Journal of Systems Engineering and Electronics
Vol. 33, No. 2, April 2022, pp.447 – 460

Knowledge transfer in multi-agent reinforcement learning with incremental number of agents

LIU Wenzhang[1], DONG Lu[2], LIU Jian[1], and SUN Changyin[1,*]

1. School of Automation, Southeast University, Nanjing 210096, China; 2. School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China

Abstract: In this paper, the reinforcement learning method for cooperative multi-agent systems (MAS) with incremental number of agents is studied. The existing multi-agent reinforcement learning approaches deal with the MAS with a specific number of agents, and can learn well-performed policies. However, if there is an increasing number of agents, the previously learned in may not perform well in the current scenario. The new agents need to learn from scratch to find optimal policies with others, which may slow down the learning speed of the whole team. To solve that problem, in this paper, we propose a new algorithm to take full advantage of the historical knowledge which was learned before, and transfer it from the previous agents to the new agents. Since the previous agents have been trained well in the source environment, they are treated as teacher agents in the target environment. Correspondingly, the new agents are called student agents. To enable the student agents to learn from the teacher agents, we first modify the input nodes of the networks for teacher agents to adapt to the current environment. Then, the teacher agents take the observations of the student agents as input, and output the advised actions and values as supervising information. Finally, the student agents combine the reward from the environment and the supervising information from the teacher agents, and learn the optimal policies with modified loss functions. By taking full advantage of the knowledge of teacher agents, the search space for the student agents will be reduced significantly, which can accelerate the learning speed of the holistic system. The proposed algorithm is verified in some multi-agent simulation environments, and its efficiency has been demonstrated by the experiment results.

Keywords: knowledge transfer, multi-agent reinforcement learning (MARL), new agents.

DOI: 10.23919/JSEE.2022.000045

Manuscript received August 13, 2021.
*Corresponding author.
This work was supported by the National Key R&D Program of China (2018AAA0101400), the National Natural Science Foundation of China (62173251; 61921004; U1713209), the Natural Science Foundation of Jiangsu Province of China (BK20202006), and the Guangdong Provincial Key Laboratory of Intelligent Decision and Cooperative Control.

1. Introduction

The research on reinforcement learning (RL) has made great progress in recent years, and many RL algorithms have been applied to solve various challenging problems [1−6]. The success of RL in single-agent systems has also been extended to multi-agent systems (MAS), where multiple agents behave together in a shared environment [7]. A direct idea of multi-agent reinforcement learning (MARL) is to learn independently by using a single-agent RL for each agent [8]. Independent learning may result in non-stationarity during the learning process, because the performance of each agent might be influenced by the policies of others [9]. To stabilize the learning process and improve the performance, the framework of centralized training and decentralized execution (CTDE) is widely used in MARL [10−13]. CTDE based algorithms collect the observations and actions of all agents during the training stage, and execute the policies independently. Most of the CTDE based algorithms assume the number of agents (denoted by N) in a given environment is fixed, and they can learn joint optimal policies for the MAS. However, if there comes M new agents, the environment for each agent will be changed, which means the joint optimal policies they learned before will lose the efficiency. In this scenario, it is computationally expensive to learn from scratch for new agents without considering the knowledge that was learned in previous. Define the previous environment with N agents as the source environment, and the current environment with $(N+M)$ agents as the target environment. To transfer the knowledge from source to the target and speed up the learning process, we propose to build a new MARL algorithm, in which the student agents (M new agents) can learn not only from the environment but also from the teacher agents (N old agents).

Recently, transfer learning (TL) algorithms have shown the efficiency in accelerating the learning from the source domain to the target domain [14–16]. Without the need for big data and long training time for target tasks, TL algorithms make full use of the knowledge learned in the source domain. For RL, the idea of TL can reduce the number of samples that are required to calculate an optimal policy by reusing the knowledge that was learned previously. By combining TL and RL, transfer RL (TRL) was proposed to deal with the knowledge transfer from one Markov decision process (MDP) to another MDP (in this paper, we call them the source MDP and the target MDP, respectively) [17–19]. If the target MDP is different from the source MDP, the optimal policy that was learned before will lose the efficiency, because the data distribution was changed. By using the experience knowledge of the previous environment, TRL aims to find an optimal policy in the target MDP without the need for retraining or large samples [20]. TRL algorithms are also applied in MAS to speed up the learning process and improve the performance [21–23]. Especially for cooperative MAS, in which agents aim to achieve common goals cooperatively, the knowledge transfer across agents [23–25] or tasks [26–28] are widely used to reduce the dependency on the samples and simplify the problem. However, most of the existing algorithms for multi-agent TRL also assume that the number of agents N is fixed, and the dimension of the input variables for each agent's policy is also determined by N.

In this paper, we aim at the MAS with incremental number of agents in a shared environment. Hence, the traditional framework of MARL cannot adjust the network structure dynamically and transfer the knowledge that was learned before. The scenarios of MAS with incremental number of agents appear commonly in many practical applications. For example, the unmanned warehouse system with new robots being added to improve the efficiency [29], the battle game with new players being joined to enlarge the problem scale [30], and the autonomous driving system in a highway environment with incremental number of new vehicles [31], etc. In these scenarios, how to let the new agents learn not only from the environment but also from the behaviors of the previously trained agents, is important to avoid learning from scratch and accelerating the training process. Denote the MAS with N agents as the source MDP and the MAS with $N+M$ agents as the target MDP. The main goal of the learning is to transfer the useful knowledge from source MDP to target MDP without the need for retraining, such that the agents in target MDP can find optimal policies with fewer training steps. In this paper, we mainly consider the cooperative MAS, where agents behave cooperatively to achieve common tasks. However, there are still some challenges to solve this problem. First of all, the learning framework for target MDP needs to be restructured without destroying the relationship between the previous agents. Secondly, the new agents need to select useful knowledge from teacher agents to avoid negative transfer in the target environment.

To solve the above challenges, we propose a new MARL algorithm to implement the knowledge transfer from source MDP to target MDP. The main ideas of this paper are extracting the experience knowledge of the teacher agents and transferring it to the student agents via online policy distillation. The contributions of this paper are listed as follows:

(i) We study the MARL with incremental number of agents, and design a new experience replay buffer for knowledge transfer (ERBKT) for new agents, such that they can learn not only from the environment but also from the teacher agents.

(ii) A new algorithm called multi-agent deep deterministic policy gradient with incremental number of agents (MADDPG-INA) is proposed to accelerate the learning process from the source MDP with N agents to the target MDP with $(N+M)$ agents.

The rest of the paper is organized as follows: The related work about this paper is introduced in Section 2. Section 3 is about the background and problem formulation. Details of the proposed algorithm are presented in Section 4. In Section 5, the simulation results are shown to verify the performance of the proposed algorithm. Finally, the paper is concluded in Section 6.

2. Related work

In recent years, the CTDE has been found useful in MARL to stabilize the learning process. For example, Lowe et al. [10] proposed multi-agent deep deterministic policy gradient (MADDPG) algorithm which collects the observations and actions of all agents during the training stage to approximate the agent-wise Q-functions, and executes the actions according to the agents' local observations with distributed policies. MADDPG applies the actor-critic architecture and DDPG algorithm [2] to calculate the policy gradient of each agent. For fully cooperative MAS, Foerster et al. [11] proposed the counterfactual multi-agent (COMA) policy gradient algorithm to address the credit assignment problem by calculating a counterfactual baseline as the advantage functions. Different from above, Sunehag et al. [12] proposed value decomposition network (VDN) to decompose the holistic Q-value function into a sum of individual agent-wise Q functions, which take the local observations and actions as input. To improve the performance of VDN, Rashid et al. [13] proposed a Q-mixing (QMIX) network to approximate the total Q-value function through a mixing network, which takes the individual agent-wise Q functions and the global state as input. Because the COMA, VDN, and QMIX have to calculate the Q-values of all actions during execution, they are unprocurable in MARL with continuous action spaces.

The problem of MARL will become more complex if the number of agents increases. Hence, the knowledge transfer across agents or tasks is considered to reduce the samples and speed up the training. In [28], Omidshafiei et al. introduced the single-task learning algorithm for concurrent interactions among decentralized agents. Then, they transferred the individual policies into one policy to implement multi-task learning via policy distillation [26,27,32]. To combine the knowledge across agents, Wadhwania et al. [24] treated the multi-agent single-task problems as single-agent multi-task problems through policy distillation and value matching methods. In [33], Chen also applied the policy distillation method and proposed a new framework for MARL to implement CTDE. Many of the TRL algorithms mentioned above are offline to deal with the tasks in the source domain and target domain. To enable online transfers, Taylor et al. [34] proposed a parallel transfer learning technique that can run the source and target tasks concurrently.

However, the algorithms above are proposed for the predefined system with a fixed number of agents. In [35], Agarwal et al. created shared agent-entity communication graph, which is invariant to the number of agents, to learn transferable cooperative behaviors in the multi-agent teams. The authors assumed that the environment can be described as a set of interactable entities, however, that will limit the applications in many practical problems. In [30], Wang et al. proposed dynamic multi-agent curriculum learning (DyMA-CL) algorithm which starts learning on an MAS with small number of agents, and the number of agents will increase progressively by curriculum learning [30]. A dynamic agent-number network (DyAN) architecture was designed in DyMA-CL to adapt to the increasing inputs of the networks. However, the DyAN in [30] learns agent-wise local Q functions directly without considering the observations and actions of other agents, which may result in local minima or non-stationarity during the training stage.

Different from the above algorithms, our algorithm implements the knowledge transfer for new agents and the learning of previous agents in parallel, and there is no assumption about the structure of the network parameters. More details about the proposed algorithms are introduced in the next sections.

3. Background and problem formulation

In this section, we first introduce the background of RL for single-agent systems and multi-agent systems. Then we present the problem formulation for TRL.

3.1 RL for single-agent systems

RL is built on MDP which can be described as a tuple $M \equiv \langle S, A, P, R, \gamma \rangle$ [36,37]. The first two elements are about the agent, which denotes the observed state space and action space, respectively. P and R are about the environment, which denote the state transition probability and reward mechanism, respectively. $\gamma \in (0,1)$ is the discount factor. Agent observes an state variable $s_t \in S$ at time step t to measure the environment and takes an action $a_t \in A$ under a policy π, and then the environment will return a reward signal $r_t = R(s_t, a_t)$ to the agent. The policy of the agent can be represented as a probability distribution $\pi : S \times A \to [0,1]$, or a deterministic function $\pi : S \to A$. The goal of the RL agent is to find an optimal policy π^* to get maximal accumulated return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. A state s under policy π is evaluated by the state value function $V^\pi(s) = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right]$, and a state-action pair under π is evaluated by the Q-value $Q^\pi(s,a) = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\right]$. The Q-value is helpful for agent to choose actions, and can also be represented as $Q^\pi(s,a) = r_t + \gamma \sum_{s'} P\left(s_{t+1} = s' | s_t = s, a_t = a\right) V^\pi(s')$. The optimal Q-function Q^* satisfies the Bellman equation as follows:

$$Q^*(s,a) = r_t + \gamma \max_{a' \in A} Q^*\left(s', a'\right) \tag{1}$$

where s' is the state at next time step.

In single agent RL, an optimal policy can be represented by $a_t = \operatorname{argmax}_{a \in A} Q^*(s_t, a)$. There are many algorithms to get the optimal policy that satisfies (1), such as Q-learning, SARSA, and policy gradient, etc.

3.2 RL for multi-agent systems

MARL is built on a decentralized partially observable Markov decision process (Dec-POMDP). A Dec-POMDP with N agents can also be described as a tuple $M^N \equiv \langle \{O^i, A^i, P^i, R^i\}_{i=1}^N, \gamma \rangle$. O^i and A^i are observation space and action space for agent i, respectively. Agent i takes actions by following an independent policy $\pi^i : O^i \times A^i \to [0,1]$, or $\pi^i : O^i \to A^i$. P^i is the state transition probability for agent i. R^i represents the reward function for agent i and $\gamma \in (0,1)$ is the discount factor. In this paper, we mainly study the model-free MARL with continuous actions, so we use the deterministic policy that outputs the actions directly ($a_t^i = \pi^i(o_t^i)$, where o_t^i and a_t^i are local observed state and local action of agent i at time step t, respectively), and the P^i is unknown during the learning process. Given that the concurrency of execution for all agents, the greedy policy in single-agent Q-learning cannot be applied in distributed policies. Instead, we apply the MADDPG [10] algorithm with the framework of multi-agent actor-critic to implement the MARL in this paper.

For a cooperative MAS, agents aim to find optimal distributed policies $\pi = \pi_{i=1}^{N}\pi^i$ coordinately to maximize the holistic objective function $J(\pi) = \sum_{i=1}^{N} J^i\left(\pi^i\right)$, where $J^i(\pi^i)$ is the expectation of accumulated total reward for agent i,

$$J^i\left(\pi^i\right) = \mathrm{E}_{o_0^i \sim O^i}\left[\sum_{t=0}^{\infty} \gamma^t r_t^i | \pi^i\right],\ i = 1, 2, \cdots, N. \tag{2}$$

Let $\pi^i(\cdot|\theta_\pi^i)$ be the parameterized policy for agent i, then the goal of the multi-agent policy gradient is to calculate the gradient $\nabla J(\pi)$ with respect to (w.r.t.) $\{\theta_\pi^i\}_{i=1}^N$, and maximize the $J(\pi)$ cooperatively via gradient ascent.

3.3 TRL

In TRL, the knowledge that is learned in source domain should be transferred to target domain, in order to reduce the training time and improve the performance for the target MDP. Let $M_S \equiv < S_S, A_S, P_S, R_S, \gamma_S >$ denotes the MDP for the source task, and $M_T \equiv < S_T, A_T, P_T, R_T, \gamma_T >$ denotes the MDP for the target task. π_S and π_T denote the policy for source task and target task, respectively. Similarly, $Q_S(s_S, a_S)$ and $Q_T(s_T, a_T)$ denote the source Q-value function and target Q-value function, respectively. Denote Π_S and Π_T as the policy space for the two different MDPs, then the main goal of TRL is finding a transfer functional $T: \Pi_S \rightarrow \Pi_T$, such that

$$\pi_T^* = T(\pi_S^*) \tag{3}$$

where $\pi_S^* \in \Pi_S$ is the optimal policy for source task, and $\pi_T^* \in \Pi_T$ is the target optimal policy that needs to be calculated.

Different from the policy transfer in (3), another approach of TRL aims to transfer the value functions from source task to target task, which is called the value transfer. Value transfer aims to find a transfer functional $G: Q_S(s_S, a_S) \mapsto Q_T(s_T, a_T)$, such that

$$Q_T^* = G(Q_S^*) \tag{4}$$

where Q_S^* and Q_T^* are optimal value functions for the two scenarios.

For a cooperative MAS with incremental number of agents, let M_S^N be the source MDP and $M_T^{N'}$ be the target MDP, where N and N' represent the number of agents and $N' \neq N$. The tasks in these two MDPs are exactly similar, and the knowledge learned in the source task is useful to deal with the target tasks. Hence, there is no need to train from scratch for the student agents like traditional learning algorithms.

4. Knowledge transfer algorithm for MAS with incremental number of agents

In this section, we first build the framework of MARL in source MDP with N agents. Then, we propose the algorithm for the MAS with incremental number of agents, which includes: parameters resetting, experience replay buffer for knowledge transfer, and knowledge transfer for new agents.

4.1 Multi-agent actor-critic in M_S^N

As mentioned in Section 3 the greedy policy for each agent based on its local Q-value function does not work in decentralized execution, because the actions for these agents are executed concurrently in MAS. Although the value decomposition based algorithms are able to learn local Q-value functions and greedy policies such as VDN [12] and QMIX [13], they cannot deal with the MAS with continuous action spaces. Hence, the framework of multi-agent actor-critic (MAAC) with centralized training and decentralized execution is used to implement the MARL in this paper.

For agent i in MAAC with continuous action space, a parameterized deterministic policy $\pi^i(o^i|\theta_\pi^i)$ takes the local observation o^i as input and outputs the actions a^i to make the agent behave independently without considering the observations and actions of the other agents. The parameterized critic takes observations and the actions of all agents to evaluate the state-action values $Q^i\left(\boldsymbol{o}, \boldsymbol{a}|\theta_q^i\right)$ for agent i, where $\boldsymbol{o} = [o^1, o^2, \cdots, o^N]$ and $\boldsymbol{a} = [a^1, a^2, \cdots, a^N]$. We use an neural network that is parameterized by θ_q^i to approximate the evaluated Q-value function for agent i as

$$\hat{Q}^i\left(\boldsymbol{o}, \boldsymbol{a}|\theta_q^i\right) = F_c^i\left(\Phi_c^i(\boldsymbol{o}, \boldsymbol{a})\right), \tag{5}$$

$$\Phi_c^i(\boldsymbol{o}, \boldsymbol{a}) = \phi_c\left(\sum_{j=1}^{N}\left(\boldsymbol{W}_{c,1}^{ij} o^j + \boldsymbol{W}_{c,2}^{ij} a^j + \boldsymbol{b}_c^{ij}\right)\right), \tag{6}$$

where j is the index of agent, $\boldsymbol{W}_{c,1}^{ij}$ is the input weight matrix for the observation variable, $\boldsymbol{W}_{c,2}^{ij}$ is the input weight matrix for the actor variable, and $\boldsymbol{b}_c^{ij}$ is the bias vector. $\phi_c^i(\cdot)$ is the activation function and the $F_c^i(\cdot)$ represents the critic network of agent i. Similarly, the policy of agent i is also approximated by another neural network as

$$\hat{\pi}^i\left(o^i|\theta_\pi^i\right) = F_a^i\left(o^i\right) \tag{7}$$

where $F_a^i(\cdot)$ represents the actor network for agent i.

According to the MADDPG proposed in [10], let $J^i(\pi^i)$ be the accumulated average return defined in (2) and Q^i be the Q-value function for agent i in M_S^N. Then the gradient of π^i w.r.t. θ_π^i is

$$\nabla_{\theta_\pi^i} J^i(\pi^i) = \mathrm{E}\left[\nabla_{\theta_\pi^i} \pi^i(o^i) \nabla_{a^i} Q^i(\boldsymbol{o}, \boldsymbol{a})\,|_{a^i=\pi^i(o^i)}\right]. \tag{8}$$

Equation (8) provides the gradient of the parameters for actor networks, which can be used to update the policies via gradient ascent. If it is assumed that the policies of other agents are fixed when the critic network of agent i is updating, then according to the Bellman equation of single-agent MDP in (1), we can also get

$$Q^{i*}(\boldsymbol{o}, \boldsymbol{a}) = r_t^i + \gamma Q^{i*}\left(\boldsymbol{o}', \boldsymbol{\pi}^*\left(\boldsymbol{o}'\right)\right) \tag{9}$$

where $\boldsymbol{o}' = \left[o^{1'}, o^{2'}, \cdots, o^{N'}\right]$ is the joint observation at next step and $\boldsymbol{\pi}^*(\boldsymbol{o}') = \left[\pi^{1*}\left(o^{1'}\right), \pi^{2*}\left(o^{2'}\right), \cdots, \pi^{N*}\left(o^{N'}\right)\right]$ is the joint optimal policy of the MAS. According to (9), the loss function of the critic network is designed as

$$L_c^i = \mathrm{E}\left[\left(y_t^i \hat{Q}^i\left(\boldsymbol{o}_t, \boldsymbol{a}_t | \theta_q^i\right)\right)^2\right] \tag{10}$$

where $y_t^i = r_t^i + \gamma \hat{Q}_{\text{tar}}^i(\boldsymbol{o}', \boldsymbol{\pi}(\boldsymbol{o}'))$, and $\hat{Q}_{\text{tar}}^i$ is the target Q-value function for agent i. Agent i will get an optimal value function Q^{i*} by minimizing (10). We approximate the expectations in (8) and (10) through sampling in the experience replay buffer D_S. After each step, the transitions $\{< o_t^i, a_t^i, r_t^i, o_{t+1}^i >\}_{i=1}^N$ are stored into the D_S. Then agents sample from D_S to update the parameters. Details of the algorithm are referred from [2] and [10].

We use the MAAC framework with MADDPG algorithm to learn the optimal policies and optimal value functions in M_S^N, which are denoted as π_S^{i*} and Q_S^{i*} $(i = 1, 2, \cdots, N)$, respectively.

4.2 MADDPG with incremental number of agents

Now, we consider there comes M new agents in the current scenario. The number of agents has been increased from N to $N' = N + M$. The previous N agents in M_S^N are treated as teacher agents, and the M new agents are called as student agents. Because the teacher agents have learned the optimal policies in M_S^N, their behaviors in the new environment will be useful to extract the knowledge that was learned before. The teacher agents first modify their network structure to adapt the target MDP $M_T^{N'}$, then they share the experience for student agents. The student agents will deal with the information from these teacher agents and choose the advised output to guide the learning process. In the new scenario, there are three issues need to be handled: (i) Adjust the network structure in the MAAC for both teacher and student agents; (ii) Get the supervising information from teacher agents as the experience knowledge; (iii) Transfer the knowledge from teacher agents to student agents during the learning process.

First of all, the input of the critic networks should be extended to include the components (observation and action variables) of the new agents, such that the MAAC framework still works in $M_T^{N'}$. Secondly, while the teacher agents are updating their policies and value functions, they also provide the advising variables as supervising information to share the experience with the student agents. We call that as online transfer, in which the teacher agents and student agents are trained concurrently [26]. Online transfer is different from traditional offline transfer. In the offline transfer of RL, the agents in target MDP first transfer the knowledge from source MDP to get a jump start of the performance, then they apply the well-initialized parameters to start RL in target MDP [17]. Offline transfer is not efficient compared with the online transfer, because it lacks the new interactive data of the target MDP during the transferring stage.

(i) Teacher agents maintain knowledge from source MDP while the student agents are initialized randomly. We use the subscript T to represent the elements in target MDP. Let $\boldsymbol{W}_{c,1}^m$ and $\boldsymbol{W}_{c,2}^m$ be the input weight matrix for the observation and action variables of mth new agent respectively, then the target Q-values for agent i can be represented by a critic network

$$\hat{Q}_T^i\left(\boldsymbol{o}_T, \boldsymbol{a}_T | \theta_q^i\right) = F_{c,T}^i\left(\Phi_{c,T}^i(\boldsymbol{o}_T, \boldsymbol{a}_T)\right) \tag{11}$$

where

$$\Phi_{c,T}^i(\boldsymbol{o}_T, \boldsymbol{a}_T) = \phi_c\Bigg(\sum_{j=1}^N (\boldsymbol{W}_{c,1}^j o^j + \boldsymbol{W}_{c,2}^j a^j + \boldsymbol{b}_c^j) + \sum_{m=1}^M (\boldsymbol{W}_{c,1}^m o^m + \boldsymbol{W}_{c,2}^m a^m)\Bigg),\ i = 1, 2, \cdots, N + M, \tag{12}$$

and $\boldsymbol{o}_T = [o^1, o^2, \cdots, o^{N+M}]$, $\boldsymbol{a}_T = [a^1, a^2, \cdots, a^{N+M}]$. For teacher agents, the parameters of $\boldsymbol{W}_{c,1}^j$, $\boldsymbol{W}_{c,2}^j$, and $\boldsymbol{b}_c^j$ are maintained from M_S^N, and $\boldsymbol{W}_{c,1}^m$, and $\boldsymbol{W}_{c,2}^m$ are set as zeros.

The target policies for new agents are represented by M reinitialized actor networks

$$\hat{\pi}_T^i\left(o^i | \theta_\pi^i\right) = F_{a,T}^i\left(o^i\right),\ i = N + 1, N + 2, \cdots, N + M, \tag{13}$$

and for teacher agents, the start policies are initialized as

$$\hat{\pi}_T^i\left(o^i | \theta_\pi^i\right) = \pi_S^{i*}\left(o^i\right),\ i = 1, 2, \cdots, N. \tag{14}$$

Here, we assume that for each teacher agent i $(i = 1, 2, \cdots, N)$, the observation spaces of $M_T^{N'}$ is the same as that of M_S^N, i.e., $O_T^i = O_S^i$ (and similarly, $A_T^i = A_S^i$). However, in some scenarios where $O_T^i \neq O_S^i$, there need an observation transition function $\varphi^i : O_T^i \to O_S^i$. φ^i satisfies $\forall o_T^i \in O_T^i$ and $\epsilon > 0$, $R_S^i(\varphi(o_T^i), a_S^i) - R_S^i(o_T^i, a_S^i) < \epsilon$, $\forall a_S^i \in A_S^i$. In this paper, we apply the prior knowledge of the environment to design the transition functions φ^i for network initialization. One can also build an observation-to-observation matching to approximate the φ^i. Another way to adapt the increasing dimensions of the observation is using the attention mechanism, which embeddings the information of entities in the environment to a vector with fixed size [30]. However, once the additional observed information in the target environment is embedded, the optimality of the teacher agents' start policies in (14) cannot be guaranteed. In addition, if the observations are laser scanning data or images, it will be intractable to embedding the information of each entity. Hence, the attention mechanism lacks generality in our settings.

After the initialization of the network parameters, the teacher agents will maintain the knowledge from source MDP and behave in the target environment with a well jump start [17]. The networks of student agents are initialized randomly. In the next part, the teacher agents will provide advising outputs for student agents to transfer the knowledge that was learnt before.

(ii) Augment the transition buffer with supervising signals from the teacher agents. During interacting with the environment, all of the teacher agents and student agents in $M_T^{N'}$ update their policies and values based on the MAAC of (11), (13), and (14). However, in order to make full use of the experience of teacher agents learned in M_N^S, we need to get the advices from teacher agents as the supervised information for student agents. Denote D_T as the experience replay buffer for knowledge transfer (ERBKT) in $M_T^{N'}$. Unlike D_S in Subsection 4.1, both of the supervised information and the transition data at each time step are stored into D_T together for online transfer.

Let a^i_{adv} and q^i_{adv} represent the advised action and advised Q-value for student agent i ($i = N+1, N+2, \cdots, N+M$), respectively. Since the optimal parameters in M_N^S are preserved by the teacher agents, a^i_{adv} and q^i_{adv} can be defined as

$$a^i_{\text{adv}} = \underset{\tilde{\pi}_T^j(o^i)}{\operatorname{argmax}} \Big\{ \hat{Q}_T^1\big(\boldsymbol{o}_T, \tilde{\pi}_1^i(\boldsymbol{o}_T)\big), \hat{Q}_T^2(\boldsymbol{o}_T, \tilde{\pi}_2^i(\boldsymbol{o}_T)), \cdots, \hat{Q}_T^N(\boldsymbol{o}_T, \tilde{\pi}_N^i(\boldsymbol{o}_T)) \Big\} \tag{15}$$

and

$$q^i_{\text{adv}} = \max \Big\{ \hat{Q}_T^1\big(\boldsymbol{o}_T, \tilde{\pi}_1^i(\boldsymbol{o}_T)\big), \hat{Q}_T^2(\boldsymbol{o}_T, \tilde{\pi}_2^i(\boldsymbol{o}_T)), \cdots, \hat{Q}_T^N(\boldsymbol{o}_T, \tilde{\pi}_N^i(\boldsymbol{o}_T)) \Big\} \tag{16}$$

where $\pi^i_j(\boldsymbol{o}_T) = \left[\hat{\pi}_T^1(o^1), \cdots, \hat{\pi}_T^j(o^i), \cdots, \hat{\pi}_T^{N+M}(o^{N+M})\right]$, $j = 1, 2, \cdots, N$. $\tilde{\pi}^i_j$ is the joint actions calculated by the actors of $N+M$ agents, while the ith action is calculated by the actor of agent j, i.e., $a^i = \hat{\pi}_T^j(o^i)$.

For student agents, $\left\{ < o^i_t, a^i_t, r^i_t, o^i_{t+1}, a^i_{\text{adv},t}, q^i_{\text{adv},t} > \right\}_{i=N+1}^{N+M}$ is the transition data at time step t, and it is stored in the experience replay buffer D_T (as shown in the left part of Fig. 1). Thus, the data for student agents in D_T contains not only the information about current environment but also the experience knowledge about the source environment. The advised information calculated here will be useful for student agents when they share a common objective, i.e., the agents in the MAS behave cooperatively to achieve similar goals. For competitive or mixed cooperative-competitive MAS, the policies and values of the teacher agents with different roles may be unable to represent the tasks of the student agents. In that scenario, we can get the advised actions and Q-values from the teacher agents with a same role as the student agents. Therefore, without loss of generality, we mainly consider the learning of cooperative MAS with incremental number of agents.

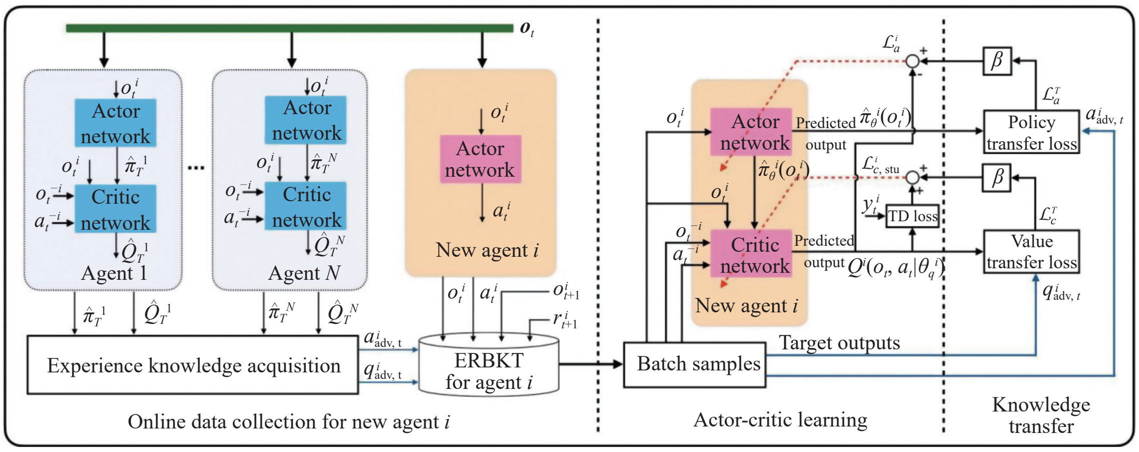


Fig. 1 Algorithm framework of MADDPG-INA

In fact, (15) builds a new policy (we denote it as π^i_{adv}) for agent i to behave in the environment under the guidance of teacher agents. According to the definition of π^i_{adv}, it is not equal to the policy of any teacher agent (the same as the advised Q functions). Hence, we learn from the advised actions and values instead of downloading the parameters of the teacher agents directly, which is different from the algorithms in [39]. Since the teacher agents and student agents share a common objective in the cooperative settings, it is reasonable to evaluate the joint policy through the critics of teacher agents. Denote Q^i_{adv} as the advised Q function of student agent i and $\tilde{\pi}'(\boldsymbol{o}_T) = \left[\hat{\pi}^1_T(o^1), \cdots, \pi^i_{\text{adv}}(o^i), \cdots, \hat{\pi}^{N+M}_T(o^{N+M})\right] (i = N+1, N+2, \cdots, N+M)$, where the policy of the student agent i is replaced by π^i_{adv}. Then, according to (16) we have

$$Q^i_{\text{adv}}\left(\boldsymbol{o}_T, \pi'\right) = q^i_{\text{adv}} \geqslant \hat{Q}^j_T\left(\boldsymbol{o}_T, \tilde{\pi}^i_j(\boldsymbol{o}_T)\right),\ j = 1, 2, \cdots, N. \quad (17)$$

In this point of view, the π^i_{adv} could be a good policy for student agent i. However, the calculation of π^i_{adv} is heavily dependent on the actors and critics of teacher agents, and it is not guaranteed to achieve the optimality of objective function (2). Hence, the student agents should be trained continuously in the target environment to get independent optimal policies, and the a^i_{adv} and q^i_{adv} should be good suggestions to help speed up the training. We will show how to train the actors and critics of student agents in the next part.

(iii) Augment the actor-critic loss functions with additional distillation losses: In this part, we modify the objective function and critic loss for each student agent, and get the gradients to optimize the parameters. With the extended experience replay buffer D_T, the objective function of the student agent i are modified as

$$J^i_{\text{stu}}\left(\hat{\pi}^i_\theta\right) = \mathrm{E}_{D_T}\left[\sum_{t=0}^{\infty} \gamma^t r^i_t | \hat{\pi}^i_\theta\right] - \beta L^T_a\left(a^i_{\text{adv},t}, \hat{\pi}^i_\theta\left(o^i_t\right)\right) \quad (18)$$

where $a^i_{\text{adv},t}$ is the advised action for student agent i at time step t, β is the transfer factor, and L^T_a is the loss function for policy transfer, which can be defined as

$$L^T_a = \mathrm{E}_{D_T}\left[\sum\left(a^i_{\text{adv},t} - \hat{\pi}^i_\theta\left(o^i_t\right)\right)^2\right]. \quad (19)$$

We then get the gradient of the policy $\hat{\pi}^i_\theta$ w.r.t. θ^i_π as

$$\nabla_{\theta^i_\pi} J^i_{\text{stu}}\left(\hat{\pi}^i_\theta\right) = \mathrm{E}_{D_T}\Big[\nabla_{\theta^i_\pi}\hat{\pi}^i_\theta\left(o^i\right)\Big(\nabla_{a^i} Q^i(\boldsymbol{o}, \boldsymbol{a})\,|_{a^i = \hat{\pi}^i_\theta(o^i)} + 2\beta\left(a_{\text{adv}} - \hat{\pi}^i_\theta\left(o^i\right)\right)\Big)\Big]. \quad (20)$$

We call (20) the multi-agent deep deterministic policy gradient with incremental number of agents (MADDPG-INA). It can be found that once we have the advised actions in D_T, the $\nabla_{\theta^i_\pi} J^i_{\text{stu}}(\hat{\pi}^i_\theta)$ will be easy to compute based on the original MADDPG [10]. In addition, for value functions of the new agents, we also modify the loss function (10) as

$$L^i_{c,\text{stu}}\left(\theta^i_q\right) = \mathrm{E}_{D_T}\left[\left(y^i_t \hat{Q}^i\left(\boldsymbol{o}_t, \boldsymbol{a}_t | \theta^i_q\right)\right)^2\right] + \beta L^T_c\left(q_{\text{adv}}, \hat{Q}^i\left(\boldsymbol{o}_t, \boldsymbol{a}_t | \theta^i_q\right)\right) \quad (21)$$

where $y^i_t = r^i_t + \gamma \hat{Q}^i_{\text{tar}}(\boldsymbol{o}', \pi(\boldsymbol{o}'))$, q^i_{adv} is the advised values for agent i, and L^T_c is the loss function for value transfer.

$$L^T_c = \mathrm{E}_{D_T}\left[\sum\left(q^i_{\text{adv}} - \hat{Q}^i\left(\boldsymbol{o}_t, \boldsymbol{a}_t | \theta^i_q\right)\right)^2\right] \quad (22)$$

The gradient of $L^i_{c,\text{stu}}$ w.r.t. θ^i_q is

$$\nabla_{\theta^i_q} L^T_c = \mathrm{E}_{D_T}\left[2\left((1+\beta)\hat{Q}^i\left(\boldsymbol{o}_t, \boldsymbol{a}_t | \theta^i_q\right) - \left(y^i_t + \beta q^i_{\text{adv},t}\right)\right)\nabla_{\theta^i_q}\hat{Q}^i\right]. \quad (23)$$

Remark 1 The choices of distillation loss. As introduced in [26], there are three kinds of losses for policy distillation: negative likelihood loss (NLL), mean square error (MSE) loss, and Kullback-Leibler (KL) divergence loss. According to the simulation results in [26], KL divergence loss performs best compared with NLL and MSE loss. However, the KL divergence loss and NLL are unprocurable when the action space is continuous. Hence, for deterministic policies with continuous action spaces, the MSE loss is used in (19) and (22). While for stochastic policies with discrete action spaces, the distillation losses in (19) and (22) can be replaced by KL divergence loss. In general, if we use the KL divergence loss in that scenario, the advised actions and values in ERBKT should be advised action distributions and advised value vectors, respectively.

With the modified objective functions and value loss functions, the student agents can learn not only from the environment but also from the teacher agents. Specifically, it could be noted that the teacher agents also keep learning to fine tune the network parameters during the learning of student agents, such that the teacher agents could provide more useful advised actions and values to improve the transfer. The method that agent transfers knowledge from the source environment while learning in the target environment is called online transfer [17,38]. Compared with the offline transfer methods (agents transfer knowledge before learning in the target environment) [39], the online transfer is more efficient because the performance of teacher agents is also being improved during transfer learning.

To show the efficiency of the policies and value functions learned with (18) and (21), we provide the following theorem.

Algorithm 1 MADDPG-INA

Input: Number of previous agents N, actor-critic network parameters for M^N_S, number of current agents N'

($N' = N + M$, M is the number of new agents).

```
Initialization: Initialize the network parameters for new agents by (11) and (13), replay buffer $D_T$, the target networks, the new component of the input of teacher agents.
1: for episodes =1 to $N_e$ do
2:   Reset environment and get initialized observations $\{o_0^i\}_{i=1}^{N'}$.
3:   for $t = 0$ to $T-1$ do
4:     $a_t^i \leftarrow \pi_T^i(s_t^i) + \mathcal{N}_t(\pi_T^i(s_t^i), \sigma)\ (i = 1,2,\cdots,N')$.
5:     Get $a_{\mathrm{adv}}^i$ and $q_{\mathrm{adv}}^i$ by (15) and (16), respectively.
6:     Execute $\{a_t^i\}_{i=1}^{N'}$, then get $\{o_{t+1}^i\}_{i=1}^{N'}$ and $\{r_{t+1}^i\}_{i=1}^{N'}$.
7:     Store $\{< o_t^i, a_t^i, r_{t+1}^i, o_{t+1}^i, a_{\mathrm{adv},t}^i, q_{\mathrm{adv},t}^i >\}_{i=N+1}^{N+M}$ and
       $\{< o_t^i, a_t^i, r_{t+1}^i, o_{t+1}^i >\}_{i=1}^{N}$ into $D_T$.
8:     Sample $N_{\mathrm{batch}}$ transitions as a mini-batch data in $D_T$.
9:     for $i = 1$ to $N$ do
10:      Update critic network for agent $i$ by minimizing (10) with the sampled data.
11:      Calculate (8) and update the actor network for agent $i$ by gradient ascent.
12:    end for
13:    for $i = N+1$ to $N'$ do
14:      Calculate (23) and update critic network for agent $i$ by gradient descent.
15:      Calculate (20) and update actor network for agent $i$ by gradient ascent.
16:    end for
17:    Update target networks:
         $\theta_q^{i'} \leftarrow \tau\theta_q^i + (1-\tau)\theta_q^{i'}\ (i = 1,2,\cdots,N')$,
         $\theta_\pi^{i'} \leftarrow \tau\theta_\pi^i + (1-\tau)\theta_\pi^{i'}\ (i = 1,2,\cdots,N')$.
18:    $o_t^i \leftarrow o_{t+1}^i\ (i = 1,2,\cdots,N')$.
19:  end for
20: end for
```

Theorem 1 For a multi-agent MDP with incremental number of agents, if the transfer factor β in (21) satisfies $0 \leqslant \beta < 2/\alpha + \gamma - 1$, then the optimality of the Q-value by minimizing (21) is guaranteed, and the performance of student agents will be no worse than the teacher agents.

Proof Denote Q as the Q-value function space, $Q^{i*} \in Q$ as the optimal Q-value function for student agent i, and let $Q_0^i \in Q$ be the initial value function for iteration. Define a functional $T: Q \to Q$ as

$$\left(TQ_{N_e}^i\right)(\boldsymbol{o},\boldsymbol{a}) = Q_{N_e}^i(\boldsymbol{o},\boldsymbol{a}) - \alpha((1+\beta)Q_{N_e}^i(\boldsymbol{o},\boldsymbol{a}) - (r_{t+1}^i + \gamma\max Q_{N_e}^i\left(\boldsymbol{o}',\cdot\right) + \beta q_{\mathrm{adv},t}^i)). \tag{24}$$

To simplify the theoretical analysis, we use the $\max Q_{N_e}^i(\boldsymbol{o}',\cdot)$ to calculate the target Q-value. With the norm of $\|\cdot\|_{\sup}$, we can get

$$\begin{aligned}
&\left\|Q_{N_e}^i - Q^{i*}\right\|_{\sup} = \left\|TQ_{N_e}^i TQ^{i*}\right\|_{\sup} = \\
&\Big\|(1-\alpha(1+\beta))\left(Q_{N_e-1}^i - Q^{i*}\right) + \\
&\alpha\gamma\left(\max Q_{N_e}^i\left(\boldsymbol{o}',\cdot\right) - \max Q^{i*}\left(\boldsymbol{o}',\cdot\right)\right)\Big\|_{\sup} \leqslant \\
&|1-\alpha(1+\beta-\gamma)|\left\|Q_{N_e-1}^i - Q^{i*}\right\|_{\sup} \leqslant \cdots \leqslant \\
&|1-\alpha(1+\beta-\gamma)|^{N_e}\left\|Q_0^i - Q^{i*}\right\|_{\sup}.
\end{aligned} \tag{25}$$

Given that the $\alpha,\gamma \in (0,1)$ and $0 \leqslant \beta < 2/\alpha + \gamma - 1$, we have $|1-\alpha(1+\beta-\gamma)| < 1$. As a result, $\lim\limits_{N_e\to\infty}|1-\alpha(1+\beta-\gamma)|^{N_e} = 0$, that means

$$\lim_{N_e\to\infty} Q_{N_e}^i - Q_{\sup}^{i*} = 0. \tag{26}$$

Therefore, we can get the optimal Q-value functions for student agents by minimizing (21).

In addition, from (24) and $TQ^{i*} = Q^{i*}$, we can find that the optimal value function for student agent i satisfies

$$Q^{\pi_i^*}(\boldsymbol{o},\boldsymbol{a}) = \frac{1}{1+\beta}\left(r_{t+1}^i + \gamma\max Q^{\pi_i^*}\left(\boldsymbol{o}',\boldsymbol{a}'\right) + \beta q_{\mathrm{adv},t}\right). \tag{27}$$

According to the Bellman optimality of $Q^{\pi_i^*}(\boldsymbol{o},\boldsymbol{a})$ and (16), for each teacher agent j we have:

$$Q^{\pi_i^*}(\boldsymbol{o},\boldsymbol{a}) \geqslant \frac{1}{1+\beta}\left(Q^{\pi_i^*}(\boldsymbol{o},\boldsymbol{a}) + \beta Q_T^{\pi_j}(\boldsymbol{o},\boldsymbol{a})\right) \geqslant Q_T^{\pi_j}(\boldsymbol{o},\boldsymbol{a}) \tag{28}$$

where $Q_T^{\pi_j}(\boldsymbol{o},\boldsymbol{a})$ is the Q-value function with policy π_j of teacher agent j. As a result, the performance of student agent will be no worse than that of the teacher agents. □

According to Theorem 1, the positive transfer of MADDPG-INA for new agents can be guaranteed. Fig. 1 shows the learning framework for ith new agent. The knowledge is transferred from teacher agents (agents $1,2,\cdots,N$) to student agent (agent i). The left part is the online data collection for new agent i, which includes the experience knowledge acquisition and the traditional interaction data collection. The central part is the actor-critic learning for agent i. The right part is the knowledge transfer mechanism which takes the advised actions and Q-values as the supervising information for the actor network and critic network of agent i. In addition, the teacher agents also keep learning in the target environment without the parts of online data collection and knowledge transfer in Fig. 1. The pseudo code of the proposed MADDPG-INA is provided in Algorithm 1. We also build the target actor-critic networks [1] for each agent to calculate the target values, and modify the parameters by soft updating with a factor τ [2]. $\mathcal{N}_t\left(\pi_T^i(s_t^i),\sigma\right)$ is the Gaussian noise at step t which is added to the policies during exploration, and the σ is the variance of $\mathcal{N}_t$.

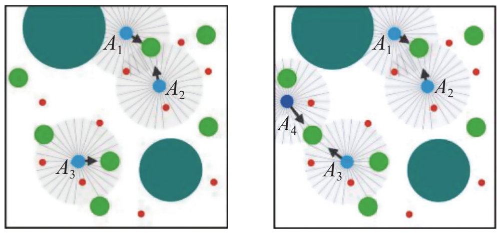
5. Simulation results

In this section, we evaluate the proposed method in two multi-agent simulation environments: waterworld with multi-agent, and cooperative box-pushing. All simulations are run at a desktop with Intel Core i7-7700k CPU@4.20 GHz.

We compare the following methods in our simulations: the MADDPG algorithm in [10], the DyMA-CL in [30], and our proposed MADDPG-INA algorithm. The network architecture of MADDPG is designed according to [10]. The architecture of DyAN in DyMA-CL algorithm is designed according to [30].

5.1 Waterworld with multi-agent

In the environment of waterworld with multi-agent, agents move in a shared area to capture the moving food targets cooperatively, and avoid the poison targets and obstacles [40]. Each agent gets the observations by thirty range-limited sensors with uniform angular spacing, and takes actions of a two-dimensional continuous force. The observation variable for each agent is 212-dimensional, and it includes the distances and velocities of the moving food targets, moving poison targets, static obstacles, and other agents. Fig. 2(a) shows the waterworld simulation environment with three agents. For agent i, it will receive a positive reward of +1.0 if it encounters a food target, and will receive +20 if it captures the food target with other agents cooperatively. We add a penalty of −2.0 if the agent encounters with a poison target, and a penalty of −1.0 if the agent collides with the obstacles, boundary or other agents.



(a) Three-agent scenario (b) Four-agent scenario with one incremental agent

●: Poison target; ●: Food target; ●: Static obstacle; ●: Agent; ●: New agent.

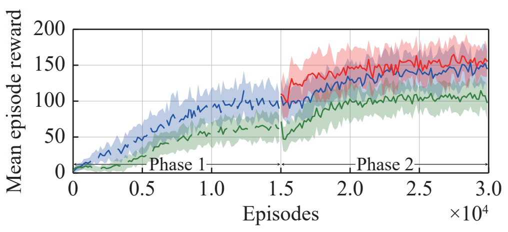
Fig. 2 Simulation A: waterworld with multi-agent

Details of the parameter setting are shown in Table 1. We use three-layer MLPs as the actor networks and the critic networks for MADDPG. The RELU activation functions [41] are used for the hidden layers of these neural networks. For each agent, the input of the actor network is the local observation, and the output is the deterministic action. The critic network for each agent takes the observations and actions for all agents as input, and it outputs the Q value for the agent. First, we train the MARL algorithms with three agents in this environment as the source models, and get the convergent parameters for each agent. Then, we consider the number of new agents $M = 1,2,3$ separately. The parameters of new agents are initialized randomly. As shown in Fig. 2(b), the new agent (or student agent) should learn to cooperate with the current three agents (or teacher agents) to get more rewards, and the current three agents should also cooperate with the new agent. Since there are more agents in the target environment, the optimal policies of the holistic MAS will get more rewards than that in the source environment.

Table 1 Parameter settings

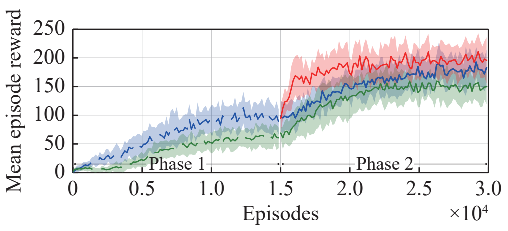
Parameter	Simulation A	Simulation B
Discount factor γ	0.99	0.99
Transfer factor β	1.0	20.0
Learning rate critic,α_c	0.001	0.001
Learning rate critic,α_a	0.001	0.001
Soft update factor τ	0.001	0.001
Batch size N_{batch}	64	64
Replay buffer size M	100 000	100 000
Initialize exploration variance σ	0.2	0.5
Hidden layer units	[100, 50, 25]	[64, 64]
Episode number N_e	15 000	24 000

For each compared method, we test the learning model at every 50 episodes, and get the performance by averaging the accumulated rewards over 10 trails (each trail contains 50 steps) at that training stage. Overall, we get the mean episode rewards over 15 experiments. As shown in Fig. 3, the curves during the learning process are separated into two phases. In phase 1, the environment contains three agents. The training processes are stopped and the models are stored after 15 000 episodes. In phase 2, we increase the number of agents and train the target models continuously to evaluate the performance. Fig. 3 is about the average rewards of the holistic team, while Fig. 4 shows the performance of the new agent(s). From both Fig. 3 and Fig. 4, it can be found that the proposed MADDPG-INA algorithm can achieve significant improvements of the learning speed and the final performance compared with the MADDPG, which trains the new agent(s) without any transfer. For DyMA-CL, the new agent(s) can perform better than MADDPG while the holistic team fails to learn a global optimal joint policy. DyMA-CL cannot learn a good policy in phase 1 for source models, which limits its performance in the future phases. Fig. 5 shows the testing performance for the models that perform best over the 15 experiments. Overall, we can find that the MADDPG-INA has got the best and stable testing performance compared with the baselines. The improvements for the performance of new agents in each scenario are significant.



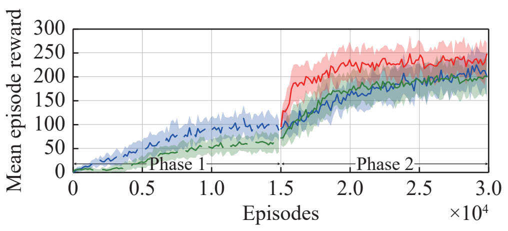
---: MADDPG: (three agents); ---: DyMA-CL: (three agents);
—: MADDPG: (one new agent); —: DyMA-CL: (one new agent);
—: MADDPG-INA: (one new agent).

(a) One new agent



---: MADDPG: (three agents); ---: DyMA-CL: (three agents);
—: MADDPG: (two new agents); —: DyMA-CL: (two new agents);
—: MADDPG-INA: (two new agents).

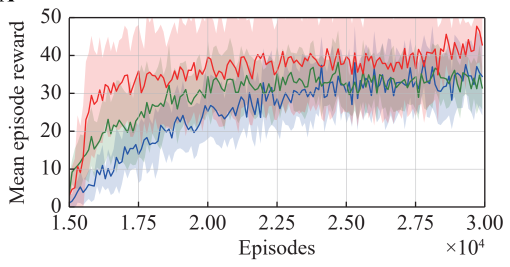
(b) Two new agents



(c) Three new agents

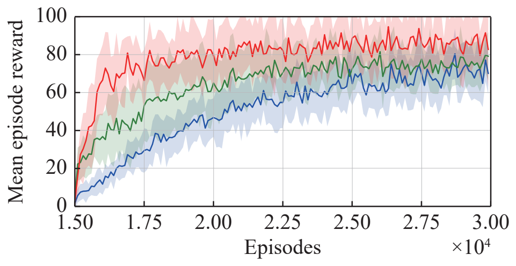
---: MADDPG: (three agents); ---: DyMA-CL: (three agents);
—: MADDPG: (three new agents); —: DyMA-CL: (three new agents);
—: MADDPG-INA: (three new agents).

Fig. 3 Compared learning performance of the whole team for simulation A



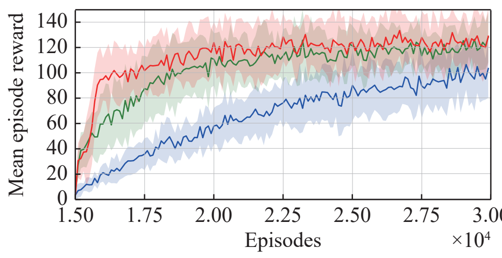
—: MADDPG: (one new agent);
—: DyMA-CL: (one new agent);
—: MADDPG-INA: (one new agent).

(a) One new agent



—: MADDPG: (two new agents);
—: DyMA-CL: (two new agents);
—: MADDPG-INA: (two new agents).

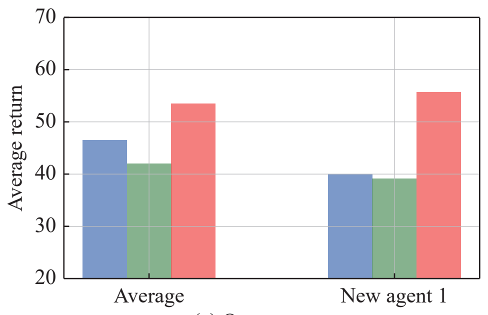
(b) Two new agents



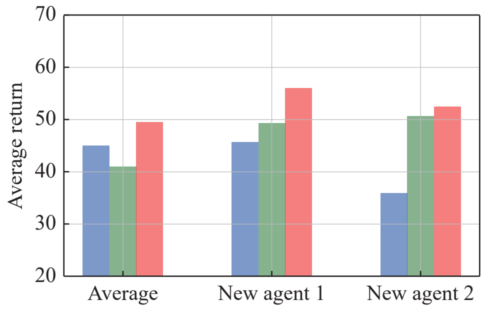
—: MADDPG: (three new agents);
—: DyMA-CL: (three new agents);
—: MADDPG-INA: (three new agents).

(c) Three new agents

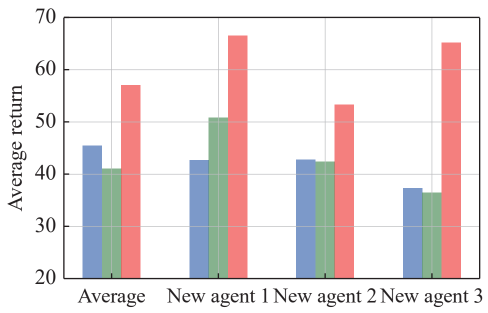
Fig. 4 Learning performance of new agent(s) for simulation A



(a) One new agent



(b) Two new agents



(c) Three new agents

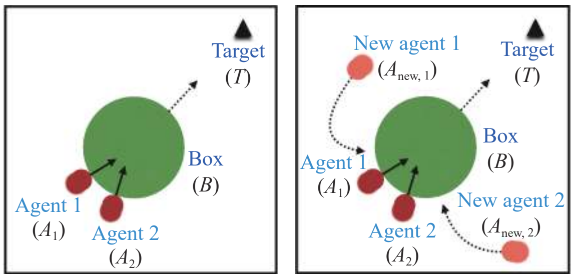
🟦: MADDPG; 🟩: DyMA-CL; 🟥: MADDPG-INA.

Fig. 5 Average return of the models that get the best performance in simulation A

From the above results in simulation A, when the new agents interact with the environment and search the optimal solutions by RL, they also ask the experienced previous agents for advices and get the supervising information to take the shortcut. Because the cooperative tasks are similar for both source and target environments, new agents can get the useful supervising information from previous agents and avoid the negative transfer. As a result, MADDPG-INA can reduce the interactive samples and speed up the learning process of new agents.

5.2 Cooperative box-pushing

The cooperative box-pushing environment consists of multiple agents, a heavy box and a marked target. For example, as shown in Fig. 6(a), two agents (A_1 and A_2) aim to push a heavy box (B) to the target (T) cooperatively as soon as possible. Each agent can observe its position, velocity, and the relative positions of the other agents, the box, and the target. They take two-dimensional (north and east) continuous actions to behave in the environment. The positions of agents, box, and the target are initialized randomly. A_1 and A_2 will get a shared reward $r_t = -\text{dist}(T, B)$, where $\text{dist}(T, B)$ represents the Euclidean distance from target to the box. In addition, each agent will be penalized with -0.1 if it fails to catch the box, and it will also be penalized with -0.1 if it collides with other agents or the wall. Specifically, the box can be pushed to move if and only if all agents in the environment apply forces to it together, which makes the problem more difficult [10,24]. Then, we increase the weight of the box, such that it can be pushed if and only if four agents push it together. Hence, there needs two additional agents to finish the target task, as shown in Fig. 6(b).



(a) Two-agent scenario (b) Four-agent scenario with two incremental agents

Fig. 6 Simulation B: box-pushing

In this simulation, we separate the learning process into four phases. In the first phase, two agents push the box cooperatively. We train the two-agent MARL models as the source solutions for the next phase. Then, there are two new agents being added at each phase. The parameter settings for this simulation are listed in Table 1. Two-layer MLPs are used as the actor and critic networks for each agent. The activation function, input and output of each agent are the same as those in simulation A. Different from simulation A, the dimension of the observation for each agent is determined by the number of agents. Hence, if there are new agents in the target environment, we should first modify the input structure for each teacher agent to adapt the target MDP.

The performance of the compared algorithms is tested by averaging the accumulated rewards over 10 trails (each trail contains 80 steps) at every 20 episodes. Fig. 7 shows the simulation results of the mean episode rewards over 15 experiments. The learning process is separated into four phases. According to the environment setting, the box can be pushed to move if and only if all agents apply forces to it together, so the beginning performance in the next phases is as worse as that in Phase 1. In this setting, agents can push the box with bigger forces when there are more agents. However, the more agents in the environment, the more difficult for them to find the box concurrently.

In Phase 1, we train the MADDPG and DyMA-CL separately with two agents as the source model for the next phase. For MADDPG-INA, it uses the MADDPG in Phase 1 as the source model in Phase 2. After 6 000 episodes, we start the learning process in the next phase with two new agents. Fig. 7 shows the learning performance of the compared algorithms across four phases. In this problem, the agents cannot get any positive reward until they catch and push the box cooperatively. Hence, the DyMA-CL with independent Q-value functions fails to learn a cooperative optimal policy. From the result of the second phase in Fig. 7, we find that the MADDPG-INA algorithm can not only converge to the optimal solution faster but also get better performance than the other algorithms. With the limitation of the space size in the environment, when there are six agents in Phase 3 and eight agents in Phase 4, the collision penalty will happen more frequently. Therefore, we can see the decreases of the final performance in the third and fourth phases. However, it is still obvious that MADDPG-INA can perform better than the other algorithms with a faster learning speed. Especially in Phase 4, the MADDPG is hardly to converge while the MADDPG-INA can still learn with an increasing performance. Fig. 8 is the average return of the models that perform best among the 15 experiments. It is also found that the MADDPG-INA can outperform the other methods with significant improvements in the last three phases.

In conclusion, if the number of agents is increasing in the cooperative box-pushing environment, it will be more difficult to find the joint optimal solutions. Firstly, the input of the Q values should be extended to include the observations and actions of the new agents. Secondly, the search space for all agents will also be enlarged. For example, the new agents should first learn to navigate to the box, then they push the box with the previous agents cooperatively. Without the help of previous agents, the new agents have to learn cooperative behaviors from scratch, so it will be hard to finish the task by the whole agents cooperatively. That is why the other algorithms fail to perform well in the target environments with incremental number of agents. However, by taking advantage of the experience knowledge that was learnt previously, the new agents can avoid many unnecessary trails to find the optimal solutions with the supervising information from previous agents. That is also the reason why MADDPG-INA can outperform other methods.

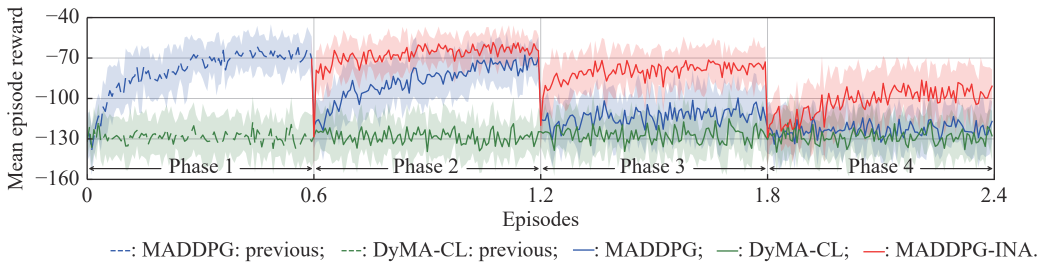


Fig. 7 Compared learning performance for Simulation B

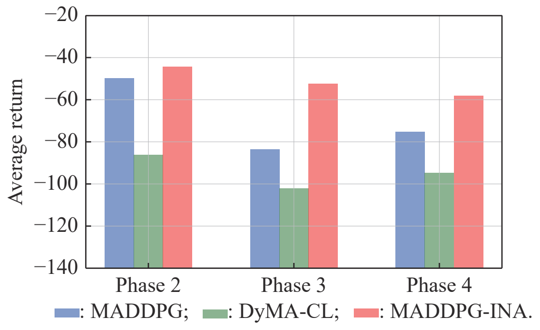


Fig. 8 Average return of the models that get the best performance in Simulation B

6. Conclusions

In this paper, we propose a new algorithm called MADDPG-INA to solve the cooperative MAS problem with incremental number of agents. The core idea of the paper is to transfer the knowledge from teacher agents to student agents to accelerate the learning process. For each student agent, we extend the traditional experience replay buffer as ERBKT to collect the advised values and actions from teacher agents, and get the supervising information. Then the student agents in the proposed algorithm learn not only from the target environment, but also from the teacher agents to transfer the knowledge that is useful for target tasks. With the advantage of the experience knowledge from teacher agents, the new agents in the MADDPG-INA algorithm can avoid many unnecessary trails, and find the optimal solutions without the need for large amounts of data. The simulation environments of waterworld and box-pushing are applied to implement the cooperative MAS with incremental number of agents, and the simulation results have verified the superiority of the proposed algorithm.

References

[1] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning. Nature, 2015, 518(7540): 529–533.
[2] LILLICRAP T P, HUNT J J, PRITZEL A, et al. Continuous control with deep reinforcement learning. https://arxiv.org/abs/1509.02971.
[3] DONG L, YUAN X, SUN C Y. Event-triggered receding horizon control via actor-critic design. Science China Information Sciences, 2020, 63(5): 150210.
[4] FUJIMOTO S, MEGER D, PRECUP D. A deep reinforcement learning approach to marginalized importance sampling with the successor representation. Proc. of the 38th International Conference on Machine Learning, 2021: 3518–3529.
[5] LI Y, QI X H, LI X D, et al. Deep reinforcement learning and its application in autonomous fitting optimization for attack areas of UCAVs. Journal of Systems Engineering and Electronics, 2020, 31(4): 734–742.
[6] GAO X, FANG Y W, WU Y L. Fuzzy Q learning algorithm for dual-aircraft path planning to cooperatively detect targets by passive radars. Journal of Systems Engineering and Electronics, 2013, 24(5): 800–810.
[7] FANG M, GROEN F C. Collaborative multi-agent reinforcement learning based on experience propagation. Journal of Systems Engineering and Electronics, 2013, 24(4): 683–689.
[8] TAMPUU A, MATIISEN T, KODELJA D, et al. Multiagent cooperation and competition with deep reinforcement learning. PloS One, 2017, 12(4): e0172395.
[9] NGUYEN T T, NGUYEN N D, NAHAVANDI S. Deep reinforcement learning for multiagent systems: a review of challenges, solutions, and applications. IEEE Trans. on Cybernetics, 2020, 50(9): 3826–3839.
[10] LOWE R, WU Y, TAMAR A, et al. Multi-agent actor-critic for mixed cooperative-competitive environments. Proc. of the Annual Conference on Neural Information Processing Systems, 2017: 6379–6390.
[11] FOERSTER J, FARQUHAR G, AFOURAS T, et al. Counterfactual multi-agent policy gradients. Proc. of the AAAI Conference on Artificial Intelligence, 2018: 2974–2982.
[12] SUNEHAG P, LEVER G, GRUSLYS A, et al. Value-decomposition networks for cooperative multi-agent learning based on team reward. Proc. of the 17th International Conference on Autonomous Agents and MultiAgent Systems, 2018: 2085–2087.
[13] RASHID T, SAMVELYAN M, SCHRODER D W, et al.

Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. Proc. of the 35th International Conference on Machine Learning, 2018: 4292–4301.
[14] PAN S J, YANG Q. A survey on transfer learning. IEEE Trans. on Knowledge and Data Engineering, 2010, 22(10): 1345–1359.
[15] LONG M S, ZHU H, WANG J M, et al. Deep transfer learning with joint adaptation networks. Proc. of the 34th International Conference on Machine Learning, 2017: 2208–2217.
[16] ZHUANG F Z, QI Z Y, DUAN K Y, et al. A comprehensive survey on transfer learning. Proceedings of the IEEE, 2021, 109(1): 43–76.
[17] TAYLOR M E, STONE P. Transfer learning for reinforcement learning domains: a survey. Journal of Machine Learning Research, 2009, 10(7): 1633–1685.
[18] LAZARIC A. Transfer in reinforcement learning: a framework and a survey. Proc. of the Reinforcement Learning, 2012: 143–173.
[19] ZHU Z D, LIN K X, ZHOU J Y. Transfer learning in deep reinforcement learning: a survey. https://arxiv.org/abs/2009.07888.
[20] TAYLOR M E, STONE P. Behavior transfer for value-function-based reinforcement learning. Proc. of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, 2005: 53–59.
[21] BOUTSIOUKIS G, PARTALAS I, VLAHAVAS I. Transfer learning in multi-agent reinforcement learning domains. Proc. of the European Workshop on Reinforcement Learning, 2011: 249–260.
[22] SILVA F L D, COSTA A H R. A survey on transfer learning for multiagent reinforcement learning systems. Journal of Artificial Intelligence Research, 2019, 64: 645–703.
[23] SILVA F L D, WARNELL G, COSTA A H R, et al. Agents teaching agents: a survey on inter-agent transfer learning. Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems, 2020: 2165–2167.
[24] WADHWANIA S, KIM D K, OMIDSHAFIEI S, et al. Policy distillation and value matching in multiagent reinforcement learning. Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2019: 8193–8200.
[25] YANG T P, WANG W X, TANG H Y, et al. Transfer among agents: an efficient multiagent transfer learning framework. https://arxiv.org/abs/2002.08030.
[26] RUSU A A, COLMENAREJO S G, GULCEHRE C, et al. Policy distillation. http://arxiv.org/abs/1511.06295.
[27] PARISOTTO E, BA J L, SALAKHUTDINOV R. Actor-mimic: deep multitask and transfer reinforcement learning. http://arxiv.org/abs/1511.06342.
[28] OMIDSHAFIEI S, PAZIS J, AMATO C, et al. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. Proc. of the 34th International Conference on Machine Learning, 2017: 2681–2690.
[29] LI Z, BARENJI A V, JIANG J Z, et al. A mechanism for scheduling multi robot intelligent warehouse system face with dynamic demand. Journal of Intelligent Manufacturing, 2020, 31(2): 469–480.
[30] WANG W X, YANG T P, LIU Y, et al. From few to more: large-scale dynamic multiagent curriculum learning. Proc. of the AAAI Conference on Artificial Intelligence, 2020: 7293–7300.
[31] CHEN D, LI Z J, WANG Y Q, et al. Deep multi-agent reinforcement learning for highway on-ramp merging in mixed traffic. https://arxiv.org/abs/2105.05701.
[32] CZARNECKI W M, PASCANU R, OSINDERO S, et al. Distilling policy distillation. Proc. of the 22th International Conference on Artificial Intelligence and Statistics, 2019: 1331–1340.
[33] CHEN G. A new framework for multi-agent reinforcement learning−centralized training and exploration with decentralized execution via policy distillation. Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems, 2020: 1801–1803.
[34] TAYLOR A, DUSPARIC I, GUÉRIAU M, et al. Parallel transfer learning in multi-agent systems: what, when and how to transfer. Proc. of the International Joint Conference on Neural Networks, 2019. DOI: 10.1109/IJCNN.2019.8851784.
[35] AGARWAL A, KUMAR S, SYCARA K P, et al. Learning transferable cooperative behavior in multi-agent teams. Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems, 2020: 1741–1743.
[36] SUTTON R S, BARTO A G. Reinforcement learning: an introduction. Cambridge: The MIT Press, 2018.
[37] BELLMAN R, KALABA RE. Dynamic programming and modern control theory. New York: Academic Press, 1965.
[38] ZHAO P L, HOI S C. OTL: a framework of online transfer learning. Proc. of the 27th International Conference on Machine Learning, 2010: 1231–1238.
[39] LI Y Y, ZHOU W, WANG H M, et al. Improving fast adaptation for newcomers in multi-robot reinforcement learning system. Proc. of the IEEE Smart World, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, 2019: 753-760.
[40] GUPTA J K, EGOROV M, KOCHENDERFER M. Cooperative multi-agent control using deep reinforcement learning. Proc. of the International Conference on Autonomous Agents and Multiagent Systems, 2017: 66–83.
[41] NAIR V, HINTON G E. Rectified linear units improve restricted boltzmann machines. Proc. of the 27th International Conference on Machine Learning, 2010: 807–814.

Biographies

LIU Wenzhang was born in 1993. He is a Ph.D. student in the School of Automation, Southeast University, Nanjing, China. He received his B.S. degree in engineering from Jilin University, Changchun, China, in 2016. He is currently working toward his Ph.D. degree in control science and engineering at Southeast University. His research interests include machine learning, deep reinforcement learning, optimal control, and multi-agent cooperative control.
E-mail: wzliu@seu.edu.cn.

DONG Lu was born in 1990. She received her B.S. degree in physics and Ph.D. degree in electrical engineering from Southeast University, Nanjing, China, in 2012 and 2017, respectively. She is currently an associate professor with the School of Cyber Science and Engineering, Southeast University. Her current research interests include adaptive dynamic programming, event-triggered control, nonlinear system control and optimization.
E-mail: ldong90@seu.edu.cn

LIU Jian was born in 1992. He received his B.S. and Ph.D. degrees from the School of Automation and Electrical Engineering, University of Science and Technology Beijing, Beijing, China, in 2015 and 2020, respectively. From September 2017 to September 2018, he was a joint training student with the Department of Mathematics, Dartmouth College, Hanover, NH, USA. From 2020 to 2021, he was a postdoctoral fellow with the School of Automation, Southeast University, Nanjing, China, where he is currently an associate professor. His current research interests include multi-agent systems, nonlinear control, event-triggered control, and fixed-time control.
E-mail: bkliujian@163.com

SUN Changyin was born in 1975. He received his B.S. degree in applied mathematics from the College of Mathematics, Sichuan University, Chengdu, China, in 1996, and M.S. and Ph.D. degrees in electrical engineering from Southeast University, Nanjing, China, in 2001 and 2004, respectively. He is currently a professor with the School of Automation, Southeast University, Nanjing, China. His current research interests include intelligent control, flight control, and optimal theory. He is an associate editor of the IEEE Transactions on Neural Networks and Learning Systems, Neural Processing Letters, and the IEEE/CAA Journal of Automatica Sinica.
E-mail: cysun@seu.edu.cn