Efficient Exploration for Multi-Agent Reinforcement Learning via Transferable Successor Features

Wenzhang Liu, Lu Dong, Member, IEEE, Dan Niu, Member, IEEE, and Changyin Sun, Senior Member, IEEE

Abstract—In multi-agent reinforcement learning (MARL), the behaviors of each agent can influence the learning of others, and the agents have to search in an exponentially enlarged joint-action space. Hence, it is challenging for the multi-agent teams to explore in the environment. Agents may achieve suboptimal policies and fail to solve some complex tasks. To improve the exploring efficiency as well as the performance of MARL tasks, in this paper, we propose a new approach by transferring the knowledge across tasks. Differently from the traditional MARL algorithms, we first assume that the reward functions can be computed by linear combinations of a shared feature function and a set of task-specific weights. Then, we define a set of basic MARL tasks in the source domain and pre-train them as the basic knowledge for further use. Finally, once the weights for target tasks are available, it will be easier to get a well-performed policy to explore in the target domain. Hence, the learning process of agents for target tasks is speeded up by taking full use of the basic knowledge that was learned previously. We evaluate the proposed algorithm on two challenging MARL tasks: cooperative box-pushing and non-monotonic predator-prey. The experiment results have demonstrated the improved performance compared with state-of-the-art MARL algorithms.

Index Terms—Knowledge transfer, multi-agent systems, reinforcement learning, successor features.

I. INTRODUCTION

REINFORCEMENT learning (RL) has made great success recently in various sequential decision-making problems, such as video games [1], [2], continuous control [3]–[5], and navigations [6]–[8], etc. However, there exist challenges of extending the RL algorithms from single agent to multi-agent systems (MAS), in which multiple agents behave together in a common environment to achieve cooperative and (or) competitive tasks [9]. On the one hand, for each agent in an MAS, its local next-state distribution may be influenced by other agents, which will bring uncertainties to the agent when the policies of others are updated. Hence, the agents will encounter the challenge of non-stationarity during the learning process [10]–[13]. On the other hand, the joint-action space of an MAS will be enlarged exponentially as the number of agents grows. That will make it more difficult for agents to explore in the joint-action space, and the agents may fall into local optimal solutions or fail to solve the tasks [14]. For a practical example, there are two agents learning to push a heavy box cooperatively from one place to somewhere else. The box can be pushed to move if and only if two agents force it together. Agents should first catch the box cooperatively, and then push it to explore in the environment. When the environment is complex and the reward function is sparse, it will be hard for agents to find the target place where the box should be pushed into.

A direct idea of extending RL to multi-agent reinforcement learning (MARL) is independent learning [15], [16], in which every agent learns to search for the optimal solution without considering the policies of others. As is mentioned, that will cause non-stationarity of the next-state distribution. Hence, the paradigm of centralized training with decentralized execution (CTDE) is widely used to learn optimal joint-policies [17], [18]. CTDE collects global information of all agents during training while executes actions according to the local observation of each agent. CTDE is useful to stabilize the training. However, a good exploring joint-policy is also important for an MAS with sparse rewards and continuous state and action spaces. Without useful data being collected during exploration, the algorithm may fall into the local optimal solutions and fail to accomplish the complex tasks. Inspired by the idea of transfer reinforcement learning (TRL) [19], we propose to first learn some easier tasks as the basic knowledge, and then calculate a good initialized policy through knowledge transfer to guide the exploration [20], [21]. The TRL generalizes the algorithms across tasks by leveraging the knowledge learned in previous, which is different from the traditional RL algorithms that learn within tasks.

Previous work shows the successor features (SFs) based TRL algorithms can provide a good initialization for target tasks to speed up the learning [22]–[26]. SFs decouple the dynamics of the environment from the reward and value functions, so they can be seen as robust features to describe different tasks [27]. When the reward function of the environ-
ment changes, the value function for the target task can be calculated quickly. From that point of view, one can first learn some related and easier tasks, and then start the learning of complex target tasks with a good initialized policy to improve the exploration. Hence, in this paper, we aim to learn centralized SFs instead of critics to evaluate different tasks, and then improve the exploration efficiency for target tasks through knowledge transfer. The core hypothesis of SFs based RL algorithms is the reward functions can be combined linearly by a shared feature function (reward features) and a set of task-specific weight vectors. Thus, once we have learned the task weights in the target domain, it will be easier to get the values through linear combination with the SFs in the source domain. That is why the SFs-based RL benefits knowledge transfer across tasks.

Most of the existing SFs-based algorithms focus on single agent systems. Since each agent in an MAS behaves in a common environment, decoupling the dynamics of the environment from the reward and value functions could be useful to share the learning. Because the reward and value functions can be computed by linear combination, it is possible to deal with unseen tasks in target domain without learning from scratch. Taking the box-pushing as an example, the agents can first learn how to push the box cooperatively to some basic places, and store the SFs as source knowledge. Then, they can get the value functions of more complex tasks through linear combinations of the source SFs and the target task weights. Finally, it will be easier for agents to get a new policy with efficient exploring ability. In this paper, we aim to deal with the MAS problems with continuous action and state spaces. To learn a deterministic policy for each agent, the framework of multi-agent deep deterministic policy gradient (MADDPG) [28] is used to implement CTDE. Differently from the original MADDPG algorithm, we build a global successor feature network (GSFN) instead of multiple agent-wise critic networks to evaluate the joint-policy. A shared GSFN can avoid recalculation of the iterative value functions for agents, and generalize the original MADDPG across different tasks. The main contributions of this paper are listed as follows:

1) We implement the MARL with continuous action and state spaces by applying the SFs to the MADDPG algorithm. A new algorithm called multi-agent deep deterministic policy gradient with successor features (MADDPG-SFs) is proposed. The critic networks in traditional MADDPG are replaced by a shared GSFN to evaluate the long-term accumulated features. The algorithm calculates the value functions of different tasks by linear combinations of the SFs and task-specific weights;

2) For the new and more complex tasks in the target domain, we improve the exploring efficiency of the MARL by transferring the knowledge that was learned before. A new algorithm called multi-agent deep deterministic policy gradient with successor features and knowledge transfer (MADDPG-SFKT) is proposed. The algorithm takes full use of the knowledge in the source domain, and calculates a well-performed joint policy to implement efficient exploration in the target environment.

The rest of the paper is organized as follows. The related work is introduced in Section II. Section III is about the research background and problem formulation. The details about our proposed algorithms are provided in Section IV. Section V shows the simulation results to verify the proposed algorithms. Finally, the conclusion is drawn in Section VI.

II. RELATED WORK

Recently, the successes of deep RL in single agent systems have raised increasing interest in MARL. For example, Tampuu et al. [16] directly extended the deep Q-learning (DQN) [1] to MASs, and trained each agent via independent Q-learning (IQL). IQL learns without considering the information of others, which may lead to non-stationarity and local optimal solutions [10], [11]. To solve that problem, the CTDE based learning framework is proposed to stabilize the learning, such as value decomposition networks (VDN) [29], Q-mixing (QMIX) [30], MADDPG [28], etc. The VDN and QMIX are both value based algorithms like IQL. Hence, they cannot provide deterministic policy for each agent with continuous action space. Besides, the calculation of the holistic Q-values in VDN or QMIX has limited the representations of the Q-functions. Differently from that, Lowe et al. [28] proposed MADDPG which was based on actor-critic and policy gradient algorithms. MADDPG collects the observations and actions of all agents to approximate the Q-value of each agent through a critic network. The policies are approximated by agent-specific actor networks. It could be an idealized setting for MARL, especially for learning deterministic policies with continuous action spaces. To make agents cooperate with each other more efficiently, Mao et al. proposed attention MADDPG (ATT-MADDPG) [31]. ATT-MADDPG enhances the centralized critic with an attention mechanism to model the policies of the teammates. However, the MADDPG-based algorithms still have limitations of exploration for complex tasks and sparse rewards.

Existing research suggests that TRL can reduce the exploration of target tasks, and avoid learning from scratch compared with traditional RL algorithms [19]–[21]. In MARL, transferring knowledge across agents [32] and tasks [33], [34] plays an important role in generalizing the algorithms to the target domain. For instance, Yang et al. [35] proposed multi-agent option-based policy transfer (MAOPT) to learn when to transfer knowledge among agents. In [36], Omidshafiee et al. learned multiple tasks for agents with partial observation through a distilling policy. However, these TRL methods extract the knowledge from the behaviors of agents, which cannot describe the relationships between the source tasks and the target tasks. By assuming the dynamics of the environment is unchanged, and the tasks differ only in the reward functions, the idea of SFs is used to extract the useful knowledge of transferring for the target tasks.

In [37], Dayan expressed the Q-value as an inner product of the immediate reward function and the proposed successor representation (SR). The SR is defined as a cumulative of discounted future states [37], [38]. SR is seen as a third alternative of RL algorithms following the model-based and model-free RL [22]. In [22], Kulkarni et al. presented deep SR (DSR) to implement the end-to-end deep reinforcement learning.
Barreto et al. extended the idea of SR from discrete to continuous spaces, and proposed the successor features and generalized policy improvement (SF&GPI) algorithms for TRL to improve the learning of unseen tasks [23, 26]. In [24], Barreto et al. further generalized the SF&GPI to any set of tasks, and used the reward features in source domain as the features of target rewards. Hence, we can build the source tasks through some basic tasks, and then form the reward features of target tasks, such as [25]. By applying the DSR and decoupling the option-value functions, Yang et al. [35] also proposed MAOPT with successor representation option (MAOPT-SRO) to learn what advice to provide and when to terminate it during the knowledge transfer among agents. In order to improve the exploration of MARL, Gupta et al. [39] proposed the universal value exploration (UneVEn) algorithm by combining the multi-agent universal SFs and the generalized policy improvement. UneVEn samples a set of related tasks near the target task and learns these tasks simultaneously to achieve better coordination between agents.

In this paper, we develop a new MARL algorithm based on SFs. To improve the efficiency of the exploration for complex tasks, we first learn some basic tasks and build the reward features. Then we combine the SFs linearly with task weights to implement knowledge transfer.

III. PRELIMINARIES

In this section, we introduce the background and problem formulation about reinforcement learning, multi-agent reinforcement learning, and the successor features for transfer reinforcement learning.

A. Reinforcement Learning

RL is a family of algorithms that agent learns to take actions to interact with the environment in order to maximize the cumulative rewards [40]. The interaction between agent and the environment is built on a Markov decision process (MDP). As usual, we describe an MDP as a tuple \( M = \langle S, A, P, R, \gamma \rangle \). The sets \( S \) and \( A \) represent the states and actions, respectively. \( P : S \times A \times S \mapsto [0,1] \) is the transition probability, that determines the next-state \( (s_{t+1} \in S) \) distribution of the agent after executing action \( a_t \in A \) in state \( s_t \in S \) at time step \( t \). Most of the RL algorithms (as well as in this paper) assume that the transition probability \( P \) is unknown. They need to sample the agent-environment interaction data from the probability distribution determined by \( P \). The reward function in \( M \) is defined as \( R : S \times A \times S \mapsto \mathbb{R} \), and we use \( r_t \) to represent the immediate reward that agent receives at time step \( t \). \( \gamma \in [0,1) \) is the discount factor.

During the interaction with environment, agent takes actions by following a policy that is defined as \( \pi : S \mapsto A \). The essential goal of an RL agent is to find an optimal policy \( \pi^* \), which can maximize the cumulative discounted rewards in the long run. In practice, we usually evaluate a state-action pair under policy \( \pi \) by a Q-value function, which is defined as \( Q^\pi(s,a) = \mathbb{E}_{s_{t+1}, \ldots}^{\pi} \sum_{k=0}^\infty \gamma^k r_{t+k} | s_t = s, a_t = a \). For \( \pi^* \), the optimal Q-value function is denoted as \( Q^*(s,a) \), which follows the Bellman equation:

\[
Q^*(s,a) = r + \gamma Q^*(s',\pi^*(s')).
\]  

There are many advanced RL algorithms aiming to find \( \pi^* \) and \( Q^* \) which satisfy (1), such as deep Q-learning (DQN) [1], deep deterministic policy gradient (DDPG) [3], and soft actor-critic (SAC) [41], etc.

B. Multi-Agent Reinforcement Learning

MARL can be built on a Markov game (MG) [42], which is an extension of MDP from single agent to multiple agents. Similarly, we describe an MG with \( N \) agents by a tuple \( M_N = \langle I, S, O, \mathcal{A}, R, P, \gamma \rangle \), where \( I = \{1, \ldots, N\} \) is the set of all agents; \( S \) is the global state space for the whole system; \( O = \{O^1, \ldots, O^N\} \) is the joint action space, and \( \mathcal{A} \) is the action space of agent \( i \); \( R = \{R^1, \ldots, R^N\} \) is the set of reward functions, and \( R^i \) is the reward function of agent \( i \); \( P : S \times \mathcal{A} \times S \mapsto [0,1] \) is the transition probability; \( \gamma \in [0,1] \) is the discount factor. In this paper, we assume all of the agents are homogeneous, in other words, they share the same state space and action space: \( S^1 = \cdots = S^N = S \), and \( \mathcal{A}^i = \cdots = \mathcal{A}^N = \mathcal{A} \).

For agent \( i (i = 1, \ldots, N) \), it takes actions in observation \( o_i \in O \) according to a policy \( \pi^i : O \mapsto \mathcal{A} \). However, the next-state distribution of agent \( i \) is determined by not only \( \pi^i \) but also the policies of other agents. Hence, once the policies of other agents are changed, the learning process of \( i \) will be non-stationary. To stabilize the learning process for each agent, the framework of CTDE is widely used in MARL [28], [43].

We collect the observations and actions for all agents during the centralized training, and define the Q-value function of agent \( i \in I \) as

\[
Q^i_\kappa(o,a) = \mathbb{E}_{o_\kappa} \sum_{k=0}^{\infty} \gamma^k r_{i,k} | o_t = o, a_t = a
\]  

where \( o = (o^1, \ldots, o^N), a = (a^1, \ldots, a^N), \pi = (\pi^1, \ldots, \pi^N) \) is the joint policy of all agents, \( r_{i,k} \) is the immediate reward of agent \( i \) at time step \( t+k \). The Q-value functions defined in (2) can be used to evaluate the current policies. In fact, many MARL algorithms take (2) as the objective functions to optimize the policies of all agents [28], [43], and these Q-value functions can be calculated by TD-learning [40].

Remark 1: The expectations in (2) are dependent of \( i \) when \( r_{i,k} \neq r_j \) and \( i \neq j \). During centralized training, the agents collect the observations and actions of all agents to approximate the value functions. Hence, the expectations in (2) differ only in the \( r_i \), \( i = 1, \ldots, N \).

C. Successor Features for Transfer Reinforcement Learning

In order to learn transferable features across different tasks for an RL agent, we first decompose the reward function as a linear combination with the following assumption [23], [26].

Assumption 1: Denote \( \phi(s,a,s') \in \mathbb{R}^d \) as features of transition \( <s,a,s'> \), and \( w \in \mathbb{R}^d \) as a weight vector. The reward function \( R : S \times A \times S \mapsto \mathbb{R} \) can be computed as

\[
R(s,a,s') = \phi(s,a,s')^T w.
\]  

In this paper, we use the superscript \( i \) to represent all the corresponding variables and values of agent \( i \).
In (3), the feature function \( \phi(s,a,s') \) is shared for different tasks. Hence, it is assumed that the dynamics of the environment remains unchanged, and the tasks differ only in the weights \( w \). If the tasks have different transition functions, it will be impractical to find a shared \( \phi \) that satisfies (3). We use \( \phi_t := \phi(st,at,st+1) \) to simplify the notations. It is found that \( \phi \) plays a role of representing the dynamics of the environment, and the different tasks are represented by the weight vector \( w \). When Assumption 1 holds, we can then rewrite the definition of Q-value function as

\[
Q_w^\pi(s,a) = E^R \left[ \sum_{k=0}^{\infty} \gamma^k \phi_{t+k}^T w | s_t = s, a_t = a \right] = E^R \left[ \sum_{k=0}^{\infty} \gamma^k \phi_{t+k} | s_t = s, a_t = a \right]^T w = \phi^T(s,a) w.
\]  

(4)

In (4), \( \psi^\pi(s,a) \) is called the SFs of \( (s,a) \) under policy \( \pi \). In addition, the SFs can also be represented as Bellman equations [23]

\[
\psi^\pi(s,a) = \phi_{t+1} + \gamma E^R [\psi^\pi(s',a') | s_t = s, a_t = a].
\]  

(5)

Hence, one can use the TD-learning based algorithms to learn the SFs. As to the weight \( w \), it can be learned by supervised learning: \( r_t \approx \phi^T w \) [23], [24], [26].

According to the definition of \( \psi^\pi(s,a) \), we can quickly evaluate the policy \( \pi \) on a target task \( w' \) by calculating \( Q_w^\pi(s,a) = \psi^\pi(s,a)^T w' \). Once we have trained the models on \( M \) different tasks: \( w_1, \ldots, w_M \), we get \( M \) corresponding optimal policies: \( \pi_1^*, \ldots, \pi_M^* \), and SFs: \( \psi_1^*, \psi_2^* \). With above knowledge that was learned before, a new policy \( \pi' \) for the target task \( w' \) can be defined as

\[
\pi'(s) = \arg \max_{a} \max_{j} \psi_{t+j}^\pi(s,a)^T w'.
\]  

(6)

Thus, we get a transferrable policy across the whole task space. (6) is called the SFs&GPI policy, which combines the SFs and GPI to achieve a policy with good jump-start of performance for the target task [23], [24].

IV. Multi-Agent Reinforcement Learning and Task Transfer With Successor Features

In this section, we first extend the idea of successor features from the discrete action space and single-agent RL to continuous action spaces and MARL, respectively. Then, we propose an SFs-based learning algorithm for MARL. Finally, we extend the algorithm to make the agents learn target tasks by transferring the knowledge of source domain.

A. Multi-Agent Reinforcement Learning via Successor Features

The goal of MARL is to find distributed and independent policies for agents, that can maximize the cumulative team rewards in the long run. According to the definition of Q-value function for agent \( i \) in (2), the \( \epsilon \)-greedy policy like \( \pi^\epsilon(a|o) = \arg \max_{a} Q^\epsilon(o,a) \) is unpractical. That happens because 1) the arg max operator can not be applied to continuous action spaces directly; 2) the observations of other agents are unavailable, and 3) the next actions of other agents can not be observed when they take actions concurrently in the environment. Hence, we use the framework of actor-critic [44] to implement MARL. The actor-critic can separate the policies from Q-values during the execution. In this paper, the deep neural networks [45] are applied to approximate the policies and Q-value functions, which play the roles of actor networks and critic networks, respectively. For agent \( i \), denote \( \pi'(o'|\theta'_i) \) as the parameterized policy, where \( \theta'_i \) are parameters of the actor network. Then, the objective function of policy \( \pi' \) could be

\[
\mathcal{J}'(\theta'_i) = E \left[ \sum_{t=t_0}^{\infty} \gamma^t r_t | o_{t_0}, \pi \right].
\]  

(7)

According to the MADDPG algorithm proposed in [28], the policy gradient of agent \( i \) can be calculated by

\[
\nabla_{\theta'_i} \mathcal{J}'(\theta'_i) = E \left[ \nabla_{\theta'_i} \pi'(o'|\theta'_i) \nabla_{\theta'_i} Q^\pi(o,a) | o_{t_0}, \pi \right].
\]  

(8)

In (8), \( Q^\pi(o,a) \) is approximated by the critic network.

Remark 2: For the MAS with fully cooperative tasks (i.e., \( r_1^t = \cdots = r_N^t \)) [46], the expectations in (7) and (8) are independent of \( i \). If the agents are homogeneous, the policies of different agents might be same when the algorithm converges [46]. However, we cannot use the copied policy for each agent during the training stage, because that will limit the exploration space for the agents to accomplish the tasks cooperatively. Therefore, we initialize the policies of different agents randomly and keep the superscript \( i \) to differentiate the expectations.

Differently from MADDPG, we apply the idea of SFs to the calculations of Q-values in this paper. Because the transition probability of the next-observation for agent \( i \) will be influenced by the observations and actions of other agents, we cannot decompose its reward function with local observations and actions. Hence, we provide the following assumption to extend the SFs to single agent scenarios to multi-agent scenarios:

Assumption 2: For an MAS with \( N \) agents, denote \( \langle o,a,o' \rangle \) as a joint transition from observations \( o = (o_1^1, \ldots, o_N^1) \) to \( o' = (o_1^{t+1}, \ldots, o_N^{t+1}) \) after executing joint actions \( a = (a_1^t, \ldots, a_N^t) \). Denote \( r_{t+1} = R(o,a,o') \) as the reward function for agent \( i \). Then, \( R^i \) can be calculated by the linear combination

\[
R^i(o,a,o') = \phi(o,a,o')^T w^i,
\]  

(9)

where \( \phi(o,a,o') \in \mathbb{R}^d \) is the features of transition \( \langle o,a,o' \rangle \), \( w^i \in \mathbb{R}^d \) is the weight vector for agent \( i \).

In Assumption 2, the inputs of the feature function \( \phi \) include the local observations and actions of all agents. It should be noted that the dynamics of the environment is also assumed to be unchanged. That means the \( \phi \) is shared across different reward functions, so the tasks differ only in the weights \( w \). That is why the \( w \) is also called “task weights”. We denote \( \phi_i \) as the immediate features at time step \( t \) to simplify the notation. The dimensions of both \( \phi \) and \( w^i \) in (9) are \( d \). It should be noted that the \( d \) is independent of the number of agents \( N \). Since the reward weight \( w^i \) represents a specific task in \( \mathbb{R}^d \), \( d \) depends on the number of basic tasks in the task space \( \mathbb{R}^d \).

During the agents-environment interaction, we store the reward data to the replay buffer. Then, we approximate the
reward function in (9) by minimizing

\[ L_i^o = \frac{1}{|D|} \sum_{(o,a) \in D} (r_i - \phi_i^Tw)^2 \]  

where \( D \) is the data set of experience replay buffer, and \( |D| \) is the cardinality of \( D \). One can see that once the feature function \( \phi(o,a,o') \) is given, we can then learn \( w' \) in (10) and approximate the corresponding reward function by supervised learning.

Since the reward function for agent \( i \) is decomposed as the linear combination in (9), the definition of Q-value function can be decomposed as

\[
Q^{\pi_i}(o,a) = \mathbb{E}^T \left[ \sum_{k=0}^{\infty} \gamma^k \phi_{i+k}^Tw_i | o_i = a_i = a \right] 
\]

By defining \( \psi_i(o,a) = \mathbb{E}^T \left[ \sum_{k=0}^{\infty} \gamma^k \phi_{i+k}^T | o_i = a_i = a \right] w_i \),

\[
Q^{\pi_i}(o,a) = \psi_i(o,a)^Tw_i. \]  

From (12), it is also found that all agents in the environment share common MASFs, and \( w_i \) plays an important role in the differences of Q-value functions. Hence, there is no need to design agent-wise critic networks like MADDPG to approximate the value functions of different agents [28]. Instead, we build one network that consists of a global successor feature network (GSFN) and multiple weight layers \( w_i, i = 1, \ldots, N \). The framework of the proposed approach is shown in Fig. 1. It should be noted that the idea of MASFs can also be applied in other structure of Q-values. For example, if the Q-function is defined on the global state \( s \in S \), then the corresponding MASFs can be represented by \( \psi_i(s,a) \).

According to the definition of MASFSs, \( \psi_i(o,a) \) also satisfies the Bellman equation

\[
\psi_i(o,a) = \phi_{i+1} + \gamma \mathbb{E}^T \left[ \psi_i(o', a') | o_i = a_i = a \right]. \]  

Denote \( \theta_\phi \) as the parameters of the GSFN, and \( \psi_i(o,a;\theta_\phi) \) as the parameterized MASFs. Then, we can approximate MASFs by minimizing the following TD errors [40]:

\[
L_\phi = \mathbb{E}_D \left[ (\psi_i(o,a;\theta_\phi) - \psi_i(o,a;\theta_\phi))^2 \right] 
\]

where \( \|\cdot\|_2 \) is the \( L^2 \)-norm in \( \mathbb{R}^d \), and \( \Psi \) is the target MASFs

\[
\Psi = \phi_{i+1} + \gamma \psi_i^\tau(o', \pi_i(o')). \]  

\( \pi_i(o') \) and \( \psi_i(o') \) in (15) represent target policy networks (parameterized by \( \{\pi_i(o')\}_{i=1}^N \) and target GSFN (parameterized by \( \theta_{\phi,\tau} \)), respectively. We initialize the parameters of these target networks as the same as current networks, and apply the “soft updating” mechanism with a factor \( \tau \) to update the target networks [3].

As introduced above, we can update the parameters of the policies by calculating (8) and gradient ascent. The Q-values are approximated by (12), in which \( \psi_i \) and \( w_i \) are updated by minimizing (14) and (10), respectively. We call this MARL method the multi-agent deep deterministic policy gradient with successor features (MADDPG-SFs), whose pseudo code is shown in Algorithm 1. The parameters updating in Algorithm 1 follows the methods in [23], [24], [26], in which the convergence is guaranteed. The proposed MADDPG-SFs is different from MADDPG in two aspects. On the one hand, the value-functions in MADDPG-SFs are calculated by linear combinations of the common MASFs and different weights, rather than by \( N \) agent-wise networks in MADDPG. Besides, the learning of the critics is separated into two independent parts: \( \psi_i(o,a;\theta_\phi) \) and \( w_i \), \( i = 1, \ldots, N \). On the other hand, with linear combination, the objective functions in MADDPG-SFs can be easily changed by the weight \( w_i \in \mathbb{R}^d \). Hence, we can evaluate the current policies on different tasks in \( \mathbb{R}^d \), which will be useful to transfer the knowledge across tasks.

![Fig. 1. Framework of MADDPG-SFs algorithm. The Q-values are computed by linear combinations with MASFs and task-specific weights. The MASFs are approximated by the GSFN.](image-url)

**Algorithm 1 MADDPG-SFs**

**Require:** Episode number \( N_e \), mini-batch size \( N_b \), discount factor \( \gamma \), learning rate \( \alpha_\phi, \alpha_\psi, \alpha_w \), soft-update factor \( \tau \).

1: for episode = 1 to \( N_e \) do
2: Reset environment: \( \{o_i^0\}_{i=1}^N \) = initial observations.
3: for \( t = 0 \) to \( T - 1 \) do
4: \( a_i^t \leftarrow \pi'(o_i^t), i = 1, \ldots, N \).
5: Execute \( a_i^t \), then we get \( \{o_i^{t+1}\}_{i=1}^N \) and \( \{r_i^t\}_{i=1}^N \).
6: Calculate features \( f_{i+1} \leftarrow \phi_{i+1}(o_i, a_i, o_{i+1}) \).
7: Append \( \{o_i^t, o_i^{t+1}, r_i^t, f_{i+1}\}_{i=1}^N, f_{i+1} > \in D \).
8: Sample \( N_b \) transitions randomly in \( D \).
9: \( \theta_\phi \leftarrow \theta_\phi - \alpha_\phi \nabla L_\phi \).
10: \( \theta_{\phi,\tau} \leftarrow \theta_{\phi,\tau} + (1 - \tau)\theta_{\phi,\tau} \).
11: for \( i = 1 \) to \( N \) do
12: \( w_i' \leftarrow w_i - \alpha_w \nabla L_i^o \).
13: \( \theta_{\phi,\tau} \leftarrow \theta_{\phi,\tau} + \alpha_w f_{i+1} \nabla \phi_{i+1} \).
14: \( \theta_{\phi,\tau} \leftarrow \theta_{\phi,\tau} + (1 - \tau)\theta_{\phi,\tau} \).
15: end for
16: \( o_i^t \leftarrow o_i^{t+1}, i = 1, \ldots, N \).
17: end for
18: end for
Differently from the single-agent RL, the CTDE is applied in MADDPG-SFs. During centralized training, the algorithm aims to learn a vector function \( \psi^t(o,a) \) rather than a scalar function \( Q^t(o,a) \). While in decentralized execution, the policy of each agent takes only its local observation as input to explore in the environment. It should be noted that the reward weights \( w \) are not always agent-wise. When a multi-agent team aims to achieve common goals (cooperative tasks), the agents share one weight to learn optimal policies.

### B. Knowledge Transfer for Target Tasks

We now consider the cooperative MAS, and introduce the method of learning target tasks by the using of MASFs. As introduced in [46], the agents share a common reward function under a fully cooperative setting of an MAS. In other words, agents have the same goal and reward weight \( w \).

For a specific feature function \( \phi(o,a,o') \) of an MG, define

\[
\Omega_\phi = \left\{ (o,a,o') = \phi(o,a,o')^T w | w \in \mathbb{R}^d \right\}
\]

as the task space induced by \( \phi \in \mathbb{R}^d \). Denote the set of tasks that were learned in previous as the source tasks \( \Omega^s \), and the set of new tasks as the target tasks \( \Omega^t \). Then, we have \( \Omega^s \subseteq \Omega_\phi \) and \( \Omega^t \subseteq \Omega_\phi \), i.e., the task space induced by \( \phi \) contains both source task space and target task space. Hence, \( \phi(o,a,o') \) should be the shared knowledge between the source tasks and the target tasks. Once the feature function \( \phi(o,a,o') \) is available, we can approximate the weights for a target task by minimizing

\[
\mathcal{L}_w = \frac{1}{|D'|} \sum_{(o',i') \in D'} (r'_i - \phi_i^T w')^2
\]

where \( r'_i \) and \( w' \) are reward at time step \( t \) and the weights for a target task, respectively. \( D' \) is the experience replay buffer for the target task.

Suppose we have learned \( M \) different tasks through MADDPG-SFs separately as the source tasks. Then, we have the knowledge base \( \{ \pi_j^s, w_j, \psi^s_j(o,a) \}_{j=1}^M \), where \( \pi_j^s = (\pi_1^j, \ldots, \pi_N^j) \) is the optimal joint policy for task \( j \). To utilize this knowledge, one can use a source policy \( \pi_j^s \) directly as the initial policy to explore in the target domain. However, selecting the best policy in \( \{ \pi_j^s \}_{j=1}^M \) for traditional MARL algorithms is challenging, because it is hard to evaluate the source policies on the target domain. However, selecting the best one can use a source policy directly as the initial policy to explore in the target domain. Hence, the optimal joint policy for task \( j \) can be calculated by

\[
\pi'_j = \arg \max_{\pi \in \pi_j^s} \mathbb{E}[Q^j_w(o_t,a_t,\pi(o_t))].
\]

The policy in (19) is an extension of SF&GPI [23] from single agent system with discrete action space to the MAS with continuous joint action space. \( \pi' = (\pi'_1, \ldots, \pi'_N) \) is approximated by \( N \) actor networks with mean-square-error losses

\[
\mathcal{L}_w = \mathbb{E} \left[ \| \hat{Q}_w(o_t,a_t,\pi'_j(o_t)) - Q^j_w(o_t,a_t,\pi'_j(o_t)) \|_2^2 \right], \quad i = 1, \ldots, N
\]

where \( \theta'_i \) is the parameter set of actor networks for agent \( i \), and \( \hat{Q}_w^T(o_t) \) represents its approximated policy. To evaluate the new policy in (19), we also learn the MASFs for \( \pi' \) by minimizing the loss

\[
\mathcal{L}_\phi = \mathbb{E}_{(a,a',\phi)} \left[ \| \psi_i^w(o,a;\theta'_i) - \Psi_i^w(o,a) \|_2^2 \right]
\]

where \( \Psi_i^w(o,a) = \phi_i + \gamma \psi_i^w(o',\pi'_i(o')) \). \( \pi'_i(o) \) and \( \psi_i^w(o,a) \) represent the target policy network (parameterized by \( \theta'_i, \phi_i \)) and the target GSFN (parameterized by \( \theta'_i, \phi_i \)), respectively. Then, the Q-value of the policy \( \pi' \) on task \( w' \) can be calculated by

\[
Q^{\pi'_i}(o,a) = \psi^{\pi'_i}(o,a)^T w'.
\]

The algorithm that learns \( w' \), \( \pi'_i \), and \( \psi^{\pi'_i}(o,a) \) is called multi-agent deep deterministic policy gradient with successor features and knowledge transfer (MADDPG-SFKT). The pseudo-code in Algorithm 2 shows the details of MADDPG-SFKT.

**Algorithm 2 MADDPG-SFKT**

**Require:** \( \{ \pi_j^1 \}^M_1, \{ \pi_j^N \}^M_1 \) for \( M \) basic tasks, episode number \( N_e \), mini-batch size \( N_b \), discount factor \( \gamma \), learning rate \( \alpha_\phi, \alpha_w, \alpha_\pi \), soft-update factor \( \tau \).

1: for episode = 1 to \( N_e \), do
2: Reset environment: \( [o_j^0]_{i=1}^{N_e} \), \( w' = \text{initial observations}. \)
3: for \( t = 0 \) to \( T - 1 \), do
4: \( a_t = \{ a_1^0, \ldots, a_N^0 \} \rightarrow \pi_1^t(o_t). \)
5: Execute \( a_t \), then get \( [o_j^1]_{i=1}^{N_e} \) and \( r_{i=1}^{N_e} \).
6: Calculate features \( \phi_{i=1} \rightarrow \Phi_{i=1} \).
7: Append \( [o_j^1, o_j^0, a_{j=1}^1, r_{i=1}^{N_e}, \phi_{i=1} \rightarrow D' \).
8: Sample \( N_b \) transitions randomly in \( D' \).
9: \( w' \rightarrow w' - \alpha_w \nabla L_w. \)
10: \( \phi'_{i=1} \rightarrow \phi'_{i=1} - \alpha_\phi \nabla L_\phi. \)
11: \( \theta'_{i=1} \rightarrow \theta'_{i=1} + (1 - \tau) \theta'_{i=1}. \)
12: for \( i = 1 \) to \( N \), do
13: \( \theta'_i \rightarrow \theta'_i - \alpha_\pi \nabla L. \)
14: end for
15: \( o_j^i \rightarrow o_j^i, \forall i = 1, \ldots, N_e. \)
16: end for

Equation (19) can be seen as the core step of knowledge transfer. Although \( \pi' \) is not the optimal joint policy for task \( w' \), we can use it as the jump-start policy to explore in the target domain at the early training stage. The following analysis will show how \( \pi' \) can outperform \( \{ \pi_j^N \}^M_1 \), as well as the difference between new policy \( \pi' \) and the optimal policy \( \pi^* \) for task \( w' \).
C. Theoretical Analysis of the Jump-Start Policy $\pi'$

Since the source domain and target domain share a common reward feature $\phi$, it can be found that the new policy $\pi'$ in (19) contains more useful information about the target tasks. To explain the superiority of the $\pi'$ compared with the source optimal policies, we follow the GPI in [23] and introduce the following theorem.

**Theorem 1:** Denote $\pi_1^*, \ldots, \pi_M^*$ as optimal joint policies for $M$ different tasks in a MAS, and $\Psi \pi_1^*(o, a), \ldots, \Psi \pi_M^*(o, a)$ as the corresponding MASFs. The approximations of those MASFs are denoted by $\hat{\Psi}_1^*(o, a), \ldots, \hat{\Psi}_M^*(o, a)$, and for all $j = 1, \ldots, M$, they satisfy

$$\|\hat{\Psi}_j^*(o, a) - \hat{\Psi}_j^*(o, a)\|_2 \leq \phi_\psi, \forall o, a \in \mathcal{O} \times \mathcal{A}$$

(23)

where $\|\cdot\|_2$ is the $L^2$-norm. Let $w'$ be the weights of a target task, and $|w'|_2 = C_{w'}$. Then, according to the exploring policy defined in (19), we have

$$\hat{Q}_{w'}^\pi(o, a) \geq \max_j \hat{Q}_{w'}^\pi(o, a) - \frac{2\epsilon_\phi C_{w'}}{1 - \gamma}, \forall o \in \mathcal{O}, \ a \in \mathcal{A}.$$  

(24)

**Proof:** From (23), we have

$$\max_j \hat{\Psi}_j^*(o, a)^T w' - \max_j \hat{\Psi}_j^*(o, a)^T w'$$

$$\leq \max_j \|\hat{\Psi}_j^*(o, a)^T w' - \hat{\Psi}_j^*(o, a)^T w'\|_2$$

$$\leq \max_j \|\hat{\Psi}_j^*(o, a) - \hat{\Psi}_j^*(o, a)\|_2 \times |w'|_2$$

$$\leq \epsilon_\phi C_{w'}$$

(25)

for all $o, a \in \mathcal{O} \times \mathcal{A}$ and $j = 1, \ldots, M$.

Denote $\hat{Q}^\pi_{w'}(o, a) = \max_j \hat{Q}_{w'}^\pi(o, a)$, $\hat{Q}^\pi_{w'}(o, a) = \max_j \hat{Q}_{w'}^\pi(o, a)$, then according to (18) and (25), we have

$$\hat{Q}_{w'}^\pi(o, a) - \hat{Q}_{w'}^\pi(o, a) \leq \epsilon_\phi C_{w'}.$$  

(26)

Define $\mathcal{T}_\pi$ as a Q-learning operator for the MAS, such that

$$\mathcal{T}_\pi(\hat{Q}^\pi_{w'}(o, a)) = r(o, a, o') + \gamma \hat{Q}^\pi_{w'}(o', \pi(o')).$$

Then, $\forall o, a \in \mathcal{O} \times \mathcal{A}$, $j = 1, \ldots, M$, we have

$$\mathcal{T}_\pi(\hat{Q}^\pi_{w'}(o, a)) \geq r(o, a, o') + \gamma \hat{Q}^\pi_{w'}(o', \pi(o'))$$

$$\geq r(o, a, o') + \gamma \hat{Q}^\pi_{w'}(o', \pi(o')) - \epsilon_\phi C_{w'}$$

$$\geq r(o, a, o') + \gamma \hat{Q}^\pi_{w'}(o', \pi(o')) - \epsilon_\phi C_{w'}$$

$$= \mathcal{T}_\pi(\hat{Q}^\pi_{w'}(o, a)) - \epsilon_\phi C_{w'}$$

$$= \hat{Q}^\pi_{w'}(o, a) - \epsilon_\phi C_{w'}.$$  

(28)

Since (28) holds for all $j = 1, \ldots, M$, then

$$\mathcal{T}^\pi_{\pi'}(\hat{Q}^\pi_{w'}(o, a)) \geq \max_j \hat{Q}^\pi_{w'}(o, a) - \gamma \epsilon_\phi C_{w'}$$

$$= \hat{Q}^\pi_{w'}(o, a) - \epsilon_\phi C_{w'}$$

$$\geq \hat{Q}^\pi_{w'}(o, a) - \epsilon_\phi C_{w'} - \gamma \epsilon_\phi C_{w'}.$$  

(29)

According to the definition of $\mathcal{T}^\pi_{\pi'}$ in (27), we have

$$\mathcal{T}^\pi_{\pi'}(\hat{Q}^\pi_{w'}(o, a)) = r(o, a, o') + \gamma \mathcal{T}^\pi_{\pi'}(\hat{Q}^\pi_{w'}(o, \pi'(o')))$$

$$\geq r(o, a, o') + \gamma [\hat{Q}^\pi_{w'}(o, \pi'(o')) - (1 + \gamma) \epsilon_\phi C_{w'}]$$

$$= \mathcal{T}^\pi_{\pi'}(\hat{Q}^\pi_{w'}(o, a)) - \gamma (1 + \gamma) \epsilon_\phi C_{w'}$$

$$\geq \hat{Q}^\pi_{w'}(o, a) - (1 + \gamma) \epsilon_\phi C_{w'} - \gamma (1 + \gamma) \epsilon_\phi C_{w'}.$$  

(30)

Repeating the step in (30), then

$$\mathcal{T}^\pi_{\pi'}(\hat{Q}^\pi_{w'}(o, a)) \geq \hat{Q}^\pi_{w'}(o, a) - \left(1 + \gamma\right) \epsilon_\phi C_{w'}(1 + \gamma + \cdots + \gamma^{n-1})$$

$$= \hat{Q}^\pi_{w'}(o, a) - \frac{1 - \gamma^n}{1 - \gamma} \epsilon_\phi C_{w'}.$$  

(31)

Hence, we have

$$\hat{Q}^\pi_{w'}(o, a) = \lim_{n \to \infty} \left(\mathcal{T}^\pi_{\pi'}(\hat{Q}^\pi_{w'}(o, a))\right)$$

$$\geq \hat{Q}^\pi_{w'}(o, a) - \frac{1 + \gamma}{1 - \gamma} \epsilon_\phi C_{w'}$$

$$\geq \hat{Q}^\pi_{w'}(o, a) - \epsilon_\phi C_{w'} - \frac{1 + \gamma}{1 - \gamma} \epsilon_\phi C_{w'}.$$  

(32)

Finally, $\hat{Q}^\pi_{w'}(o, a) \geq \max_j \hat{Q}^\pi_{w'}(o, a) - 2\epsilon_\phi C_{w'}/(1 - \gamma).$  

In Theorem 1, it is found that the Q-value of the new joint policy induced in (19) can be bounded by $\max_j \hat{Q}^\pi_{w'}(o, a)$ when $\epsilon_\phi \to 0$. Hence, $\pi'$ could be a good jump-start policy for the target task $w'$ compared with the source policies $\pi'_j$, $\forall j = 1, \ldots, M$.

At the next stage, the optimal joint policy $\pi$ and the optimal MASFs $\Psi \pi'$ could be calculated by fine-tuning based on $\pi'$ and $\Psi \pi'$, respectively. The fine-tuning follows the same steps in Algorithm 1. Fig. 2 shows the learning of $\pi'$ and $\Psi \pi'$. Since $\pi'$ is induced from $M$ source tasks, the difference between $\hat{Q}^\pi_{w'}$ and $\hat{Q}^\pi_{w'}$ should be related to the distances from the source tasks $\{w_j\}_{j=1}^M$ to the target task $w'$. The following theorem
shows the bound.

**Theorem 2:** Denote \( w_1, \ldots, w_M \) as weight vectors for \( M \) different tasks, and \( \pi^o(o,a), \ldots, \pi^{o_k}(o,a) \) as the corresponding MASFs, where \( \pi^o_1, \ldots, \pi^o_M \) are optimal joint policies for these tasks. The approximations of the MASFs are denoted as \( \hat{\pi}^o_1(o,a), \ldots, \hat{\pi}^{o_k}(o,a) \), and for all \( j = 1, \ldots, M \), they satisfy

\[
\| \hat{\pi}^o_j(o,a) - \hat{\pi}^{o_j}(o,a) \|_2 \leq \epsilon_{\phi}, \quad \forall o, a \in O \times A.
\]  

(33)

\( w' \) is the weight vector of the target task that satisfies \( \| w' \|_2 = C_{w'} \), and \( \pi' \) is the optimal joint policy of task \( w' \). Let \( \Phi_{\max} = \max_{o,a,o'} \| (\phi(o,a,o')^T w - w') \|_\infty \), and \( \delta_{w'} = \min_j \| w_j - w' \|_2 \). Then, according to the result proved in (19), we have

\[
Q^o_{w'}(o,a) - Q^o_{w'}(o,a) \leq \frac{2}{1-\gamma} (\Phi_{\max} \delta_{w'} + \epsilon_{\phi} C_{w'})
\]  

(34)

where \( Q^o_{w'} \) and \( Q^o_{w'} \) are Q-value functions of policy \( \pi' \) and \( \pi' \) on task \( w' \), respectively.

**Proof:** To simplify the notation, we use \( Q^o_{w} \) to denote \( Q^o_{w}(o,a) \). Then the differences of \( Q^o_{w'}(o,a) \) and \( Q^o_{w'}(o,a) \) can be written as

\[
Q^o_{w'} - Q^o_{w'} = Q^o_{w'} - Q^o_{w'} + Q^o_{w'} - Q^o_{w'} + Q^o_{w'} - Q^o_{w'}
\]

\[
\leq \| Q^o_{w'} - Q^o_{w'} \|_\infty + \| Q^o_{w'} - Q^o_{w'} \|_\infty + \| Q^o_{w'} - Q^o_{w'} \|_\infty
\]

(35)

where \( \| \|_\infty \) is the \( L^\infty \)-norm, i.e.,

\[
\| Q^o \|_\infty \leq \inf \{ C \geq 0 : Q^o(o,a) \leq C, \quad \forall o, a \}.
\]  

(36)

By applying the Bellman equality of the Q-value functions and the assumption in (9), we have

\[
\| Q^o_{w'} - Q^o_{w'} \|_\infty = \| \phi(o,a,o')^T w' + \gamma Q^o_{w'}(o', \pi_{w'}(o')) - \phi(o,a,o')^T w_j + \gamma Q^o_{w'}(o', \pi_{w'}(o')) \|_\infty
\]

\[
\leq \| \phi(o,a,o')^T (w' - w_j) \|_\infty + \| Q^o_{w'} - Q^o_{w'} \|_\infty
\]

(37)

Then, the following holds:

\[
\| Q^o_{w'} - Q^o_{w'} \|_\infty \leq \frac{1}{1-\gamma} \| \phi(o,a,o')^T (w' - w_j) \|_\infty
\]  

(38)

and similarly, we can also get

\[
\| Q^o_{w'} - Q^o_{w'} \|_\infty \leq \frac{1}{1-\gamma} \| \phi(o,a,o')^T (w' - w_j) \|_\infty.
\]  

(39)

In addition, according to Theorem 1, we have

\[
Q^o_{w'}(o,a) \geq \max_j Q^o_{w'}(o,a) - \frac{2\epsilon_{\phi} C_{w'}}{1-\gamma}
\]

\[
\geq Q^o_{w'}(o,a) - \frac{2\epsilon_{\phi} C_{w'}}{1-\gamma}
\]

(40)

for all \( o \in O, a \in A \), and \( j = 1, \ldots, M \). So we get

\[
Q^o_{w'} - Q^o_{w'} \leq \frac{2\epsilon_{\phi} C_{w'}}{1-\gamma}
\]

(41)

By combining (35), (38), (39), and (41), we have

\[
Q^o_{w'} - Q^o_{w'} \leq \frac{2}{1-\gamma} (\Phi_{\max} \delta_{w'} + \epsilon_{\phi} C_{w'}),
\]

(42)

for all \( o \in O, a \in A \), and \( j = 1, \ldots, M \). Since (42) holds for all \( j = 1, \ldots, M \), then we have

\[
Q^o_{w'} - Q^o_{w'} \leq \frac{2}{1-\gamma} (\Phi_{\max} \delta_{w'} + \epsilon_{\phi} C_{w'}),
\]

(43)

where \( \Phi_{\max} = \max_{o,a,o'} \| (\phi(o,a,o')^T w - w') \|_\infty \), \( \delta_{w'} = \min_j \| w_j - w' \|_2 \).

As is shown in Theorem 2, the difference between \( Q^o_{w'} \) and \( Q^o_{w'} \) can be bounded by \( \Phi_{\max} \delta_{w'} \) when \( \epsilon_{\phi} \rightarrow 0 \). \( \Phi_{\max} \) is the supremum of \( \| (\phi(o,a,o')^T w - w') \|_\infty \), which is determined by the design of the reward features \( \phi \). \( \delta_{w'} \) can be seen as the minimum distance from source tasks to the target task. That is why the more similar between the source tasks and the target task, the easier to learn in the target domain.

It seems that when \( C_{w'} = \| w' \|_2 \) in Theorems 1 and 2 is very large, the error bounds in (24) and (34) could also be so large that \( \pi' \) cannot provide good performance. However, according to the linear combinations in (18), the \( \| Q^o_{w'} \|_2 \) could also be very large caused by \( \| w' \|_2 \). Once the target task is specific, \( C_{w'} \) should be a constant value. In that scenario, if we have good approximations of \( \| \phi(o,a,o')^T w - w' \|_\infty \), then the error bounds can also be small enough compared with the Q-values. Besides, if \( \| w' \|_2 \) is very large compared with \( \| |w| \|_2 \), \( j = 1, \ldots, M \), it is indicated that there are big differences between the source domain and target domain. Hence, it is acceptable that the error bounds increase when we calculate the \( \pi' \) through knowledge transfer.

In conclusion, the Theorems 1 and 2 are proposed to explain the superiority of the new joint policy \( \pi' \) executed in the target task \( w' \). It is noted that \( \pi' \) is not the optimal joint policy for task \( w' \), however, we can use it as the jump-start policy to improve the exploration for target task. In the next section, we will show the simulation results to verify the performance of the proposed algorithms.

V. SIMULATION

In this section, to verify the theoretical analysis in section IV, we evaluate the performance of proposed algorithms on two challenging multi-agent simulation environments. All simulations are run at a desktop with Intel Core i7-7700k CPU@4.20GHz under Ubuntu 18.04 operation system.

A. Environments

1) Cooperative Box-Pushing: In a cooperative box-pushing environment, two agents aim to push a box to the target position cooperatively. As shown in Fig. 3(a), the target position is on the right part of the environment, while the box is started in the left area. Between the target and the start point of the box, there are two additional objects (obstacles) that should be avoided by the box. The box can be pushed to move if and only if two agents force it concurrently, which makes it more challenging for the box to explore the target position in

\[ \frac{2}{1-\gamma} \Phi_{\max} \delta_{w'} + \epsilon_{\phi} C_{w'} \]

2 The code is available at: https://github.com/wenzhangliu/maddpg-sfkt.git
the environment\(^3\). The observation spaces for agents are continuous, which include the information about the positions of all entities, as well as the velocities of the agents and the box. Agents take continuous actions varying from \(-1\) to \(1\) on both horizontal and vertical directions. To guide the agents to move toward the box, one agent will get a \(+1\) reward if it catches the box. The agents will get a \(+5\) reward if the box covers the target object on the right area, but they will be penalized with \(r_p\) if the box meets an obstacle. In the next part, we will show how different \(r_p \in [0, -0.5, -2.5, -5.0]\) influence the learning performance of the compared algorithms. Agents will also be penalized by \(-1\) if they collide with each other or the wall, or if the box is pushed to the wall. As one can see, the reward of \(+5\) is sparse for the two agents, which makes it more difficult to explore and learn the task.

![Fig. 3. The simulation environments: (a) Cooperative box-pushing; (b) Non-monotonic predator-prey.](image)

2) Non-Monotonic Predator-Prey: The non-monotonic predator-prey environment is extended from grid-world with discrete state spaces [14] to continuous state spaces, and from monotonic rewards [28, 47] to non-monotonic rewards. As shown in Fig. 3(b), there are three agents playing the role of predators, and one agent playing the role of prey. The maximum speed of prey is larger than those of predators. Hence, the predators have to capture the prey through cooperation. Both of the observation space and action space for each agent are continuous. Each agent can observe the positions and velocities of itself and the positions of the other agents. The actions for each agent vary from \(-1\) to \(1\) on two directions, which are the same as those of cooperative box-pushing environment. In traditional predator-prey tasks, the predators will be rewarded if any of them captures the prey [28, 47]. Differently from that, if only one or two predators capture the prey, they will be penalized. The predators will be rewarded if and only if they capture the prey concurrently, which makes it more difficult to explore in the environment and accomplish the task\(^3\). That is why the simulation environment is called non-monotonic. Here, the predators share a common reward \(r_t \in [0, r_{p1}, r_{p2}, 10]\), where \(r_{p1}\) is the penalty when only one predator captures the prey, and \(r_{p2}\) is the penalty when only two predators capture the prey concurrently. Predators will be rewarded with \(+10\) if all of them capture the prey together. In addition, there are two kinds of policies for the prey. The first one is random policy, which outputs actions by sampling in the Gaussian distribution. The other policy for prey can be trained with the DDPG algorithm [3] in previous, where three predators were controlled by proportional-integral-derivative (PID) controller separately.

B. The Design of the Features \(\phi\)

As is introduced in Section IV, the key to formulate the source tasks in prior to the target tasks, is the design of the feature function \(\phi(o,a,o')\). \(\phi(o,a,o')\) should be the shared knowledge between the new tasks and the previous learned tasks. To train agents in the above two environments via MASFs, we need first to build the feature functions \(\phi(o,a,o')\) for them. According to [23] and [24], it is found there are three methods to build the reward features. The first is shaping the reward by hand in the task space with multiple Gaussian functions [23, 48], that may cause intrinsic shaping errors. The second method achieves the \(\phi\) through multi-task learning [23]. However, once there comes new task, the learned \(\phi\) in source domain might be outdated for the unseen task. The third method uses the reward functions of basic tasks as the elements of \(\phi(o,a,o')\) directly [24], such as \(\phi(o,a,o') = [r_1(o,a,o'), \ldots, r_M(o,a,o')]^T\), where \(r_j\) represents the reward function of \(j\)-th task. If we have defined the basic tasks in the task space and approximated the corresponding reward functions, we can approximate the reward function of the target task through linear combination of \(\phi(o,a,o')\) and \(w\). In this paper, we choose the third method to design the \(\phi\) for the two simulation environments.

In cooperative box-pushing environment, we define the three objects (two obstacles and one target) in Fig. 3(a) as three basic targets for the box. Define the task of capturing the box without colliding to the wall or the other agent as another basic task. Hence, we have \(\phi = [R_1, R_2, R_3, R_4]^T\). \(R_1 = 5, R_2 = 5, R_3 = 5, R_4 = 0\) represent the agents pushing the box to obstacle 1, obstacle 2, and the target place, respectively. Otherwise, they are zeros. For \(R_4\), if the agents collide with the wall or each other, \(R_4 = -1\); if the agents catch the box, \(R_4 = 1\); otherwise, \(R_4 = 0\). Then, we can use \(w \in \mathbb{R}^4\) to represent different tasks. For example, when \(w = [-1, -1, 1, 1]^T\), i.e., \(r_p = -5\). We have \(R = \phi^T w = -R_1 - R_2 + R_3 + R_4\), which means the agents should push the box to the target place and avoid the two obstacles and collisions. If the box collides with one of the obstacles, the reward is \(-5\).

For the predator-prey environment in Fig. 3(b), denote \(N_{\text{catch}}\) as the number of predators that capture the prey concurrently, then \(N_{\text{catch}} \in \{0, 1, 2, 3\}\). We can define \(\phi = [R_{1n}, R_{2n}, R_{3n}]^T\), where

\[
R_n = \begin{cases} 10, & \text{if } n \text{ agents capture the prey} \\ 0, & \text{otherwise} \end{cases} \quad (44)
\]

for all \(n = 1, 2, 3\). In this simulation environment, the penalty of collisions among agents is not considered. Hence, we can use \(w \in \mathbb{R}^3\) to represent different tasks. For example, if \(w = [-0.2, -0.1, 1.0]^T\), then we have

\[
R = \phi^T w = \begin{cases} -2, & 1 \text{ agent captures the prey} \\ -1, & 2 \text{ agents capture the prey} \\ 10, & 3 \text{ agents capture the prey} \\ 0, & \text{otherwise} \end{cases} \quad (45)
\]

In that scenario, \(r_{p1} = -2\) and \(r_{p2} = -1\).

\(^3\) The video of the simulations: https://youtu.be/w0kscgRTGz8
C. Simulation Results and Discussions

In the experiments, we adopt deep neural networks with two hidden layers and three hidden layers to approximate the actor networks and the GSFN, respectively. Each hidden layer is non-linearized by a leaky rectified linear units (ReLU) activation function [49]. The output layer of each actor network is activated by a tanh-function, which guarantees the actions to be continuous and bounded in $[-1, 1]$. The last layer of GSFN includes $d$ nodes without activation function, where $d$ is the dimension of $\phi$. Details of the learning settings for the two environments are listed in Table I. In the following experiments, the MADDPG-SFKT contains the fine-tuning steps as shown in Fig. 2. Since the action spaces are continuous in the two simulation environments, the proposed algorithms are compared with MADDPG [28], ATT-MADDPG [31], MAOPT-SRO [35], and UneVEn [39]. We replicate the MAOPT-SRO based on MADDPG framework with the same experiment settings in [35] for fairness. To implement the value-based UneVEn in continuous action spaces, we discretize the two dimensional continuous actions as $\{[0, 0], [0, -1], [0, 1], [-1, 0], [1, 0]\}$.

1) Cooperative Box-Pushing: We first define and learn four source tasks via MADDPG-SFs: $w_1 = [0, 0, 0, 1]^T$, $w_2 = [1, 0, 0, 1]^T$, $w_3 = [0, 1, 0, 1]^T$, and $w_4 = [0, 0, 1, 1]^T$. Then, we learn the target tasks $w' = [r_p, r_p, 1, 1]$ with different penalties $r_p \in \{-0.5, -2.5, 5.0\}$, as well as the corresponding jump-start joint policies and MASFs through MADDPG-SFKT. For the other algorithms, they learn the target task $R = \phi^T w'$ from scratch. We evaluate the average return of the algorithms over five trials at every 20 episodes. The performance of learning process for each task is shown in Fig. 4, which is averaged over 10 random seeds. We select the trained model that performs best for each algorithm, and compare their average returns in Fig. 5. Furthermore, to demonstrate the efficient exploration of our method, Fig. 6 shows the states visited by the box when $r_p = -5.0$.

When $r_p = 0$, i.e., $w' = w_2 = [0, 0, 1, 1]^T$, we compare the MADDPG-SFs with the other algorithms in Fig. 4(a). It is found that the MADDPG-SFs fails to outperform the other algorithms, because it learns the $w'$ from scratch. However, it can converge to the near optimal solutions, which can be used to collect the knowledge in source domain. If we add the penalty (e.g., $r_p = -0.5$), there will be significant performance decline of the baseline algorithms (as shown in Fig. 4(b)). MADDPG-SFKT can achieve the optimal solutions after 1000 episodes, while the other MADDPG-based algorithms cannot converge until 4000 episodes. In Figs. 4(c) and 4(d), we can find that the compared baselines converge to suboptimal solutions when the $|r_p|$ increases. As for UneVEn, it performs worst compared with other methods in this environment. The UneVEn learns a set of related tasks simultaneously to improve the exploration, however, the reward for target task is too sparse for the agents to achieve. Although the MADDPG-SFKT performs worse at the early training stage, it can achieve more accumulative return in the long run. As shown in Fig. 5, when $r_p \leq -0.5$, the best trained MADDPG-SFKT models always outperform the other methods. When $r_p = 0$, $-0.5$, or $-2.5$, the ATT-MADDPG can sometimes improve the cooperation of agents with the attention mechanism, while its performance is unstable and limited.

![Fig. 4. The cooperative box-pushing simulation results with different obstacle penalties. (a) $r_p = 0$; (b) $r_p = -0.5$; (c) $r_p = -2.5$; (d) $r_p = -5.0$.](image)

![Fig. 5. The compared average return of the models for box-pushing that perform best in ten random seeds.](image)
environment. If we use $\pi'$ calculated by (19) as the jump-start policy, the agents can push the box to the target position at early training stage. For example, when the training episode is 1000, 2000, or 3000 in Fig. 6, the agents with MADDPG-SFKT algorithm can push the box to visit the target position even if they would get immediate penalty $r_p = -5.0$. To get more positive rewards at the target position in the long run, our algorithm keeps exploring the optimal path to push the box from start position to the target. That is why the performance of MADDPG-SFKT is worse than the other methods at the fist 1000 and 2000 episodes in Figs. 4(c) and 4(d), respectively. By taking full use of the knowledge in source domain, the MADDPG-SFKT shows more efficient exploration compared with other methods.

2) Non-Monotonic Predator-Prey: In predator-prey simulation, the source tasks are defined as $w_1 = [1, 0, 0]^T$, $w_2 = [0, 1, 0]^T$, $w_3 = [0, 0.5, 1]^T$, and $w_4 = [0, 0, 1]^T$. After learning the source tasks with MADDPG-SFs, we use MADDPG-SFKT to learn target tasks $w' = [r_{p_1}, r_{p_2}, 1]$ with different $r_{p_1}$ and $r_{p_2}$. The other algorithms learn the target tasks directly without knowledge transfer. We evaluate the algorithms over five trials at every 50 episodes. The experiment results are averaged over 5 random seeds. At first, the prey takes actions by sampling in a 2-d Gaussian distribution with covariance as $[[1.5, 0]^T, [0, 1.5]^T]$. The simulation results are shown in Fig. 7. Then, we control the prey by DDPG policy, which was trained in previous. The simulation results are shown in Fig. 8. We also select the model for each algorithm that performs best in five random seeds, and compare their average returns in Fig. 9.

At first, we evaluate the performance of MADDPG-SFs and the compared baselines in Figs. 7(a) and 8(a), where $r_{p_1} = r_{p_2} = 0$ (i.e., $w = w_4$). Three predators can not get positive rewards unless all of them capture the prey concurrently. It can be found that the UneVEn performs best compared with other methods. For our MADDPG-SFs, it can also get well performance without knowledge transfer. When the prey is controlled by the DDPG policy, the MADDPG, ATT-MADDPG, and MAOPT-SRO are hard to get positive rewards while UneVEn and MADDPG-SFs are still well-performed. Hence, we can get good approximations of the basic tasks in the source domain, and collect the useful knowledge for transfer. In Figs. 7(b) and 8(b), $r_{p_1} = 0$ and $r_{p_2} = 1$. That means the predators can get positive rewards when at least two of them capture the prey concurrently, which is an easier task for the algorithms. With the knowledge transferred from source domain, the MADDPG-SFKT performs best in both Figs. 7(b) and 8(b). Although the MAOPT-SRO sometimes learns good policies in this task, it can not transfer the knowledge among different tasks. Hence, its total performance is still limited in this scenario. When $r_{p_1}$ is negative while $r_{p_2} = +1$ in Figs. 7(c) and 8(c), it can be found that UneVEn can also get good performance. When both $r_{p_1}$ and $r_{p_2}$ are negative in Figs. 7 and 8, the MADDPG-SFKT can still get increasing average returns during training, while the other algorithms fail to catch the prey (as shown in Figs. 7(d)–
Fig. 7. The non-monotonic predator-prey simulation results with different groups of penalties, where the prey is controlled by (a) random policy. (b) $r_{p1} = 0$, $r_{p2} = 0$; (c) $r_{p1} = 1$, $r_{p2} = 1$; (d) $r_{p1} = 1$, $r_{p2} = 1$; (e) $r_{p1} = -1$, $r_{p2} = -1$; (f) $r_{p1} = -2$, $r_{p2} = -2$.

Fig. 8. The non-monotonic predator-prey simulation results with different groups of penalties, where the prey is controlled by pre-trained DDPG policy. (a) $r_{p1} = 0$, $r_{p2} = 0$; (b) $r_{p1} = 0$, $r_{p2} = 1$; (c) $r_{p1} = -1$, $r_{p2} = 1$; (d) $r_{p1} = -1$, $r_{p2} = 1$; (e) $r_{p1} = -2$, $r_{p2} = -1$; (f) $r_{p1} = -2$, $r_{p2} = -2$.

Fig. 9. The compared average return of the models for predators that perform best in five random seeds, where the prey is controlled by (a) random policy and (b) DDPG policy. It can also be found that our algorithm gets negative average returns before 2000 episodes and 3000 episodes in Figs. 7(e) and 7(f), respectively. However, it keeps learning and gets increasing positive returns after 3000 episodes. That indicates a strong exploration ability of the MADDPG-SFKT, which allows the agents to explore in the environment without being influenced by the immediate penalties. From the performance of the UneVEn algorithm, the negative average returns also indicate its efficient exploration. However, it fails to find the more positive rewards when $r_{p1}$ and $r_{p2}$ are both negative. From Fig. 9(a), the results of MADDPG-SFKT show better and robust performance for different groups of $r_{p1}$ and $r_{p2}$ than the other algorithms. In conclusion, the experiment results have demonstrated the stable learning ability of MADDPG-SFs algorithm, as well as the advantages of exploring in complex tasks for the MADDPG-SFKT algorithm. With the transferable MASFs, the algorithm calculates the value functions on different tasks by linear combination. The knowledge transfer in (19) works because the task weights for source domain and target domain are in the same space $\mathbb{R}^d$, and they share a common feature function $\phi$. Hence, once we have specified the features $\phi$ as
well as the dimension $d$, $\pi'$ will improve the total performance of the target tasks as a jump-start joint policy. Thanks to the knowledge in source domain, the agents can explore in the environment more efficiently, which helps the agents collect more useful data through interacting with the environment.

It seems unfair for the other methods which learn from scratch without pre-trained models. However, because they cannot build the connections between the source domain and target domain, it is hard to get a pre-trained model which contains useful knowledge for learning different target tasks. In other words, these methods have to pre-train a model from scratch for each target task. Differently from that, once we have pre-trained these source tasks by MADDPG-SFs, we can use such knowledge to learn more different target tasks, without re-training the source tasks for each target task. That is why the proposed method outperforms the other methods on the challenging tasks.

VI. CONCLUSIONS

This paper proposes an SFs-based reinforcement learning method for cooperative MASs, which transfers knowledge from source tasks to the target task. A new multi-agent reinforcement learning framework with MASFs is introduced, which can learn not only joint optimal policies but also the MASFs and task weights. For more complex task in the target domain, traditional algorithms fail to find optimal solutions because of their limited explorations. With the MASFs and the corresponding policies learned in previous, the algorithm MADDPG-SFKT is proposed to take full use of the knowledge that was learned in the source domain, and explore more efficiently in the target environment. The theoretical analysis ensures the superiority of the exploring policy compared with the source policies, and also gives the performance bound. We evaluate the proposed algorithm in cooperative box-moving and non-monotonic predator-prey environments. By increasing the difficulties in different simulation tasks, the experiment results have verified the proposed algorithm empirically and shown greater performance compared with state-of-the-art algorithms.

REFERENCES


Wenzhang Liu received the B.S. degree in engineering from the College of Communication Engineering, Jilin University, in 2016, and the Ph.D. degree in engineering from the School of Automation, Southeast University in 2021. He is currently a Postdoctoral Fellow with the School of Artificial Intelligence, Anhui University. His current research interests include multi-agent reinforcement learning, transfer reinforcement learning, and optimization, etc.

Lu Dong (Member, IEEE) received the B.S. degree in physics and the Ph.D. degree in electrical engineering from Southeast University, 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.

Dan Niu (Member, IEEE) received the Ph.D. degree in electronic science and technology from Graduate School of Information, Production and Systems, Waseda University, Japan in 2013. He has been an Associate Professor with the School of Automation, Southeast University. His research interests include AI for Industrial Control and AI for EDA.

Changyin Sun (Senior Member, IEEE) received the B.S. degree in applied mathematics from the College of Mathematics, Sichuan University in 1996, and the M.S. and the Ph.D. degrees in electrical engineering from Southeast University, in 2001 and 2004, respectively. He is currently a Professor with the School of Automation, Southeast University. His current research interests include intelligent control, flight control, and optimal theory. He is an Associate Editor of IEEE Transactions on Neural Networks and Learning Systems, Neural Processing Letters, and IEEE/CAA Journal of Automatica Sinica.