Learning Deep Decentralized Policy Network by Collective Rewards for Real-Time Combat Game

Peixi Peng[1], **Junliang Xing**[1*], **Lili Cao**[1], **Lisen Mu**[2] and **Chang Huang**[2]
[1] Institute of Automation, Chinese Academy of Sciences
[2] Horizon Robotics
{peixi.peng, lili.cao}@ia.ac.cn, junliang.xing@nlpr.ia.ac.cn, {lisen.mu, chang.huang}@horizon.ai

Abstract

The task of real-time combat game is to coordinate multiple units to defeat their enemies controlled by the given opponent in a real-time combat scenario. It is difficult to design a high-level Artificial Intelligence (AI) program for such a task due to its extremely large state-action space and real-time requirements. This paper formulates this task as a collective decentralized partially observable Markov decision process, and designs a Deep Decentralized Policy Network (DDPN) to model the polices. To train DDPN effectively, a novel two-stage learning algorithm is proposed which combines imitation learning from opponent and reinforcement learning by no-regret dynamics. Extensive experimental results on various combat scenarios indicate that proposed method can defeat different opponent models and significantly outperforms many state-of-the-art approaches.

1 Introduction

The task of real-time (RT) combat game is to coordinate multiple units to defeat their enemies controlled by the given (or built-in) opponent in a real-time scenario. It is significantly challenging to play such a game due to its extremely large state-action space which grows exponentially with the number of the controlled units. In addition, expert-level labeled data are often unreliable or simply unavailable, and the time allowed for planning is on the order of milliseconds. Deep reinforcement learning (RL) [Li, 2018] proposes a promising way to this task. That is, the allied units are modeled as agents and controlled by deep neural networks, which are trained by exploring the action-state space and calculating rewards by playing with the opponent. Since the controlled units are multiple and perform actions simultaneously, hence it is naturally modeled the task as a multi-agent reinforcement learning (MARL) problem.

In an MARL system, the first issue is the *learning paradigm*. Since the joint state-action space of all agents is known and the rewards are collective, a direct method is the centralized learning [Usunier *et al.*, 2016]. However, the joint action space grows exponentially with the number of agents, and it is hard to model the network when the number of agents is large and uncertain. To cope with these complexities, it is often necessary to resort to decentralized policies, where each agent selects its own action conditioned only on its observation. To learn effective decentralized policies, some existing works [Hu *et al.*, 2018] are designed by single-agent action exploration and ignore multi-agent cooperation. These methods often fail to encourage individual agents to sacrifice themselves for team advantages. For example, in a combat, the healthy agents should go forward to draw enemy fire and cover other injured agents. This task is always failed when each agent is only driven by its individual rewards. Hence, we model the RT combat game as a collective decentralized partially observable Markov decision process (CDec-POMDP)[Nguyen *et al.*, 2018], and follow centralized training with decentralized execution framework [Oliehoek *et al.*, 2008; Foerster *et al.*, 2018; Rashid *et al.*, 2018; Lowe *et al.*, 2017; Nguyen *et al.*, 2018]. That is, multiple agents coordinate their behavior by acting in a decentralized way which is learned from collective rewards. To this aim, a Deep Decentralized Policy Network (DDPN) is proposed in this paper to model the decentralized policies and control ally units. The input of the DDPN is the feature of an agent's observation, and the output is its policy.

Another issue is the *learning algorithm*. The standard RL

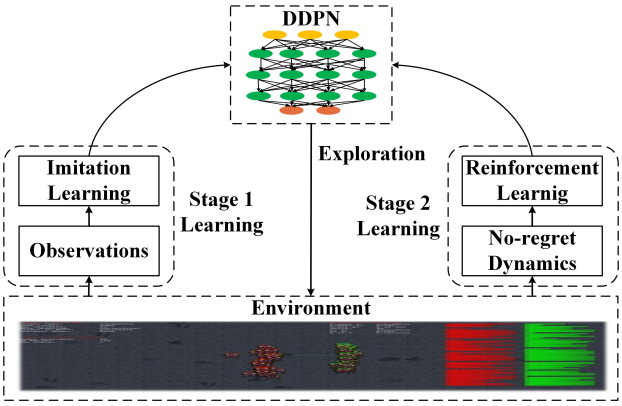

Figure 1: The framework of the proposed method.

*Contact Author. This work was supported by the Natural Science Foundation of China (NSFC) under Grants 61702515.

pipeline [Mnih *et al.*, 2015] trains the network from scratch by exploring the action-state space and calculating rewards. However, compared with the Atari games [Mnih *et al.*, 2015] and the MineCraft scenarios [Oh *et al.*, 2016], which both focus on the control of a single agent with a fixed and limited set of actions, the combat game is much more challenging because its action-state space is extremely large and the feasible action-states only occupy a very small portion of the whole space. Therefore, randomizing actions will cause the learning algorithm to reach sub-optimal plateaux with a large probability. A lot of exploration is required for learning [Usunier *et al.*, 2016]. Furthermore, randomizing actions often disorganize the agents, which leads rapidly to defeat without any effective feedback. In addition, it is hard to identify each agent action's own contribution from the collective rewards, and the optimal policy of one agent depends on other agents' policies, which make the learning unstable. To handle these difficulties, a novel learning algorithm is proposed as shown in Fig. 1, which contains the following two stages: 1) In the first stage, the ally units are controlled to explore the opponent's policy and observe the enemies' actions. After collecting several observations as training samples, the DDPN is pre-trained by online imitation learning. 2) Since our goal is to defeat the opponent rather than imitate it, the DDPN is further optimized in the second stage. The regret is used to evaluate policies and a no-regret dynamics algorithm is introduced to approximate the joint no-regret policies of multiple agents. Then, a novel RL algorithm is proposed based on the approximated no-regret policies, and we prove it is self-improved and can learn better policy from the first stage learning. These two stages of learning are complementary to each other. On one hand, the DDPN learned from stage 1 is not optimal and needs to be further optimized. On the other hand, the imitation learning can provide a good initial guidance of exploration and a reasonable starting point of RL.

In summary, the main contributions of this work are in three-fold: 1) we model the RT combat game task as CDec-POMDP and a staged learning algorithm is proposed to train the DDPN; 2) a self-improved RL algorithm is proposed based on no-regret dynamics to learn decentralized policies from collective rewards; and 3) extensive experimental results on various combat scenarios demonstrate that the proposed method can defeat different opponent models and significantly outperforms many state-of-the-art approaches.

2 Related Work

The field of AI has made significant progresses in the last decade due to the successes of deep learning [LeCun *et al.*, 2015] such as image classification [Russakovsky *et al.*, 2015] or playing various games including Go [Silver *et al.*, 2016] and the Atari video games [Mnih *et al.*, 2015]. However, it remains challenging to build high-level AI programs in complex environments. RT combat game is such a problem due to its large state-action space and lack of expert-level annotations. Recently, a few of works utilize the RT combat game to test AI algorithms and several experiment platforms [Churchill *et al.*, 2012; Tian *et al.*, 2017] have been opened.

The simplest approaches, which are widely used in most RT combat game AI bots, are hand-crafted heuristics which define static behaviors based on human prior knowledge. To obtain better solutions, a few of approaches [Churchill *et al.*, 2012; Churchill and Buro, 2013; Lelis, 2017; Moraes and Lelis, 2018] build game tree for current game states and aim to search the optimal solutions with help of heuristics. However, it is hard to search an effective policy online due to the real-time limitations. Inspired by the success of DQN [Mnih *et al.*, 2015], the other type of methods model the task as deep MARL problem and learn deep neural networks by centralized learning [Usunier *et al.*, 2016; Kong *et al.*, 2017], decentralized learning [Peng *et al.*, 2017; Hu *et al.*, 2018] or centralized learning with decentralized execution [Foerster *et al.*, 2018; Rashid *et al.*, 2018]. We also follow the later setting, and the main differences compared with [Foerster *et al.*, 2018; Rashid *et al.*, 2018] are: 1) They train the network from scratch, while the proposed DDPN is trained by staged learning. The imitation learning from opponent provides a good starting point of RL. 2) The no-regret dynamics algorithm is introduced to approximate the joint no-regret policies. We learn one agent's policy by assuming other agents are all performing actions as no-regret policies, which makes the learning more effective and stable.

The proposed method can be regarded as learning from demonstrations (LfD) [Schaal, 1997], and the demonstrations are provided by the opponent. LfD is widely-used to facilitate RL, and the direct method is to recover experts' strategies 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]. However, our task is to defeat the opponent rather than just imitate it. To learn better policies from demonstrations, a few of approaches [Hester *et al.*, 2018; Kang *et al.*, 2018] are proposed to explore the environment by LfD. These two methods are both designed for single agent tasks and try to find better policies by exploring around demonstrations. However, the joint state-action space of multiple agents is much larger and the exploration may be inefficient. Several existing works are also proposed by combining imitation learning and RL together [Silver *et al.*, 2016; Hu *et al.*, 2018]. AlphaGo [Silver *et al.*, 2016] is designed for two-agent zero-sum competitive task and cannot be applied to RT combat game directly. OGTL [Hu *et al.*, 2018] is designed for RT combat game and also starts learning by imitating the opponent. However, it relies on independent single-agent action exploration and ignores multi-agent cooperation. Different with OGTL, the DDPN is trained by collective rewards of joint actions and can learn cooperative polices more effectively.

3 Problem Definition

In RT combat game task, the enemy units and the opponent's strategy are modeled as a part of environment [Usunier *et al.*, 2016; Peng *et al.*, 2017; Foerster *et al.*, 2018; Kong *et al.*, 2017; Hu *et al.*, 2018]. Hence, we define the

task as collective decentralized partially observable Markov decision process (**CDec-POMDP**) [Nguyen *et al.*, 2018] described by a tuple $\langle \mathcal{S}, \mathcal{U}, \mathcal{A}, \mathcal{P}, \mathcal{O}, r, N, \lambda \rangle$ consisting of: **State space** $\mathcal{S}$ is a finite set of distinct states which can be visited in the game. An episode is defined as a set of game states in time series $\{S_1, S_2, ..., S_T\}$ where S_t is the game state at time t and only related to S_{t-1}. The terminal state S_T means all units of at least one player have been wiped out or T exceeds the time limitations. **Agents** $\mathcal{U} = \{u_1, ..., u_N\}$ where each agent corresponds to an ally unit and N may be varied at different states. **Action space** $\mathcal{A}$ is the finite discrete set of possible actions that can be performed for agents. **State transition function** $\mathcal{P}_t = \mathcal{P}(S_{t+1}|S_t, A)$ is induced by performing joint actions $A = \{a_i\}_{i=1}^N$ simultaneously at state S_t where a_i is performed by u_i. **Observation function** $\mathcal{O}(S) = \{O_i(S)\}_{i=1}^N$ where $O_i(S)$ is the observation of agent u_i at state S. The decentralized policy of u_i is defined as $\{p_i(a|O_i(s))\}_{a \in \mathcal{A}}$ corresponding to the probabilities of actions to perform. Although the joint state-action space is known to each agent, the observations are more robust and easy to model at game states where the numbers of units are large and uncertain. **Reward** $r(S_{t+1}, S_t, A)$ is a bounded function to measure the reward (or payoff) of the joint state-action, and it is defined as the sum of damage inflicted on the opponent units minus the damage taken, which is similar to previous works [Usunier *et al.*, 2016; Foerster *et al.*, 2018]. It is collective and shared by all agents. $\lambda \in [0, 1)$ is the discount factor.

Given any state $S \in \mathcal{S}$, initialize the game as $S_1 = S$. At each time step t, each agent u_i receives a state $S_t \in \mathcal{S}$, selects an action $a_i \in \mathcal{A}$ simultaneously, following joint policies $\pi = \{p_i\}_{i=1}^N$. All agents receive a global reward r_t, and the state is transited to S_{t+1} according to $\mathcal{P}$ until reaching a terminal state S_T. The state value $V^\pi(s)$ is calculated by the cumulative rewards with the discount factor 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. According to $V^\pi(S)$, 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 $S' \sim \mathcal{P}(S'|S, A)$. According to these definitions, we aims to find decentralized policies $\{p(a|O_i(S))\}_{a \in \mathcal{A}}$ for any agent u_i to defeat the opponent. We utilize the deep neural network parameterized by θ to model these policies, named Deep Decentralized Policy Network (DDPN). That is, for any agent (u_i) and the corresponding game state (S), the input of the DDPN is the feature of the unit's observation $f^o(u_i, S)$, and the output $\{p_\theta(a|f^o(u_i, S))\}_{a \in \mathcal{A}}$ is its decentralized policy. All units share same network parameters, and the individual policy relies on the input of DDPN.

4 Two-Stage Learning Algorithm

This section introduces the staged learning algorithm for the DDPN which is composed of imitation learning and reinforcement learning, as shown in Fig. 1.

Algorithm 1: Online imitation learning of DDPN.

Input: Initial DDPN parameters θ and data Buffer B.
Output: Updated DDPN parameters: θ
$Ep \leftarrow 1$.
while *Non-convergent* **do**
  Initialize the game state S randomly.
  while S *is non-terminal* **do**
    for $i = 1, ..., N$ **do**
      Extract the feature of its observation: $f^o(u_i, S)$, and calculate its policy $p_\theta\left(a|f^o\left(u_i, S\right)\right)$.
      Choose action $a_i \sim p_\theta\left(a|f^o\left(u, S\right)\right)$.
    for $u^e \in U^e$ **do**
      Extract the feature of its observation: $f^o(u^e, S)$, and observe its action a^e_u.
      Update B with $(f^o(u^e, S), a^e_u)$.
    Update S by performing $\{a_i\}_{i=1}^N$.
    if $Ep \mod E_{Update} = 0$ **then**
      Update θ using B by Eq. (3).
    $Ep \leftarrow Ep + 1$.

4.1 Stage 1: Imitation Learning

The opponent's policy is similar to a black box and its deterministic actions can be viewed without knowledge of its internal workings. In the first stage, the DDPN is trained by mimicking enemy units' actions. For any state $S \in \mathcal{S}$, we can observe the actions $\{a_{u^e}\}_{u^e \in U^e}$ performed by enemy units U^e. They are regarded as demonstrations to play the game, and DDPN is trained by minimizing the cross-entropy loss function:

$$\theta^* = \text{argmin}_\theta \sum\nolimits_{u^e \in U^e} -\log p_\theta(a_{u^e}|f^o\left(u^e, S\right)). \tag{3}$$

To learn DDPN efficiently, an imitation learning algorithm is proposed in Alg. 1 by online update. Alg. 1 regards the observations of enemy actions as demonstrations, and it still works when the demonstrations are human-designed heuristics or from experience replays.

4.2 Stage 2: Reinforcement Learning

Since our goal is to defeat the opponent rather than just imitate it, hence we aim to further improve the polices in the second stage. Consider the DDPN learned from the first stage as the initial policies $\pi^0 = \{p_i^0\}_{i=1,...,N}$, then $V^0(S)$ and $Q^0(S, A)$ can be calculated by simulating π^0 in Eq. (1) and Eq. (2) respectively. The regret value is introduced to represent how much the agents "regret" having used the policies $\pi = \{p_i\}_{i=1,...,N}$. It roughly evaluates the gap between the current policies and the optimal policies:

$$\mathrm{R}(\pi, S, M) = \frac{1}{M} \sum_{m=1}^{M} (\max_{A \in \mathcal{A}^N} Q^0(S, A) - Q^0(S, A^m)), \tag{4}$$

where M is the number of independent simulations, $A^m = \{a_i^m\}_{i=1,...,N}$ are joint actions and $a_i^m \sim p_i$. If the joint policies $\pi^* = \{p_i^*\}_{i=1,...,N}$ satisfy $\lim_{M \to +\infty} \mathrm{R}\left(\pi^*, S, M\right) = 0$, then π^* are the desired joint no-regret policies. As proved

Algorithm 2: The multiplicative weights algorithm.

Input: Agent u, game state S and normalized $Q^0(S,a) \in [0,1]$ for any $a \in \mathcal{A}$.
Output: $\{p^*(a|O(S))\}_{a\in\mathcal{A}}$
Initialize weight $w(a) = 1$ for each action.
while *Non-convergent* **do**
  Update $p(a|O(S)) = \frac{w(a)}{\sum_{a'\in\mathcal{A}} w(a')}$ for each action.
  Update the weight $w(a) \leftarrow w(a)\cdot(1+\eta Q^0(S,a))$ for each action (η is the update rate).
$p^*(a|O(S)) = \frac{w(a)}{\sum_{a'\in A} w(a')}$ for each action.

Algorithm 3: The no-regret dynamics algorithm

Input: The game state S, agents $u_1, ..., u_N$, and the initialized policies $\pi^0 = \{p_i^0\}_{i=1}^N$.
Output: $\{p_i^*(a|O_i(S))\}_{a\in\mathcal{A}}$ for each agent u_i.
Initialize $p_i^* = p_i^0$ for each agent u_i.
for $k = 1, 2, ..., K$ **do**
  for $i = 1, ..., N$ **do**
    Fix the polices of agents $\{u_j\}_{j\neq i}$ as $\{p_j^*\}_{j\neq i}$.
    Find the no-regret policy p_i^k for u_i by Alg. 2.
    Update $p_i^*(a|O_i(S)) = p_i^k(a|O_i(S))$.

in [Cesa-Bianchi *et al.*, 2007], the multiplicative weights (MW) algorithm (Alg. 2) converges to the no-regret policy in single-agent cases.

Unfortunately, as the joint state-action space grows exponentially with the number of agents and the no-regret policy of one agent depends on others' policies, it is difficult to solve the joint no-regret policies in multi-agent cases. To handle this challenge, Alg. 2 is extended to the no-regret dynamics algorithm (Alg. 3) to approximately estimate the joint no-regret policies $\{p_i^*\}_{i=1}^N$. Then, $\{p_i^*\}_{i=1}^N$ are further used to learn DDPN by Eq. (5) where η is the learning rate, and an asynchronous RL algorithm is proposed in Alg. 4.

$$\theta \leftarrow \theta + \eta \sum\nolimits_{a\in\mathcal{A}} \log p_\theta(a|f^o(u_i,S)) p_i^*(a|O_i(S)). \quad (5)$$

Here we theoretically analyse the effectiveness of the proposed algorithm. In each iteration of Alg. 3, each agent finds its no-regret policy by assuming other agents' policies are certain, and Alg. 3 is convergent as proven in [Roughgarden, 2016]. Although there is no theoretical guarantee that $\{p_i^*\}_{i=1}^N$ are joint no-regret policies, yet we can still prove that they are better policies than the initial policies π^0, that is $V^*(S) \geq V^0(S)$ for any state $S \in \mathcal{S}$ where $V^*(S)$ is simulated by performing $\{p_i^*\}_{i=1}^N$ in Eq. (1).

Proof: Suppose p_i^k is the policy of u_i after the k-th iteration of Alg. 3. Since Alg. 2 is no-regret for each single agent when other agents' polices are known, hence:

$$\begin{aligned}
&\mathbb{E}\left(Q^0(S, A\sim\textstyle\prod_{i=1}^N p_i^{k+1})\right) \\
&\geq \mathbb{E}\left(Q^0(S, a_N\sim p_N^k, \{a_i\}_{i=1}^{N-1}\sim\textstyle\prod_{i=1}^{N-1} p_i^{k+1})\right) \quad (6)\\
&\geq ... \geq \mathbb{E}\left(Q^0(S, A\sim\textstyle\prod_{i=1}^N p_i^k)\right),
\end{aligned}$$

where S is any game state. Consider a game initialized by $S_1 = S$, according to Eq. (6), we have:

$$\begin{aligned}
&V^0(S_1) = \mathbb{E}\left(Q^0(S_1, A\sim\textstyle\prod_{i=1}^N p_i^0)\right) \\
&\leq \mathbb{E}\left(Q^0(S_1, A\sim\textstyle\prod_{i=1}^N p_i^*)\right) \\
&\leq \mathbb{E}\left(r(S_2, S_1, A\sim\textstyle\prod_{i=1}^N p_i^*) + \lambda Q^0(S_2, A\sim\textstyle\prod_{i=1}^N p_i^*)\right) \\
&... \leq \mathbb{E}\left(\textstyle\sum_{t=1}^{T-1}\lambda^{t-1} r(S_{t+1}, S_t, A\sim\textstyle\prod_{i=1}^N p_i^*)\right) = V^*(S_1). \quad (7)
\end{aligned}$$

According to Eq. (7), Alg. 3 will find a local optimum where the starting point is the previous policy. In practice, most existing opponent policies are implemented by rule-based scripts and far from the local optimum. Hence, the proposed method can improve these policies significantly in practice. Furthermore, the learning policies are continually regarded as new initial policies of Alg. 3 during the learning procedure to improve constantly.

5 Experiments

The proposed method is tested using SparCraft [Churchill *et al.*, 2012], which is a simulator of the StarCraft local combat game and is widely adopted to test combat game algorithms [Churchill *et al.*, 2012; Churchill and Buro, 2013; Lelis, 2017; Moraes and Lelis, 2018]. It is chosen as the experimental platform because its efficient simulation function can accelerate the algorithms. In addition, unlike the StarCraft environment [Usunier *et al.*, 2016], the SparCraft implements several kinds of opponent models to better validate the robustness of the proposed method.

5.1 Experiment Settings

The action space contains three types of discrete actions: noop, move[$directions$] and attack[$enemy\_ids$], 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 state of the combat scenario. To an ally unit, [$enemy\_ids$] is the list of its attack targets, and arranged in ascending order of the remaining hit points. The feature of an agent's observation is concatenated of the properties [Foerster *et al.*, 2018] of the corresponding unit, the 10 closest ally units and 10 closest enemy units. To make sure the feature vector is unique and identical to each state, the ally and enemy units are arranged in ascending order of the distance to the agent. If the numbers of ally and enemy units are less than 10, then the feature vector is filled with 0 to make the length of feature vector identical at any state. DDPN is composed of 4 fully connected (FC) layers, 3 batch normalization layers following the first 3 FC layers respectively. A softmax layer is used to output the probabilities. The widths of the FC layers are 256, 128, 128, and $directions + enemy\_ids + 1$ respectively. The leaky rectified linear function [Maas *et al.*, 2013] is used as an activation function for the first 3 FC layers. We utilize this

Algorithm 4: RL algorithm of DDPN.

Input: DDPN parameters θ^0 learned by Alg. 1.
Output: Updated DDPN parameters θ.
Initialize: $\tilde{\theta} \leftarrow \theta^0, \theta \leftarrow \theta^0$.
$Ep = 1$.
while *Non-convergent* **do**
  Initialize game state S randomly.
  while S *is non-terminal* **do**
    for $i = 1, ..., N$ **do**
      Extract feature $f^o(u_i, S)$.
      Choose action $a_i \sim p_{\tilde{\theta}}(a|f^o(u_i, S))$.
    Calculate the joint no-regret policies by Alg. 3 which is initialized by $p_\theta(a|f^o(u_i, S))$.
    Update $\tilde{\theta}$ by Eq. (5).
    Update S by performing $\{a_i\}_{i=1}^N$.
  if $Ep \mod E_{Update} = 0$ **then**
    Update $\theta \leftarrow \tilde{\theta}$.
  $Ep \leftarrow Ep + 1$.

simple network architecture because it needs to forward the network in real time.

The opponents are the rule-based heuristics implemented in SparCraft, including: (1) **Attack-Value** (*v*) where units attack the unit with the highest value (damage per frame / current hit points) within range and a unit not within the range of any enemy moves toward the closest enemy. (2) **Attack-Closest** (*c*) which is similar to *v* except each unit attack the closest enemy unit. (3) **Kiter** (*k*) is similar to *c*, it instead moves a fixed distance away from the closest enemy when it is necessary to cool down the weapon. (4) **Kiter Value** (*kv*) will attack the enemy units as same as *v* if the weapon is ready to attack, and it will move as same to *k* otherwise. To summarize, *c* and *v* are two most straightforward and simplest heuristics who only focus on how to attack. The scripts *k* and *kv* additionally consider how to move when the weapon is not ready. The "move and attack" mode of *k* and *kv* is also known as "hit-and-run"[1] which is often used by professional players. In experiments, "DDPN_*x*" denotes the DDPN model trained by the opponent *x*. The Marine (m) and Zealot (z) are chosen as two typical unit types for ranged and melee units respectively. The test combat scenarios differ in the unit numbers and types: two small scale combats m5v5 and z5v5, a large-scale combat m30v30, a heterogeneous combat 5m5z where the ally and enemy units are both composed of 5 Marines and 5 Zealots, and two unbalanced combats m15v16 and m18v20 where we control 15(18) Marines against 16(20) Marines and our force is weaker than the opponent. Two circle regions in the combat area are chosen for ally and enemy units respectively, and each unit is born randomly in the corresponding region in each scenario. All of the win rates in the paper are accounted with 200 independent battles.

The decay factor λ is set to 0.995, and the iterations of Alg. 2 and Alg. 3 are both set to 5. To simulate Eq. (2) and Eq. (1) efficiently, actions with the maximum possibility are performed. In addition, $\eta = 0.6$ in Alg. 3 and DDPN is learned by SGD with learning rate 10^{-3}. The models are trained on GeForce GTX 1080 and tested on a desktop PC with one 2.4 GHz CPU and 8G RAM.

[1] http://tvtropes.org/pmwiki/pmwiki.php/Main/HitAndRunTactics.

Scenarios	m5v5	z5v5	m30v30	5m5z	m15v16	m18v20
UCT_*v*	0.95	0.91	0.83	0.89	0.72	0.41
A-B_*v*	0.97	0.90	0.78	**0.90**	0.66	0.39
DQN_*v*	0.95	0.89	0.86	0.66	0.37	0.15
PG_*v*	0.89	0.85	0.77	0.85	0.42	0.20
OGTL_*v*	**1.00**	**0.96**	0.92	0.76	0.86	0.45
GMEZO*	**1.00**	-	-	-	0.79	-
BicNet*	0.92	-	-	-	0.71	-
CommNet*	0.95	-	-	-	0.68	-
MS-MARL*	-	-	-	-	0.82	-
DDPN_*v*	0.99	0.95	**0.93**	0.85	**0.87**	**0.73**
UCT_*c*	0.93	0.86	0.85	0.84	0.69	0.36
A-B_*c*	**0.99**	0.90	0.79	0.89	0.70	0.32
DQN_*c*	0.97	0.92	0.74	0.63	0.35	0.11
PG_*c*	0.89	0.85	0.91	0.68	0.36	0.19
OGTL_*c*	0.94	**0.96**	0.89	0.76	0.80	0.51
DDPN_*c*	0.96	0.91	**1.00**	**0.92**	**0.84**	**0.79**
UCT_*k*	0.81	0.80	0.75	0.78	0.59	0.20
A-B_*k*	0.82	**0.85**	0.72	0.79	0.63	0.25
DQN_*k*	0.77	0.71	0.65	0.54	0.19	0.09
PG_*k*	0.76	**0.85**	0.73	0.58	0.26	0.10
OGTL_*k*	0.82	0.79	0.81	0.61	0.42	0.32
DDPN_*k*	**0.89**	0.84	**0.84**	**0.80**	**0.77**	**0.65**
UCT_*kv*	0.79	0.81	0.76	0.72	0.56	0.16
A-B_*kv*	0.80	0.83	0.75	0.74	0.55	0.16
DQN_*kv*	0.73	0.71	0.51	0.49	0.14	0.08
PG_*kv*	0.76	0.70	0.59	0.57	0.29	0.11
OGTL_*kv*	0.79	0.83	0.72	0.61	0.46	0.27
DDPN_*kv*	**0.88**	**0.85**	**0.91**	**0.78**	**0.76**	**0.61**

Table 1: Compared with other competing approaches, where "*" means the experiments are conducted on the other settings and "-" means the results are unavailable. The best result for the given scenario and opponent is in bold.

5.2 Comparative Results

We consider two type of methods as competing baselines including: 1) heuristic-based search methods, including UCT [Churchill and Buro, 2013] and Alpha-Beta (A-B) search [Churchill *et al.*, 2012] which have been implemented in SparCraft both use the opponent's policy to help search for fair comparison; and 2) deep learning based methods including two well-known general RL algorithms DQN [Mnih *et al.*, 2013] and PG [Williams, 1992], and 5 state-of-the-art deep RL algorithms designed for the RT combat game including OGTL [Hu *et al.*, 2018], GMEZO [Usunier *et al.*, 2016], BicNet [Peng *et al.*, 2017], CommNet [Sukhbaatar and Fergus, 2016] and MS-MARL [Kong *et al.*, 2017]. The DQN is re-implemented as centralized learning by the framework proposed in [Usunier *et al.*, 2016], and PG and OGTL are re-implemented in a decentralized way [Hu *et al.*, 2018] which utilizes the same feature and network architecture as us. The results of GMEZO, BicNet, CommNet and MS-MARL are reported in [Peng *et al.*, 2017; Kong *et al.*, 2017]. Due to different experimental settings, these comparative results are only indicative. As stated in [Peng *et al.*, 2017], their opponents have the similar effect to *w*, hence we compare these results with the DDPN_*w*.

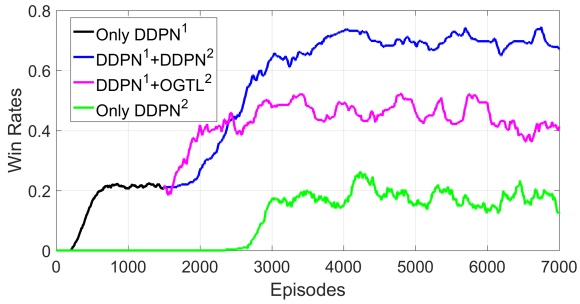

Figure 2: Win rates for DDPN and competing algorithms on $m18v20$ against with "v". "DDPN[1]+DDPN[2]" outperforms all baseline methods.

From the comparative results presented in Table 1, we observe the following key findings: 1) the win rates of our DDPN model are much larger than 0.5 in all combat scenarios, and outperforms other competing methods overall. It demonstrates the effectiveness of the proposed method. 2) UCT and Alpha-Beta can achieve high win rates in balanced combats, while the performances are very limited in unbalanced combats. It is difficult to search the effective cooperative policies from the extremely large game tree under real-time limitation (40ms). 3) DQN and PG perform well in small combats against easy opponent models (v and c), while the win rates decline dramatically in large scale combats against hard opponent models because it is hard to learn an effective network from scratch in an extremely large state-action space. 4) OGTL is also designed by combing imitation learning of opponent and RL. It performs well in balanced combats and performs poorly in unbalanced combats. The reason is that it ignores multi-agent cooperation during learning and the cooperation plays an important role in unbalanced combats. Different with OGTL, the proposed DDPN utilizes collective rewards and can learn more effective multi-agent cooperative polices.

5.3 Ablation Studies

To validate the contributions of the staged learning, two additional versions of the DDPN are evaluated in this experiment: "only DDPN[1]" which means the model is only trained by imitation learning proposed in Alg. 1, and "only DDPN[2]" which means we train the DDPN by Alg. 4 directly with random initialization. To better validate the contributions of Alg. 4, we compare the proposed method with the stage learning method OGTL as "DDPN[1]+OGTL[2]", where the network is pre-trained by Alg. 1 and further improved by the RL proposed in [Hu *et al.*, 2018]. Fig. 2 shows average win rates as a function of training episode for each method on m18v20 against with "w", and it is evident that: 1) "DDPN[1]" can learn primary policy while its win rate is low. 2) The performance of "Only DDPN[2]" is poor because it is difficult to effectively explore the large action-state space from scratch. 3) "DDPN[1]+DDPN[2]" outperforms "only DDPN[1]" and "only DDPN[2]" clearly and it indicates these two stages are indeed complementary to each other. In addition, the improvement is more significant than "DDPN[1]+OGTL[2]". The reason is that Alg. 4 utilizes collective rewards to learn multi-agent cooperative policies, while "OGTL[2]" only uses single-agent action explorations.

Scenarios	m5v5	m15v15	m18v18	m30v30	m18v20
DQN_v	0.73	0.48	0.40	0.05	0.01
GMEZO*	0.80	0.80	0.82	-	0.17
DDPN_v	**0.89**	**0.94**	**0.93**	**0.92**	**0.67**

Table 2: The comparison results on scenarios with different unit numbers. The models are all trained on m15v16. The "*" means the experiments are conducted on other platforms.

Opponents	Mean	c	v	k	kv
DDPN_c	0.86	0.93	0.95	0.79	0.78
DDPN_v	0.85	0.91	1.00	0.70	0.78
DDPN_k	0.93	1.00	1.00	0.84	0.89
DDPN_kv	0.93	1.00	1.00	0.82	0.91

Table 3: Win rates for the cross-opponent experiments on m30v30.

To evaluate the robustness to different combat scales, DDPN trained on one scenario is tested on other scenarios with a different number of units. As shown in Table 2, DDPN is more robust than the centralized learning methods GMEZO and DQN. The reason is that DDPN relies on observations of an agent, which is more robust than whole joint state-action space.

In addition, a cross-opponent experiment is conducted to validate the influence of various opponents. In particular, the DDPN model trained by one opponent will fight against to another one such as DDPN_c vs. k. The experiment is carried on m30v30 and the comparison results are shown in Table 3. DDPN not only defeats the opponent model who is used to train the network, but also is applicable to other opponents. For example, DDPN_c can defeat kv even c and kv are implemented by totally different rules. In addition, the effectiveness of the DDPN is positively correlated to the opponent. The mean win rates of k and kv are higher than c and v, where the former two heuristics are the improved versions of the latter two. It is reasonable because the DDPN is learned from the opponent and more effective opponent model leads to more effective DDPN.

6 Conclusion

This paper formulates the task of real-time combat game as a collective decentralized partially observable Markov decision process (CDec-POMDP), and designs a Deep Decentralized Policy Network (DDPN) to model the decentralized polices. To train DDPN effectively, a novel two-stage learning algorithm is proposed which first performs imitation learning from opponent and then self-improved reinforcement learning by no-regret dynamics. Experimental results in several scenarios demonstrate that the method can defeat different rule-based scripts and significantly outperforms the state-of-the-art deep learning based approaches.

References

[Cesa-Bianchi *et al.*, 2007] Nicolo Cesa-Bianchi, Yishay Mansour, and Gilles Stoltz. Improved second-order

bounds for prediction with expert advice. *Machine Learning*, 2007.

[Churchill and Buro, 2013] David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in starcraft. In *IEEE CIG*, 2013.

[Churchill *et al.*, 2012] David Churchill, Abdallah Saffidine, and Michael Buro. Fast heuristic search for rts game combat scenarios. In *AIIDE*, 2012.

[Foerster *et al.*, 2018] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI*, 2018.

[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*, 2018.

[Ho and Ermon, 2016] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NIPS*, 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 *IJCAI*, 2018.

[Kang *et al.*, 2018] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *ICML*, 2018.

[Kong *et al.*, 2017] Xiangyu Kong, Bo Xin, Fangchen Liu, and Yizhou Wang. Revisiting the master-slave architecture in multi-agent deep reinforcement learning. *arXiv*, 2017.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

[Lelis, 2017] Levi H. S. Lelis. Stratified strategy selection for unit control in real-time strategy games. In *IJCAI*, 2017.

[Li, 2018] Yuxi Li. Deep reinforcement learning. *arXiv preprint arXiv:1810.06339*, 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 *NIPS*. 2017.

[Maas *et al.*, 2013] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

[Mnih *et al.*, 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv*, 2013.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[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*, 2018.

[Nguyen *et al.*, 2018] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent rl with global rewards. In *NIPS*, 2018.

[Oh *et al.*, 2016] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. *arXiv*, 2016.

[Oliehoek *et al.*, 2008] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 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*, 2017.

[Pomerleau, 1991] D. 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 *ICML*, 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 *Aistats*, 2011.

[Roughgarden, 2016] Tim Roughgarden. *Twenty Lectures on Algorithmic Game Theory*, pages 230–247. Cambridge University Press, 1st edition, 2016.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[Schaal, 1997] Stefan Schaal. Learning from demonstration. In *NIPS*, 1997.

[Silver *et al.*, 2016] D Silver, A. Huang, C. J. Maddison, A Guez, L Sifre, den Driessche G Van, J Schrittwieser, I Antonoglou, V Panneershelvam, and M Lanctot. 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 *NIPS*, 2018.

[Sukhbaatar and Fergus, 2016] Sainbayar Sukhbaatar and Rob Fergus. Learning multiagent communication with backpropagation. In *NIPS*, 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 *ICML*, pages 3309–3318, 2017.

[Tian *et al.*, 2017] Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C. Lawrence Zitnick. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. In *NIPS*, 2017.

[Usunier *et al.*, 2016] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv*, 2016.

[Williams, 1992] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, volume 8, pages 229–256. Springer, 1992.