ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Koubele** Jméno: **Jakub** Osobní číslo: **465878**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Limitations of Reinforcement Learning Algorithms in Imperfect Information Games

Název diplomové práce anglicky:

Limitations of Reinforcement Learning Algorithms in Imperfect Information Games

Pokyny pro vypracování:

Reinforcement learning (RL) has been recently used on games with perfect information,
where it leads to state-of-the-art performance [5]. However, the games with imperfect information are
more complicated than games with complete information, and properties of reinforcement learning in
such setting is still not understood well. The student will:
1) review the existing literature on reinforcement learning in games;
2) implement and evaluate independent learning of at least two RL algorithms in normal form
games;
3) study theoretical properties of convergence process in 2), such as removal of dominated
strategies;
4) based on the limitations identified in normal form games, design sequential scenarios difficult
for independent RL methods;
5) evaluate the empirical performance of a deep reinforcement learning algorithm in scenarios
from 4).

Seznam doporučené literatury:

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press,
2018.
[2] Lattimore, Tor, and Csaba Szepesvári. &quot;Bandit algorithms.&quot; preprint (2018).
[3] Srinivasan, Sriram, et al. &quot;Actor-critic policy optimization in partially observable multiagent
environments.&quot; Advances in Neural Information Processing Systems. 2018.
[4] Zinkevich, Martin, et al. &quot;Regret minimization in games with incomplete information.&quot; Advances in
neural information processing systems. 2008.
[5] Silver, David, et al. &quot;A general reinforcement learning algorithm that masters chess, shogi, and Go
through self-play.&quot; Science362.6419 (2018): 1140-1144.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Mgr. Viliam Lisý, MSc., Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **28.01.2019** Termín odevzdání diplomové práce: **07.01.2020**

Platnost zadání diplomové práce: **20.09.2020**

Mgr. Viliam Lisý, MSc., Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Limitations of Reinforcement Learning Algorithms in Imperfect Information Games

Bc. Jakub Koubele

Supervisor: Mgr. Viliam Lisý, MSc., Ph.D.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

January 6, 2020

Abstract

This thesis examines the properties of reinforcement learning algorithms used in games with imperfect information. The imperfect information games are more complex than other settings in which the reinforcement learning algorithms are commonly used, and the convergence properties of reinforcement learning algorithms in imperfect information games are not understood well.

We examined the theoretical properties of two single-state reinforcement learning algorithms in a self-play of matrix games and evaluated their practical performance. We also evaluated the performance of two reinforcement learning algorithms in an extensive-form game with imperfect information.

Abstrakt

Tato práce zkoumá vlastnosti algoritmů posilovaného učení aplikovaných ve hrách s neúplnou informací. Hry s neúplnou informací jsou složitější než ostatní prostředí v nichž jsou algoritmy posilovaného učení běžně využívány, a vlastnosti konvergence těchto algoritmů ve hrách s neúplnou informací nejsou známé.

V práci jsme analyzovali teoretické vlastnosti dvou algoritmů posilovaného učení aplikovaných na maticové hry, a empiricky jsme vyhodnotili jejich využitelnost. Také jsme vyhodnotili empirické vlastnosti dvou algoritmů posilovaného učení aplikované na hru s neúplnou informací v extenzivní formě.

Declaration

Contents

List of Figures

List of Tables

List of Algorithms

Notation

Vectors in R^n are denoted in **bold** text. The elements of the vector are denoted using lower index, for example $\mathbf{x}_i$ denotes the i-th element of vector $\mathbf{x}$. Declaration of the vector is done by using square bracket, for example as $\mathbf{x} = [1, 2, 3]$.

Upper index is usually used to denote the time index, for example s^t denotes scalar s at time t.

Symbols $\mathbf{0}$ and $\mathbf{1}$ are used to denote vectors with constant values of 0 and 1, respectively.

1 Introduction

Reinforcement learning (RL) [15] is an area studying agents acting in an environment while maximizing some cumulative numerical reward. RL algorithms were shown to be able to learn such complex tasks as playing Atari games [8] or successfully perform maneuvers with real-world helicopter [5].

RL algorithms can also learn to play games via self-play, which consists of multiple RL agents playing against each other. This approach was used to create agents with super-human performance in games such as Go or chess[13].

Besides games with perfect information (such as mentioned Go and chess), the RL algorithms may be also applied to games with imperfect information, in which certain observations are not available to all players. Examples of games with imperfect information are card games such as poker, or computer games such as Starcraft or Dota 2. Recently, the self-play of RL agents was used to create a strong agent for the Dota 2 game [10].

However, the games with imperfect information are more complex than the games with perfect information, and the convergence properties of RL algorithm in that setting are still not understood well. For example, the games with imperfect information usually require players to develop a probabilistic policy in order to achieve optimality, which is not necessary in games with perfect information.

The aim of this thesis is to examine the properties of RL algorithms used to learn imperfect information games via self-play.

This thesis is organized as follows: Chapter 2 provides technical background on the areas of reinforcement learning, game theory, and several associated problems. Chapter 3 study the self-play properties of the Action-Value Method, a simple RL algorithm for the single-state setting. Chapter 4 examines the Gradient Bandit Algorithm, a single-state version of Policy Gradient, which is a commonly used RL algorithm. In chapter 5, the performance of two RL algorithms (Policy Gradient and Q-learning) is evaluated on Leduc Holdem, a simplified version of poker. Chapter 6 contains a conclusion of the thesis. The content of the supplementary code is described in the Appendix.

2 Background

In this chapter, we will provide an overview of reinforcement learning, regret minimization, and game theory.

We will begin by the note on the importance weighted estimator, a statistical method that is commonly used in setting where some information is not observable by the agent.

We will then introduce a stochastic bandit problem, which consists of one agent sequentially choosing from a set of actions while staying in a single state. The stochastic bandit problem is a precursor to the full reinforcement learning setting, which adds multiple states and transitions between them. In a bandit setting, the agent faces the problem of imperfect information, as he only observes the reward for the action he chooses and not the possible rewards for other actions.

Next, we will describe a problem of regret minimization in full information setting, sometimes called a prediction with expert advises. Compared to a stochastic bandit problem, the agent observes rewards for all possible actions, but we allow these rewards to be chosen adversarially to the agent. Direct reward maximization is not possible in such a setting, as the adversary can set all rewards to a minimal value. Instead of maximizing reward, the agent tries to minimize regret, which is a performance metrics suitable for the adversarial setting. Regret minimization is strongly tied to the problem of learning to play a game, as the bound on regret translates to the bound on the quality of solution in some types of games.

We will then describe adversarial bandits, which add an element of imperfect information to the problem of regret minimization.

Then, two-players, zero-sum normal form games will be introduced. These consist of two agents choosing from their set of actions in a single state; the reward of the players depends on the actions chosen by both of them.

Finally, we will extend the single state settings to multiple state ones: full reinforcement learning problem will be described, as well as extensive-form games with imperfect information.

2.1 Importance Weighted Estimator

Often, we will face the following problem: there is an unknown vector $\mathbf{x} \in R^n$, and known probability distribution $\mathbf{p} > \mathbf{0}$ over the elements of vector $\mathbf{x}$. An element $\mathbf{x}_i$ of $\mathbf{x}$ is sampled by $\mathbf{p}$ and observed (the index i of the sampled element is observed as well). Given that observation, we wish to estimate the whole vector $\mathbf{x}$.

There exist an unbiased estimator [7]

$$\hat{\mathbf{x}}_j = b + \frac{(\mathbf{x}_i - b)I(j = i)}{\mathbf{p}_i} \tag{1}$$

where $b \in R$ is a parameter called baseline, and $I(j = i)$ is an indicator function stating whether element $j = i$ was sampled. Such estimator is called an *importance weighted estimator*, since the difference between observed value and baseline is weighted by the inverse of the probability that the element would be observed.

The variance of importance weighted estimator is

$$V[\hat{\mathbf{x}}_i] = (\mathbf{x}_i - b)^2 \frac{1 - \mathbf{p}_i}{\mathbf{p}_i} \tag{2}$$

We may observe two properties of importance weighted estimator: at first, the variance of $\hat{\mathbf{x}}_i$ is decreasing with the probability $\mathbf{p}_i$ that the element $\mathbf{x}_i$ would be sampled. The estimate is then less precise for elements of $\mathbf{x}$ with small probability of being sampled. For $\mathbf{p}_i$ going to 0, the variance of $\hat{\mathbf{x}}_i$ goes to $+\infty$.

At second, the variance depends on the distance between baseline b and $\mathbf{x}_i$. As the b can be set to any constant, we would like to set it near the expected value of $\mathbf{x}_i$ to reduce the variance. For example, if we know that $\mathbf{x}$ is bounded between 0 and 1, we should also choose b to be some number in $[0, 1]$.

2.2 Stochastic Bandit Problem

Stochastic bandit problem [7] is a simple model of sequential decision making. A formal definition follows:

Definition 2.1. A stochastic bandit problem is a tuple (A, P), where

- A $= \{a_1, ..., a_k\}$, $k \in N$ is a set of actions available to the agent
- $P = \{P_1, ..., P_k\}$ is a set of probability distributions over real numbers with finite variances.

The bandit problem consists of T rounds, where $T \in \{N \cup +\infty\}$ is a time horizon, known to the agent. In each round $t \in \{1, ..., T\}$, the agent chooses an action $a_i \in A$. Then, reward x_i is sampled from the distribution P_i and agent observe and obtain this reward. The agent's task is to maximize the average reward obtained over all T rounds.

Since the agent each round observes only the reward from probability distribution corresponding to the action he took, he faces so-called *exploration versus exploitation dilemma*: Agent wants to exploit his knowledge of the environment by selecting actions that provide him with high rewards in past rounds. But simultaneously, the agent wants to learn which actions are the best one to play; to do so, he may sometimes play even actions that yielded low rewards in previous rounds.

If the probability distributions in P are know to be of a certain type, specialized algorithms for such problem may be constructed. For example, if the distributions are normal, highly efficient algorithms called *Upper Confidence Bound* [7] may be used, guarantying fast convergence of the agent's average reward to the maximal possible one.

However, in this thesis we will not work with any further assumptions about the distributions in P. The only information that we have is that the variance of these distributions if finite, as stated in the definition above.

We will now present two different algorithms for stochastic bandit problem. Both of them are simplified versions of algorithms that are commonly used in a full reinforcement learning problem: Action-Value Method is similar to a single state version of Q-learning, and Gradient Bandit Algorithm is a predecessor to Policy Gradient methods.

2.2.1 Action-Value Method with ϵ-greedy Exploration

The Action-Value Method [15] holds an estimate of the expected reward for playing each action. These estimates are simply empirical averages of rewards obtained from playing the corresponding action. The action-value method also has an exploration parameter $\epsilon \in (0, 1)$. When deciding which action to play, the Action-Value Method chooses to play the action with the highest estimated reward with probability $1 - \epsilon$, and to select the action uniformly at random otherwise (with probability ϵ). After playing the action, the reward is obtained and the estimate of the corresponding action is updated. As the estimate is a simple average of rewards, it can be efficiently updated online.

Exploration parameter ϵ ensures that each action has a probability at least $\frac{\epsilon}{k}$ to be played in each round. Therefore, with increasing timestep t, each action is played (in expectation) more times and the estimate of its average reward has a lower variance, meaning that the agent has a higher probability to correctly identify the optimal action.

However, exploring uniformly at random means that the agent has a probability $\frac{\epsilon}{k}$ to play an action which reward estimate is very low, even if the variance of that estimate is low as well. To avoid such behavior, we may choose ϵ to be a decreasing function of timestep t. Simultaneously, we would like to try each action an unbounded number of times (as time horizon T goes to $+\infty$), to avoid having a positive probability of failing to learn the optimal action. Formally, we want to have

$$\lim_{t \to \infty} \epsilon(t) = 0 \tag{3}$$

and

$$\sum_{t=1}^{\infty} \epsilon(t) = \infty \tag{4}$$

Such exploration scheme is called *greedy in the limit with infinite exploration* (GLIE)[14]. If the agent follows such scheme, he is guaranteed (with probability 1) to obtain the optimal payoff on average, as $T \to \infty$. This does not mean that the whole action-value method with ϵ-greedy exploration is the best possible way to deal with bandit problem, as we may choose to use more sophisticated algorithms to obtain faster convergence rate.

Algorithm 1: Action-Value Method

```
initialize action values vector $\mathbf{q} = \mathbf{0}$, action count vector $\mathbf{c} = \mathbf{0}$,
 exploration parameter $\epsilon \in (0, 1)$;
for $t$=1,...,$T$ do
    if $\mathbf{c}_i = 0$ for some $i$ then
        play action $i$
    end
    else
        With probability $\epsilon$, choose action uniformly at random.
         Otherwise, play action $i = \arg\max \mathbf{q}_i$
    end
    Observe and receive reward $x_i^t$ ;
    update $\mathbf{q}_i := \frac{\mathbf{q}_i \mathbf{c}_i + x_i^t}{\mathbf{c}_i + 1}$;
    update $\mathbf{c}_i := \mathbf{c}_i + 1$;
    optionaly update $\epsilon$ according to the GLIE schema
end
```

2.2.2 Gradient Bandit Algorithm

Instead of learning the value function and using it to select an action, we may try to learn the optimal policy directly. An example of this approach is a gradient bandit algorithm [15], which is a precursor to the Policy Gradient method frequently used in reinforcement learning.

Gradient bandit algorithm selects the action to play by sampling from its policy $\pi(\mathbf{w})$. That policy represents a probability distribution over the set of actions, and is parametrized by a parameter vector $\mathbf{w}$. Specifically, the gradient bandit algorithm uses softmax function to represent its policy, and the parameter vector $\mathbf{w}$ represents the parameters of softmax. The probability of playing an action a according to the softmax is defined as follows:

$$\pi_a(\mathbf{w}) = \frac{\exp(\mathbf{w}_a)}{\sum_{i \in A} \exp(\mathbf{w}_i)} \tag{5}$$

The learning process is done by updating the weights $\mathbf{w}$, such that they will put a high probability on playing actions with high rewards and low probability on actions with low rewards. To do so, we would like to compute the gradient of the expected reward and update $\mathbf{w}$ using this gradient. That is, after receiving reward x^t at round t, we would like to compute the expected reward

$$E\left[x^t | \pi(\mathbf{w})\right] = \sum_{a \in A} \pi_a(\mathbf{w}) x_a^t \tag{6}$$

and then perform a gradient ascent on the expected reward: compute the gradient

$$\nabla^t = \frac{\partial E\left[x^t | \pi(\mathbf{w})\right]}{\partial \mathbf{w}} \tag{7}$$

and update

$$\mathbf{w} := \mathbf{w} + \eta^t \nabla^t \tag{8}$$

where η is a learning rate.

The problem is that the calculation above require us to know the rewards for all actions, but we observed only a reward for the action that we played at round t. However, we may use an importance weighted estimator to get an unbiased estimate of the whole reward vector, and use this estimate instead of ground truth rewards. Thus, the gradient ascent will became a stochastic gradient ascent.

The importance-weighted estimator works with a baseline parameter b. The Bandit Gradient Algorithm chooses b as an average of rewards obtained in previous round, that is,

$$b^t = \frac{1}{t} \sum_{t'=1}^{t} r^{t'} \tag{9}$$

Algorithm 2: Gradient Bandit Algorithm

```
Initialize weight vector $\mathbf{w} = \mathbf{0}$, learning rate $\eta > 0$, baseline $b = 0$;
for $t=1,\ldots,T$ do
    select policy $\pi(\mathbf{w}) = softmax(\mathbf{w})$;
    sample action $i$ according to $\pi(\mathbf{w})$;
    observe and receive reward $x_i^t$;
    estimate $\mathbf{x}^t$ by an importance weighted estimator with parameter $b$;
    use the estimate of $\mathbf{x}^t$ to compute $\nabla = \frac{\partial E\left[x_i^t | \pi(\mathbf{w})\right]}{\partial \mathbf{w}}$;
    update $\mathbf{w} := \mathbf{w} + \eta \nabla$;
    update $b := \frac{bt + x_i^t}{t+1}$;
end
```

2.3 Regret Minimization in Adversarial Setting

The stochastic bandit setting assumes that the rewards are generated from probability distributions that do not change over time. However, this assumption may be violated in some scenarios. The most extreme case would be that the rewards are completely controlled by an adversary, who can choose them arbitrarily. While this is usually not a realistic setup, it may still be useful to model this worst-case scenario: any guarantee that we can prove for the agent in that setup will necessarily hold also for all other environments, covering a wide range of situations where stochastic bandit assumptions do not hold.

For now, we will assume that the agent observes the rewards for all possible actions (such setting is also known as a *prediction with expert advice*). We will revert to the imperfect information setting in the next section, where adversarial bandits will be introduced.

The formal definition of full information regret minimization problem [1] is as follows:

Definition 2.2. The full information regret minimization problem is a set of reward vectors $\{\mathbf{x}^1, ..., \mathbf{x}^T\}$, with $\mathbf{x}^t \in [0,1]^k$, where $k \in N$ is a number of actions available to the agent.
The problem consists of T rounds. In each round t, the agent selects a policy $\pi^t \in \Pi$, where Π is a probability simplex over actions. Then, the reward vector $\mathbf{x}^t$ is revealed, and the agent receives a reward $(\pi^t)^\intercal \mathbf{x}^t$ (i.e. a dot product between the policy and reward vector).

As the rewards may be chosen arbitrarily and the agent does not have any information about them before he selects the policy for corresponding round, it may no sense to try to maximize the cumulative reward as in the stochastic bandit setting - here, all rewards may be 0, and the agent would receive 0 cumulative reward no matter of the policy he chooses.

Instead of that, the agent tries to optimize different metric called *external regret*. Its formal definition follows:

Definition 2.3. The external regret $R_i^t \in R$ with respect to the action i at time t is

$$R_i^t = \sum_{t'=1}^{t} \mathbf{x}_i^{t'} - \sum_{t'=1}^{t} (\pi^{t'})^\intercal \mathbf{x}^{t'}$$

The external regret $R^t \in R$ of the agent at time t is

$$R^t = \max_{i \in [k]} R_i^t$$

It will also be useful to define the immediate regret vector $\mathbf{r}^t$ at time t as

$$\mathbf{r}^t = \mathbf{x}^t - \mathbf{1}(\pi^{t'})^\intercal \mathbf{x}^{t'}$$

and cumulative regret vector at time t:

$$\mathbf{s}^t = \sum_{t'=1}^{t} \mathbf{r}^{t'} = \left[R_1^t, ..., R_k^t\right]$$

The external regret compares the performance of the agent with a policy that would select the best single action (in hindsight) and play it for all rounds. The goal of the agent in regret minimization problem is to minimize his external regret. As the rewards vector may be arbitrary, we will be interested in algorithms that guarantee bound on the external regret, no matter what the actual reward vectors are.

2.3.1 Lagrangian Hedging

Lagrangian Hedging [4] is a class of algorithms for regret minimization problem. They work by maintaining the cumulative regret vector $\mathbf{s}$, and choosing the policy at each round as some function $\pi(\mathbf{s}) \rightarrow \Pi$, where Π is a set of all probability distributions over the set of actions.

Specifically, each Lagrangian Hedging algorithm define a convex potential function $F(\mathbf{s}) \rightarrow R$. The function $\pi(\mathbf{s}) \rightarrow \Pi$ is then given as a gradient of $F(\mathbf{s})$, normalized to obtain a probability distribution. Loosely speaking, the Lagrangian Hedging performs a gradient descent on $F(\mathbf{s})$. The main idea is that for a suitable choice of potential function, the gradient descent on $F(\mathbf{s})$ may be linked to the minimization of agent's external regret.

One popular choice of potential function is

$$F_1(\mathbf{s}) = \log\left(\sum_{i=1}^{k} \exp(\eta \mathbf{s}_i)\right) - \log(k) \tag{10}$$

where $\eta \in R, \eta > 0$ is a learning rate. This choice of potential function leads to a policy given as

$$\pi(\mathbf{s}) = softmax(\eta \mathbf{s}) \tag{11}$$

Algorithm with that policy is called *Hedge*[1], and has a wide range of applications e.g. in boosting of weak classifiers or in an information retrieval problem.

Another instance of Lagrangian Hedging uses a potential function

$$F_2(\mathbf{s}) = \frac{1}{2}\sum_{i=1}^{k} [\mathbf{s}_i]_+^2 \tag{12}$$

where $[\mathbf{s}_i]_+$ is a positive part of $\mathbf{s}_i$. This potential function leads to a policy which plays action i with a probability of

$$\pi_i(\mathbf{s}) = \frac{[s_i]_+}{\sum_{j=1}^{k} [s_j]_+} \tag{13}$$

For $F_2(\mathbf{s}) = 0$, the equation 13 is not defined, and the policy π is then chosen as an uniform distribution over the actions. Algorithm with that policy is called *Regret Matching.*

The external regret of Hedge is asymptotically bounded[2] as $\mathcal{O}(\sqrt{T\log(k)}$, and the Regret Matching achieves a bound of $\mathcal{O}(\sqrt{Tk}$. Since both of these bounds are sub-linear in the time horizon T, the average external regret of both algorithms is guaranteed to go to 0 in limit as $T \rightarrow \infty$. Algorithms with such property are called *no-regret* algorithms.

[1]The original formulation of Hedge algorithm [2] uses cumulative rewards instead of regrets. However, since softmax function is invariant to a shift by a constant and the regret vector is just a reward vector shifted by the utility obtained by an agent, these two definitions are equivalent. Please see a section 4.1 for a further discussion.

[2]Achieving this bound require the learning rate η of Hedge algorithm to be selected accordingly to the time horizon T. For constant η, the regret bound would become linear.

Algorithm 3: Lagrangian Hedging

```
Initialize vector of cumulative regrets $\mathbf{s} = \mathbf{0}$;
for $t$=1,...,$T$ do
    select policy $\pi(\mathbf{s}) \propto \nabla F(\mathbf{s})$. If $\mathbf{s} \leq \mathbf{0}$, $\pi(\mathbf{s})$ can be selected
      arbitrarily.;
    observe reward vector $\mathbf{x}$ and receive a reward $\mathbf{x}^T\pi(\mathbf{w})$;
    compute regret vector $\mathbf{r} = \mathbf{x} - \mathbf{1}\mathbf{x}^\intercal\pi(\mathbf{w})$;
    update $\mathbf{s} := \mathbf{s} + \mathbf{r}$
end
```

2.4 Adversarial Bandits

We will now return to the bandit setting, where the agent observes only the reward for the action he selects, but the rewards may be chosen adversely. The definition of adversarial bandit [7] is as follows:

Definition 2.4. The adversarial bandit is a set of reward vectors $\{\mathbf{x}^1, ..., \mathbf{x}^T\}$, with $\mathbf{x}^t \in [0,1]^k$, where $k \in N$ is a number of actions available to the agent.

The problem consists of T rounds. In each round t, the agent selects a policy $\pi^t \in \Pi$, where Π is a set of probability distributions over actions. Then, the agent samples action i according to the π^t, and observe and receive the reward x_i^t. The goal of the agent is to minimize his external regret.

Compared to the regret minimization in full information setting, the problem is now complicated by the aspect of incomplete information. As in stochastic bandit setup, one possible solution is to use an importance weighted estimator to estimate the whole reward vector and then use some algorithm suitable for regret minimization in full information setting with an estimated reward vector as an input.

This idea is leveraged by the *Exponential-Weight Algorithm for Exploration and Exploitation* [7], also know as *Exp3* algorithm. Exp3 works as a Hedge algorithm with a reward vector estimated by an importance weighted estimator with a parameter $b = 1$.

However, there are several caveats with the usage of importance weighted estimator in an adversarial bandit setting:

At first, for Exp3 it is more important to determine which actions are good and try to play them, rather than to focus on the avoidance of bad actions. Thus, the parameter b used in importance weighted estimator cannot be arbitrary but is selected to be 1 for that reason (the proof that provides a bound on regret of Exp3 would not work with different values of b).

At second, the importance weighted estimator requires the probability for all actions to be positive. This prohibits us to use for example Regret Matching with an importance weighted estimator, as Regret Matching can put zero probability on some actions.

At third, the importance weighted estimator has a high variance for actions with low probability. In an adversarial setting, the adversary may intentionally exploit this property against the agent. Specifically, it is possible to construct an adversarial bandit, in which the Exp3 suffers an external regret of $\frac{T}{4}$ with a probability at least $\frac{1}{65}$. While the expected external regret of Exp3 algorithm is asymptotically bounded as $\mathcal{O}(\sqrt{Tk \log(k)})$, the variance of the achieved regret is large. (The fact that the external regret of Exp3 can be linear in T with constant probability is caused by the fact that often the regret will be negative, thus the expected regret is sub-linear in T).

One way to overcome this problem is to use a modified version of importance weighted estimator given as

$$\hat{\mathbf{x}}'_j = b + \frac{(\mathbf{x}_i - b)I(i = j)}{\mathbf{p}_i + \gamma} \tag{14}$$

where $\gamma > 0$ is a parameter. That estimator is not unbiased but has a lower variance than the original importance weighted estimator. The Exp3 algorithm using estimator from equation 14 is called *Exp3-IX* [6] (the suffix *IX* stands for *Implicit Exploration*).

Another option is to mix Exp3 policy with a uniform distribution over the actions, to ensure that the probability of playing each action is high enough. The resulting algorithm is called *Exp3.P*.

Both of these techniques overcome the problem of constant probability of linear regret from which the Exp3 suffers. However, the parameters of these algorithms (the bias term γ in modified importance weighted estimator or the proportion of uniform distribution over actions) must be carefully selected, otherwise, the performance of the algorithm will be impaired.

2.5 Normal Form Games

This section will introduce normal-form games and related solution concepts. The definitions are mostly taken from (Shoham, 2008) [12].

Definition 2.5. A 2-players normal-form game is a tuple (N, A, u), where:

- $N = \{1, 2\}$ is a set of players,
- $A = A_1 \times A_2$, where A_1 (resp. A_2) is a finite set of actions available to player 1 (resp. player 2).
- $u = (u_1, u_2)$, where $u_i : A \to R$ is a payoff function for player i. If it holds that $u_1(a_1, a_2) = -u_2(a_1, a_2) \quad \forall a_1 \in A_1, a_2 \in A_2$, we call the game *zero-sum* game.

The game is played as follows: both players choose an action from their sets of actions. Then, these actions are revealed and players obtain a payoff according to the payoff function. Since players need to choose an action without knowledge of the action chosen by their opponent, normal form games involve the aspect of imperfect information.

Payoff function of such game can be conveniently represented by a matrix for each player, hence the normal-form games are also known as matrix games.

Even though the set of actions for each player is finite, players can decide for a probabilistic policy, choosing their actions randomly. Compared to a single-agent setting, this may be sometimes a necessary condition for a policy to be optimal.

Definition 2.6. A set of mixed strategies for player $i = 1, 2$ is a $\Sigma_i = \Pi(A_i)$, where $\Pi(X)$ is a set of all probability distributions over set X.

We may consider deterministic policies to be a special case of mixed strategies:

Definition 2.7. We say that the strategy $\sigma_i \in \Sigma_i$ is pure, if σ_i plays some action with probability 1.

As the utilities of players depend on actions chosen by both of them, we will need the following concept putting all their strategies together:

Definition 2.8. A strategy profile σ is an element of the Cartesian product of strategy sets of both players, $\Sigma_1 \times \Sigma_2$. We also define σ_{-i} to be a strategy profile without agent i's strategy.

As a strategy profile induces a certain probability distribution over outcomes of the game, we may extend the concept of utility from actions to strategies and define

$$u(\sigma_1, \sigma_2) = E_{a_1 \sim \sigma_1, a_2 \sim \sigma_2}[u(a_1, a_2)]$$

Even though the reward for playing some strategy depends on the strategy of the opponent, we may sometimes disregard certain strategies as 'bad', no matter what the opponent will do:

Definition 2.9. We say that strategy $\sigma_k \in \Sigma_i$ is weakly dominated by strategy $\sigma_l \in \Sigma_i$, if

$$u_i(\sigma_k, \sigma_{-i}) \leq u_i(\sigma_l, \sigma_{-i}) \quad \forall \sigma_{-i} \in \Sigma_{-i}$$

If the inequality is strict, we say that σ_k is strictly dominated. We may also speak about dominance of actions, if the strategies are pure.

Before we will define the concept of equilibrium, we will need the following concept of optimality from the point of view of one player:

Definition 2.10. Player i's ϵ-best response to the strategy profile σ_{-i} is a mixed strategy $\sigma_i^* \in \Sigma_i$ such that $u_i(\sigma_i^*, \sigma_{-i}) + \epsilon \geq u_i(\sigma_i, \sigma_{-i}) \quad \forall \sigma_i \in \Sigma_i$. If $\epsilon = 0$, we call σ_i^* just a best response.

We may now define the concept of equilibrium:

Definition 2.11. A strategy profile σ^* is called an ϵ-Nash Equilibrium if for each player i holds that

$$u_i(\sigma^*) + \epsilon \geq u_i(\sigma_i, \sigma_{-i}^*) \quad \forall \sigma_i \in \Sigma_i$$

where $\epsilon \in R$. If $\epsilon = 0$, we call σ^* just a Nash Equilibrium. In ϵ- Nash Equilibrium, each player i is playing ϵ best response against σ_{-i}^*.

Nash Equilibrium (NEQ) is the main solution concept of game theory. That is because rational players can be expected to play strategies according to NEQ - otherwise, at least one player would have an incentive to deviate from his current strategy.

In the case of two-players zero-sum games, NEQ has another appealing property: playing according to NEQ is equivalent to maximize payoff against worst-case opponent's strategy. Thus, by playing strategy from NEQ, player i is guaranteed to obtain payoff at least $u_i(\sigma^*)$. In zero-sum games, the payoff $u_1(\sigma^*)$ is called the value of the game. In some games, the value of the game can be computed without solving for the NEQ - for example, the value of any symmetric game is 0.

The value of ϵ can be used as a metric that measures the quality of NEQ approximation. Another commonly used metric is the so-called Nash distance (also known as Nash Convergence):

Definition 2.12. Let $s = (\sigma_1, \sigma_2)$ be a strategy profile in two-players game, br_1 be a player 1's best response against σ_2 and br_2 be a player 2's best response against σ_1. The Nash distance of s is then

$$\frac{u_1(br_1, \sigma_2) - u_1(s) + u_2(\sigma_1, br_2) - u_2(s)}{2} \tag{15}$$

For zero-sum games, we have $u_1(\sigma) = -u_2(\sigma)$, and the equation above simplifies to $\frac{u_1(br_1, \sigma_2) + u_2(\sigma_1, br_2)}{2}$.

We may also evaluate the quality of single strategy instead of the strategy profile. The suitable metric for this task is called exploitability:

Definition 2.13. The exploitability of strategy σ_i is

$$\min_{\sigma_{-i} \in \Sigma_{-i}} u_i(\sigma_i, \sigma_{-i}) - u(\sigma_i^*, \sigma_i^*) \tag{16}$$

Therefore, the exploitability measures the performance against the opponent's best response strategy, compared to the utility obtained in NEQ. The exploitability of strategies in NEQ is 0.

Solving a game means finding $\epsilon-$NEQ with low ϵ and / or low Nash distance (ideally, we would like to find an exact NEQ). Solving general-sum games can be hard, and no polynomial-time algorithm for an exact solution is known. However, the exact NEQ in a 2-players zero-sum game can be found by solving a linear program which size is linear in the size of the game, therefore the solution can be found in polynomial time [12]. Also, there exist iterative algorithms that can be used to compute an approximate solution via self-play. We will now describe two such algorithms: Fictitious Play and regret minimization approach.

2.5.1 Fictitious Play

Fictitious Play [3] is a simple algorithm that learns NEQ in 2 players zero-sum games by self-play. Each player holds a belief about his opponent's strategy, which is the empirical average of actions played by the opponent. At each round, both players select the action as the best response against their belief about the opponent's strategy.

Algorithm 4: Fictitious Play

```
Initialize vectors of actions counts for both players $\mathbf{c}^1 = 0, \mathbf{c}^2 = 0$;
for $t=1,\dots,T$ do
    Compute average strategies of both players $\sigma_1 \propto \mathbf{c}^1$, $\sigma_2 \propto \mathbf{c}^2$ (at
      first round when count vectors equals zeros, select the strategies
      arbitrarily);
    Player 1 choose action $i$ as a best response to $\sigma_2$;
    Player 2 choose action $j$ as a best response to $\sigma_1$;
    Increase $i$-th element of $\mathbf{c}^1$ and $j$-th element of $\mathbf{c}^2$ by 1.
end
```

In 2-players zero-sum games, Fictitious Play is guaranteed to converge to NEQ, that is:

$$\lim_{t \to \infty} (\sigma_1, \sigma_2) = (\sigma_1^*, \sigma_2^*) \tag{17}$$

However, the actual speed of convergence may be slow. Moreover, both players must be able to compute their best responses, which requires them to know their payoff matrices and to observe the opponent's actions.

2.5.2 Regret Minimization in Games

The concept of external regret minimization discussed in sections 2.3 and 2.4 has a strong tie to the quality of the solution learned by self-play. Following theorem [16] provides an upper bound on NEQ approximation based on the external regret of players:

Theorem 2.1. *Assume that 2-players zero-sum game is played repeatedly by two players for T rounds. If the average external regret of both players if lower or equal to ϵ, then their average strategies form an $2\epsilon-NEQ$.*

This theorem gives us a tool to compute an ϵ-NEQ by self-play: we may use any algorithm whose cumulative external regret is sublinear in the time horizon T. The regret bound of these algorithms will then directly translate to the bound of NEQ approximation quality. We may use either algorithm for regret minimization in a full information setting (if the players can observe expected payoff for all their actions), or algorithms for adversarial bandit setting, which require each player to observe only the payoff for action that he played at given round. Therefore, we do not require the players to compute the best response as in the Fictitious Play.

The external regret can be also used to provide a lower bound on the approximation quality of self-play algorithms[3]:

[3]The theorem 2.2 and its proof is an original contribution of this thesis.

Theorem 2.2. *Assume that 2-players zero-sum game is played repeatedly by two players for T rounds. If the average external regret of both players if larger or equal to δ, then their average strategies cannot form an ϵ-NEQ for $\epsilon < \delta$.*

Proof. Let $\bar{\sigma}_i, \bar{\sigma}_2$ be the average strategy of player i, and let σ_i^t be the strategy of player i at round t. The average external regret of player 1 at time T is

$$R_1^T = \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \bar{\sigma}_{-i}) - \frac{1}{T} \sum_{t=1}^{T} u_i(\sigma_i^t, \sigma_{-i}^t) \tag{18}$$

Using the fact that $u_1(\sigma) = -u_2(\sigma)$ for all strategy profiles σ, we have

$$R_1^T + R_2^T = \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \bar{\sigma}_2) + \max_{\sigma_2' \in \Sigma_2} u_2(\bar{\sigma}_1, \sigma_2') \tag{19}$$

Let strategy profile $(\bar{\sigma}_1, \bar{\sigma}_2)$ be an ϵ-NEQ for some $\epsilon \in R$ (each strategy profile is ϵ-NEQ for large enough ϵ). This implies

$$\max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \bar{\sigma}_2) \leq u_1(\bar{\sigma}_1, \bar{\sigma}_2) + \epsilon \tag{20}$$

and

$$\max_{\sigma_2' \in \Sigma_2} u_2(\bar{\sigma}_1, \sigma_2') \leq u_2(\bar{\sigma}_1, \bar{\sigma}_2) + \epsilon \tag{21}$$

By summing both inequalities, we obtain

$$\max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \bar{\sigma}_2) + \max_{\sigma_2' \in \Sigma_2} u_2(\bar{\sigma}_1, \sigma_2') \leq 2\epsilon \tag{22}$$

where we again used the fact that $u_1(\sigma) = -u_2(\sigma)$ for all strategy profiles σ. By equation 19, the left hand side of equation 22 is equal to $R_1^T + R_2^T$. Since $R_1^T \geq \delta, R_2^T \geq \delta$ we obtain

$$2\delta \leq \max_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \bar{\sigma}_2) + \max_{\sigma_2' \in \Sigma_2} u_2(\bar{\sigma}_1, \sigma_2') \leq 2\epsilon \tag{23}$$

that is,

$$\delta \leq \epsilon \tag{24}$$

□

The bounds from theorems 2.1 and 2.2 can be summarized as follows:

Corollary. *Assume that 2-players zero-sum game is played repeatedly by two players for T rounds. If the average external regret of both players lies in $[\delta_1, \delta_2]$, then their average strategies form an ϵ-NEQ for $\epsilon \in [\delta_1, 2\delta_2]$ and cannot form an ϵ-NEQ for any other ϵ.*

2.6 Reinforcement learning

Reinforcement learning (RL) [15] problem is similar to the stochastic bandit problem but adds the complexity of multiple states and transitions between them. There exist different formal definitions of reinforcement learning problem (including problems with continuous state spaces). For the purpose of this thesis, we will define the reinforcement learning problem as an episodic Markov decision process:

Definition 2.14. Reinforcement learning problem is a tuple $(S, A, P, R, s_{start}, S_T)$, where:

- S is a finite set of states.
- A is a finite set of actions. The set of actions available in state $s \in S$ is denoted A_s. We assume that A_s are disjunctive, that is, each action can be played only in one state.
- P is a set of P_a, where $P_a : A \to \Pi(S)$ assigns a probability distribution over states given that action $a \in A_s$ is played in state $s \in S$. The probability that the agent transit to state s' given that he played action a in state s is denoted by $Pr(s'|s, a)$.
- $R : A \to \mathbb{R}$ is a reward function. The reward for playing action $a \in A$ is denoted $R(a)$.
- $S_T \subset S$ is a set of terminal states.
- $s_{start} \in S$ is a starting state.

The interaction between the agent and the environment proceeds as follows: The agent starts in the state s_{start}. In each state s, the agent choose an action from A_s. Then, the next state s' is sampled by P_a, and agent receive reward $R(a, s')$. If $s' \in S_T$, the process ends. The goal of the agent is to maximize the sum of rewards obtained.

There exist a broad range of algorithms designed to solve the reinforcement learning problem. In this thesis, we will be interested in algorithms that learn to solve the RL task by direct interaction with the environment, without having knowledge of the full specification of the problem. That is, such algorithms interact with the environment, and observe only the states and rewards from that interaction. We will now describe two such algorithms: Q-learning, which is similar to a RL version of the Action-Value Method, and Policy Gradient, which is a RL version of the Gradient Bandit Algorithm.

2.6.1 Q-learning

Similar to the Action-Value Method for bandit setting, Q-learning [15] tries to learn a values associated with playing possible actions. The policy of the algorithm is then given by playing the action with the highest value, mixed with uniform distribution over all available actions played with probability ϵ for the purpose of exploration.

However, the value of playing an action cannot be defined only by the expected reward gained immediately after playing the action: it may be that action bring low immediate reward, but helps the agent to transit into states in which high rewards may be obtained.

Let agent's policy be $\pi : S \to \Pi(A_s)$, where $\pi(s)$ specify the probability distribution over actions A_s. Let the probability that the agent plays an action a in state s be denoted $\pi(a|s)$. The value of playing action a and then following policy π is then given by the recursive equation

$$Q^{\pi}(a) = R(a) + \sum_{s' \in S} \left(Pr(s'|s,a) \sum_{a' \in A_{s'}} \pi(a'|s') Q^{\pi}(a') \right) \tag{25}$$

We can use the equation 25 to specify the conditions of optimality: if the agent uses the policy which maximizes his expected sum of rewards, he must play the action which maximizes $Q(a)$ in each state. Let $Q^*(a)$ denotes the value of action a under the optimal policy. Then we have:

$$Q^*(a) = R(a) + \sum_{s' \in S} Pr(s'|s,a) \max_{a' \in A'_s} Q*(a') \quad \forall a \in A \tag{26}$$

The equation 26 is known as Bellman equation. The idea of Q-learning is to try to learn the function $Q^*(a)$ and use it to select the actions. Specifically, the Q-learning algorithm holds and estimated value of $Q(a)$ for each action $a \in A$. The policy of the algorithm is to the play action with the highest estimated value with probability of $1 - \epsilon$, and to select the action uniformly at random with probability ϵ, where $\epsilon \in (0,1)$ is an exploration parameter. Similar to Action-value method, the ϵ parameter should diminish in with according to the GLIE scheme.

When Q-learning plays action a in state s, it receive the reward $R(a)$ and transit to state s'. The value estimate $Q(a)$ is then updated as

$$Q(a) := Q(a) + \alpha \left(R(a) + \max_{a' \in A_{s'}} Q(a') - Q(a) \right) \tag{27}$$

where $\alpha \in (0,1)$ is the learning rate of the algorithm.

The difference between Action-Value Method and a single-state Q-learning is that the Action-Value Method uses a simple average of the rewards as an estimate of action's value, while the Q-learning uses an exponential moving average instead of simple average.

Algorithm 5: Q-learning with ϵ-greedy exploration

```
Initialize $Q(a)$ arbitrarily $\forall a \in A$;
Initialize exploration parameter $\epsilon \in (0, 1)$;
Set learning rate $\alpha \in (0, 1)$;
for each episode do
    $s = s_{start}$;
    while episode is not over do
        With probability $\epsilon$, select action $a$ uniformly at random;
        Otherwise, play action $a = \arg\max(Q(a))$;
        Receive reward $R(a)$ and transit to state $s'$;
        Update $Q(a) := Q(a) + \alpha \left(R(a) + \max_{a' \in A_{s'}} Q(a') - Q(a)\right)$ ;
        update $s = s'$;
        optionaly update $\epsilon$ according to the GLIE schema
    end
end
```

2.6.2 Policy Gradient

Policy Gradient [15] is an extension of the Gradient Bandit Algorithm to the full RL setting. In each state, the strategy of Policy Gradient is parameterized by a vector of weights $\mathbf{w}$, and the selection of action is done by sampling from $softmax(\mathbf{w})$. After each episode, the gradient of expected rewards with respect to the weight vectors is computed and used to update the weights.

There exist different variants of Policy Gradient. Then one that we will describe is also known as a REINFORCE algorithm or as a Monte Carlo Policy Gradient.

Algorithm 6: Policy Gradient

```
Initialize $\mathbf{w}^s = \mathbf{0}$ $\forall s \in S$;
Set learning rate $\eta > 0$;
for each episode do
    generate sequence of states, actions and rewards
    $s_1, a_1, r_1, ..., s_T, a_T, r_T$ using policy $\pi = softmax(\mathbf{w}^s)$ in state s;
    set $G = \sum_{t=1}^{T} r_t$;
    for each state s in generated sequence do
        estimate $\nabla G$ with respect to $\mathbf{w}^s$ using importance weighted
          estimator ;
        update $\mathbf{w}^s := \mathbf{w}^s + \eta \nabla G$
    end
end
```

The algorithm 6 can be further modified in a number of ways. For example, instead of summing all rewards from the sequence, we may sum only the rewards obtained after state s was visited. Also, the rewards are usually time-discounted, with rewards generated later in the sequence heaving lower weight. However, we will be interested in applying Policy Gradient to the extensive from games in which the rewards are obtained only in terminal states, and such modifications will, therefore, have no effect.

The Policy Gradient also uses an importance weighted estimator to obtain estimates of G if different actions would be played. The baseline parameter b of the importance weighted estimator may be set in various ways - the vanilla version of Policy Gradient uses $b = 0$, however, we may use a more sophisticated selection of b to lower the variance of the estimator.

2.7 Extensive Form Games with Imperfect Information

Extensive form games with imperfect information [11] involve multiple states and transitions between them. Also, the players face the problem of imperfect information as they cannot fully observe the current state of the game. The formal definition follows:

Definition 2.15. A two-player extensive form game with imperfect information is a tuple (N, H, A, P, f_c, I, u), where:

- $N = \{1, 2\}$ is a set of players.
- H is a finite set of sequences of actions called histories. An empty sequence is in H, and every prefix of sequence in H is also in H. $Z \subseteq H$ are terminal sequences, which are not a prefix of any other sequences.
- A is a set of actions. $A(h) = \{a : (h, a) \in H\}$ denotes the set of action available at nonterminal history h.
- $P : H \backslash Z \to N \cup \{chance\}$ is a player function. For each nonterminal sequence $h \in H \backslash Z$, it assigns a player which acts in h. Besides the players 1 and 2, the h can be assigned to the chance, meaning that the action in h will be selected at random.
- $f_c : \{h : P(h) = chance\} \to \Pi(A(h))$ is a function that for each history controlled by the chance specify the probability distribution over actions played in that history.
- $I = \{I_1, I_2\}$, where I_i is a partition of $\{h \in H : P(h) = i\}$ for player i with the property that $A(h) = A(h')$ whenever h and h' are in the same member of partition. Members of the partition are called information sets.
- $u = (u_1, u_2)$ where $u_i : Z \to R$ is a payoff function of player i. If $u_1(z) = -u_2(z) \forall z \in Z$, the game is called zero-sum.

The game is played as follows: in each history h, the player $P(h)$ select his action. The player does not observe h, but only the information set to which h belongs. When the game reaches terminal history $z \in Z$, both players receive rewards given by their corresponding payoff function, and the game ends. The goal of the players is to maximize their rewards.

As in matrix games, we denote σ_i the strategy of player i, where σ_i specify the probability distribution over all actions available to player i. All the definitions involving strategies (such as dominance, Nash Equilibrium and so on) can be directly applied to the extensive form games with imperfect information.

The definition of extensive-form games with imperfect information allows games in which the players forgot the information which was previously available to them. We say that the game is of perfect recall if both players never forget any information. For the rest of this thesis, we will consider only games with perfect recall. Also, we will focus only on zero-sum games.

Matrix games are a subset of extensive-form games with imperfect information. They may be represented by an extensive form game in which player 1 selects his action at first, but this action is not revealed to player 2. Player 2 is then in a history defined by the action taken by player 1, but all these possible histories are in the same information set. When player 2 selects his action, the game terminates and players receive the utility given by the payoff function.

Extensive form games with imperfect information can be also converted to matrix game if each pure strategy in extensive form game is considered to be an action in the matrix game. However, this conversion leads to exponential blow up in the size of the game, as the extensive form is much more concise representation than the matrix game.

The NEQ of zero-sum extensive form game with incomplete information can be found by solving a linear program, which size is linear in the size of the game.[12]

The regret minimization approach can also be used to find an approximate NEQ in extensive form games. It involves minimizing so-called *counterfactual regret*, which is a local variant of regret minimized independently in each information set. As shown by (Zinkevich et al., 2008)[17], the minimization of counterfactual regrets leads to the minimization of the overall regret in the whole game. The resulting algorithm is called Counterfactual Regret Minimization (CFR), and its variants are used in practice to solve large games (such as poker).

3 Self-play of Action-Value Method with ϵ-greedy Exploration

In this chapter, the properties of the Action-Value Method (algorithm 1) in self-play in matrix games will be analyzed and discussed.

3.1 Averaging of strategies

We will begin by observing the fact that the action-value method always plays a mixture of pure strategy (given by the argument of maxima of estimated actions values) and uniform distribution. The proportion of these two parts is given by the exploration parameter ϵ. As the ϵ is usually selected to be small (e.g. 0.1), the selected strategy is close to pure strategy.

However, there exist games in which any pure strategy is the worst possible strategy in the term of exploitability. For example, consider the game of rock-paper-scissors, which has the payoff matrix given by table 1. Each player has 3

	Rock	Paper	Scissors
Rock	0	-1	1
Paper	1	0	-1
Scissors	-1	1	0

Table 1: Rock-paper-scissors game

possible actions: rock, paper and scissors, and the elements of matrix show the payoff of player 1 for each combination of player's actions (rows correspond to actions of player 1 and columns correspond to actions of player 2). The game is zero-sum, and the payoffs of player 2 are minus the payoffs of player 1. In the NEQ, each player is uniformly mixing among all actions, and the value of the game is 0.

Against any pure strategy, the best response of an opponent will cause the player to obtain a payoff of -1, thus the exploitability of any pure strategy is 1, which is the maximum exploitability across all possible strategies in this game. If the Action-Value Method has an exploration parameter ϵ, the exploitability of its strategy will be $1 - \frac{2}{3}\epsilon$. Thus, the game of rock-paper-scissors provides a simple example of a game in which the strategy played by Action-Value Method cannot be a good approximation of NEQ.[4]

However, instead of the current strategy of the algorithm, we may use the average strategy from the whole duration of self-play. That approach is inspired by other self-play algorithms: both for Fictitious Play and regret minimization approach, it is the average strategy that converges to NEQ, not the current strategy played by the algorithm at the last round. The convergence of average

[4]That holds for low ϵ. For $\epsilon = 1$, the action-value method will uniformly mix among all actions, which is a NEQ strategy in the game of rock-paper-scissors. However, such behavior will not be a good strategy in other games, as will be shown later on.

strategy is a weaker condition than the convergence of current strategy: if the current strategy converges, then the average strategy must converge as well, however, it may be that the average strategy converges to NEQ and current strategy does not. Therefore, for the rest of this chapter, we will focus only on the convergence of average strategies. Also, we will average the actions only from the rounds in which the player was not exploring (adding the actions from exploration moves will simply mean mixing the learned strategy with uniform distribution among all actions).

3.2 Bound on approximation quality

In the stochastic bandit problem, the exploration move (playing random action) may hinder the agent's average obtained reward, as he will play actions that yield low expected reward. However, the exploration allows the agent to obtain samples from the probability distribution associated with each action, and this will cause the agent's estimate of action values to be consistent (i.e. to converge to the ground truth values in probability).

We will now show that in the self-play of matrix game, the exploration may prevent the agent from correctly learning the values of actions obtained in equilibrium. As a consequence, the agent may fail to play actions that are played in NEQ.

Proposition 1. *Assume that 2-players zero-sum game is played in self-play by Action-Value Method with constant exploration parameter $\epsilon \in (0, 1)$. There exists a game with payoffs normalized between -1 and 1, in which the average strategies of agents will converge to $\frac{\epsilon}{2}$-NEQ.*

Proof. Consider the following game:

	Left	Right
Up	-1	0
Down	1	1-$\frac{\epsilon}{2}$

For player 1, the estimated value of action Down will always be higher than the estimated value of action Up. Therefore, player 1 will play Up only as an exploration move, which will happen with probability $\frac{\epsilon}{2}$ (player is exploring with probability ϵ, and the probability of playing Up during exploration is $\frac{1}{2}$).

The expected payoff of player 2 for playing Left is then $-1 + \epsilon$, and the expected payoff for playing Right is $-1 + \epsilon - \frac{\epsilon^2}{4}$. Since the player 2's estimated payoff will converge in probability to the expected values, he will learn to play only the action Left (the action Right will be played only for the purpose of exploration).

The only NEQ in this game is (Down, Right). The players will learn to play strategy profile (Down, Left), which is an $\frac{\epsilon}{2}$-NEQ. Namely, the player 2 will fail to correctly play the equilibrium action Right and will be exploitable for the utility of $\frac{\epsilon}{2}$. □

Compared to the stochastic bandit problem, the agent in self-play becomes a part of the environment for the other player. If the agent then plays suboptimal action for the purpose of exploration, the other player will learn to exploit this play. However, since no suboptimal action can be played in the equilibrium, the other player's estimated actions values will become inconsistent with the equilibrium values.

The proposition above directly implies that in order to learn an equilibrium, it is necessary to update the exploration parameter so that it diminishes to zero. Combined with the fact that each action must be sampled infinitely often (otherwise, there is a positive probability that suboptimal action will be learned), we get the following corollary:

Corollary. *Assume that 2-players zero-sum game is played in self-play by the Action-Value Method with the exploration parameter given as a function of timestep $\epsilon(t)$. Assuming that $lim_{t\to\infty}\epsilon(t)$ exists, the GLIE schema (given by equations 3 and 4) is a necessary condition for the convergence of average strategies to NEQ.*

3.3 Removal of strictly dominated actions

Strictly dominated strategies are never played in NEQ. Therefore, any self-play algorithm that strives to learn NEQ must learn not to play strictly dominated strategies. However, I managed to prove only a relatively weak version of this property for the Action-Value Method:

Theorem 3.1. *Assume that 2-players zero-sum game is played in a self-play by Action-Value Method with exploration parameter $\epsilon(t)$ updated by the GLIE schema. If one player has only 2 actions and one of them is strictly dominated, then the player will learn not to play the strictly dominated action.*

To prove theorem 3.1, it will be useful to introduce the following definition:

Definition 3.1. We say that action $a_i \in A_i$ strongly dominates action $a'_i \in A_i$, if

$$\min_{a_{-i}\in A_{-i}} u_i(a_i, a_{-i}) \geq \max_{a'_{-i}\in A_{-i}} u_i(a'_i, a'_{-i}) \tag{28}$$

and

$$\min_{a_{-i}\in A_{-i}} u_i(a_i, a_{-i}) > \min_{a'_{-i}\in A_{-i}} u_i(a'_i, a'_{-i}) \tag{29}$$

The definition above says that one action strongly dominates the other one if each outcome from playing the first action is at least as good as any outcome from playing the second action, and the inequality holds as strict for at least one opponent's action.

In a self-play of the Action-Value Method, the value estimate of action that strongly dominates another one will be at least as high as the estimate of the dominated action. Moreover, this inequality will become strict when the first player plays the dominating action and the second player plays the action corresponding to the lowest utility for the first player. With the GLIE scheme and infinite time horizon, such outcome will be sampled at least once with probability 1.

We may now prove theorem 3.1:

Proof. Without loss of generality, assume that player 1 has 2 actions, and the first one strictly dominates the second one. Player 2 has $n \in N$ actions. The payoff matrix for player 1 is given by a matrix $P \in R^{2 \times n}$.

We will distinguish two cases: whether or not is the action 2 of player 1 strongly dominates by his action 1.

If it is strongly dominated, the value estimate of action 1 will become higher with probability 1, and the player 1 will thus learn to not play the strictly dominated action 2.

If the action 1 does not strongly dominate the action 2, by the definition of strong dominance and by the fact that action 2 is strictly dominated, there exist actions i and j of player 2 such that $P_{2,i} > P_{1,j}$. From the strict dominance of player 1's action 2 we obtain $P_{1,j} > P_{2,j}$ and $P_{1,i} > P_{2,i}$. This implies that the player's 2 action i is strongly dominated by action j, since $P_{1,i} > P_{2,i} > P_{1,j} > P_{2,j}$.

As action i is strongly dominated and both players are using GLIE schema to update $\epsilon(t)$, the action i will be eventually played with arbitrary low (but positive) probability (since it will be played only in the exploration moves, and $lim_{t \to \infty} \epsilon(t) = 0$). For any $\delta \in R, \delta > 0$ the estimates of player 1's action values will converge with probability 1 to δ-neighborhood of the values they will reach in a game in which the action i would not exist.

This will hold for any strongly dominated action of player 2. For any $\delta \in R, \delta > 0$ the estimates of player's 1 action values will converge with probability 1 to δ-neighborhood of the values they will reach in a game in which the strongly dominated actions of player 2 would not exist.

Let P' denotes the payoff matrix P without the strongly dominated actions of player 2. We have $P'_{1,i} \geq P'_{2,j} \forall i, j \in \{1, ...n\}$ by the absence of player 2's strongly dominated actions, and $P'_{1,i} > P'_{2,i} \forall i \in \{1, ...n\}$ since action 2 of player 1 is strictly dominated. Therefore, action 2 of player 1 is strongly dominated, and player 1 will learn to not play it.

□

The proposition that the Action-Value Method will learn not to play strictly dominated actions in a game of general size show up to be non-straightforward to prove, and I didn't manage to prove nor disprove it. We will now show some counter-intuitive behavior of the Action-Value Method, to illustrate why finding the proof or counterexample is somehow difficult.

For a numerical example, consider the game with payoffs given by table 2.

Table 2: Game used in the numerical example

	left	right
up	4	2
mid	3	1
down	0	5

Assume that the game is played in self-play by an Action-Value Method with constant[5] exploration parameter $\epsilon = 0.3$. Assume that the estimated action values are $q_{up} = 2.5$, $q_{mid} = \frac{31}{13} \doteq 2.38$ and $q_{down} = \frac{20}{9} \doteq 2.22$ for player 1, and $q_{left} = -\frac{1027}{1259} \doteq -0.82$ and $q_{right} = -\frac{5504}{1554} \doteq -3.54$ for player 2. Also assume that the count of times the actions were played is $c_{up} = 1000$, $c_{mid} = 13$ and $c_{down} = 1800$ for player 1 and $c_{left} = 1259$ and $c_{right} = 1554$ for player 2. There exist a possible history of the players' exploration moves that will lead to such values.

The action up of player 1 strictly dominates the action mid. Also, the estimated value 2.5 of action up is currently higher than the estimated value 2.38 of action mid. If they do not play exploratory moves, the players now play action profile (up, left) which correspond to the actions with the currently highest estimated values for both players.

We may run Monte Carlo simulation of the game to estimate how the actions with the highest estimated values will change. We may expect that players will after some time switch from playing (up, left) to (up, right) since the expected utility from playing right is higher than the expected utility from playing left for player 2.

However, it shows up that there is about 57% probability that the player 1 will switch to play the dominated action mid instead, i.e. the actions with highest estimated values will switch from (up, left) to (mid, left). In the remaining 43%, the player 2 will switch to play right.

The reason why player 1 will switch to play dominated action with such high probability is given by the fact that he sampled that action only a few times in history, compared to the number of times he sampled action up. Therefore, the expected increase of estimated value for action mid is higher than the expected increase for the estimated value of action up, even though the expected payoff for playing action up is higher than the expected value for playing action mid.

3.4 Empirical Evaluation

To estimate the practical performance of Action-Value Method in self-play, we run a number of simulations.

The simulation were run with various settings of the parameters: we varied the time horizon, value or update scheme of ϵ and the size of the game. We tried the time horizons of $10^4, 10^5, 10^6$ and 10^7 timesteps. For the exploration

[5]The example will also work for ϵ updated according to the GLIE schema, we choose constant ϵ for simplicity.

parameter ϵ, we tried constants values of $0.2, 0.1, 0.05$ and 0.01, as well as GLIE schema of $\epsilon(t) = \frac{10^5}{10^6 + t}$, $\epsilon(t) = \frac{10^5}{5 \times 10^6 + t}$ and $\epsilon(t) = \frac{10^4}{10^5 + t}$. The sizes of games were set to be 3×3, 5×5 and 10×10. The payoffs were generated from a uniform distribution and normalized to be between 0 and 1 in each game.

We run 1000 simulations for each possible combination of the parameters above, and report the average Nash distances of time-averaged strategies. Their standard deviations are reported in parentheses.

Table 3: Empirical evaluation of the Action-Value Method in self-play

Game size 3×3

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$\epsilon = 0.2$	0.019 (0.017)	0.018 (0.016)	0.018 (0.015)	0.016 (0.015)
$\epsilon = 0.1$	0.020 (0.021)	0.014 (0.015)	0.012 (0.012)	0.010 (0.010)
$\epsilon = 0.05$	0.029 (0.036)	0.021 (0.025)	0.017 (0.021)	0.013 (0.016)
$\epsilon = 0.01$	0.061 (0.069)	0.049 (0.058)	0.044 (0.051)	0.038 (0.048)
$\epsilon = \frac{10^5}{10^6+t}$	0.019 (0.022)	0.014 (0.016)	0.010 (0.012)	0.011 (0.017)
$\epsilon = \frac{10^5}{5\times10^6+t}$	0.049 (0.058)	0.037 (0.045)	0.031 (0.039)	0.030 (0.037)
$\epsilon = \frac{10^4}{10^5+t}$	0.018 (0.020)	0.014 (0.016)	0.013 (0.020)	0.015 (0.026)

Game size 5×5

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$\epsilon = 0.2$	0.034 (0.020)	0.029 (0.016)	0.028 (0.015)	0.028 (0.015)
$\epsilon = 0.1$	0.040 (0.028)	0.029 (0.022)	0.024 (0.018)	0.023 (0.014)
$\epsilon = 0.05$	0.056 (0.043)	0.039 (0.031)	0.032 (0.029)	0.031 (0.028)
$\epsilon = 0.01$	0.098 (0.057	0.81 (0.590)	0.068 (0.056)	0.065 (0.052)
$\epsilon = \frac{10^5}{10^6+t}$	0.040 (0.028)	0.028 (0.021)	0.024 (0.021)	0.032 (0.031)
$\epsilon = \frac{10^5}{5\times10^6+t}$	0.086 (0.069)	0.063 (0.049)	0.056 (0.045)	0.054 (0.045)
$\epsilon = \frac{10^4}{10^5+t}$	0.038 (0.029)	0.030 (0.023)	0.034 (0.032)	0.046 (0.051)

Game size 10×10

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$\epsilon = 0.2$	0.054 (0.021)	0.045 (0.017)	0.042 (0.018)	0.042 (0.018)
$\epsilon = 0.1$	0.065 (0.027)	0.048 (0.022)	0.041 (0.021)	0.040 (0.022)
$\epsilon = 0.05$	0.087 (0.036)	0.058 (0.027)	0.049 (0.027)	0.046 (0.024)
$\epsilon = 0.01$	0.138 (0.055)	0.116 (0.054)	0.084 (0.046)	0.073 (0.045)
$\epsilon = \frac{10^5}{10^6+t}$	0.065 (0.025)	0.046 (0.021)	0.041 (0.021)	0.069 (0.037)
$\epsilon = \frac{10^5}{5\times10^6+t}$	0.121 (0.048)	0.085 (0.042)	0.078 (0.037)	0.067 (0.038)
$\epsilon = \frac{10^4}{10^5+t}$	0.064 (0.025)	0.049 (0.023)	0.071 (0.032)	0.122 (0.060)

We may observe two interesting patterns in the results:

At first, the Nash distance almost always decreases with a longer time horizon, thus, running the self-play for a longer time leads to better results.

At second, the optimal value of ϵ varies with the time horizon and size of the game. With a lower time horizon, the exploration parameter should be high, otherwise, the agent will not manage to explore enough. For a longer time horizon, the exploration parameter can be lower. Regarding the size of the game, higher ϵ works better in larger games, where the agent needs to explore more available actions. This is in line with optimal values of parameters for other algorithms - for example, the Exp3 algorithm selects a lower learning rate (thus exploring more) for environments with more actions.

As a benchmark, we also run the simulations with the Exp3 algorithm. The strategies of Ex3 were averaged across time, and the resulting Nash distances are reported in table 4. Overall, the Exp3 performed much better than the Action-Value Method.

Table 4: Self-Play of Exp3 Algorithm

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
Game size 3×3	0.022 (0.010)	0.007 (0.003)	0.003 (0.002)	0.001 (0.001)
Game size 5×5	0.036 (0.012)	0.013 (0.005)	0.004 (0.002)	0.002 (0.001)
Game size 10×10	0.058 (0.014)	0.022 (0.005)	0.007 (0.002)	0.002 (0.001)

We also tried to find a specific game in which the self-play of Action-Value Method will fail to converge. To do that, we generated 1000 random games of size 5×5, and run a self-play with parameter $\epsilon = 0.1$ and time horizon $T = 10^8$. For each game, we run the self-play 10 times and averaged the Nash distance reached in these 10 runs. The game leading to the largest Nash distance has the average Nash distance of 0.116 and has the payoff matrix given by table 5.

Table 5: Game hard for the Action-Value Method

0.56	0.37	0.42	0.32	0.01
0.18	0.62	0.89	0.40	0.83
0.80	0.98	0.01	0.47	0.07
0.13	0.93	0.00	0.02	1.00
0.19	0.19	0.21	0.98	0.98

It is hard to deduce anything about the limit convergence for $T \to \infty$. We know that in order to reach an exact NEQ, we will need to progressively lower ϵ to 0 according to the GLIE schema, but we do not know whether this is sufficient condition for the convergence. In the experiments, the Nash distance usually improved when the self-play was run for a longer time, but the game in table 5 is an example of a game in which even the time horizon of 10^8 leads to fairly bad performance.

3.5 Conclusion

In this chapter, we analyzed the properties of Action-value Method in the self-play of matrix games.

We provided a bound on the NEQ approximation quality with respect to the exploration parameter ϵ. Consequently, following a GLIE schema for the updates of ϵ is a necessary condition for the convergence, but we do not know whether this condition is also sufficient.

We also show that Action-Value Method will learn not to play strictly dominated actions in games with size $2 \times n$ where the strictly dominated action belongs to the player with only 2 actions.

Finally, we run a series of experiments to evaluate the practical performance of the Action-Value method. It showed up that correctly setting the exploration parameter ϵ is crucial for the performance of the algorithm. However, the overall performance is significantly worse than the benchmark of Exp3 algorithm.

4 Self-play of the Gradient Bandit Algorithm

In this chapter, the properties of the Gradient Bandit Algorithm (algorithm 2) in self-play will be analyzed and discussed.

We will show that in some simplified setting, the Gradient Bandit Algorithm may be approximated by an instance of the Lagrangian Hedging. We will prove a regret bound for the approximating algorithm and evaluate the quality of the approximation.

Then, we will evaluate the performance of the Gradient Bandit Algorithm in a self-play of matrix games.

4.1 Equivalence of Regrets and Rewards

We will begin by observing that the gradient ascent on (possibly estimated) reward vector $\mathbf{x}$ is equivalent to the gradient ascent on immediate regret vector $\mathbf{r}$ for the Gradient Bandit Algorithm. This comes from the fact that immediate regret vector is just a reward vector shifted by a constant (the expected utility obtained by the agent), and that the softmax function is invariant to a shift by a constant. That is, for each $i \in \{1, ..., k\}$ we have:

$$\frac{\partial \mathbf{x}^\intercal \pi(\mathbf{w})}{\partial \mathbf{w}_i} = \pi_i(\mathbf{w}) \left(\mathbf{x}_i - \sum_{j=1}^{k} \pi_j(\mathbf{w}) \mathbf{x}_j \right) \tag{30}$$

$$= \pi_i(\mathbf{w}) \left(\mathbf{x}_i - \mathbf{x}^\intercal \pi(\mathbf{w}) - \sum_{j=1}^{k} \pi_j(\mathbf{w}) \left(\mathbf{x}_j - \mathbf{x}^\intercal \pi(\mathbf{w}) \right) \right) \tag{31}$$

$$= \pi_i(\mathbf{w}) \left(\mathbf{r}_i - \sum_{j=1}^{k} \pi_j(\mathbf{w}) \mathbf{r}_j \right) = \frac{\partial \mathbf{r}^\intercal \pi(\mathbf{w})}{\partial \mathbf{w}_i} \tag{32}$$

The first equality comes from the definition of softmax derivatives. The second equality comes from adding $0 = \mathbf{x}^\intercal \pi(\mathbf{w}) - \mathbf{x}^\intercal \pi(\mathbf{w})$ and from the fact that $\sum_{j=1}^{k} \pi_j(\mathbf{w}) = 1$. The third equality comes from the definition of immediate regret.

4.2 Approximation of the Gradient Bandit Algorithm in a Full Information Setting

We will now analyze the Gradient Bandit Algorithm in a full information setting - we will assume that the agent can directly observe the whole reward vector $\mathbf{x}$, instead of relying on its estimate.

The update of weight vector can be then written as

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \frac{\partial (\mathbf{r}^t)^\intercal \pi(\mathbf{w})}{\partial \mathbf{w}} \tag{33}$$

The equation above can be seen as a difference equation, which solution depends on the sequence of regret vectors. To gain an insight into the behavior of the Gradient Bandit Algorithm, it would be useful to find an analytical solution for the weight vector at given time. However, finding such solution showed up to be difficult.

However, we may try to approximate the difference equation above by a differential equation, which solution would be a continuous function. For a low learning rate η, the solution of a discrete difference equation would be similar to the solution of differential equation evaluated at timesteps $t \in N$. We will now formulate and solve (at least for a special case of two actions) such differential equation. The quality of the approximation will be evaluated later on in section 4.4

To obtain the difference equation, we will replace the discrete weight update by a derivative:

$$\mathbf{w}'(t) = \eta \frac{\partial (\mathbf{r}^{\lceil t \rceil})^\intercal \pi(\mathbf{w}(t))}{\partial \mathbf{w}(t)} \tag{34}$$

The weight vector is now stated as a function of time $t \in R^0_+$. To state sequence of regret vectors as a function of time, the value of vector $\mathbf{r}^{\lceil t \rceil}$ is used at time t. Thus, the $\mathbf{r}^{\lceil t \rceil}$ is a piecewise constant function of time t.

The equation 34 specify a partial differential equation, which is hard to solve. To further simplify the analysis, we will now turn to the setting with two actions, where we may use sigmoid function instead of softmax. This will lead to an ordinary differential equation, for which we will be able to find an analytical solution.

For the rest of this section, we will assume that the number of available actions is $k = 2$, therefore $\mathbf{w}, \mathbf{r} \in R^2$. We have

$$\begin{aligned} softmax(\mathbf{w}) &= \left[\frac{\exp(\mathbf{w}_1)}{\exp(\mathbf{w}_1) + \exp(\mathbf{w}_2)}, \frac{\exp(\mathbf{w}_2)}{\exp(\mathbf{w}_1) + \exp(\mathbf{w}_2)} \right] && (35) \\ &= \left[\frac{\exp(\mathbf{w}_1 - \mathbf{w}_2)}{1 + \exp(\mathbf{w}_1 - \mathbf{w}_2)}, \frac{1}{1 + \exp(\mathbf{w}_1 - \mathbf{w}_2)} \right] && (36) \\ &= [\sigma(\mathbf{w}_1 - \mathbf{w}_2), 1 - \sigma(\mathbf{w}_1 - \mathbf{w}_2)] && (37) \end{aligned}$$

where $\sigma(x) = \dfrac{\exp(x)}{1 + \exp(x)}$ is a sigmoid function. Therefore, the value of softmax with two elements can be expressed as a function of only one value, the $\mathbf{w}_1 - \mathbf{w}_2$.

Also, we have

$$\frac{\partial \mathbf{r}^\intercal \pi(\mathbf{w})}{\partial \mathbf{w}_1} = \sigma'(\mathbf{w}_1 - \mathbf{w}_2)(\mathbf{r}_1 - \mathbf{r}_2) \tag{38}$$

and

$$\frac{\partial \mathbf{r}^\intercal \pi(\mathbf{w})}{\partial \mathbf{w}_2} = -\sigma'(\mathbf{w}_1 - \mathbf{w}_2)(\mathbf{r}_1 - \mathbf{r}_2) \tag{39}$$

We may observe that the equation 34 now depends only on $\mathbf{w}_1(t) - \mathbf{w}_2(t)$

and $\mathbf{r}_1^{\lceil t\rceil} - \mathbf{r}_2^{\lceil t\rceil}$:

$$\begin{aligned}\mathbf{w}'(t) &= \eta\frac{\partial(\mathbf{r}^{\lceil t\rceil})^\intercal \pi(\mathbf{w}(t))}{\partial \mathbf{w}(t)} \\ &= \eta\left[\sigma'(\mathbf{w}_1(t)-\mathbf{w}_2(t))(\mathbf{r}_1^{\lceil t\rceil} - \mathbf{r}_2^{\lceil t\rceil}), -\sigma'(\mathbf{w}_1(t)-\mathbf{w}_2(t))(\mathbf{r}_1^{\lceil t\rceil} - \mathbf{r}_2^{\lceil t\rceil})\right]\end{aligned} \tag{40}$$

Let $w(t) = \mathbf{w}_1(t) - \mathbf{w}_2(t)$ and $r(t) = \mathbf{r}_1^{\lceil t\rceil} - \mathbf{r}_2^{\lceil t\rceil}$. We may solve the following ordinary differential equation for $w(t)$ and use it to find a solution of equation 34:

$$\begin{aligned}w'(t) = \mathbf{w}_1'(t) - \mathbf{w}_2'(t) &= 2\eta\sigma'(\mathbf{w}_1(t)-\mathbf{w}_2(t))(\mathbf{r}_1^{\lceil t\rceil} - \mathbf{r}_2^{\lceil t\rceil}) \\ &= 2\eta\sigma'(w(t))r(t)\end{aligned} \tag{41}$$

The equation 41 may be solved by a separation of variables. Let $G(x)$ be a primitive function to $\dfrac{1}{\sigma'(x)}$ on R, and $H(t)$ be a primitive function to $2\eta r(t)$ on R_0^+. Then the solution of equation 41 has a form

$$w(t) = G^{-1}(H(t)+c) \tag{42}$$

where $c \in R$.

We have

$$\int \frac{1}{\sigma'(x)} = \int \frac{(\exp(-x)+1)^2}{\exp(-x)} = 2x + \exp(x) - \exp(-x) + c \tag{43}$$

which gives us

$$G(x) = 2x + \exp(x) - \exp(-x) \tag{44}$$

Also,

$$H(t) = 2\eta\sum_{t'=1}^{\lfloor t\rfloor}\left(\mathbf{r}_1^{t'} - \mathbf{r}_2^{t'}\right) + 2\eta(t-\lfloor t\rfloor)\left(\mathbf{r}_1^{\lceil t\rceil} - \mathbf{r}_2^{\lceil t\rceil}\right) \tag{45}$$

Specifically, for time t being an nonnegative integer, we have

$$H(t) = 2\eta\sum_{t'=1}^{t}\left(\mathbf{r}_1^{t'} - \mathbf{r}_2^{t'}\right) = 2\eta\left(\mathbf{s}_1^t - \mathbf{s}_2^t\right) \quad \forall t \in N_0 \tag{46}$$

where $\mathbf{s}^t$ is a cummulative regret vector at time $t \in N$. Plugging equations 44 and 46 into equation 42, we obtain

$$w(t) = G^{-1}\left(2\eta\left(\mathbf{s}_1^t - \mathbf{s}_2^t\right) + c\right) \quad \forall t \in N_0 \tag{47}$$

where G^{-1} is an inverse to function G given by equation 44. Unfortunately, this inverse cannot be expressed as a combination of elementary functions, and we will leave it in this form. Nevertheless, G^{-1} is defined on R, since $G(x)$ is strictly increasing on R.

Also note that $w(t)$ is defined on R_0^+ (such solution may be obtained by plugging equation 45 into equation 42), however, we are interested only in the solution for $t \in N_0$.

We also know the initial condition $w(0) = 0$. Using it, we may solve for the constant c in equation 47, obtaining $c = 0$. This gives us

$$w(t) = G^{-1}\left(2\eta\left(\mathbf{s}_1^t - \mathbf{s}_2^t\right)\right) \quad \forall t \in N_0 \tag{48}$$

Finally, we may use $w(t)$ to obtain the solution for the weight vector $\mathbf{w}(t)$ as

$$\mathbf{w}(t) = \mathbf{w}(\mathbf{s}^t) = \left[\frac{1}{2}G^{-1}\left(2\eta\left(\mathbf{s}_1^t - \mathbf{s}_2^t\right)\right), -\frac{1}{2}G^{-1}\left(2\eta\left(\mathbf{s}_1^t - \mathbf{s}_2^t\right)\right)\right] \tag{49}$$

This equation gives us the following algorithm which approximates the original Bandit Gradient Algorithm in a full information setting with 2 available actions:

Algorithm 7: Approximation of the Gradient Bandit Algorithm in a full information setting

```
initialize $\mathbf{s} = \mathbf{0}, \eta > 0$;
for $t=1,...,T$ do
    compute $\mathbf{w}$ as a function of $\mathbf{s}$ according to the equation 49;
    select policy $\pi(\mathbf{w}) = softmax(\mathbf{w})$;
    observe reward vector $\mathbf{x}$ and receive a reward $\mathbf{x}^\intercal \pi(\mathbf{w})$;
    compute regret vector $\mathbf{r} = \mathbf{x} - \mathbf{1}\mathbf{x}^\intercal \pi(\mathbf{w})$;
    update $\mathbf{s} := \mathbf{s} + \mathbf{r}$
end
```

The function $\pi(\mathbf{w}(\mathbf{s}))$ is defined in R^2 and is continuous in its whole domain. Thus, there exist a function $F(\mathbf{s})$ such that

$$\nabla F(\mathbf{s}) = \pi(\mathbf{w}(\mathbf{s})) \tag{50}$$

Therefore, the algorithm 7 select its policy as a gradient of the function $F(\mathbf{s})$, which implies that the algorithm 7 is an instance of the Lagrangian Hedging (algorithm 3).

4.3 Regret Bound for the Approximation of Gradient Bandit Algorithm

As shown in the previous section, the approximation of Gradient Bandit Algorithm is an instance of Lagrangian Hedging. There exists the following theorem [4], which provides a regret bound for the Lagrangian Hedging, given the properties of potential function $F(\mathbf{s})$:

Theorem 4.1. *Let potential function F be convex and satisfy*

$$F(\boldsymbol{s}) \leq 0 \quad \forall \boldsymbol{s} \leq \boldsymbol{0} \tag{51}$$

Also, let F satisfies

$$F(\boldsymbol{s}+\boldsymbol{d}) \leq F(\boldsymbol{s}) + \boldsymbol{d}\nabla F(\boldsymbol{s}) + C\left\|\boldsymbol{d}\right\|^2 \quad \forall \boldsymbol{s}, \boldsymbol{d} \in R^k \tag{52}$$

and

$$[F(\boldsymbol{s})+A]_+ \geq \inf_{\boldsymbol{s}'\leq \boldsymbol{0}} B\left\|\boldsymbol{s}-\boldsymbol{s}'\right\|^p \quad \forall \boldsymbol{s} \in R^k \tag{53}$$

where $\|.\|$ is an Euclidean norm (the upper indices 2 and p stand for the power), and $A \geq 0$, $B > 0$, $C > 0$ and $1 \leq p \leq 2$.

Then, the Lagrangian Hedging algorithm using potential function F achieves an external regret of

$$R^T \leq ((TCk + A)/B)^{1/p} \tag{54}$$

where $k \in N$ is a number of available actions and $T \in N$ is a time horizon. For $p = 1$, the bound above is $\mathcal{O}(T)$. However, if we know the time horizon T in advance, we can choose a learning rate $\eta > 0$ and use the potential function

$$F'(\boldsymbol{s}) = F(\eta \boldsymbol{s}) \tag{55}$$

which leads to modification of parameters as $A \to A$, $B \to \eta B$, $C \to \eta^2 C$ and $p \to p$. Specifically, if A, B and C are constants, we can set

$$\eta = \sqrt{\frac{A}{TCk}} \tag{56}$$

which will lead to the bound on external regret of

$$R^T \leq (2/B)\sqrt{ACTk} = \mathcal{O}(\sqrt{T}) \tag{57}$$

The theorem 4.1 is taken from (Gordon, 2005)[4] and its formulation is slightly modified to match the external regret minimization problem. Specifically, we set constants M and D used in (Gordon, 2005) to $M = 1$ and $D = k$, and used Euclidean norm instead of an arbitrary norm (for further discussion, please see sections **8.1.1** and **A.4** of (Gordon, 2005)).

The proof of theorem 4.1 is based on the fact that Lagrangian Hedging performs a gradient descent on $F(\mathbf{s})$. For a gradient descent to work well, the curvature of F must be bounded, which is stated by an equation 52. To link the gradient descent on F to regret minimization, the $F(\mathbf{s})$ must be low for $\mathbf{s}$ with low external regret R^t and high for $\mathbf{s}$ with high regret. This is stated by the requirements 51 and 53.

If we know the time horizon T in advance, we know that $\mathbf{s}$ will stay in $[-T, T]^k$ for the whole interaction (since each element of $\mathbf{r}$ lies in $[-1, 1]$). The properties of F required by the proof then need to be only satisfied in $[-T, T]^k$ and can be violated elsewhere. We will use this observation to select the parameter A as a function of T further on.

We will now provide a regret bound for the approximation of Gradient Bandit Algorithm in a full information setting (with a number of actions $k = 2$):

Theorem 4.2. *Using learning rate η, the algorithm 7 achieves an external regret of*

$$R^T \leq \frac{\sqrt{2}}{4}\eta T + \frac{4\log(T\eta + 4)}{\eta} \tag{58}$$

For $\eta = \Theta\frac{1}{\sqrt{T}}$, the regret bound is asymptotically bounded as

$$R^T \leq \mathcal{O}(\sqrt{T}\log(T)) \tag{59}$$

The proof will be a direct application of Theorem 4.1, which require us to know the function F. The algorithm 7 select its policy as

$$\pi(\mathbf{s}^t) = softmax(\mathbf{w}(\eta\mathbf{s}^t)) = \nabla F(\eta\mathbf{s}^t) \tag{60}$$

where $\mathbf{w}(\mathbf{s})$ is given by the equation

$$\mathbf{w}(\mathbf{s}) = \left[\frac{1}{2}G^{-1}\left(2\left(\mathbf{s}_1^t - \mathbf{s}_2^t\right)\right), -\frac{1}{2}G^{-1}\left(2\left(\mathbf{s}_1^t - \mathbf{s}_2^t\right)\right)\right] \tag{61}$$

with $G(x) = 2x + exp(x) - exp(-x)$. The F is unique up to an integration constant. We choose this constant such that

$$F(\mathbf{0}) = \mathbf{0} \tag{62}$$

Now, the function F is fully specified by equations 60 and 62, and we are ready to prove the Theorem 4.2

Proof. We will begin by showing that F is convex. To do that, we will inspect its Hessian and show that it is positive definite for all $\mathbf{s} \in R^2$. Since $\pi(\mathbf{s})$ is a gradient of F, the Jacobian of $\pi(\mathbf{s})$ is a Hessian of F, that is:

$$\nabla^2 F(\mathbf{s}) = \begin{bmatrix} \frac{\partial \pi_1(\mathbf{s})}{\partial \mathbf{s}_1} & \frac{\partial \pi_1(\mathbf{s})}{\partial \mathbf{s}_2} \\ \frac{\partial \pi_2(\mathbf{s})}{\partial \mathbf{s}_1} & \frac{\partial \pi_2(\mathbf{s})}{\partial \mathbf{s}_2} \end{bmatrix} \tag{63}$$

We have

$$\begin{aligned} \frac{\partial \pi_1(\mathbf{s})}{\partial \mathbf{s}_1} &= softmax_1(w(\mathbf{s}))(1 - softmax_1(w(\mathbf{s})))\frac{\partial w_1(\mathbf{s})}{\partial \mathbf{s}_1} - \\ & softmax_1(w(\mathbf{s}))softmax_2(w(\mathbf{s}))\frac{\partial w_2(\mathbf{s})}{\partial \mathbf{s}_1} > 0 \quad \forall \mathbf{s} \in R^2 \end{aligned} \tag{64}$$

where $w(\mathbf{s})$ is given by the equation 60. The inequality above holds since each element of output of softmax function lies in $(0, 1)$, $w_1(\mathbf{s})$ is strictly increasing in s_1 and $w_2(\mathbf{s})$ is strictly decreasing in s_1.
Similarly, we have

$$\frac{\partial \pi_2(\mathbf{s})}{\partial s_2} > 0 \quad \forall \mathbf{s} \in R^2 \tag{65}$$

Also,

$$\begin{aligned}\frac{\partial \pi_1(\mathbf{s})}{\partial \mathbf{s}_2} &= softmax_1(w(\mathbf{s}))(1 - softmax_1(w(\mathbf{s})))\frac{\partial w_1(\mathbf{s})}{\partial \mathbf{s}_2} - \\ & softmax_1(w(\mathbf{s}))softmax_2(w(\mathbf{s}))\frac{\partial w_2(\mathbf{s})}{\partial \mathbf{s}_2} < 0 \quad \forall \mathbf{s} \in R^2\end{aligned} \tag{66}$$

which implies that also

$$\frac{\partial \pi_2(\mathbf{s})}{\partial \mathbf{s}_1} < 0 \quad \forall \mathbf{s} \in R^2 \tag{67}$$

Therefore, the Hessian $\nabla^2 F(\mathbf{s})$ have positive elements on the diagonal and negative elements elsewhere. Any such matrix is positive definite, which implies the convexity of F.

Next, we need to show that $F(\mathbf{s})$ is non-positive for $\mathbf{s} \leq \mathbf{0}$. This follows directly from the fact that $F(\mathbf{0}) = \mathbf{0}$, $\frac{\partial F(\mathbf{s})}{\partial \mathbf{s}_1} > 0$ and also $\frac{\partial F(\mathbf{s})}{\partial \mathbf{s}_2} > 0$.

We will now bound the second-order directional derivative of F and use that bound to satisfy the condition 52. We have

$$\frac{\partial \pi_1(\mathbf{s})}{\partial \mathbf{s}_1} < \frac{1}{8}, \quad \frac{\partial \pi_2(\mathbf{s})}{\partial \mathbf{s}_2} < \frac{1}{8} \tag{68}$$

which together with equations 66 and 67 implies that the second-order directional derivative of F in any direction is upper-bounded by $\frac{\sqrt{2}}{8}$.

Let $\mathbf{s} \in R^2$ be chosen arbitrarily. We will denote

$$H(\mathbf{d}) = F(\mathbf{s}) + \mathbf{d}\nabla F(\mathbf{s}) + C\left\|\mathbf{d}\right\|^2 \tag{69}$$

We want to show that $F(\mathbf{s}+\mathbf{d}) \leq H(\mathbf{d}) \quad \forall \mathbf{d} \in R^2$. Let $\mathbf{d} \in R^2$ be arbitrary and

$$\phi : R \to R, \quad \phi(t) = F(\mathbf{s} + t\mathbf{d}) \tag{70}$$

We have

$$F(\mathbf{s}+\mathbf{d}) = F(\mathbf{s}) + \int_0^1 \phi'(t)dt = F(\mathbf{s}) + \int_0^1 \left(\phi'(0) + \int_0^t \phi''(z)dz\right)dt \tag{71}$$

Similarly, let

$$\psi : R \to R, \quad \psi(t) = H(t\mathbf{d}) \tag{72}$$

from which we have

$$H(\mathbf{d}) = H(\mathbf{0}) + \int_0^1 \psi'(t)dt = H(\mathbf{0}) + \int_0^1 \left(\psi'(0) + \int_0^t \psi''(z)dz\right)dt \tag{73}$$

It is $H(\mathbf{0}) = F(\mathbf{s})$, and $\phi''(t)$ is a second-order directional derivative of F. Also, $\phi'(0) = \psi'(0)$ and $\psi''(t) = 2C$. Putting $C = \frac{\sqrt{2}}{16}$, we obtain

$$\begin{gathered} F(\mathbf{s}+\mathbf{d}) = F(\mathbf{s}) + \int_0^1 \left(\phi'(0) + \int_0^t \phi''(z)dz\right) dt \leq \\ F(\mathbf{s}) + \int_0^1 \left(\phi'(0) + \int_0^t \frac{\sqrt{2}}{8} \quad dz\right) dt = H(\mathbf{d}) \end{gathered} \tag{74}$$

Now it remains to provide a lower bound for the growth of F. Let $A = 2\log(T\eta + 4)$, $B = 1/2$ and $p = 1$. We will distinguish two cases: when $\mathbf{s} \geq \mathbf{0}$, and otherwise.

For $\mathbf{s} \geq \mathbf{0}$, let $\mathbf{s}' = \mathbf{0}$. We will use a first-order Taylor polynomial of F in $\mathbf{0}$. We have

$$T^{F,\mathbf{0}} : R^2 \to R, \quad T^{F,\mathbf{0}}(\mathbf{x}) = F(\mathbf{0}) + \nabla F(\mathbf{0})\mathbf{x} = \left[\frac{1}{2}, \frac{1}{2}\right]^\intercal \mathbf{x} \tag{75}$$

Since F is convex, it is

$$T^{F,\mathbf{0}}(\mathbf{x}) \leq F(\mathbf{x}) \quad \forall \mathbf{x} \in R^2 \tag{76}$$

We have

$$T^{F,\mathbf{0}}(\mathbf{s}) = \left[\frac{1}{2}, \frac{1}{2}\right]^\intercal \mathbf{s} = \left[\frac{1}{2}, \frac{1}{2}\right]^\intercal \|\mathbf{s}\| \frac{\mathbf{s}}{\|\mathbf{s}\|} \geq \frac{\|\mathbf{s}\|}{2} \tag{77}$$

The last inequality holds since $\frac{\mathbf{s}}{\|\mathbf{s}\|}$ is a unit vector with non-negative elements. From that, we obtain

$$[F(\mathbf{s}) + A]_+ \geq T^{F,0}(\mathbf{s}) + A \geq \frac{1}{2}\|\mathbf{s}\| \tag{78}$$

which makes equation 53 satisfied for the case of $\mathbf{s} \geq \mathbf{0}$.

The second case is that one element of $\mathbf{s}$ is negative. Without loss of generality, we will assume $\mathbf{s}_1 < 0$ and $\mathbf{s}_2 \geq 0$. Let $\mathbf{s}' = [\mathbf{s}_1, 0]$.

Again, we want to use a first-order Taylor polynomial in $\mathbf{s}'$ to provide a lower bound for the $F(\mathbf{s})$. However, now we don't know the value of $F(\mathbf{s}')$, therefore, we will provide a lower bound for $F(\mathbf{s}')$ at first. Let

$$\bar{\pi} : R^2 \to R^2, \quad \bar{\pi}(\mathbf{s}) = softmax(\bar{w}(\mathbf{s})) \tag{79}$$

where

$$\bar{w}(\mathbf{s}) = \left[-\frac{1}{2}\log\left(\frac{\mathbf{s}_2 - \mathbf{s}_1 + 2}{2}\right), \frac{1}{2}\log\left(\frac{\mathbf{s}_2 - \mathbf{s}_1 + 2}{2}\right)\right] \tag{80}$$

It holds that

$$\bar{\pi}_1(\mathbf{s}) \geq \pi_1(\mathbf{s}) \quad \forall \mathbf{s} \in R^2, \mathbf{s}_1 \leq \mathbf{s}_2 \tag{81}$$

Now we can write

$$
\begin{aligned}
F(\mathbf{s}') = F(\mathbf{0}) + \int_0^{\mathbf{s}_1} \pi_1([t,0])dt &\geq F(\mathbf{0}) + \int_0^{\mathbf{s}_1} \bar{\pi}_1([t,0])dt \\
= \int_0^{\mathbf{s}_1} \frac{1}{1+\exp\left(\log\left(\frac{t+2}{2}\right)\right)}dt &= \int_0^{\mathbf{s}_1} \frac{2}{t+4}dt \\
\geq \int_0^{-T\eta} \frac{2}{t+4}dt &= -2\log(T+4)
\end{aligned}
\tag{82}
$$

The last inequality comes from the fact that $\mathbf{s}_1 \geq -T\eta$, since the minimal immediate regret of each action is -1 in each of T rounds.

Using the obtained lower bound for $F(\mathbf{s}')$, we have the lower bound for the first-order Taylor expansion of F at $\mathbf{s}'$:

$$
\begin{aligned}
T^{F,\mathbf{s}'} : R^2 \to R, \quad T^{F,\mathbf{s}'}(\mathbf{x}) &= F(\mathbf{s}') + \nabla F(\mathbf{s}')\mathbf{x} \\
&\geq -2\log(T\eta+4) + \nabla F(\mathbf{s}')\mathbf{x}
\end{aligned}
\tag{83}
$$

Similarly as in the first case, we have

$$
\begin{aligned}
T^{F,\mathbf{s}'}(\mathbf{s}) = F(\mathbf{s}') + \pi(\mathbf{s}')^\intercal(\mathbf{s}-\mathbf{s}') &= F(\mathbf{s}') + \pi(\mathbf{s}')^\intercal \|\mathbf{s}-\mathbf{s}'\| \frac{\mathbf{s}-\mathbf{s}'}{\|\mathbf{s}-\mathbf{s}'\|} \\
= F(\mathbf{s}') + \pi(\mathbf{s}')^\intercal \|\mathbf{s}-\mathbf{s}'\| [0,1] &\geq -2\log(T\eta+4) + \frac{\|\mathbf{s}-\mathbf{s}'\|}{2}
\end{aligned}
\tag{84}
$$

In the last inequality, we also used the fact that $\pi_2(\mathbf{s}') \geq \frac{1}{2}$. Finally, we obtain

$$
[F(\mathbf{s}) + A]_+ \geq T^{F,\mathbf{s}'}(\mathbf{s}) + A \geq \frac{1}{2}\|\mathbf{s}-\mathbf{s}'\| \tag{85}
$$

which makes condition 53 satisfied for the case of $\mathbf{s}_1 < 0, \mathbf{s}_2 \geq 0$. The case of $\mathbf{s}_1 \geq 0, \mathbf{s}_2 < 0$ can be proven analogically, and the case of $\mathbf{s}_1 < 0, \mathbf{s}_2 < 0$ is satisfied trivially by putting $\mathbf{s}' = \mathbf{s}$.

Therefore, the function F satisfied all the assumptions of theorem 4.1 with parameters $A = 2\log(T\eta+4)$, $B = 1/2$, $C = \frac{\sqrt{2}}{16}$ and $p = 1$. With learning rate $\eta > 0$, the external regret of Lagrangian Hedging using function $F'(\mathbf{s}) = F(\eta\mathbf{s})$ is bounded as

$$
R^T \leq \frac{\sqrt{2}}{4}\eta T + \frac{4\log(T\eta+4)}{\eta} \tag{86}
$$

We may now select η to minimize the bound on R^T. Unfortunately, if we take the derivative of the right-hand side of inequality 86 with respect to η, put it equal to 0 and try to solve for η, it show up that such solution cannot be written as a combination of elementary functions (but the optimal value of η for given T may be still found easily using numerical methods). However, we may achieve an optimal asymptotic regret bound of $\mathcal{O}(\sqrt{T}\log(T))$ by choosing $\eta = \Theta\frac{1}{\sqrt{T}}$. For example, we may set

$$\eta = \frac{1}{\sqrt{T}} \tag{87}$$

using which the Lagrangian Hedging achieves a regret

$$R^T \leq \frac{\sqrt{2T}}{4} + \sqrt{T}4\log(\sqrt{T}+4) = \mathcal{O}(\sqrt{T}\log(T)) \tag{88}$$

$\square$

4.4 Evaluating the Quality of the Approximation

In the previous sections, we derive an approximation (algorithm 7) of the Gradient Bandit Algorithm and show that the resulting approximation is a no-regret algorithm with regret bound of $\mathcal{O}(\sqrt{T}\log(T))$. However, we did not evaluate the quality of the approximation yet.

Ideally, we would like to have a bound on the difference between policy chosen by the Gradient Bandit Algorithm and the policy chosen by the approximation algorithm. Such bound could be directly combined with the regret bound on the algorithm 7, resulting in a regret bound on the original Gradient Bandit Algorithm. However, I did not manage to derive such bound, and we will rely only on the empirical evaluation.

To evaluate the similarity between Gradient Bandit Algorithm and its approximation, we run a series of experiments. In each experiment, we generated a sequence of rewards vectors $\{\mathbf{x}^1, ...\mathbf{x}^T\}$, with $\mathbf{x}^t \in R^2$. Rewards are generated from a uniform distribution with values between 0 and 1. Then, we run both algorithms on the sequence of reward vectors and measure the difference in their policies.

For the time horizon T, we used values of $10^4, 10^5, 10^6$ and 10^7. Regarding the value of learning rate η, we know that it should be selected as $\Theta\left(\frac{1}{\sqrt{T}}\right)$. We tried functions $\eta(T) = \frac{c}{\sqrt{T}}$ for $c \in \{\frac{1}{2}, \sqrt{\log(2)}, 1, 2, 5, 10\}$. Specifically, the function $\eta(T) = \sqrt{\frac{\log(2)}{T}}$ corresponds to the optimal learning rate of Hedge algorithm. We also used the value of η given by numerically optimizing the bound in equation 86. The values obtained by the numerical optimization roughly correspond to $\eta(T) = \frac{9}{\sqrt{T}}$ for the values of T mentioned above.

For each combination of the time horizon and learning rate, we run 5 experiments. From each experiment, we record the maximum absolute difference of probability of playing the action 1 assigned by both algorithms (the maximum is taken across all timesteps until the time horizon is reached). The averages of these maxima are reported in table 6 (the averaging is done across these 5 experiments, each with a different sequence of reward vectors). Also, the maximal difference observed in any of the experiments is reported in the square brackets.

We can observe several patterns from the results in table 6.

At first, the approximation seems to be fairly good for the lower values of η. This means that the Gradient Bandit Algorithm selects almost the same policy as the algorithm 7, which is known to be a no-regret algorithm.

At second, the maximal difference of the policies seems not to increase with a longer time horizon, at least not significantly. This is caused by the fact that longer time horizon T implies lower value of the function $\eta(T) = \frac{c}{\sqrt{T}}$. For larger T, the policies have more time to diverge, but the approximation is of higher quality due to the lower value of η, and these two effects seem to counter each other.

At third, the effect of changing the constant c in a function $\eta(T) = \frac{c}{\sqrt{T}}$ seems to have a highly nonlinear effect on the divergence of the policies. Changing the constant c yields more than a proportional change in the approximation quality.

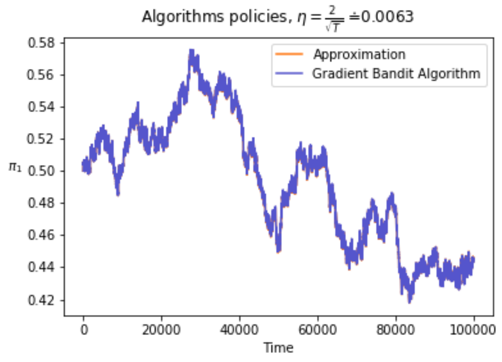
To illustrate the effect of different learning rates on the approximation quality, we plot the policies of both algorithms with different learning rates in figure 1, using time horizon $T = 10^5$. For $\eta = \frac{2}{\sqrt{T}} \doteq 0.0063$, the resulting policies are almost identical, with maximum absolute difference about $1e-5$. (Both policies are plot in the figure, but since they are almost identical, the values of the policy of the approximating algorithm 7 cannot be seen well.) For a learning rate of $\eta = \frac{10}{\sqrt{T}} \doteq 0.0316$, we can see that the policies somehow diverge, with maximum absolute difference about 0.06.

Table 6: Maximal difference between policies of the Gradient Bandit Algorithm and its approximation

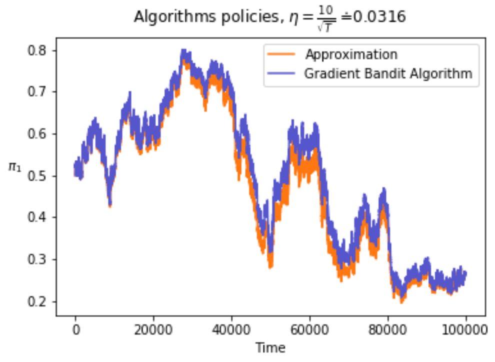
	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$\eta = \frac{1}{2\sqrt{T}}$	3.65e-5 [6.60e-5]	1.86e-5 [5.44e-5]	4.40e-5 [6.58e-5]	5.50e-5 [8.38e-5]
$\eta = \sqrt{\frac{\log(2)}{T}}$	5.32e-5 [1.21e-4]	1.14e-4 [2.17e-4]	1.47e-4 [3.10e-5]	2.46e-4 [3.62e-4]
$\eta = \frac{1}{\sqrt{T}}$	1.55e-4 [4.93e-4]	2.52e-4 [3.14e-4]	2.29e-4 [5.82]	2.00e-4 [6.13e-4]
$\eta = \frac{2}{\sqrt{T}}$	0.0020 [0.0038]	0.0015 [0.0022]	0.0017 [0.0030]	0.0017 [0.0031]
$\eta = \frac{5}{\sqrt{T}}$	0.0214 [0.0283]	0.0165 [0.0241]	0.0146 [0.0299]	0.0192 [0.0294]
$\eta = \frac{10}{\sqrt{T}}$	0.0581 [0.0811]	0.0699 [0.1200]	0.0613 [0.1096]	0.0441 [0.0648]
η optimized numerically	0.0392 [0.0505]	0.0495 [0.0761]	0.0386 [0.1312]	0.0892 [0.1279]

4.5 Further Discussion

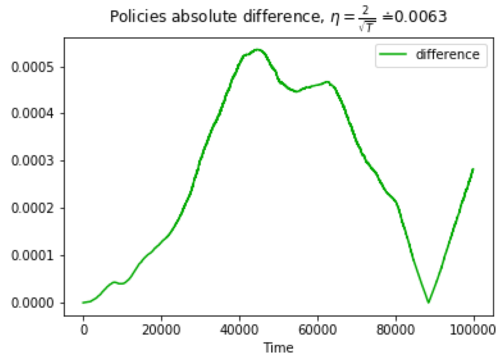
In the previous sections, we have shown that in a simplified setting with two actions and full information, the Gradient Bandit Algorithm may be approximated by a no-regret instance of Lagrangian Hedging, and the quality of the



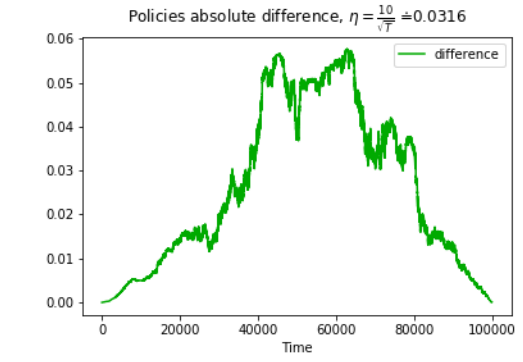
(a) Policies for $\eta \doteq 0.0063$



(b) Policies for $\eta \doteq 0.0316$



(c) Absolute difference of policies for $\eta \doteq 0.0063$



(d) Absolute difference of policies for $\eta \doteq 0.0316$

Figure 1: Comparison of approximation quality for different learning rates

approximation seems to be relatively good, according to the empirical evaluation. We will now discuss how several factors may influence the performance of the Gradient Bandit Algorithm.

We derived the approximation algorithm only for the case of two actions, since the partial differential equation involving a softmax function may be reduced to an ordinary differential equation involving a sigmoid function in that case. As the softmax is a fairly straightforward extension of the sigmoid function to the multivariable setting, we suspect that a similar approximation may be derived for the setting with more than two actions.

However, (Omidshafiei et al. 2019) [9] shown that the Gradient Bandit Algorithm yields significantly worse results for the environment with 3 actions, compared to an environment with only 2 actions. This result is only empirical, and we do not know whether Gradient Bandit Algorithm or its approximation has sub-linear regret bound for more than 2 actions. The regret bounds of both Hedge and Regret Matching algorithms depend on the number of actions k. The regret bound of Hedge scales as $\mathcal{O}(\log(k))$, while the regret bound of Regret Matching scales as $\mathcal{O}(\sqrt{k})$. The higher external regret in an environment with a higher number of available actions is common, but we do not know the exact effect on the regret of the Gradient Bandit Algorithm.

If the Gradient Bandit Algorithm is applied in a bandit setting (which adds the effect of imperfect information), it uses an importance weighted estimator with baseline given as the average of historically obtained rewards. However, we do not know the effect of the baseline on the performance of the algorithm - for example, the proof that the expected regret of Exp3 algorithm is sublinear relies on the fact that Exp3 uses a baseline $b = 1$. Usage of the wrong baseline can hinder the performance of the algorithm.

The difference between policies chosen by Gradient Bandit Algorithm and algorithm 7 was evaluated empirically. For an optimal value of the learning rate of algorithm 7, the maximal difference of policies was relatively large. However, for smaller values of the learning rate, the approximation was much better, with absolute differences of order $1e-4$. To achieve a regret bound $\mathcal{O}(\sqrt{T}\log(T))$ of the algorithm 7, we need to select the learning rate as $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$. We suspect that there exist a certain values of $c \in R$ such that the function $\eta(T) = \frac{c}{\sqrt{T}}$ leads to a good regret bound and simultaneously to a good quality of the approximation, however, I did not manage to prove any bound on the difference between Gradient Bandit Algorithm and algorithm 7 and the analysis is thus based only on the empirical evidence from the simulations performed in the section 4.4.

4.6 Empirical Evaluation of Self-play

To evaluate the quality of strategies obtained by self-play of the Gradient Bandit Algorithm, we run a series of experiments, similarly as in Chapter 3. We used time horizons of $10^4, 10^5, 10^6$ and 10^7, sizes of game 3×3, 5×5 and 10×10 and learning rates $\eta(T) = \dfrac{c}{\sqrt{T}}$ for $c \in \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$. For each combination of these parameters, we generated 1000 random games, and report the average Nash distance from the self-play of the Gradient Bandit Algorithm (the standard deviations are reported in parentheses). We run these experiments both for the current strategy of the algorithm at the end of the self-play and for the strategy averaged across all timesteps.

Overall, the averaged strategies seem to provide a better approximation of the NEQ than current strategies as the end of the self-play. This is in line with the fact that convergence of current strategies to NEQ is a stronger property than the convergence of averaged strategies.

The averaged strategies of the Gradient Bandit Algorithm seem to provide slightly better results than the Action-Value Method, although the difference is not so significant. The results of the Gradient Bandit Algorithm are still much worse than the benchmark of Exp3 algorithm.

Table 7: Self-play of the Gradient Based Algorithm, averaged strategy

Game size 3×3

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$c = 0.1$	0.1296 (0.0395)	0.0758 (0.0169)	0.0355 (0.0085)	0.0145 (0.0053)
$c = 0.2$	0.0977 (0.0251)	0.0475 (0.0121)	0.0199 (0.0066)	0.0092 (0.0058)
$c = 0.5$	0.0537 (0.0142)	0.0249 (0.0069)	0.0113 (0.0059)	0.0045 (0.0033)
$c = 1.0$	0.0325 (0.0096)	0.0148 (0.0035)	0.0072 (0.0047)	0.0031 (0.0039)
$c = 2.0$	0.0222 (0.0066)	0.009 (0.005)	0.0041 (0.0039)	0.0019 (0.0021)
$c = 5.0$	0.0128 (0.0062)	0.0059 (0.0043)	0.0023 (0.0015)	0.0012 (0.002)
$c = 10.0$	0.0117 (0.0076)	0.0038 (0.003)	0.0025 (0.0065)	0.0011 (0.0019)

Game size 5×5

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$c = 0.1$	0.1447 (0.0419)	0.1184 (0.0203)	0.0561 (0.0131)	0.0298 (0.0126)
$c = 0.2$	0.1225 (0.0322)	0.0782 (0.0135)	0.0391 (0.0125)	0.0213 (0.0125)
$c = 0.5$	0.0882 (0.0158)	0.0455 (0.0123)	0.0251 (0.0135)	0.0135 (0.0084)
$c = 1.0$	0.0596 (0.012)	0.034 (0.0143)	0.0165 (0.011)	0.0134 (0.0153)
$c = 2.0$	0.0409 (0.0142)	0.0222 (0.012)	0.0134 (0.0081)	0.0088 (0.007)
$c = 5.0$	0.0276 (0.0131)	0.0137 (0.0088)	0.0109 (0.0097)	0.011 (0.012)
$c = 10.0$	0.027 (0.0173)	0.0144 (0.0112)	0.0131 (0.0132)	0.0106 (0.012)

Game size 10×10

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$c = 0.1$	0.1377 (0.0309)	0.1338 (0.0319)	0.0926 (0.0138)	0.0546 (0.0153)
$c = 0.2$	0.1358 (0.0263)	0.1121 (0.0237)	0.068 (0.0161)	0.0481 (0.0223)
$c = 0.5$	0.1269 (0.0245)	0.0742 (0.0124)	0.0514 (0.0175)	0.0427 (0.027)
$c = 1.0$	0.0934 (0.0137)	0.0588 (0.0209)	0.0398 (0.0193)	0.0446 (0.0287)
$c = 2.0$	0.0697 (0.0165)	0.0501 (0.0254)	0.0421 (0.0248)	0.0439 (0.0251)
$c = 5.0$	0.0526 (0.0215)	0.0465 (0.023)	0.045 (0.0272)	0.0591 (0.0382)
$c = 10.0$	0.0521 (0.0245)	0.0472 (0.0235)	0.0471 (0.0306)	0.0595 (0.0397)

Table 8: Self-play of the Gradient Based Algorithm, current strategy

Game size 3×3

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$c = 0.1$	0.098 (0.0221)	0.0568 (0.0345)	0.067 (0.066)	0.0606 (0.0745)
$c = 0.2$	0.0635 (0.0175)	0.0616 (0.0559)	0.0782 (0.0761)	0.1055 (0.0956)
$c = 0.5$	0.0541 (0.0477)	0.0721 (0.0751)	0.0914 (0.0903)	0.0946 (0.1051)
$c = 1.0$	0.0649 (0.0573)	0.0758 (0.0766)	0.0944 (0.0848)	0.0969 (0.1155)
$c = 2.0$	0.0802 (0.0883)	0.0835 (0.0956)	0.1028 (0.1027)	0.124 (0.1206)
$c = 5.0$	0.0904 (0.0955)	0.0959 (0.1067)	0.1109 (0.1174)	0.1315 (0.1261)
$c = 10.0$	0.1481 (0.1324)	0.1358 (0.1326)	0.1199 (0.1295)	0.1534 (0.14)

Game size 5×5

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$c = 0.1$	0.1328 (0.0318)	0.0903 (0.0259)	0.1244 (0.0713)	0.1708 (0.0898)
$c = 0.2$	0.1036 (0.0188)	0.0976 (0.0471)	0.1304 (0.078)	0.1691 (0.0928)
$c = 0.5$	0.0924 (0.0472)	0.1203 (0.0718)	0.1483 (0.0872)	0.181 (0.1035)
$c = 1.0$	0.1154 (0.0646)	0.1349 (0.0774)	0.1787 (0.1094)	0.1897 (0.1095)
$c = 2.0$	0.1293 (0.0827)	0.169 (0.1095)	0.1867 (0.1031)	0.2226 (0.1124)
$c = 5.0$	0.1767 (0.0943)	0.1871 (0.0989)	0.1917 (0.1099)	0.2205 (0.1162)
$c = 10.0$	0.2016 (0.1219)	0.1962 (0.1114)	0.2293 (0.1239)	0.2319 (0.1235)

Game size 10×10

	$T = 10^4$	$T = 10^5$	$T = 10^6$	$T = 10^7$
$c = 0.1$	0.1345 (0.0289)	0.1179 (0.022)	0.1178 (0.0383)	0.1929 (0.0649)
$c = 0.2$	0.1305 (0.0281)	0.1004 (0.0262)	0.1683 (0.0539)	0.2291 (0.0762)
$c = 0.5$	0.1061 (0.0219)	0.1592 (0.0543)	0.2225 (0.073)	0.2875 (0.0799)
$c = 1.0$	0.1189 (0.0441)	0.2117 (0.0702)	0.2633 (0.0799)	0.3201 (0.0865)
$c = 2.0$	0.1803 (0.068)	0.2313 (0.0791)	0.2858 (0.0822)	0.3374 (0.0795)
$c = 5.0$	0.2372 (0.0691)	0.2694 (0.0741)	0.3258 (0.0794)	0.3418 (0.0781)
$c = 10.0$	0.2956 (0.0766)	0.338 (0.0745)	0.3438 (0.0824)	0.3579 (0.0844)

4.7 Conclusion

In this chapter, we analyzed the properties of the Gradient Bandit Algorithm. We have shown that for a simplified setting with full information and two actions, the Gradient Based Algorithm can be approximated by an instance of Lagrangian Hedging with a regret bound of $\mathcal{O}\left(\sqrt{T}\log(T)\right)$. We empirically evaluated the quality of the approximation, which seems to be reasonably good, at least for low learning rates.

We then run a series of experiments to evaluate the performance of the Gradient Bandit Algorithm in matrix games. While slightly better than the Action-Value Method, the Gradient bandit Algorithm still performs significantly worse than the benchmark of Exp3 algorithm.

5 Sequential Setting

In this section, we will analyze the performance of reinforcement learning algorithms in an extensive-form game with imperfect information. Compared to the single state setting of matrix games, extensive form games add the complexity of multiple states and transitions between them. We will focus on two reinforcement learning algorithms: Q-learning and Policy Gradient.

Q-learning is similar to the Action-Value Method examined in chapter 3 - both algorithms are trying to learn values associated with playing different actions, and select the action with the highest value while mixing this strategy with a uniform distribution over all available actions to ensure exploration.

Policy Gradient's strategy is given by softmax function over all available actions, where the parameters of softmax are updated by a gradient of expected reward. The Policy Gradient uses an importance weighted estimator to estimate the values of actions that were not played.

We will evaluate both algorithms on the game of Leduc Holdem, which is a simplified version of poker.

5.1 Leduc Holdem

Leduc Holdem is a simplified version of poker, which will be used as a testbed in this chapter to evaluate the performance of reinforcement learning algorithms.

The game is played with a deck of 6 cards: 2 × J, 2 × Q and 2 × K. At the beginning of the game, each player receives one private card from the deck. The player does not observe the private card of his opponent. Each player also posts ante of 1\$ to the pot. Then a round of betting occurs, in which each player has an opportunity to bet an additional 1\$ to the pot. Facing a bet from the opponent, each player may either call (putting an equivalent amount of money to the pot) or fold and give up the game. If any of the players fold, the game immediately ends and the remaining player wins the whole pot.

If none of the players fold in the first betting round, one public card is drawn from the deck and revealed to both players. Then the second round of betting occurs, in which players may bet 2\$ to the pot. If none of the players fold, each player reveals his private card, and the player with a stronger card wins the pot. If one player has the same private card as the public card, his card is considered stronger. If none of the players' cards match the public card, the strengths of the cards are evaluated as $K \succ Q \succ J$.

The game has 6121 states, 378 information sets of player 1 and 252 information sets of player 2. As such, the game is significantly more complex than matrix games that we used for experiments in chapters 3 and 4 while being still small enough so Nash Equilibrium may be easily found by a sequence form linear program or by the CFR algorithm.

5.2 Playing Against a Fixed Opponent

To start with, we evaluated the performance of the reinforcement learning algorithm against a fixed opponent. By fixing the opponent's strategy[6], the imperfect information game is effectively turned into an episodic Markov Decision Process.

To obtain a reasonable fixed strategy, we solved the Leduc Holdem using CFR algorithm and get the strategies corresponding to the Nash Equilibrium[7]. The value of the game is a -0.058, thus the player 2 has a slight advantage in the game (this is a common theme in poker-like games, where acting after the opponent allows the player to obtain more information by observing opponent's action, thus gaining an advantage.)

We fixed the strategy of player 2 to be a Nash Equilibrium strategy and use the Q-learning and the Policy Gradient to control the decision making of player 1.

5.2.1 Q-learning Playing Against a Fixed Opponent

We implemented the Q-learning (algorithm 5). We experimented with learning rates $\alpha \in \{0.0001, 0.001, 0.005, 0.01\}$ and exploration parameters $\epsilon \in \{0.05, 0.1, 0.2\}$. For each combination of these parameters, we let the Q-learning to play for $T = 10^6$ episodes. The results are provided in figure 2.

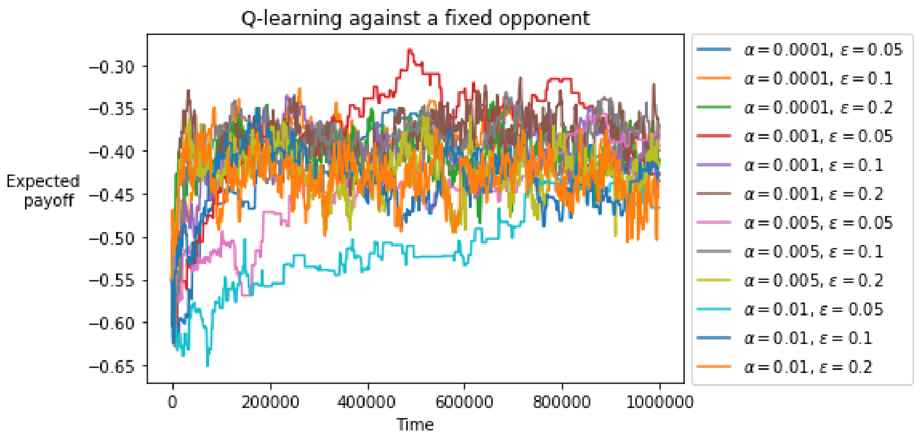


Figure 2: Expected payoff of the Policy Gradient against fixed opponent

The Q-learning struggles to learn the optimal policy, which would yield an expected payoff equal to the value of the game (which is -0.058). One reason why this setting may be hard for Q-learning is the fact that rewards are obtained

[6]Meaning that the strategy is not changed between episodes.

[7]The CFR algorithm provides an only approximate solution, and the resulting strategy profile which we obtained is ϵ-NEQ with ϵ =6.3e-4. As this ϵ is practically very small, we will refer to this solution as a Nash Equilibrium for simplicity.

only in the terminal states. It may be hard to learn the optimal actions' values using the ϵ-greedy exploration strategy since the rewards are usually obtained long time after playing particular action. We may speculate that the usage of more sophisticated exploration strategies would result in a better performance of the Q-learning.

5.2.2 Policy Gradient Playing Against a Fixed Opponent

We implemented the Policy Gradient (algorithm 6). To reduce the variance of the importance weighted estimator, we used a baseline in each state equal to the value of the average utility obtained in sequences that reached this particular state. This mimics the baseline used in the single-state setting by the Gradient Bandit Algorithm. We tried learning rates of $\eta \in \{5, 1, 0.5, 0.01, 0.05, 0.01, 0.005, 0.001\}$ and let the algorithm to play for 10^6 episodes. The results are provided in figure 3.

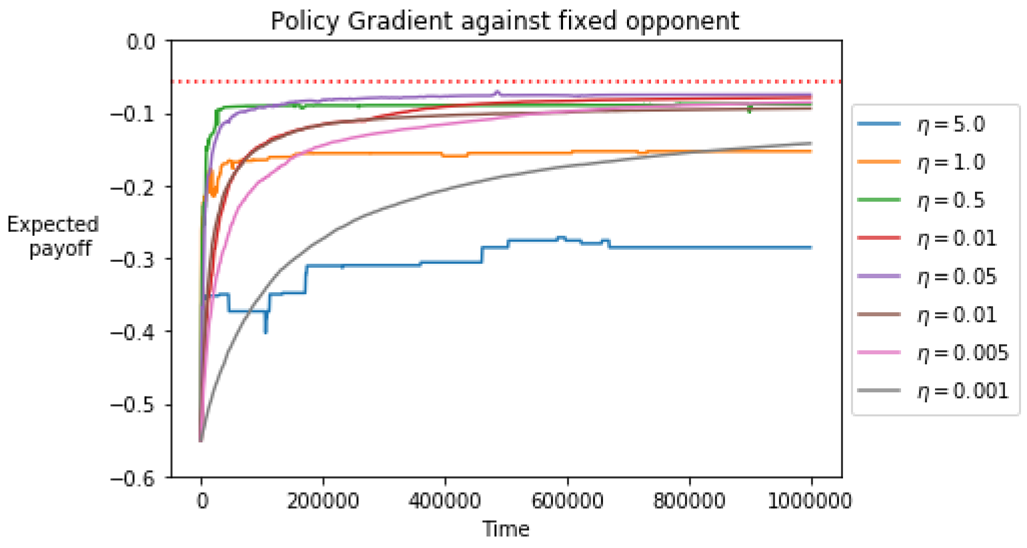


Figure 3: Expected payoff of the Policy Gradient against fixed opponent

The performance of the Policy Gradient depends on the learning rate used. For large learning rates, the algorithm did not explore enough and the learned strategy yields low expected payoff. For small learning rates, the convergence was too slow. However, for the learning rates in between, the Policy Gradient performed well and learned near-optimal strategy (the best possible strategy in this setting would achieve the value of the game, which is -0.058).

5.3 Self-play in Leduc Holdem

We will now evaluate the performance of Q-learning and Policy Gradient in the self-play of Leduc Holdem. For both algorithms, we used the time-averaged strategy instead of the current strategy to measure the quality of the NEQ approximation.

5.3.1 Self-play of the Q-learning

We run the self-play of Q-learning with the same parameters (learning rate and exploration parameter) as against the fixed opponent. The resulting Nash distances are plotted in figure 4. For all combinations of parameters used, the

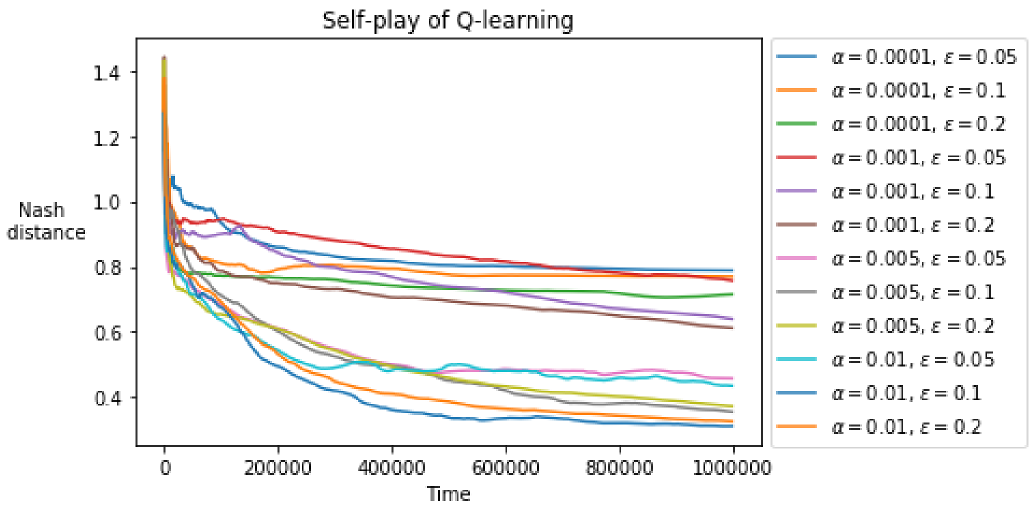


Figure 4: Self-play of the Q-learning in Leduc Holdem

Q-learning produced a strategy profiles with large Nash distance. This is in line with the poor performance of Q-learning against the fixed opponent. We may speculate that usage of different parameters or exploration strategies would yield better results, however, the vanilla version of Q-learning that we used seems not be able to learn a good approximation of NEQ via self-play.

5.3.2 Self-play of the Policy Gradient

We run the Policy Gradient in self-play with the same learning rates as in section 5.2.2. The resulting Nash distances are plotted in figure 5. Even though the Policy Gradient performed well against a fixed opponent, it failed to produce a good approximation of NEQ in a self-play. The results achieved with the best learning rates are comparable to the best results achieved by the Q-learning.

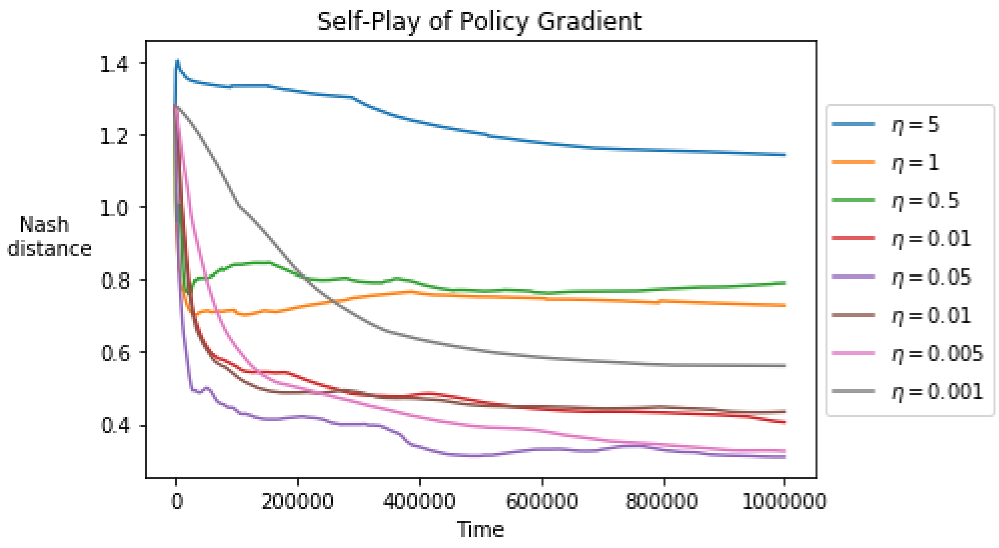


Figure 5: Self-play of the Policy Gradient in Leduc Holdem

5.4 Conclusion

In this chapter, we evaluated the Q-learning and the Policy Gradient on the game of Leduc Holdem.

At first, we run both algorithms against a fixed opponent who was playing a Nash Equilibrium strategy. While Policy Gradient achieves to learn a near-optimal policy when using a suitable learning rate, our implementation of Q-learning failed to achieve a good performance in this setup. We assume that Q-learning would need to use a different set of parameters or exploration strategy in order to learn an optimal policy.

We then evaluated the performance of both algorithms in self-play. The average strategy of Policy Gradient failed to converge close to NEQ, even for the learning rates that lead to a near-optimal policy against a fixed opponent. The averaged strategies failed to converge to NEQ as well, although this is not surprising given the poor performance of Q-learning against a fixed opponent.

6 Conclusion

This thesis examined the properties of RL algorithms applied to the games with imperfect information.

After providing a technical background in chapter 2, we focused on the analysis of the Action-Value Method, which is a simple algorithm for the bandit setting. We have shown that there exists a matrix game in which the updates of the exploration parameter ϵ of the Action-Value Method must follow the GLIE schema, otherwise, it is not possible to guarantee the convergence of average strategies to Nash Equilibrium. However, we do not know whether this is also a sufficient condition for convergence.

We also proved that in a special setting of $2 \times n$ matrix games, the Action-Value Method will learn not to play strictly dominated action, if that action belongs to the players with only 2 actions.

We then examined the empirical performance of the Action-Value Method in the self-play of matrix games. Overall, the performance was significantly worse than the benchmark of Exp3 algorithm.

In chapter 4, we analyzed the Gradient Bandit Algorithm. We have shown that for a special setting with full information and 2 actions, the Gradient Bandit Algorithm can be approximated by an instance of Lagrangian Hedging with asymptotic regret bound of $\mathcal{O}(\sqrt{T}\log(T))$. We then empirically evaluated the equality of the approximation, which seems to be reasonably good, at least for lower learning rates.

Then, the performance of the Gradient Bandit Algorithm in the self-play of matrix games was evaluated. Although it provides slightly better results than the Action-Value Method, the quality of the NEQ approximation was still much worse than the one obtained by the Exp3 algorithm.

At last, we evaluated the performance of Q-learning and Policy Gradient in the game of Leduc Holdem. Although the Policy Gradient was able to learn near-optimal policy against a fixed opponent playing Nash Equilibrium strategy, the self-play of Policy Gradient failed to achieve a good approximation of the NEQ. The Q-learning failed to learn NEQ via self-play as well, however, the implementation of Q-learning that we used struggles to even find an optimal policy against a fixed opponent.

We conclude that imperfect information games seem to be hard to solve via self-play of RL algorithms.

References

[1] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games.* Cambridge university press, 2006.

[2] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[3] Drew Fudenberg, Fudenberg Drew, David K Levine, and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

[4] Geoffrey J Gordon. No-regret algorithms for structured prediction problems. Technical report, Carnegie Mellon University, 2005.

[5] H Jin Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in neural information processing systems*, pages 799–806, 2004.

[6] Tomáš Kocák, Gergely Neu, Michal Valko, and Rémi Munos. Efficient learning by implicit exploration in bandit problems with side observations. In *Advances in Neural Information Processing Systems*, pages 613–621, 2014.

[7] Tor Lattimore and Csaba Szepesvári. Bandit algorithms. *preprint*, 2018.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Rémi Munos, Julien Pérolat, Marc Lanctot, Audrunas Gruslys, Jean-Baptiste Lespiau, and Karl Tuyls. Neural replicator dynamics. *CoRR*, abs/1906.00190, 2019.

[10] OpenAI. Openai five, https://blog.openai.com/openai-five/, 2018.

[11] Martin J Osborne and Ariel Rubinstein. *A course in game theory.* MIT press, 1994.

[12] Yoav Shoham, Kevin Leyton-Brown, et al. Multiagent systems. *Algorithmic, Game-Theoretic, and Logical Foundations*, 2009.

[13] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[14] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000.

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

[16] Kevin Waugh. Abstraction in large extensive games. 2009.

[17] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2008.

Appendix: Description of the Supplementary Code

This section describes the supplementary code which may be found on the attached CD.

The self-play experiments from chapters 3 and 4 are written in Kotlin programming language, and may be found in the folder *Kotlin* on the CD.

The remaining code is written in Python programming language (version 3.7), and is located in the folder *Python* on the CD.

The Monte Carlo estimation from the numerical example in section 3.3 may be found in file *numerical_example_chapter_3.py*. The code used to examine the quality of the approximation of the Gradient Bandit Algorithm is located in file *approximation_quality.py*. The code from chapter 5 is in the file *extensive_game.py*.