Online Minimax Q Network Learning for Two-Player Zero-Sum Markov Games

Yuanheng Zhu, *Member, IEEE*, and Dongbin Zhao, *Fellow, IEEE*

***Abstract*— The Nash equilibrium is an important concept in game theory. It describes the least exploitability of one player from any opponents. We combine game theory, dynamic programming, and recent deep reinforcement learning (DRL) techniques to online learn the Nash equilibrium policy for two-player zero-sum Markov games (TZMGs). The problem is first formulated as a Bellman minimax equation, and generalized policy iteration (GPI) provides a double-loop iterative way to find the equilibrium. Then, neural networks are introduced to approximate Q functions for large-scale problems. An online minimax Q network learning algorithm is proposed to train the network with observations. Experience replay, dueling network, and double Q-learning are applied to improve the learning process. The contributions are twofold: 1) DRL techniques are combined with GPI to find the TZMG Nash equilibrium for the first time and 2) the convergence of the online learning algorithm with a lookup table and experience replay is proven, whose proof is not only useful for TZMGs but also instructive for single-agent Markov decision problems. Experiments on different examples validate the effectiveness of the proposed algorithm on TZMG problems.**

***Index Terms*— Deep reinforcement learning (DRL), generalized policy iteration (GPI), Markov game (MG), Nash equilibrium, Q network, zero sum.**

I. Introduction

THE Markov decision process (MDP) is a common tool to describe the interaction process between an agent and an environment [1]. Based on MDPs, reinforcement learning (RL) has been fully developed these years to address single-agent optimal decision problems [2]. However, in many real-world scenarios, there are more than just one agent existing in the same environment and jointly impacting system evolution. The existence of cooperation and competition in such game problems has attracted great attention from many fields, such as economic, traffic, and military, to name a few. The Markov games (MGs) (or stochastic games in some studies [3]) are seen as an extension of MDPs to game scenarios [4]–[6]. In MGs, multiple agents make a series of decisions to maximize a common or individual interest.

Two-player zero-sum MG (TZMG) is a special case that includes two players with totally opposite interests [7]. They share the same reward function, but one aims to maximize the sum of future rewards, while the other tries to minimize it. Existing research has paid considerable attention to turn-based games and has achieved encouraging advances [8]. In these games, players make decisions exactly one after the other, so one always knows the other-side moves. The Monte Carlo tree search [9] is a common approach, and its combination with deep RL (DRL) has beaten the professional human players [10] and other strongest artificial intelligence (AI) systems [11]. In contrast to turn-based games, simultaneous games require both sides to make decisions at the same time. Players have no knowledge of what the opponent is playing, so decision strategies can be more tricky and unpredictable. In this article, we focus on this type of TZMGs.

The Nash equilibrium is an important concept in game theory [12]. At equilibrium, no players will alter their policies without sacrificing their own interests. Especially, for TZMGs, the Nash equilibrium represents that the highest payoff a player can achieve faced with the worst opponent. Finding the Nash equilibrium policy is critical for a player in TZMGs. In continuous-time systems, the Nash equilibrium corresponds to the solution of Hamilton–Jacobi–Isaacs equation and can be reached by a deterministic policy [13]. Developed from that theory, H_∞ robust control is formulated as a zero-sum game by taking controller as the maximizing player and disturbance as the minimizing player. Adaptive dynamics programming has shown promising performance in finding the Nash equilibrium policies for H_∞ problems [14]–[16].

In discrete domains, the Nash equilibrium is not just restricted to deterministic policies. One simple example is the paper–rock–scissor game, in which the Nash equilibrium strategy is to randomly choose one of three actions with equal probability. Deterministically choosing any action is easily exploited by other players. The challenge of finding such Nash equilibrium is to solve a minimax optimization among one side's all stochastic policy set and the other's all-action set, which can be clearly formulated as a linear programming [4]. Following the development in MDPs, dynamic programming (DP) has been studied in finite TZMGs [17], and the combination with value approximation makes it possible to deal with large-scale and continuous-state games [18], [19]. To avoid the dependence on the game model, Littman [4]

Manuscript received March 15, 2020; revised August 24, 2020; accepted November 25, 2020. Date of publication December 11, 2020; date of current version March 1, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2018AAA0102404 and Grant 2018AAA0101005. *(Corresponding author: Dongbin Zhao.)*

The authors are with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: yuanheng.zhu@ia.ac.cn; dongbin.zhao@ia.ac.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2020.3041469.

Digital Object Identifier 10.1109/TNNLS.2020.3041469

extended Q-learning from single-agent MDPs to multiagent MGs and proposed the minimax-Q learning algorithm to learn Nash equilibrium from game data. However, the algorithm is inefficient in training and updating, making the final results far from the equilibrium.

In recent years, DRL has made great success in many complicated decision problems [20], but it mainly focuses on single-agent MDPs [21], [22]. Self-play is a simple but effective way to extend existing algorithms to zero-sum games [23]. An agent learns to improve its competitiveness by playing against itself. The success of self-play includes early TD-Gammon [24] and recent AlphaGo [10]. However, some studies [25] have demonstrated that simple self-play cannot ensure the learning convergence. One probable reason is the nonstationary property of opponent policies, inducing the agent to continuously alter its policy and fall into strategy circles. In order to increase model stability, Vinyals *et al.* [26] proposed the idea of population-based training in self-play, in which the learnable agent was not just playing against one opponent, but an opponent league. All players in the league were past agents in history. Based on that, the developed AlphaStar has reached the grandmaster level in StarCraft II and outperformed 99.8% human players. In addition to the convergence issue, self-play requires the opponent is able to utilize own player policies. The requirement limits self-play to only symmetric games. For nonsymmetric games, the two players may have totally different action sets.

In this article, we study the TZMGs with simultaneous decisions of both sides and propose the minimax Q network (M2QN) algorithm to online learn the Nash equilibrium in a model-free way. The algorithm stems from generalized policy iteration (GPI) and is combined with DRL techniques to improve the learning process. Q function is combined with neural networks (NNs) to address large-scale state space, and the learnable weights are trained with game data. Experience replay is introduced to increase data efficiency and speed up the learning process. The convergence of M2QN with tabular representation is theoretically proved by a stochastic approximation theory. Compared with self-play algorithms, the advantages of M2QN are the theoretical foundation of convergence to the Nash equilibrium and the applicability to both symmetric and nonsymmetric games.

The remainder of this article is organized as follows. Section II introduces the basic concepts of TZMG. Section III describes DP methods to the Bellman minimax equation for the Nash equilibrium. Section IV proposes an online learning algorithm to learn the Nash equilibrium with game data for large-scale TZMG problems. Section V tests the performance of proposed algorithm on three different examples. In Section VI, we draw our conclusions.

II. Preliminary

A. Two-Player Zero-Sum Markov Game

TZMG can be described by six-tuple $\{(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \gamma)\}$ [18], where $\mathcal{S}$ is the game state set, $\mathcal{A}$ is one player's action set, $\mathcal{O}$ is the other player's action set, $\mathcal{P}$ is the state transition function, $\mathcal{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discounted factor. At each step, two players decide and execute their actions simultaneously in the game. The new state satisfies distribution $\mathcal{P}(s_{t+1}|s_t, a_t, o_t)$, and the environment feedbacks a reward signal $r_{t+1} = \mathcal{R}(s_t, a_t, o_t)$. We simply admit that the reward is bounded and deterministic for ease of analysis. The two players have totally the opposite interests. One aims to maximize the sum of future rewards, also known as return

$$J(s_0) = \sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{1}$$

while the other tries to minimize it. Our goal is to design a player that is competitive in the game. Without loss of generality, we denote the maximizer as own player and the minimizer as the opponent. Own player chooses actions according to $a_t \sim \pi_t(s_t)$, and opponent player chooses actions according to $o_t \sim \mu_t(s_t)$. $\pi = \{\pi_0, \pi_1, \dots\}$ and $\mu = \{\mu_0, \mu_1, \dots\}$ compose own and opponent policies, respectively.

Note that, in traditional MDPs, the problem is formulated by five-tuple $\{(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)\}$ with just one action set [2]. The next state and received reward are conditioned on single-agent behavior, $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$, $r_{t+1} = \mathcal{R}(s_t, a_t)$. The corresponding target is to find a policy that maximizes the return in the same form of (1).

B. Nash or Minimax Equilibrium

In MGs, no player's policy is always optimal because its return is conditioned on other behaviors. Two well-known concepts to measure the performance of one player against others are the best response [27] and the Nash equilibrium [28]. For TZMGs that have two players with conflicting interests, the definitions are summarized as follows.

Definition 1 (TZMG Best Response): Given the opponent policy μ, an own policy π^b is called the best response if there are no policies better than π^b

$$J(s_0; \pi^b, \mu) \geq J(s_0; \pi, \mu) \quad \forall \pi.$$

Conversely, given the own policy π, the best response μ^b of the opponent player has

$$J(s_0; \pi, \mu^b) \leq J(s_0; \pi, \mu) \quad \forall \mu.$$

Definition 2 (TZMG Nash Equilibrium): The Nash equilibrium corresponds to a pair of (π^*, μ^*) that are both the best responses to each other

$$J(s_0; \pi, \mu^*) \leq J(s_0; \pi^*, \mu^*) \leq J(s_0; \pi^*, \mu) \quad \forall \pi, \mu.$$

In TZMGs, there always exists the Nash equilibrium, and it is equivalent to the minimax solution

$$J(s_0; \pi^*, \mu^*) = \max_{\pi} \min_{\mu} J(s_0; \pi, \mu) = \min_{\mu} \max_{\pi} J(s_0; \pi, \mu).$$

The Nash equilibrium specifies the maximum return that a player can have in front of the worst opponent. It is meaningful when we have no knowledge of opponent or if the opponent is a learnable agent and updates its behavior according to our policy. Equipped with π^*, our return is no less than the Nash equilibrium. However, if altering to other policies, the worst return is lower than $J(s_0; \pi^*, \mu^*)$. Therefore, we are interested in finding the Nash equilibrium policy.

C. Minimax Values and Equations

Existing literature pays much attention on stationary policy, that is, $\pi : \mathcal{S} \to \mathcal{A}$ and $\mu : \mathcal{S} \to \mathcal{O}$. Denote the expected return at the Nash equilibrium as the minimax value, $V^*(s) = \mathbb{E}[J(s; \pi^*, \mu^*)]$, and the existence of V^* has been proven for a more general class of games in [17]. By the principle of optimality, we write V^* into the Bellman minimax equation

$$V^*(s) = \max_{\pi(s)\in\Pi} \min_{o\in\mathcal{O}} \sum_{a\in\mathcal{A}} \pi(a|s) \Bigg[\mathcal{R}(s,a,o) + \gamma \times \sum_{s'\in\mathcal{S}} \mathcal{P}(s'|s,a,o) V^*(s') \Bigg].$$

In the above minimax optimization, $\pi(s)$ indicates an action distribution at a given s, and Π denotes the set of all possible distributions over own action set $\mathcal{A}$. The maximization searches for distribution because not all finite games have pure (deterministic) strategy Nash equilibriums. Some games have only mixed (stochastic) strategy Nash equilibriums [12]. Here, we consider deterministic policies as a special case of stochastic policies. The second minimization in the equation searches for a deterministic action over opponent action set $\mathcal{O}$. It is based on the fact that, once one player's policy is fixed, the problem left in TZMG becomes single-agent MDP, and a deterministic policy is sufficient to be optimal [29], [30].

In many studies, state-action values are also frequently used. The minimax Q value for TZMGs is defined as

$$Q^*(s,a,o) = \mathcal{R}(s,a,o) + \gamma \sum_{s'} \mathcal{P}(s'|s,a,o) V^*(s')$$

and we can have V^* with

$$V^*(s) = \max_{\pi(s)\in\Pi} \min_{o\in\mathcal{O}} \sum_{a\in\mathcal{A}} \pi(a|s) Q^*(s,a,o).$$

Thus, the Bellman minimax equation for the minimax Q value has

$$Q^*(s,a,o) = \mathcal{R}(s,a,o) + \gamma \sum_{s'} \mathcal{P}(s'|s,a,o) \times \max_{\pi} \min_{o'} \sum_{a'} \pi(a'|s') Q^*(s',a',o'). \quad (2)$$

According to the Sharpley theory [3], there exists a unique solution Q^* to (2).

D. Linear Programming for Minimax Solution

Once we have obtained Q^*, the Nash equilibrium policy π^* is given by

$$\pi^*(s) = \arg\max_{\pi} \min_{o} \sum_{a} \pi(a|s) Q^*(s,a,o). \quad (3)$$

In (2) and (3), there exists a minimax optimization of π. In fact, we can formulate it as a linear programming

$$\begin{cases} \max & c \\ \text{s.t.} & \sum_a p(a) Q^*(s,a,o) \geq c \quad \forall o \in \mathcal{O} \\ & \sum_a p(a) = 1 \\ & p(a) \geq 0 \quad \forall a \in \mathcal{A}. \end{cases}$$

The optimum solution $p(a)$ specifies the action distribution of minimax policy $\pi^*(a|s) = p(a)$ at a given s. The optimum objective c is equal to $\max_\pi \min_o \sum_a \pi(a|s) Q^*(s,a,o)$. In the following, the optimum solution of the linear programming with given Q is denoted by $\pi = \mathcal{G}(Q)$.

Throughout this article, the action sets $\mathcal{A}$ and $\mathcal{O}$ are assumed to be finite. It is because the Nash equilibrium policy is obtained by the minimax linear programming in (3), which is not well-defined for infinite actions. In practice, to deal with infinite actions, one solution is to discretize continuous action space or cluster infinite actions and then formulate the linear programming on finite-action sets.

III. Dynamic Programming for Bellman Minimax Equation

According to Section II, obtaining Q^* and π^* requires solving the Bellman minimax equation. In MDPs, DP plays an important role in solving the Bellman optimality equation [31], [32]. We can also apply DP to TZMGs. The following two operators are defined beforehand:

$$[\mathcal{T}(Q)](s,a,o) = \mathcal{R}(s,a,o) + \gamma \sum_{s'} \mathcal{P}(s'|s,a,o) \times \max_{\pi} \min_{o'} \sum_{a'} \pi(a'|s') Q(s',a',o')$$

$$[\mathcal{T}_\pi(Q)](s,a,o) = \mathcal{R}(s,a,o) + \gamma \sum_{s'} \mathcal{P}(s'|s,a,o) \times \min_{o'} \sum_{a'} \pi(a'|s') Q(s',a',o').$$

The first $\mathcal{T}$ calculates the next-step value by minimax. The second $\mathcal{T}_\pi$ follows the given own policy π and minimizes the next-step value over opponent action set, so it can be seen as the minimizer's Bellman optimality equation with the two-player Q value. Some common properties are shared by both $\mathcal{T}$ and $\mathcal{T}_\pi$.

1) The operator is monotone and γ-contracting.
2) It has a unique fixed point.

The first property is easily verified, and the second follows the Sharpely theory [3] for $\mathcal{T}$ and the Bellman optimality theory [1] for $\mathcal{T}_\pi$.

A. Value Iteration

Given an initial Q_0, we iteratively calculate a series of Q functions based on $\mathcal{T}$ by

$$Q_{i+1} = \mathcal{T}(Q_i).$$

Using the γ-contracting and fixed-point properties, it is easily proved that $\lim_{i\to\infty} Q_i = Q^*$ [18].

B. Policy Iteration

Policy iteration (PI) includes a policy evaluation step and a policy improvement step [18]. Given an initial own policy π_0, iterate the following two steps to produce a sequence of policies $\{\pi_i\}$.

1) Solve for a Q value by $Q_{\pi_i} = \mathcal{T}_{\pi_i}(Q_{\pi_i})$.
2) Generate a new policy following $\pi_{i+1} = \mathcal{G}(Q_{\pi_i})$.

The first equality is, in fact, the Bellman optimality equation of opponent's MDP when own player is fixed to policy π_i. Q_{π_i} represents the minimum Q value at the worst opponent. From the definition of $\mathcal{G}$, we have $\mathcal{T}_{\pi_{i+1}}(Q_{\pi_i}) \geq \mathcal{T}_{\pi}(Q_{\pi_i}) \forall \pi$. After specifying $\pi = \pi_i$, it becomes

$$\mathcal{T}_{\pi_{i+1}}(Q_{\pi_i}) \geq \mathcal{T}_{\pi_i}(Q_{\pi_i}) = Q_{\pi_i}.$$

Then, we can repeatedly apply $\mathcal{T}_{\pi_{i+1}}$ to both sides, and based on the monotonic property, the inequality always holds

$$(\mathcal{T}_{\pi_{i+1}})^{\infty}(Q_{\pi_i}) \geq Q_{\pi_i}.$$

The left-hand side of the above equation is the opponent's MDP value-iteration process. Based on the γ-contracting and fixed-point properties, it converges to $Q_{\pi_{i+1}}$, and thus, we have $Q_{\pi_{i+1}} \geq Q_{\pi_i}$. The inequality holds for all i, so the produced $\{Q_{\pi_i}\}$ is a monotonically increasing sequence and, finally, converges to Q^*. The corresponding policy sequence is also convergent, $\lim_{i\to\infty} \pi_i = \pi^*$.

In single-agent MDPs, PI is widely used to solve for the optimal value and optimal policy [30], [33]. However, the policy evaluation in MDPs is a linear equation in values, while, in TZMGs, it is nonlinear in Q_{π_i} due to the nonlinearity of $\mathcal{T}_{\pi_i}$. The policy evaluation here is more complicated than the one in MDPs. In addition, policy improvement in the two cases is also different. MDPs improve policy with a maximum operator, while TZMGs use a minimax optimization.

C. Generalized Policy Iteration

Reviewing value iteration (VI), the iterative process can be rewritten as two steps.

1) $\pi_{i+1} = \mathcal{G}(Q_i)$.
2) $Q_{i+1} = \mathcal{T}_{\pi_{i+1}}(Q_i)$.

PI can be rewritten as follows.

1) $\pi_{i+1} = \mathcal{G}(Q_{\pi_i})$.
2) $Q_{\pi_{i+1}} = (\mathcal{T}_{\pi_{i+1}})^{\infty}(Q_{\pi_i})$.

The difference lies in the number of operators $\mathcal{T}_{\pi_{i+1}}$ in the second step. In VI, the operator is performed once, while, in PI, it is repeated infinitely for the accurate evaluation. Even though VI requires less computation per iteration, some studies have shown that PI provides faster convergence.

Between these two types of iterations, there is a third type of DP, GPI [19], which generalizes the two methods. It updates the value for finite times and can be summarized as two steps.

1) $\pi_{i+1} = \mathcal{G}(Q_i)$.
2) $Q_{i+1} = (\mathcal{T}_{\pi_{i+1}})^{n}(Q_i)$.

The second step is an optimistic policy evaluation, and n indicates the number of inner loop evaluating iterations. The evaluation does not converge to the true policy value $Q_{\pi_{i+1}}$ but calculates only an approximate Q_{i+1}. Then, a new policy is generated from the optimistic Q value, and it moves to the next iteration. When $n = 1$, GPI becomes VI. When $n \to \infty$, GPI becomes PI. By choosing the appropriate value of n, we can find the balance that combines both less computation of VI and faster convergence of PI. In original PI, we have shown that $\{Q_{\pi_i}\}$ is monotonically increasing. This property is still preserved in GPI under certain conditions.

Theorem 1: If $Q_0 \leq \mathcal{T}(Q_0)$, the sequence $\{Q_i\}$ generated by GPI is monotonically increasing and, finally, converges to Q^*.

Proof: With the initial condition and the definition of $\mathcal{T}$, we have

$$Q_0 \leq \mathcal{T}(Q_0) = \mathcal{T}_{\pi_1}(Q_0) \leq (\mathcal{T}_{\pi_1})^2(Q_0).$$

The second inequality is obtained by applying $\mathcal{T}_{\pi_1}$ to the first inequality and following its monotonicity. Then, repeatedly, we have

$$Q_0 \leq \mathcal{T}_{\pi_1}(Q_0) \leq \cdots \leq (\mathcal{T}_{\pi_1})^n(Q_0) \leq (\mathcal{T}_{\pi_1})^{n+1}(Q_0).$$

Note that the last term is bounded by $(\mathcal{T}_{\pi_1})^{n+1}(Q_0) \leq \mathcal{T}(\mathcal{T}_{\pi_1})^n(Q_0)$, so we have

$$Q_0 \leq Q_1 \leq \mathcal{T}(Q_1).$$

In addition, $Q_0 \leq (\mathcal{T})^{\infty}(Q_0) = Q^*$. Then, by induction, $\{Q_i\}$ is monotonically increasing and has an upper bound

$$Q_0 \leq Q_1 \leq \cdots \leq Q_i \leq Q_{i+1} \leq \mathcal{T}(Q_{i+1}) \leq Q^*.$$

The sequence is convergent, and the converged Q_∞ satisfies

$$\pi_\infty = \mathcal{G}(Q_\infty)$$
$$Q_\infty = (\mathcal{T}_{\pi_\infty})^n(Q_\infty) = (\mathcal{T}_{\pi_\infty})^{2n}(Q_\infty) = \cdots = Q_{\pi_\infty}$$

so Q_{π_∞} is also the fixed point of $\mathcal{T}$

$$Q_{\pi_\infty} = \mathcal{T}_{\pi_\infty}(Q_{\pi_\infty}) = \mathcal{T}(Q_{\pi_\infty}).$$

Based on the uniqueness of $\mathcal{T}$, we conclude $Q_{\pi_\infty} = Q^*$. ■

Theorem 1 shows that, as a generalization of VI and PI, GPI still possesses the monotone property under certain conditions. In [19], a more general theorem on the convergence of GPI was proven, and we present it in the Appendix for ease of reference. Regardless of function approximation, Theorem 2 gives a simpler error bound between Q_i and Q^*, which directly follows the original full theorem. With the increase in iterations, the error eventually reduces to zero. The used σ-weighted L_p-norm measures the error bound and is defined by $\|h\|_{p,\sigma} = (\sum_x |h(x)|^p \sigma(x))^{(1/p)}$ for a function h and a distribution σ.

Theorem 2: Let ρ and σ be distributions over state-action pairs. Let p, q, and q' be such that $(1/q)+(1/q') = 1$. Starting from Q_0, after i iterations of GPI, we have

$$\|Q^* - Q_{\pi_i}\|_{p,\rho} \leq \frac{2\gamma^i}{1-\gamma}\left(C_q^{i,i+1,0}\right)^{\frac{1}{p}} \min(\|d_0\|_{pq',\sigma}, \|b_0\|_{pq',\sigma})$$

where $d_0 = Q^* - Q_0$ and $b_0 = Q_0 - \mathcal{T}_{\pi_1}(Q_0)$.

IV. Minimax Q Network Learning

DP provides a fundamental approach to solve TZMG Nash equilibrium, but the limitation is obvious. It relies on model dynamics and is only feasible to small-scale problems. In many real-world games, the state space is immense, and the system transition dynamics is not available. The recent development of DRL, especially DQN [21], provides innovative approaches to similar problems in MDPs. Motivated by that, we first introduce NNs to approximate complicated Q functions in TZMGs. Because GPI generalizes both VI and PI, we study the

update of Q approximators in the framework of GPI. With the techniques of minibatch training, experience replay, and target network, the network parameters are trained by game data to reach the Nash equilibrium. Some other DQN techniques [34], [35] are introduced as well to improve the learning efficiency.

A. Neural Networks for Q Function

A direct approach to represent the Q function is in a lookup table, but it is infeasible for large-scale problems. Existing literature has applied approximation techniques, such as factored linear architecture [18] and regression tree [19] to TZMGs, but NN is the most powerful one. Herein, the Q network is defined with the game state being network input and Q values of each (a, o)-pair being network output. The network is denoted by $q(s, a, o|\theta)$ with θ being adjustable parameters.

In approximate GPI, suppose that, at the $(i-1)$th iteration, we have had the NN-based Q function, and the network parameters are denoted by θ_{i-1}. The improved policy is $\pi_i(s) = [\mathcal{G}(q(\theta_{i-1}))](s)$. Start the next optimistic policy evaluation of π_i with the initial $\theta_{i,0} = \theta_{i-1}$. At each inner loop step $1 \le j \le n$, update the Q network over a data set of m samples $\{(s_k, a_k, o_k)\}$ from the product space $\mathcal{S} \times \mathcal{A} \times \mathcal{O}$. The target values and the loss function are defined by

$$y_k = [\mathcal{T}_{\pi_i}(q(\theta_{i,j-1}))](s_k, a_k, o_k)$$
$$L(\theta) = \frac{1}{2m}\sum_k (q(s_k, a_k, o_k|\theta) - y_k)^2.$$

Then, use supervised learning to train network parameters for the best fit of target values

$$\theta_{i,j} = \arg\min_{\theta} \; L(\theta)$$

and continue the next update. After n inner loop steps, we take $\theta_i = \theta_{i,n}$ as the optimistic evaluation of π_i and start the next GPI iteration.

As illustrated in [19], if there exist approximation errors at each update of the Q function, the gaps between the true policy evaluation Q_{π_i} and the minimax Q^* are still bounded. The complete statement of the theorem is presented in the Appendix for reference. It is clearly observed that, when $i \to \infty$, the bound converges to $2\gamma C(1-\gamma^n)/(1-\gamma)^3$, where C is a constant related to model dynamics and approximation errors. The smaller n is, the closer the Nash equilibrium GPI gets. However, it should also be noted that an accurate policy evaluation requires a large n, which is important for the subsequent policy improvement step to make significant progress. To balance the abovementioned two aspects, Perolat *et al.* [19] suggest that, for complicated tasks, one can start approximate GPI with a large n to increase the convergence rate and then gradually reduce the value of n until to 1 for small convergence errors.

B. Online Learning

Since the model is not available, the agent has to interact with the environment and collect online observations to learn its policy. For complicated MDPs, DQN incorporates experience replay and target networks to increase stability and improve data efficiency for online training of deep networks [21]. Motivated by that, we propose the M2QN algorithm that adopts the two important techniques of DQN for the learning of the Nash equilibrium.

Algorithm 1 M2QN Learning Algorithm

Require: initial θ_0, $\theta_{eval} = \theta_0$, $\theta_{target} = \theta_0$, initial state distribution d, experience memory $\mathcal{D} = \emptyset$, target update steps T, inner-loop evaluating iterations n.

initialize the state at $t = 0$ with $s_0 \sim d$
for $t = 0, 1, \ldots$ **do**
  choose own action according to ϵ-greedy policy

$$a_t \sim \begin{cases} \pi_t = [\mathcal{G}(q(\cdot|\theta_t))](s_t) & 1-\epsilon_t \text{ probability} \\ random\ a \in \mathcal{A} & \epsilon_t \text{ probability} \end{cases}$$

  choose opponent action according to $o_t \in \mu_t(s_t)$
  execute a_t and o_t and observe r_{t+1} and s_{t+1}
  store $(s_t, a_t, o_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
  sample a m-size minibatch $\{(s_k, a_k, o_k, r_{k+1}, s_{k+1})\}$ from $\mathcal{D}$
  for all samples calculate target values

$$y_k = r_{k+1} + \gamma \min_{o'} \sum_{a'} \pi_{eval}(a'|s_{k+1}) q(s_{k+1}, a', o'|\theta_{target})$$

  where $\pi_{eval} = \mathcal{G}(q(\cdot|\theta_{eval}))$
  define loss $L(\theta_t) = \frac{1}{2m}\sum_k (q(s_k, a_k, o_k|\theta_t) - y_k)^2$
  train network parameters with gradient descent $\theta_{t+1} = \theta_t - \alpha \frac{\partial L(\theta_t)}{\partial \theta_t}$
  if $t \pmod T == 0$ **then**
    update the target network $\theta_{target} = \theta_{t+1}$
  end if
  if $t \pmod{nT} == 0$ **then**
    update the evaluation network $\theta_{eval} = \theta_{t+1}$
  end if
end for

With the definition of the above Q network, the whole learning process is presented in Algorithm 1. In the framework of GPI, θ_{eval} indicates the parameters of Q_{i-1}, and π_{eval} is the evaluated policy $\pi_{\text{eval}} = \mathcal{G}[q(\theta_{\text{eval}})]$. θ_{target} corresponds to the last inner loop $Q_{i,j-1}$ and is used to calculate target values for the current $Q_{i,j}$ parameterized by θ_t. The current θ_t is trained by minibatch gradient descent over the experience memory $\mathcal{D}$ and then replaces the target θ_{target} after every T steps. After the target network is updated n times, the evaluation is completed, and θ_{eval} is replaced by the current θ_{t+1}. The algorithm continues with the latest networks.

Now, we analyze the complexity of M2QN at each step. At the online action selection stage, the action distribution is solved by linear programming with Q values $q(s_t, s, a|\theta_t)$. The complexity of linear programming varies by different solvers. For the common interior-point method, the complexity is $O(|\mathcal{A}|^{3.5})$. In addition, the Q values $q(s_t, s, a|\theta_t)$ are obtained by a prediction process of NN, and we denote the complexity as $\mathcal{C}_p(1)$. At the training stage, m historical observations are sampled from the experience set. To calculate each target value, one has to solve the next action distribution with an

evaluated policy and predict the next Q values with the target network. Then, a maximum over opponent action set $\mathcal{O}$ is followed. After obtaining all target values, the rest is the fitting of the current Q network to target values, whose complexity can be denoted by $\mathcal{C}_f(m)$. The overall complexity at each step is $O(\mathcal{C}_p(1)+\mathcal{C}_p(m)+m|\mathcal{A}|^{3.5}+m|\mathcal{O}|+\mathcal{C}_f(m))$. If own agent stops training, the complexity is left with online action selection, which is equal to $O(\mathcal{C}_p(1)+|\mathcal{A}|^{3.5})$.

C. Convergence of M2QN in Tabular Representation

To show the convergence of M2QN, we consider the special case where the Q function is explicitly represented in a lookup table and n chooses 1. The evaluation Q_{eval} is then equal to the target Q_{target}, and GPI is reduced to VI. The $\mathcal{T}_{\pi_i}$ operator in the optimistic policy evaluation is simply the minimax operator $\mathcal{T}$.

At the tth step of the training stage, denote the minibatch samples from $\mathcal{D}$ as $\{(s_{k_l^t}, a_{k_l^t}, o_{k_l^t}, r_{k_l^t+1}, s_{k_l^t+1})\}$ with subscript $1 \le l \le m$. The update of Q table according to m samples can be mathematically described by

$$Q_{t+1}(s,a,o) = Q_t(s,a,o) + \sum_{1\le l\le m} \frac{\alpha_t}{m}\chi_{k_l^t}(s,a,o) \times (\overline{\mathcal{T}}_{k_l^t}(Q_{\text{target}}) - Q_t(s,a,o)) \quad \forall s,a,o \tag{4}$$

where $\overline{\mathcal{T}}_{k_l^t}$ specifies that the value is based on the observation of sample k_l^t

$$\overline{\mathcal{T}}_{k_l^t}(Q_{\text{target}}) = r_{k_l^t+1} + \gamma \max_{\pi}\min_{o'}\sum_{a'}\pi(a')Q_{\text{target}}(s_{k_l^t+1}, a', o').$$

$\chi_{k_l^t}(s,a,o)$ is an indicator that produces 1 if (s,a,o) is identical to the lth sample $(s_{k_l^t}, a_{k_l^t}, o_{k_l^t})$ and 0 otherwise. $\alpha_{t,l} \in (0,1)$ is the learning rate. For every T steps, the target Q_{target} is replaced by the latest Q_{t+1}. After iteratively substituting the right-hand side of (4) to the left-hand side, we obtain the changes of Q at every $t = 0, T, 2T, \ldots, iT, (i+1)T, \ldots$

$$Q_{(i+1)T}(s,a,o) = \rho^{iT,(i+1)T}Q_{iT}(s,a,o) + \sum_{\tau=iT}^{(i+1)T-1}\sum_{l=1}^{m} v_l^{\tau,(i+1)T}\overline{\mathcal{T}}_{k_l^\tau}(Q_{iT})$$

where the coefficients are defined by

$$\begin{aligned}
\rho^{t_0,t} &= \left(1-\sum_l \frac{\alpha_{t-1}}{m}\chi_{k_l^{t-1}}\right)\cdots\left(1-\sum_l \frac{\alpha_{t_0}}{m}\chi_{k_l^{t_0}}\right)\\
v_l^{t_0,t} &= \left(1-\sum_l \frac{\alpha_{t-1}}{m}\chi_{k_l^{t-1}}\right)\cdots\left(1-\sum_l \frac{\alpha_{t_0+1}}{m}\chi_{k_l^{t_0+1}}\right)\\
&\quad\times\frac{\alpha_{t_0}}{m}\chi_{k_l^{t_0}}\\
v_l^{t_0+1,t} &= \left(1-\sum_l \frac{\alpha_{t-1}}{m}\chi_{k_l^{t-1}}\right)\cdots\left(1-\sum_l \frac{\alpha_{t_0+2}}{m}\chi_{k_l^{t_0+2}}\right)\\
&\quad\times\frac{\alpha_{t_0+1}}{m}\chi_{k_l^{t_0+1}}\\
&\vdots\\
v_l^{t-1,t} &= \frac{\alpha_{t-1}}{m}\chi_{k_l^{t-1}}.
\end{aligned}$$

One important property of the above coefficients is that

$$1 = \rho^{t_0,t} + \sum_l v_l^{t_0,t} + \sum_l v_l^{t_0+1,t} + \cdots + \sum_l v_l^{t-1,t}.$$

Theorem 3: Given a finite TZMG $(\mathcal{S},\mathcal{A},\mathcal{O},\mathcal{P},\mathcal{R},\gamma)$, the tabular M2QN algorithm with $n=1$ converges w.p.1 to the minimax Q^* at Nash equilibrium as long as

$$\sum_i \beta_i(s,a,o) = \infty, \quad \sum_i \beta_i^2(s,a,o) < \infty \tag{5}$$

for all $(s,a,o) \in \mathcal{S}\times\mathcal{A}\times\mathcal{O}$, where $\beta_i(s,a,o) = 1 - \rho^{iT,(i+1)T}(s,a,o)$.

Proof: The convergence proof is based on the stochastic approximation theory. For any (s,a,o) visited at the training stage between the iTth and $(i+1)T$th steps, define $\Delta_i(s,a,o) = Q_{iT}(s,a,o) - Q^*(s,a,o)$ and

$$F_i(s,a,o) = \frac{1}{\beta_i(s,a,o)}\sum_{\tau=iT}^{(i+1)T-1}\sum_{l=1}^{m} v_l^{\tau,(i+1)T} \times (\overline{\mathcal{T}}_{k_l^\tau}(Q_{iT}) - Q^*(s,a,o)).$$

Then, their update of Q values along the increase in i can be seen as a random process

$$\Delta_{i+1}(s,a,o) = (1-\beta_i(s,a,o))\Delta_i(s,a,o) + \beta_i(s,a,o)F_i(s,a,o).$$

Since $\alpha_t \in (0,1)$, we have $\beta_i \in (0,1)$. Moreover, the expectation of F_i satisfies

$$\begin{aligned}
&\mathbb{E}[F_i(s,a,o)]\\
&= \frac{1}{\beta_i}\sum_{\tau=iT}^{(i+1)T-1}\sum_l v_l^{\tau,(i+1)T}\\
&\quad\times\Bigg(\mathcal{R}(s,a,o) + \gamma\sum_{s'}\mathcal{P}(s'|s,a,o)\\
&\qquad\times\max_{\pi}\min_{o'}\sum_{a'}\pi(a')Q_{iT}(s',a',o') - Q^*(s,a,o)\Bigg)\\
&= [\mathcal{T}(Q_{iT})](s,a,o) - Q^*(s,a,o)
\end{aligned}$$

and the γ-contracting property of $\mathcal{T}$ infers that

$$\|\mathbb{E}[F_i(s,a,o)]\|_\infty \le \gamma\|Q_{iT} - Q^*\|_\infty = \gamma\|\Delta_i\|_\infty.$$

The variance of F_i is bounded by

$$\begin{aligned}
\mathbf{var}[F_i(s,a,o)] &= \mathbb{E}[(F_i(s,a,o))^2] - (\mathbb{E}[F_i(s,a,o)])^2\\
&\le C\left(1+\|\Delta_i\|_W^2\right)
\end{aligned}$$

because of the finiteness of $\mathcal{R}$, Q^*, m, and T. C is a positive constant, and $\|\cdot\|_W$ refers to some weighted maximum norm. Based on the stochastic approximation theory [36], Δ_i converges to zero w.p.1, namely, Q_t converges to Q^* w.p.1. ■

The abovementioned convergence analysis utilizes the γ-contraction of $\mathcal{T}$ when GPI chooses $n=1$. For single-agent MDPs, Scherrer *et al.* [32] have demonstrated that GPI with $n>1$ is not a contraction in any norms. This proposition also holds for TZMGs, so Theorem 3 only considers $n=1$ case. For $n>1$ cases, the convergence of M2QN requires further

investigation or new theorems and is empirically verified by a variety of experiments in Section V.

Another requirement for the convergence is the condition in (5). From the fact that $\beta_i \in (0, 1)$, $\sum_i \beta_i(s, a, o) = \infty$ implies that all (s, a, o)'s are infinitely visited throughout the whole online process. In the algorithm, own player's action set is explored by the ϵ-greedy strategy in which there is a probability to choose random actions. However, the opponent action set still needs to be explored. Littman [4] investigated two approaches to form opponents. One is to use an existing player as the opponent, so μ_t is a predefined policy. However, the opponent may have no exploration capability, degrading the accuracy of the learned Q function, or the exploration is totally random, wasting time on useless actions. When faced with other opponents, own player may perform poorly with the defective Q function. Another approach is self-play in which the opponent adopts the same policy as own player. By playing with itself, the agent can gradually improve its own policy. However, self-play requires the game to be symmetric, that is, the decision process of own player is the same as the one of the opponent. Such a requirement limits its application to nonsymmetric games. Besides, self-play is easy to be stuck at local optima or chase circles in strategy space.

Differing from the abovementioned two approaches, here, we propose an active way for the selection of μ_t. Note that the Q function is capable of not only generating own minimax policy but also synthesizing the opponent minimized policy that is against own one. In the M2QN algorithm, when the own player follows π_{eval}, the opponent worse action is determined based on the current Q_t by

$$\mu_t(s) = \arg\min_o \sum_a \pi_{\text{eval}}(a|s) Q_t(s, a, o).$$

In a given game, such minimized policy acts as the worst opponent and informs own agent the lowest Q value for its current policy. It serves as a training partner that actively finds its own weaknesses. With this partner, the own agent gradually improves its performance and, eventually, approaches the best response to the worst opponent, namely, the Nash equilibrium. In the implementation, the minimized opponent policy adopts ϵ-greedy strategy to ensure that the action set is fully explored.

D. Dueling Network and Double Q-Learning

In [35], a dueling Q network structure is proposed to improve DQN for MDPs. The main concept is to separate a network into two streams. One stream is a scalar that represents the state-dependent value function, while the other is a vector that represents the action-dependent advantage function. These two streams are aggregated at the output layer for the state-action Q function. With the stream separation, the network is capable of learning the value of a state without having to learn the effect of each action and is also capable of determining a greedy action without considering the effect of the state value. Experiments on the Atari problems [35] show that the dueling structure leads to better policy evaluation and consequently better results.

For TZMGs, dueling structure is also applicable to our M2QN. The network structure is presented in Fig. 1.

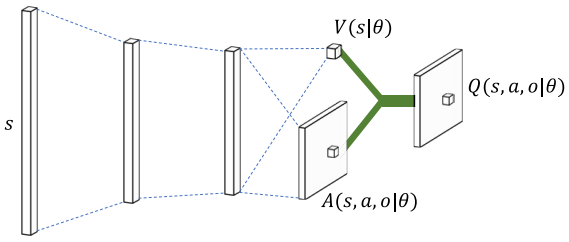


Fig. 1. Dueling network structure in M2QN.

Compared with dueling DQN, the difference lies in the action-dependent stream that outputs an $|\mathcal{A}| \times |\mathcal{O}|$-sized matrix $A(s, a, o)$ for the advantage of (a, o) pair at a given s. The two streams are finally combined to produce the Q function with

$$Q(s, a, o|\theta) = V(s|\theta) + A(s, a, o|\theta) - \frac{1}{|\mathcal{A}|}\frac{1}{|\mathcal{O}|}\sum_{a' \in \mathcal{A}}\sum_{o' \in \mathcal{O}} A(s, a', o'|\theta).$$

The last item in the above equation is to cancel out the state-dependent effect in the advantage stream.

Another issue that is commonly suffered by Q-learning algorithms is the maximization (or minimization) bias. Reviewing the target values at Line 8 in Algorithm 1, the minimum operator uses the same value to both select and to evaluate the opponent action, resulting in minimization bias in target estimates. Double Q-learning [37] proposes to use two different Q functions to decouple the selection from the evaluation, and it has been successfully combined with DQN [34] to reduce overestimation. In order to apply double Q-learning to M2QN, we use the current Q function to select the opponent worst action and use the target Q function to evaluate the worst-action value. The calculation at Line 8 in Algorithm 1 is replaced by

$$o_k^{\min} = \arg\min_{o'} \sum_{a'} \pi_{\text{eval}}(a'|s_{k+1}) q(s_{k+1}, a', o'|\theta_t)$$

$$y_k^{\text{double}} = r_{k+1} + \gamma \sum_{a'} \pi_{\text{eval}}(a'|s_{k+1}) q\left(s_{k+1}, a', o_k^{\min}|\theta_{\text{target}}\right).$$

Then, the gradient-descent update takes y_k^{double} as the target value for each minibatch sample to train the current Q network. In Section V, double Q-learning is performed among all M2QN experiments, and the dueling structure is implemented when NN approximation is required.

V. Experiments

A. Tabular M2QN on Symmetric TZMG: Soccer Game

To test the algorithm, we first consider the soccer game proposed in [4], as shown in Fig. 2. The game is on a board by 4×5, and there are two players, A and B. At the beginning of one game, two players are randomly initialized on two different squares, and a ball is allocated to one player at random, marked by a circle in the figure. At each step, they can choose one of five actions to move: Left, Up, Right, Down, and Stay. Once they have selected their actions, the two moves are executed in random order. If the move would take the player

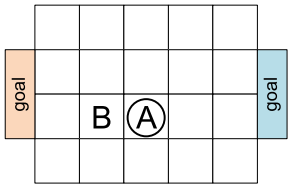


Fig. 2. Soccer game.

to the opponent square, the possession of the ball yields to the stationary player, and the move is canceled. Once a player takes the ball to the appropriate goal (left for A and right for B), that player receives a reward of one point and the game ends. It is a symmetric MG, so the strategy for one player works for the other. To avoid infinite game, every step has a 0.01 probability of being terminated as a draw.

The game states are enumerated by two-player positions and ball possession. The reward function outputs $+1$ if A goals and -1 if B goals. Otherwise, it is zero. The discounted factor selects 0.95. Because the state set is finite and small, DP (e.g., VI) directly solves the Nash equilibrium value and policy. A lookup Q table is adopted for the implementation of our M2QN algorithm. To encourage exploration, the ϵ-greedy exploration rate decreases linearly from 1 to 0.1 over 20 000 steps and is fixed to 0.1 afterward. The maximum size of memory is set to 10 000, and the least size for experience replay is 1000. Each replay chooses a batch size of 16.

M2QN is applied to learn the policy for player A, and the opponent during online learning chooses the training partner from the same Q table. We first set target update steps T to 100 and inner loop evaluating iterations n to 5. After every 10 000 steps, we evaluate the learned policy by playing against DP policy for 1000 rounds and recording their win rates. The effect of exploration is not included in the evaluation. The learning curves of M2QN are presented in Fig. 3, and the results are averaged over three repeated experiments to reduce random errors. DP plays the Nash equilibrium policy, so, at initial steps, the M2QN policy has low win rates. With the increase in learning steps, M2QN uses collected observations to update the Q table and adjust policy. Its win rate rises continually, while DP drops, reflecting that M2QN is getting more and more competitive. The win rates between our M2QN and DP are quite close after 600 000 steps, indicating that M2QN has learned a near Nash-equilibrium policy, even though it is an online model-free algorithm.

Fig. 4 plots the Q values and action probabilities of final M2QN at a given state (as illustrated in Fig. 2). For ease of comparison, the Q values and action probabilities of DP at the same state are also presented. In this game, to avoid losing ball possession and to do better than breaking even, the offensive player must choose a probabilistic policy to reduce the chance of being exploited. As shown in Fig. 4(b), the Nash equilibrium policy for player A at the given state is to choose Up and Stay actions with the same probability. The worst expected return is $0.5 * 0.29 + 0.5 * 0.31 = 0.3$ under the condition that player B chooses Up or Stay action. As to M2QN, it learns the Q table based on observations in contrast to model-based DP, so its Q values and action

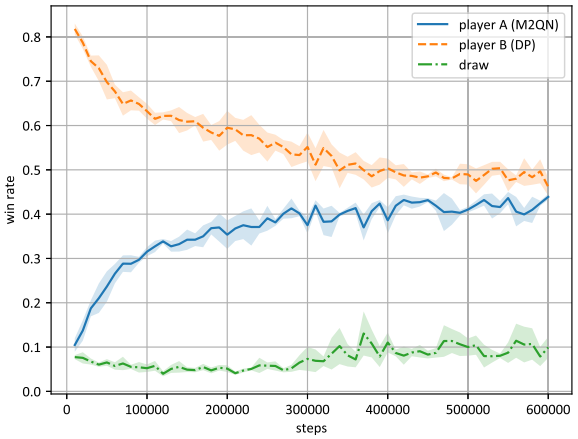
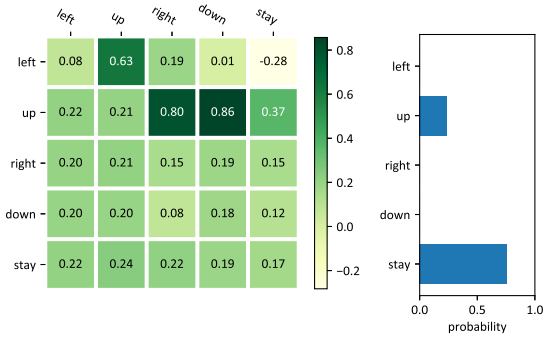
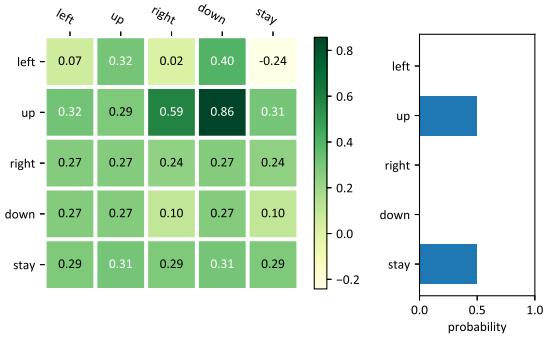


Fig. 3. Evaluation of M2QN policy throughout training. The x-axis shows online learning steps. The y-axis shows win rates between the learned policy and DP policy.



(a)



(b)

Fig. 4. Q values and action distributions of player A by (a) M2QN and (b) DP. The row of Q matrix represents action entry for player A, and the column represents action entry for player B. The action distributions are solved by linear programming according to their Q matrices.

probabilities are not exactly equal to DP. However, the shape of its policy is consistent with DP, and the worst expected return is $0.24 \times 0.31 + 0.76 \times 0.29 = 0.2948$ on the condition that player B chooses Stay. It is just slightly lower than the best 0.3.

Now, to investigate the impact of algorithm parameters on the learning process, we repeat the abovementioned experiment but with different target update steps T and inner loop evaluating iterations n. The results are depicted in Fig. 5. It shows win rates of learned M2QN policies against DP policy

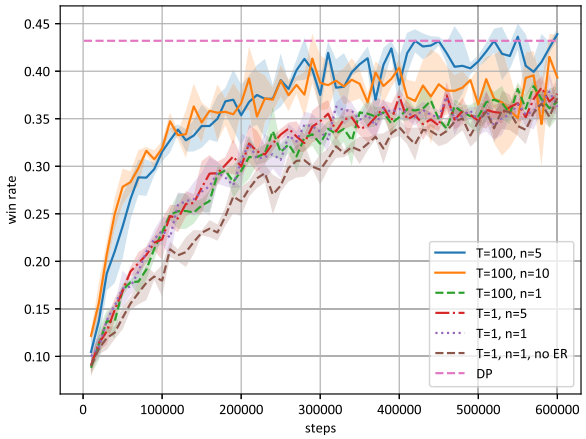


Fig. 5. Impact of algorithm parameters to the M2QN learning. The y-axis shows the win rate of learned policy tested against DP policy. The horizontal line represents the win rate of DP policy against itself.

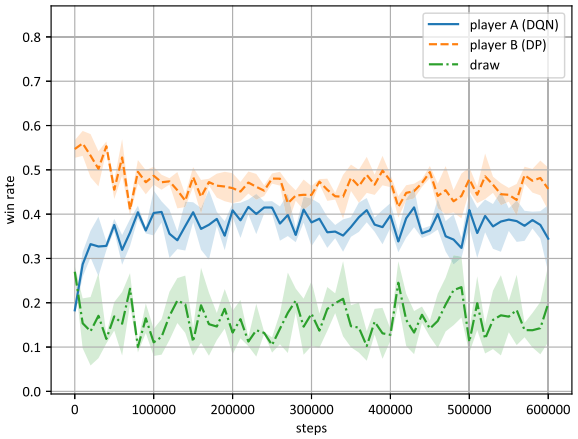


Fig. 6. Evaluation of player A policy throughout self-play DQN training. The x-axis shows online learning steps. The y-axis shows win rates between the learned policy and DP policy.

along with learning steps. For comparison, the win rate of DP policy against itself is also plotted as a horizontal line (dashed pink). Due to the probability of draw termination at each step, the win rate of DP against itself is 0.432, not 0.5. However, still, its Nash equilibrium policy has the highest win rate, and all M2QN learning curves are approaching it. Note that the curve of M2QN with $T = 100$ and $n = 10$ (orange) has the fastest rising speed at the initial stage, but the curve with $T = 100$ and $n = 5$ (blue) finally converges to DP performance most closely. It is consistent with the GPI convergence theorem stating that a large n speeds up the initial learning, but a small n minimizes the final convergence error. The plot also shows that continuously reducing T or n further slows down the learning process.

Note that the minimax-Q algorithm proposed in [4] can be seen as a special case of our M2QN with $T = 1$ and $n = 1$ but without experience replay. We also repeat the algorithm to the soccer game and plot the result (dashed brown) in Fig. 5. Even though the final results are close to experience-replay ones, the climbing speed is slow. With experience replay, the online learning algorithm updates Q values more efficiently by repeatedly utilizing history.

In the literature, the conventional single-agent RL algorithm is combined with self-play to deal with game problems. In this experiment, we combine DQN with self-play to learn the player policy for a soccer game. The target update frequency chooses the same $T = 100$ steps, and the rest parameters keep the same as M2QN. The learning curves of self-play DQN are depicted in Fig. 6. It is clear to see that self-play DQN improves faster than M2QN, but the curve shows high vibration after win rate climbs to a certain level, reflecting the convergence of self-play is not stable.

Now, to show the exploitability of different algorithms, we design a series of test experiments to train DQN challengers against each of the DP policy, final M2QN policies ($T = 100, n = 5$), and final self-play DQN policies. In each test, player A policy is fixed, so the game is reduced to a single-agent MDP problem, and DQN can learn the best response of player B to its opponent. We record the average win rates of DQN challengers over 10 000 online steps and plot

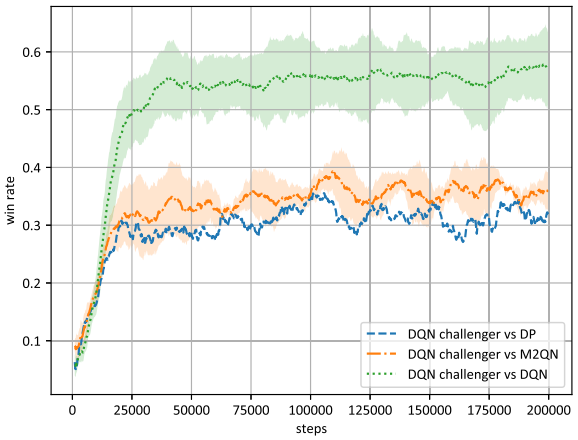


Fig. 7. Win rates of DQN challengers during online learning against fixed player A with DP policy, final M2QN policies, and final self-play DQN policies.

the test results in Fig. 7. In the beginning, DQN challengers start from very low win rates. With the increase in learning steps, their win rates rise and quickly converge. The challengers versus self-play DQNs finally have much higher win rates than the others, showing that self-play DQN policies have the highest exploitability. Challengers versus M2QN policies have slightly higher win rates than the one versus DP policy, indicating M2QN learns near Nash equilibrium. The advantage of the Nash equilibrium lies in that it is the most "safe" policy even against the worst opponent. In many cases, the player is facing an intelligent opponent that continually adapts its behaviors. A vulnerable player is easily misguided to a poor policy and lets the opponent exploit it.

In the above experiments, Q values of all state-action pairs are stored in a lookup table, whose update suffers from a heavy computational burden. NNs approximate Q functions with much fewer parameters, so we repeat the M2QN algorithm but with NNs on a soccer game. The network input is composed of normalized positions of two players and a Boolean value indicating ball owner. The input layer is followed by two hidden layers with the same layer size of 50, and the output

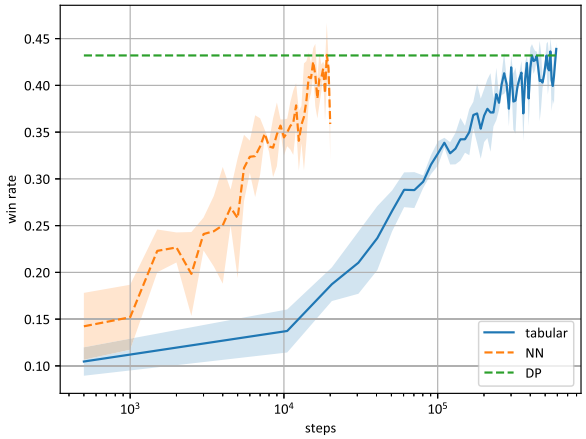


Fig. 8. Comparison of NN-based M2QN and tabular M2QN learning curves. The x-axis shows the logarithmic learning steps, and the y-axis shows the evaluated win rates of learned policies against DP policy. The horizontal line represents the win rate of DP policy against itself.

corresponds to the Q values of each (a, o) pair. A dueling structure is adopted in the network. The algorithmic parameters choose $T = 100$ and $n = 5$. The training uses the Huber loss function, the batch size chooses 16, and the learning rate is 0.001. Double Q-learning is applied in the algorithm. Fig. 8 depicts the NN-based M2QN learning curve along with the logarithmic learning steps, and for comparison, the result of tabular M2QN with the same T and n is also presented. Due to the approximation of NNs, M2QN learns policies more than ten times faster than in a lookup table. This speedup is especially important for large-scale games to significantly reduce learning steps. NNs generalize the update of one state-action pair to others and, therefore, increase sample efficiency. However, NN approximation errors have a negative effect on the convergence, which is reflected in the plot that NN-based M2QN has obvious fluctuation at the final stage than tabular M2QN. For large-scale problems, such sacrifice of accuracy to economical computation is often preferable.

B. NN-Based M2QN on Nonsymmetric TZMG: Guarding a Territory

The second case considers guarding a territory in a grid world. The problem was first proposed in [38] for continuous space and then modified by [39] for a discrete grid world. As shown in Fig. 9, there is an invader and a guard in the grid, and at each step, they choose one of five actions to move: Left, Up, Right, Down, and Stay. The goal of the invader is a cell marked as territory. An invasion succeeds if the invader reaches the territory before being captured or the capture happens at the territory. The guard aims to intercept the invader as far from the territory as possible. The nine cells centered on the guard constitute its capture area. The game ends when the successful invasion or capture happens, or the current step is determined as a draw with a probability of 0.01. Then, the positions of guard and invader are randomly reset for the next round. The game is nonsymmetric since two-side players require totally different strategies.

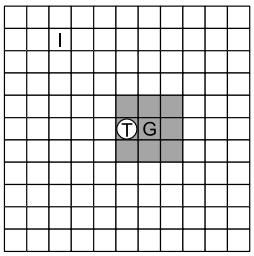


Fig. 9. Guarding a territory in a grid world. "I" represents the invader, "G" represents the guard, and "T" represents the territory. The gray squares around guard represent its capture area.

In our experiment, we consider the grid with 11×11 size. The reward at the perspective of guard is defined as

$$r_{t+1} = \begin{cases} -10, & \text{if invasion succeeds} \\ |x_I - x_T| + |y_I - y_T|, & \text{if capture succeeds} \\ 0, & \text{otherwise} \end{cases}$$

where the first condition chooses a negative value as invasion reward, and the second condition uses the Manhattan distance between invader and territory as capture reward. The discounted factor chooses $\gamma = 0.95$. The state space is larger than the soccer game, but DP is still feasible, so we take DP invader as a baseline. Tabular M2QN consumes too much time to learn, so we turn to NN-based M2QN. A three-layer network is defined for the Q function. The input is composed of horizontal and vertical distances of guard and invader to territory and is connected to two hidden layers of both size 50. Further adding layers and nodes in the network is not necessary and, in turn, increases the number of learnable weights. Each layer is followed by ReLU activation, except the output layer uses linear activation. The training uses the Huber loss function, the batch size chooses 16, and the learning rate is 0.001. Experience memory has a maximum length of 20 000. The ϵ-greedy exploration starts from 1 and linearly decreases to 0.1 after 40 000 steps.

Now, we apply M2QN to learn the policy for the guard. The training partner from the same Q network acts as the opponent during online learning. Fig. 10 shows the plots of the performance of M2QN guard throughout training, as a function of learning steps, on the evaluation against DP invader for 1000 rounds. The results are obtained under $T = 100$ and $n = 5$ and are averaged over three experiments. The mean reward and win rates between DP guard and DP invader are also presented in the same figure. M2QN achieves a performance close to DP, demonstrating that, in spite of the model-free constraint and NN approximation, the algorithm still learns the near Nash equilibrium based on online data. We also try different values of T and n, but find that there is no significant influence on the curves. The similar performance can be explained by the generalization of NNs. Since the problem is not that complicated, the minibatch training at one step is sufficient for the network to make improvements. Smaller T and n make the algorithm iterate more frequently, while larger T and n let each iteration achieve more improvement. It should be noted in the figure that the final win rate of the M2QN guard is higher than 50% because the game is nonsymmetric

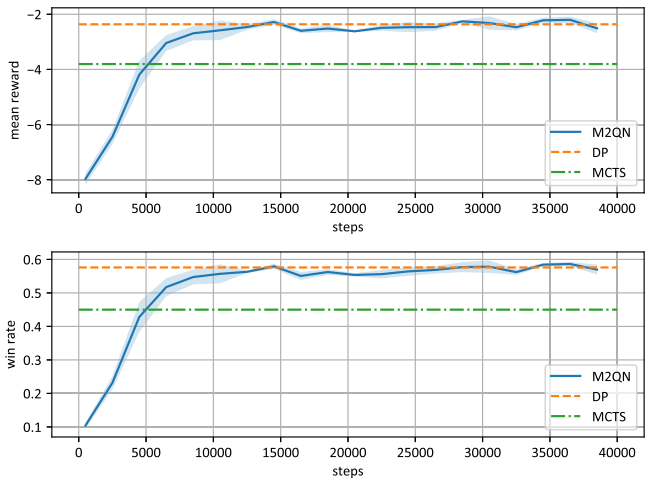


Fig. 10. Evaluation of M2QN guard policy throughout the training. The x-axis shows online learning steps. The y-axis results are established by testing the learned policy against the invader policy found by DP. The top plot gives the mean reward that the guard receives every round, and the bottom plot shows the win rates of the learned guard. The two horizontal lines are the win rates of DP guard (dashed orange) and MCTS guard (dash-dotted green) against DP invader, respectively.

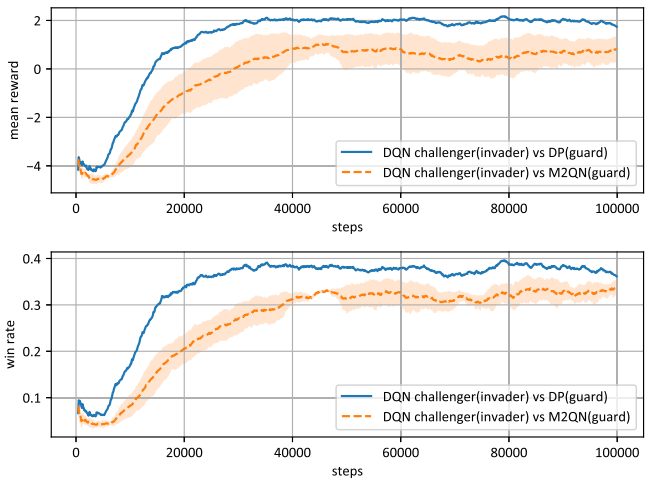


Fig. 11. Mean rewards and win rates of challenging DQN invaders during online learning against fixed guards with DP policy and final M2QN policies.

and the inflated capture area makes capture easier than the precise invasion.

Due to the nonsymmetry, self-play is not applicable to learn a guard policy in the game. Instead, we take the model-based Monte Carlo tree search (MCTS) as a comparison. At each step, MCTS makes 1000 simulations and chooses the node with the most visits as the guard action. The evaluation of MCTS guard against DP invader in Fig. 10 shows that MCTS performs worse than DP and M2QN. One reason is that MCTS is primarily designed for turn-based games, where one player's best action is conditioned on the other move. However, for MGs where two players simultaneously act, the best action for one side may be stochastic. Another reason is that MCTS relies on rollouts to evaluate action performance. The accurate evaluation requires considerable rollouts, but it alternatively increases the decision time, limiting the application to finite-time decision scenarios.

We also design a series of test experiments to study the exploitability of the above-learned policies. In each test, the guard policy is fixed to one of the learned policies by M2QN and DP, and a challenging invader is trained with DQN to learn the best response. The target update steps of DQN choose 100, and the rest parameters follow the same as M2QN. The learning process of DQN challengers is recorded in Fig. 11. The curves plot one-round mean rewards and win rates of challenging invaders over past 10 000 online steps. In both tests, the DQN challengers continuously optimize the invader performance, reflected in the rise of mean reward and win rate until convergence. An interesting observation is that the challenging invader against M2QN shows lower scores and slower rising curves. After a deep investigation, we find that the test with M2QN has a longer round length than the test with DP, meaning that the M2QN guard is slightly less efficient in capturing challenging invader than the DP guard. As a result, there are more possibilities to terminate the game as a draw, which dilutes the reward and win rate of the DQN challenger. However, this experiment is sufficient to show that NN-based M2QN still learns a policy that is not easily exploited by opponents.

Note that, in the abovementioned M2QN learning, we all use the training partner μ_t to produce opponent actions. The training partner actively finds weaknesses of the current M2QN agent and, in the meantime, keeps exploration with the ϵ-greedy policy. The generated online observations are capable of training the policy to the Nash equilibrium. To illustrate the effectiveness of such an active opponent, we repeat the M2QN experiment but replace the opponent with the DP agent and a random agent, respectively. Fig. 12 shows the learning curves of M2QN with three different partners. Even though the DP partner uses the Nash equilibrium policy, the M2QN agent fails in maintaining the learned policy at the equilibrium. In addition, the rising rate of the curve with DP partner is slower than the others at the early stage. It is because the fixed DP policy is not capable of exploration in the opponent action space. The learned Q function is not accurate enough to find the equilibrium policy. In contrast, the M2QN partner and the random partner fully explore opponent actions, and the algorithm converges to the Nash equilibrium. However, the random agent selects actions at random, wasting resources on these totally useless actions. The M2QN partner balances the exploitation and exploration by combining the current minimized policy with the ϵ-greedy strategy and gradually reducing the ϵ value with the increase in learning steps.

C. NN-Based M2QN on Real-Time and Large-Scale TZMG: Fighting Game

In the third case, we test the proposed algorithm on the real-time fighting game and choose the FightingICE[1] as the game environment (as shown in Fig. 13) [40]. There are two players in the game, and each controls a character to fight. There are up to 41 candidate actions that a player can

[1]http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index.htm

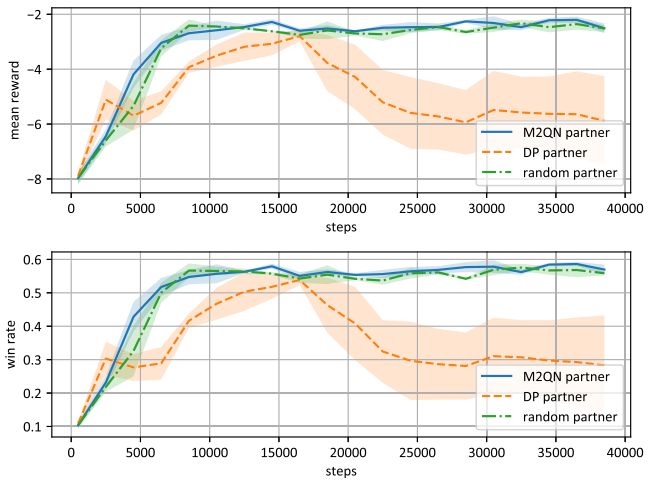


Fig. 12. Learning curves of M2QN in the guarding-a-territory game with M2QN partner, DP partner, and random partner, respectively. The mean reward and win rate at any steps are evaluated by a DP invader.

Fig. 13. FightingICE screenshot.

choose, but the decision time is limited to 16.67 ms. Each action has a series of frames to animate, and the player can make a decision only after the last action animation is completed. The opponent will drop a certain number of health points if the attack action succeeds in hitting it. One property of FightingICE is that it provides a simulator that is capable of rolling out two-side actions from any given frame. Because of that, tree-based search methods have been widely used in designing fighting AIs and have shown promising performance [41], [42]. However, here, we apply our model-free M2QN algorithm and learn a fighting AI from scratch.

To convert the problem to an MG, we first define a state space with a dimension of 163. The state vector is composed of current own and opponent player information, including position, energy, speed, game state, and action. The action set includes 41 candidate actions for both sides. The difference of two-side dropped health points at each frame is selected as the reward signal, and the reward is discounted after one frame with a factor of 0.998.

Compared with a soccer game and guarding-a-territory game, fighting game is more difficult because of the large-scale state space and action set and the complicated model dynamics. Hence, more hidden layers and larger layer sizes should be used. A Q network with three hidden layers is adopted, and from the input side to the output side, the hidden layers have 500, 500, and 300 hidden nodes, respectively. During the game, whenever the player has a chance to make a decision, M2QN predicts the Q values according to the current state and decides the minimax action. However, there is also a probability of ϵ selecting random actions to explore the action set. The exploration rate decreases linearly from 1 to 0.1 after 750 game rounds and then remains 0.1 afterward. Once the action is commanded, the algorithm records the current state, own action, and opponent action for experience replay. The experience memory also stores all rewards received between two successive decision times. The memory has a maximum length of 100 000. For ease of training, FightingICE is set to unlimited game mode, in which two players have infinite health points, and each round lasts 60 s.

Compared with the abovementioned two game problems, the fighting game faces more challenges because the actions are effective only if the opponent is within the attack range. Otherwise, the actions are wasted. Moreover, some actions can be performed only if the player has a certain amount of energy. At the beginning of a round, the player has zero energy. As the fight proceeds, the player collects energy when it hits the opponent or receives damages. When enough energy is collected, the player is capable of performing advanced actions that cause more damages to the opponent. However, for training, it is difficult for the M2QN player to observe the M2QN partner performing advanced actions at the early stage since the M2QN partner is also trained from scratch. Skillful opponents make it easier for the M2QN player to learn how to deal with advanced actions. Therefore, in this experiment, four MCTS-based fighting AIs are adopted as training partners in addition to the M2QN partner. A mixed group of opponents facilitates the learning process and helps to evaluate the current M2QN performance. In each game, there is a probability of 0.5 selecting the M2QN partner as an opponent. Otherwise, the rest AIs have an equal probability of being selected. Besides, we randomly change the player sides for fair competition. The training uses a batchsize of 32 and a learning rate of 0.0001.

Throughout the training, the target update steps always choose $T = 2000$. At the early stage, to speed up the learning process, we choose a large policy evaluating iterations $n = 5$. After 1125 rounds of training, n drops to 1 to reduce convergence error. We record the game results and health differences after each round and plot the evolution of M2QN performance against four fighting AIs in Fig. 14. The curves are smoothed by averaging win rates and health differences over 120 rounds during training. At the initial stage, since M2QN learns the fighting policy from scratch, the win rates are zero, and the health is far below opponents. With the increase in training rounds, win rates rise gradually, and health differences change from negative to positive values after a certain number of rounds. It is observed that, faced with different fighting AIs, M2QN requires different numbers of rounds to increase the corresponding win rates and health differences. Even though these bots are all based on MCTS, they differ in some details of implementation, such as search depth, pruning, and scripted-based actions at certain states. Finally, the M2QN player beats all AI opponents with far more

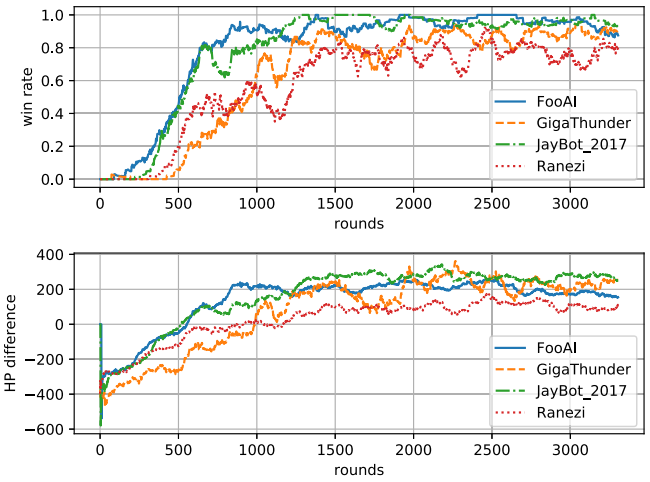


Fig. 14. Evolution of M2QN against AI opponents throughout the training. The x-axis shows training rounds. The y-axis shows win rates and health differences that are averaged over 120 rounds during training. The four AI opponents are GigaThunder, FooAI, jayBot_2017, and Ranezi. They are, respectively, the first-, second-, and third-place winners of the 2017 Fighting Game AI competition and the second-place winner of the 2016 Fighting Game AI competition. All of them are designed based on MCTS.

than 50% win rates. It demonstrates that, even for complicated real-time games with huge state space and action sets, our M2QN still learns competitive policies from game data.

VI. Conclusion

A two-player zero-sum game is always a challenging task in AI. In this article, we combine game theory with recent DQN techniques to deal with TZMGs that require simultaneous decision-making of two sides. The proposed M2QN algorithm is developed from GPI and combines with NNs for large-scale problems. It learns the Nash equilibrium with game data in a model-free way and is applicable to both symmetric and nonsymmetric games. The online convergence of M2QN with tabular Q representation is proven with stochastic approximation theory. Experimental examples validate that M2QN is applicable to a variety of TZMG problems.

One limitation of the current algorithm is that it only considers games with finite action sets. Some DRL algorithms for MDPs, such as DDPG, PPO, and TRPO, define a policy network for large-scale action set or continuous action space. Their policy parameters are adjusted along the policy gradient based on return or evaluated value. However, when extending these algorithms to TZMGs, the biggest difficulty is the estimation of an accurate policy gradient. In MGs, the value or return of one player policy is conditioned on the other behaviors, making the policy gradient noisy and inconsistent. In the future, we plan to investigate a stable and unbiased policy gradient estimation for TZMGs.

Appendix

Theorem 4 (Convergence of Approximate GPI [19]): Consider applying approximate GPI to TZMG problems with an initial value $\hat{Q}_0$. At each outer loop iteration, given last approximated $\hat{Q}_{i-1}$, update the policy by $\pi_i = \mathcal{G}(\hat{Q}_{i-1})$. Then, let $\hat{Q}_{i,0} = \hat{Q}_{i-1}$, and perform the optimistic policy evaluation with approximation

$$\hat{Q}_{i,j} = \text{Fit}(\mathcal{T}_{\pi_i}(\hat{Q}_{i,j-1})), \quad j = 1, \ldots, n$$

where $\text{Fit}(\cdot)$ indicates a mapping from input functions to approximation space with the best fit. The approximation error of the jth inner loop update is denoted by $\epsilon_{i,j} = \hat{Q}_{i,j+1} - \mathcal{T}_{\pi_i}(\hat{Q}_{i,j})$. After n times of inner loop update, the next $(i+1)$th iteration starts with the new $\hat{Q}_i = \hat{Q}_{i,n}$. Let ρ and σ be distributions over state-action pairs. Let p, q, and q' be such that $(1/q) + (1/q') = 1$. Starting from $\hat{Q}_0$, after i iterations of approximate GPI, the Nash equilibrium error has

$$\begin{aligned}\|Q^* - Q_{\pi_i}\|_{p,\rho} &\le \frac{2(\gamma - \gamma^i)(1-\gamma^n)}{(1-\gamma)^3}\left(\mathcal{C}_q^{1,i,0,n,0}\right)^{\frac{1}{p}} \sup_{l,j} \|\epsilon_{l,j}\|_{pq',\sigma} \\ &\quad + \frac{2\gamma^i}{1-\gamma}\left(\mathcal{C}_q^{i,i+1,0}\right)^{\frac{1}{p}} \min(\|d_0\|_{pq',\sigma}, \|b_0\|_{pq',\sigma})\end{aligned}$$

where Q^* is the minimax Q value of the game and Q_{π_i} is the exact Q value of policy π_i. In addition, $d_0 = Q^* - \hat{Q}_0$, $b_0 = \hat{Q}_0 - \mathcal{T}_{\pi_1}(\hat{Q}_0)$, and

$$\mathcal{C}_q^{j,i,d} = \frac{(1-\gamma)^2}{\gamma^j - \gamma^i} \sum_{l=j}^{i-1} \sum_{k=l}^{\infty} \gamma^k c_q(k+d)$$

$$\mathcal{C}_q^{j,i,j',i',d} = \frac{(1-\gamma)^3}{(\gamma^j - \gamma^i)(\gamma^{j'} - \gamma^{i'})} \sum_{l=j}^{i-1} \sum_{l'=j'}^{i'-1} \sum_{k=l+l'}^{\infty} \gamma^k c_q(k+d)$$

with the norm of a Radon–Nikodym derivative[2]

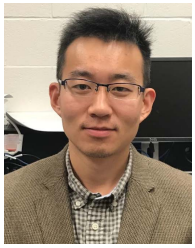
$$c_q(k) = \sup_{\pi_1,\mu_1,\ldots,\pi_k,\mu_k} \left\| \frac{d(\rho \mathcal{P}_{\pi_1,\mu_1} \ldots \mathcal{P}_{\pi_k,\mu_k})}{d\sigma} \right\|_{q,\sigma}.$$

References

[1] R. A. Howard, *Dynamic Programming and Markov Processes*. Hoboken, NJ, USA: Wiley, 1960.
[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[3] L. S. Shapley, "Stochastic games," *Proc. Nat. Acad. Sci. USA*, vol. 39, no. 10, pp. 1095–1100, Oct. 1953.
[4] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings 1994*, W. W. Cohen and H. Hirsh, Eds. San Francisco, CA, USA: Morgan Kaufmann, 1994, pp. 157–163.
[5] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.
[6] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 3, no. 1, pp. 73–84, Feb. 2019.
[7] H. Soo Chang, J. Hu, M. C. Fu, and S. I. Marcus, "Adaptive adversarial multi-armed bandit approach to two-person zero-sum Markov games," *IEEE Trans. Autom. Control*, vol. 55, no. 2, pp. 463–468, Feb. 2010.
[8] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.
[9] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[10] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[2] $\mathcal{P}_{\pi,\mu}(s'|s) = \sum_a \sum_o \pi(a|s)\mu(o|s)\mathcal{P}(s'|s,a,o)$ is the stochastic kernel that characterizes the transition probability from s to s' by two player policies π and μ.

[11] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.
[12] R. B. Myerson, *Game Theory*. Cambridge, MA, USA: Harvard Univ. Press, 2013.
[13] A. J. van der Schaft, "L2-gain analysis of nonlinear systems and nonlinear state-feedback H_∞ control," *IEEE Trans. Autom. Control*, vol. 37, no. 6, pp. 770–784, Jun. 1992.
[14] Y. Zhu, D. Zhao, and X. Li, "Iterative adaptive dynamic programming for solving unknown nonlinear zero-sum game based on online data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 714–725, Mar. 2017.
[15] Y. Zhu, D. Zhao, X. Yang, and Q. Zhang, "Policy iteration for H_∞ optimal control of polynomial nonlinear systems via sum of squares programming," *IEEE Trans. Cybern.*, vol. 48, no. 2, pp. 500–509, Feb. 2018.
[16] Q. Wei, D. Liu, Q. Lin, and R. Song, "Adaptive dynamic programming for discrete-time zero-sum games," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 4, pp. 957–969, Apr. 2018.
[17] S. D. Patek, "Stochastic and shortest path games: Theory and algorithms," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, USA, 1997.
[18] M. G. Lagoudakis and R. Parr, "Value function approximation in zero-sum Markov games," in *Proc. 18th Conf. Uncertainty Artif. Intell. (UAI)*. San Francisco, CA, USA: Morgan Kaufmann, 2002, pp. 283–292.
[19] J. Perolat, B. Scherrer, B. Piot, and O. Pietquin, "Approximate dynamic programming for two-player zero-sum Markov games," in *Proc. 32nd Int. Conf. Mach. Learn.*, in Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., Lille, France, vol. 37, Jul. 2015, pp. 1321–1329.
[20] Z. Tang, K. Shao, D. Zhao, and Y. Zhu, "Recent progress of deep reinforcement learning: From AlphaGo to AlphaGo zero," *Control Theory Appl.*, vol. 34, no. 12, pp. 1529–1546, 2017.
[21] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
[22] Y. Zhu and D. Zhao, "Vision-based control in the open racing car simulator with deep and reinforcement learning," *J. Ambient Intell. Humanized Comput.*, pp. 1–13, Sep. 2019, doi: 10.1007/s12652-019-01503-y.
[23] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," 2016, *arXiv:1603.01121*. [Online]. Available: http://arxiv.org/abs/1603.01121
[24] G. Tesauro, "Temporal difference learning and TD-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.
[25] M. Bowling and M. Veloso, "Rational and convergent learning in stochastic games," in *Proc. 17th Int. Joint Conf. Artif. Intell. (IJCAI)*, vol. 2. San Francisco, CA, USA: Morgan Kaufmann, 2001, pp. 1021–1026.
[26] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019.
[27] D. Fudenberg and D. K. Levine, *The Theory of Learning in Games*. Cambridge, MA, USA: MIT Press, 1998.
[28] J. F. Nash, "Equilibrium points in n-person games," *Proc. Nat. Acad. Sci. USA*, vol. 36, no. 1, pp. 48–49, Jan. 1950.
[29] D. Zhao and Y. Zhu, "MEC—A near-optimal online reinforcement learning algorithm for continuous deterministic systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 2, pp. 346–356, Feb. 2015.
[30] Y. Zhu, D. Zhao, and H. He, "Invariant adaptive dynamic programming for discrete-time optimal control," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 50, no. 11, pp. 3959–3971, Nov. 2020.
[31] D. P. Bertsekas, "Value and policy iterations in optimal control and adaptive dynamic programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 500–509, Mar. 2017.
[32] B. Scherrer, M. Ghavamzadeh, V. Gabillon, and M. Geist, "Approximate modified policy iteration," in *Proc. 29th Int. Conf. Int. Conf. Mach. Learn.*, 2012, pp. 1889–1896.
[33] Y. Zhu and D. Zhao, "Comprehensive comparison of online ADP algorithms for continuous-time optimal control," *Artif. Intell. Rev.*, vol. 49, no. 4, pp. 531–547, Apr. 2018.
[34] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell. (AAAI)*, 2016, pp. 2094–2100.
[35] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, 2016, pp. 1995–2003.
[36] T. Jaakkola, M. I. Jordan, and S. P. Singh, "Convergence of stochastic iterative dynamic programming algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 1994, pp. 703–710.
[37] H. V. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Red Hook, NY, USA: Curran Associates, 2010, pp. 2613–2621.
[38] R. Isaacs, *Differential Games: A Mathematical Theory With Applications to Warfare and Pursuit, Control and Optimization*. New York, NY, USA: Wiley, 1965.
[39] X. Lu and H. M. Schwartz, "An investigation of guarding a territory problem in a grid world," in *Proc. Amer. Control Conf.*, Jun. 2010, pp. 3204–3210.
[40] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting game artificial intelligence competition platform," in *Proc. IEEE 2nd Global Conf. Consum. Electron. (GCCE)*, Oct. 2013, pp. 320–323.
[41] S. Yoshida, M. Ishihara, T. Miyazaki, Y. Nakagawa, T. Harada, and R. Thawonmas, "Application of Monte-Carlo tree search in a fighting game AI," in *Proc. IEEE 5th Global Conf. Consum. Electron.*, Oct. 2016, pp. 1–2.
[42] M. Ishihara, T. Miyazaki, C. Y. Chu, T. Harada, and R. Thawonmas, "Applying and improving Monte-Carlo tree search in a fighting game AI," in *Proc. 13th Int. Conf. Adv. Comput. Entertainment Technol. (ACE)*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1–6.



Yuanheng Zhu (Member, IEEE) received the B.S. degree in automation from Nanjing University, Nanjing, China, in 2010, and the Ph.D. degree in control theory and control engineering from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2015.

From 2015 to 2017, he was an Assistant Professor with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, where he is currently an Associate Professor. From 2017 to 2018, he was a Visiting Scholar with the Department of Electrical, Computer, and Biomedical Engineering, University of Rhode Island, Kingston, RI, USA. His current research interests include deep reinforcement learning, game theory, game intelligence, and multiagent learning.

Dr. Zhu has served as the Chair of the IEEE Computational Intelligence Society (CIS) Travel Grant Subcommittee in 2016. He also serves as the Chair of the 2020 IEEE CIS Summer Schools Subcommittee.

Dongbin Zhao (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees from the Harbin Institute of Technology, Harbin, China, in 1994, 1996, and 2000, respectively.

He was a Post-Doctoral Fellow with Tsinghua University, Beijing, China, from 2000 to 2002. He has been a Professor with the Institute of Automation, Chinese Academy of Sciences, Beijing, since 2002, and a Professor with the University of Chinese Academy of Sciences, Beijing. From 2007 to 2008, he was a Visiting Scholar with The University of Arizona, Tucson, AZ, USA. He has published six books and over 90 international journal articles. His current research interests are in the areas of deep reinforcement learning, computational intelligence, autonomous driving, game artificial intelligence, robotics, smart grids, and so on.

Dr. Zhao was the Chair of the Technical Activities Strategic Planning Sub-Committee in 2019, the Beijing Chapter from 2017 to 2018, the Adaptive Dynamic Programming and Reinforcement Learning Technical Committee from 2015 to 2016, and the Multimedia Subcommittee of the IEEE Computational Intelligence Society (CIS) from 2015 to 2016. He is also the Chair of the Distinguished Lecturer Program. He also serves as an Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON CYBERNETICS, *IEEE Computation Intelligence Magazine*, and so on. He is also a guest editor of several renowned international journals. He is also involved in organizing many international conferences.