Several Reinforcement Learning Methods in Mean-Field Games with Binary Action Spaces

by

Chi Zhang

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs

in Partial Fulfillment of the requirements for the Degree of

Master of Science

in

Statistics

Carleton University

Ottawa, Canada

Abstract

Recent years have witnessed significant progress in the sub-field of machine learning known as reinforcement learning, in which interactions between intelligent agents and the environment enable agents to learn and solve sequential decision-making problems through accumulating rewards with delays. Despite much success in single-player settings, reinforcement learning in multi-agent domains remains a challenging task in many aspects. In this thesis, the mean-field approach will be used to study binary action space stochastic games with a sufficiently large number of players that can be generalized to the multi-population case. Based on the mean-field approximation, several algorithms will be implemented and compared in numerical experiments to visualize their convergence to the equilibrium policy.

Acknowledgement

It is my greatest honor to dedicate this thesis to my supervisor, Minyi Huang, without whom by no means could I finish this thesis. It was Professor Huang who introduced me to the mean-field games theory; it was Professor Huang who academically guided me and financially supported me; it was Professor Huang who cheered me up when I was deeply frustrated by the slow progress of my work. During the pandemic, face-to-face communication was unavailable, but Professor Huang strived to maintain a routine video communication with me. Moreover, Professor Huang invited me to a variety of virtual academic conferences, which facilitated me to grasp a deeper understanding of the mean-field game theory and reinforcement learning.

I would like to also dedicate my thesis to my parents. Due to Covid-19, I had to spend another academic year to complete my thesis, and my parents were still willing to support me, both financially and emotionally. My family originally comes from Wuhan, where the first outbreak of Covid-19 was observed, so last year was a tough year for my entire family. I hope this thesis marks a new beginning for me and my family.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Classical Multi-Agent Markov Games

In a typical single agent reinforcement learning setting, the environment is usually defined as a Markov decision process with a quintuple (S, A, P, c, γ), where S is the state spaces, A_x is the action spaces when the agent is in state x, P is a stochastic kernel specifies the transition law from the state-action pair $x_t = x, a_t = a$ to x_{t+1}, c is the one-stage cost (or alternatively the immediate reward depending on the context) that returns a scalar signal quantitatively measuring the quality of the transition, and $\gamma \in [0, 1]$ is the discount factor that adjusts the relative importance of costs at different stages.

Let x_t denotes the intelligent agent's state at time t. The agent chooses an action $a_t \in A_x$ when $x_t = x$, and the system will transition to x_{t+1} following $P(\cdot|x_t, a_t)$, incurring an immediate cost c. The cost-to-go function at time k under policy π is:

$$J_k^\pi(x) = \mathbb{E}[\sum_{t=k}^{\infty} \gamma^{t-k} c_{t+1} | X_k = x, \pi], \tag{1}$$

which is called the value function in [28].

In this thesis, the optimization problem for the agent is to minimize the cost function. The agent aims at finding the optimal policy $\pi^\star = \{\pi_k^\star, \pi_{k+1}^\star, \ldots\}$, a sequence of stochastic kernels that specifies the value function:

$$V_k(x) = \inf_{\pi \in \Pi_k} J_k^\pi(x), \tag{2}$$

which is called the optimal state-value function under policy π in [28]. When the notation

R indicating an immediate reward is used instead, the optimal policy $\pi^\star$ is essentially maximizing the reward function that can be defined similarly as in the equation (1).

The action-value function (or Q function) under policy π of a state-action pair (x, a) at time k can be defined as:

$$Q_k^\pi(x,a) = \mathbb{E}[\sum_{t=k}^{\infty} \gamma^{t-k} c_{t+1} | x_k = x, a_k = a, \pi]. \tag{3}$$

A generalization to the multiplayer Markov games can be formulated by a sextuple (N, S, A, P, c, γ) in which N denotes the number of agents involved in the system. x_t^i and a_t^i are the state and action of agent $i = \{1, 2, \ldots, N\}$, and c_{t+1}^i is the cost associated with the agent after the execution of action a_t^i in state x_t^i. The transition stochastic kernel and the cost function for agent i, compared with them in the single player dynamics, may depend on the joint state and action denoted by $x_t^{i,-i}$ and $a_t^{i,-i}$, respectively. As a result, the actions and states of other agents must be taken into consideration when determining the Q function of the i^{th} agent following policy π:

$$Q_k^{\pi,i}(x,a) = \mathbb{E}[\sum_{t=k}^{\infty} \gamma^{t-k} c_{t+1}^i | x_k^i = x, a_k^i = a, \pi]. \tag{4}$$

To compare two different policies, namely, π_1 and π_2, we say that π_1 is better than or equal to π_2 if the value function associated with policy π_1 is less than or equal to that of π_2 for all states. Mathematically speaking, $\pi_1 \geq \pi_2$ if and only if :

$$v^{\pi_1}(x) \leq v^{\pi_2}(x) \quad \forall x \in S. \tag{5}$$

Due to the complicated nature of multi-agent reinforcement learning, it is problematic for us to directly apply single agent reinforcement learning algorithms. At time t, each agent chooses an action a_t^i and moves to state x_{t+1}^i based on the states and actions of all participating agents. At time $t+1$, they will similarly choose an action a_{t+1}^i based on the joint state and action. From the perspective of a specific agent, namely, the i^{th} agent, the environment is constantly changing due to all agents acting and learning simultaneously, which makes it essentially unstable. That is to say, even if an agent chooses the same action in the same state, the corresponding cost and transition may not remain the same as a result of other agents' actions and states. In order to keep the environment stable, a tempting method is to model the entire joint space as the environment. However, the dimension of the environment will explode as the number of participating agents increases and eventually causes the computational burden to grow exponentially. Even if the state and action are finite, each agent adds its own state action to the joint space, thus increasing the computational complexity [5].

1.2 Mean-Field Games

To overcome the dimensionality difficulty in large population dynamic games, Huang-Caines-Malhame [11, 14] and Lasry-Lions [18, 19] have independently introduced the mean field game framework in which each agent has a negligible impact on others but they collectively exert a significant influence on any particular agent when the number of non-cooperative participating agents is large. Mean-field type control can also be applied to cooperative games, in which a central controller, in contrast to the non-cooperative games in which selfish agents manage to improve the system cooperatively to achieve

some optima [29].

Furthermore, the joint impact on an individual agent can be approximated by its local mean-field states and/or local mean-field actions [1]:

$$
\begin{aligned}
s_t^{-i}(s) &= \frac{1}{N-1} \sum_{j=1, j\neq i}^{N} \mathbb{1}(x_t^j = s) \\
\alpha_t^{-i}(a) &= \frac{1}{N-1} \sum_{j=1, j\neq i}^{N} \mathbb{1}(a_t^j = a),
\end{aligned}
\tag{6}
$$

where N is the number of agents influencing the mean-field.

As $N \to \infty$, the local empirical mean-field states and local empirical mean-field actions can be viewed as a population state distribution $\mu_t^i(s)$ and $u_t^i(a)$. With the homogeneity property shared among players, all players are indistinguishable and interchangeable, allowing us to drop the index and focus on a random representative agent. The system dynamics can be then decoupled from the whole states and actions of agents, but instead, the mean-field becomes a part of the environment. Mathematically, we have

$$
\begin{aligned}
x_{t+1} &\sim P(\cdot|x_t, a_t, \mu_t, u_t) \\
c_{t+1} &= c(x_t, a_t, \mu_t, u_t)
\end{aligned}
\tag{7}
$$

Finding the multi-agent games equilibrium now turns to solving the optimization problem with respect to the mean-field for a single agent. If the mean-field is fixed and not affected by agents' actions, the environment is stationary under their actions and the optimization problem is solvable using the fixed point (or top-down) approach [15]. In this manner, the stationarity and dimensionality problems are circumvented.

1.3 MFG with Binary Action Spaces Model

In this thesis, the major focus will be put on a special form of mean-field games, in which Huang and Ma [13] have proved that there is a unique equilibrium with a nice threshold property. We will briefly introduce the binary action spaces model in this subsection and offer numerical results in Section 4 and 5 using tabular mean-field Q-learning and deep mean-field Q-learning algorithms.

The binary action spaces model is made up of N homogeneous players each with a continuous state space $S = [0, 1]$ and an action space $A = \{a_0, a_1\}$, which can be deemed as whether doing a specific action or not. The transition kernel for a player depends only on its current state and action:

$$\begin{aligned} &P(x_{t+1} \in B | x_t = x, a_t = a_0) = P^0(B|x), \\ &P(x_{t+1} = 0 | x_t = x, a_t = a_1) = 1, \end{aligned} \tag{8}$$

where B is a subspace of the state space on $[x, 1]$. The agents are coupled only through their cost function:

$$c(x_t^i, x_t^{(N)}, a_t^i) = R(x_t^i, x_t^{(N)}) + \rho \mathbb{1}(a_t^i = a_1), \tag{9}$$

where ρ is the effort cost associated with taking the action a_1 and $x_t^{(N)}$ is the population average state $x_t^{(N)} = \frac{1}{N} \sum_{i=1}^{N} x_t^i$. The mean-field theory suggests z_t, a deterministic value, as an approximation of $x_t^{(N)}$. The discounted finite time horizon expected total cost of agent i taking action a_k^i in state x_k^i at time $k, k < T < \infty$, is therefore:

$$\bar{J}_s^i(x, z_{s,T}, a_{s,T}^i) = \mathbb{E}[\sum_{t=s}^{T} \gamma^{t-s} c(x_t^i, z_t, a_t^i | x_s^i = x)], \tag{10}$$

where γ is the discount factor ranging from 0 to 1, and $z_{s,T}$ and $a^i_{s,T}$ are the history of the mean-field sequence and control sequence for the i^{th} player, respectively. The dynamic programming equation to solve the value function $V(t,x) = \inf_{a^i_{t,T}} \bar{J}^i_t(x, z_{t,T}, a^i_{t,T})$ for the finite time horizon system becomes:

$$\begin{aligned} V(x,t) &= \min_{a^i_t \in A^i}[c(x, z_t, a^i_t) + \gamma \mathbb{E}[V(x^i_{t+1}, t+1)|x^i_t = x]] \\ &= \min[\gamma \int_0^1 V(y, t+1)P(dy|x) + R(x, z_t), \\ &\qquad \gamma V(0, t+1) + R(x, z_t) + \rho] \end{aligned} \tag{11}$$

with terminal condition

$$V(x,T) = R(x, z_T)$$

Since all players are interchangeable, it is safe for us to drop the superscript i in the value function and only solve the generic value function.

In practice, we are often more interested in a stationary and time-independent version of the value function. For sufficiently large t, if the solutions are expected to be in a steady-state, then $V(x,t)$ and z_t will be nearly time-independent, so we can transform the system into a stationary version:

$$\begin{aligned} V(x) &= \min_{a^i \in A^i}[c(x, z, a^i) + \gamma \mathbb{E}[V(x^i_{t+1})|x^i_t = x]] \\ &= \min[\gamma \int_0^1 V(y)P(dy|x) + R(x, z), \gamma V(0) + R(x, z) + \rho] \end{aligned} \tag{12}$$

An additional consistence condition will also to be imposed as the case with other mean-field games:

$$z = \int_0^1 x\mu(dx) \tag{13}$$

With mild conditions held, Huang and Ma [12, 13] have showed that there is a unique solution to the system of equations (12) - (13). Furthermore, a intriguing characteristic of the equilibrium policy is its threshold structure such that there exists a threshold parameter θ satisfying:

$$\pi(x_t^i, \theta) = \begin{cases} a_0 & x_t^i < \theta \\ a_1 & x_t^i \geq \theta \end{cases} \tag{14}$$

The threshold property may help us verify whether the solution obtained from reinforcement learning algorithms is indeed optimal. If the optimal action $\pi^*(x)$ changes multiple times on $x \in [0, 1]$, then there is a possibility that we have run into a problem in finding the optimal policy.

In this thesis the cost function $c(x_t^i, z_t, a_t^i)$ is taken as $(.2 + z_t)x_t^i + .5a_t^i$. The transition law given action a_0 is uniform on $[x_t^i, 1]$. The parameterization settings are proved not to violate uniqueness and existence conditions. If discretization of the state space is necessary for some algorithms, for example, tabular Q-learning, then the state space will be discretized using step Δ such that $S = \{0, \Delta, 2\Delta, \ldots, 1\}$.

In Section 1.1, we have introduced a condition for a policy being better than another, but what if neither of two policies is greater than the other using equation (5)? To numerically compare two policies, we compute the policy performance with initial state distribu-

tion ξ (assuming discretization of the state space):

$$J_{\pi,z} = \mathbb{E}_{X\sim\xi}[V_{\pi,z}(X)] = \sum_{i=1}^{N} V_{\pi,z}(x_i)\xi(x_i), \tag{15}$$

where N is the size of the state space. If ξ is discrete and takes values uniformly on $\{\xi_1, \xi_2, \ldots, \xi_N\}$, then the performance is simplified to the average of state value functions $\frac{1}{N}\sum_{i=1}^{N} V_{\pi,z}(\xi_i)$. If ξ is continuous, we can sample N agents whose states $\{x_1, x_2, \ldots, x_N\}$ follow ξ and then approximate the policy performance using $J_{\pi,z} = \frac{1}{N}\sum_{i=1}^{N} V_{\pi,z}(x_i)$. Thus a comparison between discrete state space and continuous state space is available.

1.4 Mean-Field Games in Multiple Populations

The assumption that all participating players are interchangeable and they each have a negligible impact on the system is usually challenged in real-world applications. In many areas, the mean-field theory may describe the environment better if more flexibility is allowed in model designation. A typical example illustrating the shortcomings of this assumption is the economic market, in which some major companies may have a say in price decisions regardless of the size of the market. Therefore, assuming all companies in the market to share the same insignificant impact on the system when the market size is extremely large may fail to grasp the relationship between major corporations and minor competitors, a key characteristic of the market dynamics. Huang [10] has proposed an extension of the traditional mean-field model to the mixed players model where a major player and a large number of minor players coexist and pursue their individual objectives.

Bensoussan, Huang, and Lauriére [4] have studied the mean-field mean-field games

and mean-field type control problems with multiple populations. Masaaki Fujii [8] has analyzed the mean-field games and mean-field type control problem with multiple populations in a broader sense. The problem is divided into three categories based on whether agents are cooperative: 1) all agents are non-cooperative and pursue their individual goals; 2) agents within each population are cooperative but compete with agents in other populations; 3) the agents in some populations are cooperative but those in the other populations are not.

Huang [10] has also studied the mean-field games with mixed players and proposed a major-minor model, which is similar to the first case in Fujii's classification in that all agents are non-cooperative. In this thesis, we will focus on the major-minor model in [10] and find the equilibrium policy of the system using reinforcement learning algorithms in Section 6.

An $N + 1$ players major-minor model involves N minor players and 1 major player whose state spaces and action spaces are denoted by S_i and A_i, $i = 0$ for the major player and $i = 1, 2, \ldots, N$ for the minor players. In Huang [10], the state spaces and action spaces are assumed to be finite for both major player and minor players. If the minor players are further assumed to share the common state space S and action space A (as the example in [10]), then $S_i = S_j = S$ and $A_i = A_j = A$ for $i, j \geq 1$, so we can partition the common state space into $\{s_1, s_2, \ldots, s_K\}$, where K is the size of the state space. The i^{th} player's state-action pair at time t is (x_t^i, a_t^i).

To model the interaction between players using mean-field approximation, Huang has

introduced a random measure process that approximates the states of minor players:

$$
\begin{aligned}
I^{(N)}(t) &= (I_1^{(N)}(t), I_2^{(N)}(t), \ldots, I_K^{(N)}(t)), t \in \mathbb{Z}_+, \\
I_k^{(N)}(t) &= \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(x_t^i = s_k), k \in \{1, 2, \ldots, K\}.
\end{aligned}
\tag{16}
$$

A policy for the i^{th} player $\pi^i(\cdot|x_t^i)$ at time t is a probability measure on its action space A^i that assigns a probability to a certain action based on the current state i.e., $\sum_{a_t^i \in A^i} \pi^i(a_t^i|x_t^i) = 1, \pi^i(a_t^i|x_t^i) \geq 0$.

In this thesis, we assume that the major player only influences the minor players through their cost functions, so the state of the i^{th} minor player at time t can be defined as (x_t^i, x_t^0) to incorporate the influence from the major player.

Assuming Markovian property for the system, the transition laws for the major player and minor players given a state-action pair (x, a) at time t share a similar form:

$$
P^i(x_{t+1}^i = x'|x_t^i = x, a_t^i = a), \tag{17}
$$

where i indicates the index of the player. The transition law under a policy π can be obtained using the law of total probability.

$$
P^i(x_{t+1}^i = x'|x_t^i = x, \pi) = \sum_{a^i \in A^i} \pi(a^i|x) P^i(x'|x, a^i) \tag{18}
$$

Based on the mean-field approximation of the states of minor players, the infinite horizon discounted cost function of a minor player may differ from that of the major player in

including the state of the major player:

$$
\begin{aligned}
J^0 &= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t c(x_t^0, a_t^0, I^{(N)}(t))] \\
J^i &= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t c(x_t^i, x_t^0, a_t^i, I^{(N)}(t))].
\end{aligned}
\tag{19}
$$

A random process $\theta(t)$ taking values on a probability simplex $\mathcal{D}$ is introduced to approximate $I^{(N)}(t)$ as $N \to \infty$:

$$
\mathcal{D} = \{(\lambda_1, \ldots, \lambda_K) \in \mathbb{R}_+^k | \sum_{j=1}^{k} \lambda_j = 1\}. \tag{20}
$$

Due to major player's influence on minor players' cost functions, the generic minor player's action may be impacted by both its own state x_t^i and the state of the major player x_t^0. Huang has proposed an update rule for $\theta(t)$:

$$
\begin{aligned}
\Psi = \{\phi(x_t^0, \theta(t)) = (\phi_1, \ldots, \phi_K) | \phi_k \geq 0, \sum_{k \in S} \phi_k = 1\}, \\
\theta(t+1) = \psi(x_t^0, \theta(t)),
\end{aligned}
\tag{21}
$$

where ψ is from a function class Ψ. That is to say, given the state of the major player and a valid distribution on S, the Ψ operator returns a valid distribution on S as the state distribution of the minor players [10].

In the framework in [10] to solve the optimization problem in the major-minor model, the original problem can be decomposed two sub-problems: the limiting problem of the major player and the limiting problem of the minor player, so we will have two dynamic equations to solve.

Given a state-pair of the major player and the state distribution of minor player (x^0, θ), the dynamic equation for the major player is therefore [10]:

$$\begin{aligned} v(x^0, \theta) &= \min_{a^0 \in A^0} \{c^0(x^0, \theta, a^0) + \gamma \mathbb{E}[v(x^0_{t+1}, \theta(t+1))]\} \\ &= \min_{a^0 \in A} \{c^0(x^0, \theta, a^0) + \gamma \sum_{k \in S^0} P^0(k|x^0, a^0) v(k, \psi(x^0, \theta))\}. \end{aligned} \tag{22}$$

Assuming finiteness of the action space of the major player, an optimal policy $\hat{\pi}^0 \in \Pi^0$ solving the dynamic equation (22) exists and is determined as a stationary Markov policy of its current state and the state-distribution of minor players $\hat{\pi}^0(x^0, \theta)$. The optimal policy needs not to be unique, and it is possible that Π^0 contains multiple elements.

When the major player chooses a policy from Π^0, the optimization problem for minor players and be reformulated as a Markov decision problem with state $(x^0_t, x^i_t, \theta(t)$ and control a^i_t. The dynamic equation for the minor players is then given by [10]:

$$\begin{aligned} v(x^i, x^0, \theta) &= \min_{a^i \in A} \{c(x^i, x^0, \theta, a^i) + \gamma \mathbb{E}[v(x^i_{t+1}, x^0_{t+1}, \theta(t+1))]\} \\ &= \min_{a^i \in A} \{c(x^i, x^0, \theta, a^i) + \\ &\qquad \gamma \sum_{\substack{j \in S, \\ k \in S^0}} P^i(j|x^i, a^i) P^0(k|x^0, \hat{\pi}^0) v(j, k, \psi(x^0, \theta))\}, \end{aligned} \tag{23}$$

where the transition law given a state and policy is as defined in equation (18) - (19). Similarly, at least one optimal strategy exists and spans a policy set Π^i. Although it is possible that several policies are equally optimal, we will force each minor player to select the same policy to simplify the problem.

Huang [10] has proved that given a pair of policy $(\hat{\pi}^0, \hat{\pi})$, $(x^0(t), x^i(t), \theta(t))$ forms a

Markov chain with stationary transition probabilities. With x^0 and θ fixed, the transition probability from state s_i to state s_j is $P(s_j|s_i, \hat{\pi}(s_i, x^0, \theta))$. It enables us to derive another update rule for $\theta(t) : \theta(t+1) = \theta(t)P^\star(x^0(t), \theta(t))$:

$$\begin{bmatrix} P(s_1|s_1, \hat{\pi}(s_1, x^0, \theta)) & \dots & P(s_K|s_1, \hat{\pi}(s_1, x^0, \theta)) \\ \dots & \dots & \dots \\ P(s_1|s_K, \hat{\pi}(s_K, x^0, \theta)) & \dots & P(s_K|s_K, \hat{\pi}(s_K, x^0, \theta)) \end{bmatrix} \tag{24}$$

where $P^\star$ is a $K \times K$ transition matrix.

Compared with the equation (13), the equation (21) and (24) play the role of consistence condition in the equation system (12)-(13), so the equation system (21) - (24) in the major-minor model is the counterpart of the equation system (12)-(13) in the mean-field games with binary action spaces model [10].

1.5 Our Approaches and Contributions

In this thesis, we implement two mean-field tabular Q-learning algorithms and four mean-field deep Q-learning algorithms as outlined in following sections to approximate the equilibrium in the binary action spaces model. Whenever discretization of the state space is necessary, we discretize the state space by the discretization step $\Delta = 0.01$ so that the partitioned state space is $\{0, 0.01, 0.02, \dots, 1\}$ with size $N = 101$.

Our contributions are as follows:

1. We apply tabular learning methods in binary action spaces model and find that the approximate equilibrium usually leads to a contradiction to the conclusion that there should be a unique threshold in the equilibrium policy. We propose two methods

to estimate the true unique threshold based on our approximation of the equilibrium and demonstrate that these two estimations are close to the true threshold.

2. We show that the Melowmax operator can be applied to our binary action spaces model and confirm that the introduction of a Mellowmax operator can improve the learning accuracy.

3. Our experiments suggest that the mean-field deep Q-learning and mean-field actor-critic algorithms in [30] are viable in our binary action spaces model with some modifications.

4. Our numerical experiments show that the threshold estimated using deep Q-learning algorithms in general oscillates more violently than that using tabular algorithms.

5. We conduct numerical experiments following the policy-based algorithm in [27] and improve its efficiency by avoiding discretizing the state space.

6. This thesis is the first effort to utilize reinforcement learning algorithms in the major-minor model.

2 Q-Learning in Reinforcement Learning

Sometimes it is difficult to obtain an exact solution to the value function or we do not even know the model structure, so we are interested in finding an approximate equilibrium. Q-learning related algorithms are a powerful method to approximate the equilibrium. This section serves as an overview of Q-learning algorithms that will be used in the thesis.

2.1 Basic Q-Learning

Based on equation (4), a popular off-policy TD control algorithm known as Q-learning is proposed to approximate the optimal Q function for a given state. In the case of single player reinforcement learning, the update rule is:

$$Q_{t+1}(x_t, a_t) = (1 - \alpha_t)Q_t(x_t, a_t) + \alpha_t(c_{t+1} + \gamma \min_{a \in A_x} Q_t(x_{t+1}, a)), \tag{25}$$

where α_t is the learning rate at time t controlling to what extent the newly learned information will override the old one. If the action-values of all state-action pairs can be stored in a huge table known as Q-table, then we can update the Q-table every time a transition experience (x, a, c, x') is available, where x is the current state, a is the action taken in state x, c is the immediate cost, and x' is the new state. For example, if the Q-table is initialized to be 0 for all entries, the Q-table at time 1 given a transition (x, a, c, x') will be all zeroes except for the action-value function associated with the state-action pair (x, a), which will be updated to be αc as an approximation of the true action-value. As more data, or more transition experiences, are available, we will be dynamically update the Q-table and have a more accurate approximation.

Even-Dar and Mansour [7] recommend using a decaying learning rate in (25):

$$\alpha_t(x, a) = \frac{1}{[\#(x, a, t)]^\omega}, \quad \omega \in (\frac{1}{2}, 1], \tag{26}$$

where $[\#(x, a, t)]$ is the number of times that the agent visits the state-action pair (x, a) until time t. α_t is called a linear learning rate when $\omega = 1$ and a polynomial learning rate when $\omega \in (\frac{1}{2}, 1)$.

Q-learning is guaranteed to converge with following mild conditions:

$$\begin{aligned}
&\sum_{t=1}^{\infty} \alpha_t(x,a) = \infty, \\
&\sum_{t=1}^{\infty} \alpha_t^2(x,a) < \infty,
\end{aligned} \tag{27}$$

each state-action pair will be visited infinitely often.

It is easy to see that $\alpha_t(x,a) = \frac{1}{[\#(x,a,t)]^{\omega}}$ with $\omega \in (\frac{1}{2}, 1]$ satisfies the conditions.

$$\begin{aligned}
\sum_{t=1}^{\infty} \alpha_t(x,a) &= \sum_{t=1}^{\infty} \frac{1}{[\#(x,a,t)]^{\omega}} \\
&\geq \sum_{t=1}^{\infty} \frac{1}{t^{\omega}} = \infty, \\
\sum_{t=1}^{\infty} \alpha_t^2(x,a) &= \sum_{t=1}^{\infty} \frac{1}{[\#(x,a,t)]^{2\omega}} \\
&\leq \sum_{t=1}^{\infty} \frac{1}{t^{2\omega}} < \infty,
\end{aligned}$$

where we make use of $[\#(x,a,t)] \leq t$ and the divergence property of p-series. Paraphrasing the condition that each state-action pair is visited infinitely often in mathematical language, we have $\pi(x,a) > 0, \forall x \in s$ and $a \in A_x$. ϵ-greedy and Boltzmann exploration policies both ensure that this condition is satisfied.

Additional to carefully choosing the proper learning rates α, another challenge in reinforcement learning is to balance the ratio of exploration and exploitation, i.e., how to choose action a_t. In the absence of exploration, the agent will always choose the current optimal action $a_t^{\star} = \arg\min_{a \in A_x} Q(x,a)$, resulting in no learning and a violation of the

convergence condition of Q-learning. On the contrary, too much exploration will lead the agent to sometimes waste time trying current non-optimal actions and slow down the learning process.

ϵ-greedy and Boltzmann (or softmax) exploration will be covered in this thesis.

- ϵ-greedy exploration: it can be thought of as a combination of a purely random exploration with uniform probability on the action space and a purely greedy policy. The exploration parameter $\epsilon_t \in [0, 1]$ specifies the probability that the agent takes a exploring action (randomly chooses an action among all available actions). With probability 1-ϵ the agent performs the best action hinged on the learned Q function by time t. If $\epsilon > 0 \ \forall t$, then the convergence condition for Q-learning will not be violated. [28].

- Boltzmann (softmax) exploration: unlike the ϵ-greedy exploration that assigns equal probability to all non-optimal actions, the Boltzmann exploration assigns probabilities according to the Q-values of available actions using Boltzmann distribution:

$$\pi_{Boltzmann}(x_t, a) = \frac{exp(\beta \ Q(x_t, a))}{\sum_{a' \in A_{x^t}} exp(\beta \ Q(x_t, a'))}, \tag{28}$$

  where β is the temperature parameter controlling the exploration rate. For a given state, equation (28) gives the probability of selecting action a. Due to the non-negative property of the exponential function, the convergence condition for Q-learning will not be violated if the Boltzmann policy is used.

Since the agent knows little about the environment and the learned Q-values may not carry much meaningful information at the beginning of learning, it is reasonable to explore

more in the early phase of learning. The agent may make better use of the Q-values as more information can be gleaned from the past experience.

With finite and small number of state and action, the tabular version of Q-learning can be implemented using a large Q-table $Q : S \times A \rightarrow \mathbb{R}$ containing all state-action pairs (x, a). Each state-action pair is usually initialized with 0. However, randomly initialize Q-values may help break ties if multiple actions are associated with the same Q-value. Alternatively, the agent can utilize random tiebreak.

There are some instances that a certain state-action pair (x, a) is unavailable. An unavailable action is different from an undesired action. An undesired action means that in the environment it is possible to take such action, but the consequences are exceptionally bad. In this case, we can define the action-value $Q(x, a)$ to be an extreme large value (if the cost function is used) or an extreme small value (if the reward function is used) so that the agent will learn not to choose this action in the learning phase. An unavailable action means that the action is forbidden in the environment. For example, in an auction game where borrowing is not allowed, bidding when the agent's wealth is 0 can be considered an unavailable action. Thus, the unavailable actions need to be precluded in the environment dynamics.

This can be done by assigning ∞ or $-\infty$, depending on the context, rather than 0 to $Q(x, a)$, if the Boltzmann exploration strategy is used. There is no need to specify the available action spaces for each state $x \in S$, for the probability of selecting a in state s as defined in equation (28) is 0, implying exclusion of unavailable actions in A_x.

Nevertheless, it is necessary to find the available action space for state x if the ϵ-greedy exploration strategy is preferred, because all actions including the unavailable ones are as-

signed positive probabilities when the generic agent explores the environment. In practice, the agents should be only presented with available action spaces $A_x = \arg_{a \in A} Q(x, a) < \infty$ or $A_x = \arg_{a \in A} Q(x, a) > -\infty$ in the exploration phase.

In the case of episodic games, $Q(x^{terminal}, \cdot) = 0$, where $x^{terminal}$ denotes the terminal state after which the environment will end or be reset. In the setting of continuing tasks, we may keep updating the Q-table until a given convergence tolerance has been achieved or the maximum number of updates has been reached.

Algorithm 1: Basic single agent Q-learning

Data: learning rate: $\alpha_{i,t}$,
exploration parameter : $\epsilon_{i,t}$ for ϵ-greedy exploration or $\beta_{i,t}$ for Boltzmann exploration
Result: estimated $Q^\star$

```
initialize Q-table based on the model dynamics;
for episode i = 1, 2, 3, ... do
    randomly initialize x_{i,0} ∈ S ;
    for t = 0, 1, 2, ... do
        choose action a_{i,t} following a pre-specified exploration strategy;
        observe the next state x_{i,t+1} and cost c_{i,t+1};
        update Q-table using equation (25);
        if x_{i,t+1} = s^{terminal} then
            break;
        end
    end
end
return Q*
```

Having properly initialized the Q-table, we can implement the basic single agent Q-learning algorithm as outlined in algorithm 1. Although the Q-learning algorithm proves to converge, it leads to a significant bias in practice. Due to the minimization operation in equation (25), the agent may observe inaccurate Q-values. For example, let $q(x, a)$ be the

true value of the state-action pair (x, a) and $Q(x, a)$ be the corresponding estimated value of this pair with $\arg\min_{a \in A_x} q(x, a) = a^\star$. Because $Q(x, a^\star)$ is an estimate, it is uncertain and distributed around $q(x, a^\star)$. With positive probability, $Q(x, a^\star)$ will be below $q(x, a^\star)$ due to noise in the environment. In this sense, a bias may occur every time step when the agent selects action and updates the Q-function using the same sampled trajectory, i.e., the same Q-table. [28]

In the latest version of *Reinforcement Learning: an Introduction*, Sutton and Barto discuss the bias incurred by minimization (or maximization) operation in Q-learning and include double learning as an improvement of basic Q-learning [28]. The major difference between the basic Q-learning algorithm and the double Q-learning algorithm is that the latter utilizes multiple (2 in general) Q-tables so that the action selection and value update are based on different Q-tables. If the agent chooses an action a according to $Q^a(x, \cdot)$, he/she will update $Q^a(x, a)$ using a modified version of updating rule (25):

$$Q^a_{t+1}(x_t, a_t) = (1 - \alpha_t) Q^a_t(x_t, a_t) + \alpha_t (c_{t+1} + \gamma \min_{a \in A_x} Q^b_t(x_{t+1}, a)). \tag{29}$$

Double Q-learning can be extended to multi Q-learning by introducing n Q-tables ($n \geq 2$), and the update for $Q^a(x_t, a_t)$ makes use of the average of the rest $n - 1$ Q-tables. Duryea, Ganger, and Hu [6] have analyzed the performance of the multi Q-learning algorithm. If the environment is completely unknown and the agent prefers a purely exploratory strategy, the introduction of a larger n may help the estimated Q-table remain stable. However, the difference between double Q-learning and multi Q-learning is not noticeable if strategies like ϵ-greedy is used.

Other algorithms such as double SARSA and double expected SARSA can be similarly

Algorithm 2: Double Q-learning

Data: learning rate: $\alpha_{i,t}$,
exploration parameter : $\epsilon_{i,t}$ for ϵ-greedy exploration or $\beta_{i,t}$ for Boltzmann exploration

Result: estimated $Q^{\star}$

initialize $Q^a(x,a)$, $Q^b(x,a)$ based on the model dynamics;
for *episode* $i = 1, 2, 3, \ldots$ **do**
    randomly initialize $x_{i,0} \in S$;
    for $t = 0, 1, 2, \ldots$ **do**
        choose action $a_{i,t}$ following a pre-specified exploration strategy based on Q^a and Q^b;
        observe the next state $x_{i,t+1}$ and cost $c_{i,t+1}$;
        if $Uniform(0,1) > .5$ **then**
            $Q^a(x_{i,t}, a_{i,t}) = (1-\alpha_t)Q^a(x_{i,t}, a_{i,t}) + \alpha_t(c_{t+1} + \gamma \min_{a \in A_x} Q^b(x_{i,t+1}, a))$
        else
            $Q^b(x_{i,t}, a_{i,t}) = (1-\alpha_t)Q^b(x_{i,t}, a_{i,t}) + \alpha_t(c_{t+1} + \gamma \min_{a \in A_x} Q^a(x_{i,t+1}, a))$
        end
        if $x_{i,t+1} = s^{terminal}$ **then**
            break;
        end
    end
end
return $Q^a(x,a)$, $Q^b(x,a)$

implemented by replacing the minimization operation in equation (25) with selecting an action using a policy derived from Q^a and Q^b for double SARSA or with the expected state-action value of the next state following the same policy.

2.2 Deep Q-Learning

In reality it is often unrealistic to ensure the state space to be small and finite, which makes the tabular version of Q-learning infeasible. Functional approximation offers a remedy for it. Instead of storing all action-values in a extremely large table, we can utilize some functions to approximate the state-value function or action-value function (Q-function) that generalize from sampled trajectories to the entire function, even if some some states or state-action pairs have not been visited in the trajectories.

If a linear approximator is used, then $Q(x, a)$ can be approximated as the inner product between a weight vector $\mathbf{w}$ and feature vector $\mathbf{x}(s)$:

$$\hat{Q}(s, a, w) = \mathbf{w}^T \mathbf{x}(s) = \sum w_i x_i(s), \tag{30}$$

where a feature vector $\mathbf{x}(s)$ represents the input of state s. If the state space is one-dimensional, $\mathbf{x}(s) = s$ suffices to represent the input. However, if the state space is with two dimensions, $\mathbf{x}(s) = (s_1, s_2, s_1 s_2)$ may incorporate the interaction term between two dimensions [28].

Sometimes non-linear function approximators will be preferred to better learn complex environments in which the relationship between inputs and outputs cannot be represented linearly. It has been a popular choice to estimate action-value functions $Q(x, a, \phi) \approx$

$Q^\star(x, a)$ using a neural network as a non-linear approximator, where ϕ is the weights of the neural network representing action-values, in the deep Q-learning (DQN) algorithm [22]. A neural network function approximator with weights ϕ is usually called a Q-net and can be trained to better approximate the action-value function by minimizing the loss function

$$L_t(\phi_t) = (c + \gamma \min_{a \in A_x} Q(x_{t+1}, a, \phi_{t-1}) - Q(x_t, a_t, \phi_t))^2. \tag{31}$$

In addition to applying neural network approximation, the DQN algorithm improves the general Q-learning via introducing an experience replay buffer and a separate target network [21].

The agent's interaction with the environment at each time step t $(x_t, a_t, c_{t+1}, x_{t+1})$ will be stored in a large data set $\mathcal{D}$, which usually only contains the last N_{buffer} steps of experience, where N_{buffer} is the predefined buffer size. If the replay buffer is full, the earliest sample will be dropped so that it can store the latest experience. With a replay buffer, a single experience can be sampled and learned multiple times, which improves data efficiency.

Furthermore, the replay buffer can help the learning algorithm attain stability. It is known that functional approximation like neural networks may lead to instability or even divergence in reinforcement learning due to the correlations between consecutive transition experiences, and the correlations between the Q-function that we aim at approximating and the target Q-function in the loss function.

Storing the past transitions in a replay buffer and sampling from the shuffled replay buffer rather than learning solely from the previous transition experience may reduce the correlation between transitions and downplay the seriousness of the instability issue [21].

To update Q-function, a mini-batch of size K will be uniformly sampled from $\mathcal{D}$.

Uniform sampling assumes all experiences share the same level of importance, which sometimes is not the case in reality. Schaul, Quan, Antonoglou, and Silver in [24] argue that the agent may learn more effectively and efficiently from some transitions with high expected learning progress measured by the magnitude of the temporal-difference (TD) error.

Additionally, a separate target network denoted by weight parameter ϕ^- is introduced to stabilize the learning process. The target network will be updated every C steps with learning rate τ:

$$\phi^- \leftarrow \tau\phi + (1-\tau)\phi^-, \tag{32}$$

where C is known as the update frequency, and kept fixed during C steps. The delayed update for the target network allows us to generate the target for Q-learning using a fixed neural network during the C steps, which prevents the network from oscillating by reducing the correlation with the target [21].

In spite of these breakthroughs, the deep Q-learning algorithm may not work perfectly as expected. In contrast to the tabular Q-learning algorithms in which the agent's performance usually remain stable if not improve as more data are obtained from the training phase, the deep Q-learning algorithm may run into a well-known problem called "catastrophic forgetting" in which the agent's performance may suddenly deteriorate after some learning steps. Mnih et al. have proposed either clipping the gradient of the error or the rewards to avert a momentous change in the neural network parameters, but the "catastrophic forgetting" may still occur.

There are numerous variants of deep Q-learning algorithms. In this thesis, we will only include the actor-critic method. Compared with Algorithm 3, the actor-critic deep

Algorithm 3: Deep Q-learning (DQN)

Data: target network update rate: τ, target network update frequency C,experience replay size: N_{buffer}, mini-batch size: K, exploration parameter: β, discount factor: γ

initialize Q with random weights ϕ;
initialize target network Q^{-} with weights $\phi^{-} = \phi$;
initialize an empty experience replay buffer $\mathcal{D}$;
for *episode* $i = 1, 2, 3, \ldots$ **do**
    randomly initialize $x_{i,0} \in S$;
    for $t = 0, 1, 2, \ldots$ **do**
        choose action $a_{i,t}$ following a pre-specified exploration strategy;
        observe the next state $x_{i,t+1}$ and cost $c_{i,t+1}$;
        store $(x_{i,t}, a^{i,t}, c_{i,t+1}, x_{i,t+1})$ in the replay buffer $\mathcal{D}$;
        sample a mini-batch of transitions from $\mathcal{D}$;
        update Q-network using equation (31);
        if $x_{i,t+1} = s^{terminal}$ **then**
            break;
        end
    end
    update the target network Q^{-} using (32)
end
return Q, Q^{-}

Q-learning replaces the exploration strategy with an actor network and in practice learns more stably and accurately.

2.3 Mellowmax and DeepMellow

In this section we will introduce an alternative operator to the minimization operator in updating the Q-function in (25) known as the Mellowmax operator (abbreviated as *mm*). According to Asadi and Littman [3] the Mellowmax operator for state x takes a vector of the action-values associated with state x as the argument:

$$mm_\omega(x) = \frac{1}{\omega}\ln(\frac{1}{|A_x|}\sum_{a\in A_x}\exp(\omega Q(x,a))), \omega \in (-\infty, 0) \cup (0, \infty). \tag{33}$$

Thus, $mm_\omega(x)$ is equivalent to $mm_\omega(Q(x,\dot{}))$ or $mm_\omega(\mathbf{X})$, where $\mathbf{X} = Q(x,\dot{})$.

The Mellowmax operator is theoretically superior to the minimization operator in that it may reduce the overestimation problem and potentially remove the need for a separate target network in deep Q-learning. In [17], Kim, Asadi, Littman, and Konidaris mainly consider the case when $\omega > 0$, but in this thesis we will include the case when $\omega < 0$ when proving its properties. A comparison between algorithms using the Mellowmax operator and those without it will be presented in Section 4 and 5 to see if it indeed can improve the Q-learning accuracy.

We start by demonstrating the relationship between the Mellowmax operator and a minimization operator. As $\omega \to \infty$ (resp. $\omega \to -\infty$), the Mellowmax operator degenerates

to a maximization (resp. minimization) operator. Define $m = \min_{a \in A_x} Q(x, a)$, we have

$$\begin{aligned}
\lim_{\omega \to -\infty} mm_\omega(x) &= \lim_{\omega \to -\infty} \frac{1}{\omega} \ln(\frac{1}{|A_x|} \sum_{a \in A_x} \exp(\omega Q(x, a))) \\
&= \lim_{\omega \to -\infty} \frac{1}{\omega} \ln(\frac{1}{|A_x|} \exp(\omega m) \sum_{a \in A_x} \exp(\omega(Q(x, a) - m))) \\
&= \lim_{\omega \to -\infty} \frac{1}{\omega} (\ln(\exp(\omega m)) - \ln(|A_x|) + \ln \sum_{a \in A_x} \exp(\omega(Q(x, a) - m))) \\
&= m + \lim_{\omega \to -\infty} \frac{1}{\omega} (-\ln(|A_x|) + \ln \sum_{a \in A_x} \exp(\omega(Q(x, a) - m)))
\end{aligned} \tag{34}$$

Since $Q(x, a) \geq m$ for $\forall a \in A_x$, $0 \leq Q(x, a) < \infty$, and then we can conclude that $\lim_{\omega \to -\infty} mm_\omega(x) = m = \min_{a \in A_x} Q(x, a)$. Similarly, $mm_\omega(x)$ in the case when ω approaches ∞ can be proved to act as a maximization operator.

When $\omega \to 0$, $\lim_{\omega \to 0} mm_\omega(x)$ approaches the mean of the action-values associated with state, i.e., $\lim_{\omega \to 0} mm_\omega(x) = \frac{1}{|A_x|} \sum_{a \in A_x} Q(x, a)$. As $\omega \to 0$, obviously $\ln(\frac{1}{|A_x|} \sum_{a \in A_x} \exp(\omega Q(x, a))) \to 0$, so L'Hôspital's rule can be applied, yielding:

$$\begin{aligned}
\lim_{\omega \to 0} mm_\omega(x) &\overset{\text{H}}{=} \lim_{\omega \to 0} \frac{\frac{1}{|A_x|} \sum_{a \in A_x} Q(x, a) \exp(\omega Q(x, a))}{\frac{1}{|A_x|} \sum_{a \in A_x} \exp(\omega Q(x, a))} \\
&= \frac{\lim_{\omega \to 0} \sum_{a \in A_x} Q(x, a) \exp(\omega Q(x, a))}{\lim_{\omega \to 0} \sum_{a \in A_x} \exp(\omega Q(x, a))} \\
&= \frac{1}{|A_x|} \sum_{a \in A_x} Q(x, a)
\end{aligned} \tag{35}$$

The Mellowmax operator outperforms other operators such as softmax (Boltzmann) and ϵ-greedy operator in that, as argued by Asadi and Littman [3], it enjoys two nice properties, non-expansion and differentiability, and can relieve the overestimation problem,

which may potentially reduce the need of a separate target network in deep Q-learning.

Despite the non-expansion property of ϵ-greedy operator, it is not differentiable. On the contrary, the Boltzmann operator is differentiable, but it's lack of non-expansion property may make it suffer from oscillating between multiple fixed points and therefore inferior to the Mellowmax operator [20, 23]. Failure to satisfy the non-expansion requirement may lead to unstable or even divergent learning behavior when estimating the value function.

To establish the non-expansion property of the Mellowmax operator, we need to show that the inequality $|mm_\omega(x) - mm_\omega(y)| \leq \max_{a \in A_x} |Q(x,a) - Q(y,a)|$ holds for $\forall x \in S$. Assuming $Q(x,\cdot)$ and $Q(y,\cdot)$ are $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$, where $n = |A_x| = |A_y|$ is the size of the shared action space, we denote the difference between $Q(x,\cdot)$ and $Q(y,\cdot)$ by Δ, i.e., $\Delta_i = x_i - y_i, i \in \{1, 2, \ldots, n\}$. Without loss of generality, we assume $mm_\omega(x) \geq mm_\omega(y)$, then:

$$
\begin{aligned}
|mm_\omega(x) - mm_\omega(y)| &= \frac{1}{\omega}\ln(\frac{1}{n}\sum_{i=1}^{n}\exp(\omega x_i)) - \frac{1}{\omega}\ln(\frac{1}{n}\sum_{i=1}^{n}\exp(\omega y_i)) \\
&= \frac{1}{\omega}\ln\frac{\sum_{i=1}^{n}\exp(\omega x_i)}{\sum_{i=1}^{n}\exp(\omega y_i)} \\
&= \frac{1}{\omega}\ln\frac{\sum_{i=1}^{n}\exp(\omega(y_i + \Delta_i))}{\sum_{i=1}^{n}\exp(\omega y_i)}
\end{aligned}
\tag{36}
$$

when $\omega > 0$, we have:

$$\begin{aligned}
|mm_\omega(x) - mm_\omega(y)| &\leq \frac{1}{\omega}\ln\frac{\sum_{i=1}^{n}\exp(\omega(y_i + \max(|\Delta|)))}{\sum_{i=1}^{n}\exp(\omega y_i)} \\
&= \frac{1}{\omega}\ln\frac{\exp(\omega\max(|\Delta|))\sum_{i=1}^{n}\exp(\omega y_i)}{\sum_{i=1}^{n}\exp(\omega y_i)} \\
&= \frac{1}{\omega}\ln(\exp(\omega\max(|\Delta|))) \\
&= \max(|\Delta|).
\end{aligned}$$

When $\omega < 0$, we have:

$$\ln\frac{\sum_{i=1}^{n}\exp(\omega(y_i + \Delta_i))}{\sum_{i=1}^{n}\exp(\omega y_i)} \geq \ln\frac{\sum_{i=1}^{n}\exp(\omega(y_i + \max(|\Delta|)))}{\sum_{i=1}^{n}\exp(\omega y_i)},$$

so

$$\begin{aligned}
|mm_\omega(x) - mm_\omega(y)| &\leq \frac{1}{\omega}\ln\frac{\sum_{i=1}^{n}\exp(\omega(y_i + \max(|\Delta|)))}{\sum_{i=1}^{n}\exp(\omega y_i)} \\
&= \frac{1}{\omega}\ln\frac{\exp(\omega\max(|\Delta|))\sum_{i=1}^{n}\exp(\omega y_i)}{\sum_{i=1}^{n}\exp(\omega y_i)} \\
&= \frac{1}{\omega}\ln(\exp(\omega\max(|\Delta|))) \\
&= \max(|\Delta|).
\end{aligned}$$

Thus, the $mm_\omega(x)$ is a non-expansion.

The derivatives of the Mellowmax function $mm_\omega(x)$ can be determined by taking partial differentiation with respect to each of the element in $Q(x, \cdot)$ and the temperature parameter ω.

- For an action $a \in A_x$, the partial derivative with respect to $Q(x, a)$ is given as:

$$\begin{aligned} \frac{\partial mm_\omega(x)}{\partial Q(x,a)} &= \frac{1}{\omega} \frac{\frac{1}{|A_x|}\exp(\omega Q(x,a))\omega}{\frac{1}{|A_x|}\sum_{k\in A_x}\exp(\omega Q(x,k))} \\ &= \frac{\exp(\omega Q(x,a))}{\sum_{k\in A_x}\exp(\omega Q(x,k))} \geq 0 \end{aligned} \tag{37}$$

  It is noteworthy that the partial derivative suggests Mellowmax as a non-decreasing function of each element in $Q(x, \cdot)$. It makes perfect sense in that as the action-function for a specific action increases, say, $\exists a \in A_x, Q'(x, a) > Q(x, a)$, the following inequality $\sum_{a\in A_x}\exp(\omega Q'(x,a)) > \sum_{a\in A_x}\exp(\omega Q(x,a))$ will hold regardless of the value of ω.

- Taking partial differentiation with respect to ω, we have:

$$\begin{aligned} \frac{\partial mm_\omega(x)}{\partial \omega} &= \frac{\partial \frac{\ln(\frac{1}{|A_x|}\sum_{a\in A_x}\exp(\omega Q(x,a)))}{\partial \omega}\omega - \ln(\frac{1}{|A_x|}\sum_{a\in A_x}\exp(\omega Q(x,a)))}{\omega^2} \\ &= \frac{\frac{\sum_{a\in A_x}\omega Q(x,a)\exp(\omega Q(x,a))}{\sum_{a\in A_x}\exp(\omega Q(x,a))} - \ln(\frac{1}{|A_x|}\sum_{a\in A_x}\exp(\omega Q(x,a)))}{\omega^2} \end{aligned} \tag{38}$$

Thus, the Mellowmax function is differentiable, which allows gradient-based optimizations like adaptive Mellowmax deep Q-learning.

In the discipline of deep reinforcement learning, Kim et al. [17] have challenged the importance of the delayed update of a separate target network. Because of the delayed update, the action-value functions are not continually updated, which hinders faster learning. Besides, a separate target network requires extra memory. Since a major contribution of a separate target network is to stabilize learning and reduce bias, the introduction of

Mellowmax operator can alleviate the overestimation bias and stablize learning due to its non-expansion property, which may ultimately undermine the necessity to include the target network in the DeepMellow algorithm. Hence, it is reasonable for DeepMellow to remove the target network and delayed update to save memory and speed up learning process.

Despite its arguably advantage over the DQN algorithm, implementing DeepMellow can be a computationally cumbersome task in that there is an additional yet crucial hyper-parameter ω needs to be specified in advance. If ω is too large, Mellowmax is like a maximization/minimization operator and fails to alleviate the overestimation problem. On the contrary, a extremely small ω will behave like an arithmetic average operator assigning equal probabilities to each action in the action space, which ignores the information that the agent has learned from past experiences. A moderate value of ω clearly outperforms extremes, but the optimal ranges of ω vary with model settings [17].

Kim and Konidaris [16] have proposed an adaptive method to tune for ω in practice shown in Algorithm 4, in which 2 batches of transitions will be sampled from the replay buffer. One is to learn the Q-network, and the other is to tune the temperature parameter ω.

A frequently encountered problem in the algorithm implementation is the overflow caused by the exponentiation operation in Mellowmax. Extreme large values of exponents yield either infinity or 0 and taking logarithm of infinity or 0 is likely to produce an error in gradients computation. In this sense, Mellowmax may not only fail to relieve the overestimation problem but also potentially result in a new issue. A practical solution is to shift the original values obtained from the Q-network by a constant prior to taking exponentiation

Algorithm 4: Mellowmax deep Q-learning (DeepMellow) with adaptive tuning for ω

Data: Mellowmax temperature parameter: ω, experience replay size: N_{buffer}, mini-batch size for updating Q-network : K, mini-batch size for updating ω: M, exploration parameter: β, discount factor: γ

initialize Q with random weights ϕ;
initialize an empty experience replay buffer $\mathcal{D}$;
initialize ω and a reference parameter $\overline{\omega}$;
for *episode* $i = 1, 2, 3, \ldots$ **do**
    randomly initialize $x_{i,0} \in S$;
    for $t = 0, 1, 2, \ldots$ **do**
        choose action $a_{i,t}$ following a pre-specified exploration strategy;
        observe the next state $x_{i,t+1}$ and cost $c_{i,t+1}$;
        store $(x_{i,t}, a^{i,t}, c_{i,t+1}, x_{i,t+1})$ in the replay buffer $\mathcal{D}$;
        sample a mini-batch of transitions from $\mathcal{D}$ to update Q-network;
        sample another mini-batch of transitions from $\mathcal{D}$ to update ω;
        update Q-network using equation (31);
        update ω by minimizing $J = \frac{1}{2}(r^m + \gamma\, mellowmax_{\omega} Q(x, \cdot) - Q(x_m, a_m))^2$;
        if $x_{i,t+1} = s^{terminal}$ **then**
            break;
        end
    end
end
return Q

and then shift back by the same constant. The shifted Mellowmax is:

$$mm_\omega(x) = \frac{1}{\omega} log(\frac{1}{|A_x|} \sum_{a \in A_x} exp(\omega(Q(x,a) - c))) + c, \tag{39}$$

where the constant c is the maximum of $Q(x, \cdot)$ as suggested by Asadi and Littman [3]. In practice, we take c as the minimum of $Q(x, \cdot)$.

2.4 Equilibrium policy as a threshold policy

As discussed in Section 1, [13, 12] have established a threshold property of the equilibrium policy in the binary action spaces model. However, the policy obtained from tabular Q-learning algorithms usually does not demonstrate a perfect threshold property in that there may be an interval $\theta \in S$ in which the optimal policy $\pi^*(x), x \in \theta$ keeps changing according to x.

Suppose the state space S is partitioned into K states: $\{s_0, s_1, \ldots, s_K\}$. In order to construct a threshold policy based on the Q-table, we can estimate the threshold by segmented regression. Without a threshold in S, the optimal policy will remain the same for $s_i, \forall i \in \{1, 2, \ldots, K\}$. Assume, without loss of generality, the optimal action to be a_0 and the value function $V(s_i) = \min_{a \in A} Q(s_i, a) = Q(s_i, a_0) \forall i \in \{1, 2, \ldots, K\}$. The segmented regression of the value function will return a break point only if the regression of the action value function associated with action a_0 has a break point. If both the regressions of $Q(s_i, a_0)$ and $Q(s_i, a_1)$, $i = 1, 2, \ldots, K$ have no break points, it is reasonable to assert that the break point obtained from the segmented regression of the value function indicates a change of the optimal action, and thus it can serve as an estimate of the threshold (segment

threshold).

Another method to estimate the true threshold is performing regression the action-value function associated with each individual action and then find the intersection of the regression functions. If there is no intersection between two regression curves, then there will be no threshold, since a certain action will always outperforms the other. If there is only one intersection, say $s^* \in S$, then the optimal action will only change at s^*, so the intersection can also serve as an estimate of the threshold (intersect threshold).

Because it is the estimated threshold policy rather than the Q-table that determines the value function, we cannot directly apply equation (2) and (15) to obtain the performance of the policy. It is possible that following the threshold policy, the agent may choose a sub-optimal action yielding a higher cost in the Q-table. The performance of a threshold policy assuming uniform initial condition on the state distribution is $J^{\pi_{threshold}} = \frac{1}{K}\sum_{i=1}^{K} Q(s_i, a)$, where a follows a threshold policy as outlined in equation (14).

It is striking that the intersect threshold is in general higher than the segment threshold, and therefore the performance following the intersect threshold policy is slightly different from that following the segment threshold policy. However, the relationship between the performance of two kinds of threshold policies is hard to determine. Denote the intersect threshold and segment threshold by θ_1 and θ_2, respectively.

$$J^{\pi_{\theta_1}} - J^{\pi_{\theta_2}} = \sum_{i=0}^{K_1} Q(s_i, a_0) + \sum_{i=K_1+1}^{K} Q(s_i, a_1) - \sum_{j=1}^{K_2} Q(s_j, a_0) - \sum_{K_2+1}^{K} Q(s_j, a_1),$$

where K_1 and K_2 are the largest integers such that $\pi(s_{K_1}) = a_0, \pi(s_{K_2} = a_0)$. Without loss

of generality, we assume $\theta_1 < \theta_2$. We have:

$$\begin{aligned} J^{\pi_{\theta_1}} - J^{\pi_{\theta_2}} &= -\sum_{i=K_1+1}^{K_2} Q(s_i, a_0) + \sum_{i=K_1+1}^{K_2} Q(s_i, a_0) \\ &= \sum_{i=K_1+1}^{K_2} (Q(s_i, a_1) - Q(s_i, a_0)). \end{aligned}$$

Thus, the sign of $J^{\pi_{\theta_1}} - J^{\pi_{\theta_2}}$ can be either positive or negative depending of the Q-table.

Owing to the uniqueness of the threshold in the binary action spaces model, having multiple break points or intersections implies a problem either in the learning of the Q-table or the regression procedures.

3 Policy-Based Methods

Based on the binary action spaces model, Jayakumar Subramanian and Aditya Mahajan [27] have proposed a policy-based algorithm to find the mean-field equilibrium. The basic idea behind the algorithm is as follows. After randomly choosing an initial guess of the policy parameter θ_0 and mean-field z_0, we can evaluate the performance $J^{\pi_\theta, z}$ of this pair of parameters and iteratively update using two-timescale stochastic approximation defined as:

$$\begin{aligned} z_{k+1} &= z_k + \beta_k[\hat{z}_{k+1} - z_k], \\ \theta_{k+1} &= \theta_k + \alpha_k \hat{G}_{z_k, \theta_k}, \end{aligned} \tag{40}$$

where $\hat{z}_{k+1}(x) = \frac{1}{m}\sum_{j=1}^{m} \mathbb{1}(x^j = x)$ is an estimate of the mean-field at time $k+1$ and $\hat{G}_{z_k,\theta_k}$ is an estimate of the gradient of $J^{\pi_\theta, z}$. Moreover, Subramanian and Mahajan [27] suggest

an asynchronous update for θ_k, z_k. That is to say, a prespecified number of times will be iterated to find $\hat{z}_{k+1}$ at time k with β_k being fixed at 1, and then algorithms such as simultaneous perturbation stochastic approximation may be applied to estimate the gradient of the performance $J^{\pi_\theta, z}$ and update θ_{k+1}.

The process of calculating the fixed mean-field distribution z_k, i.e., stationary, for every iteration of θ_k is shown in Algorithm 5 assuming the knowledge of the equilibrium policy being a threshold policy. The algorithm takes an initial distribution of states, a random but fixed policy θ, the number of iterations for each step of updating policy B, and the batch size m as input and returns a mean-field distribution. Monte Carlo policy evaluation can be used to evaluate the policy performance in line 8 and 9 in Algorithm 6.

In the binary action spaces model, the transition dynamics depend only on the empirical mean of the mean-field distribution, so it is superfluous to have the full information of the mean-field distribution. Line 7 and line 9 in Algorithm 5 can be replaced by computing the empirical mean of $z_{i,t+1} \approx \frac{1}{N} \sum_{j=1}^{N} x_{i,t+1}^{j}$, which produce a scalar output. We can hereby avoid discretizing the continuous state space $[0, 1]$ and reduce the complexity of the algorithm.

Once the stationary mean-field z has been found, we turn to find a good estimation for the gradient of the performance. In the case when the policy function π is not directly differentiable with respect to θ, the simultaneous perturbation stochastic approximation (Algorithm 6) is a good candidate algorithm for gradient approximation and optimization with the presence of noise [25].

The state $X_{i,t} = x_{i,t}^1, x_{i,t}^2, \ldots, x_{i,t}^N$ and action $A_{i,t} = a_{i,t}^1, a_{i,t}^2, \ldots, a_{imt}^N$ in algorithm 1. The Rademacher distribution can be obtained by the transformation $2X - 1$, where X is the

Algorithm 5: Stationary mean-field

Data: fixed policy parameter: θ, initial distribution: ξ, B: iteration count, m: batch size

Result: stationary mean-field z

for $i = 1, 2, \ldots, m$ **do**
  sample $X_{i,0} \sim \xi$;
  set $z_{i,0} = \xi$;
  for $t = 0, 1, \ldots, B$ **do**
    sample actions $A_{i,t} \sim \pi(X_{i,t}, \theta, z_{i,t})$;
    sample next state $X_{i,t+1} \sim P(X_{i,t}, A_{i,t}, z_{i,t})$;
    set $z_{i,t+1}(x) = \frac{1}{N} \sum_{j \in N} \mathbb{1}(x^j_{i,t+1} = x)$;
  end
  the limiting distribution of $z_i(x) = z_{i,B+1}(x)$
end
return $z(x) = \frac{1}{m} \sum_{i=1}^{m} z_i(x)$

Algorithm 6: Simultaneous perturbation stochastic approximation

Data: policy parameter: θ, initial distribution: ξ, B: mean-field iteration count, m: mean-field batch size, K: iteration count

Result: estimated equilibrium: $(\theta^\star, z^\star)$

for *iteration* $k = 1, 2, \ldots, K$ **do**
  z_k = Stationary mean-field(θ,ξ,B,m);
  η_k = Rademacher(± 1);
  $c_k = \frac{c}{k^{\gamma_k}}$;
  $a_k = \frac{a}{(A+k)^{\alpha_k}}$;
  $\theta^+_k = \theta + \eta\, c_k$;
  $\theta^-_k = \theta - \eta\, c_k$;
  $J^{\theta^+}_k$ = PolicyEvaluation(θ^+_k, ξ, z_k);
  $J^{\theta^-}_k$ = PolicyEvaluation(θ^-_k, ξ, z_k);
  $\hat{G} = \frac{J^+_k - J^-_k}{2 c_k \eta}$;
  $\theta_{k+1} = \theta_k - a_k \hat{G}$
end
return θ_{k+1}

Bernoulli distribution with $p = .5$. The gain sequences a_k and c_k are crucial to the performance of SPSA. Fortunately, Spall [25] provides detailed guidance on how to construct these sequences to make the algorithm converge effectively and accurately. According to Spall:

$$\begin{aligned} a_k &= \frac{a}{(A+k)^\alpha} \\ c_k &= \frac{c}{k^\gamma}, \end{aligned} \tag{41}$$

where $\alpha = .602$ and $\gamma = .101$ for small iteration count K and $\alpha = 1$ and $\gamma = \frac{1}{6}$ for large K are practically effective choices. It is recommended to choose A and a by the set of equations: $A = \frac{K}{10}$ and $\frac{a}{A+1}\hat{G} = \Delta\theta$ and set c roughly equal to the standard deviation of the measurement noise in the objective function (the performance function J in this example). Because $\hat{G}$, $\Delta\theta$, and the standard deviation of noise are unknown *a priori*, running $J^{\pi_{\theta},z}$ several times may help us estimate c, and this estimate can further determine a by simulating J^{θ^+} and J^{θ^-} [26].

In practice, the set of parameters $a = 0.5, A = 10, c = 0.15$ works well for both the discretization and non-discretization case. We have repeated the algorithm 10 times for each case and summarized the results in figure 1 with the 95% confidence intervals plotted in the shaded regions. The horizontal black line denotes the theoretical solutions as computed in [12]. The graphs show that the policy, performance, and mean-field almost converge to the theoretical solution within 25 steps and the variations across multiple runs all decreases rapidly.

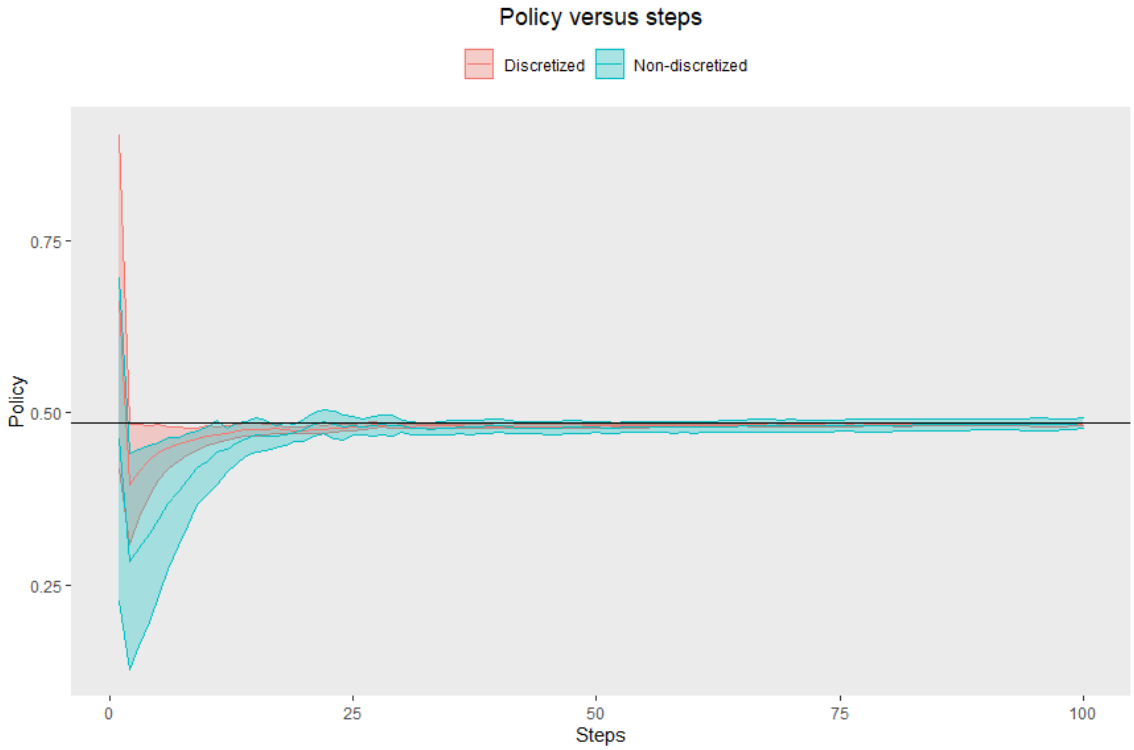
It is worth noting that the non-discretization method yields results generally closer to the theoretical optima despite higher variation when calculating the threshold policy

parameter θ. As can be seen from 1b, the SPSA algorithm with discretization tends to underestimate, though slightly, the cost at almost every step, which agrees with our finding in 1d that the value function without discretization is marginally higher, particularly when the state is below 0.25.

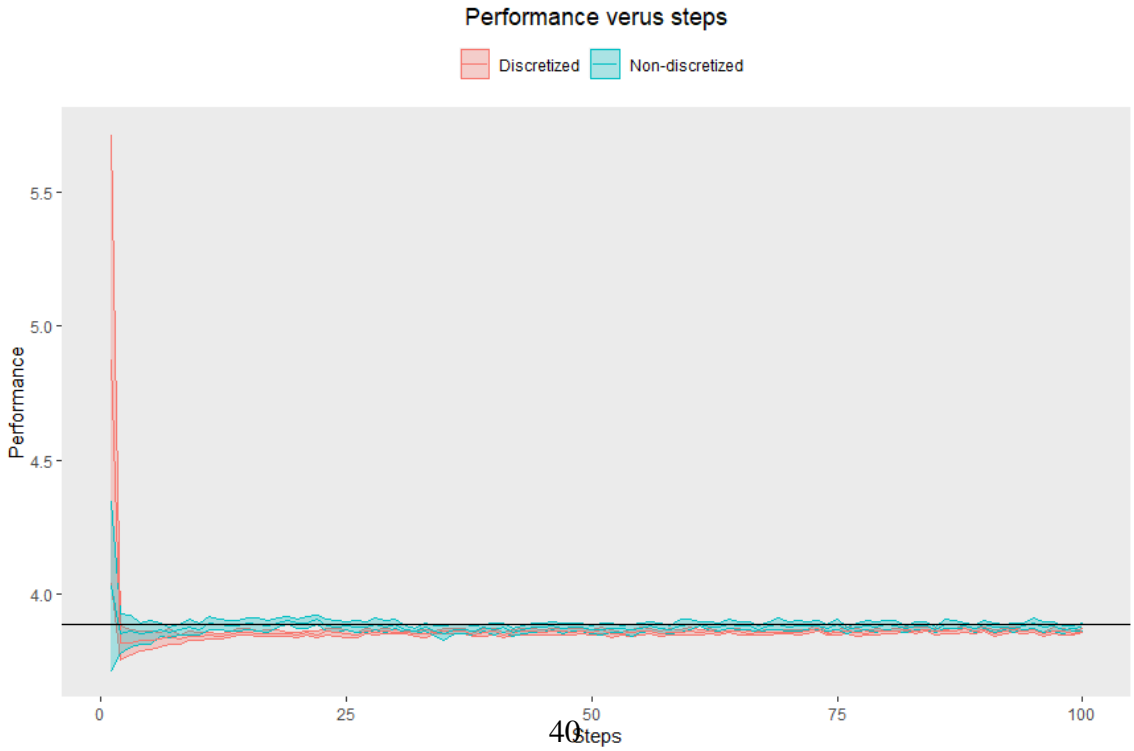
In general, the gradient-based algorithm using simultaneous perturbation stochastic approximation works satisfactorily efficiently and accurately, and discretizing the continuous state space does not result in a significant loss of accuracy.

4 Tabular Q-Learning Methods

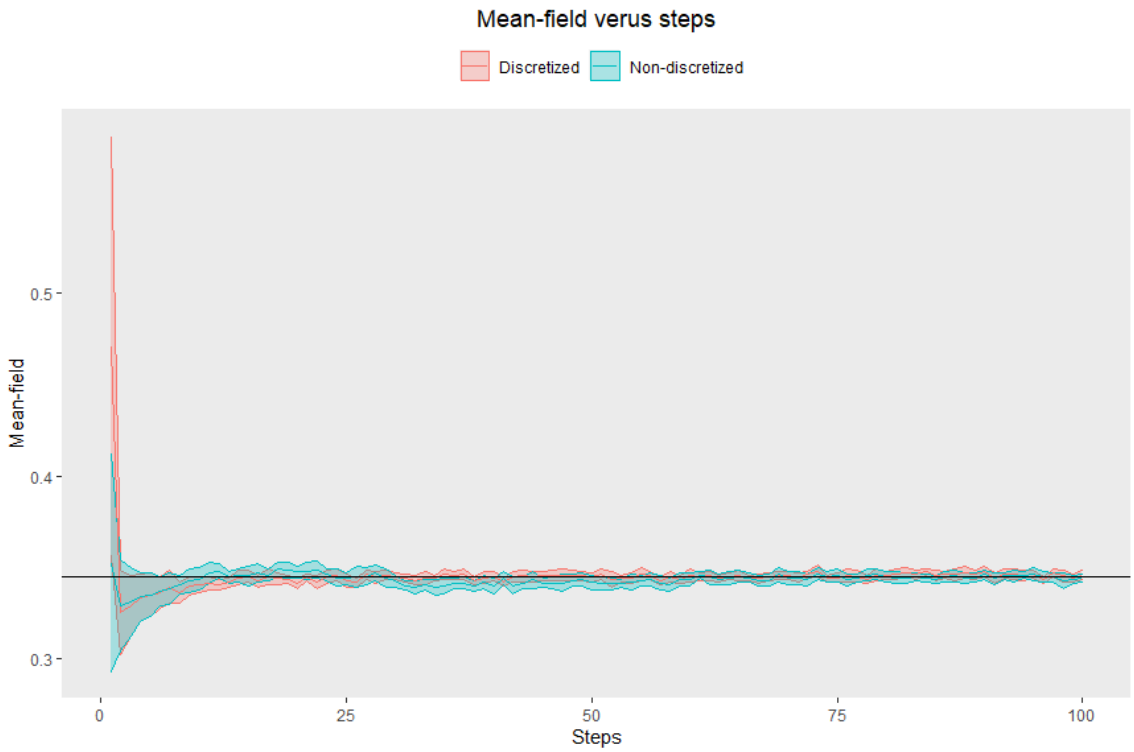
In the following sections, two tabular Q-learning algorithms to solve the mean-field game of the binary action spaces model, Angiuli, Fouque, and Lauriére's two tome-scales mean-field Q-learning (U2-MF-QL) and Guo, Hu, Xu, and Zhang's general mean-field Q-leaning (G-MFQ), will be presented with numerical examples. These two algorithms are based on tabular Q-learning, so the state space S will be uniformly discretized into 101 cells, i.e., $S = \{0, .01, .01, \ldots, .99, 1\}$. Equation (15) will be used to compare the performance of different algorithms.



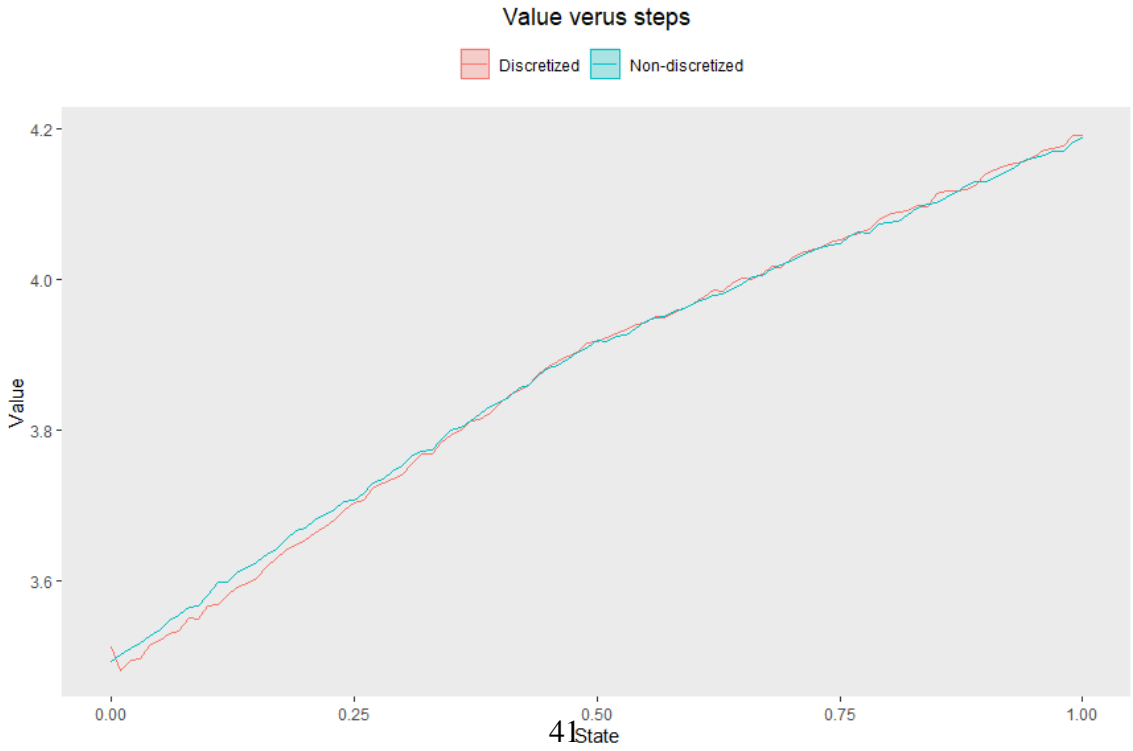
(a) Convergence of the threshold policy



(b) Convergence of the performance



(c) Convergence of the mean-field



(d) The final state-value function

Figure 1: Policy based method with SPSA algorithm

4.1 Unified Two-Timescale Mean Field Q-learning

Similar to Subramanian and Mahajan, Angiuli, Fouque, and Lauriére [2] are also inspired by the two-timescale approach:

$$\begin{aligned} Q_{k+1}(x_k, a_k) &= (1 - \alpha_k^Q) Q_k(x_k, a_k) + \alpha_k^Q (c(x_k, \mu_k, a_k) + \gamma \min_{a' \in A_{x_k}} Q_k(x_{k+1}, a')), \\ \mu_{k+1} &= (1 - \alpha_k^\mu)\mu_k + \alpha_k^\mu \delta(x_{k+1}), \end{aligned} \tag{42}$$

where α^Q and α^μ are learning rates not only controlling the convergence speed of Q-function and state distribution μ but also determining whether the output of the algorithm is within mean-field game or mean-field control framework. It has been proved that if $\alpha_k^\mu < \alpha_k^Q$, then the Q-function converges faster and the algorithm solves a mean-field game problem, or otherwise a mean-field control problem will be solved.

Angiuli, Fouque, and Lauriére's algorithm assumes an episodic structure of the model dynamics. Since our binary action spaces model is non-episodic, the episode in the algorithm can be considered as an update frequency for the learning rate of the state distribution μ. In other words, the learning rate for the Q-table will be updated every step but that of μ will be updated every N_{epi} steps.

Since implementing the algorithm requires finiteness of time steps, state spaces, and action spaces, some truncation and discretization procedures are necessary to ensure the feasibility of tabular Q-learning algorithms. In the binary action spaces model, the shared state space S is continuous and it will be uniformly discretized with step size $\Delta = .01$ into 101 states: $\{s_1, s_2, \ldots, s_{101}\}$. It is noteworthy that there is no general guideline on how a continuous space should be projected onto a finite space, and discretization procedures

Algorithm 7: Unified Q-learning

Data: episode length: N_{epi}, finite state space :S, finite action space: A, initial state distribution: ξ, convergence tolerance for Q-function: tol_Q, convergence tolerance for state distribution: tol_μ

Result: estimated Q-function and state distribution μ

initialization: $Q_0(x,a) = 0$ for all $(x,a) \in S * A$, $\xi = [\frac{1}{|S|,\ldots,|S|}]$ unless otherwise specified;

set $\mu_{0,N_{epi}} \leftarrow \xi$;

for *episode k = 1,2,3,...* **do**

  initialization: sample $x_{k,1} \sim \mu_{k-1,N_{epi}}$, set $Q_k = Q_{k-1}$;

  for $t = 1,2,3,\ldots,N_{epi}$ **do**

    update $\mu_{k,t}$ using equation (42) and (43);

    sample action $a_{k,t} \sim Q_k(x_{k,t,\cdot})$;

    observe the next state $x_{k,t+1}$;

    receive cost $c(x_{k,t}, \mu_{k,t}, a_{k,t})$;

    update $Q_k(x_{k,t}, a_{k,t})$ using equation (42) and (43);

  end

  if $\|\mu_k - \mu_{k-1}\| < tol_\mu \quad \&\& \quad \|Q_k - Q_{k-1}\| < tol_Q$ **then**

    break

  end

end

return Estimated Q-function and μ

may vary according to the specific architecture of models of interest.

Picking coefficients of learning rates are of greater importance in the unified Q-learning algorithm, as the learning rates decide which type of problem the algorithm is trying to solve. Following the theoretical results in [7], Angiuli, Fouque, and Lauriére define the learning rates:

$$\begin{aligned} \alpha^{Q}_{k,t,x,a} &= (\frac{1}{1+\#(k,t,x,a)})^{\omega^{Q}}, \\ \alpha^{\mu}_{k} &= (\frac{1}{1+k})^{\omega^{\mu}}, \end{aligned} \tag{43}$$

where $\#(k,t,x,a)$ denotes the number of times the representative agent visits a state-action pair (x,a) until episode k and time t in the episodic setting, or until time t in the k^{th} update of Q-table in the non-episodic setting. In addition, ω^{Q} is restricted within $(\frac{1}{2},1)$, while ω^{μ} can take values within $(0,1)$.

Unlike Algorithm 5 and Algorithm 6, in which the number of participating agents N is included, the unified Q-learning algorithm only takes the state distribution as input and approximates the mean-field using $z = \frac{1}{101}\sum_{i=1}^{101}\mu(s_i), i = \{1,2,\ldots,101\}$. At step t, the generic agent updates his/her estimate of the mean-field using a one-hot vector of the observed next state $x_{t+1} \in S$ and receives a cost depending on his/her chosen action, current state, and the mean-field, with which the Q-table will be updated. The update rule for the Q-table can be altered to enable other update methods such as double Q-learning and Mellowmax Q-learning.

In order to visually compare different implementations of the unified mean-field Q-learning, we have repeated Algorithm 7 for 10 times using Mellowmax Q-learning that replaces the minimization operator in equation (42) with Mellowmax operator as defined

in equation (39), 10 times using basic Q-learning without modifications applied to equation (42), and 10 times using double Q-learning that combines equation (29) and (42). Each sample run consists of 200 episodes (outer iterations) and each episode is made up of 30000 steps (inner iterations). ω^Q and ω^μ are fixed at 0.55 and 0.75, respectively, for all time steps. The additional hyper-parameter ω in the Mellowmax operator stays at -500. The shaded regions denote the 95% confidence intervals of the estimation.

To begin with, we will examine which estimate of the true threshold performs better in unified mean-field Q-learning. Figure 2 shows two threshold policies in contrast with the theoretical optimal threshold. Also, the state observing the first change in the optimal policy is plotted.

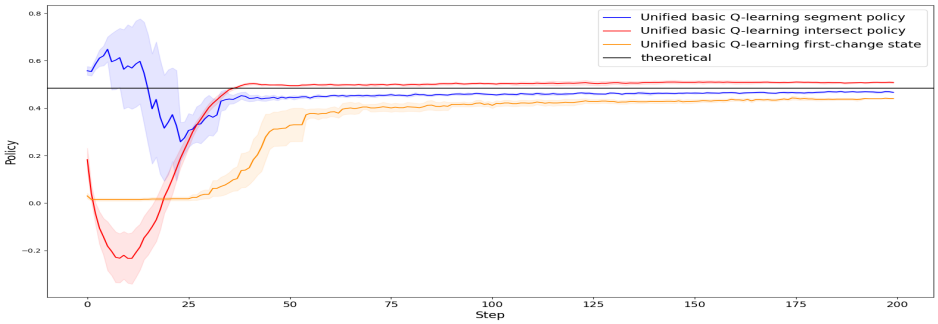
As discussed in Section 2.4, the intersection threshold is always higher. However, we cannot tell if either threshold is superior to the other. As can be seen from the figure, three algorithms all indicate a bias in estimating the true threshold, and the first-change state clearly goes furthest away from the optimal. The mean-squared errors are summarized in the table below:

	Basic Q-learning	Double Q-learning	Mellowmax Q-learning
Segment Threshold	0.0098	0.0038	0.0035
Intersect Threshold	0.0437	0.002	0.003

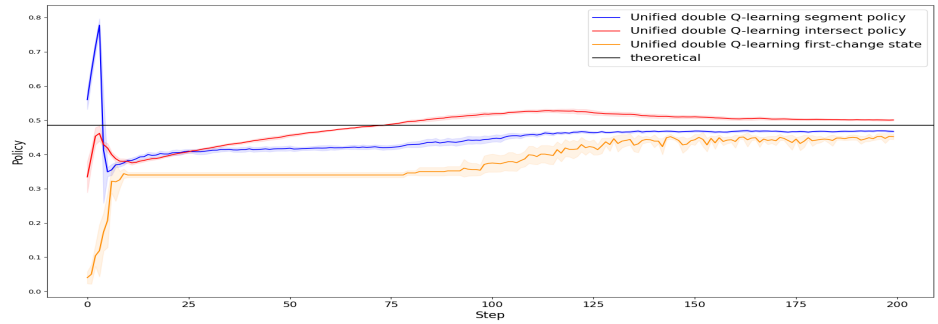
Table 1: Mean-squared errors of threshold estimation using unified Q-learning

It can be concluded from the table that double Q-learning and Mellowmax Q-learning help us find a better estimation of the true threshold of the optimal policy.

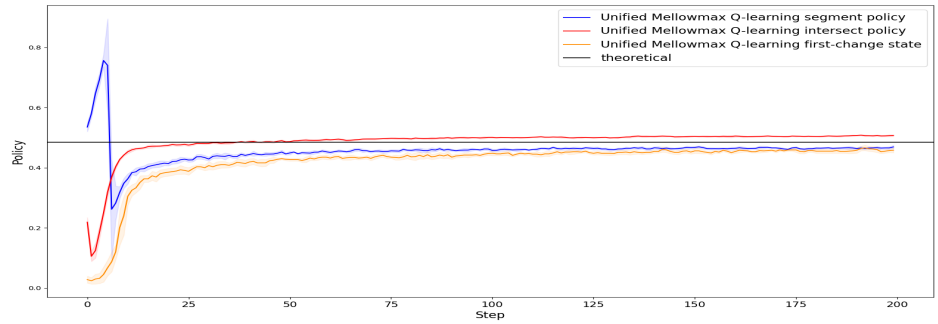
Figure 3 depicts the performance of unified Q-learning using different threshold estimations and Q-table update rules. It is interesting that it is the update rule that dominantly



(a) Unified basic Q-learning policy



(b) Unified double Q-learning policy



(c) Unified Mellowmax Q-learning policy

Figure 2: Policy comparison of different thresholds

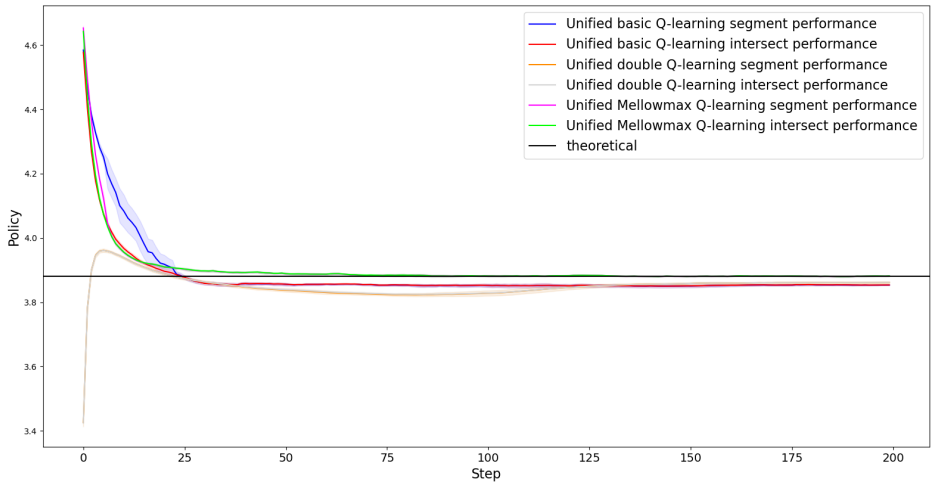


Figure 3: Performance of unified Q-learning using different threshold

determines the performance and the difference between performance following the segment policy and intersection policy is almost visually negligible. In the long run six lines arguably converge to three lines corresponding to three types of update rules for the Q-table. Although double Q-learning theoretically outperforms basic Q-learning in that it may avoid bias, the performance of basic Q-learning is closer to the theoretical optimum in the first 100 runs. In particular, from step 50 to step 100 the performance of double Q-learning experiences an unexpected deterioration. Taken together, Figure 2 and 3 suggest a preference for Mellowmax Q-learning in spite of its relatively longer running time.

Overall, these results indicate that the unified Q-learning algorithms all learn the optimal policy within 100 steps. As discussed above, the basic Q-learning may suffer more severely from bias due to the minimization operation, and both double Q-learning and Mellowmax Q-learning are supposed to alleviate the problem. This can be verified by the

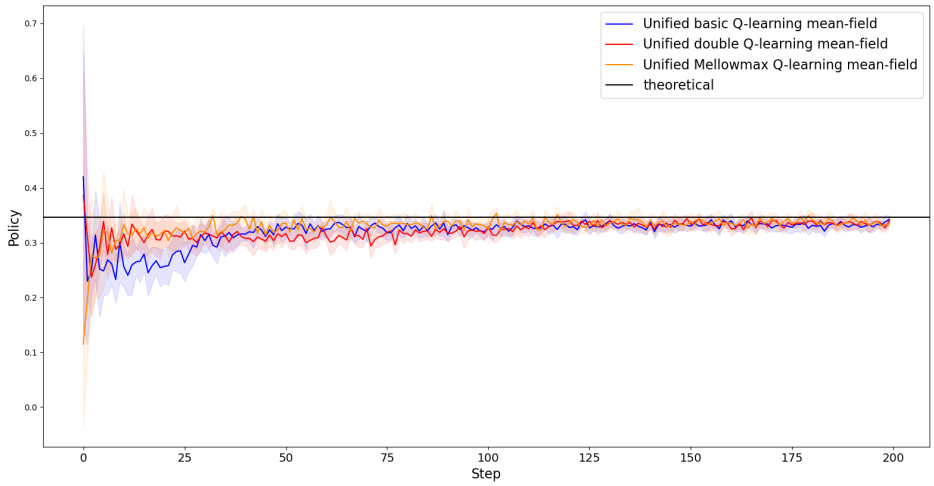


Figure 4: Convergence of the mean-field using unified Q-learning

graph that the performance of equilibrium policy derived from the basic Q-learning algorithm deviates the most from the theoretical equilibrium, despite unnoticeable differences in policy and mean-field. The double Q-learning algorithm relieves the overestimation problem to some extent, but the Mellowmax Q-learning performs the best among these 3 candidate algorithms in terms of policy and performance.

4.2 General Mean-Field Q-Learning

The general mean-field games framework in Guo's [9] G-MFQ algorithm extends to the case where the reward function and environment dynamics depend not only on the current state x_t, chosen action a_t, and state distribution μ_t but also on the actions of all participating players. The iterations are on the joint state-action distribution $\mathcal{L}$ instead of only its marginal state distribution μ_t.

Algorithm 8: GMF-Q

Data: Q-learning maximum steps: T, finite state space: S, finite action space: A, initial joint state-action distribution: L, softmax temperature parameter: c,

Result: estimated Q-function and state distribution μ

initialization: L_1;

for $k = 1, 2, 3, \ldots, T$ **do**

  Perform Q-learning for T_k iterations to estimate Q-table Q_k based on the fixed joint distribution L_k;

  Compute the policy $\pi_k(x) = softmax_c(Q_k(x, \cdot))$;

  Sample $s \sim \mu_k$, the state marginal distribution of L_k;

  Update $\tilde{L}_{k+1}$ based on s, π_k, L_k;

  Find $\mathcal{L}_{k+1} = \textbf{Proj}(\tilde{\mathcal{L}}_{k+1})$

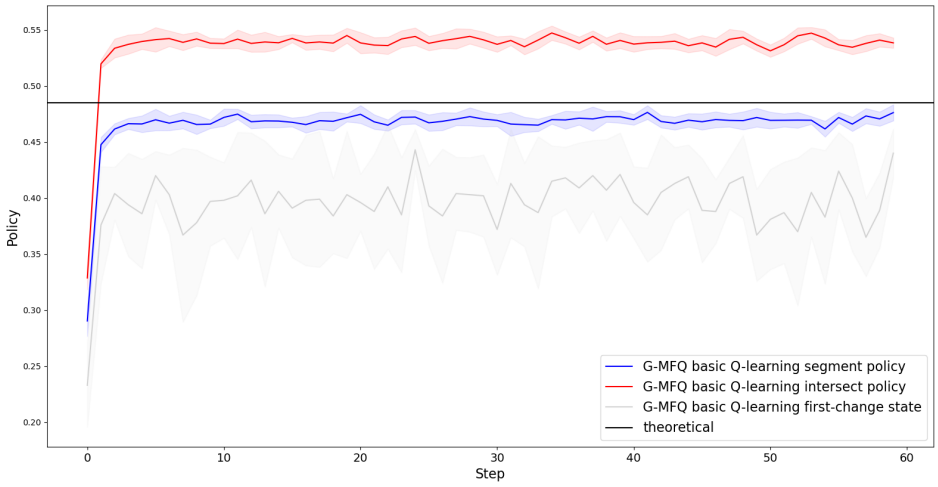
end

return Estimated Q-function and L

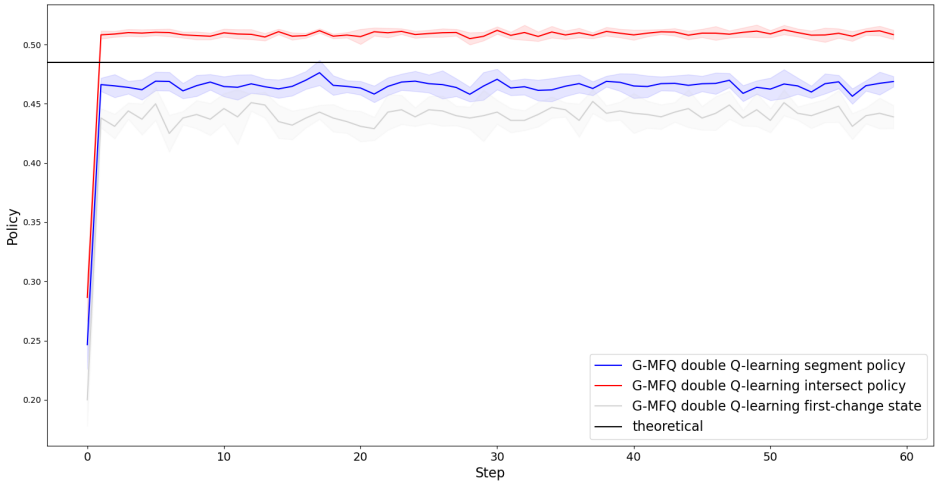
A major difference between the GMF-Q algorithm and the Unified 2 time-scales mean-field Q-learning algorithm is that the former learns a new Q-table for each step in the outer iteration, but the latter only learns a single Q-table throughout the whole learning process. Although the G-MFQ algorithm may require fewer steps in the outer iteration, it may take longer execution time for the G-MFQ algorithm because of its learning process within the inner iteration.

The GMF-Q algorithm also differs from the Unified Two Time-Scales Mean-Field Q-learning algorithm (Algorithm 8) in that the joint state-action distribution $\mathcal{L}$ in the GMF-Q algorithm will not be updated until the agent learns the new Q-table, but the state distribution μ and the Q-table in the latter will be updated each step within the inner loop simultaneously.

For each step within the outer iteration, the representative agent, given the fixed joint distribution $\mathcal{L}$, will start over and learn the Q-table using various tabular Q-learning algo-

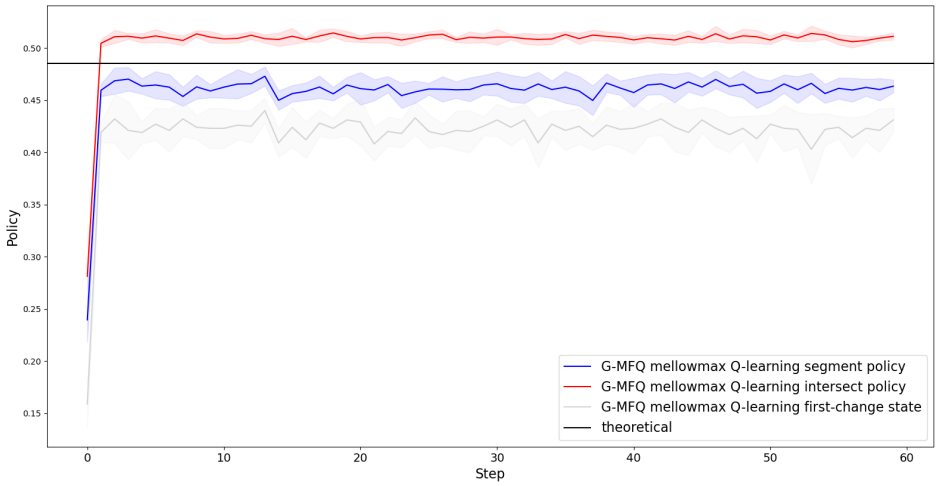


(a) G-MFQ basic Q-learning policy



(b) G-MFQ double Q-learning policy

rithms. The learning rate in Q-learning algorithms is $|\#(x,a,t)+1|^{-h}$, where $\#(x,a,t)$ is the number of times that the agent visits the state-action pair (x,a) prior to time t within each outer iteration and $h \in (\frac{1}{2},1)$ controls the convergence rate. Once an estimate of the



(c) G-MFQ mellow max Q-learning policy

Figure 5: Estimated threshold using G-MFQ

Q-table has been obtained, the agent will compute the Boltzmann policy given the Q-table and the environment will evolve according to this policy. The temperature parameter in the Boltzmann policy exploration is shown to be robust, but our experiment suggest $c = 2$ may work better. The projection in line 7 is defined as

$$\mathbf{Proj}_{S_\epsilon}(\mathcal{L}) = \underset{L^{(1)},\ldots,L^{(N_\epsilon)}}{\arg\min}\, d_{TV}(\mathcal{L}^{(i)}, \mathcal{L}). \tag{44}$$

In practice, it is sufficient to truncate $\tilde{\mathcal{L}}_k$ to a certain number of digits [9].

To delve deeper into to what extent do different Q-table update rules change the results obtain from Algorithm 8, we conduct 10 experiments and exhibit the results in Figure 5 and 6. Each experiment consists of 60 episodes and each episode is made up of 400,000 steps for basic Q-learning and double Q-learning and 200,000 steps for Mellowmax Q-

learning. The shaded regions are 95% confidence intervals based on these 10 experiments. Because of a complete Q-learning process within each episode, Algorithm 8 takes much longer to execute as compared to Algorithm 8.

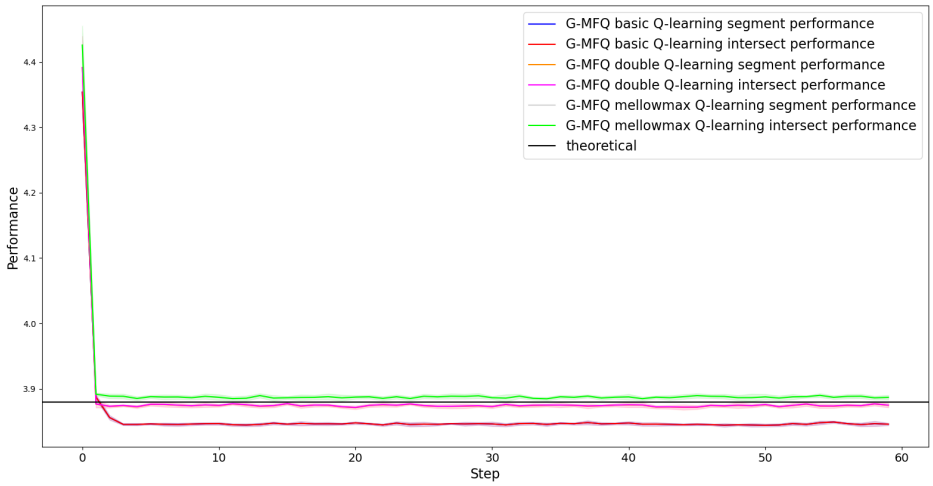
The estimated thresholds and the true threshold are shown in Figure 5. The lines referring to intersect thresholds are always higher than those those referring to segment thresholds, which is consistent with our findings in Section 4.1. As can be seen from the figure, Algorithm 8 converges extremely fast, as there is not significant improvement after 5 episodes.

Noticeably, Mellowmax Q-learning with $\omega = -500$ and double Q-learning are close to each other in both policy estimation and performance estimation, with basic Q-learning deviating furthest from the theoretical equilibrium, particularly the intersect threshold obtained from basici Q-learning, which is rough 0.55. The mean-squared errors are summarized in Table 2.

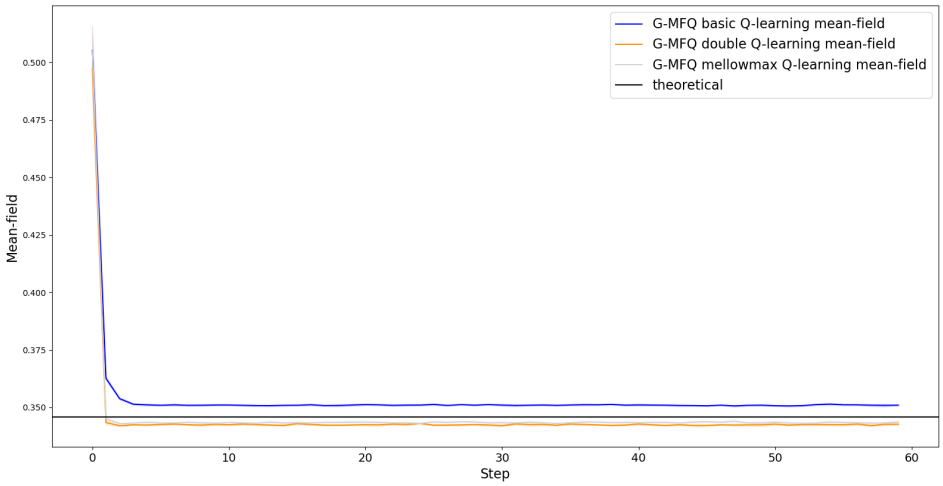
	Basic Q-learning	Double Q-learning	Mellowmax Q-learning
Segment Threshold	0.001	0.0015	0.0018
Intersect Threshold	0.0034	0.0013	0.0013

Table 2: Mean-squared errors of threshold estimation using G-MFQ

That Mellowmax Q-learning and double Q-learning perform better when compared with the theoretical equilibrium undoubtedly supports the assertion that these two methods are effective in reducing overestimation bias. Since we run the Mellowmax Q-learning algorithm only 200,000 steps within each episode, it is hard to determine if Mellowmax Q-learning will perform better when more training steps are available and eventually out-



(a) G-MFQ Q-learning performance



(b) G-MFQ Q-learning Mean-field

Figure 6: Estimated mean-field and corresponding performance using G-MFQ

perform double Q-learning.

5 Neural Network Methods

To get rid of discretizing the continuous state space, we can use neural networks to approximate action-value functions and policy functions. Yang et al. [30] have developed practical algorithms combining deep Q-learning and mean-field theory. The mean-field action-value function (Q-function) incorporating the influences from other players on the i^{th} player at time t is $Q_t^j(x_t^i, a_t^i, \mu_t)$. Yang et al. modify equation (25) by replacing $\min Q(x_{t+1}, \cdot)$ with the mean-field value function, because in their model settings the reward (cost) depends on the actions of neighboring players and there is no guarantee that the optimal action is attainable. For example, we assume that the cost will be the smaller if the percentage of local agent choosing the same action become higher. The action a will be the optimal action for the i^{th} agent if the majority of the local agents prefer the action a. Thus, the optimal action for the i^{th} agent is jointly determined by all agents.

Nevertheless, in our binary action spaces model, the cost depends only on the current state-action pair and the mean-field state, which are deterministic at time t, so we are not obliged to follow the exact the same update rule as in Yang et al. [30] and propose our own version of the mean-field actor-critic algorithm.

Algorithm 9: Mean-Field Actor-Critic

Data: Target network update frequency : C, mini-batch size : K, target network update rate : τ

Initialize a reply buffer $\mathcal{D}$, critic network Q_ϕ and Q_{ϕ^-}, and actor network π_θ and π_{θ^-};

while *training continues* **do**

  For each player $x^j = s$, samples an action a using π_θ, receives a cost c, and observe the next state s' and the mean-field state $\bar{s}$;

  Store the transitions $(s, a, c, s', \bar{s})$;

  Sample a mini-batch of K transitions from $\mathcal{D}$;

  Update the critic network Q_ϕ;

  Update the actor network π_θ;

  if $\#(steps) \mod C = 0$ **then**

    Update the target critic network Q_{ϕ^-};

    Update the target actor network π_{θ^-};

  end

end

We test 4 algorithms in this section: Mean-Field Double Deep Q-learning (DQN), Mean-Field Actor-Critic (MFAC), DeepMellow with fixed ω, and DeepMellow with adaptive ω.

In order to implement these algorithms within the mean-field games framework, we are ought to slightly modify the algorithms presented in Section 2. The mean-field system is made up of $N = 100$ players whose states at time step t are denoted by $\mathbf{X}_t = x_t^1, x_t^2, \ldots, x_t^N$.

The players interact with the environment, choose actions $\mathbf{A}_t = a_t^1, a_t^2, \ldots, a_t^N$, receive cost $\mathbf{C}_{t+1} = c_{t+1}^1, c_{t+1}^2, \ldots, c_{t+1}^N$ from the environment, and proceed to the next state $\mathbf{X}_{t+1} = x_{t+1}^1, x_{t+1}^2, \ldots, x_{t+1}^N$ following the environment dynamics as specified in Section 1.3. The transition quadruple $(\mathbf{X}_t, \mathbf{A}_t, \mathbf{C}_{t+1}, \mathbf{X}_{t+1})$ will be stored in a replay buffer of size $100 \cdot 1000 = 100000$. A mini-batch of $K = 128$ individual experiences will be sampled from the buffered at each time step to update the Q-network. In mean-field deep Q-learning and mean-field actor-critic, the separate target network will be fixed for 10 steps and then updated using equation (32) with $\tau = (1 + |\#(steps)|)^{-.7}$. The action will be chosen by the actor network in mean-field actor-critic and $\epsilon-$greedy exploration in other 3 algorithms. ϵ starts from 1 and decays to .1 with the decay rate .999.

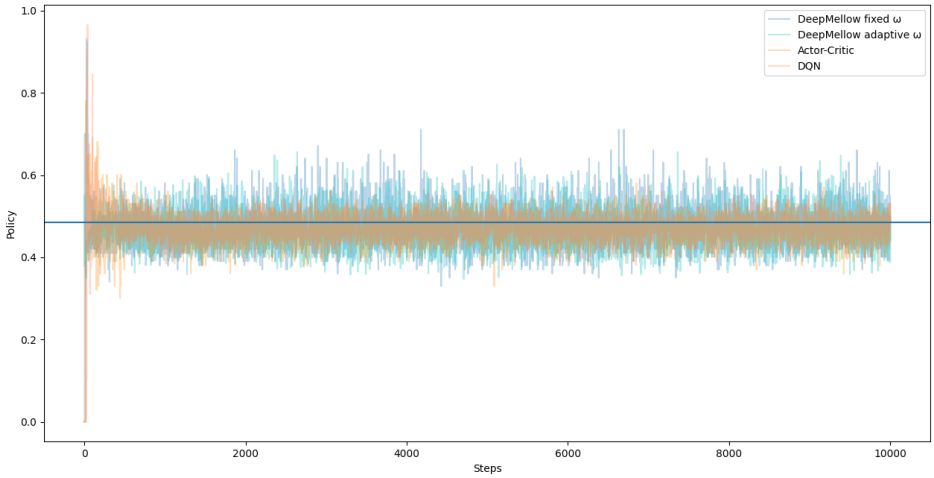
We will implement DeepMellow with ω fixed at -500 and adaptive ω starting from -500. Since we are trying to find a policy minimizing the cost function, it is reasonable to start at a negative ω. An additional $M = 128$ experiences will be sampled in the DeepMellow algorithm with adaptive ω to adjust ω. The learning rate for the adaptive ω is fixed at .0001.

The neural network approximating the Q-function consists of two hidden layers with ten neurons and a tanh activation function. It takes the state and mean-field as input and outputs an approximation of all action values associated with the input state. The activation function for the output layer is linear, as the neural networks are trying to approximate specific Q-values of each state-action pair without bounds. If an additional actor network is used, the same neural network structure will be applied except for the activation function for the output layer, which is replaced by a softmax function to gives a valid probability vector of actions.

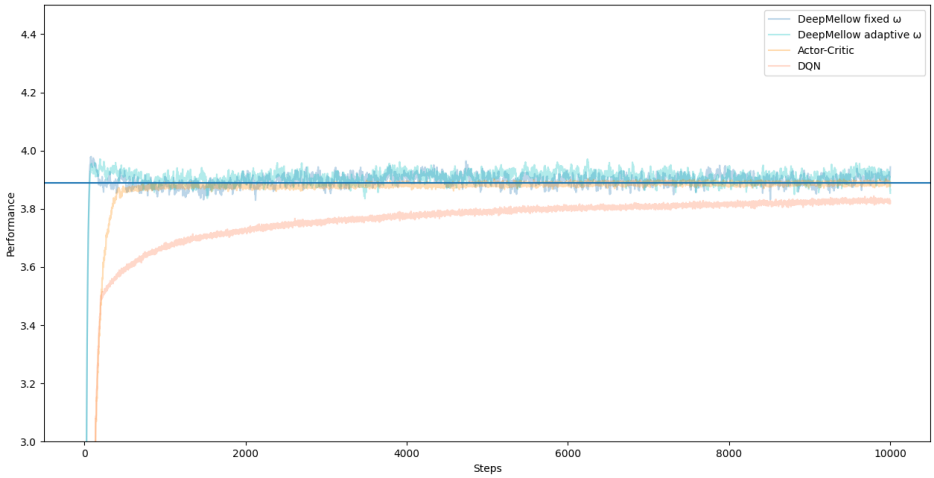
It is interesting that the DeepMellow algorithm with adaptive ω always suggests an decrease in ω. However, ω does not change too much over the whole learning process. If the adaptive method suggests an ω with extremely large magnitude, then the Mellowmax operator will act like a maximization/minimization operator and the need for the Mellowmax operator is questioned.

Figure 7 graphically summarizes these four algorithms for further analysis. The estimated policies are close to but oscillate around the theoretical equilibrium. Compared with those of two DeepMellow algorithms, the magnitudes of variation in mean-field actor-critic and mean-field deep Q-learning are smaller across steps. While mean-field deep Q-learning performs worst in terms of estimating equilibrium performance, mean-field actor-critic has the most stable and accurate estimate for performance. The estimated performances of two DeepMellow algorithms are less stable and marginally higher than the true value. Two DeepMellow algorithms in general both have a more accurate estimation of equilibrium mean-field. The value function is estimated by the neural network at $t = 10000$.

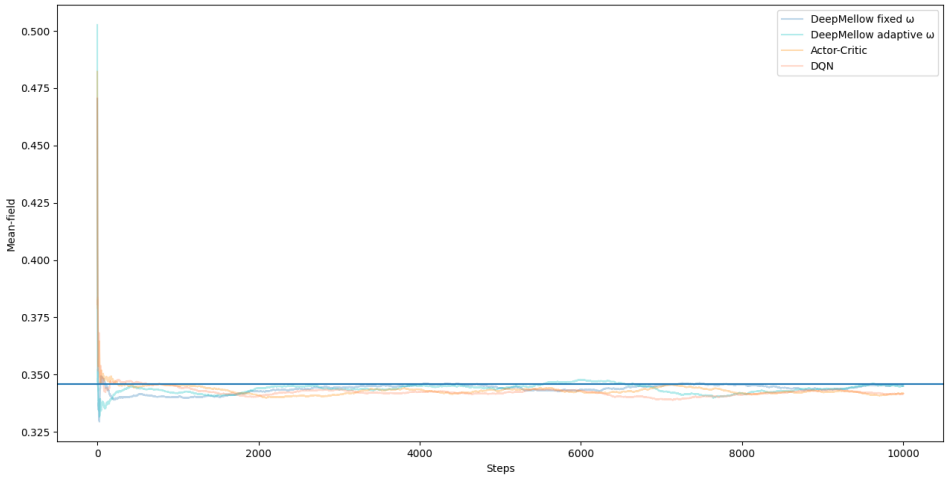
An improvement of neural network methods is that the optimal policy is of a threshold structure, so we do not have to estimate the true threshold using segment threshold and intersect threshold. Nevertheless, the estimated threshold in the approximated equilibrium policy using these deep mean-field Q-learning algorithms generally oscillates more violently as compared with that using tabular methods. As can be seen from the plot, the estimated policy threshold keeps wavering between 0.4 and 0.6.



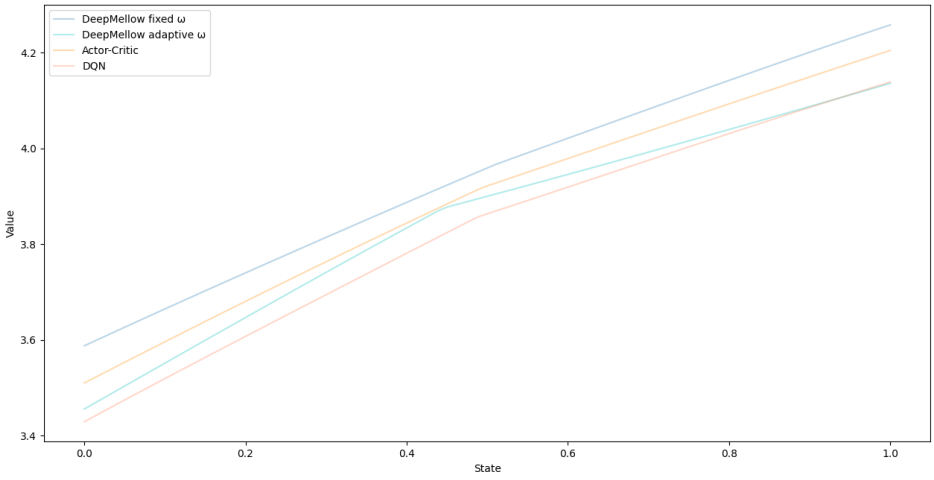
(a) Policy comparison using neural network methods



(b) Policy comparison using neural network methods



(c) Mean-field comparison using neural network methods



(d) Comparison of value functions comparison using neural network methods

Figure 7: Convergence using neural network methods

6 Mean-Field Games with Multiple Populations

Section 1.4 has introduced us to the major-minor model as an extension of the mean-field games model. Although the major-minor model assumes finiteness of the state spaces for both the major and minor players, due to the relatively large size of the state space of the minor players, we will find the optimal policy using an actor-critic based method.

Because the major player does not share the same cost function and transition rule with the minor players, it is inappropriate to assume they share the same Q-network. In order to manifest the distinction between two population of players, we will construct neural networks for each population and store their the transition experiences in two different replay buffers $\mathcal{D}_{major}$ and $\mathcal{D}_{minor}$. Let the critic networks be denoted by Q_{major} and Q_{minor} and the actor networks by π_{major} and π_{minor}.

Moreover, the environment dynamics must not be the same as those specified in Section 1.3, as the major player should also be included. For illustrative purposes, we assume the major player and minor players share the same state space $S = [0, 1]$ and binary action space $A = \{0, 1\}$. We also define the cost function and transition law for the minor players to be:

$$
\begin{aligned}
c(x_t^i, x_t^0, a_t^i, \theta(t)) &= (.2 + \theta(t))x_t^i + .5a_t^i + \kappa x_t^0, \\
P(x_{t+1}^i | x_t^i, x_t^0, a_t^i = a_0, \theta(t)) &= \begin{cases} 0 & x_{t+1}^i < x_t^i \\ \frac{1}{(1-x_t^i)} & x_{t+1}^i \geq x_t^i \end{cases}, \\
P(x_{t+1}^i | x_t^i, x_t^0, a_t^i = a_1, \theta(t)) &= \begin{cases} 1 & x_{t+1}^i = 0 \\ 0 & x_{t+1}^i > 0 \end{cases},
\end{aligned}
\tag{45}
$$

where κ controls how significant the major player's impact on the minor players it is. If κ is close to 0, we expect the optimal policy for the minor players to be similar to that obtained in Section 5. The cost function and transition law for the major player is defined as:

$$
\begin{aligned}
c(x_t^0, a_t^0, \theta(t)) &= (.2 + \theta(t))(x_t^0)^2 + .5(a_t^0)^2, \\
P(x_{t+1}^0 | x_t^0, a_t^0 = a_0, \theta(t)) &= \begin{cases} 0 & x_{t+1}^0 < x_t^0 \\ \frac{1}{(1-x_t^0)} & x_{t+1}^i \geq x_t^i \end{cases}, \\
P(x_{t+1}^0 | x_t^0, a_t^0 = a_1, \theta(t)) &= \begin{cases} \frac{1}{x_t^0} & x_{t+1}^0 < x_t^0 \\ 0 & x_{t+1}^0 \geq x_t^0 \end{cases}.
\end{aligned} \tag{46}
$$

Following the model formulation in Section 1.4, we denote the states of the major and minor players in the $N + 1$ major-minor model at time t by x_t^0 and $\mathbf{X}_t = x_t^1, \ldots, x_i^N$, where $N = 100$. The players interact with the environment, choose actions a_t^0 and $\mathbf{A}_t = a_t^1, a_t^2, \ldots, a_t^N$ following $\pi_{major}(x_t^0, \theta(t))$ or $\pi_{minor}(x_t^i, x_t^0, \theta(t))$, receive costs c_t^0 and $\mathbf{C}_{t+1} = c_{t+1}^1, c_{t+1}^2, \ldots, c_{t+1}^N$ from the environment, and proceed to the next state at $t+1$, x_{t+1}^0 and $\mathbf{X}_{t+1} = x_{t+1}^1, x_{t+1}^2, \ldots, x_{t+1}^N$. The transition quadruples $(x_t^0, a_t^0, c_{t+1}^0, x_{t+1}^0)$ and $(\mathbf{X}_t, \mathbf{A}_t, \mathbf{C}_{t+1}, \mathbf{X}_{t+1})$ will be stored in $\mathcal{D}_{major}$ and $\mathcal{D}_{minor}$, respectively. Since the amount of data generated by the major and minor players differs greatly, we define the batch size to be $K_1 = 8$ for Q_{major} and π_{major}, and $K_2 = 128$ for Q_{minor} and π_{minor}. The separate target network will be fixed for 10 steps and then updated using equation (32) with $\tau = 1$ for the first 20 updates (or episodes) and $\tau = (1 + |\#(steps)|)^{-.7}$ to speed up the learning in the early stage.

The implementations of neural networks for the major player and minor players are similar except for the input dimension. The actor networks and critic networks are con-

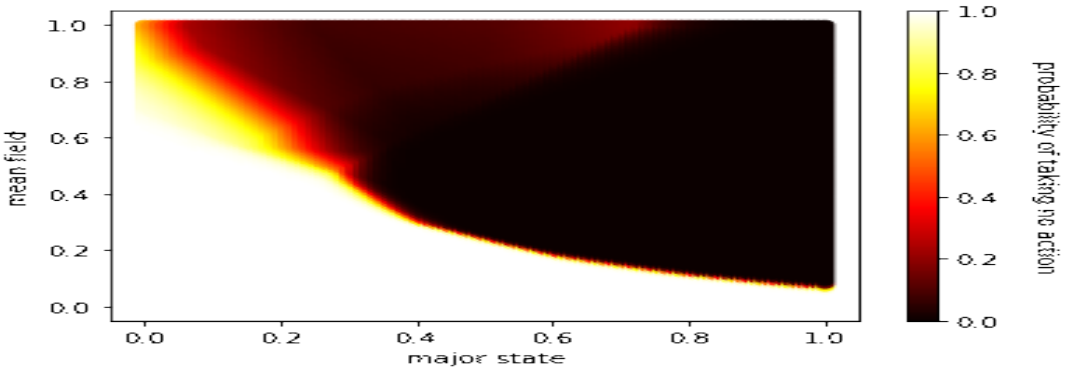
structed with two hidden layers with ten neurons and tanh activation. The output layer of the critic network has a linear activation, while that of the actor network has a softmax function as activation. In the major neural network, the input shape is given as (2,), while that in the minor neural network is given as (3,).

In order to investigate the major player's influence on the minor players, we will compute the optimal policy for $\kappa = 0.1$ and $\kappa = 1$ and compare the optimal policy for the major player and minor in heat maps below. When $\kappa = 0.01$, if the mean-field can be assumed to be around 0.345 as the case when there is no major player, then the policy structure is similar to that in the binary action spaces model.

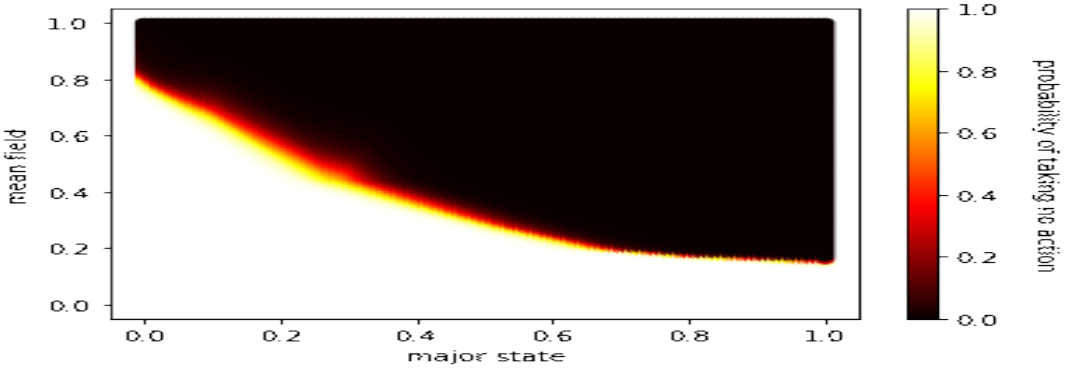
7 Conclusion

This thesis reviews the binary action spaces model, summarizes two tabular Q-learning algorithms and two neural network based Q-learning algorithms with application in the mean-field games structure, and visually present the convergence results of these algorithms. In practice, tabular Q-learning algorithms perform poorly in that they cannot find an equilibrium of a threshold structure. This thesis offers two methods estimating the threshold based on the Q-table. On the contrary, deep Q-learning algorithms keep oscillating around the true threshold.

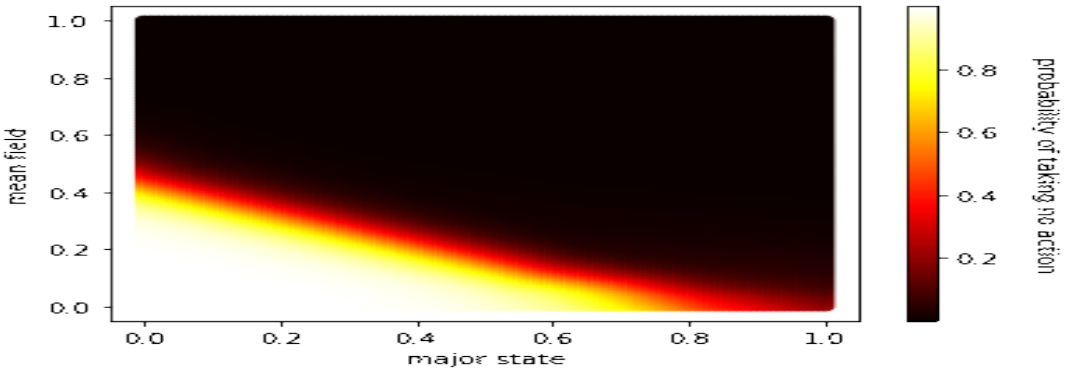
An effort to generalize mean-field reinforcement learning algorithms applicable in the binary action spaces model to the major-minor model has been made, but we have still not yet fully explored the nature of the major-minor model. More works on the major-minor model, especially on approximating equilibrium using reinforcement learning algorithms,



(a) Major player $\kappa = .1$

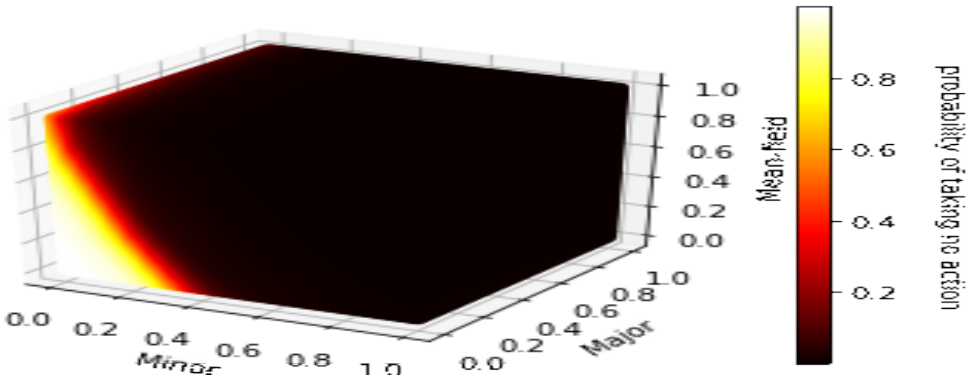


(b) Major player $\kappa = 1$

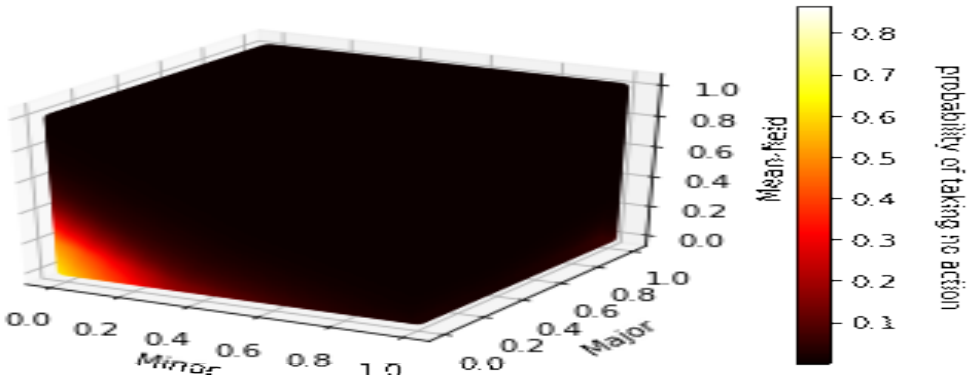


(c) Major player $\kappa = .01$

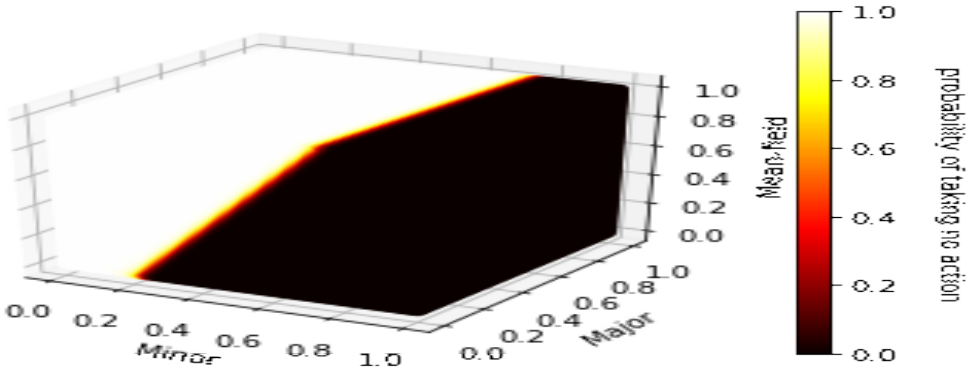
are of our interests in the future.



(d) Minor player $\kappa = .1$



(e) Minor player $\kappa = 1$



(f) Minor player $\kappa = 0.01$

Figure 8: Major-Minor model with different κ

References

[1] Mridul Agarwal et al. “Reinforcement Learning for Mean Field Game”. In: *arXiv preprint arXiv:1905.13357* (2019).

[2] Andrea Angiuli, Jean-Pierre Fouque, and Mathieu Laurière. “Unified reinforcement Q-learning for mean field game and control problems”. In: *arXiv preprint arXiv:2006.13912* (2020).

[3] Kavosh Asadi and Michael L Littman. “An alternative softmax operator for reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 243–252.

[4] Alain Bensoussan, Tao Huang, and Mathieu Laurière. “Mean field control and mean field game models with several populations”. In: *arXiv preprint arXiv:1810.00783* (2018).

[5] Lucian Busoniu, Robert Babuska, and Bart De Schutter. “A comprehensive survey of multiagent reinforcement learning”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172.

[6] Ethan Duryea, Michael Ganger, and Wei Hu. “Exploring deep reinforcement learning with multi q-learning”. In: *Intelligent Control and Automation* 7.4 (2016), pp. 129–144.

[7] Eyal Even-Dar, Yishay Mansour, and Peter Bartlett. “Learning Rates for Q-learning.” In: *Journal of machine learning Research* 5.1 (2003).

[8] Masaaki Fujii. *Probabilistic Approach to Mean Field Games and Mean Field Type Control Problems with Multiple Populations*. CARF F-Series CARF-F-467. Center for Advanced Research in Finance, Faculty of Economics, The University of Tokyo, Nov. 2019. URL: https://ideas.repec.org/p/cfi/fseres/cf467.html.

[9] Xin Guo et al. "A general framework for learning mean-field games". In: *arXiv preprint arXiv:2003.06069* (2020).

[10] Minyi Huang. "Mean Field Stochastic Games with Discrete States and Mixed Players". In: *Game Theory for Networks*. Ed. by Vikram Krishnamurthy et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 138–151. ISBN: 978-3-642-35582-0.

[11] Minyi Huang, Peter E Caines, and Roland P Malhamé. "Individual and mass behaviour in large population stochastic wireless power control problems: centralized and Nash equilibrium solutions". In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*. Vol. 1. IEEE. 2003, pp. 98–103.

[12] Minyi Huang and Yan Ma. "Binary Mean Field Stochastic Games: Stationary Equilibria and Comparative Statics". In: *Modeling, Stochastic Control, Optimization, and Applications*. Springer, 2019, pp. 283–313.

[13] Minyi Huang and Yan Ma. "Mean field stochastic games with binary action spaces and monotone costs". In: *arXiv preprint arXiv:1701.06661* (2017).

[14] Minyi Huang, Roland P. Malhamé, and Peter E. Caines. "Large population stochastic dynamic games: closed-loop McKean-Vlasov systems and the Nash certainty

equivalence principle". In: *Communications in Information Systems* 6.3 (2006), pp. 221–252. DOI: cis/1183728987. URL: https://doi.org/.

[15] Minyi Huang and Mengjie Zhou. "Linear quadratic mean field games: Asymptotic solvability and relation to the fixed point approach". In: *IEEE Transactions on Automatic Control* 65.4 (2019), pp. 1397–1412.

[16] Seungchan Kim and George Konidaris. "Adaptive Temperature Tuning for Mellowmax in Deep Reinforcement Learning". In: *the NeurIPS 2019 Workshop on Deep Reinforcement Learning*. 2019.

[17] Seungchan Kim et al. "DeepMellow: Removing the Need for a Target Network in Deep Q-Learning". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 2733–2739. DOI: 10.24963/ijcai.2019/379. URL: https://doi.org/10.24963/ijcai.2019/379.

[18] Jean-Michel Lasry and Pierre-Louis Lions. "Jeux à champ moyen. I – Le cas stationnaire". In: *Comptes Rendus Mathematique* 343.9 (2006), pp. 619–625. ISSN: 1631-073X. DOI: https://doi.org/10.1016/j.crma.2006.09.019. URL: https://www.sciencedirect.com/science/article/pii/S1631073X06003682.

[19] Jean-Michel Lasry and Pierre-Louis Lions. "Mean field games". In: *Japanese journal of mathematics* 2.1 (2007), pp. 229–260.

[20] Michael L Littman and Csaba Szepesvári. "A generalized reinforcement-learning model: Convergence and applications". In: *ICML*. Vol. 96. Citeseer. 1996, pp. 310–318.

[21] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[22] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[23] Ling Pan et al. “Reinforcement learning with dynamic boltzmann softmax updates”. In: *arXiv preprint arXiv:1903.05926* (2019).

[24] Tom Schaul et al. “Prioritized experience replay”. In: *arXiv preprint arXiv:1511.05952* (2015).

[25] James C Spall. “An overview of the simultaneous perturbation method for efficient optimization”. In: *Johns Hopkins apl technical digest* 19.4 (1998), pp. 482–492.

[26] James C Spall. “Implementation of the simultaneous perturbation algorithm for stochastic optimization”. In: *IEEE Transactions on aerospace and electronic systems* 34.3 (1998), pp. 817–823.

[27] Jayakumar Subramanian and Aditya Mahajan. “Reinforcement learning in stationary mean-field games”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019, pp. 251–259.

[28] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[29] Yaodong Yang and Jun Wang. “An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective”. In: *arXiv e-prints* (2020), arXiv–2011.

[30] Yaodong Yang et al. “Mean Field Multi-Agent Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 5571–5580. URL: http://proceedings.mlr.press/v80/yang18d.html.