Hybrid Learning for Multi-agent Cooperation with Sub-optimal Demonstrations

Peixi Peng[1,2*], **Junliang Xing**[1*] and **Lili Cao**[1]
[1] Institute of Automation, Chinese Academy of Sciences
[2] Peking University
{peixi.peng, junliang.xing, lili.cao}@ia.ac.cn

Abstract

This paper aims to learn multi-agent cooperation where each agent performs its actions in a decentralized way. In this case, it is very challenging to learn decentralized policies when the rewards are global and sparse. Recently, learning from demonstrations (LfD) provides a promising way to handle this challenge. However, in many practical tasks, the available demonstrations are often sub-optimal. To learn better policies from these sub-optimal demonstrations, this paper follows a centralized learning and decentralized execution framework and proposes a novel hybrid learning method based on multi-agent actor-critic. At first, the expert trajectory returns generated from demonstration actions are used to pre-train the centralized critic network. Then, multi-agent decisions are made by best response dynamics based on the critic and used to train the decentralized actor networks. Finally, the demonstrations are updated by the actor networks, and the critic and actor networks are learned jointly by running the above two steps alliteratively. We evaluate the proposed approach on a real-time strategy combat game. Experimental results show that the approach outperforms many competing demonstration-based methods.

1 Introduction

Decentralized partially observable Markov decision processes (Dec-POMDPs) [Oliehoek and Amato, 2016] are applicable to many multi-agent cooperative tasks such as multi-agent robotics [Chen *et al.*, 2017], taxi fleet optimization [Nguyen *et al.*, 2018], and combat games [Foerster *et al.*, 2018]. In these applications, a team of agents coordinates their behavior while acting in a decentralized way and aims to achieve a unified goal or the largest team utility. However, since the joint state-action space can be huge, it is very challenging to learn effective decentralized policies, especially in environments where the reward signals are sparse.

A widely used approach to sparse-reward reinforcement learning (RL) problems is learning from demonstrations (LfD) [Schaal, 1996]: task replays performed by experts available for access. One straightforward method is to recover experts' policies from available demonstrations [Pomerleau, 1991; Ross *et al.*, 2011; Sun *et al.*, 2017; Ho and Ermon, 2016; Song *et al.*, 2018]. These works assume that experts perform well, and agents will perform well by mimicking experts' actions. However, due to the huge state action space of multi-agent systems, it is hard to obtain the best policy even for experts. The available demonstrations are often sub-optimal. Another type of LfD method, termed as inverse reinforcement learning [Ng and Russell, 2000], formulates the task as solving an inverse problem and strives to infer the hidden reward function from demonstrations. However, the rewards are global and shared by all agents in cooperative tasks. Multi-agent credit assignment poses a challenge as the joint actions generate a global (or collective) reward that may not be decomposable among agents. One agent's best policy depends on other agents' policies, which makes learning complicated and unstable.

In this work, we consider a problem setting in the centralized learning and decentralized execution framework [Lowe *et al.*, 2017], where learning takes place in a domain simulator, and multiple agents execute their actions, respectively. It proposes a hybrid learning approach from sub-optimal demonstrations. Fig. 1 plots its learning flowchart with three main steps: 1) the centralized critic network is trained using the expert demonstrations which are not optimal; 2) given the centralized critic, the joint decisions of multiple agents at each state are modeled as a combinatorial optimization problem and learned in a decentralized manner; and 3) the actor networks are regarded as new expert policies and used to update demonstrations. The critic and actor networks are learned jointly by performing the above two steps alternatively. As the worst-case complexity of finding the global optimal solution in the second step grows exponentially for the number of agents, this work introduces a best response dynamics algorithm to find the joint local optimal policies alternatively, where each agent' action is a best response to others' actions. Each agent action's advantage function is estimated by calculating its contribution in the joint local opti-

*Equal contribution. The corresponding author is Junliang Xing. This work was supported by the Key-Area Research and Development Program of Guangdong Province under Grant 2019B010153002, and the Natural Science Foundation of China (NSFC) under Grants 61702515 and 61672519.

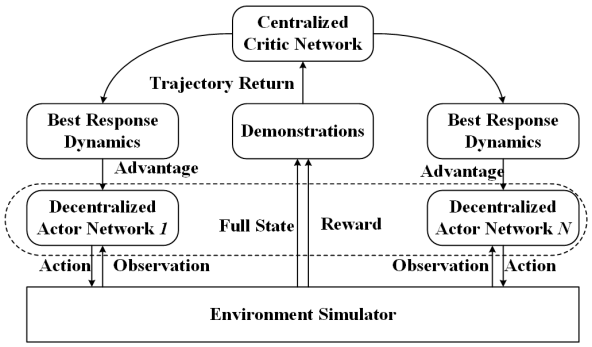

Figure 1: The flowchart of the approach. The training is centralized, while only the actor networks (in the dashed) are executed.

mal policies, and it is proved (see Sec. 4.2) that this advantage function is effective to guide agents' actor networks to learn better policies than demonstration actions.

The proposed approach is evaluated on Real-Time Strategy (RTS) combat game [Churchill *et al.*, 2012]. Experimental results show that it significantly improves over demonstrations and outperforms many demonstration-based methods. For example, other competing methods improve the mean win rate from 0.19 in demonstrations to 0.57, while the proposed approach achieves 0.80 in mean win rate.

2 Related Work

Many real-world applications [Chen *et al.*, 2017; Nguyen *et al.*, 2018] are modeled as cooperative multi-agent learning tasks, which aim to learn decentralized polices from global or collective rewards. The centralized learning and decentralized execution framework [Oliehoek *et al.*, 2008] is an increasing paradigm for these problems and has recently attracted attention in RL community [Foerster *et al.*, 2018; Rashid *et al.*, 2018; Lowe *et al.*, 2017; Nguyen *et al.*, 2018], which assumes that there is access to a domain simulator. The learning takes place in the simulator where extra state information is available, and agents can communicate freely. In contrast, each agent selects its action conditioned only on its observation during execution.

RL entails the knowledge of the reward function or observations of immediate rewards. However, it is challenging to define the rewards of most states in many multi-agent systems because it needs to take the joint states of all agents into account. By contrast, it is often quite natural to express a task goal as a sparse reward function [Vecerik *et al.*, 2017]. A typical example is multi-robot navigation [Chen *et al.*, 2017], where the rewards are only observed directly when collisions occur, or robots reach their destinations. A widely used approach to these problems is learning from demonstrations (LfD) [Schaal, 1996]. The direct method is to recover experts' policies from demonstrations by supervised learning [Pomerleau, 1991; Ross *et al.*, 2011; Sun *et al.*, 2017] or generative adversarial learning [Ho and Ermon, 2016; Song *et al.*, 2018], which make the learned policies close to the expert policies.

Unfortunately, it is hard to collect high-quality demonstrations in many tasks, and the available demonstrations are often sub-optimal. To learn better policies from demonstrations, several methods combine imitation learning and RL together [Silver *et al.*, 2016; Hu *et al.*, 2018]. However, AlphaGo [Silver *et al.*, 2016] is designed for the two-agent zero-sum competitive task, and OGTL [Hu *et al.*, 2018] relies on independent rewards and single-agent action exploration. Both cannot be directly applied to cooperative tasks with collective rewards. Recently, a few approaches [Vecerik *et al.*, 2017; Hester *et al.*, 2018] are proposed to explore the sparse-reward environment by LfD. These methods are all designed for single-agent tasks and try to find better policies by exploring demonstration actions. However, the joint state-action space of multiple agents is much larger, and the exploration may be inefficient. Besides, inverse reinforcement learning (IRL) [Ng and Russell, 2000] is designed to infer the hidden reward function from demonstrations. Most existing multi-agent IRL approaches are designed for two-agent zero-sum games or non-cooperative tasks [Lin *et al.*, 2018]. In a cooperative setting, even if the hidden rewards are inferred, it is still hard to learn the agent's best policy as it relies on the policies of other agents.

3 Problem Statement

A cooperative multi-agent task can be described as a Dec-POMDP [Rashid *et al.*, 2018] $\langle \mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{O}, r, N, \lambda \rangle$ which contains: **State space** $\mathcal{S}$ is a finite set of distinct states which can be visited. **Agents** $\mathcal{G} = \{g_1, g_2, ..., g_N\}$ denotes N agents. **Action space** $\mathcal{A} = \{\mathcal{A}_g\}_{g \in G}$ where $\mathcal{A}_g$ defines the set of candidate actions that can be performed by agent g. We particularly consider the discrete action space problem where $\mathcal{A}_g$ is a finite set. **State transition function** $\mathcal{P}_t = \mathcal{P}(s_{t+1}|s_t, A)$ is induced by performing joint actions $A = \{a_g\}_{g \in \mathcal{G}}$ at state s_t. **Observation function** $\mathcal{O}(s) = \{O^g(s)\}_{g \in \mathcal{G}}$ where $O^g(s)$ is the observation of agent g at state s. The decentralized policy of g is defined as $\{p_g(a|O^g(s))\}_{a \in \mathcal{A}^g}$ corresponding to the probabilities of actions to perform. **Reward** $r(s', s, A)$ is a bounded function which is used to measure the reward (or payoff) of the joint state-action where $s' \sim \mathcal{P}(s'|s, A)$. It is collective and shared by all agents in the cooperative setting. In our task, the rewards are sparse which are zero in most states. $\lambda \in [0, 1)$ is a discount factor. Given any initial state $s_1 \in \mathcal{S}$. At each time step t, each agent g selects an action $a_g \in \mathcal{A}_g$ simultaneously, following joint policies $\pi = \prod_{g \in \mathcal{G}} p_g$. All agents receive a global reward r_t, and the state is transited to s_{t+1} according to $\mathcal{P}$. In an episodic problem, this process continues until the state reaches a terminal state s_T. The state value $V^\pi(s)$ is calculated by the cumulative rewards with λ:

$$V^\pi(s) = \mathbb{E}\left(\sum_{t=1}^{T-1} \lambda^{t-1} r_t\right), \tag{1}$$

where $\mathbb{E}$ is the expectation. If $V^{\pi_1}(s) \geq V^{\pi_2}(s)$ for any $s \in \mathcal{S}$, then π_1 is better than π_2. According to the state value, the state-action value $Q^\pi(s, A)$ is defined as:

$$Q^\pi(s, A) = \mathbb{E}\left(r(s', s, A) + \lambda V^\pi(s')\right), \tag{2}$$

where $s' \sim \mathcal{P}(s'|s, A)$. **Demonstrations** are defined as an observation set $\{s_k, \{a_g(s_k)\}_{g \in \mathcal{G}}\}_{k=1}^K$ which shows how experts have performed in different states. Here s_k is the visited

state and $\{a_g(s_k)\}_{g\in\mathcal{G}}$ are all agents' actions performed by experts. Similar to [Pomerleau, 1991; Ho and Ermon, 2016], a parameterized state-action mapping $A^D(s)=\{a_{\theta_g^D}\}_{g\in\mathcal{G}}$ is used to clone the experts' behaviors, and it is trained by minimizing the loss $\theta_g^D=\text{argmin}_\theta\sum_{k=1}^K L(a_\theta(s_k), a_g(s_k))$, where $L(\cdot)=0$ if $a_\theta(s_k)=a_g(s_k)$ and positive otherwise.

A motivating application for the above Dec-POMDP is playing a real-time strategy (RTS) combat game, where the task is to coordinate multiple allied units to defeat their enemies in real-time scenarios. Since the number of units is large and uncertain, it is necessary to resort to decentralized polices where each learning agent controls an allied unit [Hu *et al.*, 2018; Foerster *et al.*, 2018; Rashid *et al.*, 2018]. The global rewards are easily defined when the combat ends or attacks happen, while it is hard to evaluate the combat situation when the agents are only moving [Churchill *et al.*, 2012]. The human-designed heuristics are often sub-optimal [Churchill *et al.*, 2012]. In the paper, the heuristics are used to generate demonstrations, and our goal is to learn effective decentralized polices to control the allied units for higher win rates.

4 Method

As shown in Fig. 1, we follow the centralized training and decentralized execution framework. Two types of networks are modeled containing a centralized critic network parameterized by θ^V and N decentralized actor networks parameterized by $\{\theta_g^\pi\}_{g\in\mathcal{G}}$ for each agent. The critic is used to model collective rewards to make the learned policies cooperative, and the actor networks are used to model the decentralized policies. In the proposed approach, the inputs of the actor networks and the critic network are different. Therefore, different from the single-agent actor-critic methods [Li, 2018], these two types of networks have independent parameters. The critic network and actor networks are both trained in the simulator, while only the actor networks are used to execute.

4.1 Critic Network Learning

Although the available demonstrations are sub-optimal, it is still reasonable to approximately evaluate the state value by demonstration actions. Given any state $s\in\mathcal{S}$, initialize game with s and all agents performs demonstration actions $A_g^D(s)$ simultaneously. Simulate the game until the state reaches a terminal state s_T, the state value based on demonstration actions $V^D(s)$ can be calculated by Eq. (1). Then the state s is regarded as a training sample, and its label is the simulated $V^D(s)$, which is similar to expert trajectory return [Hester *et al.*, 2018]. After collecting enough samples, the critic network is trained in a supervised manner:

$$\theta^V=\arg\min_\theta\sum_s(V(s,\theta)-V^D(s))^2, \tag{3}$$

where $V(s,\theta)$ is the output of the critic network. In an episodic problem, the state value often involves the task goal. Hence, the sparsity of the state values could be much less than rewards, which makes it easier to train the critic network.

4.2 Actor Network Learning

According to the critic network, the centralized Q value of any state-action pair (s,A) are estimated by Eq. (2), written as $Q^\theta(s,A)$. Then the joint multi-agent decisions are modeled as a combinatorial optimization problem which aims to maximize Q value:

$$A^*(s)=\arg\max_{A\in\mathcal{A}}Q^\theta(s,A). \tag{4}$$

As the worst-case complexity of Eq. (4) grows exponentially concerning the number of agents, we alternatively solve a joint local optimal solution $\tilde{A}=\{\tilde{a}_g\}_{g\in G}$ where each agent's action is its best response to others' actions:

$$Q^\theta(s,a_g,\tilde{A}_{-g})\le Q^\theta(s,\tilde{A}), \tag{5}$$

where a_g is any unilateral deviation and $\tilde{A}_{-g}=\{\tilde{a}_{g'}\}_{g'\in\mathcal{G}}^{g'\neq g}$. To solve Eq. (5), the best response dynamics algorithm is introduced in Alg. 1. It can be proven that Alg. 1 converges to a local optimum [Roughgarden, 2016]: In each iteration, the centralized state-action value Q increases. Since the state-action value is bounded, hence Alg. 1 eventually halt.

Algorithm 1: The best response dynamics algorithm.

Input: State $s\in\mathcal{S}$, agents $\mathcal{G}$, critic network and demonstration actions $A^D(s)=\{a_g^D(s)\}_{g\in\mathcal{G}}$.
Output: $\tilde{A}(s)=\{\tilde{a}_g(s)\}_{g\in\mathcal{G}}$.
Initialize $\tilde{A}(s)=A^D(s)$.
for $iter=1,2,...,Iterations$ **do**
  for $g\in G$ **do**
    Calculate the response for each action a by the critic network: $Q^\theta(s,a,\tilde{A}_{-g})$.
    Choose the action $a_g(s)=\arg\max_{a\in\mathcal{A}^g}Q^\theta(s,a,\tilde{A}_{-g})$.
    Update $\tilde{A}$ by $\tilde{a}_g(s)=a_g(s)$.

Here are some analyses about Alg. 1: Firstly, given the critic network, the Q value is estimated using Eq. (2) which is solved by one-step forward simulation. Besides, Alg. 1 requires the full state and free communications of all agents. Fortunately, Alg. 1 is only employed in training which is centralized and carried out in the simulator, and these requirements are fulfilled in many realistic simulators [Chen *et al.*, 2017; Tumer and Agogino, 2007]. Secondly, Alg. 1 needs "$Iterations\times N$ " computations where each computation contains a one-step simulation and forwarding the critic network once. That is, it is efficient with linear complexity concerning the agents' number. Thirdly, although Alg. 1 achieves local optimum [Roughgarden, 2016] and there is no theoretical guarantee of global optimality, yet it is still effective to learn better policy from initial actions in practice. Due to the strong fitting ability of deep neural networks, the fitting error of Eq. (3) could be reduced by using enough training samples. Consider an ideal condition where the fitting error of Eq. (3) is ignored, we can prove $\tilde{A}$ is better than demonstration actions A^D, that is, $\tilde{V}(s)\ge V^D(s)$ for any $s\in\mathcal{S}$ where $\tilde{V}$ and V^D are calculated by Eq. (1) by simulating $\tilde{A}$ and A^D.

Proof: For any $s\in\mathcal{S}$, consider an episode initialized by $s_1=s$. Since Alg. 1 is initialized from A^D, and the state-

Algorithm 2: The proposed learning algorithm.

Input: Demonstration actions $A^D(s) = \{a_g^D(s)\}_{g\in\mathcal{G}}$.
Output: Critic network parameters θ^V and actor networks parameters $\{\theta_g^\pi\}_{g\in\mathcal{G}}$.

```
Generate training samples (s, V^D(s)) by A^D(s) and
  store the samples to a data buffer B.
Update θ^V by Eq. (3).
Initialize {θ_g^π}_{g∈G} randomly.
for ep = 1, ..., Episodes do
    Initialize s_1 randomly.
    for t = 1, ... do
        if s_t is terminal then
            Break;
        Calculate Ã(s_t) by Alg. 1.
        for g ∈ G do
            Calculate advantage function by Eq. (7)
              and update θ_g^π by Eq. (8).
        Perform actions
          {a^g = arg max_a p(a|O^g(s_t), θ_g^π)}_{g∈G} and
          collect reward r_t.
    Calculate the state value V(s_t) = Σ_{i=t} λ^{i-t} r_i for
      any s_t and update B with {s_t, V(s_t)}.
    if ep mod E_Update = 0 then
        Update θ^V using B by Eq. (3).
```

action value increases in every iteration of Alg. 1, we have:

$$
\begin{aligned}
&V^D(s_1) = \mathbb{E}\left(Q^D(s_1, A^D(s_1))\right) \le \mathbb{E}\left(Q^D(s_1, \tilde{A}(s_1))\right) \\
&= \mathbb{E}\left(r(s_2, s_1, \tilde{A}(s_1)) + \lambda V^D(s_2)\right) \\
&\le \mathbb{E}\left(r(s_2, s_1, \tilde{A}(s_1)) + \lambda r(s_3, s_2, \tilde{A}(s_2)) + \lambda^2 Q^D(s_3, \tilde{A}(s_3))\right) \\
&... \le \mathbb{E}\left(\sum_{t=1}^{T-1} \lambda^{t-1} r(s_{t+1}, s_t, \tilde{A}(s_t))\right) = \tilde{V}(s_1).
\end{aligned} \tag{6}
$$

In summary, $\tilde{A}$ is a better policy than demonstration actions A^D. Hence, it can be used to guide the actor networks. Specifically, the advantage function of an agent action (a_g) can be estimated by the Q value the agent can gain when performing the action in $\tilde{A}$, that is:

$$\mathbb{A}^g(s, a_g) = Q^\theta(s, a_g, \tilde{A}_{-g}) - \min_{a'\in\mathcal{A}_g} Q^\theta(s, a', \tilde{A}_{-g}), \tag{7}$$

where $\mathbb{A}^g(s, a_g)$ is the continuous form extended from $\tilde{A}$ and $\tilde{a}_g(s) = \arg\max_{a_g\in A_g} \mathbb{A}^g(s, a_g)$. Generally speaking, an action with higher $\mathbb{A}^g(s, a_g)$ should have a larger probability to perform. Hence, the actor network θ_g^π is updated as:

$$\theta_g^\pi \leftarrow \theta_g^\pi + \eta \sum\nolimits_{a_g} \nabla_{\theta_g^\pi} \log p(a_g|O^g(s), \theta_g^\pi)\mathbb{A}^g(s, a_g), \tag{8}$$

where $p(a_g|O^g(s), \theta_g^\pi)$ is the output of the actor network θ_g^π and η is the learning rate.

4.3 Joint Update

According to Eq. (6), the initial demonstration actions are used to generate expert trajectory returns and train the critic network, then Alg. 1 finds a better policy than demonstration actions, which is used to learn actor networks. Moreover, the actor networks can also be used to generate action as $a^\theta(s) = \arg\max_{a\in A_g} p(a|O^g(s), \theta_g^\pi)$, and these actions are considered as new demonstration actions. An asynchronous RL algorithm concludes this iterative learning procedure in Alg. 2. In summary, the critic network is used to train the actor networks by Eq. (8), then the actor networks update the state values, and the updated state values will be used to train the critic network by Eq. (3) alliteratively.

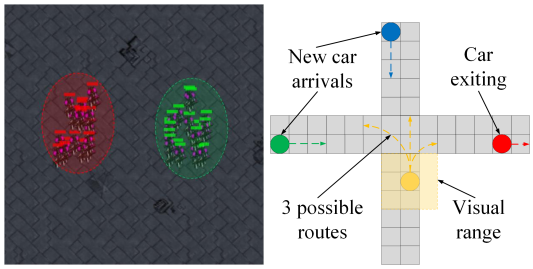

Figure 2: Left: RTS combat game task where we control red units against to the green units. Right: Traffic junction task where agent-controlled cars (circles) pass through the junction without colliding.

5 Experiments

5.1 RTS Combat Game

As enemy units and the opponent's strategy are modeled as a part of environments, RTS combat game has been widely utilized to test the multi-agent cooperative algorithms [Peng *et al.*, 2017; Lowe *et al.*, 2017; Foerster *et al.*, 2018; Rashid *et al.*, 2018]. It is a significant challenge because the state-action space is extremely large and the time allowed for planning is on the order of milliseconds. Our approach is tested using SparCraft [Churchill *et al.*, 2012], which is a simulator of the StarCraft local combat game and is widely adopted to test AL algorithms [Churchill and Buro, 2013; Lelis, 2017; Moraes and Lelis, 2018]. We choose it as the experimental platform because it implements several different heuristics which can be regarded as demonstrations.

The **action space** of each agent contains three types of discrete actions: noop, move$[1, ..., directions]$ and attack$[1, ..., enemy_ids]$, where $directions$ is set to 4 corresponding to left, right, up and down, and $enemy\_ids$ is the number of enemy units at the initial game state. To an allied unit, $[1, ..., enemy\_ids]$ is a list of its attack targets, and it is arranged in ascending order of the remaining hit points (HP). The **reward** is calculated when the terminal state is reached or the attacks happen. It equals the HP change of all units compared to the previous state. The rewards are normalized by dividing the sum of all allied units' HP at the initial state, and are sparse when the units are only moving. The **feature vector** of an agent's observation consists of: the attributes of the corresponding unit, the ten closest allied units and ten closest enemy units. To make the feature vector unique and identical to each state, the allied units and enemy units are arranged in ascending order of the distance to the corresponding unit. If the numbers of allied and enemy units are less than

10, then the attributes of allied and enemy units are filled with 0 respectively to make the length of feature vector identical at any state. For the critic network, the global state feature contains the statistical properties of all allied and enemy units, respectively, such as the mean and variance of the current HP, weapon cooldown, and positions. We employ the statistical properties as the feature of the critic network rather than concatenating properties of all units because the low-dimensional feature leads to less critic network parameters to handle over-fitting. The **actor and critic networks** are both composed of 4 fully connected (FC) layers, and three batch normalization layers following the first 3 FC layers, respectively. A softmax layer is used in the actor networks to output the probability distribution. The widths of the FC layers in the actor networks and critic network are 512, 256, 256, $directions + enemy\_ids + 1$ and 256, 128, 128, 1 respectively. To recover actions from demonstrations, the network architecture of θ_g^D (Sec. 3) is the same as the actor network. We use this simple architecture because the actor networks need to make decisions in real-time. As suggested in [Gupta *et al.*, 2017], the homogeneous agents share the same actor network parameters to make learning more efficient.

The iterations of Alg. 1, E_{Update} of Alg. 2 and λ are set to 7, 500 and 0.995, respectively. The mean win rates and terminal hit points reward [1] over 100 battles are used as evaluation metrics. We utilize 2 rule-based heuristics as our demonstration policies, including: Attack-Closest (*c*) where agents attack the closest enemy within weapon range, and any agent not within the range of any enemy moves toward the closest enemy, and Attack-Weakest (*w*) where agents attack the enemy with minimum remaining hit points. These two heuristics are both sub-optimal. For example, they both ignore multi-agent cooperation and make decisions for each agent independently. We generate 10,000 observations by these two heuristics as our demonstrations, and 100,000 trajectories are simulated to pre-train the critic network in Alg. 2. All networks are optimized by SGD with learning rate 10^{-3}. For each combat scenario, two circular regions are chosen for allied and enemy units, respectively, and each unit is randomly born in the corresponding region. Our experiments are conducted on cross-validation setting that the model learned from *c* will fight with *w* and vice versa. The Marine (m) is chosen as the tested unit type. The test combat scenarios differ in different scales and difficulties: a small scale combat "m5v5", a large scale combat "m30v30", and two unbalanced combats "m18v20" and "m24v30" where we control 18 (24) Marines against 20 (30) Marines. These combats, especially the unbalanced combats, require effective multi-agent cooperation to defeat enemies. In test, the legal action with the maximum probability according to the actor network is performed. The models are trained on GeForce GTX 1080 and tested on a PC with one 2.4 GHz CPU and 8G RAM.

5.2 Experimental Results

The compared methods are demonstration based ones: 1) The demonstration policy (D), i.e, the corresponding heuristic; 2) Heuristic-based search methods, including UCT [Churchill and Buro, 2013], Alpha-Beta (A-B) [Churchill *et al.*, 2012], and profile search [Churchill and Buro, 2013], which have been implemented in SparCraft; 3) Demonstration-based learning methods, including behavioral cloning (BC) [Pomerleau, 1991], DQfD [Hester *et al.*, 2018], and OGTL [Hu *et al.*, 2018]. We choose these methods because they can learn decentralized policies and are designed by totally different principles. BC is a baseline of imitation learning [Ho and Ermon, 2016; Song *et al.*, 2018], which aims to recover experts' policies from demonstrations, DQfD utilizes demonstrations to guide the exploration of RL in sparse reward problems, and OGTL [Hu *et al.*, 2018] pre-train the network by demonstrations and refine the network by RL. All these methods are re-implemented by the same setting and network architecture. The only difference is that DQfD and OGTL need rewards of single-agent action, which is same to [Hu *et al.*, 2018].

	m5v5		m30v30		m18v20		m24v30	
Metrics	W	R	W	R	W	R	W	R
D_*c*	0.43	-0.02	0.75	0.18	0.32	-0.11	0.12	-0.26
UCT_*c*	0.93	0.06	0.88	0.20	0.40	-0.14	0.13	-0.25
A-B_*c*	0.96	0.07	0.85	0.21	0.36	-0.17	0.12	-0.24
Profile	0.84	0.14	0.99	0.33	0.61	0.06	0.30	-0.11
BC_*c*	0.42	-0.04	0.76	0.19	0.28	-0.12	0.10	-0.29
DQfD_*c*	0.85	0.16	0.90	0.41	0.47	-0.10	0.23	-0.20
OGTL_*c*	0.89	0.08	0.93	0.46	0.48	-0.07	0.26	-0.16
Ours_*c*	**0.98**	**0.18**	**1.00**	**0.51**	**0.87**	**0.24**	**0.76**	**0.19**
D_*w*	0.61	0.02	0.13	-0.23	0.03	-0.36	0.00	-0.49
UCT_*w*	0.93	0.01	0.73	0.09	0.37	-0.19	0.00	-0.36
A-B_*w*	0.92	0.02	0.71	0.09	0.34	-0.26	0.00	-0.38
Profile	0.91	0.16	**0.96**	0.21	0.39	-0.08	0.02	-0.29
BC_*w*	0.60	0.02	0.13	-0.23	0.02	-0.37	0.01	-0.49
DQfD_*w*	0.89	0.08	0.81	0.17	0.30	-0.14	0.03	-0.37
OGTL_*w*	0.97	0.09	0.78	0.15	0.36	-0.12	0.06	-0.38
Ours_*w*	**0.99**	**0.19**	0.91	**0.22**	**0.71**	**0.16**	**0.58**	**0.03**

Table 1: A comparison with other demonstration based approaches where "_*c*" ("_*w*") means the used demonstrator policy is *c* (*w*). "W" and "R" are mean win rates and terminal hit points rewards respectively. The best result for the given scenario is in bold.

The result in Table 1 verifies: 1) Our approach outperforms other methods as well as the demonstration policies significantly in most testing scenarios. The advantage is evident in unbalanced combats. For example, in m24v30 with demonstration *w*, the win rates of other methods are all bellow 0.10, while our approach gets 0.58. 2) The search-based methods UCT and Alpha-Beta can improve demonstration policies clearly in balanced combats, while the improvement is very limited in unbalanced combats. It is difficult to search the effective multi-agent cooperative policies from the huge game tree under real-time limitations. Profile search selects the heuristic without additional exploration, and it is limited when the candidate heuristics are both sub-optimal. Besides, these methods rely on global state and free communication among agents, while our method only requires agents' ob-

[1] The sum of the allied units' HP minus those of all enemies.

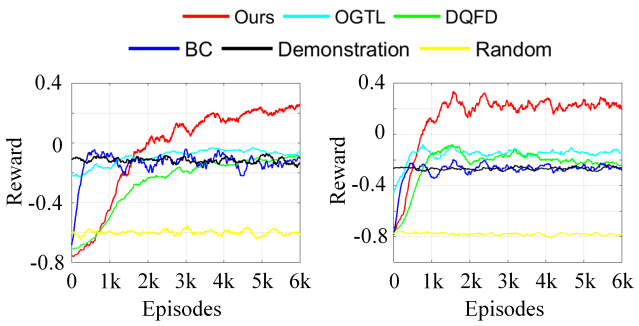

Figure 3: The terminal HP rewards of training episodes in m18v20 (Left) and m24v30 (Right) by c. "Random" means the network are set randomly, and OGTL is pre-trained by imitation learning. The shown reward is the mean value of the following 100 episodes.

servations in execution. 3) Fig. 3 indicates the terminal hit points reward curve with different training episodes of some learning-based methods. BC converges quickly, and the final performance is close to the demonstrator policy. Compared with BC, our method outperforms the demonstrator policy. The reason is that we only utilize demonstrations to pre-train the critic network, and Alg. 1 improves the initial policy effectively. DQfD constraints the learning policy close to demonstrations, and it limits the model when the demonstrations are sub-optimal. OGTL works well in balanced combats while performs poorly in unbalanced combats because it ignores multi-agent cooperation during learning, and the cooperation is more critical in unbalanced combats.

5.3 Further Evaluation

To validate the generality of the proposed approach, an additional experiment is conducted on the traffic junction task [Sukhbaatar and Fergus, 2016]. As shown in Fig. 2, the task consists of a 4-way junction on a 14×14 grid. New cars enter the grid with probability P_{arrive} from each of the four directions, and the total car number at any time step is limited to 10. Each car is randomly assigned to one of 3 possible routes and keeps to the right-hand side of the road. Once a car arrives at its destination (i.e., moves outside the junction area), it will be removed. A collision happens when two cars move to the same location. The episode is terminated after 40 steps and is viewed as a failure if one or more collisions have occurred. A car is allowed to perform two types of actions at each time step: advances by one cell while keeping on its route or stay still at the current position.

A collision incurs a reward $r_c = -5$, and each car gets reward of $r_a = 1$ if it moves outside the junction area. Thus, the global reward at time t is $N_c^t r_c + N_a^t r_a$ where N_c^t and N_a^t are numbers of collisions and arrivals. The reward is sparse and shared by all agents. Most hyper-parameters are the same as the RTS combat task. The only differences are $E_{Update} = 100$, and 20,000 samples are generated to pre-train the critic network in Alg. 2. Each car can only observe other cars in its vision range (a surrounding 3×3 neighborhood), and can communicate to all other cars in training. In the test phase, the communication is unavailable. For the actor network, the feature vector is the observation of a car, which is the same to [Sukhbaatar and Fergus, 2016]. For the critic network, the representation of each car combines its observation and action one-hot encoding. The feature vector concatenates representations of all cars each way. For positions without a car, a nominal car whose representations are 0 is assigned to make the length of feature vector identical. The network architectures are the same to the RTS combat task except that the output node number is 2 in the actor network.

Methods	D	BC	Ours w/o D	Ours
FR (%)	0.72	0.72	0.35	**0.19**

Table 2: Experimental results on the traffic junction task. "FR" means failure rate.

The comparative results are shown in Table 2. We do not evaluate DQFD and OGTL in this task because the collisions are related to multiple agents, and the reward cannot be decomposed directly. The network architecture and features used in BC are the same as the actor network for a fair comparison. BC can achieve very close results with "D" because the used demonstrator policy is easy to imitate, and the network needs to learn "forward" at any time. The proposed approach outperforms the demonstration and BC clearly, and it indicates the proposed approach is general to different tasks.

6 Conclusion

This paper introduces a method to learn decentralized policies by global rewards from sub-optimal demonstrations. It proposes a new multi-agent actor-critic learning method to tackle the challenges from the sub-optimality of the demonstrations and multi-agent credit assignment challenge. Extensive experimental results on the RTS combat game demonstrate the effectiveness of the proposed approach.

References

[Chen *et al.*, 2017] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *International Conference on Robotics and Automation*, pages 285–292, 2017.

[Churchill and Buro, 2013] David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *IEEE's Conference on Computational Intelligence in Games*, pages 1–8, 2013.

[Churchill *et al.*, 2012] David Churchill, Abdallah Saffidine, and Michael Buro. Fast heuristic search for RTS game combat scenarios. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 112–117, 2012.

[Foerster *et al.*, 2018] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, pages 2974–2982, 2018.

[Gupta *et al.*, 2017] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control us-

ing deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83, 2017.

[Hester *et al.*, 2018] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Horgan Dan, John Quan, Andrew Sendonaris, and Gabriel Dulacarnold. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, pages 3223–3230, 2018.

[Ho and Ermon, 2016] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[Hu *et al.*, 2018] Yue Hu, Juntao Li, Xi Li, Gang Pan, and Mingliang Xu. Knowledge-guided agent-tactic-aware learning for StarCraft micromanagement. In *International Joint Conference on Artificial Intelligence*, pages 1471–1477, 2018.

[Lelis, 2017] Levi H. S. Lelis. Stratified strategy selection for unit control in real-time strategy games. In *International Joint Conference on Artificial Intelligence*, pages 3735–3741, 2017.

[Li, 2018] Yuxi Li. Deep reinforcement learning. *arXiv preprint arXiv:1810.06339*, 2018.

[Lin *et al.*, 2018] Xiaomin Lin, Peter A. Beling, and Randy Cogill. Multiagent inverse reinforcement learning for two-person zero-sum games. *IEEE Transactions on Games*, 10(1):56–68, 2018.

[Lowe *et al.*, 2017] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[Moraes and Lelis, 2018] Rubens O. Moraes and Levi H. S. Lelis. Asymmetric action abstractions for multi-unit control in adversarial real-time games. In *AAAI Conference on Artificial Intelligence*, pages 876–883, 2018.

[Ng and Russell, 2000] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670, 2000.

[Nguyen *et al.*, 2018] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent rl with global rewards. In *Advances in Neural Information Processing Systems*, pages 8102–8113, 2018.

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

[Oliehoek *et al.*, 2008] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *JAIR*, 32(1):289–353, 2008.

[Peng *et al.*, 2017] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play StarCraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[Pomerleau, 1991] Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[Rashid *et al.*, 2018] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4292–4301, 2018.

[Ross *et al.*, 2011] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, volume 15, pages 627–635, 2011.

[Roughgarden, 2016] Tim Roughgarden. *Twenty Lectures on Algorithmic Game Theory*. Cambridge University Press, 2016.

[Schaal, 1996] Stefan Schaal. Learning from demonstration. In *Advances in Neural Information Processing Systems*, pages 1040–1046, 1996.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Song *et al.*, 2018] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 7461–7472, 2018.

[Sukhbaatar and Fergus, 2016] Sainbayar Sukhbaatar and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2252–2260, 2016.

[Sun *et al.*, 2017] Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International Conference on Machine Learning*, pages 3309–3318, 2017.

[Tumer and Agogino, 2007] Kagan Tumer and Adrian K. Agogino. Distributed agent-based air traffic flow management. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 1–8, 2007.

[Vecerik *et al.*, 2017] Matej Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothorl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.