A Multi-branch Ensemble Agent Network for Multi-agent Reinforcement Learning

Renlong Chen[1,2] and Ying Tan[1,2(✉)]

[1] Nanjing Kangbo Intelligent Health Academy, Nanjing 211100, China
[2] Key Laboratory of Machine Perception (Ministry of Education), Peking University, Beijing 100871, China
{reo,ytan}@pku.edu.cn

Abstract. Multi-agent Reinforcement Learning (MARL) has drawn wide attention in recent years as a bunch of real-world complex scenes can be abstracted as multi-agent systems (MAS). Partially observable cooperative multi-agent setting, in which agents have to learn to coordinate with allies by actions conditioning on their own partial observation and share a single global reward each time-step, is the most concerned MAS by existing MARL algorithms with centralized training and decentralized executing. One key challenge is how to make effective oriented exploration. In this work, we propose a new agent network called Multi-branch Ensemble Agent Network (MEAN) to encourage the oriented exploration. We evaluate our MEAN with existing Q-learning based MARL algorithms on StarCraft II micro-management challenges. Extensive evaluations show that algorithms equipped with MEAN achieve much better performance on both homogeneous and heterogeneous scenarios compared with the initial algorithms.

Keywords: Multi-agent reinforcement learning · Multi-branch ensemble · Multi-agent systems · Swarm intelligence

1 Introduction

Over the past decades, multi-agent reinforcement learning (MRAL) has been studied extensively and developed a lot, which plays an important role in addressing many real-world problems, such as autonomous car designing, game strategy, robot swarm coordination, complex control tasks, etc. In a multi-agent system (MAS), agents are required to obtain a policy which can coordinate with other agents to gain maximized accumulated global rewards.

While there are many challenges for optimizing a policy of MRAL. Firstly, the size of joint action space expands exponentially as the number of agents grows [4,10]. Secondly, the learning process is unstable and non-markovian because the agent not only interacts with environment but also interacts with other agents with changing policies [6,7]. Furthermore, the credit assignment is also one of the biggest challenges in fully cooperative scenarios. Those challenges make it extremely difficult to train agents individually in MAS. The paradigm of centralized training with decentralized execution (CTDE) [9] has drawn more attention recently for alleviating the above constraints.

Y. Tan et al. (Eds.): DMBD 2021, CCIS 1454, pp. 485–498, 2021.
https://doi.org/10.1007/978-981-16-7502-7_44

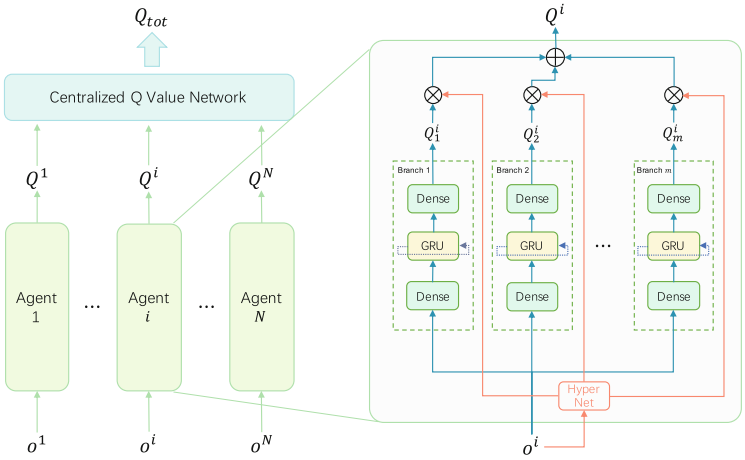


Fig. 1. The MEAN Framework. The left block is the overall architecture consisting of individual value function networks for N agents and a centralized value function network to assign credit. The right block is MEAN, including m branches. Each branch is a basic DRQN composed of a fully-connected layer and a GRU layer, followed by a fully-connected layer with dimension of number of actions.

In CTDE, each agent can only select its action conditioning on the local observation. To coordinate all agents, a centralized Q value function shared by all agents is usually needed during centralized training. The centralized Q value function builds a connection between the global reward and individual value function of each agent. However, it is often impractical to learn a centralized value function directly due to the curse of dimensionality, quite apart from connecting the individual value functions. Many algorithms tend to put forward some structural hypothesis. For example, VDN [14] assumes that the centralized value is the sum of individual values from all agents, QMIX [10] assumes that the centralized value function is a monotonic function of individual value functions. QTRAN [13] further relaxes those restrictions to a more general assumption that optimal joint action derived from centralized value is equivalent to optimal action chosen from agents' individual Q value functions.

Those algorithms under CTDE structure are usually based on Q learning. Despite Q learning has achieved great success in single agent environment tasks, where the environment has dense rewards that are easy to find by taking random sequences of actions. However, it tends to fail when the rewards are sparse and hard to find. Poor exploration is an inherent defect of Q learning [3] as it follows the bellman optimal equation to iterate value function. ε-greedy is one of the most extensively used techniques to encourage exploration in Q learning. ε-greedy generates random actions instead of action which makes Q value optimal by a decay probability. ε-greedy can only provide random exploration in practice. Unfortunately, random exploration usually does not work in multi-agent systems because coordinated actions are rarely searched by random exploration. Oriented exploration is needed in multi-agent tasks.

To tackle down this problem, we propose a novel agent architecture called Multi-branch Ensemble Agent Network (MEAN). MEAN improves oriented exploration by introducing multi-branch ensemble in individual Q value function which makes the final individual value as a sample from a distribution. The pseudo distribution contains the information learned from the past training episode transition data, meanwhile the structure makes sure that the ensembled Q value is an unbiased estimation of original individual Q value. The knowledge distillation furthermore helps each branch to learn from ensembled individual Q value.

We evaluate MEAN with VDN, QMIX and QTRAN which are most representative Q-learning based CTDE algorithms on a range of StarCraft II micro-management tasks. Experiments show that our MEAN achieves significant improvement comparing to original algorithms and helps to explore winning states in very early episodes. Ablation experiments show how knowledge distillation and exploration loss improve oriented exploration.

The remains of this paper are as follow, we first introduce some background knowledge for multi-agent reinforcement learning and multi-branch ensemble. Then we describe Multi-branch Ensemble Agent Network Architecture. Next, evaluation of the proposed method in StarCraft Multi-Agent Challenge are provided. Conclusion of this paper is given in the last section.

2 Background

2.1 Dec-POMDP

A multi-agent task could be formulated as a decentralized partially observable Markov decision process (Dec-POMDP) [2]. It's formally defined as a tuple

$$G = \langle N, s, \vec{A}, T, \vec{r}, \vec{O}, Z, \gamma \rangle \tag{1}$$

where N is the number of agents. s denotes the state of the environment to which agents have no access during interaction with the environment. $\vec{A} = (A_1, A_2, \ldots, A_N)$ denotes the set of joint action where A_i is the local actions set agent i can take at each time-step which controlled by its own policy $\pi_i : O_i \times A_i \to [0, 1]$. State transition function is $T(s'|s, \vec{a}) : S \times \vec{A} \times S \to [0, 1]$. The joint reward function $\vec{r} = (r_1, r_2, \ldots, r_N) : S \times \vec{A} \to \vec{N}$ consists of individual reward r_i. $\vec{O} = (O_1, O_2, \ldots, O_N)$ denotes the set of joint observation controlled by the observation function $Z : S \times \vec{A} \to \vec{O}$. The scenario discount factor is $\gamma \in [0, 1]$.

There are generally two types of multi-agent tasks according to whether there are competitive goals among allies or not. We focus on fully cooperative tasks. In those settings, agents will share an identical reward $r = r_1 = r_2 = \cdots = r_N$. The multi-agent system aims at learning a policy $\pi_i(a_i|o_i)$ which maximizes the expectation of discounted accumulated return $\mathbb{E}(G)$ where $G = \sum_{t=0}^{T} \gamma^t r^t$, T is the accumulated horizon.

Algorithm 1. Training Procedure for MEAN

Initialize individual Q network and target network with parameters θ and θ^- respectively, centralized Q network with parameters θ^c, replay buffer $\mathcal{D}$ with capacity $N_{\mathcal{D}}$, training batch size N_b and other hyper-parameters
for each training step **do**
  for each episode **do**
    for $t = 1$ to $max\ time - step$ **do**
      Obtain global state s_t
      for $i = 1$ to $Number\ of\ agents$ **do**
        Obtain observation o_i^t for each agent i
        Compute weights $\boldsymbol{g}$ according to partial observation o_i^t
        Compute individual Q value Q^i according to outputs of each branch and weights $\boldsymbol{g}$
      end for
      Execute joint action $\boldsymbol{a}_t$ in environment
      Obtain the global reward r_t
    end for
    Store episode transitions in $\mathcal{D}$, replacing the oldest episode if $|\mathcal{D}| \geq N_{\mathcal{D}}$
  end for
  Sample a batch of N_b episodes from $\mathcal{D}$
  Calculate TD-loss $\boldsymbol{\mathcal{L}}_e$ according to Eq. 6 and $\boldsymbol{\mathcal{L}}_b$ for each branch if needed
  Calculate knowledge distillation loss $\boldsymbol{\mathcal{L}}_{kd}$ according to Eq. 7
  Calculate exploration loss according to Eq. 11
  Update θ by minimizing $\boldsymbol{\mathcal{L}} = \boldsymbol{\mathcal{L}}_{TD} + \alpha\boldsymbol{\mathcal{L}}_{kd} + \beta\boldsymbol{\mathcal{L}}_e$
  Update target network parameters $\theta \rightarrow \theta^-$ every C training steps
end for

2.2 Reinforcement Learning

Reinforcement Learning [15] has been widely investigated to solve the single agent POMDP problems. Q-learning uses value iteration to update an action-value function $Q(s,a) = \mathbb{E}[G|S = s, A = a]$ by $a^* = \arg\max_a Q(s,a)$. However, when using Q-learning to solve some complex problems, it encounters the problem of high dimensional curse on both state and action spaces since traditional Q-learning uses table or parameterized function to represent the Q function. Deep Q Network (DQN) [8] tackles down this problem by using a deep neural network to represent Q function. There have been several techniques applied to stabilize the training process of DQN, such as target network and experience replay. The original DQN updates parameters θ by minimizing the following TD loss

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \tag{2}$$

where $Q(s', a'; \theta^-)$ is the target network whose parameters θ^- synchronizes with the main network parameters θ.

2.3 Multi-branch Ensemble Knowledge Distillation

There have been a lot of existing methods for knowledge distillation. A typical distillation process starts with a larger network with high-capacity parameters or architecture as a teacher model, followed by training a smaller student network targeting predicting the teacher network's outputs or some high-level feature representations [1,5,11]. [5] improved a small network by distilling knowledge from a larger teacher network. The rationale behind knowledge distillation is that a larger network's prediction provides extra supervision than conventional supervised learning with objective function using training data labels. However, conventional knowledge distillation methods require a two-stage training process or rely on a pre-trained powerful teacher model. [18] proposed a multi-branch ensemble strategy for one-stage online distillation.

3 Methods

3.1 Hypothesis Constraints for Centralized Value Function

The joint action space of all agents in a multi-agent system surges exponentially with respect to the increase of the number of agents, which makes it impractical to learn a single value function for all agents in some scenarios, implying that utilizing an individual value function conditioning on local observation for each agent is one of the feasible ways. Meanwhile, to coordinate allies' behaviors, we have to train a centralized value function for all agents. To learn a stable and trainable centralized value function, we have to incorporate a hypothesis constraint for centralized value function. One practical hypothesis constraint is as follow

$$\arg\max_{\boldsymbol{a}} Q_{tot}(\boldsymbol{o}, \boldsymbol{a}) = \begin{pmatrix} \arg\max_{a^1} Q^1(o^1, a^1) \\ \vdots \\ \arg\max_{a^N} Q^N(o^N, a^N) \end{pmatrix} \tag{3}$$

Equation 3 establishes a structural constraint linkage between individual value functions and the centralized value function, therefore, this hypothesis constraint could be regarded as a form of credit assignment. However, it is impractical to use Eq. 3 directly in training phase because it lacks a formulaic factorization to compute.

The following non-negative linear assumption is a sufficient condition for Eq. 3

$$\begin{aligned} Q_{tot}(\boldsymbol{o}, \boldsymbol{a}) &= \sum_{i=1}^{N} \alpha^i Q^i(o^i, a^i) \\ \alpha^i &\geq 0 \end{aligned} \tag{4}$$

where Q_{tot} denotes the centralized Q value function and Q^i is the individual Q value function for agent i. VDN simply sets all combination coefficients $\alpha^i, i \in \{1, \cdots, N\}$ to 1, which is a quite strong constraint. QMIX relaxes the constraint to a general additive value factorization by enforcing $\partial Q_{tot}/\partial Q^i \geq 0, i \in \{1, \cdots, N\}$. Therefore, VDN can be regarded as a special case of the QMIX algorithm. QTRAN relaxes the constraint even further and exploration in a larger hypothesis space structured by a sufficient and

necessary condition of Eq. 3. To this end, QTRAN has to optimize the joint value function in the full joint action space, so that QTRAN suffers from computational challenge and the scalability. Applicable range of QTRAN are therefore limited.

Table 1. The features of all scenarios in experiments

Map name	Allies	Enemies
3m	3 Marines	3 Marines
8m	8 Marines	8 Marines
2s3z	2 Stalkers & 3 Zealots	2 Stalkers & 3 Zealots
3s5z	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots
1c3s5z	1 Colossi, 5 Stalkers & 5 Zealots	1 Colossi, 5 Stalkers & 5 Zealots
3s_vs_3z	3 Stalkers	3 Zealots

3.2 Multi-branch Ensemble Agent Network

Here we describe our Multi-branch Ensemble Agent Network and Fig. 1 depicts the overview of MEAN architecture. For description convenience, we take vanilla VDN [14] as a backend mixer example. It is straightforward to apply MEAN to algorithms with other Q value mixers, such as QMIX [10] and QTRAN [13]. To this end, we assume that VDN has a mixer network as well, which simply sums all individual Q value to get Q_{tot}.

During the execution phase, at time-step t, agent i takes as input the agent's observation o_t^i, each branch with parameters θ_j computes individual value function $Q_j^i(o_t^i, a^i; \theta_j)$ based on those inputs. The Hyper network produces the weight coefficient $\boldsymbol{g}(o_t^i)$. We therefore ensemble outputs of m branches by the hyper network to the individual value function we use in execution as

$$Q^i(o_t^i, a^i; \theta) = \sum_{j=0}^{m} g_j \cdot Q_j^i(o_t^i, a^i; \theta_j) \tag{5}$$

During the training phase, the centralized Q value is produced by $Q_{tot}(\boldsymbol{o}, \boldsymbol{a}, s; \theta) = \sum_{i=1}^{N} Q^i(o^i, a^i)$. The vanilla algorithm trains the network by TD loss defined as

$$\boldsymbol{\mathcal{L}}_e(\theta) = \sum_{i=1}^{b} [y_{tot}^i - Q_{tot}(\boldsymbol{o}, \boldsymbol{a}, s; \theta))]^2 \tag{6}$$

where b is the batch size of sampled transitions per training iteration, $y_{tot} = r + \gamma \max_{a'} Q_{tot}(\boldsymbol{o'}, \boldsymbol{a'}, s'; \theta^-)$ and θ^- denotes the parameters of the target network.

In algorithms which satisfy the hypothesis of Eq. 4, we can similarly get the Q_{tot}^i composed of the Q_j^i by each branch. We can enhance the learning process of each branch by minimizing the TD loss w.r.t Q_{tot}^i defined as $\boldsymbol{\mathcal{L}}_b(\theta) = \sum_{i=1}^{b} \sum_{j=1}^{m} [y_{tot}^i - Q_{tot}^j(\boldsymbol{o}, \boldsymbol{a}, s; \theta))^2]$, where m denotes the number of branches, $Q_{tot}^j = \sum_{i=1}^{N} \alpha^i Q_j^i(o^i, a^i)$. In those setups, the final TD loss is $\boldsymbol{\mathcal{L}}_{TD} = \boldsymbol{\mathcal{L}}_e + \boldsymbol{\mathcal{L}}_b$, otherwise, the final TD loss $\boldsymbol{\mathcal{L}}_{TD}$ is identical with $\boldsymbol{\mathcal{L}}_e$.

Knowledge Distillation. Besides the TD loss w.r.t Q_j^i, we can enhance each branch by distilling knowledge from ensembled teacher back into branches. To quantify the alignment between branches and the ensembled teacher, we use

$$\mathcal{L}_{kd} = \sum_{i=1}^{N} \sum_{j=1}^{m} \|Q_j^i - Q^i\|_2^2 \tag{7}$$

where $\|\cdot\|_2^2$ is the squared L_2-norm. For algorithms with Actor-Critic architecture, we use Kullback Leibler Divergence

$$\mathcal{L}_{kd} = \sum_{i=1}^{N} \sum_{j=1}^{m} p_e(\boldsymbol{a}|\boldsymbol{o};\theta_e) \log \frac{p_e(\boldsymbol{a}|\boldsymbol{o};\theta_e)}{p_j(\boldsymbol{a}|\boldsymbol{o};\theta_j)} \tag{8}$$

where $|A|$ represents the number of action dimension, p_e and p_j represent the softmax predictions of actions of the ensembled teacher and branch respectively, θ_e and θ_j are for the parameters of the ensembled teacher and branch respectively.

Table 2. Mean performance of the test win percentage

Map	IQL	COMA	VDN		QMIX		QTRAN	
			Original	MEAN	Original	MEAN	Original	MEAN
3m	**100**	91	97	**100**	95	96	86	66
8m	91	94	75	**99**	91	98	93	86
2s3z	39	66	80	91	86	**99**	32	95
3s5z	0	0	30	91	78	**95**	0	20
1c3s5z	7	30	90	99	**100**	**100**	42	90
3s_vs_3z	0	0	56	98	14	**100**	0	21

Oriented Exploration. A large number of works apply ε-greedy action selector to implement exploration in Q learning. The basic idea of ε-greedy is choosing random actions for agents with a decay probability. However, ε-greedy exploration is a random strategy which provides none oriented exploration. Despite the action selection in Q learning does not require a softmax probability, softmax function is monotonically increasing, which indicates $\arg\max_{\boldsymbol{a}} Q(\boldsymbol{o},\boldsymbol{a}) = \arg\max_{\boldsymbol{a}} \sigma(Q(\boldsymbol{o},\boldsymbol{a}))$, we can consider an action selection probability w.r.t the Q value instead in the following discussion.

Different from the policy gradient based methods, we cannot sample actions by the probability. Nevertheless, in multi-branch ensemble architecture, the final probability is composed of probabilities from each branch. Therefore, in multi-branch ensemble setup, the probability is no longer deterministic. Meanwhile, since each branch is trained under the same objective function, the probability from each branch can be considered as a sample from a distribution. Consider a m-branch agent with binary actions,

assume that $P(a = 1) = \theta_i$ of branch i is sampled from a normal distribution with mean value μ and variance σ^2. The overall mean value and variance of final probability $\theta = \sum_{i=1}^{m} g_i \theta_i$ are as follow

$$\begin{aligned} \mathbb{E}(\theta) &= \mathbb{E}(\sum_{i=1}^{m} g_i \theta_i) = \mu \sum_{i=1}^{m} g_i = \mu \\ \mathbb{D}(\theta) &= \mathbb{D}(\sum_{i=1}^{m} g_i \theta_i) = \sigma^2 \cdot \sum_{i=1}^{m} \sum_{j=1}^{m} g_i g_j = \sigma^2 \end{aligned} \tag{9}$$

In order to encourage exploration, a larger variance σ^2 is needed. To satisfy this requirement, we can penalize the normalized cosine similarity of outputs from each branch by squared Frobenius norm. Let $\mathbf{H}^i$ be the normalized matrix of branch outputs of agent i, the exploration loss encourages orthogonality between branches

$$\mathcal{L}_e = \sum_{i=1}^{N} \|{\mathbf{H}^i}^\top \mathbf{H}^i\|_F^2 \tag{10}$$

where $\|\cdot\|_F^2$ denotes squared Frobenius norm. However, using Eq. 10 to encourage exploration will hurt the Q value training process because squared Frobenius norm would cause the mean value of weighted Q value converged to $\mathbf{0}$. We propose a revised version of exploration loss defined as:

$$\mathcal{L}_e = \sum_{i=1}^{N} \|(\mathbf{H}^i - \overline{\mathbf{H}}^i)^\top (\mathbf{H}^i - \overline{\mathbf{H}}^i)\|_F^2 \tag{11}$$

We experiment with those two kinds of exploration losses, which we will discuss in detail.

Here we describe the overall goal of training as minimizing the following loss:

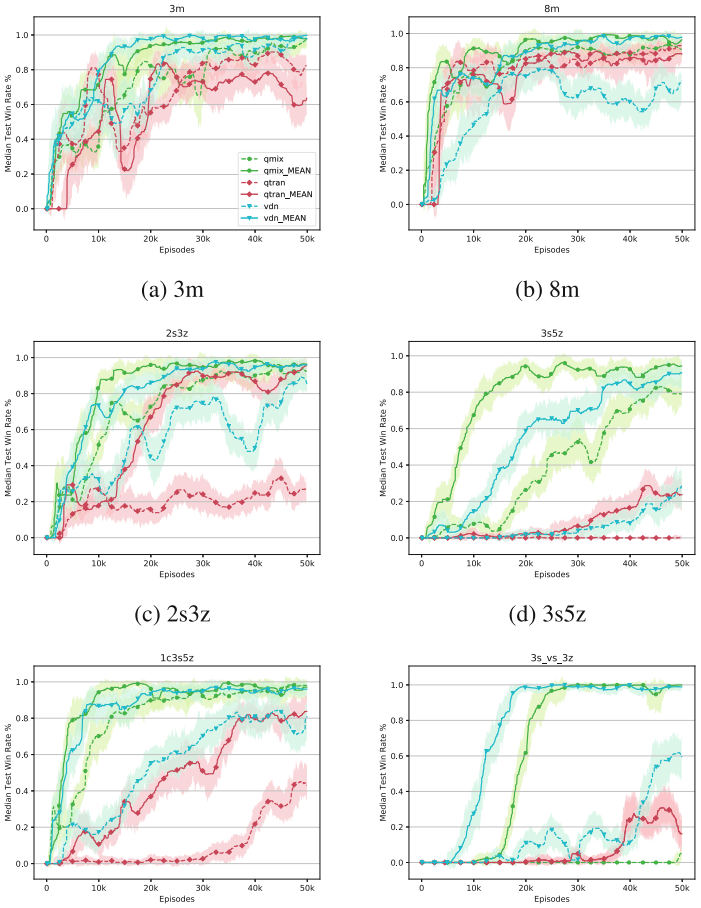
$$\mathcal{L} = \mathcal{L}_{TD} + \alpha \mathcal{L}_{kd} + \beta \mathcal{L}_e \tag{12}$$

where α, β are hyper-parameters to control the interaction of the loss terms. The general training procedure for MEAN is provided in Algorithm 1.

4 Experiments and Results

4.1 Experimental Setup

In this section, we describe our experimental setup. We test our method in the StarCraft Multi-Agent Challenge (SMAC) environment [12]. In this environment, each agent controls an individual unit. There is a built-in multi-level AI strategy controlling the enemy units by handcrafted heuristics. In our experiments, we set the difficulty level of built-in AI to level 7, which means "very difficult". The final goal is to defeat the enemy by eliminating all opponent units. Efficacious micro-managements of units are supposed



(a) 3m (b) 8m

(c) 2s3z (d) 3s5z

(e) 1c3s5z (f) 3s_vs_3z

Fig. 2. Median test win rate of VDN, QMIX, QTRAN and revised algorithms with our MEAN architecture in six different scenarios. Revised algorithms and original algorithms are shown in solid lines and dashed lines of the same color respectively. 25–75% percentiles are plotted shaded. All plots share the same legend in (a).

to cause damage on enemies to the full while minimize damage received. It's a challenging task to learn an efficacious cooperative strategy in such a POMDP environment, therefore SMAC became a widely used benchmark for evaluating MARL methods.

4.2 Training Configurations

The architecture of agent's Q network used in both compared algorithms and single branch network in our method is a DRQN, which contains an embedding layer and a GRU layer followed by a fully connected layer with $|A|$ outputs. The hidden state's

dimension of the embedding layer and GRU layer is 64. The individual Q network takes as input the agent's local observation and the last chosen action. In our experiments, we share the parameters across all agents' individual Q networks. To yield diverse strategies, we concatenate a one-hot vector of agent's number to the original input. The architectures of our method's mixer networks are identical with their original architectures. All hidden states' dimensions are 64 as well. We set γ at 0.99. Replay buffer size is set to 5000 episodes. In each training phase, 32 episodes are sampled from replay buffer. All Target networks used are updated after every 200 training phases. We use ε-greedy action selector in which ε anneals from 1. to 0.05 at the first 50000 steps to encourage exploration in the earliest training phase and remains 0.05 to ensure a minimal randomness during the whole training. We test all methods at every 50 training interactive episodes on 20 evaluation episodes with ε set to 0. Considering the balance of performance and training speed, we use 4 branches for each agent with MEAN and take one branch which generates the biggest mean coefficient in test rounds.

4.3 Results

Our method can easily be applied to the existing state-of-the-art algorithms such as VDN [14], QMIX [10] and QTRAN [13] by changing the agent network. To evaluate the improvement of our method comparing with the original version, we test them on several StarCraft II Micro-management Maps, including 3m, 8m, 2s3z, 3s5z, 1c3s5z, 3s_vs_3z, which contain both homogeneous and heterogeneous scenarios.

Table 1 shows the features of all maps we considered. All maps have different number or types of agents. The map name indicates the scenario setup. m, s, z and c denote different types of agents, which are Marine, Stalker, Zealot and Colossus respectively. In Map 3m, both sides have 3 Marines, there are 8 Marines on both sides in Map 8m similarly. Map 2s3z, 3s5z and 1c3s5z are heterogeneous & symmetric, there are 2 Stalkers & 3 Zealots, 3 Stalkers & 5 Zealots and 1 Colossus & 2 Stalkers & 3 Zealots on both sides of those scenarios respectively. Map 3s_vs_3z is asymmetric where ally has 3 Stalker while enemy has 3 Zealots.

The main evaluation metric is the win percentage of evaluation episodes over the course of training. The resulting plots include the median performance as well as the 25–75% percentiles to avoid outliers' effect. The learning curves of comparing algorithms on all scenarios are shown in Fig. 2. Quantitative comparisons of all algorithms after training for 50,000 episodes are provided in Table 2. The performance of IQL [16] and COMA [4] are also provided in Table 2 for reference.

Overall, we could see that our MEAN architecture improves all comparing algorithms in maps 2s3z, 3s5z, 1c3s5z, 3s_vs_3z. In homogeneous and symmetric scenarios such as 3m and 8m, MEAN improves the performance of VDN and QMIX. Even though both VDN and QMIX achieve over 95% mean win rate in Map 3m, MEAN improves the performance of these two algorithms. In Map 8m, MEAN helps VDN to win almost all test runs. However, MEAN fails to improve the performance of QTRAN on these two maps. The rationale behind is that symmetric scenarios with small group scale

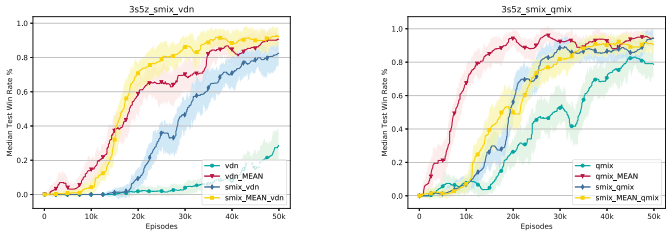
brings more uncertainties, which means random exploration is enough to search winning states. Meanwhile QTRAN has been suffered from the unstable training process. The oriented exploration strategy of MEAN worsens the effects of instability which preponderates over the benefit of oriented exploration. When it comes to harder scenarios where random exploration is no longer capable to search winning state like heterogeneous & symmetric scenarios, such as 2s3z, 3s5z, 1c3s5z, MEAN improves QTRAN significantly. In those scenarios, VDN and QMIX with MEAN achieve over 90% mean win rate. In Map 3s_vs_3z, only VDN and QMIX with MEAN have learned an effective policy to ensure success. As depicted in Fig. 2, algorithms with MEAN start to win battles earlier than their original versions, especially in Map 3s5z, 1c3s5z and 3s_vs_3z where random exploration cannot provide effective exploration.

We also compare our MEAN with SMIX (λ) [17]. SMIX (λ) alleviates the sparse experiences and unstable nature of MAS by enhancing the quality of centralized value function. SMIX (λ) is beneficial to other CTDE methods as well by replacing their centralized value function estimator. We apply our MEAN on SMIX (λ) and compare all three modifications with VDN and QMIX in Map 3s5z and 3s_vs_3z. Experiments show that algorithms with MEAN achieve the best performance. To be specific, vdn_MEAN and smix_MEAN_vdn outperform vdn and smix_vdn. MEAN helps algorithms to explore winning states in the very early episodes. All four compared QMIX algorithms have learned effective policy in Map 3s5z and only the original QMIX failed in Map 3s_vs_3z. There are significant gaps between the learning curves of qmix_MEAN and smix_MEAN_qmix in both scenarios which indicate that MEAN architecture improves complex centralized value function less than simple centralized value function (Fig. 3).

4.4 Ablation

We also perform ablation experiments to investigate the influences of knowledge distillation loss and exploration loss. We take VDN as an example back-end for comparison in Map 3s5z as this map is both difficult and heterogeneous.

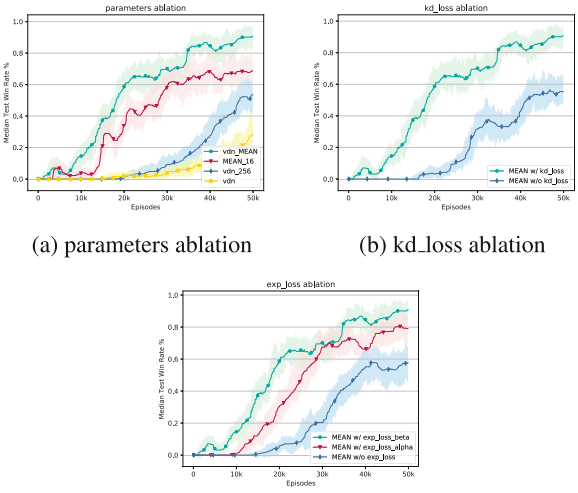
Amount of Parameters. Despite only one branch is used in tests, we also test the original VDN with hidden size of 256 to show that simply scaling up the amount of parameters cannot perform adequate exploration in difficult tasks. Meanwhile, we scale down the amount of parameters trained in each branch to 16 hidden unit per layer, and use all 4 branches in tests marked as vdn_MEAN_16. Experiment results in Fig. 4a show that both VDNs with MEAN outperform original VDNs in Map 3s5z which indicates that the improvement brought by MEAN does not benefit from the large amount of parameters during training. We will furthermore discuss the influences of knowledge distillation loss and exploration loss.



(a) 3s5z (b) 3s5z

(c) 3s_vs_3z (d) 3s_vs_3z

Fig. 3. Median test win rate of VDN, QMIX and revised algorithms with SMIX and our MEAN architecture in 3s5z and 3s_vs_3z.



(a) parameters ablation (b) kd_loss ablation

(c) exp_loss ablation

Fig. 4. Median test win rate of ablation experiments in 3s5z.

Knowledge Distillation Loss. To investigate the influence of knowledge distillation loss (kd_loss), we compare MEAN with and without kd_loss. As shown in Fig. 4b, the learning curve of MEAN without kd loss is more flat which means the learning of

individual value function is slower than that of with kd_loss. The experiment results conform to the anticipated benefit of knowledge distillation.

Exploration Loss. To investigate the influence of exploration loss (exp_loss), we first compare exp loss with Eq. 10 and Eq. 11 abbreviated as exp_loss_α and exp_loss_β respectively. Figure 4c shows that MEAN with exp_loss_β achieves more stable training process and higher performance. The result has verified the discussion above. Next, we evaluate MEAN with and without exploration loss. As depicted in Fig. 4c, the oriented exploration that exp_loss brought works well which reveals the necessity of exploration loss and oriented exploration.

5 Conclusion

In this paper, Multi-branch Ensemble Agent Network is proposed to solve the oriented exploration problem in Dec-POMDP multi-agent systems. A new norm function is introduced to encourage oriented exploration besides random exploration provided by ε-greedy. It is shown that MEAN architecture has explored winning state in the very early episodes and significantly improves performance of varied comparing algorithms with DRQN agents and the revised algorithms achieve state-of-the-art performance on various scenarios of StarCraft II micro-management tasks.

Acknowledgement. This work is supported by the Swarm Intelligence Project of Nanjing Kangbo Intelligent Health Academy, and partially supported by Science and Technology Innovation 2030 - "New Generation Artificial Intelligence" Major Project (Grant Nos.: 2018AAA0102301 and 2018AAA0100302) and by the National Natural Science Foundation of China (Grant No. 62076010).

References

1. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Advances in Neural Information Processing Systems, pp. 2654–2662 (2014)
2. Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of Markov decision processes. Math. Oper. Res. **27**(4), 819–840 (2002)
3. Burda, Y., Edwards, H., Storkey, A., Klimov, O.: Exploration by random network distillation. arXiv preprint arXiv:1810.12894 (2018)
4. Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., Whiteson, S.: Counterfactual multi-agent policy gradients. arXiv preprint arXiv:1705.08926 (2017)
5. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
6. Laurent, G.J., Matignon, L., Fort-Piat, L., et al.: The world of independent learners is not Markovian. Int. J. Knowl. Based Intell. Eng. Syst. **15**(1), 55–64 (2011)
7. Lowe, R., Wu, Y.I., Tamar, A., Harb, J., Abbeel, O.P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: Advances in Neural Information Processing Systems, pp. 6379–6390 (2017)
8. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)

9. Oliehoek, F.A., Spaan, M.T., Vlassis, N.: Optimal and approximate Q-value functions for decentralized POMDPs. J. Artif. Intell. Res. **32**, 289–353 (2008)
10. Rashid, T., Samvelyan, M., De Witt, C.S., Farquhar, G., Foerster, J., Whiteson, S.: QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. arXiv preprint arXiv:1803.11485 (2018)
11. Romero, A., et al.: FitNets: hints for thin deep nets. arXiv preprint arXiv:1412.6550 (2014)
12. Samvelyan, M., et al.: The StarCraft multi-agent challenge. CoRR abs/1902.04043 (2019)
13. Son, K., Kim, D., Kang, W.J., Hostallero, D.E., Yi, Y.: QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. arXiv preprint arXiv:1905.05408 (2019)
14. Sunehag, P., et al.: Value-decomposition networks for cooperative multi-agent learning based on team reward. In: AAMAS, pp. 2085–2087 (2018)
15. Sutton, R.S., Barto, A.G., et al.: Introduction to Reinforcement Learning, vol. 135. MIT Press Cambridge (1998)
16. Tan, M.: Multi-agent reinforcement learning: independent vs. cooperative agents. In: Proceedings of the Tenth International Conference on Machine Learning, pp. 330–337 (1993)
17. Wen, C., Yao, X., Wang, Y., Tan, X.: SMIX (λ): enhancing centralized value functions for cooperative multi-agent reinforcement learning. In: AAAI, pp. 7301–7308 (2020)
18. Zhu, X., Gong, S., et al.: Knowledge distillation by on-the-fly native ensemble. In: Advances in Neural Information Processing Systems, pp. 7517–7527 (2018)