A Deep Policy Inference Q-Network for Multi-Agent Systems

Zhang-Wei Hong*, Shih-Yang Su*, Tzu-Yun Shann*, Yi-Hsiang Chang, Chun-Yi Lee
National Tsing Hua University
{williamd4112, at7788546, arielshann, shawn420}@gapp.nthu.edu.tw, cylee@cs.nthu.edu.tw

ABSTRACT

We present DPIQN, a deep policy inference Q-network that targets multi-agent systems composed of controllable agents, collaborators, and opponents that interact with each other. We focus on one challenging issue in such systems—modeling agents with varying strategies—and propose to employ "policy features" learned from raw observations (e.g., raw images) of collaborators and opponents by inferring their policies. DPIQN incorporates the learned policy features as a hidden vector into its own deep Q-network (DQN), such that it is able to predict better Q values for the controllable agents than the state-of-the-art deep reinforcement learning models. We further propose an enhanced version of DPIQN, called deep recurrent policy inference Q-network (DRPIQN), for handling partial observability. Both DPIQN and DRPIQN are trained by an adaptive training procedure, which adjusts the network's attention to learn the policy features and its own Q-values at different phases of the training process. We present a comprehensive analysis of DPIQN and DRPIQN, and highlight their effectiveness and generalizability in various multi-agent settings. Our models are evaluated in a classic soccer game involving both competitive and collaborative scenarios. Experimental results performed on 1 vs. 1 and 2 vs. 2 games show that DPIQN and DRPIQN demonstrate superior performance to the baseline DQN and deep recurrent Q-network (DRQN) models. We also explore scenarios in which collaborators or opponents dynamically change their policies, and show that DPIQN and DRPIQN do lead to better overall performance in terms of stability and mean scores.

KEYWORDS

Deep Reinforcement Learning; Opponent Modeling; Multi-agent Learning

1 INTRODUCTION

Modeling and exploiting other agents' behaviors in a multi-agent system (MAS) have received much attention in the past decade [4, 12, 16, 23, 24]. In such a system, agents share a common environment, where they can act and interact independently in order to achieve their own objectives. The environment perceived by each agent, however, changes over time due to the actions exerted by the others, causing non-stationarity in each agent's observations. A non-stationary environment prohibits an agent from assuming that the others have specific strategies and are stationary, leading to increased complexity and difficulty in modeling their behaviors. In collaborative or competitive scenarios, in which agents are required to cooperate or take actions against the others, modeling environmental dynamics becomes even more challenging. In order to act optimally under such scenarios, an agent needs to predict other agents' policies and infer their intentions. For scenarios in which the other agents' policies dynamically change over time, an agent's policy also needs to change accordingly. This further necessitates a robust methodology to model collaborators or opponents in an MAS.

There is a significant body of work on MAS. The literature contains numerous studies of MAS modeling [8, 11, 28] and investigations of non-stationarity issues [17]. Most of previous researches on opponent or collaborator modeling, however, were domain-specific. Such models either assume rule-based agents with substantial knowledge of environments [3, 5], or exclusively focus on one type of applications such as poker and real-time strategy games [7, 30]. A number of early RL-based multi-agent algorithms have been proposed [6, 9, 10, 13, 19, 33]. Some researchers presumed that the agent possesses a priori knowledge of some portions of the environment to ensure convergence [9]. Techniques presumed that the agent knows the underlying MAS structures [6, 10, 13] have been explored. The use of other agents' actions or received rewards were suggested in [13, 19]. These assumptions are unlikely to hold in practical scenarios where an agent has no access to such information. While approaches for eliminating the need of prior knowledge of the environment have been attempted [2, 23, 25, 37], they are still limited to simple grid-world settings, and are unable to be scaled to more complex environments.

In recent years, a special field called deep reinforcement learning (DRL), which combines RL and deep neural networks (DNNs), has shown great successes in a wide variety of single-agent stationary settings, including Atari games [15, 27], robot navigation [38], and Go [32]. These advances lead researchers to start extending DRL to the multi-agent domain, such as investigating multi-agents' social behaviors [22, 35] and developing algorithms for improving the training efficiency [14, 16, 24]. Recently, representation learning in the form of auxiliary tasks has been employed in several DRL methods [20, 26, 29, 31]. Auxiliary tasks are combined with DRL by learning additional goals [20, 26, 31]. As auxiliary tasks provide DRL agents much richer feature representations than traditional methods, they are potentially more suitable for modeling non-stationary collaborators and opponents in an MAS.

In the light of the above issues, we first present a detailed design of the deep policy inference Q-network (DPIQN), which aims at training and controlling a single agent to interact with the other agents in an MAS, using only high-dimensional raw observations

---

(e.g., images). DPIQN is built on top of the famous deep Q-network (DQN) [27], and consists of three major parts: a feature extraction module, a Q-value learning module, and an auxiliary policy feature learning module. The former two modules are responsible for learning the Q values, while the latter module focuses on learning a hidden representation from the other agents' policies. We call the learned hidden representation "policy features", and propose to incorporate them into the Q-value learning module to derive better Q values. We further propose an enhanced version of DPIQN, called deep recurrent policy inference Q-network (DRPIQN), for handling partial observability resulting from the difficulty to directly deduce or infer the other agents' intentions from only a few observations [15]. DRPIQN differs from DPIQN in that it incorporates additional recurrent units into the DPIQN model. Both DPIQN and DRPIQN encourage an agent to exploit idiosyncrasies of its opponents or collaborators, and assume that no priori domain knowledge is given. It should be noted that in the most related work [16], the authors trained their agent to play a two-player game using handcrafted features and fixed the opponent's policy in an episode, which is not practical in real world environments.

To demonstrate the effectiveness and generalizability of DPIQN and DRPIQN, we evaluate our models on a classic soccer game environment [12, 23, 36]. We jointly train the Q-value learning module and the auxiliary policy feature learning module at the same time, rather than separately training them with domain-specific knowledge. Both DPIQN and DRPIQN are trained by an adaptive training procedure, which adjusts the network's attention to learn the policy features and its own Q-values at different phases of the training process. We present experimental results in two representative scenarios: 1 vs. 1 and 2 vs. 2 games. We show that DPIQN and DRPIQN are much superior to the baseline DRL models in various settings, and are scalable to larger and more complex environments. We further demonstrate that our models are generalizable to environments with unfamiliar collaborator or opponents. Both DPIQN and DRPIQN are able to change their strategies in response to the other agents' moves.

The contributions of this work are as follows:

- DPIQN and DRPIQN enable an agent to collaborate or compete with the others in an MAS by using only high-dimensional raw observations.
- DPIQN and DRPIQN incorporate policy features into their Q-value learning module, allowing them to derive better Q values in an MAS than the other DRL models.
- An adaptive loss function is used to stabilize the learning curves of DPIQN and DRPIQN.
- Unlike the previous works [7, 16, 36] only focusing on competitive environments, our models are capable of handling both competitive and collaborative environments.
- Our models are generalizable to unfamiliar collaborators or opponents.

The remainder of this paper is organized as follows. Section 2 introduces background materials related to this paper. Section 3 describes the proposed DPIQN and DRPIQN models, as well as the training and generalization methodologies. Section 4 presents our experimental results, and provides a comprehensive analysis and evaluation of our models. Section 5 concludes this paper.

2 BACKGROUND

RL is a technique for an agent to learn which action to take in each of the possible states of an environment $\mathcal{E}$. The goal of the agent is to maximize its accumulated long-term rewards over discrete time steps [23, 34]. The environment $\mathcal{E}$ is usually formulated as a Markov decision process (MDP), represented as a 5-tuple $(s, a, \mathcal{T}, \mathcal{R}, \gamma)$. At each timestep, the agent observes a state $s \in \mathcal{S}$, where $\mathcal{S}$ is the state space of $\mathcal{E}$. It then performs an action a from the action space $\mathcal{A}$, receives a real-valued scalar reward r from $\mathcal{E}$, and moves to the next state $s' \in \mathcal{S}$. The agent's behavior is defined by a policy π, which specifies the selection probabilities over actions for each state. The reward r and the next state s' can be derived by $r = \mathcal{R}(s, a, s')$ and $\mathcal{T}(s', s, a) = \Pr(s'|s, a)$, where $\mathcal{R}$ and $\mathcal{T}$ are the reward function and the transition probability function, respectively. Both $\mathcal{R}$ and $\mathcal{T}$ are determined by $\mathcal{E}$. The goal of the RL agent is to find a policy π which maximizes the expected return G_t, which is the discounted sum of rewards given by $G_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau$, where T is the timestep when an episode ends, t denotes the current timestep, $\gamma \in [0, 1]$ is the discount factor, and r_τ is the reward received at timestep τ. The action-value function (abbreviated as Q-function) of a given policy π is defined as the expected return starting from a state-action pair (s, a), expressed as $Q^\pi(s, a) = \mathbb{E}[G_t|s_t = s, a_t = a, \pi]$.

The optimal Q-function $Q^*(s, a)$, which provides the maximum action values for all states, is determined by the *Bellman optimality equation* [34]:

$$Q^*(s, a) = \sum_{s'} \mathcal{T}(s', s, a)[r + \gamma \max_{a'} Q^*(s', a')] \tag{1}$$

where a' is the action to be selected in state s'. An optimal policy π^* is then derived from Eq. (1) by selecting the highest-valued action in each state, and can be expressed as $\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a)$.

2.1 Deep Q-Network

DQN [27] is a model-free approach to RL based on DNNs for estimating the Q-function over high-dimensional and complex state space. DQN is parameterized by a set of network weights θ, which can be updated by a variety of RL algorithms [15, 27]. To approximate the optimal Q-function given a policy π and state-action pairs (s, a), DQN incrementally updates its set of parameters θ such that $Q^*(s, a) \approx Q(s, a, \theta)$.

The parameters θ are learned by gradient descent which iteratively minimizes the loss function $L(\theta)$ using samples (s, a, r, s') drawn from an experience replay memory Z. $L(\theta)$ is expressed as:

$$L(\theta) = \mathbb{E}_{s, a, r, s' \sim U(Z)}[(y - Q(s, a, \theta))^2] \tag{2}$$

where $y = r + \gamma \max_{a'} Q(s', a', \theta^-)$, $U(Z)$ is a uniform distribution over Z, and θ^- represents the parameters of the target network. The target network is the same as the online network, except that its parameters θ^- are updated by the online network at predefined intervals. Both the experience replay memory and the target network enhance stability of the learning process dramatically.

2.2 Deep Recurrent Q-Network

Deep recurrent Q-network (DRQN) is proposed by [15] to deal with partial observability caused by incomplete and noisy state information in real-world tasks. It is developed to train an agent in an environment modeled as a partial observable Markov decision
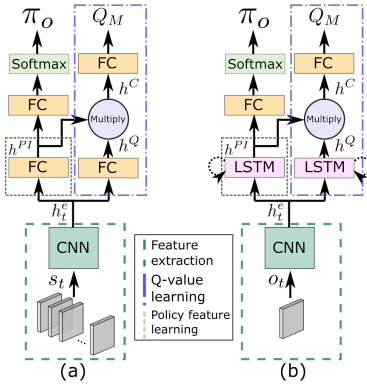
Figure 1: Architectures of DPIQN (a) and DRPIQN (b).

Figure 2: Generalized architecture of DPIQN and DRPIQN.

process (POMDP), in which the state of the environment is not fully observable or determinable from a limited number of past states. DRQN models $\mathcal{E}$ as a 6-tuple $(s, a, \mathcal{T}, \mathcal{R}, \gamma, o)$, where o is the observation perceived by the agent. Instead of using only the last few states to predict the next action as DQN, DRQN extends the architecture of DQN with Long Short Term Memory (LSTM) [18]. It integrates the information across observations by an LSTM layer to a hidden state h and an internal cell state c, which recurrently encodes the information of the past observations. The Q-function of DRQN is represented as $Q(o, h, a)$. DRQN has been demonstrated to perform better than DQN at all levels of partial information [15].

2.3 Q-learning in Multi-Agent Environments

In a real-world MAS, the environment state is affected by the joint action of all agents. The Q-function of an agent is thus dependent on the actions of the others. Adaptation of the Q-function definition becomes necessary, such that the other agents' actions are taken into consideration.

In order to re-formulate the Q-function for multi-agent settings, we assume that the environment $\mathcal{E}$ contains a group of $N + 1$ agents: a controllable agent and the other N agents. The latter can be either collaborators or opponents. The joint action of those N agents is defined as $a_o \in \mathcal{A}_o$, where $\mathcal{A}_o \colon \mathcal{A}_1 \times \mathcal{A}_2 \cdots \times \mathcal{A}_N$, and $\mathcal{A}_1 \sim \mathcal{A}_N$ are the action spaces of agents $1 \sim N$, respectively. The reward function $\mathcal{R}_M$ and the state transition function $\mathcal{T}_M$, therefore, become $r = \mathcal{R}_M(s, a, a_o, s')$ and $\mathcal{T}_M(s', s, a, a_o) = \Pr(s'|s, a, a_o)$, where the subscript M denotes multi-agent settings. We further define the other N agents' joint policy as $\pi_o(a_o|s)$. Based on $\mathcal{R}_M$ and $\mathcal{T}_M$, Eq. (1) can then be rewritten as:

$$Q_M{}^*(s, a|\pi_o) = \sum_{a_o} \pi_o(a_o|s) \sum_{s'} \mathcal{T}_M(s', s, a, a_o)$$
$$\left[\mathcal{R}_M(s, a, a_o, s') + \gamma \mathbb{E}_{a'} \left[Q_M{}^*(s', a'|\pi_o) \right] \right]$$
(3)

Please note that in the above equation, the Q-function is conditioned on π_o, rather than a_o. Conditioning the Q-function on a_o leads to an explosion of the number of parameters, and is not suitable for an MAS with multiple agents.
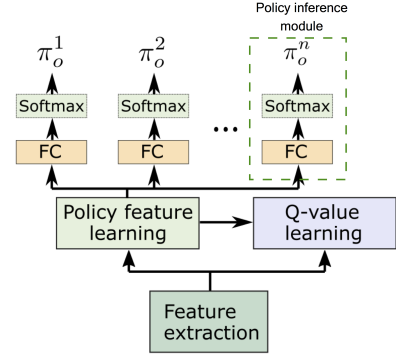
3 DEEP POLICY INFERENCE Q-NETWORK

In this section, we present the architecture and implementation details of DPIQN. We first outline the network structure, and introduce the concept of policy features. Then, we present a variant version of DPIQN, called DRPIQN, for dealing with partial observability. We next explain the training methodology in detail. Finally, we discuss the generalization methodology for environments with multiple agents.

3.1 DPIQN

The main objective of DPIQN is to improve the quality of state feature representations of an agent in multi-agent settings. As mentioned in the previous section, the environmental dynamics are affected by multiple agents. In order to enhance the hidden representations such that the controllable agent can exploit the other agents' actions in an MAS, DPIQN learns the other agents' "policy features" by auxiliary tasks. Assume that the environment contains a controllable agent and a target agent. Policy features are defined as a hidden representation used by the controllable agent to infer the policy of the target agent, and are represented as a vector h^{PI}. The approximated representation encodes the spatial-temporal features of the target agent's policy π_o, and can be obtained by observing the target agent's behavior for a series of consecutive timesteps. By incorporating h^{PI} into the model of the controllable agent, DPIQN is able to learn a better Q-function than traditional methodologies. DPIQN adapts itself to dynamic environments in which the target agent's policy changes over time. The results are presented in Section 4.

The network architecture of DPIQN is illustrated in Fig. 1(a), which first extracts image features h_t^e, and computes the controllable agent's Q-value $Q_M(s, a|h^{PI}, h_t^e; \theta)$ and the target agent's policy π_o. Note that Q_M is conditioned on the policy feature vector h^{PI}, rather than the target agent's action a_o or policy π_o. We omit h^{PI} and h_t^e in Q_M in the rest of the paper for simplicity. DPIQN consists of three parts: a feature extraction module, a Q-value learning module, and an auxiliary policy feature learning module. The feature extraction module is a convolutional neural network (CNN) shared between the latter two modules, and is responsible for extracting the spatio-temporal features from the latest k inputs (i.e. observations). The extracted feature at timestep t are denoted as h_t^e. In our experiments presented in Section 4, the inputs applied at

the feature extraction module are raw pixel data.

Both the Q-value learning module and the policy feature learning module take h_t^e as their common input. In DPIQN, these two modules are composed of a series of fully-connected layers (abbreviated as FC layers). The Q-value learning module is trained to approximate the optimal Q-function Q_M^*, while the policy feature learning module is trained to infer the target agent's next action a_o. Based on h_t^e, these two modules separately extract two types of features denoted as h^Q and h^{PI} using two FC layers followed by non-linear activation functions. The policy feature vector h^{PI} is further fed into an FC layer and a softmax layer, which are collectively referred to as a policy inference module, to generate the target agent's approximated policy $\pi_o(a_o|h_t^e)$. The approximated policy π_o is then used to compute the cross entropy loss against the true action of the target agent. The training procedure is discussed in subsection 3.3. To derive the Q values of the controllable agent, the policy feature vector h^{PI} is fed into the Q-value learning module, and merged with h^Q by multiplication to generate h^C, as annotated in Fig. 1. h^C is then processed by an FC layer to generate the Q-value of the controllable agent.

3.2 DRPIQN

DRPIQN is a variant of DPIQN motivated by DRQN for handling partial observability, with an emphasis on decreasing the hidden state representation noise from strategy changing of the other agents in the environment. For example, the policy π_o of an opponent in a competitive task may switch from a defensive mode to an offensive mode in an episode, leading to an increased difficulty in adapting the Q-function and the approximated policy feature vector h^{PI} of the controllable agent to such variations. This becomes even more severe in multi-agent settings when the policies of all agents change over time, resulting in degradation in the stability of h^{PI} (and hence, h^C). In such environments, inferring the policy of a target agent becomes a POMDP problem: the intention of the target agent cannot be directly deduced or inferred from only a few observations.

DRPIQN is proposed to incorporate recurrent units in the baseline DPIQN model to deal with the above issues, as illustrated in Fig. 1 (b). DRPIQN takes a single observation as its input. It similarly employs a CNN to extract spatial features h^e from the input, but uses the LSTM layers to encode temporal correlations between a history of them. Due to its capability of learning long-term dependencies, the LSTM layers are able to capture a better policy feature representation. We show in Section 4 that DRPIQN demonstrates superior generalizability to unfamiliar agents than the baseline models.

3.3 Training with Adaptive Loss

In this section, we provide an overview of the training methodology used for DPIQN and DRPIQN. Algorithm 1 provides the pseudocode of the training procedure. Our training methodology stems from that of DQN, with a modification of the definition of loss function. We propose to adopt two different loss function terms L^Q and L^{PI} to train our models. The former is the standard DQN loss function. The latter is called the policy inference loss, and is obtained by computing the cross entropy loss between the inferred policy π_o

---

Algorithm 1 Training Procedure of DPIQN

1: Initialize replay memory Z, environment $\mathcal{E}$, and observation s
2: Initialize network weights θ
3: Initialize target network weights θ^-
4: **for** timestep $t = 1$ to T **do**
5:　　Take a with ϵ-greedy based on $Q_M(s, a; \theta)$
6:　　Execute a in $\mathcal{E}$ and observe μ_o, r, s'
7:　　Clip r between $[-1, 1]$
8:　　Store transition (s, a, a_o, r, s') in Z
9:　　Sample mini-batch $(s^j, a^j, a_o^j, r^j, s^{j'})$ from Z
10:　Set $y = \begin{cases} r^j & \text{for terminal } s^{j'} \\ r^j + \gamma \max_{a^{j'}} Q_M(s^{j'}, a^{j'}; \theta^-) & \text{otherwise} \end{cases}$
11:　Compute L^Q based on y, and compute L^{PI}
12:　Compute $\lambda = \frac{1}{\sqrt{L^{PI}}}$
13:　Compute gradient G based on L^Q, L^{PI}, λ
14:　Perform gradient descent G
15:　Update $s = s'$
16:　Update $\theta^- = \theta$ for every C steps
17: **end**

---

and the ground truth one-hot action vector μ_o of the target agent. L^{PI} is expressed as:

$$L^{PI} = H(\mu_o) + D_{KL}(\mu_o \| \pi_o) \tag{4}$$

where $H(\mu_o)$ is the entropy of μ_o, and $D_{KL}(\mu_o \| \pi_o)$ stands for the Kullback-Leibler divergence of π_o from μ_o. The aggregated loss function can be expressed as:

$$L = E_{mini-batch \sim U(Z)}[(\lambda L^Q + L^{PI})] \tag{5}$$

where λ is called the adaptive scale factor of L^Q. The function of λ is to adaptively scale L^Q at different phases of the training process. It is defined as:

$$\lambda = \frac{1}{\sqrt{L_t^{PI}}} \tag{6}$$

In the initial phase of the training process, L^{PI} is large, corresponding to a small λL^Q. A small λL^Q encourages the network to focus on learning the policy feature vector h^{PI}. When the network is trained such that L^{PI} is sufficiently small, λL^Q becomes dominant in Eq. (5), turning the network's attention to optimize the Q values. We found that without the use of λ, the Q values tend to converge in an optimistic fashion, leading to a degradation in performance. The intuition behind λ is that if the controllable agent possesses sufficient knowledge of the target agent, it is able to exploit this knowledge to make a better decision. The use of λ significantly improves the stability of the learning curve of the controllable agent. A comparison of performance with and without the usage of λ is presented in Section 4.5. Note that during testing, the forward path only calculates the Q values of the controllable agent, and does not need to be fed with the moves of the target agent.

3.4 Generalization

DPIQN and DRPIQN are both generalizable to complex environments with multiple agents. Consider a multi-agent environment in which both cooperative and competitive agents coexist (e.g., a soccer game). The policies of these agents are diverse in terms of
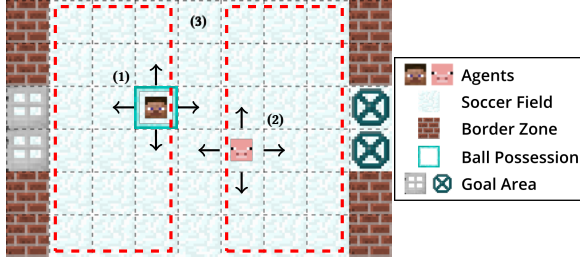
Figure 3: Illustration of the soccer game in 1 vs. 1 scenario. (1) is our controllable agent, (2) is the rule-based opponent, (3) is the start zone of the agents. The agent who possesses the ball is surrounded by a blue square.

Figure 4: Policy of the rule-based agent in each episode.

Table 1: Hyperparameters used in our experiments.

List of Hyperparameters	
Epoch length	10,000 timesteps (2,500 training steps)
Optimizer	Adam [21]
Learning rate	$\underset{\text{(Initial)}}{0.001} \rightarrow \underset{\text{(Epoch 600)}}{0.0004} \rightarrow \underset{\text{(Epoch 1000)}}{0.0002}$
ϵ-greedy	$\underset{\text{(Initial)}}{1.0} \rightarrow \underset{\text{(Epoch 100)}}{0.1}$
Discount factor	0.99
Target network	Update every 10,000 steps
Replay memory	1,000,000 samples
History length	12 (DQN/DPIQN) 1 (DRQN/DRPIQN)
Minibatch size	32

Table 2: Evaluation result of 1 vs. 1 scenario.

Model	Training	Hybrid	Offensive	Defensive
DQN	Hybrid	-0.063	-0.850	0.000
	Offensive	0.312	<u>0.658</u>	0.113
	Defensive	-0.081	-1.000	<u>0.959</u>
DRQN	Hybrid	0.028	-0.025	0.168
DPIQN	Hybrid	**0.999**	**0.989**	0.986
DRPIQN	Hybrid	**0.999**	0.981	**1.000**

4 EXPERIMENTAL RESULTS

In this section, we present experimental results and discuss their implications. We start by a brief introduction to our experimental setup, as well as the environment we used to evaluate our models.

4.1 Experimental Setup

We perform our experiments on a soccer game environment illustrated in Fig. 3. We begin with explaining the environments and game rules. The hyperparameters we used are summarized in Table 1. The source code is developed based on Tensorpack[1], which is a neural network training interface built on top of TensorFlow [1].

Environment. Fig. 3 illustrates the soccer game environment used in our experiments. The soccer field is a grid world composed of multiple grids of 32×32 RGB pixels and is divided into two halves. The game starts with the controllable agent and the collaborator (Fig. 3(1)) randomly located on the left half of the field, and the opponents (Fig. 3(2)) randomly located on the right half of the field, except the goals and border zones (Fig. 3(3)). The initial possession of the ball (highlighted by a blue rectangle) and the modes of the agents (offensive or defensive) are randomly determined for each episode. In each episode, each team's objective is to deliver the ball to the opposing team's goal. An episode terminates immediately once a team scores a goal. A reward of 1 is awarded if the controllable agent's team wins, while a penalty reward of -1 is given if it loses the match. If neither of the teams is able to score within 100 timesteps, the episode ends with a reward of 0. Each agent in
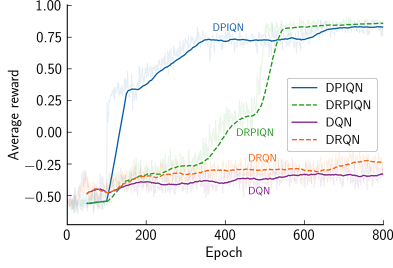
their objectives and tactics. The aim of the collaborative agents (collaborators) is to work with the controllable agent to achieve a common goal, while that of the competitive agents (opponents) is to act against the controllable agent's team. Some of the agents are more offensive, while others are more defensive. In such a heterogeneous environment, conditioning the Q-function of the controllable agent on the actions of distinct agents would lead to an explosion of parameters, as mentioned in Section 3.3. To reduce model complexity and concentrate the controllable agent's focus on the big picture, we propose to summarize the policies of the other agents in a single policy feature vector h^{PI}. We extend the policy feature learning module in DPIQN and DRPIQN to incorporate multiple policy inference modules to learn the other agents' policies separately, as illustrated in Fig. 2. Each policy inference module corresponds to either a collaborator or an opponent. The loss function term L^{PI} in Eq. (5) is then modified as:

$$L^{PI} = \frac{1}{N} \sum_{i=0}^{N} H(\mu_o^i) + D_{KL}(\mu_o^i \| \pi_o^i) \qquad (7)$$

where N is the total number of the target agents, and i indicates the i-th target agent. The training procedure is the same as Algorithm 1, except lines 8~10 are modified to incorporate a_o^i, π_o^i, and μ_o^i. The learned h^{PI}, therefore, embraces the policy features from the collaborators and opponents. The experimental results of the proposed generalization scheme is presented in Sections 4.3, 4.4 and 4.5.

Figure 5: Learning curve comparison in the 1 vs. 1 scenario.
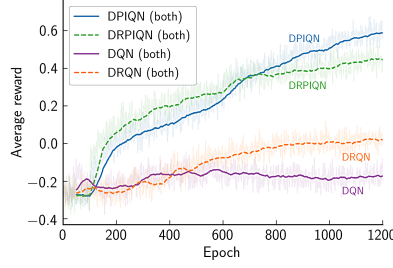
Figure 6: Learning curve comparison in the 2 vs. 2 scenario.

Table 3: Evaluation result of 2 vs. 2 scenario.

Model	Training	Testing
DQN	-	-0.152
DRQN	-	-0.031
DPIQN	Both	0.761
	O-only	0.645
	C-only	0.233
DRPIQN	Both	0.695
	O-only	0.714
	C-only	**0.854**

the field chooses from five possible actions: *move, N, S, W, E* and *stand still* at each time step. If the intended position of an agent is out of bounds or overlaps with that of the other agents, the move doesn't take place. In the latter case, the agent who originally possesses the ball passes/loses it to the collaborator/opponent who intend to move to the same position. In our simulations, the controllable agent receives inputs in the form of resized grayscale images $o \in \mathbb{R}^{84 \times 84 \times 1}$ of the current entire world (Fig. 3).

1 vs. 1 Scenario. In this scenario, the game is played by a controllable agent and an opponent on a 6×9 grid world (Fig. 3). The opponent is a two-mode rule-based agent playing according to Fig. 4. In the offensive mode, the opponent focuses on scoring a goal, or stealing the ball from the controllable agent. In the defensive mode, the opponent concentrates on defending its own goal, or moving away from the controllable agent when it possesses the ball. We set the frame skip rate to 1, which is optimal for DQN after an exhaustive search of hyperparameters.

2 vs. 2 Scenario. In this scenario, each of the two teams contains two agents. The two teams compete against each other on a grid world with larger areas of goals and border zones. We consider two tasks in this scenario: our controllable agent has to collaborate with (1) a rule-based collaborator or (2) a learning agent to compete with the rule-based opponents. The sizes of the grid world are 13×10 and 21×14, respectively. In addition, the rule-based agents in this scenario also play according to Fig. 4. When a teammate has the ball, the rule-based agent does its best to obstruct the nearest opponent. We set the frame skip rate to 2 in this scenario due to the increased size of the environment.

4.2 Performance Comparison in 1 vs.1 Scenario

Table 2 compares the controllable agent's average rewards among three types of the opponent agent's modes in the testing phase for four types of models, including DQN, DRQN, DPIQN, and DRPIQN. The average rewards are evaluated over 100,000 episodes. The types of the opponent agent's modes include "hybrid", "offensive", and "defensive". A hybrid mode means that the opponent's mode is either offensive or defensive, and is determined randomly at the beginning of an episode. Once the mode is determined, it remains fixed till the end of that episode. The second and third columns of Table 2 correspond to the opponent's modes in the training and testing phases, respectively. All models are trained for 2 million timesteps, corresponding to 800 epochs in the training phase. The highest average rewards in each column are marked in

bold. The results show that DPIQN and DRPIQN outperform DQN and DRQN in all cases under the same hyperparameter setting. No matter which mode the opponent belongs to, DPIQN and DRPIQN agents are both able to score a goal for around 99% of the episodes. The results indicate that incorporating the policy features of the opponent into the Q-value learning module does help DPIQN and DRPIQN to derive better Q values, compared to those of DQN and DRQN. Another interesting observation from Table 2 is that DQN agents are also able to achieve sufficiently high average rewards, as long as the mode of the opponent remains the same in the training and testing phases. When DQN agents face unfamiliar opponents in the testing phase, they play poorly and lose the games most of the time. This indicates that DQN agents are unable to adapt themselves to opponents with different strategies and, hence, non-stationary environments. We have also observed that DPIQN and DRPIQN agents tend to play aggressively in most of the games, while DQN and DRQN agents are often confused by the opponent's moves.

Fig. 5 plots the learning curves of the four models in the training phase. The numbers in Fig. 5 are averaged from the scores of the first 500 episodes in each epoch. The opponent's modes in all of the four cases are random. It can be seen that DPIQN and DRPIQN learn much faster than DQN and DRQN. DRPIQN's curve increases slower than DPIQN's due to the extra parameters from the LSTM layers in DRPIQN's model. Please note that the final average rewards of DPIQN and DRPIQN converge to around 0.8. This is largely due to the ϵ-greedy technique we used in the training phase. Another reason is that in the testing phase, the average rewards are evaluated over 100,000 episodes, leading to less variations.

4.3 Collaboration with Rule-based Agent

Table 3 compares the average rewards of the controllable agent's team in the 2 vs. 2 scenario for four types of models used to implement the controllable agent. Similarly, the average rewards are evaluated over 100,000 episodes. Both the collaborator and opponents are rule-based agents, and are set to the hybrid mode. The second column of Table 3 indicates which rule-based agent's policy features are learned by DPIQN and DRPIQN agents in the training phase. "Both" means that DPIQN and DRPQIN agents learn the policy features of both the collaborator and the opponents, while "C-only"/"O-only" represents that only the policy features of the collaborator/the opponents are considered by our agents, respectively. We denote them as *DPIQN/DRPIQN (B)*, *DPIQN/DRPIQN (C)*, and *DPIQN/DRPIQN (O)* for these three different settings. All models are
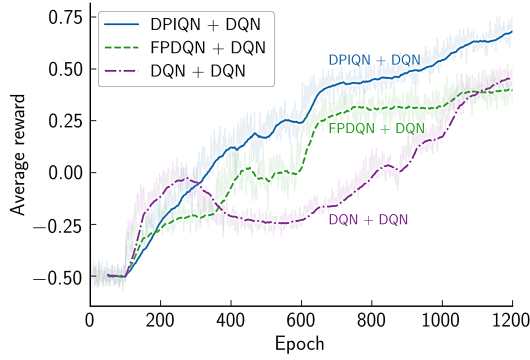
Figure 7: Mean rewards of different collaboration teams.

Model	Scoring Rate	Draws	Avg. rewards
DPIQN	31.54%	**11.97%**	**0.829**
FPDQN	0.00%	36.83%	0.555
DQN	58.70%	39.77%	0.529

Table 5: Impact of unfamiliar agents on average reward.

		Unfamiliar-O	Unfamiliar-C
		1 vs. 1 Scenario	
DPIQN		0.909 (90%)	-
DRPIQN		**0.947 (94%)**	-
		2 vs. 2 Scenario (rule-based)	
DPIQN	Both	0.501 (65%)	0.645 (84%)
	O-only	0.488 (75%)	0.535 (82%)
	C-only	0.076 (32%)	0.189 (81%)
DRPIQN	Both	0.534 (76%)	0.565 (81%)
	O-only	**0.578 (80%)**	**0.625 (87%)**
	C-only	0.625 (73%)	0.695 (82%)

trained for 3 million timesteps (1,200 epochs). The highest average reward in the third column is marked in bold.

From the results in Table 3, we can see that DPIQN and DRPIQN agents are much superior to DQN and DRQN agents in the 2 vs 2 scenario. For most of the episodes, DQN and DRQN agents' teams lose the game with their average rewards less than zero. On the other hand, DPIQN and DRPIQN agents' teams are able to demonstrate a higher goal scoring ability. We have observed in our experiments that our agents have learned to pass the ball to its teammate, or save the ball from its teammate chased by the opponents. This implies that our agents had learned to collaborate with its teammate. On the contrary, the baseline DQN and DRQN agents are often confused by the opponent team's moves, and are relatively conservative in deciding their actions. As a result, they usually stand still in the same place, resulting in losing the possession of the ball. Lastly, it is noteworthy that both DRPIQN (C) and DPIQN (O) achieve high average rewards. This indicates that for our models, it is sufficient to only model a subset of agents in the environment. Therefore, we consider DPIQN and DRPIQN to be potentially scalable to a more complex environment.

Fig. 6 plots the learning curves of DPIQN (B), DRPIQN (B), DQN, and DRQN in the training phase. Similarly, the numbers in Fig. 6 are averaged from the scores of the first 500 episodes in each epoch. It can again be observed that the learning curves of DPIQN and DRPIQN grow much faster than those of DQN and DRQN. Even at the end of the training phase, the average rewards of our models are still increasing. From the results in Table 3 and Fig. 6, we conclude that DPIQN and DRPIQN are generalizable to complex environments with multiple agents.

4.4 Collaboration with Learning Agent

To test our model's ability to cooperate with a learning collaborator, we conduct further experiments in the 2 vs. 2 scenario. As the environment contains two learning agents in this setting, we additionally introduce fingerprints DQN (FPDQN) [2] [14], a DQN-based multi-agent RL method to tackle non-stationarity, as a baseline model. Each of the three types of models, including DQN, FPDQN and DPIQN, has to team up with a learning DQN agent to play against the opposing team. All models mentioned above adopt the

---
[2]FPDQN takes as input (s, ϵ, e), where e is the current training epoch, as suggested in the original paper.

same parameter settings, and the rule-based agents in the opposing team are set to the hybrid mode. The average rewards are similarly evaluated over 100,000 episodes. Please note that DPIQN only models its collaborator, i.e. the DQN agent, in the experiment.

Fig. 7 plots the learning curves of the three teams in the training phase. The numbers in Fig. 7 are similarly averaged from the scores of the first 500 episodes in each epoch. It can be observed that the learning curves of both DPIQN and FPDQN grow much faster and steadier than DQN. It takes DQN 1,000 epochs to reach the same level of performance as FPDQN. On the other hand, DPIQN consistently receives higher average rewards than the other models. From the results in Fig. 7, we show the effectiveness of our model in a multi-agent learning setting.

In Table 4, we report the scoring rate for each controllable agent, the percentage of draws, as well as the evaluation results in the testing phase. We observe that the DPIQN agent only scores 31.54% of the time, and runs forward to support its teammate's attack whenever the collaborator gains possession of the ball and advances towards the goal. On the contrary, when DPIQN agent possesses the ball, we do not observe any similar behavior from the DQN collaborator. The result shows that DPIQN is superior in cooperating with and assisting its teammate through modeling the collaborator. Furthermore, although FPDQN learns faster than DQN agents' team, it does not seem to show any sign of collaboration with its teammate, and spends most of its time staying in the offensive half. Therefore, it achieves only a 0.00% scoring rate despite its higher average rewards than the DQN team. We also observe that the DQN agents' team hardly ever steals the ball from the rule-based agents, and therefore often ends up in draws. In addition, when one of the DQN agents possesses the ball, the other DQN tends to stand still, instead of assisting its teammate. This behavior is similar to what we have observed in DPIQN's team. We therefore conclude that DQN agents lack the ability to cooperate efficiently. From the result in Figure 7 and Table 4, we further show that our proposed model
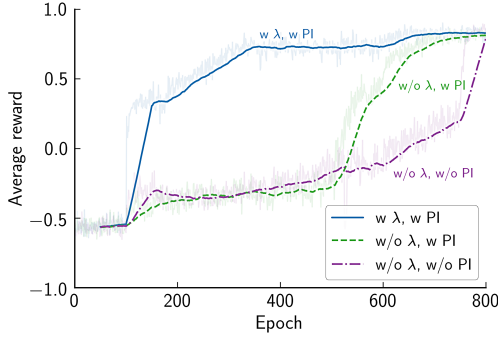
Figure 8: Impact of λ and L^{PI} on average reward.

Figure 9: Impact of λ and L^{PI} on L^Q.

is capable of handling non-stationarity, as well as improving the overall performance of RL agents.

4.5 Generalizability to Unfamiliar Agent

In this section, we show that DPIQN and DRPIQN are capable of dealing with unfamiliar agents whose policies change over time. We evaluate our models in the two scenarios discussed above, and summarize our results in Table 5. In the training phase, DPIQN and DRPIQN agents are trained against rule-based agents with fixed policies in an episode. However, in the testing phase, the policies of the collaborator or opponents are no longer fixed. Each of them may randomly updates its policy mode (either offensive of defensive) with an irregular update period ranging from 4 to 10 timesteps. The update period is also randomly determined. This setting thus makes the controllable agent unfamiliar with the policies of its collaborator or opponents in the testing phase, allowing us to validate the generalizability of DPIQN and DRPIQN. Note that the average rewards listed in Table 5 are evaluated over 100,000 episodes.

Table 5 compares two cases for validating the generalizability of DPIQN and DRPIQN: unfamiliar opponents (abbreviated as unfamiliar-O) and unfamiliar collaborator (abbreviated as unfamiliar-C). These two cases correspond to the second and third columns of Table 5, respectively. We report results for the 1 vs. 1 and 2 vs. 2 scenarios in separate rows. The numbers in the parenthesis are ratios of the average rewards in Table 5 to those of the corresponding entries in Tables 2 & 3. In the 1 vs. 1 scenario, DPIQN and DRPIQN are able to achieve average rewards of 0.909 and 0.947, respectively, even when confronted with an unfamiliar opponent. In the 2 vs. 2 scenario, both DPIQN and DRPIQN maintain their performance in most of the cases. It can be seen that DRPIQN performs slightly better than DPIQN in these two scenarios when playing against unfamiliar opponents. We have observed that DRPIQN (O) achieves the highest average reward ratios among all of the cases. One potential explanation is that DRPIQN (O) focuses only the policy features of the opponents in the training phase, therefore is able to adapt itself to unfamiliar opponents better than the other settings. Table 5 also indicates that DPIQN and DRPIQN perform better when facing with unfamiliar collaborators than unfamiliar opponents. We have observed that when collaborating with an unfamiliar agent, DPIQN and DRPIQN agents tend to score a goal by itself, due to its lack of knowledge about the collaborator's intentions.
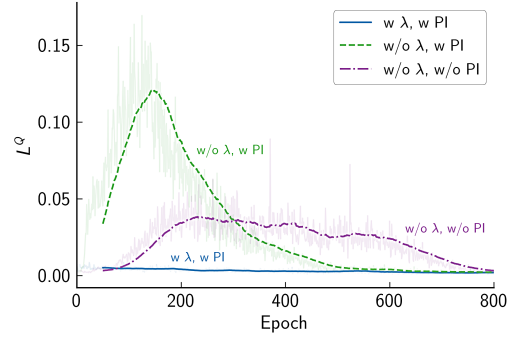
4.6 Ablative Analysis

We further investigate the effectiveness of our adaptive loss function by a detailed analysis of L^{PI} and L^Q. Moreover, we plot the learning curves of three different cases, and show that our adaptive loss design helps accelerate convergence and stabilize training. We focus exclusively on DPIQN, as DRPIQN produces similar results.

Fig. 8 illustrates the learning curves of DPIQN in the 1 vs. 1 scenario. These three curves correspond to DPIQN models trained with or without the use of λ and L^{PI} in Eq. 5. Although all of the three cases converge to an average reward of 0.8 at the end, the one trained with both λ and L^{PI} converges much faster than the others. We further analyze L^Q for the three cases in Fig. 9. It is observed that the DPIQN model trained with both λ and L^{PI} shows less fluctuations in L^Q and thus better stability than the other two cases in the training phase. We conclude that both policy inference and adaptive loss are essential to DPIQN's performance.

5 CONCLUSION

In this paper, we presented an in-depth design of DPIQN and its variant DRPIQN, suited to multi-agent environments. We presented the concept of policy features, and proposed to incorporate them as a hidden vector into the Q-networks of the controllable agent. We trained our models with an adaptive loss function, which guides our models to learn policy features before Q values. We extended the architectures of DPIQN and DRPIQN to model multiple agents, such that it is able to capture the behaviors of the other agents in the environment. We performed experiments for two soccer game scenarios, and demonstrated that DPIQN and DRPIQN outperform DQN and DRQN in various settings. Moreover, we verified that DPIQN is capable of dealing with non-stationarity by conducting experiments where the controllable agent has to cooperate with a learning agent, and showed that DPIQN is superior in collaboration to a recent multi-agent RL approach. We further validated the generalizability of our models in handling unfamiliar collaborators and opponents. Finally, we analyzed the loss function terms, and demonstrated that our adaptive loss function does improve the stability and learning speed of our models.

6 ACKNOWLEDGMENTS

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467* (Mar. 2016).

[2] Sherief Abdallah and Victor Lesser. 2008. A multiagent reinforcement learning algorithm with non-linear dynamics. *J. Artificial Intelligence Research (JAIR),* vol. 33, no.1, pp. 512-549 (Dec. 2008).

[3] Hidehisa Akiyama, Shigeto Aramaki, and Tomoharu Nakashima. 2012. Online cooperative behavior planning using a tree search method in the robocup soccer simulation. In *Proc. IEEE Conf. Intelligent Networking and Collaborative Systems (INCoS).* pp. 170-177.

[4] Stefano V Albrecht and Peter Stone. 2017. Autonomous agents modelling other agents: A comprehensive survey and open problems. *arXiv:1709.08071* (Sep. 2017).

[5] Aijun Bai, Feng Wu, and Xiaoping Chen. 2015. Online planning for large markov decision processes with hierarchical decomposition. *ACM Trans. Intelligent Systems and Technology (TIST),* vol. 6, no. 4, pp. 45:1-45:28 (Aug. 2015).

[6] Bikramjit Banerjee and Jing Peng. 2007. Generalized multiagent learning with performance bound. *J. Autonomous Agents and Multi-Agent Systems (JAAMAS),* vol. 15, no. 3, pp. 281-312 (Dec. 2007).

[7] Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. 1998. Opponent modeling in poker. In *Proc. AAAI Conf. Artificial Intelligence.* pp. 493-499.

[8] Daan Bloembergen, Karl Tuyls, Daniel Hennes, and Michael Kaisers. 2015. Evolutionary Dynamics of Multi-Agent Learning: A Survey. *J. Artificial Intelligence Research (JAIR),* vol. 53, no. 1, pp. 659-697 (Aug. 2015).

[9] Michael Bowling. 2005. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems (NIPS).* pp. 209-216.

[10] Michael Bowling and Manuela Veloso. 2002. Multiagent learning using a variable learning rate. *J. Artificial Intelligence,* vol. 136, no. 2, pp. 215-250 (Apr. 2002).

[11] Alvaro Ovalle Castaneda. 2016. Deep reinforcement learning variants of multi-agent learning algorithms. *Master's thesis, School of Informatics, University of Edinburgh* (2016).

[12] Brian Collins. 2007. Combining opponent modeling and model-based reinforcement learning in a two-player competitive game. *Master's thesis, School of Informatics, University of Edinburgh* (2007).

[13] Vincent Conitzer and Tuomas Sandholm. 2007. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *J. Machine Learning,* vol. 67, no. 1-2, pp. 23-34 (May 2007).

[14] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, and Shimon Whiteson. 2017. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proc. Int. Conf. Machine Learning (ICML).* pp. 1146-1155.

[15] Matthew J. Hausknecht and Peter Stone. 2015. Deep recurrent Q-Learning for partially observable MDPs. *arXiv:1507.06527* (Jul. 2015).

[16] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. 2016. Opponent modeling in deep reinforcement learning. In *Proc. Machine Learning Research (PMLR).* pp. 1804-1813.

[17] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. 2017. A survey of learning in multiagent Environments: Dealing with non-stationarity. *arXiv:1707.09183* (Jul. 2017).

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation,* vol. 9, no. 8, pp. 1735-1780 (Mar. 1997).

[19] Junling Hu and Michael P. Wellman. 2003. Nash Q-learning for general-sum stochastic games. *J. Machine Learning Research (JMLR),* (Dec. 2003), pp. 1039-1069.

[20] Max Jaderberg, Volodymyr Mnih, Wojciech Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. 2016. Reinforcement learning with unsupervised auxiliary tasks. In *Proc. Int. Conf. Learning Representations (ICLR).*

[21] Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. (May 2015).

[22] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. In *Proc. Conf. Autonomous Agents and MultiAgent Systems (AAMAS).* International Foundation for Autonomous Agents and Multiagent Systems, pp. 464-473.

[23] Michael L Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proc. Machine Learning Research (PMLR).* pp. 157-163.

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv:1706.02275* (Jun. 2017).

[25] Richard Mealing and Jonathan L. Shapiro. 2013. Opponent modeling by sequence prediction and lookahead in two-player games. In *Int. Conf. Artificial Intelligence and Soft Computing (ICAISC).* pp. 385-396.

[26] Piotr W. Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. 2016. Learning to navigate in complex environments. In *Proc. Int. Conf. Learning Representations (ICLR).*

[27] 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. 2015. Human-level control through deep reinforcement learning. *Nature,* vol. 518, no. 7540, pp. 529-533 (Feb. 2015).

[28] Ann Nowé, Peter Vrancx, and Yann-Michaël De Hauwere. 2012. Game theory and multi-agent reinforcement learning. In *Reinforcement Learning: State-of-the-Art.* pp. 441-470.

[29] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *Proc. Int. Conf. Machine Learning (ICML).*

[30] Frederik Schadd, Sander Bakkes, and Pieter Spronck. 2007. Opponent modeling in real-time strategy games. In *Proc. GAME-ON.* pp. 61-70.

[31] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. 2016. Loss is its own Reward: Self-Supervision for Reinforcement Learning. *arXiv:1612.07307* (Dec. 2016).

[32] 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. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature,* vol. 529, no. 7587, pp. 484-489 (Jan. 2016).

[33] Satinder P. Singh, Michael J. Kearns, and Yishay Mansour. 2000. Nash convergence of gradient dynamics in general-sum games. In *Proc. Conf. Uncertainty in Artificial Intelligence.* pp. 541-548.

[34] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to reinforcement learning.* MIT Press.

[35] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicenteand. 2015. Multiagent cooperation and competition with deep reinforcement learning. *arXiv:1511.08779* (Nov. 2015).

[36] William Uther and Manuela Veloso. 1997. *Adversarial reinforcement learning.* Technical Report. Carnegie Mellon University. Unpublished.

[37] Chongjie Zhang and Victor Lesser. 2010. Multi-agent learning with policy prediction. In *Proc. AAAI Conf. Artificial Intelligence.* pp. 927-934.

[38] Marvin Zhang, Zoe McCarthy, Chelsea Finn, Sergey Levine, and Pieter Abbeel. 2016. Learning deep neural network policies with continuous memory states. In *Proc. IEEE Conf. Robotics and Automation (ICRA).* pp. 520-527.