Knowledge-Guided Agent-Tactic-Aware Learning for StarCraft Micromanagement

Yue Hu[1,*], **Juntao Li**[1,*], **Xi Li**[1,†], **Gang Pan**[1], **Mingliang Xu**[2]
[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[2] Zhengzhou University, Zhengzhou, China
{huyue76, juntaoli, xilizju, gpan}@zju.edu.cn, iexumingliang@zzu.edu.cn

Abstract

As an important and challenging problem in artificial intelligence (AI) game playing, StarCraft micromanagement involves a dynamically adversarial game playing process with complex multi-agent control within a large action space. In this paper, we propose a novel knowledge-guided agent-tactic-aware learning scheme, that is, opponent-guided tactic learning (OGTL), to cope with this micromanagement problem. In principle, the proposed scheme takes a two-stage cascaded learning strategy which is capable of not only transferring the human tactic knowledge from the human-made opponent agents to our AI agents but also improving the adversarial ability. With the power of reinforcement learning, such a knowledge-guided agent-tactic-aware scheme has the ability to guide the AI agents to achieve a high winning-rate performance while accelerating the policy exploration process in a tactic-interpretable fashion. Experimental results demonstrate the effectiveness of the proposed scheme against the state-of-the-art approaches in several benchmark combat scenarios.

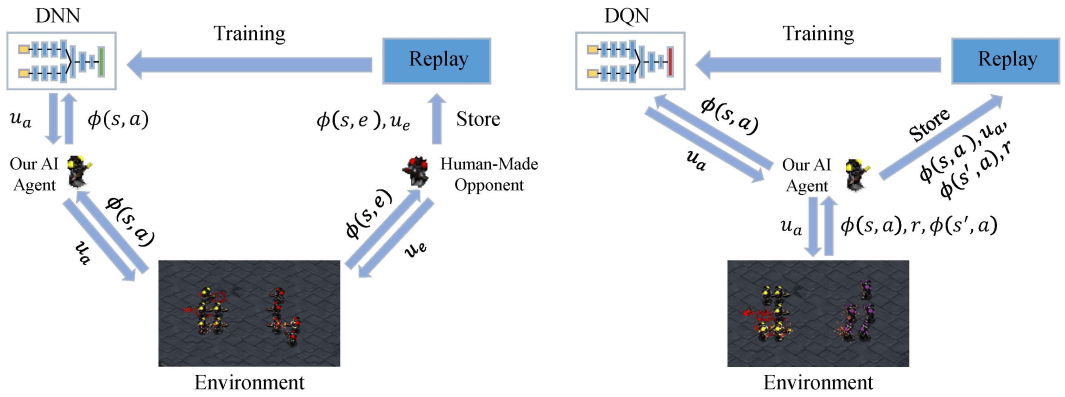
1 Introduction

In recent years, StarCraft micromanagement [Ontanón *et al.*, 2013; Wender and Watson, 2012; Synnaeve *et al.*, 2016] based on deep reinforcement learning (DRL) [Mnih *et al.*, 2015; Li, 2017; Silver *et al.*, 2016; Lillicrap *et al.*, 2015; Levine *et al.*, 2016; Mirowski *et al.*, 2017; Pasunuru and Bansal, 2017; Li *et al.*, 2017; Silver *et al.*, 2017] has attracted considerable attention in the fields of artificial intelligence and machine learning. Typically, this research problem is studied from three different perspectives: 1) learning architecture construction, 2) model training strategy and 3) multi-agent communication mechanism. For 1), a variety of deep learning architectures are proposed for effective reinforcement learning such as the centralized critic and decentralized actor network [Foerster *et al.*, 2017a; Lowe *et al.*, 2017] as well as the master-slave architecture [Kong *et al.*, 2017]. For 2), a large body of research work [Usunier *et al.*, 2017; Foerster *et al.*, 2017b] aims at designing feasible offline/online training strategies for improving the agent-specific learning performance (e.g., accuracy, convergence, stability, etc.). For 3), a variety of researchers [Foerster *et al.*, 2016; Sukhbaatar *et al.*, 2016; Peng *et al.*, 2017] focus on modeling the inter-agent interactions in terms of different message passing mechanisms. In principle, the aforementioned approaches rely heavily on pure data-driven exploration learning from scratch and are incapable of explicitly modeling the agent-tactic-aware learning process, resulting in the uninterpretable learning results. Moreover, the data-driven learning strategies for these approaches are usually carried out over the replay data produced by themselves in a batch-mode learning fashion, resulting in the loss of adversarial information against the opponents. Therefore, we mainly concentrate on constructing an interpretable agent-tactic-aware learning scheme that learns the adversarial knowledge from the opponent.

In the literature, conventional reinforcement learning approaches for StarCraft micromanagement are usually implemented under the circumstances of multi-agent game playing [Usunier *et al.*, 2017; Foerster *et al.*, 2017a; Peng *et al.*, 2017]. In essence, their primary goal is to build AI agents with effective and efficient policies, which are able to achieve the high winning-rate performance against the computer built-in opponent agents. Such policies are typically trained to pursue the reward maximum objective from the replay data, generated by agents through interacting with the environment. As a result, such a game playing pipeline relies heavily on bottom-up data-driven learning methodologies (lacking enough interpretability). In addition, this game playing pipeline pays insufficient attention to the power of effectively inheriting top-down human prior knowledge about game playing as well as explicitly modeling the adversarial tactics against the opponent agents and often treats the opponent agents as a part of the environment without the awareness of the opponent existence.

Motivated by the above observations, we propose a novel knowledge-guided agent-tactic-aware learning scheme: opponent-guided tactic learning (OGTL), which effectively

[*] Authors contributed equally to this work.
[†] Corresponding author.



(a) First stage: learning from the human-made opponent (b) Second stage: self-improving

Figure 1: Illustration of the whole architecture of the proposed approach: opponent-guided tactic learning (OGTL). In (a), we store the feature maps $\phi(s,e)$ and actions u_e into the replay and then we train the DNN using the stored data. After training, our AI agents can learn the opponent agents' tactic. In (b), our AI agents perform the action u_a when receiving the feature map $\phi(s,a)$ and then obtain the next feature map $\phi(s',a)$ and the reward r. With the interaction with the environment, we train our AI agents using the replay generated by our AI agents based on DQN.

leverages human knowledge to learn tactic-aware policy functions, making the learning process more interpretable. In the proposed learning scheme, we first build human-made opponent agents inheriting human knowledge about game playing with some particular pre-defined tactics. Subsequently, our AI agents are enabled to play against the human-made opponent agents and consequently learn their corresponding tactic-aware policy functions directly from the replay data of opponent agents. After that, our AI agents further refine the learned policy function leveraging deep reinforcement learning against the computer built-in opponent agents. Hence, our proposed scheme is based on two-stage cascaded learning, where the first stage is for tactic-aware knowledge transfer from the human-made opponent agents to our AI agents while the second stage is to reinforce the adversarial capabilities of our AI agents against the computer built-in opponent agents. In this way, our proposed scheme is explicitly tactic-aware with an interpretable policy exploration process of effectively learning the opponent's tactics, and has the capability of accelerating the policy exploration process because of introducing human knowledge compared with the conventional reinforcement learning approaches. Therefore, the main contributions of this work are two-fold:

- Under the StarCraft micromanagement settings, we propose a novel knowledge-guided agent-tactic-aware learning scheme that effectively leverages human tactic knowledge into the policy function learning process, which is able to accelerate the policy exploration process in a tactic-interpretable fashion.
- We propose a two-stage cascaded learning pipeline, which consists of tactic-aware knowledge transferred from the human-made opponent agents to our AI agents as well as adversarial capability reinforcement of our AI agents against the computer built-in opponent agents. With this learning pipeline, our AI agents can achieve a high winning-rate performance.

2 Our Approach

2.1 Problem Formulation

We focus on the micromanagement task, which consists of the combat environment, our units and opponent units. In our settings, we treat every unit as an agent. Without loss of generality, we assume there are n our AI agents and m opponent agents in the combat environment. Our AI agents are defined as $(a_1, a_2, ...a_n)$ and opponent agents are defined as $(e_1, e_2, ...e_m)$. The state obtained from the environment is denoted by $s \in S$, where S is the state space, shared among all the agents. We use $\phi(s,a)$ or $\phi(s,e)$ to denote the feature map which is extracted from the state in the specific agent viewpoint. The detail of feature map extraction will be explained in Section 2.2. When our AI agent receives the feature map $\phi(s,a)$, it performs the action $u_a \in U$ by its own policy $\pi_a : \phi(s,a) \rightarrow u_a$, where U is the action space. Similarly, the opponent agent performs the action $u_e \in U$ using policy $\pi_e : \phi(s,e) \rightarrow u_e$ when receiving the feature map $\phi(s,e)$.

Our method has two stages. In the first stage, we build human-made opponent agents with some particular predefined tactics. Thus, the human-made opponent has a predefined policy π_e. Our AI agents are able to play against the human-made opponent agents, transferring tactic-aware knowledge from the human-made opponent agents to ourselves. In this stage, we train a network $F(\phi(s,a);\theta_1)$ that gives us a probability distribution on the space of possible actions U, to learn the tactic-aware policy π_a^1, with which we choose the action with the highest probability. Here, θ_1 are the parameters of the network F.

In the second stage, to refine the learned policy π_a^1, we leverage deep reinforcement learning to learn a more sophisticated policy π_a^2 by combatting against the computer built-in opponent agents. In this stage, the whole game is considered as a stochastic game G [Nisan *et al.*, 2007; Shapley, 1953], defined by $G = \langle S, U, P, R, \gamma \rangle$. Here, P denotes the state transition function, R denotes the reward

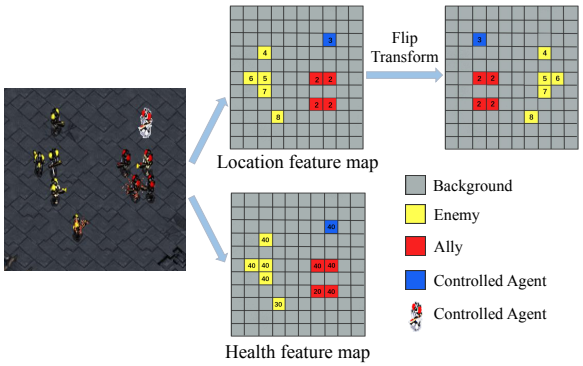


Figure 2: The left part is the real game screen. We map the game screen into two feature maps displayed in the mid part of the figure. In the location feature map, the background is encoded with number zero and other parts are encoded by the numbers displayed in the map. In the health feature map, we put the real value of the current health of each agent in the corresponding position. The right part is the processed location feature map after flip transformation.

function and $\gamma \in [0, 1)$ is a discount factor. We use a network $Q(\phi(s, a), u_a; \theta_2)$ that gives us a state-action value (evaluating the value of the current state and action) of possible actions, to learn the policy π_a^2, with which we choose the action with the highest value. Here θ_2 are the parameters of network Q initialized with θ_1. The whole architecture of the proposed approach is shown in Figure 1.

2.2 Feature Representation

Conventional studies like [Usunier *et al.*, 2017; Foerster *et al.*, 2017a; Kong *et al.*, 2017] used the parameterized features to represent game state s for StarCraft micromanagement tasks, which are based on the unit position, relative distance, unit health, unit type and other information. The parameterized features are encoded by game state parameters without explicit spatial information. However, the spatial information is crucial for the process of tactic learning, since many tactics are based on the specific spatial information. Therefore, we use a structured feature representation to express the spatial information explicitly.

We encode the game state s into two feature maps: location feature map and health feature map, which is displayed in Figure 2. The size of feature map can be equal to or scaled down at the same proportion to the game screen. In the location feature map, we use different numbers to indicate the currently controlled agent, allies, enemies and the background. In addition, the position of the number corresponds to the real position of agents in the game screen, so the position information of our AI agents and opponent agents can be obtained from the location feature map. In the health feature map, we put the current health value of the agents in the current position of the agents. This encoding process is denoted by $\phi(s, a)$ for our AI agents or $\phi(s, e)$ for opponent agents.

In our settings, feature maps encoded using the same game state s are different between our AI agents and opponent agents. Because in the opponent agents' feature map, the position of enemies is usually in the left part, different from our AI agents' feature map, where the position of enemies is usually in the right part. In our training process, we train our AI agents using the opponent replay data. To unify the position of enemies in feature maps of both opponent agents and our AI agents, we flip the opponent agents' feature map during the training process.

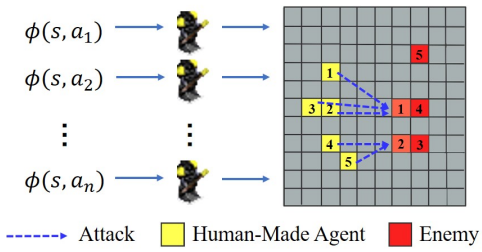


Figure 3: Illustration of the tactic: WC. The yellow grid denotes the human-made agent; the red grid denotes the enemy. In addition, the darkness of the color indicates the health value: the darker color marks higher health value. In the figure, agents No.1 to No.3 choose to attack the No.1 enemy, because No.1 enemy is the weakest and closest enemy. Similarly, the No.4 and No.5 agents attack the No.2 enemy.

The representation of the two feature maps and flip transformation is shown in Figure 2. With our feature maps, our AI agents can learn tactics with the specific position and health information. In the meantime, our AI agents can answer the following questions after training. Where am I? Where are my allies? Which enemy is the closest or the weakest?

2.3 Opponent-Guided Tactic Learning

Our knowledge-guided agent-tactic-aware learning scheme: opponent-guided tactic learning (OGTL) is a two-stage cascaded learning pipeline. The first stage is learning tactic-aware policy from the human-made opponent agents, and the second stage is a self-improving process.

First stage: learning from the human-made opponent

In this stage, our AI agents play against the human-made opponent agents to learn opponent's tactics. Firstly, we leverage human knowledge to build a script opponent to perform actions in predefined tactics, like attacking the closest or the weakest enemies. The illustration of the tactic, attacking the weakest and closest enemy (WC) is shown in Figure 3, with which agents will attack the weakest enemy, and when the health of two enemies is the same, agents choose the closest one.

Then, we train our AI agents to play against this human-made opponent agents. During the combat, the opponent agents will choose the action based on the current feature map and predefined tactic-aware policy π_e. Since the mapping from feature maps to actions can reflect the tactic, we collect those feature maps and actions in the replay D_1 for our agents to learn from.

To learn the tactic-aware policy π_e, we use deep neural network (DNN), which can be defined as $F(\phi(s, a); \theta_1)$, taking feature map $\phi(s, a)$ as the input. The output of the DNN is the probability distribution on the space of possible actions U. The loss function is defined as follows:

$$\mathcal{L}_1(\theta_1) = \mathbb{E}_s\big(\|\mathbf{u_e} - F(\phi(s, a); \theta_1)\|_2^2 \big) \quad (1)$$

where $\|\cdot\|_2$ denotes L2 norm, $\mathbf{u_e}$ is the one-hot label, encoding the opponent agent action and the $\phi(s,a)$ is flip transformed using $\phi(s,e)$ in the replay D_1, where the detail of flip transformation is explained in Section 2.2. After training, our AI agents learn a tactic-aware policy π_a^1, with which we choose the action with the highest probability.

The first stage's training process is an iterative process, during which our AI agents first play against the opponent agents to collect data and then we train our AI agents from the gathered data. Therefore, the training process is online and dynamic with the following two advantages: 1) our training data are collected during the combats, so we do not need to prepare a lot of data in advance. 2) during the training process, our AI agents are improving, so the data collected from the opponent agents are changing, which help our AI agents to learn the tactics under different circumstances and make our AI agents more adaptive.

After the first training stage, our AI agents have learned some tactics like attacking the weakest and closest enemy from the opponent agents. However, in this stage, our AI agents learn the predefined tactics without exploration, so we refine the learned policy with deep reinforcement learning.

Second stage: self-improving

In this stage, our AI agents play against the computer built-in opponent agents, and our goal is to improve the learned policy π_a^1. We apply the deep reinforcement learning algorithm deep Q-network (DQN) [Mnih *et al.*, 2015] to refine the tactics. The key goal of DQN is to learn the Q-function using the DNN. Here, Q-function is a state-action value function, used for choosing the appropriate action. Thus, our AI agents are trained to learn the policy π_a^2, with which we choose the action with the highest state-action value.

We use the first stage's learned model's weights θ_1 to initialize the DQN's weights θ_2. In this stage, the loss function is defined as follows:

$$\mathcal{L}_2(\theta_2) = \mathbb{E}_{s,u_a,r,s'}[(y^{DQN} - Q(\phi(s,a),u_a;\theta_2))^2] \quad (2)$$

$$y^{DQN} = r + \gamma \max_{u'_a} \hat{Q}(\phi(s',a),u'_a;\theta'_2) \quad (3)$$

where y^{DQN}, formulated in Eq. 3, is the Q-learning target; γ is the discount factor; $\hat{Q}$ is the target network, cloned from the network Q every C steps; θ_2 are the parameters of the network Q and θ'_2 are the parameters of the target network $\hat{Q}$; s' and u'_a are the next state and next action; r is the reward computed by our reward function. The overall algorithm is described in Algorithm 1.

Algorithm 1 Opponent-Guided Tactic Learning

```
Set learning-rate η = 0.001, collection number M = 4
Set train epochs E = 10, minibatch size B = 64
Set discount factor γ = 1, C = 100
Set T1 = 200, T2 = 500000, N = 10000
Initialize replay D1,D2 to capacity N
//First stage:
Initialize network weights θ1
for i = 0 to T1 do
    if i mod M ≠ 0 then
        while episode (combat) not ended do
            Observe the feature map φ(st, a)
            Execute action uat
            Obtain φ(st, e) and uet
            Store (φ(st, e),uet) in D1
    else
        for epoch = 0 to E do
            for step = 0 to N/B do
                Sample random minibatch of
                two-tuple (φ(sj, e),uej) from D1
                Compute L1(θ1) using Eq. 1
                θ1 ← θ1 + η ∂/∂θ1 L1(θ1)
//Second stage:
Initialize network weights θ2 with θ1
for i = 0 to T2 do
    while episode (combat) not ended do
        Observe the feature map φ(st, a)
        Execute action uat
        Store (φ(st, a),uat,rt,φ(st+1, a)) in D2
    Sample random minibatch of
    four-tuple (φ(sj, a),uaj,rj,φ(sj+1, a)) from D2
    Compute L2(θ2) using Eq. 2
    θ2 ← θ2 + η ∂/∂θ2 L2(θ2)
    Every C steps reset Q̂ = Q
```

2.4 Analysis of Learned Tactics

Our proposed method OGTL is a knowledge-guided agent-tactic-aware learning scheme, and the whole training process is interpretable. Therefore, after two stages' training, our AI agents have learned the tactics preserved some of human-made opponent agents' tactics. We take attacking the weakest and closest tactic, for example, to analyze the learned tactics with our feature maps. In Figure 4, we compare the policy learned after the first stage and the second stage.

In Figure 4(a), agents No.1 to No.3 are attacking the No.2 enemy, the weakest and closest enemy to them. Similarly, agents No.4 and No.5 are attacking the No.4 enemy. Thus, our AI agents have learned a tactic-aware policy, attacking the weakest and closest enemy. Furthermore, we leverage deep reinforcement learning to refine our agents' learned policy. After the second stage, in Figure 4(b), our AI agents' policy has improved, with which our AI agents focus fire on No.2 enemy although No.2 enemy is not the weakest and closest enemy to agents No.4 and No.5. Because in this scenario, focus fire on the weakest and closest enemy to them all can achieve higher winning rate than WC.

3 Experiments

3.1 Experimental Setup

To evaluate our proposed method, we conducted experiments on the StarCraft platform. In the experiment, our method controls one group of agents to defeat the other group of agents controlled by the computer build-in opponent. We perform our learning method based on DQN compared with ZO [Usunier *et al.*, 2017], BiCNet [Peng *et al.*, 2017] and CommNet [Sukhbaatar *et al.*, 2016], all of which perform

Figure 4: Analysis of tactics: we add health bar to display the health of agents. After training, our AI agents play against computer built-in AI agents. (a) the learned tactic after the first stage and (b) the improved tactic after the second stage.

well in the StarCraft micromanagement task. In addition, our method is compared with the WC, the human-made script. Following the similar experiment setup as [Usunier *et al.*, 2017], we build several micromanagement scenarios, like m5v5, w15v17 and w18v20. M5v5 means our AI controls 5 agents, the computer built-in opponent controls 5 agents and the unit type is marine. W15v17 means our AI controls 15 agents, the computer built-in opponent controls 17 agents, and the unit type is wraith. The w18v20 scenario only changes the number of units compared with w15v17.

3.2 Baselines

Before conducting the experiment, we simply introduce the baselines. 1) DQN combines Q-learning with DNN to learn a Q-function. 2) ZO is based on greedy MDP, making agents aware of each other's next action and adding the episodic noises when performing the exploration. 3) BiCNet adopts the actor-critic architecture combining with the bidirectional RNN [Schuster and Paliwal, 1997] to encourage the agents to communicate. 4) CommNet leverages a message channel to implement the communication mechanism between agents. 5) Attack the weakest and closest enemy (WC) means the human-made script controls the agents to attack the weakest enemy and when the health of two enemies is the same, WC chooses the closest one.

3.3 Implementation Details

State feature maps

Different from the conventional parameterized features, we use two feature maps, health and location, to represent the game state. The real game screen size is 256x256, which is very sparse and needs lots of resources to process. To solve this problem, we map the whole game into a map, sized 85x85. In the health feature map, we put the current health value of the agents in the current position of the agents. In the location feature map, we use number 3 to denote the currently controlled agent and number 2 to indicate its allies, so that the network can learn about the difference between the currently controlled agent and its allies. To distinguish different enemies, we sort the enemies using their ID value in ascending order and the $index$ is the order of each enemy. We use the $index + 4$ to indicate the different enemies in the location feature map.

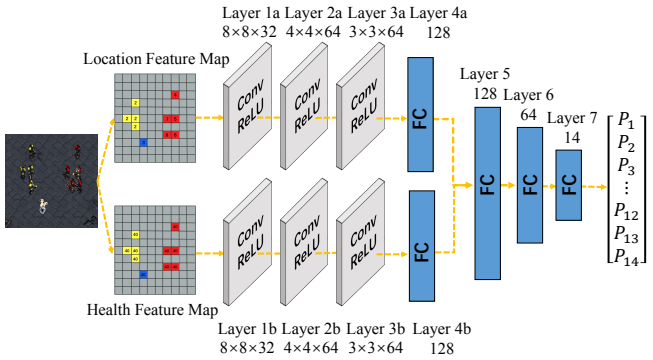


Figure 5: Illustration of the network architecture

Action definition

The definition of action space is similar to [Usunier *et al.*, 2017]. Each agent can move to eight directions, hold the position and attack the specific enemy. Therefore, the number of action space is nine plus the number of opponent's agents. Apart from this, since our action space has attack actions binding to specific enemies, when one enemy dies, we should not choose that action. Therefore, when the chosen action is invalid, we will choose the second best action and iterate this process until we find a valid action.

Reward definition

Our reward function is based on the health change and kill bonus. The value of kill bonus is 100, a large value to encourage the agents to kill the low-health enemy first. In this way, our agents are able to learn to attack the low-health enemy much more easily.

Model architecture

We describe our network architecture implementation, displayed in Figure 5. In both training stages, we use the same network architecture with different losses. And all of our AI agents share the same network. The network is implemented using full-connected and convolutional layers to learn the tactics and improve the learned tactics. Since our state is constructed with two feature maps, the network is built to process these two feature maps at the same time and then concatenate the two extracted features to make the final decision. Layers 1 to 3 are convolutional layers with the ReLU activation function. Layers 4 to 6 are full-connected layers, with the ReLU activation function. Layer 7 is the output layer with action size hidden units, and the activation function of this layer is softmax. The output of the network is the probability distribution on the space of possible actions U.

Training process

In the first stage, we launch two games, one for our AI agents and the other for human-made opponent agents in the training scenario, where the number of our AI agents and opponent agents is the same. We train our AI agents to play against the opponent agents. We collect the opponent feature map $\phi(s, e)$ and action u_e in the replay D_1 for every M episodes and each

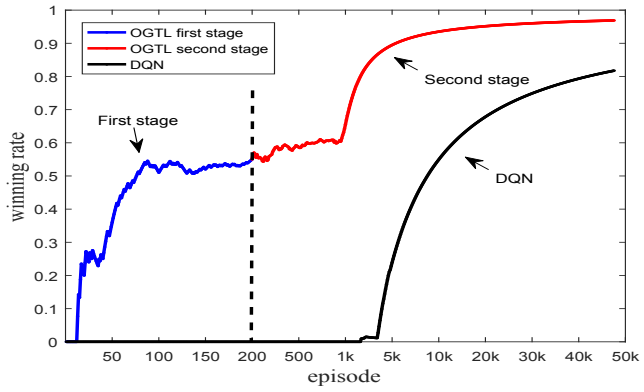


Figure 6: Average winning rate for OGTL and DQN on m5v5. The blue line is the first stage's average learning curve of OGTL, and the red line is the second stage's average learning curve of OGTL. The black line is the DQN's average learning curve.

time after collecting, we train the network $F(\phi(s,a);\theta_1)$ for E epochs. Every time after training, our agents perform better and are more aware of opponent's tactics. When our AI agents learn well enough, the first stage is over. In the second stage, we use the learned model to play against computer built-in opponent agents to improve the learned tactics using DQN. In this stage, we can train the DQN model, initialized by the weights of learned model F, in symmetrical or asymmetrical scenarios, where the number of opponent agents and the number of our AI agents can be different (the number of opponent agents is the same to that in the first training scenario). The detailed training process is displayed in Algorithm 1.

3.4 Results

In the experiment, our AI agents learn from the WC tactic in the first stage. Figure 6 shows the average winning rate of OGTL and DQN on the scenario m5v5. We train 100 models for testing and record the average wining rate learning curve. For DQN and the second stage of OGTL, we freeze the network for every 500 episodes and evaluate the learned model for 100 episodes per method. For the first stage of OGTL, we store the models for every 10 episodes and test models in m5v5 scenario playing against the computer built-in opponent agents.

From Figure 6, we can see that the first stage training process is quick. When the first stage is over, the model can achieve about 50 percent winning rate. The red line in Figure 6 denotes the second stage training process. In the second stage , our model can achieve 90 percent winning rate after only 5000 episodes, which is much faster than the DQN. In the meantime, the final winning rate is almost the same as DQN's and the learned tactics are preserved.

For all the results that we present in Table 1, we run models using the deterministic policy over 1000 episodes. OGTL learns from the opponent with WC tactic in the first stage. And the data in ZO, BiCNet, CommNet are directly quoted from their papers. The settings of ZO are almost the same as ours. Both BiCNet and CommNet use the local view while our settings use the full view.

Combat	OGTL	DQN	ZO	BiCNet	CommNet	WC
m5v5	0.96	0.99	**1.00**	0.92	0.95	0.76
w15v17	**0.74**	0.16	0.49	0.53	0.47	0.19
w18v20	**0.80**	0.31	0.76	_	_	0.22

Table 1: Winning rates of different methods on the training scenarios. The best result for a given map is in bold and the blank (_) result means the method is not tested on that scenario in their paper.

From the data in Table 1, our method OGTL outperforms these baselines in most of scenarios. In scenario m5v5, although ZO and DQN perform better than OGTL, OGTL can achieve 90 percent winning rate after only 5000 episodes, displayed in Figure 6, which is much faster than both of them. In both w15v17 and w18v20 scenarios, our method outperforms all the other baseline algorithms. And in the w15v17 scenario, the second best baseline is BiCNet with 53 percent winning rate, while OGTL has 74 percent winning rate, much higher. In addition, in the latter two scenarios (w15v17 and w18v20), our method learns from the opponent with the WC tactic in the first stage and uses DQN in the second stage, but has a much higher winning rate than both of them.

From the tactic perspective, these algorithms perform quite well in scenario m5v5. Both DQN and our method learn to focus fire, the difference is that our method often firstly eliminates the weakest and closest enemy, which is similar to the predefined tactics. In scenario w15v17, our method outperforms all these other methods. In this scenario, since the cooldown time of wraith weapon is very long, the key point to win the combat is to learn a balance between focusing the fire and splitting the fire. Our method first learns tactics about attacking the weakest and closest enemy in the first stage and then improves the tactics to learn the balance point in the second stage. The scenario w18v20 is very similar to w15v17.

4 Conclusion

In this paper, we have proposed a novel knowledge-guided agent-tactic-aware learning scheme applied in the StarCraft micromanagement task. Opponent-guided tactic learning scheme is a two-stage cascaded learning pipeline. By learning from the opponent, our AI agents have transferred the tactic-aware knowledge from the human-made opponent agents to ourselves in the first stage. In the second stage, our AI agents have refined the adversarial capabilities by combating with the computer built-in opponent agents. Through OGTL, our AI agents can get a better result in less training time compared with traditional reinforcement learning algorithms. The experimental results have demonstrated that the proposed learning scheme can achieve higher winning rate in some scenes of the StarCraft micromanagement tasks than the current state-of-the-art methods.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants (U1509206, 61472353, and 61751209), in part by the National Basic Research Program of China under Grant Grant 2015CB352302.

References

[Foerster *et al.*, 2016] Jakob Foerster, Yannis Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[Foerster *et al.*, 2017a] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.

[Foerster *et al.*, 2017b] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Philip Torr, Pushmeet Kohli, Shimon Whiteson, et al. Stabilising experience replay for deep multi-agent reinforcement learning. *International Conference on Machine Learning*, 2017.

[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 preprint arXiv:1712.07305*, 2017.

[Levine *et al.*, 2016] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[Li *et al.*, 2017] Xuijun Li, Yun-Nung Chen, Lihong Li, and Jianfeng Gao. End-to-end task-completion neural dialogue systems. *arXiv preprint arXiv:1703.01008*, 2017.

[Li, 2017] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[Lillicrap *et al.*, 2015] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[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 6382–6393, 2017.

[Mirowski *et al.*, 2017] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *International Conference on Learning Representations*, 2017.

[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*, 518(7540):529–533, 2015.

[Nisan *et al.*, 2007] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. *Algorithmic game theory*, volume 1. Cambridge University Press Cambridge, 2007.

[Ontanón *et al.*, 2013] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

[Pasunuru and Bansal, 2017] Ramakanth Pasunuru and Mohit Bansal. Reinforced video captioning with entailment rewards. *arXiv preprint arXiv:1708.02300*, 2017.

[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.

[Schuster and Paliwal, 1997] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[Shapley, 1953] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[Sukhbaatar *et al.*, 2016] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[Synnaeve *et al.*, 2016] Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.

[Usunier *et al.*, 2017] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *International Conference on Learning Representations*, 2017.

[Wender and Watson, 2012] Stefan Wender and Ian Watson. Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar. In *Computational Intelligence and Games, 2012 IEEE Conference on*, pages 402–408. IEEE, 2012.