Cooperative Multi-Agent Policy Gradients with Sub-optimal Demonstration

Peixi Peng¹, Junliang Xing¹
1, Institute of Automation, Chinese Academy of Sciences

Abstract

Many reality tasks such as robot coordination can be naturally modelled as multi-agent cooperative system where the rewards are sparse. This paper focuses on learning decentralized policies for such tasks using sub-optimal demonstration. To learn the multi-agent cooperation effectively and tackle the sub-optimality of demonstration, a self-improving learning method is proposed: On the one hand, the centralized state-action values are initialized by the demonstration and updated by the learned decentralized policy to improve the sub-optimality. On the other hand, the Nash Equilibrium are found by the current state-action value and are used as a guide to learn the policy. The proposed method is evaluated on the combat RTS games which requires a high level of multi-agent cooperation. Extensive experimental results on various combat scenarios demonstrate that the proposed method can learn multi-agent cooperation effectively. It significantly outperforms many state-of-the-art demonstration based approaches.

Introduction

By combining with deep learning models (LeCun, Bengio, and Hinton 2015), reinforcement learning (RL) has achieved successful applications to many challenging problems in these years such as Go (Silver et al. 2017), general Atari game-playing (Mnih et al. 2015) and robot motor control (Levine et al. 2016). Most of these works involve only a single agent. However, many real-world problems such as autonomous vehicle coordination (Cao et al. 2012), network packet delivery (Wang and Sang 2005) and playing combat games (Usunier et al. 2016) are naturally modelled as cooperative multi-agent systems (Stone and Veloso 2000). The aim is coordinating multiple agents to achieve a unified goal or maximum team benefits by cooperation.

Since the joint state-action space of the agents grows exponentially with the number of agents, RL methods designed for single agents typically fare poorly on such tasks. To cope with this complexity, it is often necessary to resort to decentralised policies, in which each agent selects its own action conditioned only on its local action-observation history (Foster et al. 2018). RL entails the knowledge of the reward function, or at least observations of immediate reward. Unfortunately, the rewards of multi-agent states need to take into account a wide range of factors such as the joint states of all agents. Hence, it is very difficult to calculate the reward of most states in many reality multi-agent systems. By contrast, it is often quite natural to express a task goal as a sparse reward function (Vecerik et al. 2017). For example, the terminal state reward of a combat game can be easily defined by victory or defeat, while it is hard to evaluate the combat situation at non-terminal states. Another typical example is the multiple robot coordination (Chen and et al. 2010) (Everett, Chen, and How 2018) where the rewards are only observed directly in a few situations such as collision or arrival at the destination. A widely-used approach to the sparse rewards is learning from demonstrations (LfD) (Schaal 1997; Hester and et al. 2017). In many reality applications, although there is very little or no knowledge of the reward function, but there is access to demonstrations performed by experts in the task. A natural method is to recover experts’ strategies from available demonstrations by supervised learning (Ross, Gordon, and Bagnell 2010). It assumes that the experts are performing well. Agents should also be able to perform well by simply mimicking experts’ moves. Another type of approaches, termed inverse reinforcement learning (IRL) (Ng and Russell 2000), formulates the task as solving an inverse problem, and strives to infer the hidden reward function from expert demonstrations. Most existing IRL methods assume that the demonstrations are generated by an optimal strategy. However, due to the extremely large state action space of multi-agents systems, it’s hard to design an optimal strategy manually even for experts. The available demonstrations are always sub-optimal, and need to be improved rather than just imitating the demonstrator or inferring the reward from the demonstration directly.

Another crucial challenge is multi-agent credit assignment (han Chang, Ho, and Kaelbling 2004): In cooperative setting, the agents actions are performed simultaneously and the state-action value are centralized and shared by all agents. Hence, even if the centralized state-action value can be evaluated from a demonstration, it is still hard to identify each agent action’s own contribution. Some existing works design individual reward functions for each agent by one agent action exploration (Tamppu et al. 2017). However, these rewards ignore multi-agent cooperation and often fail to encourage individual agents to sacrifice themselves for team advantages. For example, in a combat, the
The rest of the paper is organized as follows: Section 2 re-views the related work and highlights our technical contributions. Section 3 formulates the multi-agent learning task by the cooperative game. The learning algorithm is described in Section 4. Extensive experimental analysis and comparisons are discussed in Section 5. The paper concludes with Section 6.

**Related Work**

Multi-agent Reinforcement Learning (MARL) methods have historically been applied in many settings (Tuyls and Weiss 2012, Busoniu, Babuska, and De Schutter 2008, Tuyls and Weiss 2012). They have been restricted to tabular methods and simple environments. Motivated from the success of deep RL (Mnih et al. 2015) where value/policy is approximated by deep neural networks, the deep MARL methods can scale to high dimensional input and action spaces (Tampuu et al. 2017). Independent Q-learning (Tan 1993) is proposed to learn decentralised value functions or policies for each agent independently. It is extended to deep neural networks in (Tampuu et al. 2017) by combining with DQN (Mnih et al. 2015). Some methods (Foerster et al. 2017, Omidsfahieie et al. 2017) are proposed to address learning stabilisation following this idea. However, these methods always learn each agent independently and ignore the cooperation among agents.

Another type of methods focuses on centralised learning of joint actions which can naturally handle cooperation problems. However, the joint action space will be extremely large when the system contains a lot of agents, which may lead the learning inefficient. Coordination graphs (Guestrin, Koller, and Parr 2002) is proposed to exploit conditional independencies between agents by decomposing a global reward function into a sum of agent-local terms. Sparse cooperative Q-learning (Kok and Vlassis 2006) proposes a tabular Q-learning algorithm that learns to coordinate the actions of a group of cooperative agents only in the states where such coordination is necessary. Recently, more centralised learning methods are proposed to multi-agent communication by combing with deep learning. CommNet, as designed in (Sukhbaatar and Fergus 2016), uses a centralised network architecture to handle communication between agents. The bidirectionally-coordinated network (BiCNet) is introduced in (Peng et al. 2017) to maintain a scalable and effective communication protocol among multiple units, and it additionally requires estimating individual agent rewards. However, only centralized state-value is known in our cooperative setting. GMEZO (Usunier et al. 2016) models the multi-agent system as a greedy Markov decision procedure and present an RL algorithm using first-order optimisation. To handle an extremely large number of agents, mean field RL (Yang et al. 2018) is proposed to describe the interaction of an agent with its neighboring agents. Recently, several works are designed as hybrid approaches. QMIX (Kashid et al. 2018) decomposes the centralized state-action values as a group of single agent’s values and enforce that \( \arg \max \) performed on the centralized value yields the same result as a set of individual \( \arg \max \) operations performed on each agent. This assumption fails in some cooperation tasks where an agent should sacrifice itself to the benefit of the other agents.

To cope with these difficulties, a novel RL method is proposed to learn multi-agent cooperation by sub-optimal demonstrations. As shown in Fig. 1, the proposed method involves self-improvement: On the one hand, although the demonstrations are not optimal, it still can be assumed that experts perform well. Hence, the state-action value can be initialized by the experts’ demonstrations at the early stage of learning. To take the sub-optimality of demonstration into account, the state-action value is updated by the learned policy during the learning procedure. In other words, the sub-optimal demonstrations offer initial guidance to ensure that the learned policy is better than experts.

On the other hand, the tasks are modeled as the cooperative game and the Nash Equilibrium strategies are found by current centralized state-action value. These Nash Equilibrium strategies can be used to learn decentralised policies and it is the main difference between the proposed method with other multi-agent policy methods (Foerster et al. 2018, Sukhbaatar and Fergus 2016). The Nash Equilibrium reduces the state-action exploration space and ensures more effective learning of multi-agent cooperation. These two steps can complement and improve each other: the centralized state-action value can be used to find the Nash Equilibrium strategies to learn better decentralized policy; whilst the learned decentralized policies can help to calculate more accurate state-action values.

To summarize, the main contributions of this work are threefold: (1) a novel learning algorithm is proposed to learn multi-agent cooperation with sparse reward by sub-optimal demonstration, (2) the task is modeled as cooperative game and Nash Equilibrium strategy is found to learn decentralized policy, and (3) the state-action value function is updated in learning to tackle the sub-optimality of demonstration.

The rest of the paper is organized as follows: Section 2 reviews the related work and highlights our technical contributions. Section 3 formulates the multi-agent learning task by the cooperative game. The learning algorithm is described in Section 4. Extensive experimental analysis and comparisons are discussed in Section 5. The paper concludes with Section 6.
team. In addition, several methods are designed by learning decentralized policy using centralized critic. COMA \cite{Foerster2017} estimates a counterfactual advantage function to update decentralized policy. Similarly, \cite{Lowe2017} learn a centralised critic for each agent and apply this to competitive games with continuous action spaces. In these methods, one agent’s policy is updated by assuming other agents’ policies is known on-policy. Hence, they have poor sample efficiency and are prone to getting stuck in sub-optimal local minima \cite{Rashid2018}. Different with them, the proposed method solves the the Nash Equilibrium to learn decentralised policies. The Nash Equilibrium reduces the exploration state-action space and leads to more effective learning of cooperation. A similar method is proposed in \cite{Lanctot2017} where one agent update it’s policy by sampling other agent’s policies from their individual meta-strategies. Compared with \cite{Lanctot2017}, the proposed method assumes all agents are performing as Nash Equilibrium jointly and it is more reasonable to multi-agent cooperation. All of these deep MARL methods assume that the reward is available at every state.

**Learning from Demonstration (LfD)** \cite{Schaal1997} has received increasing attention as a promising way to tackle the problem of sparse rewards. Imitation learning is primarily concerned with recovering the strategy of the demonstrator in a supervised fashion. DAGGER \cite{Ross2010} formulates the task as essentially a regression problem and sets the objective of minimizing the prediction error. Deeply AggreVaTed \cite{Sun2017} extends DAGGER to continuous action spaces by combining with deep neural networks. These methods just imitate the demonstration and can not improve upon the expert. To improve the demonstration, several methods \cite{Mnih2015, Hu2018} train the network by combining imitation learning and RL together. However, AlphaGo \cite{Mnih2015} is designed for a single agent and OGTL \cite{Hu2018} ignores multi-agent cooperation during learning. Recently, a few approaches are proposed to explore the sparse-reward environment by LfD. For instance, DQfD \cite{Hester2017} introduces LfD into DQN adding demonstration data into the replay buffer. Similarly, DDPGfD \cite{Vecerik2017} extends LfD to continuous action domains. POiD \cite{Kang2018} leverage demonstrations to guide exploration through enforcing occupancy measure matching between the learned policy and current demonstrations. These methods are all designed for single agent systems and aim at exploring around demonstration to find a better policy. However, the joint state-action space of multiple agents is very large and the optimal policy may be far from experts’ policy with a large probability. Another type of LfD methods is inverse reinforcement learning (IRL) \cite{Ng2000} which targets at recovering the reward function of a given task from samples, and it is extended to two-player zero-sum game \cite{Syed2008} by alternating between fitting the reward function and selecting the policy. GAIL \cite{Ho2016} applies the IRL to high-dimensional continuous control problems. The multi-agent IRL methods \cite{Reddy2012, Lin2018} are proposed to a non-cooperative game and zero-sum stochastic games respectively. All these methods assume that the demonstration is optimal. Besides these methods, \cite{Wang2018} propose to handle sub-optimal demonstrations in two-player zero-sum competitive games. The centralized rewards are inferred by two-player zero-sum constraint. A network is utilized to approximate the rewards by centralized learning. However, the proposed method focuses on learning decentralized policy in multi-agent cooperative tasks which don’t satisfy the two-player zero-sum constraint.

**Problem Definition**

In the Markov Decision Procedure (MDP) model, the multi-agent system with sparse reward has the following components:

- **States**: The states are modeled by a time series \( S = \{s_0, s_1, ..., s_T\} \) where \( s_t \) stands for the state at time \( t \) and \( s_T \) is the terminal state.

- **Agents**: \( G = \{g_1, g_2, ..., g_N\} \) are agents and \( N \) is the number of the agents. Note the proposed method can be still applicable when \( N \) is varied at different states.

- **Actions**: \( A^g(s)|s \in S\) defines the action space for agent \( g \in G \). \( A^g(s) \) is the finite set of candidate action set that can be performed by agent \( g \) at state \( s \). For simplicity, suppose that the candidate actions set of agent \( g \) is same at every state, then \( A^g \) refers to the available action set for agent \( g \). At each time step \( t \), each agent \( g \) simultaneously chooses an action \( a^g_t \in A^g \), forming a joint action set \( A_t = \{a^g_t\}_{g \in G} \).

- **State transition**: \( P_{t+1} = P(s_{t+1}|s_t, A_t) \) is the state transition probability. In an MDP, the transition probability depends only on the current state and the agents’ actions at the corresponding state.

- **Decentralized policy**: Given a state \( s \) and an agent \( g \in G \), the decentralized policy is the set of probability distributions over all actions: \( p(a|g, s) \) where \( a \in A^g \). The decentralized policy learning is to find a optimal policy \( \{p^g(a|g, s)\}_{a \in A^g} \) where \( \theta \) is the model parameters (i.e., the network parameters in the proposed method).

- **Rewards**: The reward function \( R(s_{t+1}|s_t, A_t) \) is bounded and used to measure the reward (or payoff) of the states transition: \( s_t \times A_t \times s_{t+1} \rightarrow R \). We use \( R(s_{t+1}) \) to represent it for simplicity. In our task, rewards can only be defined in a few states. Without loss of generality, the rewards of the terminal states can be calculated as \( R(s_T) \) and \( R(s_i) = 0 \) at other states.

- **State-action value (Q)**: Given a policy \( \pi \), the \( p \) based state-action value \( Q^\pi(s, a) \) can be calculated by the cumulative rewards:

\[
Q^\pi(s, a) = \mathbb{E}_{(s_{t+1} \sim \mathbb{P}_{s_{t+1}, a_{t+1} \sim \pi_t}^{s_t, a_t \sim \pi})_{t=1}^{T-1}} \lambda^{T-1} R(s_{t+1})
\]

\[
= \mathbb{E}_{(s_{t+1} \sim \mathbb{P}_{s_{t+1}, a_{t+1} \sim \pi_t}^{s_t, a_t \sim \pi})_{t=1}^{T-1}} \lambda^{T-1} R(s_T),
\]

where \( s_1 \sim \mathbb{P}(s_1|s, A), \lambda \in [0, 1] \) is a discount factor and \( \mathbb{E} \) is the expectation. The state-action value measures
best cumulative rewards as \( Q^\ast(s, A) = \sup_\eta Q^\eta(s, A) \). Since the multiple agents’ actions are performed simultaneously, all the agents share the state-action value.

- **Demonstrations:** The demonstrations usually fall into two types: rule-based heuristics and experience data. The former are always designed by expert’s prior knowledge. The heuristics are task-specific and the proposed method is independent of the knowledge of its internal workings. The only requirement is the heuristics can be simulated. The later are the observation set \( D = \{ s_m, a^{g_1}(s_m), \ldots, a^{g_\mathcal{N}}(s_m) \}_{m=1}^M \) where shows how experts have performed in the task. Here \( s_m \) is the visited state and \( \{ a^{g_1}(s_m), \ldots, a^{g_\mathcal{N}}(s_m) \} \) are each agent’s actions which are performed by experts at the corresponding state \( s_m \). Besides, \( m \) stands for the \( m \)-th observation and is irrelevant to the time. In order to make the expert’s knowledge available at arbitrary states, a parameterized policy \( p_{\theta_D} \) is used to imitate the experts. It can be trained by minimizing the cross-entropy loss function:

\[
\sum_{m=1}^M \sum_{g \in \mathcal{G}} - \log(p_{\theta_D}(a^g(s_m)|s_m)))
\]

To summarize, these two types of demonstrations can both be written as a policy \( p^\eta(a|g, s) \). A demonstration is optimal if the multi-agent system can get the highest state-action value according to the demonstration policy at every state, i.e. \( Q(s, A) = Q^{\mathcal{D}}(s, A) \) where \( Q^{\mathcal{D}}(s, A) \) is a simplification to \( Q^\eta(s, A) \). Otherwise, the demonstration is sub-optimal.

**Learning Algorithm**

**Cooperative Game and Pure Nash Equilibrium**

The multi-agent cooperation can be naturally modelled as a cooperative game which focuses on how much collective payoff a set of agents can gain by forming a coalition. Given the state-action value \( Q(s, A) \), the pure Nash Equilibrium (PNE) at state \( s \) is a set of actions \( \hat{A} = \{ \hat{a}^g \}_{g \in \mathcal{G}} \) that if for every unilateral deviation \( a^{g'} \in A^{g'} \):

\[
Q(s, a^{g'}, \hat{A}^{-g'}) \leq Q(s, \hat{A})
\]

where \( \hat{A}^{-g'} = \prod_{g \neq g'} \{ \hat{a}^g \} \). To solve \( \hat{A} \), the best response dynamic algorithm is utilized in Alg. 1. It can be easily proved that Alg. 1 is convergent to the PNE: In every iteration, the state reward caused by deviator action strictly increases. Since the game is finite and the state-action value is bounded, hence-best-response dynamics eventually halts, necessarily at a PNE. Alg. 1 can be considered as joint state-action space exploration. It approximates the optimal policy from an exponential space (\( O(|A|^{|\mathcal{G}}|) \)) in polynomial complexity \( O(|A||\mathcal{G}|) \).

**Decentralized Policy Gradient**

PNE is a set of discrete actions which may lose useful information. Considering two typical states, the responses (Alg. 1) of two actions \( a_1 \) and \( a_2 \) are 0.51 and 0.49 respectively in state 1, while their responses are 0.99 and 0.01 respectively in state 2. It indicates \( a_1 \) and \( a_2 \) have a similar effect at state 1, while \( a_1 \) is superior than \( a_2 \) significantly at state 2. However, PNE just save \( a_1 \) in both states and cannot show the difference. To tackle this problem, we recover the response value for each action by \( Q(s, a^{g'}, \hat{A}^{-g'}) \) where \( \hat{A} \) is PNE. \( Q(s, a^{g'}, \hat{A}^{-g'}) \) can be considered as the continuous form extended from PNE and \( \hat{a}^g = \arg \max_{a^g \in A^g} Q(s, a^g, \hat{A}^{-g'}) \). Generally speaking, an action with higher \( Q(s, a^{g'}, \hat{A}^{-g'}) \) should have a larger probability of being performed. Hence, the objective policy of the performed actions of \( g \in \mathcal{G} \) can be estimated by:

\[
\hat{p}(a|g, s) = \frac{Q(s, a^g, \hat{A}^{-g'}) - \min_{a^{g'} \in A^{g'}} Q(s, a^{g'}, \hat{A}^{-g'})}{\sum_{a' \in A^{g'}} Q(s, a', \hat{A}^{-g'}) - |A^{g'}| \min_{a^{g'} \in A^{g'}} Q(s, a^{g'}, \hat{A}^{-g'})},
\]

where \( a \in A^{g} \). Hence, the distribution of parameterized policy \( p^\theta \) should be close to \( \hat{p}(a|g, s) \), and \( \theta \) can be learned by minimum the Kullback-Leibler divergence (\( D^{KL} \)) between \( p^\theta \) and \( \hat{p} \):

\[
\theta^* = \arg \min_{\theta} D^{KL}(\hat{p}||p^\theta)
\]

Then, the decentralized policy gradient \( \nabla_{\theta} \) according to \( \theta \) can be calculated by (5), and \( \theta \) can be learned by \( \theta^{k+1} = \theta^k + \eta^k \nabla_{\theta} \) where \( k \) stands for the \( k \)-th iteration of learning and \( \eta^k \) is the learning rate.

\[
\nabla_{\theta} = \sum_{g \in \mathcal{G}} \sum_{a \in A^{g}} \nabla_{\theta} \log(p_{\theta}(a|g, s)) \times \hat{p}(a|g, s).
\]

**Online Update to Q**

As stated in (5) and (6), the parameter \( \theta \) can be trained by the policy gradient which is dependent on the \( Q(s, A) \). However, \( Q(s, A) \) is unknown because the optimal policy is unknown. Fortunately, although the available demonstration is sub-optimal, the experts still perform reasonably well. Hence, \( Q(s, A) \) can be approximated by \( Q^{\mathcal{D}}(s, A) \) at the beginning of learning. \( Q^{\mathcal{D}}(s, A) \) is similar to expert trajectory return (Hester and etc. 2017), that is, simulating a MDP from

**Algorithm 1:** The best response dynamics algorithm in multi-agent cooperative game.

**Input:** The state \( s_t \), multiple agents \( G = \{ g_1, \ldots, g_N \} \) and \( Q \) function.

**Output:** The PNE \( \hat{A} = \{ \hat{a}^{g_1}, \ldots, \hat{a}^{g_N} \} \).

**for** \( \text{iter} = 1, 2, \ldots, \text{Iterations} \) **do**

**for** \( g \in G \) **do**

- Calculate the response for each action \( a \):
  \( Q(s, a, \hat{A}^{-g}) \)
- Choose the action \( a^g = \arg \max_{a^g \in A^g} Q(s, a^g, \hat{A}^{-g}) \)
- Update \( \hat{A} \) by \( \hat{a}^g = a^g \)


s until the terminal state where the reward is available. In this case, the Lemma 1 are introduced.

**Lemma 1:** If the PNE (i.e., the output of Alg. 1) are calculated by $Q^D(s, a)$, then $Q^N(s, a) \geq Q^D(s, a)$ where $Q^N(s, a)$ is the state-action value \((7)\) according to the PNE.

**Proof:** Consider a MDP initialized by $s_0 = s$. Since the PNE in Alg. 1 are initialized from the demonstration, and the state-action value strictly increases in every iteration, we have:

\[
Q^D(s, a) = E[s_{t+1} \sim P_{t+1}, A_{t+1} = A_t, \{A_i \sim P_i\}_{i>0} | s_t, A_t] T^{T-1} R(s_T) \\
\leq E[s_{t+1} \sim P_{t+1}, A_0 = A_0, A_i \sim A_i, \{A_i \sim P_i\}_{i>1} | s_t, A_t] T^{T-1} R(s_T) \\
\leq E[s_{t+1} \sim P_{t+1}, A_0 = A_0, A_1 \sim A_1, A_i \sim A_i^2, \{A_i \sim P_i\}_{i>2} | s_t, A_t] T^{T-1} R(s_T) \\
\ldots \\
\leq E[s_{t+1} \sim P_{t+1}, A_0 = A_0, A_i \sim A_i | s_t, A_t] T^{T-1} R(s_T) = Q^N(s, a),
\]

(7)

According to Lemma 1, $Q^N(s, a)$ is closer to $Q(s, a)$ than $Q^D(s, a)$ (i.e., $Q(s, a) \geq Q^N(s, a)$). Unfortunately, it is hard to calculate $Q^N(s, a)$ in practice. The calculation of $Q^N(s, a)$ relies Alg. 1 in each state transition of (7). It would be time consuming and leads to the learning inefficient. To reduce computational cost, $Q^g(s, a)$ is used to replace $Q^N(s, a)$:

\[
Q^g(s, a) = E[s_{t+1} \sim P_{t+1}, A_i = \arg\max_p \{P_i\}_{i>0} T^{T-1} R(s_T),
\]

(8)

where $s_0 = s$ and $A_0 = A$. In the learning procedure, the DPN is initialized randomly and can’t model the PNE effectively at some cases during learning procedure. To ensure that the $Q$ robust, it is calculated as (9) in the proposed method:

\[
Q(s, A) \approx \max(Q^g(s, A), Q^D(s, A)).
\]

(9)

Alg. 2 concludes the proposed learning algorithm. It is based on self-improvement. That is, the $Q(s, A)$ are used to guide to update the network and the network can improve $Q(s, A)$ in turn. Note that the joint action-state spaces of multiple agents are extremely large and the feasible action states only occupy a very small portion. To ensure that the train samples effective and abundant, a staged exploration method is utilized. At the early stage of learning, the exploration actions are performed according to the PNE to collect effective train samples efficiently. Then the actions are guided by the network to cover more cases.

**Experiments**

**Combat in RTS Game**

The proposed method are tested in combat RTS game where the task is to coordinate multiple allied agents to defeat their enemies controlled by an opponent in a real-time scenario. The combat game has a high requirement for multi-agent cooperation and has been widely utilized to test multi-agent based methods (Usunier et al. 2016; Peng et al. 2017; Kong et al. 2017; Lowe et al. 2017; Foerster et al. 2018; Rashid et al. 2018; Churchill, Saffidine, and Buro 2012; Churchill and Buro 2013; Leolis 2017). From a machine learning point of view, combat game provides a challenging environment to test AI algorithms because its state-action space is extremely large and the time allowed for planning is on the order of milliseconds. For example, in a specific 15 vs. 15 scenario, the number of points in joint state-action space is almost $10^{13}$, which is much larger than for Go ($10^{127}$) (Silver et al. 2016). The proposed method is testing using SparCraft (Churchill, Saffidine, and Buro 2012), which is a simulator of the StarCraft local combat game and is widely adopted to test combat game algorithms (Churchill, Saffidine, and Buro 2012; Churchill and Buro 2013; Leolis 2017). It is chosen as the experimental platform because of its effective forward simulation function which can make the RL algorithm, especially Alg. 1 more efficiently. In addition, the SparCraft implements several different rule-based heuristics which can be used as our sub-optimal demonstrations. The source code and the trained models will be opened to facilitate further studies of this problem. In our experiment, the combat game is modeled as a sparse-reward task: the reward is only defined at the terminal state (10) where all agents of one player have been wiped out. In (10), $G^a$ are allied agents and $G^e$ are enemies.

\[
R(s_T) = \sum_{g \in G^a} hp(g) - \sum_{g \in G^e} hp(g),
\]

(10)

**Settings**

The decay factor $\lambda$ can be set to any positive number because it doesn’t affect the objective policy (4). The iterations in Alg. 1 is set to 10 in our experiments. The win rate over 100 battles and the normalized rewards at terminal state are used as evaluation metrics. We utilize 2 rule-based heuristics implemented in SparCraft as our demonstrators, including: (1) **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 (2)
Attack-Weakest ($w$): where agents attack the enemy with minimum remaining hp within weapon range, and an agent not within the range of any enemy moves toward the closest enemy. In our experiments, two types of demonstrations are given for each demonstrator: the heuristic simulator and the observed data. The data are used to recover the policy by imitation learning \(Q^D\) and the learned policy is further used to calculate \(Q^D\). As same as most MARL works (Usunier et al. 2016; Peng et al. 2017; Kong et al. 2017; Foerster et al. 2018), the opponent’s policy can be simulated as a part of environment in learning. It is reasonable because our goal is to learn multi-agent cooperation rather than competition. In all the tested combat scenarios, two base points in the combat area are chosen for allied and enemy units respectively. The positions of each player’s agents are initialized randomly within a range of the appropriate base point. That is, the scenarios are different in each combat. Our experiments are conducted on a cross-validation set initialized randomly within a range of the appropriate base point, respectively. The positions of each player’s agents are chosen for allied and enemy units respectively. The environments of each combat area are chosen for allied and enemy units respectively. The positions of each player’s agents are initialized randomly within a range of the appropriate base point. That is, the scenarios are different in each combat. Our experiments are conducted on a cross-validation setting that the model learned from demonstrator $c$ will fight with $w$ and vice versa. The Terrain Marine $(m)$ is chosen as the test agent 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 and our force is much weaker than enemy. These combats, especially the unbalanced combats, require the multiple agents to cooperate effectively to defeat the enemy. The models are trained on GeForce GTX 1080 and tested on a desktop PC with one 2.4 GHz CPU and 8G RAM running in C++. The network can make decision in real time because it only needs forwarding a simple network only once in making each decision.

Feature and Network Architecture

In the experiments, the action space $A$ contains two types of discrete actions: move\(\{\text{directions}\}\) and attack\(\{\text{enemy}_ids\}\), where \{\text{directions}\} is set to 4 corresponding to left, right, up and down, and \{\text{enemy}_ids\} is the number of enemy units in the start game state of the train combat scenario. To an allied agent $g$, the list \{\text{enemy}_ids\} is the list of attack targets of $g$, and it is arranged in two parts sequentially: the enemy agents inside and outside the weapon range of $g$ respectively. In each part, the list is arranged by the ascending order of the remaining hit points $hp$. The feature vector $(f(g, s))$ is concatenated of 4 parts: (1) The properties of $g$ such as maximum $hp$, velocity, weapon damage, maximum cooldown ($cd$, the number of frames to wait before being able to attack again), damage per frame, current $hp$, position (coordinates on the map) and current $cd$. (2) The properties of the enemy agents which are concatenated by the order of \{enemy_ids\}. (3) The properties of the allied agents which are concatenated by ascending order of the distance from the allied agent to the corresponding agent $g$. (4) The statistical properties of the allied and enemy units respectively including the mean, the minimum and the maximum value of the current hit points and the center positions. The positions of the agents except $g$ are all the relative coordinates respect to $g$. If the number of allied and enemy agents in a game state is smaller than initial game state, that is, some agents have been wiped out, then these eliminated agents are replaced by nominal agents whose properties are all 0 to make the length of the feature vector identical. In test combat scenarios, the legal action with maximum probability according to the DPN is chosen to be performed. The network is composed of 4 fully connected (FC) layers, 3 batch normalization layers following the first 3 FC layers respectively and a softmax layer to output the probability distribution over the whole action space $A$. The widths of the FC layers are 256, 128, 128 and directions + enemy_ids respectively. The leaky rectified linear function (Maas, Hannun, and Ng 2013) is used as an activation function for the first 3 FC layers.

Comparison Results

Under this setting that the demonstrations are given, the compared methods can be categorised into two groups: (1) Heuristic-based search methods, including UCT (Churchill and Buro 2013), Alpha-Beta (A-B) search (Churchill and Buro 2013) and profile search (Churchill and Buro 2013). The former two methods use the demonstration policy to guide game tree search. Profile search is proposed to search the optimal heuristic for each agent and two demonstration policy are both used as the candidate policies in the experiments. (2) Deep LfD methods, including DQfQ (Hester and etc. 2017), DDPGfD (Vecerik et al. 2017) and OGTL (Hu et al. 2018). These methods are re-implemented with the same feature and experimental setting for fair comparison. The first 2 methods are all designed for single agent, while they can be easily extended to multi-agent setting by combining with IQL (Tan 1993). OGTL is the latest LfD method designed for multi-agent learning to our knowledge. It utilizes the human-made data to pre-train the network and then fine-tuned by RL where rewards are available at every state. For fair comparison, OGTL is pre-trained by the observed data and $Q$ is calculated as $(Q)$.

The comparative results presented in Table 1 are the basis of the following: (1) The search based methods UCT and Alpha-Beta can improve demonstration policy clearly in balanced combats, while the improvement is very limited in unbalanced combats. It is difficult to search the effective policy from the extremely large game tree under real-time limitation (40ms). These methods can only search around the demonstration policy. Profile search select the heuristic without additional exploration, hence it can’t find effective cooperative policy because the used heuristics are both sub-optimal. (2) DQfQ suffer the poor performance because its learning policy is constrained to be close to the demonstration, while it limits the model when the demonstration is sub-optimal. DDPGfD and OGTL perform well in balanced combats, and suffer poorly in unbalanced combats because the cooperation is ignored during learning. (3) The proposed method outperforms other methods as well as the demonstration policy significantly in all testing scenarios. The advantage is more obvious in unbalanced combats. For example, in $m24v30$ with demonstration $\nu$, the win rates of other method are all bellow 0.10, while the proposed method can get 0.58. These advantages indicates the proposed method can significantly improve the demonstration and learn multi-
agent cooperation effectively.

| Scenarios     | m3v5 | m30v30 | m18v20 | m24v30 |
|---------------|------|--------|--------|--------|
| Metrics       | W    | R      | W      | R      | W      | R      |
| D<sub>c</sub> | 0.43 | -0.02  | 0.75   | 0.18   | 0.32   | -0.11  | 0.12   | -0.26  |
| UCT<sub>c</sub> | 0.93 | 0.06   | 0.88   | 0.13   | 0.40   | -0.14  | 0.13   | -0.25  |
| A-B<sub>c</sub> | 0.96 | 0.07   | 0.85   | 0.16   | 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  |
| DQQQ<sub>c</sub> | 0.54 | 0.01   | 0.81   | 0.23   | 0.37   | -0.10  | 0.14   | -0.20  |
| DDPGID<sub>c</sub> | 0.86 | 0.21   | 0.95   | 0.05   | 0.48   | -0.19  | 0.16   | -0.17  |
| OGTL<sub>c</sub> | 0.89 | 0.24   | 0.93   | 0.16   | 0.38   | -0.05  | 0.13   | -0.21  |
| Ours(O<sub>c</sub>) | 0.94 | 0.43   | 0.97   | 0.49   | 0.86   | 0.22   | 0.73   | 0.15   |
| Ours(H<sub>c</sub>) | 0.98 | 0.46   | 1.00   | 0.51   | 0.87   | 0.24   | 0.76   | 0.21   |
| D<sub>θ</sub> | 0.61 | 0.02   | 0.13   | -0.23  | 0.03   | -0.36  | 0.00   | -0.49  |
| UCT<sub>θ</sub> | 0.93 | 0.01   | 0.73   | 0.09   | 0.37   | -0.19  | 0.00   | -0.36  |
| A-B<sub>θ</sub> | 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  |
| DQQQ<sub>θ</sub> | 0.69 | 0.08   | 0.34   | -0.13  | 0.27   | -0.19  | 0.04   | -0.35  |
| DDPGID<sub>θ</sub> | 0.78 | 0.15   | 0.75   | 0.09   | 0.69   | 0.13   | 0.08   | -0.31  |
| OGTL<sub>θ</sub> | 0.81 | 0.14   | 0.73   | 0.15   | 0.76   | 0.19   | 0.06   | -0.38  |
| Ours(O<sub>θ</sub>) | 0.96 | 0.89   | 0.81   | 0.72   | 0.68   | 0.15   | 0.52   | 0.01   |
| Ours(H<sub>θ</sub>) | 0.99 | 0.48   | 0.84   | 0.26   | 0.71   | 0.16   | 0.58   | 0.03   |

Table 1: A comparison with other demonstration based approaches where “c” (“θ”) means the used demonstration policy is c (θ). (H) and (O) stand for the heuristic simulator and observed data respectively. “D” is the demonstration policy. The best result for the given scenario is in bold.

**Further Evaluation**

We perform ablation experiments to validate two key elements of the proposed method. First, the proposed method (6) is compared with two other MARL methods IQL (Tan 1993) and COMA (Foerster et al. 2018). Since they are both designed for the case in which the reward can be calculated at every state, the Q used in these methods are all calculated from (9) for fair comparison. As shown in Table 2, IQL performs well in balanced scenarios whereas poorly in unbalanced scenarios. The reason is that IQL learn each agent independently without effective cooperation. The win rate of COMA is fairly high in small balanced battle but apparently fell in large and unbalanced battles. Different with COMA, the proposed method learn decentralized policies according to Nash Equilibrium. The guidance of Nash Equilibrium reduce the multi-agent exploration space and leads to more effective and stable cooperation learning.

Then, the contribution of “online update of Q” is evaluated. In this experiment, the proposed method is compared with two ablation methods: “Ours<sup>D<sub>c</sub></sup>” and “Ours<sup>θ<sub>c</sub></sup>” where Q is replaced by Q<sup>D</sup> and Q<sup>θ</sup> respectively throughout the learning procedure. From the results in Table 3 it is evident that: (1) “Ours<sup>D<sub>c</sub></sup>” performs well, thus demonstrating the effectiveness of using sub-optimal demonstration. (2) The performance of “Ours<sup>θ<sub>c</sub></sup>” is very poor because the network cannot provide an effective Q when the rewards are sparse. (3) “Ours” outperforms “Ours<sup>D<sub>c</sub></sup>” and “Ours<sup>θ<sub>c</sub></sup>” in all cases which demonstrates (9) contribute positively.

To better understand the proposed method, here we give two intuitive examples to show what the proposed method have learned:

1. Cover and withdraw. Fig 2 is a typical battle scene from m18v20. We can find that the learned policy will keep most units engaged in combat to maximize the damage. Also, the healthy unit (B) will step forward to cover other ally units and the near death unit (A) will escape from the battlefield to keep alive. (2) Fire allocation to maximum team damage. The used demonstration policies focus fire to eliminate a enemy agent quickly, which may lead “over kill”. For example, coordinate all agents to attack a near death unit. In this case, most attacks will be invalid because the target will be eliminated by one attack. The ratio of the invalid attacks are counted to estimate this situation. In m30v30 scenario, the ratio of demonstration w is 43%, while it is only 6% to the learned policy. It shows the proposed method can allocate fire more effectively.

![Figure 2: A typical battle scene where the units with red hp bar are controlled by the learned policy and the green hp bar units are opponent units.](image367x277to510x359)

Table 2: A comparison with different multi-agent reinforcement learning methods where Q is calculated by (9) using the heuristic simulator.

| Scenarios     | m5v5 | m30v30 | m18v20 | m24v30 |
|---------------|------|--------|--------|--------|
| Metrics       | W    | R      | W      | R      | W      | R      | W      | R      |
| IQL<sup>θ</sup> | 0.89 | 0.28   | 0.88   | 0.33   | 0.69   | 0.08   | 0.47   | -0.05  |
| COMA<sup>θ</sup> | 0.99 | 0.41   | 0.77   | 0.18   | 0.61   | 0.17   | 0.26   | -0.11  |
| Ours<sup>θ</sup> | 0.98 | 0.46   | 1.00   | 0.51   | 0.87   | 0.24   | 0.76   | 0.21   |

Table 3: A comparison with different calculations of Q. (H) is used as demonstration.

The proposed method is evaluated on the combat of RTS game which has a high requirement for multi-agent cooperation. Extensive experimental results on various

---

Footnote: More demos will be released with the source codes.
ous combat scenarios demonstrate that the proposed method can learn the multi-agent cooperation effectively and it significantly outperforms many state-of-the-art demonstration based approaches.

References

[Busoniu, Babuska, and De Schutter 2008] Busoniu, L.; Babuska, R.; and De Schutter, B. 2008. A comprehensive survey of multi-agent reinforcement learning. IEEE Transactions on Systems Man and Cybernetics 38(2):156–172.
[Cao et al. 2012] Cao, Y.; Yu, W.; Ren, W.; and Chen, G. 2012. An overview of recent progress in the study of distributed multi-agent coordination. IEEE Transactions on Industrial Informatics 9(1):427–438.
[Chen and etc. 2016] Chen, Y. F., and etc. 2016. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In Proc. ICRA.
[Churchill and Buro 2013] Churchill, D.; and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in starcraft. In Proc. CIG, 1–8. IEEE.
[Churchill, Saffidine, and De Schutter 2008] Churchill, D.; Saffidine, A.; and De Schutter, B. 2008. A comprehensive survey of multi-agent reinforcement learning. IEEE Transactions on Systems Man and Cybernetics 38(2):156–172.
[Cao et al. 2012] Cao, Y.; Yu, W.; Ren, W.; and Chen, G. 2012. An overview of recent progress in the study of distributed multi-agent coordination. IEEE Transactions on Industrial Informatics 9(1):427–438.

[Lowe et al. 2017] Lowe, R.; WU, Y.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. In Proc. NIPS.
[Maas, Hannun, and Ng 2013] Maas, A. L.; Hannun, A. Y.; and Ng, A. Y. 2013. Rectifier nonlinearities improve neural network acoustics models. In Proc. ICML, volume 30, 3.
[Mnih et al. 2015] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. Nature 518(7540):529.
[Ng and Russell 2000] Ng, A. Y., and Russell, S. J. 2000. Algorithms for inverse reinforcement learning. In Proc. ICML.
[Omidshafiee et al. 2017] Omidshafiee, S.; Pasiz, J.; Amato, C.; How, J. P.; and Vian, J. 2017. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In Proc. ICML.
[Peng et al. 2017] Peng, P.; Yuan, Q.; Wen, Y.; Yang, Y.; Tang, Z.; Long, H.; and Wang, J. 2017. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. arXiv preprint arXiv:1703.10069.
[Rashid et al. 2018] Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Proc. ICML.
[Reddy et al. 2012] Reddy, T. S.; Gopikrishna, V.; Zaruba, G.; and Huber, M. 2012. Inverse reinforcement learning for decentralized non-cooperative multiagent systems. In Systems, Man, and Cybernetics.
[Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, I.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489.
[Silver et al. 2017] Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; and Bolton, A. 2017. Mastering the game of go without human knowledge. Nature 550(7676):364–369.
[Stone and Veloso 2000] Stone, P., and Veloso, M. 2000. Multiagent systems: A survey from a machine learning perspective. 345–383.
[Sukhbaatar and Fergus 2016] Sukhbaatar, S., and Fergus, R. 2016. Learning multiagent communication with backpropagation. In Proc. NIPS, 2244–2252.
[Sun et al. 2017] Sun, W.; Venkatraman, A.; Gordon, G. J.; Boots, B.; and Bagnell, J. A. 2017. Deeply AggreVaTeD: Differentiable imitation learning for sequential prediction. In Proc. ICML.
[Syed and Schapire 2008] Syed, U., and Schapire, R. E. 2008. A game-theoretic approach to apprenticeship learning. In Proc. NIPS.
[Syed, Bowling, and Schapire 2008] Syed, U.; Bowling, M.; and Schapire, R. E. 2008. Apprenticeship learning using linear programming. In Proc. ICML.
[Tampuu et al. 2017] Tampuu, A.; Matiisen, T.; Kodelja, D.; Kuzovkin, I.; Korjus, J.; Aru, J.; Aru, J.; and Vicente, R. 2017. Multiagent cooperation and competition with deep reinforcement learning. PloS one 12(4).
[Tan 1993] Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. Machine Learning Proceedings 330–337.
[Tuyls and Weiss 2012] Tuyls, K., and Weiss, G. 2012. Multiagent learning: Basics, challenges, and prospects. *Ai Magazine* 33(3):41–52.

[Usunier et al. 2016] Usunier, N.; Synnaeve, G.; Lin, Z.; and Chin-tala, S. 2016. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*.

[Vecerik et al. 2017] Vecerik, M.; Hester, T.; Scholz, J.; Wang, F.; Pietquin, O.; Piot, B.; Heess, N.; Roth′rl, T.; Lampe, T.; and Riedmiller, M. 2017. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv*.

[Wang and Klabjan 2018] Wang, X., and Klabjan, D. 2018. Competitive multi-agent inverse reinforcement learning with sub-optimal demonstrations. In *Proc. ICML*.

[Wang and Sang 2005] Wang, Y., and Sang, D. 2005. *Multi-agent framework for third party logistics in E-commerce*. Pergamon Press, Inc.

[Yang et al. 2018] Yang, Y.; Luo, R.; Li, M.; Zhou, M.; Zhang, W.; and Wang, J. 2018. Mean field multi-agent reinforcement learning. In *Proc. ICML*. 