no longer hold. Also, the increase in performance resulting from the reduction of overestimation is not shown.

In this section, we deploy the double QMIX method in StarCraft II Micromanagement. We show that double QMIX performs constantly better than mono QMIX (ie. monotonic value function factorisation) in 3 complex settings with 5, 8, 9 agents respectively. The results of VDN method in StarCraft II micromanagement tasks are constantly outperformed by mono QMIX (Rashid et al., 2018), so we do not plot the results of VDN.

5.2.1 Experimental Setup

In order to have a fair comparison with mono QMIX, we use the identical StarCraft II micromanagement setup and the same network structure as in (Rashid et al., 2018). We state the games' setup in next paragraph, but refer readers to (Rashid et al., 2018) to find the detailed network structure.

Each game consists of 2 identical teams in competition. The first team is controlled by RL algorithms, and the second team is controlled by StarCraft II AI heuristics with difficulty set to medium. We test over 3 complex settings: 2 Stalkers and 3 Zealots (2s3z: 5 agents to control), 3 Stalkers and 5 Zealots (3s5z: 8 agents to control), and 1 Colossus, 3 Stalkers and 5 Zealots (1c3s5z: 9 agents to control). The action space of agents consists of `move[direction]`, `stop`, `attack[enemy_id]`, and `noop`. NORTH, EAST, SOUTH, and WEST are the four directions allowed to move. Each agent has a sight range: only informa-

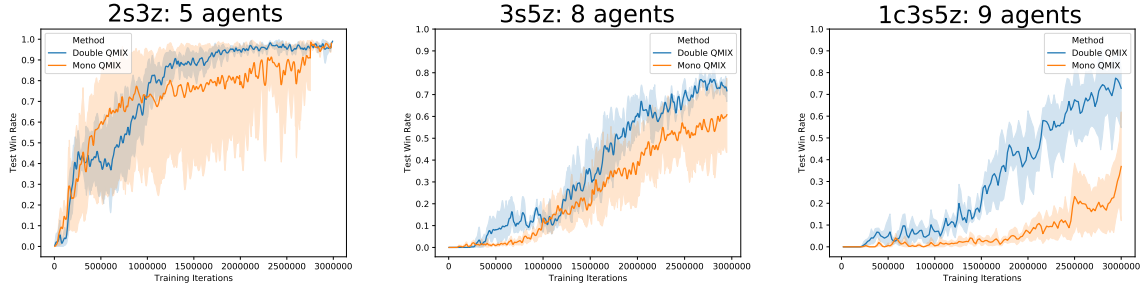


Figure 2: The blue curve is for double QMIX, and the orange curve is for mono QMIX. 3 plots are for the settings of 2 Stalkers 3 Zealots, 3 Stalkers and 5 Zealots, and 1 Colussus 3 Stalkers 5 Zealots (from left to right order). The bold lines are the mean test winning rates, and the shaded regions cover 90% of confidence levels.

tion of agents (allies or enemies) within the range can be observed, forming a partial observation of the battleground. The *attack-move* and *auto-fire* within sight range options are disabled. For agents within the sight range, their distance, relative x, relative y, and unit_type are provided as an observation vector z_t . The global state information, available to the *mixing network*, is a feature vector consisting of the union set of all observations to different agents, health, shield_points, and cooldown_time of all agents. Stalkers, Zealots and Colossi have 80, 100, 200 hit points respectively. They also have 80, 50, and 150 shield points respectively. All agents have the same sight range of 9 and the same shooting range of 6. At each time step, all agents receive a shared reward, equivalent to the total damage on all enemies. Additionally, 10 points given for killing an enemy, and 200 points for killing all. The maximal rewards achievable in an episode is normalized to 20. The maximal length of an episode is 200. If there are enemies alive, we treat it as a failure.

All but one hyper-parameters we used for double QMIX are the same as stated in (Rashid et al., 2018), which is tuned for mono QMIX. These hyper-parameters are comprised of ϵ in ϵ -greedy agent action selection linearly decaying from 1.00 to 0.05 over 50 thousand time steps and then keeping constant, $\gamma = 0.99$, 5000 episodes of replay buffer size, training mini-batch size of 32, and learning rate for RMSprop (Hinton et al.,) setting to 5×10^{-4} . We set the target network update period, the only hyper-parameter we optimize, to 1000 training episodes, as it is noticed that longer update period has a better decorrelation effect during computing the computed target action (Van Hasselt et al., 2016).

5.2.2 Results

The plots of test winning rates (using greedy action selection) against the training episodes in 3 settings

are shown in Figure 2. The test winning rates are obtained every 2000 training episodes by greedy sampling 20 independent episodes. We average the test winning rates across 4 runs for each settings.

In all 3 settings, even though all but one hyper-parameters are not tuned for double QMIX, it shows its advantages over mono QMIX, in terms of the means and the variances of test winning rates. Compare 2s3z and 3s5z settings. The number of agents that are needed to control increases from 5 to 8, though the types of agents in the group is not changed: Stalkers and Zealots. Double QMIX displays a larger performance gain when the number of agents increases. Compare 3s5z and 1c3s5z settings. Although there is only 1 more agent in team in 1c3s5z setting, the mean test winning rates sharply decrease when using mono QMIX. The addition of 1 more agent type (Colossus), whose hit point and shield point are different from those of other 2 types of agents, does not negatively affect the double QMIX method based on the similar test winning rate trajectories in 3s5z and 1c3s5z settings.

6 CONCLUSION AND FUTURE WORKS

In this work, we formally prove that overestimations exist in multi-agent value mixing Q -learning under certain conditions. We show that the idea presented in double DQN can be generalized to multi-agent settings. We present the adapted method, double QMIX, and illustrate its performance gains against other value mixing methods in StarCraft II micro-management tasks. In future, we hope to extend the expressiveness of the *mixing network* and keep the stability of action value estimation to model a richer class of joint action functions and to improve the performance to an even higher level.

REFERENCES

- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.
- Bellman, R. (1966). Dynamic programming. *Science*, 153(3731):34–37.
- Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075.
- Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2.
- Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Guestrin, C., Koller, D., and Parr, R. (2002). Multiagent planning with factored mdps. In *Advances in neural information processing systems*, pages 1523–1530.
- Ha, D., Dai, A. M., and Le, Q. V. (2017). Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.
- Hasselt, H. V. (2010). Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621.
- Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.
- Hausknecht, M. J. (2016). *Cooperation and communication in multiagent deep reinforcement learning*. PhD thesis.
- Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., and Graepel, T. (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865.
- Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.
- Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.
- Kraemer, L. and Banerjee, B. (2016). Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94.
- Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.
- Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016). Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.
- Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.
- Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390.
- Matarić, M. J. (1997). Reinforcement learning in the multi-robot domain. In *Robot colonies*, pages 73–83. Springer.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.
- Oliehoek, F. A. (2012). Decentralized pomdps. In *Reinforcement Learning*, pages 471–503. Springer.
- Oliehoek, F. A., Amato, C., et al. (2016). *A concise introduction to decentralized POMDPs*, volume 1. Springer.
- OpenAI (2018). Openai five. <https://blog.openai.com/openai-five/>.
- Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Dickstein, J. S. (2017). On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2847–2854. JMLR. org.
- Rashid, T., Samvelyan, M., Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4292–4301.
- Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.
- Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. (2019). The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188. International Foundation for Autonomous Agents and Multiagent Systems.
- Singh, S. P. and Bertsekas, D. P. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in neural information processing systems*, pages 974–980.
- Son, K., Kim, D., Kang, W. J., Hostallero, D. E., and Yi, Y. (2019). Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learn-

- ing. In *International Conference on Machine Learning*, pages 5887–5896.
- Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., et al. (2018). Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2085–2087. International Foundation for Autonomous Agents and Multi-agent Systems.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.
- Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.
- Tan, M. (1993). Multi-agent reinforcement learning: independent versus cooperative agents. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann Publishers Inc.
- Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219.
- Thrun, S. and Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*.
- Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.
- Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T. P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. (2017). Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782.
- Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.