A Decision Process Based on Learning Statistical Probability in Game

JeongWon Hahn

The Graduate School
School of Electrical and Electronic Engineering
Yonsei University

A Decision Process Based on Learning Statistical Probability in Game

A Master's Thesis
Submitted to the Department of
School of Electrical and Electronic Engineering
and the Graduate School of Yonsei University
in partial fulfillment of the
requirements for the degree of
Master of Engineering

JeongWon Hahn

December 2017

This certifies that the master's thesis
of JeongWon Hahn is approved.

DaeEun Kim

Thesis Supervisor: DaeEun Kim

Euntai Kim

Euntai Kim

Andrew Teoh Beng Jin

The Graduate School
School of Electrical and Electronic Engineering
Yonsei University

December 2017

Abstract

Artificial intelligence is a field that has been studied for a long time. Especially, artificial intelligence in games is a field that is studied continuously as a field of machine learning because it has certain rules, limited search space and terminate condition is necessarily existed. In addition, game artificial intelligence can be a better field of study because it is possible to extend to real world problems by learning and seeing its performance. Some games, even if the search space is limited, have a wide range of difficulties for humans or machines to search all possibilities, and a typical example is Go. Go has about 10^{360} search spaces, and other games have much fewer search ranges. For example, about 10^{123} for Chess, 10^{58} for Othello, and about 10^{32} for Checker.(Bouzy and Chaslot, 2006) Although the performance of the computer has recently improved, it is impossible to search within a reasonable time using a general-purpose PC. Therefore, it has been suggested that better search algorithms and ways of solving problems by modifying representations of states for solving problems have been proposed. We will take a detailed look at one of these games, Othello, using statistical models.

Statistics are mathematical expressions of the phrase “The past is a mirror of the future”. There is a phrase by Edward Hallet Carr, “History is a continuous process of interaction between the historian and facts, an unending dialogue between the present and the past”. In other words, given the statistical (=past experience), getting good results when performing a specific action relies on a simple learning method that relies on past experience and memory. This statistical method requires a lot of data to learn ‘what actions to take in a particular situation’, and it is difficult to judge similar situations. In recent years, research is under way to allow similar views on similar situations such as Deep Neural Network.

This paper shows that it is possible to learn from these statistical perspectives and shows that using the strategy learned in statistical methods, it is possible to obtain good results without searching the entire node. After learning this, we want to check the possibility of learning by way of viewing through features using the local view rather than using the whole. Lastly, we try to find strategies using reinforcement learning.

Acknowledgements

This thesis would not have been achieved without the support of many people. I firstly thank my supervisor Prof. DaeEun Kim, who always guides me, and Prof. Euntai Kim and Prof. Andrew Teoh Beng Jin, the professors who volunteered to be my thesis committee. They gave me advice, and these brought enormous improvement of my thesis. I also show my gratitude to the members of the Biological Cybernetics Lab. Especially, I give thanks to Seulgee Kim, Wonki Lee, and Changmin Lee who are always beside me, and support me with anything they can. I also give my thanks to my technical supports, Jaehyun Soh, JeongHoon Cheon, Byungmun Kang, Hyeokhyeon Kwon, Euihyun Baek, Chanyung Kim, Jaewoo Jung, and Mincheol Kim. I really thank all of my family for supporting me for the entire of my life. Finally, I will give all these glories to God.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*JeongWon Hahn*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial intelligence has been studied with great interest since when the first computers were developed to make things possible using machines that people cannot do. There are two ways to construct such artificial intelligence: using hand-crafted features and learning through machine learning (Legg and Hutter, 2007). Of these, machine learning is a field of research that has great potential for development. Especially in recent years, learning methods using artificial neural networks among learning algorithms are very popular (LeCun et al., 2015; Silver et al., 2016). In this paper, we will study on the basis of this latest technology. Although the content covered in this paper is not an up-to-date deep learning, let's look at a simpler representation of the beginning and a methodology that can be learned.

1.1 Decision Process and Machine Learning

The field of machine learning through statistics has been actively studied. In this way, the field of machine learning has been studied continuously through various methods. In recent years, there have been a lot of researches on mixing artificial neural networks with machine learning (Zeng et al., 2014; Schmidhuber, 2015). This study achieves this purpose through various methods (Schmidhuber, 2015). In field of game theory, Reinforcement Learning are used that a kind of learning methods (Riedmiller et al., 2009; Riedmiller, 2005). In particular, the use of such reinforcement learning and artificial intelligence has been studied, The success of Deep Q-Network from google deepmind (Mnih et al., 2015) was enough to enthuse many people. Some of them have succeeded in the Atari game, an old video game, using this DQN (Mnih et al., 2013). Especially Deep Learning has emerged very much recently because it has won the Go, which is considered to be difficult to predict its behavior because it requires too much computation amount (Silver et al., 2016).

This seems to be a recent study, but all of these studies based on probability have been studied for a long time. The decision process is to make more valuable choices based on probability(empirical). Decision is also used to classify images, to classify emotions, and to estimate the value of choice. For example, in the problem that moving a disc to a place while playing a game, selecting a good spot for each spot value can also be regarded as a decision problem. Here, Also many people has studied these kind of decision through neural networks as well as these probabilities (Bishop, 1995). There is a field of study on game theory which has studied the extension of search through Bayes probability (He et al., 2008). There is also a way to solve the chess problem by using the probability of such a Bayes (Fernández and Salmerón, 2008). Solving the probabilistic problem is a field that is been studied a lot, and the whole principle is still unknown.

As for machine learning, there is the word learning. It means commit to memory. In other words, machine learning means that the machine remembers past experiences (Bishop, 2006). By doing so, we can empirically determine the situation by acting on the machine to learn the past (Kodratoff, 2014). It is possible to find out how to get the agent to empirically know the value of a specific action to an agent, and then get the better results by performing the same task. The purpose of machine learning is to train the machine to solve classification or decision problems, and to experience its value in order to make better choices in the same situation. Until now, there has been a problem that this machine learning can only judge the same experience, but the recent developments seem to solve the problem of judging whether experiences or situations are similar (Chen and Asch, 2017).

Particularly in game theory, there are cases where the decision process is used to solve the problem. In the case of the Othello game, several programs have been developed since the 1980s (Rosenbloom, 1982; Lee and Mahajan, 1990; Buro, 1995, 1997b, 2003). In addition, it was not a probabilistic solution but a genetic algorithm (Alliot and Durand, 1995), and artificial neural network (Leouski, 1995; Abdelbar and Tagliarini, 1998). There is also an effort to solve problems by using recent reinforcement learning and artificial neural network together (Van Der Ree and Wiering, 2013).

1.2 Motivation and Objectives

Google deepmind's efforts showed enough potential for future development, and therefore, we have tried to do research from the basics. The first time we started this research, we began to think about how to find a better algorithm through evolutionary computation in the place where we worked previously. In the meantime, we thought that the method of solving the problem by giving proper gain to several parameters with the concept of weight is similar to finding a better algorithm. From there we saw Othello games using these weighted pieces counters. This board game has the characteristic that all units(=discs) have the same value and each position can be expressed by weight. We also felt that the process of looking at a situation in this board game and making the best choice in that situation could be the same as recognizing situations in other studies in the future and making better choices in specific situations. Therefore, we began the study above, and what we want to achieve through this study are as follows.

First, we want to study machine learning. Learning the whole of machine learning is too broad, but we have been studying relevant previous research and how they were doing machine learning. We have also attempted to solve the problem in the previously known method in a more intuitive and easy way. This attempt was made from a stochastic approach. Expression of probability using conditional probability and learning method was studied. Next, we studied the possibility of learning the same with various expressions. We have tried to express various situations from the same perspectives, and we have studied this local feature extraction because we cannot know global information in the real world. Finally, we have studied reinforcement learning, which is the basis of recent deep learning methods. This was done in order to learn more about future research as a step to learn the basics in order to study deep learning which can solve various problems recently.

1.3 Organization of the dissertation

In this chapter, we introduced the motivation of research, the concept of pattern recognition and machine learning, and the objectives of what we want to do, which we propose and investigate in this paper.

The next chapter, chapter 2, we will introduce the probability-based learning methods, the way the existing game artificial intelligence has been constructed, and how to search a value of the situation and solve problems. And also we will show recent work for game of Othello AI programs.

In chapter 3, we will express a system as the decision problem and describe solving this with Bayes probability. Through the example problem of Othello, we have defined the solution process as a decision problem and studied how to make a choice in a probabilistic way. This method has tried to solve the problem more easily than known methods.

In chapter 4, we have done research that looks at problems through various perspectives. We thought in the real world, the entire information of a system is unknown. In this respect, we tried to solve the problem by evaluating the value of the situation using only a local view, and continuing with the choice of maximizing the evaluated value.

In chapter 5, we will show what we studied about reinforcement learning, which is a basis of Deep Q-Network(DQN). In order to catch up with recent research, we examined the reinforcement learning types, TD-Learning and Q-Learning, and applied this learning method to Othello. This learning method is the basic idea of Deep Learning method in recent years, and we consider it as a step to build up from the basics before further study of future research.

The last chapter, chapter 6, have conclusions of our works. We show our importance of our works, and possibility to improve. We also conclude shortage, and advantage of our works.

Chapter 2

Background

In the early 1950s, research on artificial intelligence began to try to imitate human intelligence. The brain is an endless field of research and a subject of inquiry. One way of doing this is by using statistics, which was one of several ways to make the current choices based on historical information. Thus, research has been conducted on how statistics can be used in a better way, and humans conquered various games one by one. The game has rules, and there is a limited search area. It is also possible to react variously according to the opponent's behaviour through the interaction between the opponent and the agent. Because of this, it has been the subject of research, and in recent years many games have been played. In particular, there has been much progress in the 2-player board games, the range of search is so wide that the computer is victorious in Go, which has been considered impossible for computers (Bouzy and Cazenave, 2001; Baudiš and Gailly, 2011; Silver et al., 2016). We are going to look at some of the areas of AI which are more likely to experience evolution like this. Especially, we would like to study more about artificial intelligence through Othello, which is one of the 2-player board games, because the game is not too monotonous like tic-tac-toe and not too complex like Chess. Therefore, we will study using the statistics created during the Othello game and examine the progress.

2.1 Search Algorithm

Search algorithm is a very important way to organize a search tree in a game AI. The most classic artificial intelligence that a person thinks while playing a game is to use the information “When I act like this, what will my opponent behave?” When constructing the tree in this way, there will be a way to find the number of all cases, and there will be a way to selectively browse. However, people do not think about about all of the cases and instead, people think about the more valuable case. Then you will need a way to evaluate these parts as ‘good’ nodes, and those methods will be the alpha-beta pruning method and the Monte-Carlo Tree Search(MCTS) method introduced below.

2.1.1 Minimax Algorithm

When we play games, we want to have a greater future value than the present value of our actions. This process is the Search Algorithm, and we will first look at the Minimax algorithm, which is the basic of it. As you can see from the word, Minimax(sometimes MinMax or MM (Barua, 2013)) algorithm can be used to determine the value of the future by assuming that the agent chooses the best action among possible choices for the agent and chooses the best action among the actions that are allowed for opponents. The name is Minimax algorithm because the opponent’s best action is the worst (min) action in my position. The Minimax algorithm is a formula from a 2-player zero-sum game (Narahari, 2014). This algorithm is a search method that considers future values by searching more depth. In general games, this minimax value takes the maximum value from the position of one player, which can be expressed formally as follows (Myerson, 2013).

$$V(s_t) = \min_{a_{-i} \in A_{t+(2n+1)}} \max_{a_i \in A_{t+2n}} V(s_{t+1} | a_i, a_{-i}) \tag{2.1}$$

Where:

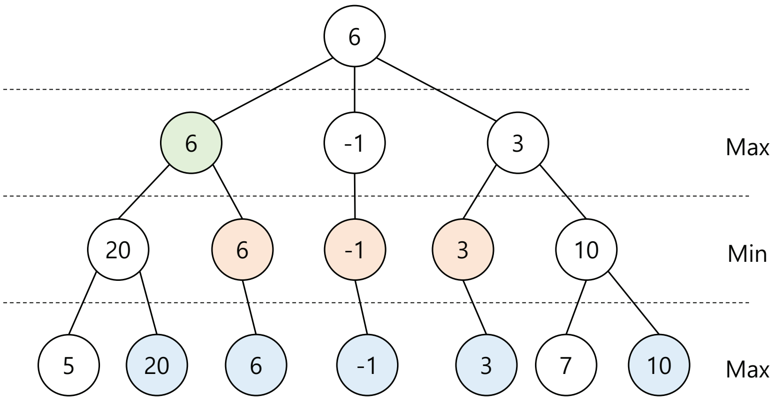


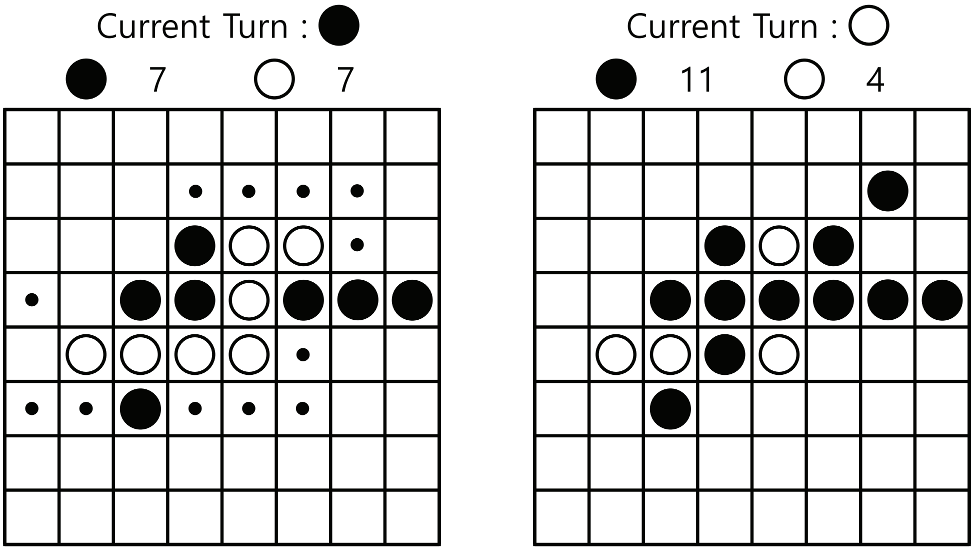
Figure 2.1: Minimax Tree Example. Search pre-defined depth, and backtrack to the top of node, choose min value or max value each depth.

- $\mathbf{S}_t$ is current state.
- $\mathbf{V}(s_t)$ is the value function of current state.
- $\mathbf{V}(s_{t+1}|a_i, a_{-i})$ is the value function of next state when take a action which is a_i and a_{-i}
- $\mathbf{A}$ is all the action set that player can choose.
- $\mathbf{a}_i$ is the action taken by player.
- $\mathbf{a}_{-i}$ is the action taken by opponent.

If we look at figure 2.1 above, we can select the maximum value (from equation 2.1, it is a_i) among the selectable values in the s_{t+1} the minimum value of the next turn (the maximum value of the opponent, from equation 2.1, it is a_{-i}) is selected. As described above, the Minimax algorithm selects the largest of the future value by performing the Min-Max step a predetermined number of times.

Specifically, we call it as greedy algorithm when just 0 depth search in this minimax algorithm. The greedy algorithm is a method of choosing the most valuable thing in every moment without caring about the opponent.

For example, greedy method in figure 2.2, it shows the best choice which is placed on (2,7) because it can flip three discs when placed. So using greedy method it chose



(a)

(b)

Figure 2.2: Greedy (0 depth min-max) Example in othello. (a) Before place a disc, (b) After choosing to place (2,7) in (a), from (a) it has 12 candidates, maximum flips are 3. the other candidates can only flip 1 or 2.

(2,7). Using Minimax, however, it has more valuable choice in future. Therefore, It it important to search more in depth and make good strategy in this kind of games.

2.1.2 $\alpha - \beta$ Pruning

Minimax may appear to be a good search algorithm for searching multiple depths, but checking all cases requires too much computation time. So, to overcome these limitations, people started looking for algorithms to search in better (more valuable) direction. That is $\alpha - \beta$ pruning algorithm. In $\alpha - \beta$ pruning algorithm, It generates search tree in similar ways to minimax, but it searches more depths when it increases the value. In the minimax algorithm, if it searches all cases, it has a lot of searching time and space. For example, if it has 3 choices per 1 depth, it needs 3^{depth} cases to search. This is due to the exponential increase in depth as the depth increases, which makes it impossible to search at polynomial time.

When expanding a tree in minimax, as shown in figure 2.3, when we evaluate at the

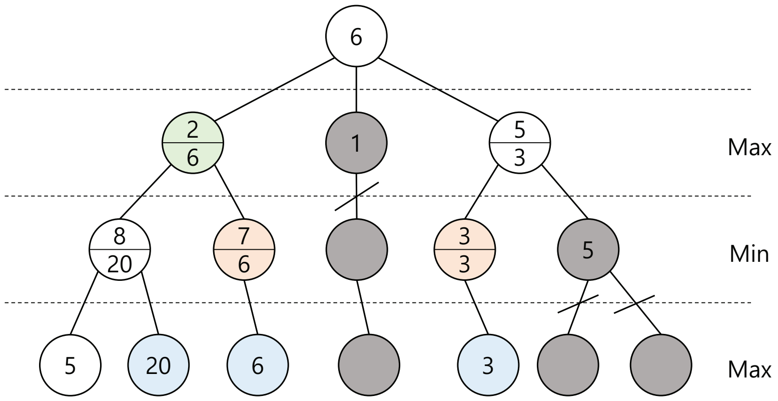


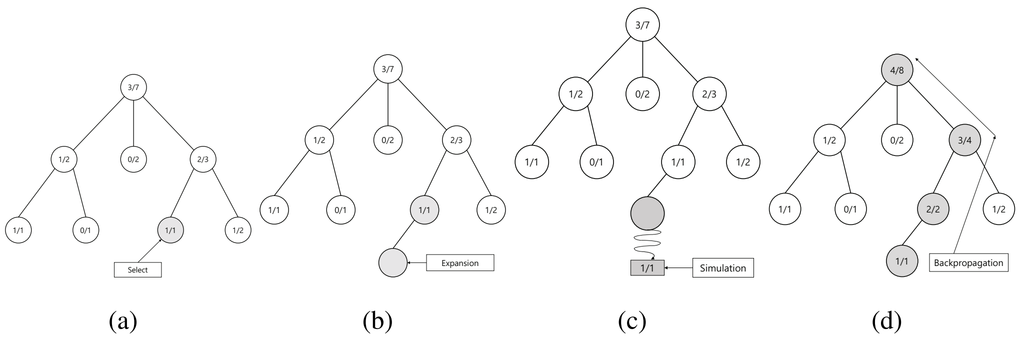
Figure 2.3: Example of $\alpha - \beta$ pruning. If you look at the value of the node according to the max or min rule at the same depth, you extend the search only in directions that increase or decrease from the previous one. The gray color node is the pruning node in the search.

same depth (= step), we extend the search only in the direction in which the value increases or decreases further (Marsland, 1986). This reduces the search for unnecessary nodes and improves the search speed, allowing for more cases to be searched. When determining the value of each position in order, the min-max step will not proceed any further if there is no better result (Heineman et al., 2016). Using the alpha-beta pruning algorithm, the time required is reduced by square root ($O(b^d)$ becomes $O(\sqrt{b^d})$). but, in average it becomes $O(b^{\frac{3d}{4}})$ (McCarthy, 2006).

2.1.3 Monte-Carlo Tree Search Algorithm

The origin of Monte-Carlo Tree Search was Monte-Carlo Evalutaions(MCEs). It was originally introduced for 2-player board games like Tic-Tac-Toe, Othello, and Chess (Chaslot, 2010). This method used a lot of game like Go (Silver et al., 2016), Backgammon (Tesauro and Galperin, 1996), Chess (Ciancarini and Favini, 2010) and Settlers of Catan (Szita et al., 2009).

The most basic version of MCEs works in the following ways: Simulate value from position P. Simulation action is randomly selected in self-play until the end of the



(a) (b) (c) (d)

Figure 2.4: 4 steps of MCTS (a) Selection, (b) Expansion, (c) Simulation, (d) Backpropagation

game. Each simulation we give as output a payoff value. So using the payoff value, we carry out the next step. It is the basic version of MCEs. But MCEs has a problem that the randomly selected step has limitation. So the next step that they take is the Monte-Carlo Tree Search(MCTS). MCTS is an improved model of minimax algorithm. It is the best-first search method that does not require a positional evaluation function. The best-first search means the MCTS doesn't fully search, It means it doesn't search all nodes. It searches the best one which it can take. So it has 4 steps to search, 1. Selection, 2. Expansion, 3. Simulation, 4. Backpropagation, to reduce the nodes of search (Brügmann, 1993).

First, Selection is a process of determining in which direction to search based on the present. Second, Expansion is an additional search for the selected direction. Third, Simulation : the path selected in the extension is processed in a random manner until it is finished. If there is no rule to get the simulation result, sometime it will search all the node until end of the game. Last, Back-propagation is a process of updating all the nodes that belong to the result. Using these 4 steps, a search tree is built until predefined computational quantity is met. The quantity is time, memory, or iteration. And then it returns the action (in some games it means position) what agent can be selected in the game.

So, in this time, we would like to show you the search algorithm, not the learning method described above. The search method using mini-max algorithm has a problem that the time required expands exponentially because it searches the entire section. Normally, it increases in the form of a square. It can't be solved within polynomial time. Instead it takes N^M time, an example of Mini-max algorithm that searches 4 Depth, assuming that we can choose 8 position to place, requires time of $Const \times 8^4$. Also, even if we use the minimax algorithm to search up to 4 depth, you can't be sure of winning. This uncertainty makes problem more complicated. Therefore, most games, not just Othello, encounter the problem of size limitations of this search and need a better algorithm to determine the behaviour within a fixed time. One of those algorithm is Monte-Carlo Tree Search algorithm.

The MCTS algorithm can be described as an optional full search. Of course, the MCTS algorithm will also explore the number of all cases if there is plenty of time, but in most cases, the MCTS is applied to avoid it. To apply the MCTS as described above(Chapter 2.2), first we need to define the values that the MCTS should have. The number of explore, selection algorithm, how to simulate the result. Using these items, we can construct MCTS, and execute 4 steps that include Selection, Expansion, Simulation, and Back-propagation. At this time, the result of the MCTS depends on the simulation and back-propagates only the result winning or losing. This is because UCT method is applied which has a problem that the node which visited in the similar situation is continuously repeatedly visited. The UCT algorithm does not only rely on existing wins and losses but also the time it takes to search the node that has not been visited yet, and is closer to BFS even though it may give a slightly worse result. However, it is the mostly used algorithm among various MCTS algorithms since it does not lose the function of MCTS, but is biased toward DFS. So selection function most commonly used is called UCT(Upper Confidence Bounds for Trees). The UCT(Upper Confidence Bounds for Trees) algorithm reflects time consumption of trees that have been visited. It shows parts of $\sqrt{\frac{lnt}{n_i}}$. And the UCT algorithm goes as follows (Kocsis and Szepesvári, 2006) :

$$\frac{w_i}{n_i} + c\sqrt{\frac{lnt}{n_i}} \tag{2.2}$$

Where:

- w_i stands for the number of winnings after the i-th move.
- n_i stands for the number of simulations after the i-th move.
- c is the exploration parameter, $sqrt(2)$ is a good first guess for this number, but in practice, you'll have to tune it experimentally.
- t stands for the total number of simulations, equal to the sum of all n_i. Or, another way to think about this is that it's the n_i of the parent node.

So using this algorithm, called UCT(Upper Confidence Bounds for Trees), pseudocode is as follows(Algorithm 1, Chaslot et al. (2008)) :

Algorithm 1 Pseudo code for select rule for MCTS based UCT

```
function UCT_MCTS($TreeRoot$)
    Let $Child[1 \ldots N]$ is arrays that store UCT score of all the children.
    Let N is the number of children.
    for $i = 1toN$ do
        $Child[i] = ComputeUCT(TreeRoot.child)$
    end for
    Select max index of $Child[1 \ldots N]$.
end function
function ComputeUCT($Node$)
    if $Node.Visit$ is false then
        return $ConstA$.
    end if
    return $\frac{Node.Wins}{Node.Visits} + ConstB\frac{\log Root.Visits}{Node.Visits}$
end function
```

2.2 Learning Methods

So far we have thought about how to build search trees and how to evaluate the value of certain moments. From now on, we want to look a little about how learning will be possible. Statistics are mathematical expressions of the phrase “The past is a mirror of the future”. And also there is a phrase by Edward Hallet Carr, “History is a continuous process of interaction between the historian and facts, an unending dialogue between the present and the past”. In other words, given the statistical (=past experience), getting good results when performing a specific action is a learning method that relies on past experience and memory, which is a simple method of learning. This statistical method requires a lot of data to learn what actions to take in a particular situation, and it is difficult to judge similar situations. In recent years, research is under way to allow similar views on similar situations such as Deep Neural Network. There are various ways of learning. There is a simple way to create a random variable using statistics and to evaluate it as a random variable, and there is also a method of learning by evaluating the state by classifying the good and bad of the current state (Sutton and Barto, 1998). Beyond this kind of learning, there is also a way to evaluate what action is good in a particular state, and there is also a way to learn how to express this state. Among these methods, we will start from the viewpoint of the simplest statistics and study various learning methods.

2.2.1 Conditional Probability & Bayes’ theorem

Probability is the possibility of event A occurring. The conditional probability is the probability that event B occurs when event A occurs.

In probability theory, conditional probability is the probability of an event given that(by assumption or evidence) another event has occurred (Ross, 2014). The conditional probability is when event A occurs, event B occurs at the same time. In other words, it is formally expressed form of the probability of event B occurring when there is a certain observation of event A occurring. “The conditional probability of A given B” or “the probability of A under the condition B” is usually written as $P(A|B)$.

Using this conditional probability, we can try to learn a value of specific state. In game theory, to understand what action led the agent to victory, we need to see what action the agent did when he won. The words that “when he won.” refers to the conditional probability. If so, then this is expressed as conditional probability. When agent does action ‘A’, it can expressed by “The probability of winning when action A is given”, and it looks like $P(Win|do\,A)$

This conditional probability means the probability of a hypothesis being expressed when a specific action is performed (Grinstead and Snell, 2012).

In probability theory and statistics, Bayes’ theorem explains the probability of an event based on prior knowledge of the conditions that can be associated with the event. In game of Othello, for example, positions can be used to estimate the likelihood of winning more accurately compared to estimating the likelihood that win is related to a position and Bayes’ theorem gives the probability of having victory without knowing its position. Also, the formula of Bayes’ theorem, which is the most basic, is as follows.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.3}$$

where A and B are events and $P(B) \neq 0$ and $P(B)$ can represent as follows.

$$P(B) = \sum_{i=1}^{n} P(A_i)P(B|A_i) \tag{2.4}$$

For example, in table 2.1 $P(A|B)$ is calculated as follows.

$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{i}{i+k} \times \frac{i+k}{i+j+k+l} \div \frac{i+j}{i+j+k+l} = \frac{i}{i+j}$. If you know the overall probabil-

relative size	B	B^c
A	i	k
A^c	j	l

Table 2.1: To example of Bayes' theorem, it is relative size of event A or B occurring.

ity and conditional probability of each event in this way, you can change the condition before and after the occurrence of the event. This formula is Bayes' theorem, so you can try different things like changing the likelihood and a posteriori.

2.2.2 Maximum Likelihood & Maximum A Posteriori

There is a problem that this probability (= statistic) must be learned in order to perform machine learning using probability. If you think about the process of machine learning, you will learn in the following order. First, input will be training data and this will be used to create a function to find the desired hypothesis using learning algorithm A. This hypothesis can be regarded as a random variable and a process of finding such a probability density as such. For example, assume that the data is spread over a Gaussian distribution. At this time, it is possible to judge whether a part of a random sample belongs to a specific section, and to estimate a percentage of a general data belonging to that section. Learning these probabilities is an example of machine learning. If so, we can see the learning process as a process of finding a probability density function parameter.

Maximum Likelihood Estimation (MLE) is a method of estimating the parameters of a random variable, and is a method of parameter estimation based solely on a given Observation or data. As a simple example, if you want to find P(Front) when you get a front with a probability of P(Front) and you throw a coin with the backside probability of P(Back), you can get this $P(Front) = \frac{the\ number\ of\ fronts}{total\ number\ of\ attempts}$.

So let's look at another example that distinguishes whether a person is a man or a woman when a shadow is shown. Our observation is a shadow, and the class that we need to estimate is whether it is a man or woman. If you express it using Likelihood, it becomes $P(Shadow|Man)$ or $P(Shadow|Woman)$. In other words, Likelihood can be described as $P(Observation|Class)$. But here we may consider posterior, not just likelihood. Following Bayes' theorem, posterior can represented as equation 2.5 (Grinstead and Snell, 2012).

$$posterior = \frac{prior \times likelihood}{evidence} \tag{2.5}$$

Expressing this again with probability using equation 2.3, it becomes as follows (Grinstead and Snell, 2012) :

$$P(Class|Observation) = \frac{P(Observation) \times P(Observation|Class)}{P(Class)} \tag{2.6}$$

From the example above, posterior is $P(Man|Shadow)$ or $P(Woman|Shadow)$. Thus, a posterior is $P(Class|Observation)$ in general expression. At first glance, it is simply a pun, but it has a great meaning. In the above example, the ML method refers to the probability of 'man is shadow', and MAP refers to the probability of 'shadow is man'. Although the ML method may seem more intuitive, the ML method is a method of finding an observation when a class is given, and the MAP is a method of finding a class when an observation is given. Therefore, the MAP method is a better way to find the class.

2.2.3 Naïve Bayes

So far, we have looked at the probability approach for machine learning. From now on, we would like to talk about Naive Bayes. The Naïve Bayes classification method is known as the supervised learning method in machine learning. This method basically uses Bayes' Theorem to classify, and the Naïve Bayes classification assume that all the events are independent. It means each event does not affect each other, and Equation 2.8 is an estimated likelihood with naïve bayes. Also, the method of estimating likelihood by using the single evidence is shown by Equation 2.7. It is another representation of Equation 2.6.

$$P(H_i|E) = \frac{P(E|H_i) \times P(H_i)}{\sum_{k=1}^{m} P(E|H_k) \times P(H_k)} \tag{2.7}$$

But now we are going to look at the probability by using two or more evidence using Naïve Bayes classification (Zhang, 2004).

$$\begin{aligned} P(H_i|E_1, E_2, \ldots, E_n) &= \frac{P(E_1, E_2, \ldots, E_n|H_i) \times P(H_i)}{\sum_{k=1}^{m} P(E_1, E_2, \ldots, E_n|H_k) \times P(H_k)} \\ &= \frac{P(E_1|H_i) \times P(E_2|H_i) \times \cdots \times P(E_n|H_i) \times P(H_i)}{\sum_{k=1}^{m} P(E_1|H_k) \times P(E_2|H_k) \times \cdots \times P(E_n|H_k) \times P(H_k)} \end{aligned} \tag{2.8}$$

Using Equation 2.8, we can classify by using multiple evidence after learning by using a single evidence.

For example, when we want to distinguish spam mails, an observation value becomes the keyword and a class becomes whether it is spam mails or not. Here is an example: First, there are 60 mails in total, of which 45 are spam mails. 50 of them contain the word ‘news’, of which 40 are classified as spam mails, 30 of them contain the word ‘job’, 20 of which are classified as spam mails. Expressing it formally it becomes $P(spam) = \frac{45}{60}$, $P(keyword = NEWS) = \frac{50}{60}$, $P(keyword = job) = \frac{30}{60}$. And conditional probability will be $P(spam|keyword = NEWS) = \frac{40}{50}$ $P(spam|keyword = job) = \frac{20}{30}$. If so, when news and job are together, the probability of spam mails is as follows. $P(spam|news, job) = \frac{P(news|spam) \times P(job|spam) \times P(spam)}{P(news|spam) \times P(job|spam) \times P(spam) + P(news|spam^c) \times P(job|spam^c) \times P(spam^c)}$ $\frac{\frac{40}{45} \times \frac{20}{45} \times \frac{45}{60}}{\frac{40}{45} \times \frac{20}{45} \times \frac{45}{60} + \frac{10}{15} \times \frac{10}{15} \times \frac{15}{60}} = 72\%$. Thus, if both the keywords ‘news’ and ‘job’ are present, the probability of spam mail can be calculated as 72 %.

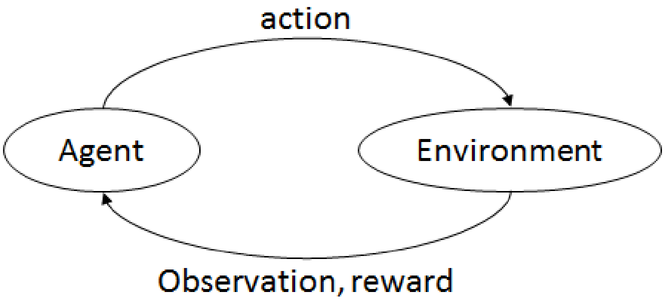


Figure 2.5: basic reinforcement learning model

2.2.4 Reinforcement Learning

The last one is reinforcement learning. Reinforcement learning is a concept introduced in the book of Barto (1997), an area of active research in the artificial intelligence field. Reinforcement learning is a very intuitive learning method. Reinforcement learning is a learning method in which there is a lot of experience existing in the absence of correct answers, thus conducting rewarded behaviors and better behaviors from past experiences. In this reinforcement learning, the definition of action(behavior), environment, and reward(feedback) is very important when modeling the system. It is like training a puppy with an anchovy (reward) is given when it does the right thing and punish is given when it does something wrong. The learner is not told which actions to take, as in most forms of machine learning, but instead discovers which action yields the most rewards by trying them (Barto, 1997).

In this way, Reinforcement Learning allows agents to learn information while interacting with the environment, and through this, searches the optimal policy. Using these reinforcement learning methods, Google's Deepmind has recently learned to play Atari games to get better results than traditional AI (Mnih et al., 2013). This reinforcement learning is modelled on several assumptions. First, Reward Hypothesis: All goals can be shown to maximize compensation. An intuitive example is that when a person acts, he or she chooses what action will benefit me and maximize the benefits. If you are to choose from a job with a monthly salary of 1 million won and 5 million won, you will choose the latter. These thoughts are the reward hypothesis that maximizes compensation. This reward hypothesis is expressed in mathematical form as follows (Sutton and Barto, 1998) : The state-value-function represent merits or demerits in current state (Bellman, 1952).

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \tag{2.9}$$

The value of any state $s(V_\pi(s), S_t = s)$ that is expected when following strategy π at some time t (E_π : Mean when nondeterministic) is express as the sum $(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots)$ of future rewards. Here, γ is a discount factor. Normally, γ take 0 to 1. If in case that $\gamma = 0$, it means that $(t+1)$ compensation is only considered next time. In this case, the advantage here is that you can quickly determine the best behavior. And also when $\gamma = 1$, it considers all the future value, so it can't determine the best behavior, but it can determine the best one of the agent knows. So, depending on the problem, γ needs to be optimized. The method of learning according to the formula introduced above is called TD-Learning, and the expression of TD-Learning is as follows (Sutton and Barto, 1998).

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s)) = (1 - \alpha)V(s) + \alpha(r + \gamma V(s')) \tag{2.10}$$

However, since the state-value-function does not consider into account the transitions from different states, it has a problem that if a bad reward exists, all the surrounding states are also affected by the bad reward. Therefore, if an action-value-function is constructed by adding an action based on a state, the state transition can be expressed more accurately by the action. And the represent of action-value-function is mathematically demonstrated as follows (Maei et al., 2010) :

$$Q_\pi(s,a) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a] \tag{2.11}$$

This equation looks as if $V_\pi(s)$ adds action $A_t = a$.

$$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})) \tag{2.12}$$

Based on these expressions, the Q-learning algorithm (Action-value-function learning algorithm) can be written as:

1. Select an action a_t following policy π and execute it
2. Check immediate reward value r_t
3. Observe a next state s_{t+1}
4. Update the Q function table for $Q(s_t, a_t)$ using equation 2.12.
5. Increase t

Repeat all these sequences above.

In addition to the development of Q-learning, many game AIs have been created by incorporating the concept of deep learning of neural networks Watkins and Dayan (1992).

2.3 Game of Othello

We chose the Othello game to test the learning method described above, so in this section we would like to briefly describe what the othello game is. This section gives a brief description of the nature of the game.

Othello, also known as Reversi is a 2-player board game, played on an 8 x 8 board (Moriarty and Miikkulainen, 1995). It is played with 64 identical black or white pieces(=discs). Initially, it begins with four discs placed center diagonally (figure 2.6 (a)). When a disc is placed, all discs in different color between a player's discs can be flipped. The discs colored white on one side is black on the other, so when you flip it, it then turns into a different color. Also, due to the nature of flipping of these

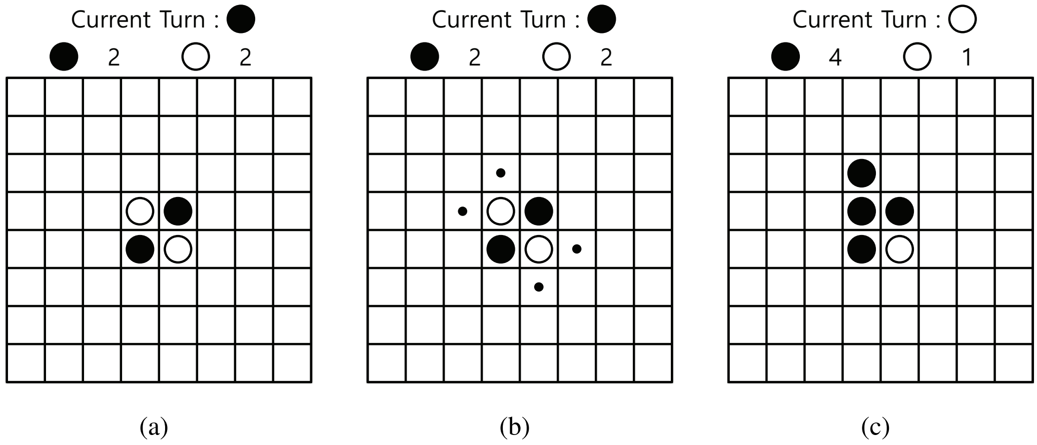


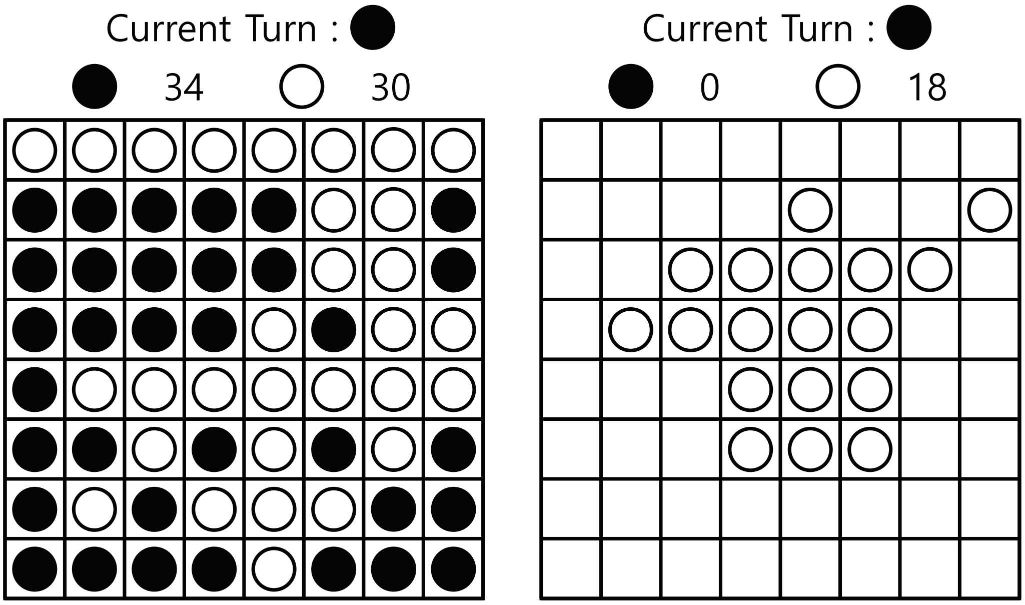
Figure 2.6: (a)Beginning of othello, (b) placeable positions marked as small black dots, (c) After placing black disc on (3,4) point.

opponent's discs, it is imperative that the opponent's discs be placed at top, bottom, left, or right diagonal. (figure 2.6 (b), Only small dot positions can placed) The game ends when all the blanks are filled with discs, or when two players no longer have slots to place discs, the player with more discs wins.

Starting with four discs in the middle, as shown in figure 2.6 (a). As shown in figure 2.6 (b), discs can be placed when the relative disc jumps over the diagonals, or the upper, or the lower, or the left, or right sides. If you put it in position (3,4), you can flip all the discs in between. As you can see, the Othello game has the property of reversing all opponent discs between discs and discs, so there is no guarantee of your winning simply because you currently have a large number of discs, you never know how the game will end. Also, there are cases where you have to fill all the points of the board like figure 2.7, and the game ends because there is no more place to place it.

2.3.1 Othello Program

For the experiment, we need to make an Othello program, which has a quite simple AI. So we made a Othello program that can verse each AI using C++ language. sometimes it needed to show or save image, so we used OpenCV API. The first version of the



(a) (b)

Figure 2.7: (a) General case that finish an Othello game (black wins), (b) Some special case that no more black discs place (white wins).

Othello program showed all placeable positions like - in figure 2.6 (b) (a small dot means placeable positions). It helped humans to play with AI. But to learn, it needed to be improved. So now the Othello program became capable of playing simple AI Vs. simple AI. Read an opponents weight value from another file and calculate weight summation value with current board state. But the weight consist of random values so it acted randomly. However it could repeat same position with same sequence. Of course it could change responses when other sequence was placed. And then, evaluating function was made and it became capable to estimate results. So major routine has check winning rate, and using statistic information, it could learn by itself. We called it as 1 iteration.

$$\text{Simple AI's policy} = \arg\max_{a} \left(E_{S_t}\right) \tag{2.13}$$

2.3.2 Weighted Pieces Counter

Whether it is the real world or the game, it is very important to evaluate a specific situation. You can evaluate its value in a number of situations, but that would result in too many cases. In many of these cases, it is important to create an appropriate strategy, and the assessment of the situation is the basis for creating such a strategy. For example, in the searching algorithm above, when searching for the process of winning is conducted, but the value of a situation is unknown, the search is meaningless. Therefore, it is very important to define these features. In this paper, we will use the Weighted Pieces Counter method, one of the known methods, to find the learning method. The Position-Weighted Pieces Counter (WPC) method is a way of evaluating the position of each board. There are various evaluation methods depending on the game, and among them, the following method is effective because it is important whether or not the Othello game occupies a specific position. The WPC method is very simple to present the board state (Lucas and Runarsson, 2006).

The WPC equation as follows:

$$E_{S_t} = \sum_{i,j=1}^{8} f(Weight_{(i,j)}) \cdot B_{(i,j)} \tag{2.14}$$

where: **B** is a board state.

As followed, representing the board state as a single value is a very simple and good way. This WPC method can also be described as a single layer neural network model. As shown in figure 2.8, it is possible to try to find the weight through this WPC method. The estimated E_{S_t} represents the state of the current board, and there is a simple strategy to play the game using the value of this WPC.

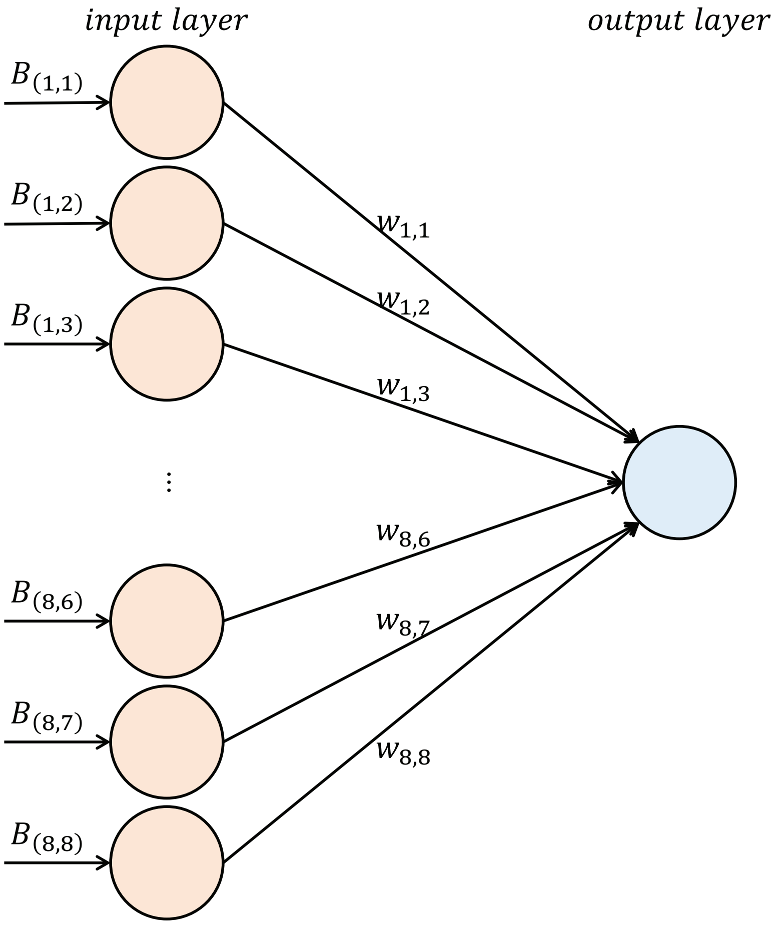


Figure 2.8: Single Layer Perceptron model of WPC method

2.4 Recent Work for Game of Othello AI

Until now, people have studied artificial intelligence through various learning methods. Among them, research on artificial intelligence in the Othello game has been going on since the 1980s, and in 1981, the way to make the world-championship-level Othello program was introduced. And then the game has been conquered by various approaches. In 1981, The Othello AI program was developed, starting with IAGO (Rosenbloom, 1982), and BILL 1.0, which was upgraded to BILL 3.0 in the early 1990s (Lee and Mahajan, 1990). Since the introduction of these programs, various attempts have been made such as introducing the LOGISTELLO program in 1995 (Buro, 1995, 1997b, 2003). In the IAGO program, an evaluation function using stability and mobility was created, and using this evaluation function, they tried to make

1.00	-0.25	0.10	0.05	0.05	0.10	-0.25	1.00
-0.25	-0.25	0.01	0.01	0.01	0.01	-0.25	-0.25
0.10	0.01	0.05	0.02	0.02	0.05	0.01	0.10
0.05	0.01	0.02	0.01	0.01	0.02	0.01	0.05
0.05	0.01	0.02	0.01	0.01	0.02	0.01	0.05
0.10	0.01	0.05	0.02	0.02	0.05	0.01	0.10
-0.25	-0.25	0.01	0.01	0.01	0.01	-0.25	-0.25
1.00	-0.25	0.10	0.05	0.05	0.10	-0.25	1.00

Table 2.2: The weights ($w_{(i,j)}$) for the heuristic player in (Ishii and Hayashi, 1999; Lucas and Runarsson, 2006).

the game AI that maximizes the evaluation value. BILL then tried to tune parameters in such stability and mobility, and to create a quadratic discriminant function for learning. In addition, a study using the Best-First Minimax Search (Korf and Chickering, 1994), which improved Minimax search, was conducted in this BILL program, and also tried to use the genetic algorithm (Alliot and Durand, 1995). In the learning using this genetic algorithm, the evaluation function was created by using the static square values and the liberty score, and they show that the learning was possible by modifying the internal factor through crossover and mutation. In addition, among the learning methods, there has been an effort to solve the problem by using the structure of the neural network to evolve the standard positional strategy, the mobility strategy, and the strategy of the outside of tournaments (Moriarty and Miikkulainen, 1995). It is a method of learning 'temporal difference learning' by using the neural network structure, which is constructed by dividing the whole board into 8 by using the symmetry of the top, bottom, left and right was also studied (Leouski, 1995). There is also an attempt to learn by creating a HONEST network with the neural network structure of the BILL program evaluate function (Abdelbar and Tagliarini, 1998). In addition to these feature tuning attempts, improvements in search algorithms have also been studied. Buro (1997a) introduced the concept of Multi-ProbCut in the program using LOGISTELLO, which showed improved search algorithm compared to existing ProbCut or brute-force search method, and showed better performance. Buro (2002) introduced improvements in evaluation using the generalized linear evaluation model (GLEM), improved search using ProbCut, and early evaluation methods using previous game information with the opening book framework. In addition to the improvements of these evaluation methods and the improvements of searching, reinforcement learning is also being tried. There is also a method of reinforcement learning using a radial basis function(RBF) network (Ishii and Hayashi, 1999), and a method of trying the concept of Q-learning (van Eck and van Wezel, 2004; Van Der Ree and Wiering, 2013). Lucas and Runarsson (2006) showed a learning method using TD-Learning among reinforcement learning methods. In addition, a method for constructing the evaluation function by constructing an artificial neural network has been studied (Binkley et al., 2007). Not only that, there are a lot of studies like a way to combine evolution and reinforcement learning (Binkley et al., 2007; Szubert et al., 2009), comparison of search using Monte-Carlo method and $\alpha\beta - pruning$ search (Nijssen, 2007), comparison of learning rates in TD-Learning (Lucas, 2008a), or a method of learning by grouping board states into n-tuples and expressing the situation more precisely (Lucas, 2008b). Also recent studies include improvement of search algorithm using Monte-Carlo Tree Search(MCTS) (Robles et al., 2011) and further study with network of n-tuple (Krawiec and Szubert, 2011; Jaśkowski, 2014). Also, in some papers, a complete search has been attempted by reducing 8x8 size to 6x6 size (Takeshita et al., 2017). So far, various studies have been going on, but there are less things to be introduced in a simpler way. Therefore, in this paper, we will try to apply artificial intelligence in this Othello game by applying various learning methods with the goal of introducing the contents very simply based on statistics.

2.5 Summary of Chapter 2

In this chapter, we introduced different background ideas. The field of artificial intelligence has been getting much attention from past to present. First of all, we introduced the importance of search and its method in 2-player board game, and looked over how to use probability and reinforcement learning among learning methods. We also examined what learning methods have been applied in game of Othello.

But, in this part, another approach shows more complex method rather than using more intuitive probabilities. So we want to try to solve the problem in a very simple way.

Chapter 3

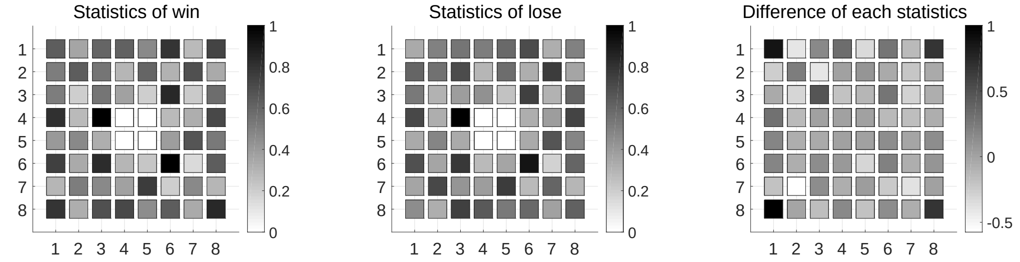
Learning with Bayes Probability

To find a good strategy, it is necessary to analyze statistics. We suggest that one of statistics analyze technique based on Bayes theorem. We will create statistical data through competition with several random weighted opponents, and analyze the process of these statistics. This section is being prepared for submission as a scientific paper (Hahn, JeongWon and Kim, DaeEun, 2018a).

3.1 Experimental Setup

3.1.1 Statistic Calculation in Game of Othello

Using statistic information is a simple way to get pattern about board. So, we want to check out a result of winning or losing, in each position. It is known that there are good and bad places to win in the Othello game (Lucas and Runarsson, 2006). Therefore, to confirm this, first, we collected all the records that occured in each place (victorious and defeated) by using random move agent. According to figure 3.1, you can see which position the disc was placed in the victory or defeat, and a specific position has a relatively higher number of victories. We gather these statistics and define it as a decision problem, and will try to make a probability model.



(a)

(b)

(c)

Figure 3.1: Check each statistics of winning (a), defeat (b), and difference of ratio (c).

In this case, we examined the ratio of discs placed when winning each spot(figure 3.1 (a)), the ratio of discs placed when defeated(figure 3.1 (b)), and the difference of each ratio. (figure 3.1 (c)) The result of this ratio is the case where the side-edge is the most probable to win, and we try to learn a definite place to win or lose by using this ratio.

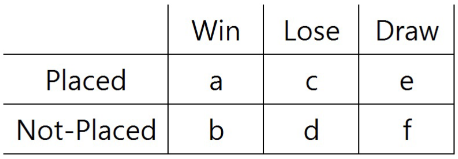
3.1.2 Mathematical Modeling

In this section, we want to represent probability in mathematical ways. Above this, we have defined the statistic model. Without any prior knowledge, the agent can only get information about victory or defeat of the game in each position. Therefore, we try to make a position value table about victory or defeat using the statistics of whether or not a disc has been placed at such a position. Finally, we want to maximize the probability of winning and also minimize the probability of losing. If we can get weight map which makes the probability of win approach 1, it is a perfect position evaluation result.

The goal of this system:

$$\underset{WeightMap}{\text{argmax}}\ P(Win)|WeightMap \tag{3.1}$$

Expressed as a decision problem, we need to define what is a measurement and what is a decision. Then, in this game, we will have to decide whether we should place the decision on the spot or not, this became from historical experience (same as statistics



	Win	Lose	Draw
Placed	a	c	e
Not-Placed	b	d	f

Figure 3.2: Bayesian probability in this game. This is the result that counted all position each cycle. Using this we can calculate a conditional probability in each position.

). Thus, the measurement will become historical experiences, which means statistic result. Measurement will be the result after the game is finished. That is, information about winning or losing, will be measurement.

Now that we have completed the definition above, you can see it as a classifying problem or decision problem. We have two methods for classifying. One is Likelihood, and the other is Posteriori.

The probability of likelihood looks like this

$$P(measurement|class) \tag{3.2}$$

and the probability of posteriori looks like this

$$P(class|measurement) \tag{3.3}$$

and these probability come from the bayes rule:

$$P(class|measurement) = \frac{P(measurement|class)P(class)}{P(measurement)} \tag{3.4}$$

In our problem, measurement is win or lose or draw and decision(class) is place or not, from the information in figure 3.2.

Likelihood Model looks as follows :

$$P(Win|Placed)\ and\ P(Lose|Placed) \tag{3.5}$$

and each equation that from figure 3.2 :

- $P(Win|Placed)$ can be represented by $\frac{a}{a+c+e}$
- $P(Lose|Placed)$ can be represented by $\frac{c}{a+c+e}$

also, A Posteriori Model looks as follows :

$$\begin{aligned} P(Placed|Win)\, and\, P(Placed|Lose) \\ where\, P(Placed|Win) = \frac{P(Win|Placed)P(Placed)}{P(Win)} \end{aligned} \tag{3.6}$$

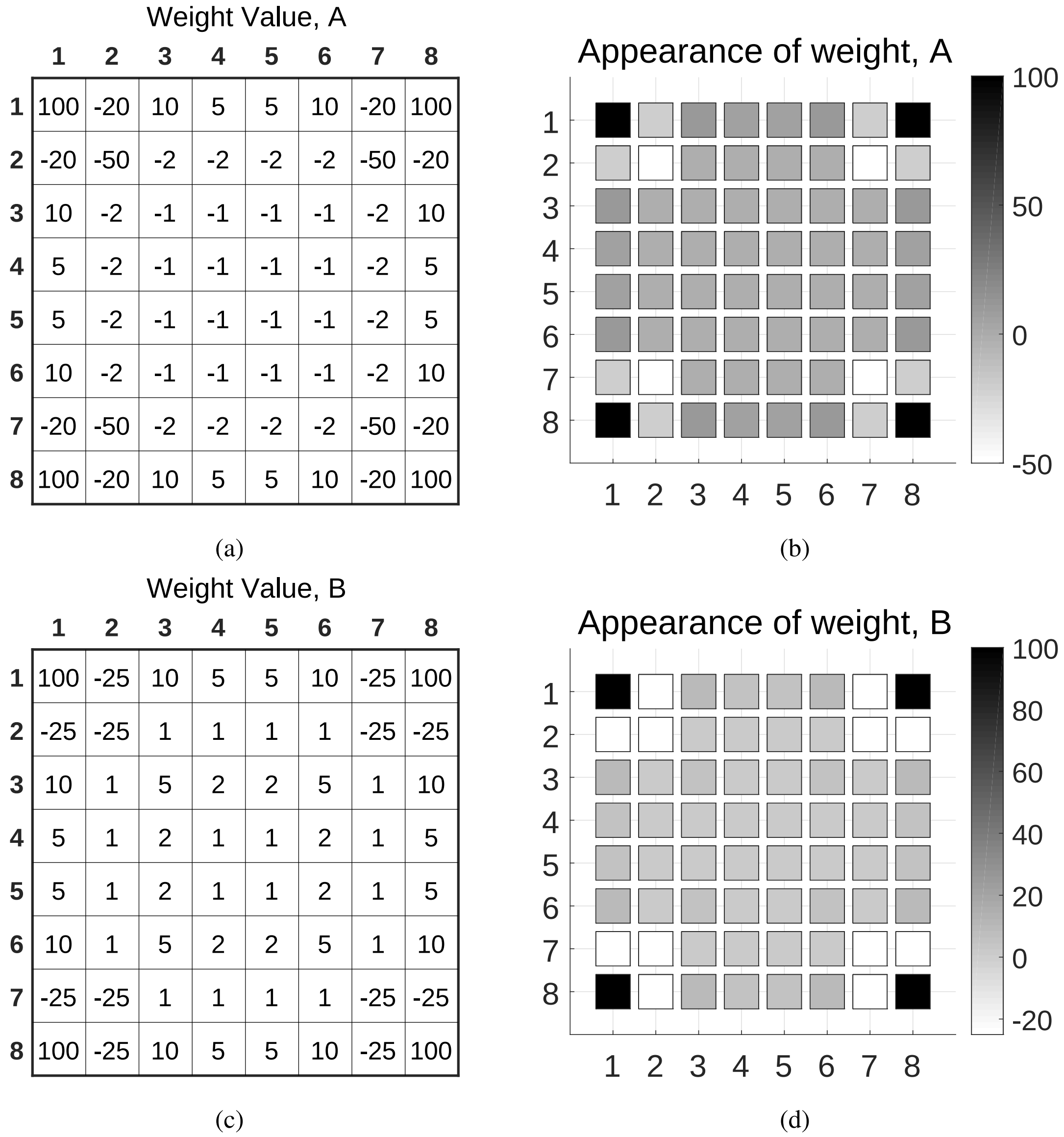
and each equation that from figure 3.2 :

- $P(Placed|Win)$ can be represented by $\frac{a}{a+b}$ $(\frac{a}{a+c+e} \times \frac{a+c+e}{a+b+c+d+e+f} \div \frac{a+b}{a+b+c+d+e+f})$
- $P(Placed|Lose)$ can be represented by $\frac{c}{c+d}$ $(\frac{c}{a+c+e} \times \frac{a+c+e}{a+b+c+d+e+f} \div \frac{c+d}{a+b+c+d+e+f})$

3.1.3 Evaluate the Board

If the good and bad of each place can be represented statistically, we will be able to make one evaluation function whether or not we are occupying that place in the Othello game. This evaluation function can be expressed as the Weighted Pieces Counter introduced in Chapter 2, and the known weight map examined in some papers is composed as follows (figure 3.3) from van Eck and van Wezel (2004) and Lucas and Runarsson (2006). Each weight map have similar shapes even though its values are different.

It will be position evaluation functions and we can calculate the functions like this. It is position weighted piece counter(WPC), which is a linear weighted board function (Lucas and Runarsson, 2006; Liskowski, 2012). The input value **B** depends on the occupation of the paricular board location. For example, +1 when black occupies the board, -1 when white occupies, 0 empty location. WPC is explicitly used to evaluate how beneficial a given state is for a particular player. The WPC evaluation function of an Othello game strategy takes the form of a vector of 64 weights.



Weight Value, A

	1	2	3	4	5	6	7	8
1	100	-20	10	5	5	10	-20	100
2	-20	-50	-2	-2	-2	-2	-50	-20
3	10	-2	-1	-1	-1	-1	-2	10
4	5	-2	-1	-1	-1	-1	-2	5
5	5	-2	-1	-1	-1	-1	-2	5
6	10	-2	-1	-1	-1	-1	-2	10
7	-20	-50	-2	-2	-2	-2	-50	-20
8	100	-20	10	5	5	10	-20	100

(a)

(b)

Weight Value, B

	1	2	3	4	5	6	7	8
1	100	-25	10	5	5	10	-25	100
2	-25	-25	1	1	1	1	-25	-25
3	10	1	5	2	2	5	1	10
4	5	1	2	1	1	2	1	5
5	5	1	2	1	1	2	1	5
6	10	1	5	2	2	5	1	10
7	-25	-25	1	1	1	1	-25	-25
8	100	-25	10	5	5	10	-25	100

(c)

(d)

Figure 3.3: Known weight value and apperance (a,b) from van Eck and van Wezel (2004) and (c,d) from Lucas and Runarsson (2006).

3.2 Experiment

In this paper, we try to determine which place is a better place through the probability model. A better place means a way to win more, so in this paper the overall goal is equal to equation 3.1. ł Ł classification ł Likelihood A Posteriori weight , 1 , weight ł . To achieve the goal, we construct a weight map using likelihood and a posteriori as classification model. And this is not a simple one-time classification, but we want to experiment repeatedly collecting the weight.

$$W_{i,j} = W_{i,j} + \frac{1}{N}\left(\frac{\partial h(\mathbf{x},t)}{\partial t}|_{pos=i,j}\right) \tag{3.7}$$

We also compare Likelihood and a Posteriori, in which using the difference of value in the case of victory and defeat means choosing a larger value in the classify method more stochastically. No matter what probability model we use, in the end we have to make a better choice. It is the goal of this game to compare placements in multiple selectable places to see if it is worth more to win, and to place if it seems that there is a chance of winning. In other words, if we re-express it, we compare the probability of winning in each place among multiple selectable places ($P(Win, Place) > P(Lose, Place)$), The best choice is made by comparing the most likely.

3.2.1 Likelihood-Estimation Model

First step, we choose, Maximum Likelihood(ML) Estimation. Because ML model provides more insight from statistics by how it looks. So we thought about how to use a statistical result to learn. Statistical learning needs a lot of opponents, so we made it random (It has random weight). Then we versed it to Learner. In other words, it was like The learner Vs. type 1 to N Opponents. So in this experiment, we can acknowledge what position is good or not from cumulating a position record that no matter who wins.

Then we want to use this information to make a weight map in WPC. Weight map has a meaning that each position has a value about the disc position. Several methods exist to learn this position map, but in this paper we want to use conditional probability. A simple way, we record all positions that are placed when a player wins and loses. This record represents each position has good or not to place.

The purpose of this system is eventually represented by equation 3.1. Therefore, in this system, the derivation of weight is defined as follows, and we want to proceed in the direction of reducing this error. Here we can use the derivative of the weight of time to update the weight, and the result is the same as equation 3.8.

$$\begin{aligned} \frac{\partial h(\mathbf{x},t)}{\partial t}|_{pos} &= (Weight_{t,pos} - Weight_{t-1,pos}) \\ &= \frac{1}{k}(P(Win|Placed)|_{pos} - P(Lose|Placed)|_{pos}) \\ where\, \frac{1}{k} &\approx (1-P(Win))\, in\, method\, 1 \end{aligned} \tag{3.8}$$

Here, we denote the derivative of $P(Win)$ as $(P(Win|Placed) - P(Lose|Placed))$ where we use the probability of win given placed, because we look decision and measurement as Likelihood modeling in the decision problem. Here, the reason for using the information of the probability of lose is that $1-P(Win) = P(Lose)+P(Draw)$, assuming $P(Draw) \approx 0$. Update weight map uses equation 3.7. It represents how valuable a certain position is for win. In other words, when we place a certain position, probability of winning is higher than probability of losing. It means valuable. So, in this case $P(Win|Placed)$ is represented by $\frac{a}{a+c+e}$ and $P(Lose|Placed)$ is represented by $\frac{c}{a+c+e}$ from figure 3.2

If the probability of win given placed is greater than the probability of lose given placed it has +1 reward. If the probability of win given placed is less than the probability of lose given placed it has -1 reward. And the other case that the same probability between both of them, do nothing. It means there is no reward. It can be rewritten as following equation 3.9, especially method 3.

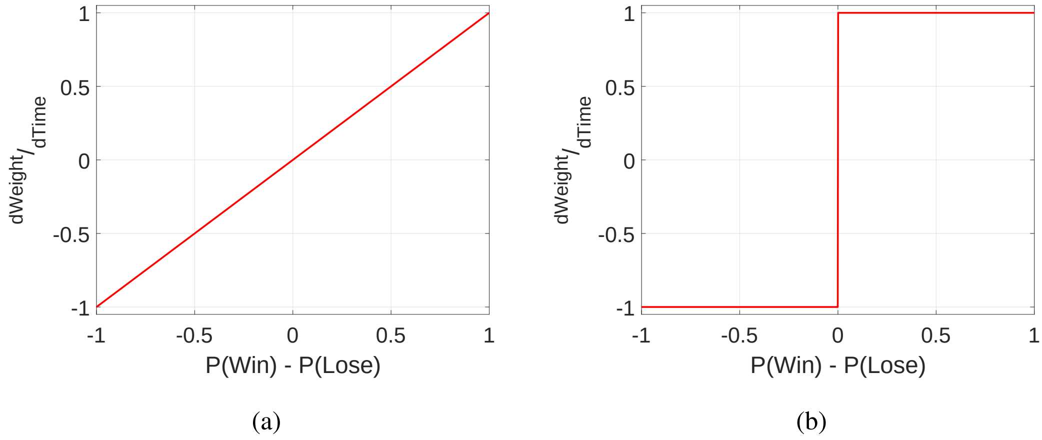


Figure 3.4: Weight Map changes and probability relationship graph. (a) linear value learning system(equation 3.9, method 2), (b) Biased value learning system. (equation 3.9, method 3)

$$\frac{\partial f(x,t)}{\partial t}|_{i,j} \triangleq \begin{cases} (1-P(Win))(P(Win|Placed)|_{i,j} - P(Lose|Placed)|_{i,j}), & \text{method 1} \\ const(\beta)(P(Win|Placed)|_{i,j} - P(Lose|Placed)|_{i,j}), & \text{method 2} \\ h_{step}(P(Win|Placed)|_{i,j} - P(Lose|Placed)|_{i,j}), & \text{method 3} \end{cases}$$

$$where\, h_{step}(x) = \begin{cases} -1, & \text{x} < 0 \\ 0, & \text{x} = 0\; in\, method\, 3 \\ +1, & \text{x} > 0 \end{cases} \tag{3.9}$$

According to this rule(equation 3.9), $W_{i,j}$ represents that how valuable position (i,j) is. We had several trials to learn weight map(=preference position value map). The first attempt was to try β as a step function in equation 3.9, and weight map has a range of -100 to 100. In the second experiment, we assumed that $\beta = 1$. Since $W_{i,j}$ ranges from -100 to 100, it looks like $\beta = 0.01$ ranges from -1 to 1. In other words, we thought about a situation where there are more victories and fewer updates.

First, the results of ML model are shown in figure 3.5. This graph shows it can't learn at all. No matter how good information comes in, it seemed impossible to find a better place for victory. We thought that these things would continue to change without the

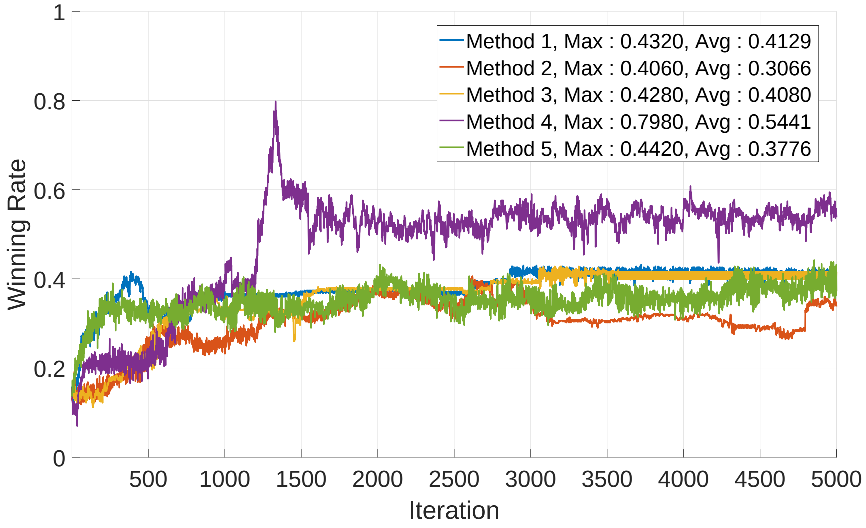


Figure 3.5: The result of likelihood estimation model verses with training set. Method 1 to 3 written above equation 3.9 and method 4 and 5 is result of take moving average filter with method 3 and 1.

weights stabilizing, and we wanted to proceed with the review using an average filter. In the figure 3.5, method 4 uses the moving average filter for method 3, and method 5 uses the moving average filter for method 1. Each window of moving average filter is 50. However, in this Likelihood model, the average filter did not show the possibility of increasing the winning rate. (figure 3.5, Method 4 and 5) The likelihood model was impossible to learn in any way. We decided to try again with the A Posteriori method because the winning rate of current method shows failed saturation.

In this case, we thought the linear value and the biased value are changed too much or too less to saturation, so the gain is changed to make difference bigger or smaller, but in all cases failed to learn. Of course, according to β, it changes a little bit. But all cases were saturated about 42% winning rate. The difference was just about how much fluctuation existed.

We looked at one of these ways to change the winning rate and weight. Figure 3.6 (1,1) and weight changes at (3,3) position. According to the known weight map (figure 3.3), this position is not saturating to ±100, but it should have a value near 0, but it

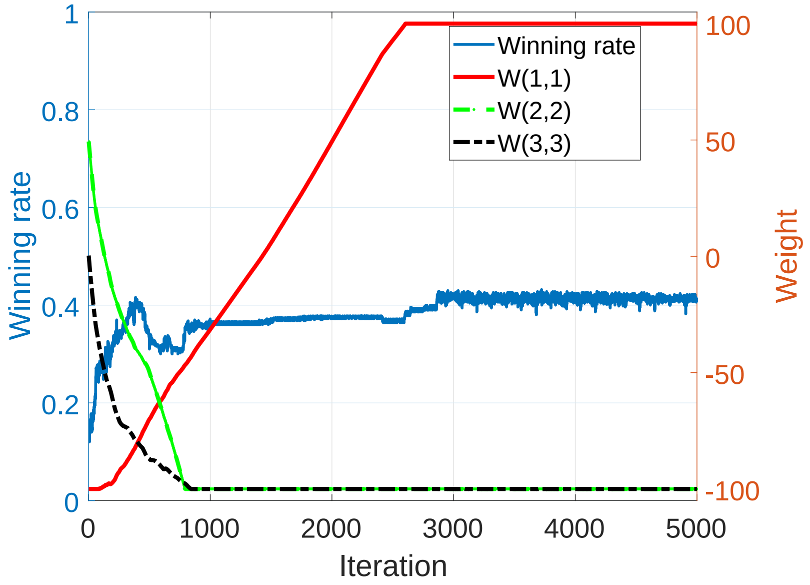


Figure 3.6: The variation of weight in the method 1 from figure 3.5. (1,1) position and (3,3) position are shown together in one graph.

converges to -100. In the known weight map, the value of the outer position and the value of the inner position are completely different. However, when we compare the results obtained through the Likelihood model, we can see that the values(=$|weight|$) of (1,1) and (3,3) are the same. Since the value of each position does not converge properly, the winning rate are considered to be a mess, and the following A posteriori model is examined.

3.2.2 A Posteriori-Estimation Model

It looks as if likelihood model failed to saturate enough probability of winning. Because the known weight map (figure 3.3) has about 80% winning rate. So second step that we choose is A Posteriori-Estimation. Likelihood ł Bayes Ł Posteriori . Here, MAP model is to define the derivation of weight as following equation 3.10, assuming that the weight constituting $P(Win)$ is made through MAP.

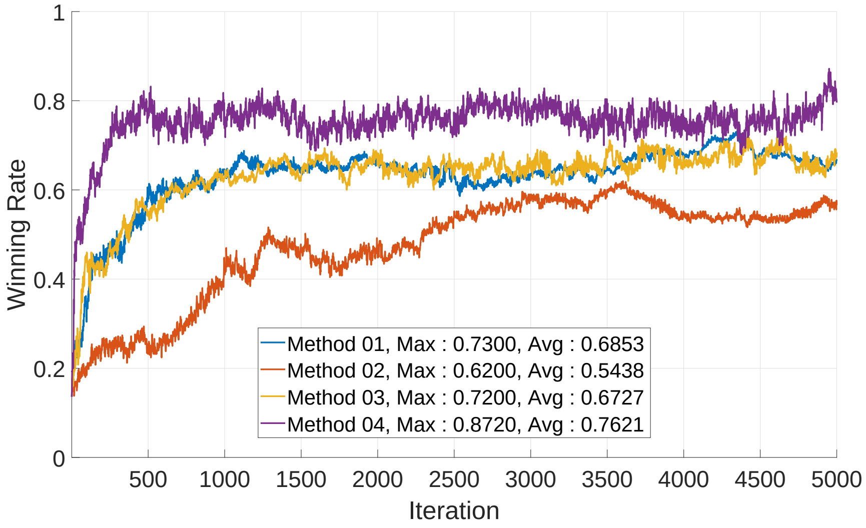


Figure 3.7: The result of a posteriori estimation model verses with training set. Method 1 to 3 written above equation 3.11. Method 4 applied average filter with method 3.

$$\begin{aligned}\frac{\partial f(\mathbf{x},t)}{\partial t}|_{i,j} &= (Weight_{i,j|t} - Weight_{i,j|t-1})\\ &= \frac{1}{k}(P(Placed|Win)|_{i,j} - P(Placed|Lose)|_{i,j})|_t \\ where\,\frac{1}{k} &\approx (1-P(Win))\,in\,method\,1\end{aligned} \tag{3.10}$$

So, now figure it out using the probability of place given win or lose that using MAP model. In this case $P(Placed|Win)$ is represented by $\frac{a}{a+b}$ and $P(Lose|Placed)$ is represented by $\frac{c}{c+d}$ from figure 3.2. We will look at each case in the same way as before. (Methods from equation 3.11)

$$\frac{\partial f(\mathbf{x},t)}{\partial t}|_{i,j} \triangleq \begin{cases} (1-P(Win))(P(Placed|Win)|_{i,j} - P(Placed|Lose)|_{i,j}), & \text{method 1}\\ const(\beta)(P(Placed|Win)|_{i,j} - P(Placed|Lose)|_{i,j}), & \text{method 2}\\ h_{step}(P(Placed|Win)|_{i,j} - P(Placed|Lose)|_{i,j}), & \text{method 3}\end{cases} \tag{3.11}$$

We experimented several methods in the above equation 3.11, and this time, we were able to see a winning percentage of over 50%. However, method 2 did not saturate and was shaking, and showed lower winning rate than other methods. Therefore, we tried

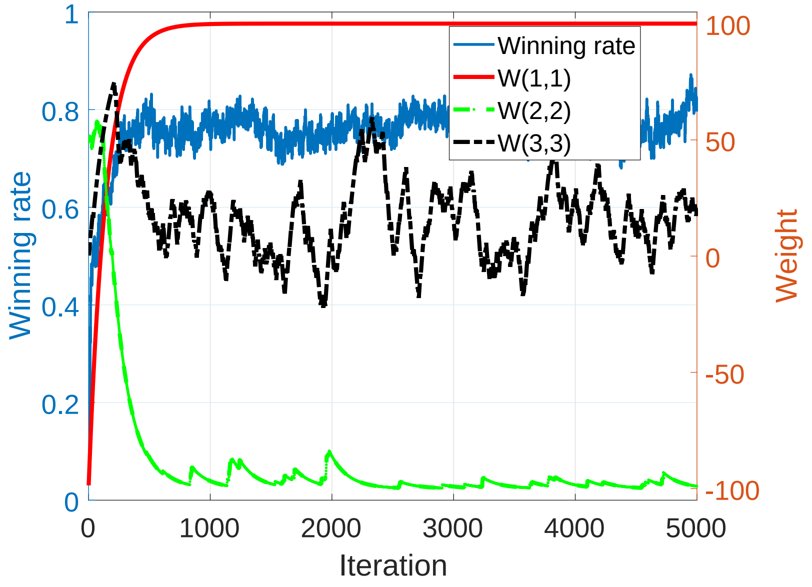
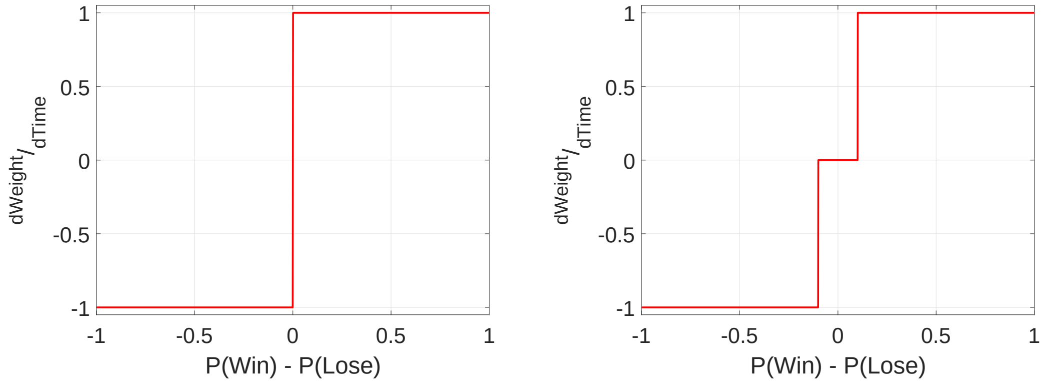


Figure 3.8: The variation of weight in the method 4 from figure 3.7. (1,1) position and (3,3) position are shown together in one graph. Weight of (3,3) position has been continuously changed.

to compare it with β, but it could not exceed the maximum 70%. According to the definition which is the derivation of weight, the change in weight is most accurately represented by method 1, but in practice, it can be seen that there is no difference from method 3. However, these two equations can not be seen in the same way, because in method 1, the larger the $P(Win)$, the smaller the weight and saturation changes. However, in method 3, the weight is continuously changed according to the probability. So, to solve this problem, we looked for the weight update expression as follows (equation 3.12).

$$W_{(i,j)}|t+1 = (1-\alpha)W_{(i,j)}|t + \alpha\frac{\partial f(\mathbf{x},t)}{\partial t}|_{i,j} \tag{3.12}$$

These four results are the same as figure 3.7. Here, the choice of α was chosen to be 0.5 based on $(1-\alpha)^{100}$. We have experimented a few cases, but the difference according to α did not have a significant effect on the winning rate. Only the difference in the magnitude of fluctuation and the speed of convergence was affected. Experimental



(a)

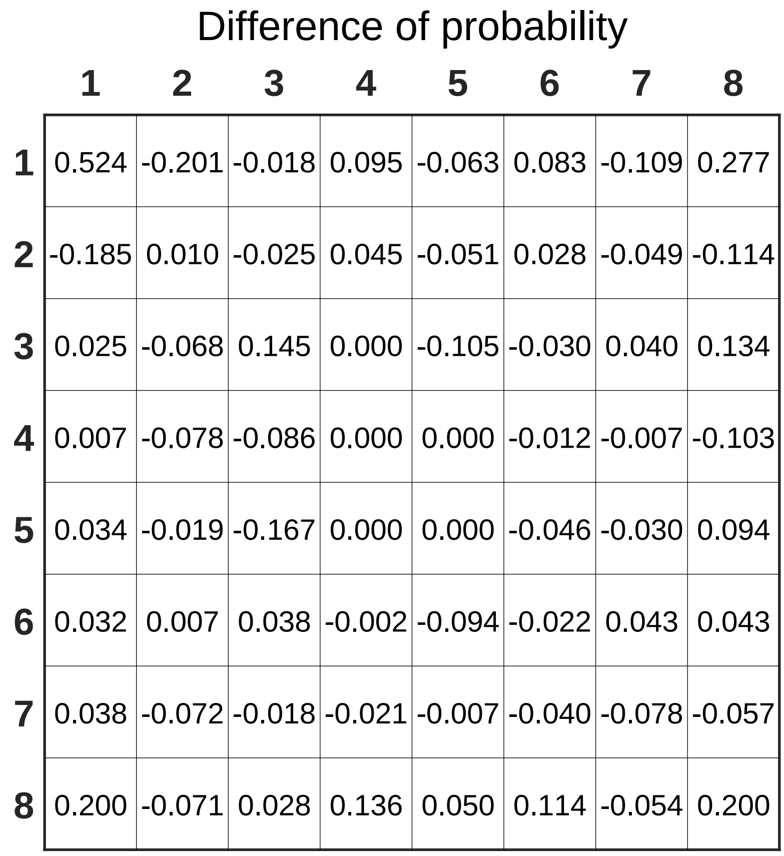
(b)

Figure 3.9: WeightMap learning curve. (a) previous learning from figure 3.4, (b) with dead zone, i.e. 0.1, Update weight -1 when $P(Placed|Win) - P(Placed|Lose) < -0.1$ and +1 when $P(Placed|Win) - P(Placed|Lose) > 0.1$

results show that the larger the α is, the faster the convergence is but the greater the fluctuation is. On the contrary, the slower the convergence rate is, the more the fluctuation decreases. Method 4 seems like learning. However, if we look at the change in weight, we can see that the figure 3.8 continues to change. To solve this, we introduce the next section, dead zone.

3.2.3 Dead Zone

In the two experiments above, we confirmed two formulas: 'likelihood $= P(measurement|class)$' and 'a posteriori $= P(class|measurement)$'. In order to maximize $P(measurement)$, we can confirm that the differential is correct by measuring the condition. In this section, we can see that the weight changes continuously when we look at figure 3.8 in the model above. This suggests that even if the value of $(Placed|Win) - P(Placed|Lose)$ is close to 0 in the probability model, it is updated by the step function as ± 1. Therefore, if $P(Placed|Win) - P(Placed|Lose)$ value is moderately small, we try to solve the problem without updating the weight. The weight update used here is the figure 3.9, (b).



Difference of probability

	1	2	3	4	5	6	7	8
1	0.524	-0.201	-0.018	0.095	-0.063	0.083	-0.109	0.277
2	-0.185	0.010	-0.025	0.045	-0.051	0.028	-0.049	-0.114
3	0.025	-0.068	0.145	0.000	-0.105	-0.030	0.040	0.134
4	0.007	-0.078	-0.086	0.000	0.000	-0.012	-0.007	-0.103
5	0.034	-0.019	-0.167	0.000	0.000	-0.046	-0.030	0.094
6	0.032	0.007	0.038	-0.002	-0.094	-0.022	0.043	0.043
7	0.038	-0.072	-0.018	-0.021	-0.007	-0.040	-0.078	-0.057
8	0.200	-0.071	0.028	0.136	0.050	0.114	-0.054	0.200

Figure 3.10: In the course of learning, we showed the difference of probability $P(Placed|Win) - P(Placed|Lose)$ in each position. This is a very small value, but you can see that all the board conditions change.

The reason for the decision of dead zone here was to see the weight change. The following figure 3.10 shows the $P(Placed|Win) - P(Placed|Lose)$ in each position. In the figure, all the values do not need to be changed, but only a partial change will make the entire board stable.

Therefore, we introduced this dead zone. We applied dead zone to method 4 in the above figure 3.7. In other cases, the dead zone did not show any improvement. Particularly, in case of method 1, the dead zone and $(1 - P(Win))$ parts work together and the probability of winning is lower than 0.5. When we applied dead zone, we could see the saturation being stable, but we could see that it did not rise more than expected. We were able to see some of the dead zone levels change, but it could not go up and saturation stopped at a certain level. (figure 3.11) This is because the movements of the

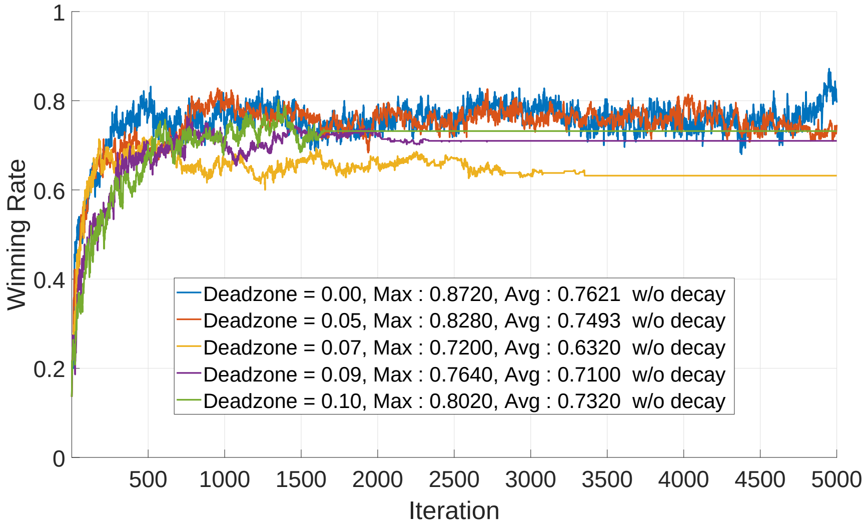


Figure 3.11: Winning rate according to dead zone range against the training set. It applied a posteriori approach. It seems to unnecessary but it shows an effect which more stable.

opponents are not totally random, but because they move based on their weight map which was constructed randomly. Therefore, all the weights are reduced little by little regardless of probability. In other words, we introduced a method to decay the weight slightly without fixing the weight without changing the dead zone.

When decay and dead zone were applied together, the problem could finally be solved. Without dead zone, it saturated about 70 to 80% winning rate. But, in this time, when dead zone is applied, it goes 80 to 90% rate. It also is quite similar to a weight map (figure 3.14) compared with a known one (figure 3.3).

3.2.4 Temporal Difference Learning

In order to make sure that the estimation model through Bayesian analysis is effective, we try to compare it with other existing approaches which is Temporal Difference Learning (Lucas and Runarsson, 2006). Temporal difference learning is one of the unsupervised learning methods that reflects the future value to the present and makes a choice with the future value. This learning method is called temporal difference

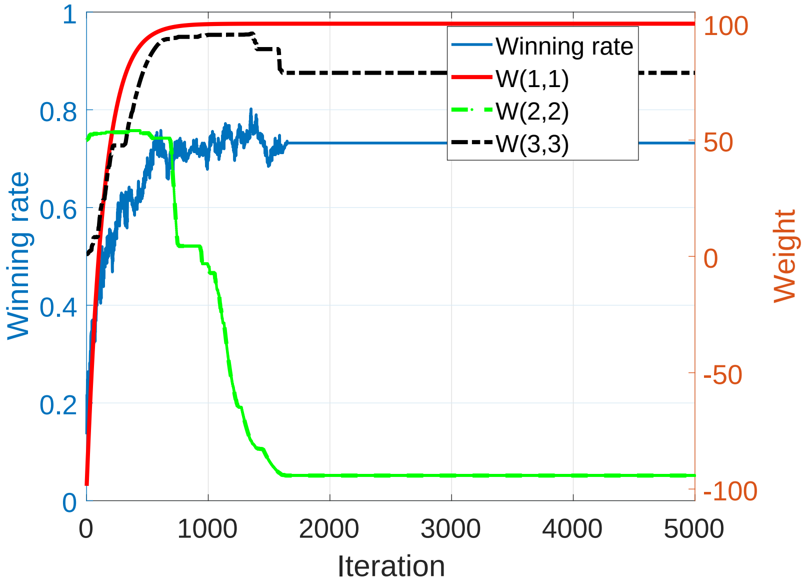


Figure 3.12: The variation of weight in the method 4 adapted deadzone(0.1) from figure 3.7. (1,1) position and (3,3) position are shown together in one graph. Weight of (3,3) saturated but converges to a different value from the known value.

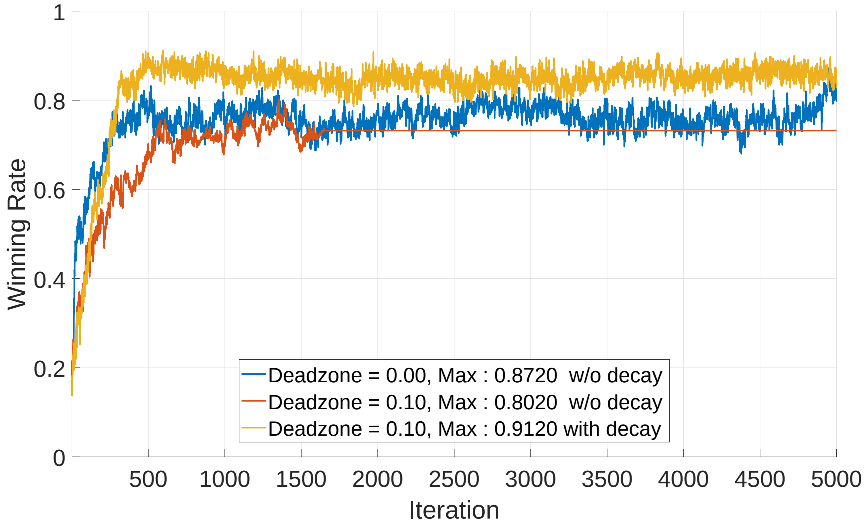


Figure 3.13: The result of applying deadzone and decay method together when using a posteriori approach. It against the training set.

learning because there is a difference in the temporal relationship between the present and the future. This method of learning using TDL is a pre-existing method and we want to compare this method with the previous results in Othello game. The TDL method uses the occupied state of each position and the weight of the position. In TDL, the weight is updated in a gradient-descent method. Let $\widehat{x}$ be the board that when the agent moved, and $\widehat{x}'$ be the board that after the agent moved. The evaluation function updated as follows:

$$\begin{aligned} w_i &= w_i + \alpha[v(\widehat{x}') - v(\widehat{x})]\frac{\partial v(\widehat{x})}{\partial w_i} \\ &= w_i + \alpha[v(\widehat{x}') - v(\widehat{x})](1 - v(\widehat{x})^2)x_i \end{aligned}$$

where

$$v(\widehat{x}) = tanh(f(\widehat{x}) = \frac{2}{1 + exp(-2f(\widehat{x})} - 1$$

$$x_i = +1,\ 0\ or\ -1(\text{+1 if agent's discs, 0 for empty and -1 for opponent's discs})$$

$$(3.13)$$

And $f(\widehat{x})$ same as WPC. Also when update the function if $\widehat{x}$ met terminate condition, w_i will be updated with $v(\widehat{x}') = +1,\ 0\ or\ -1$ (+1 if the winner is agent, -1 when the other, and 0 for a draw). This is based on Sutton and Barto (1998), and the formulation of it in equation 3.13 came from Lucas and Runarsson (2006). At each turn of the game, the agent update the evaluation function.

3.2.5 MCTS Algorithm

Because of winning rate limited under 0.9, we felt that it has limitation with just WPC algorithm. So, in this time, we would like to show you the search algorithm, not the learning method described above. The search method using mini-max algorithm has a problem that the time required expands exponentially because it searches the entire section. Normally, it increases in the form of square. It can't be solved within polynomial time, it takes N^M time, and example of Mini-max algorithm that searches 4 depth, assuming that we can choose 8-position to place, requires $Const * 8^4$ time. Also, even if we use the minimax algorithm to search up to 4 depth, you can't be sure of winning. This uncertainty makes problem more complicated. Therefore, most games, not just Othello, encounter the problem of size limitations of this search and introduce a better algorithm to determine the behavior within a fixed time. One of those algorithms is Monte-Carlo Tree Search algorithm which is described in chapter 2.

The MCTS algorithm can be described as an optional full search. Of course, the MCTS algorithm will also explore the number of all cases if there is plenty of time, but in most cases, MCTS is applied to avoid it. To apply the MCTS as described above(chapter 2.1.3), first we need to define the values what we use that the MCTS should have. The number of exploration, selection algorithm, the method that how to simulate, and the result. The selection function most commonly used is called UCT and described in chapter 2.1.3 (Kocsis and Szepesvári, 2006).

with this algorithm, the result is as follows:

Methods	WPC	MCTS
$P(Win)$ with 100 search	0.866	0.70
$P(Win)$ with 500 search	0.866	0.78

In MCTS, we do not directly use WPC value, but extend node with possibility of victory. In other words, the information about how much each node is likely to win is compressed to a winning or losing by a simulation result. Therefore, we think that it is not enough to depend on the simulation results, and we modified the equation a little to reflect the end result of the final real game.

so we modified it as follows:

$$(1-\lambda)\frac{w_{b,i}}{w_{b,i}} + \lambda(\frac{w_{s,i}}{n_{s,i}} + c\sqrt{\frac{lnt}{n_{s,i}}}) \tag{3.14}$$

Where:

- $w_{s,i}$ stands for the number of wins after the ith move using position evaluation data(= simulation).
- $n_{s,i}$ stands for the number of simulations after the ith move using position evaluation data(= simulation).
- $w_{b,i}$ stands for the number of wins after the ith move.
- $n_{b,i}$ stands for the number of simulations after the ith move.
- c is the exploration parameter, $sqrt(2)$ is a good first guess for this number, but in practice, you'll have to tune it experimentally.
- t stands for the total number of simulations, equal to the sum of all $n_{s,i}$. Or, another way to think about this is that it's the $n_{s,i}$ of the parent node.

With this equation, we expect to achieve better results than the existing Othello program that uses MCTS because it reflects not only searching simulation but evaluation that made it before. Before simulation to the end of the game, the next select is performed using the previously evaluated weights. When after completing the simulation to the end of the game, it back-propagates the result. And the result that is back-propagated, has proper weights. So we can select the next node using combination of these two factors. Also, because there is a log (t) part that is identical to the general MCTS, it is possible to search for nodes that have not been visited previously.

Algorithm 2 Pseudo code for Monte-Carlo Tree Search in Othello

```
function ComputeModifiedUCT($Node$)
    if $Node.Visit$ is false then
        return $ConstA$.
    end if
    return $(1-\lambda)\frac{w_{b,i}}{w_{b,i}}+\lambda(\frac{Node.Wins}{Node.Visits}+ConstB\frac{log Root.Visits}{Node.Visits})$
end function
```

With this algorithm, the result is as follows:

Methods	WPC	MCTS	Modified MCTS
$P(Win)$ with 100 search	0.866	0.70	0.72
$P(Win)$ with 500 search	0.866	0.78	0.81

However, adding the actual game results together still made MCTS scores worse than using WPC. Using WPC strategy, the larger WPC value, the better behavior is produced. It is because it expresses information about whether it wins or loses as a part of evaluation function by compression. So we modified WPC to normalize by applying absolute value and add it to the value of node in MCTS.

Algorithm 3 Pseudo code for Monte-Carlo Tree Search in Othello

```
function ComputeModifiedUCT($Node$)
    if $Node.Visit$ is false then
        return $ConstA$.
    end if
    return $(1-\lambda)\frac{w_{b,i}}{w_{b,i}}+\lambda(\frac{Node.Wins}{Node.Visits}+ConstB\frac{\log Root.Visits}{Node.Visits})+\beta AWPC(Board)$
end function
```

$$AWPC(Board) = \frac{|\sum w_{(i,j)} \cdot B_{(i,j)}, B_{(i,j)=Black}|}{|\sum w_{(i,j)} \cdot B_{(i,j)}, B_{(i,j)=Black}| + |\sum w_{(i,j)} \cdot B_{(i,j)}, B_{(i,j)=White}|} \tag{3.15}$$

With this algorithm, the result is as follows:

Methods	WPC	MCTS	Modified MCTS	MCTS+WPC
$P(Win)$ with 100 search	0.866	0.70	0.72	0.88
$P(Win)$ with 500 search	0.866	0.78	0.81	0.91

Finally, it is confirmed that the WPC value should be maintained to obtain better results even if MCTS is used. It can be seen that MCTS with another strategy is a more powerful classifier than original MCTS.

Methods	Method1	Method2	Method3	Method4	Method5	Method6
Likelihood	0.432	0.406	0.428	0.798	-	-
Posteriori	0.730	0.620	0.720	0.872	0.802	0.912

Table 3.1: Comparison table according to learning method which value is max(P(Win)). Method 1, 2 and 3 are in accordance with equation 3.9 and 3.11, Method 4 applied equation 3.12 to Method 3, Method 5 applied deadzone to Method 4, and Method 6 is the result of applying dead zone and decaying in Method 4.

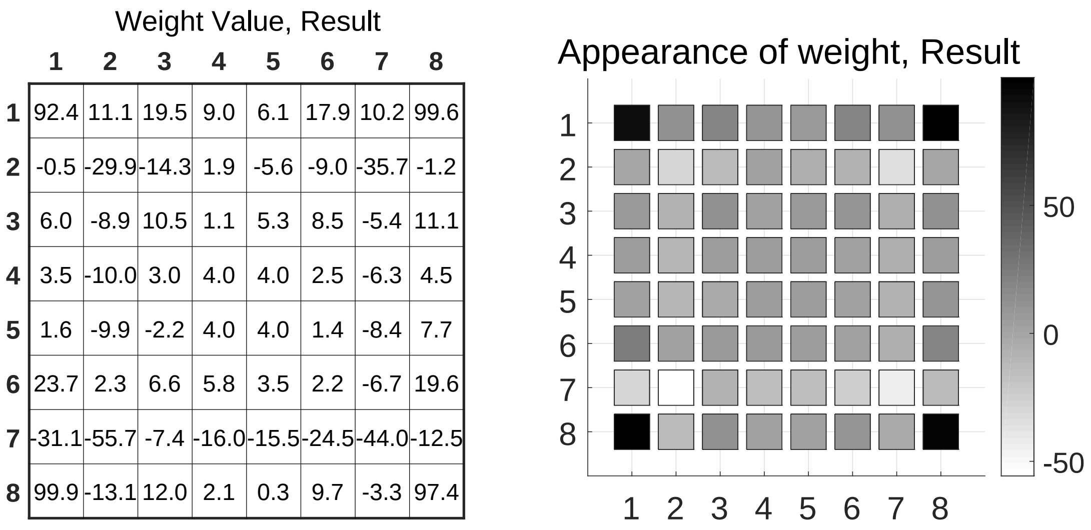
3.3 Experimental Result

3.3.1 Learning Method

To find proper position evaluation value, we had tried several methods. What we tried were biased value update (equation 3.9 and 3.11, (3)), using linear value update (equation 3.9 and 3.11, (2)) with Likelihood-Estimation method and A Posteriori-Estimation method.

First of all, in order to examine the probability model, several experiments were conducted to find the appropriate number of opponents. In order to examine how much random opponents are needed to learn, we have created from 100 opponents to 5000 opponents, but we have experimentally confirmed that 500 opponents are enough. Therefore, we chose the number of opponents to be 500.

Also, the formula for updating the weight is defined through likelihood and posteriori, and the results are shown (figure 3.5 and figure 3.7). As a result of comparison, it was found that it is correct to use posteriori, which is a measurement as a given condition, as a feature to win. In addition, we succeeded in learning the desired shape through the method of updating the difference of probabilty little by little and ignoring too small values like the gradient descent method (equation 3.12). It is solved by a simple conditional probability rather than a complex method known before. The figure 3.14 is the weight value finally obtained through this A posteriori method, and when it is



Weight Value, Result

	1	2	3	4	5	6	7	8
1	92.4	11.1	19.5	9.0	6.1	17.9	10.2	99.6
2	-0.5	-29.9	-14.3	1.9	-5.6	-9.0	-35.7	-1.2
3	6.0	-8.9	10.5	1.1	5.3	8.5	-5.4	11.1
4	3.5	-10.0	3.0	4.0	4.0	2.5	-6.3	4.5
5	1.6	-9.9	-2.2	4.0	4.0	1.4	-8.4	7.7
6	23.7	2.3	6.6	5.8	3.5	2.2	-6.7	19.6
7	-31.1	-55.7	-7.4	-16.0	-15.5	-24.5	-44.0	-12.5
8	99.9	-13.1	12.0	2.1	0.3	9.7	-3.3	97.4

(a)

(b)

Figure 3.14: The result of learning algorithm, weight value and appearance (a,b) from figure 3.13 method.

used, the winning rate with random weighted opponents is up to 91%. This is because the difference between ± 1% is only shown by repeated experiments.

Whatever the weight update function using the MAP estimation, it seemed to fall into the local minima problem in all cases. To resolve this local minima problem, my approach was to use decaying weight method (Thomassen, 1998). We thought it can solve local minima problem, as we thought it will work better, but felt quite a shortage. It couldn't take over the winning rate with the known weight map. We thought too large movements in weight value was the reason. So, we use dead zone or roulette to regularize it. Finally it worked. The winning rate came about 80 to 90% and weight map appearance also looked like the known reference (figure 3.3 and figure 3.14).

3.3.1.1 Comparing other approaches

The results of the learning through TDL were the same as those shown in Lucas and Runarsson (2006). It is important to note that when learning through TDL, you do not reflect the choice of opponents, but only with the agent's choice. At the beginning of

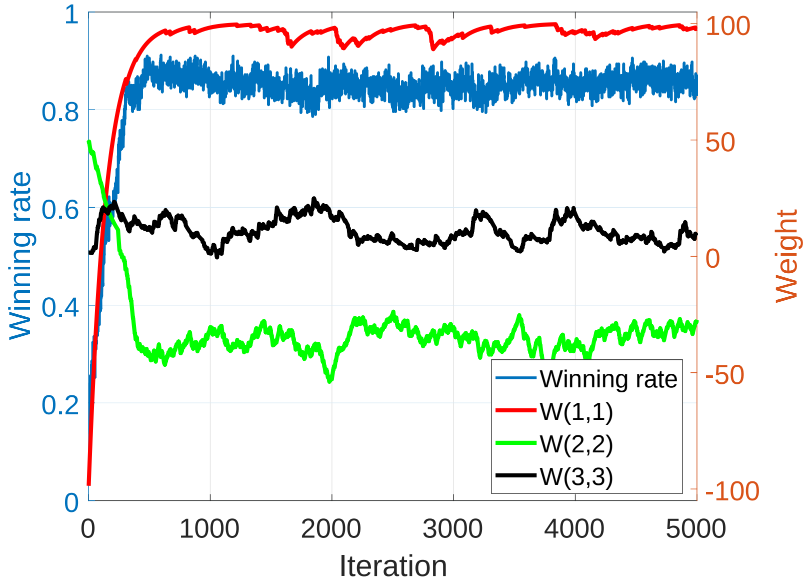


Figure 3.15: The variation of weight in the method 4 adapted dead zone(0.1) and decaying factor from figure 3.7. (1,1) position, (2,2) position, and (3,3) position are shown together in one graph.

the learning, the weight is initialized to 0, and the training set consists of opponents that perform random moves. Since the $\varepsilon - greedy$ algorithm is used, the agent also performs a purely random move with p = 0.1. In other cases, the agent chooses to maximize the estimated value. And also the learning rate $\alpha = 0.01$ was used for the learning and α decreasing by a factor of 0.95 every 45,000 games played.

We compared the results of learning using Posteriori through Bayes analysis and the results of learning through TDL (figure 3.16). It showed that learning was possible through a quite simple learning method, and although it was only a randomly move opponent, it showed a slightly better result than TDL. This confirms that a simple learning method can produce useful results.

3.3.2 Searching Algorithm

Previously, several search algorithms have been studied to proceed with the evaluation. The first one we had tried was the Mini-max algorithm, which we did not write in detail

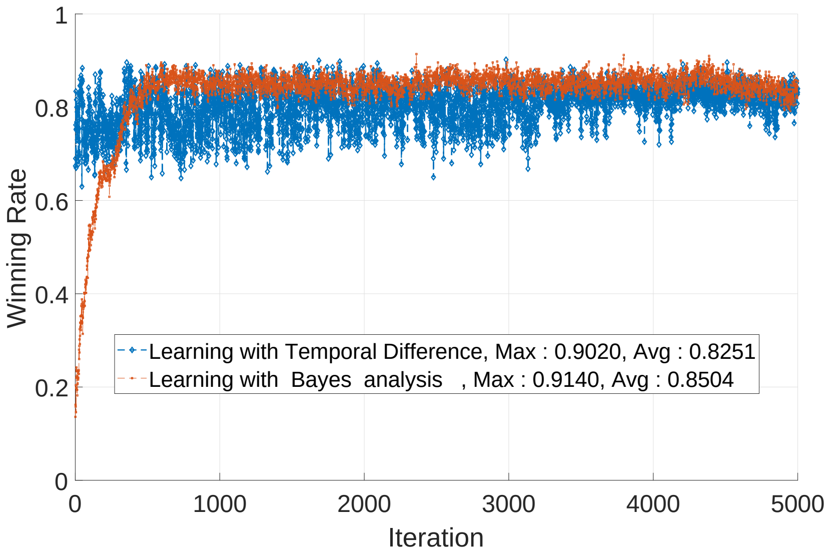


Figure 3.16: Comparison of TDL Learning Method and Bayes Analysis. Each method was learning with random opponents in the training phase, and then compared with the winning rate of 500 random weight opponents.

above in the paper, because it was difficult to see dramatic performance improvements when searching more depth. So instead of search algorithm, we chose the WPC method using learning weight map. When we used mini-max with this WPC method, it didn't shows effect for searching more depth. So we just searched 1 depth only (chose the best one we can select). It is believed that the weight value represents a value whether it is worth it. In this case, however, the progress of the winning rate was limited, and the MCTS algorithm was introduced to overcome these limitations. The MCTS, which is made by developing the mini-max algorithm further, is a concept that searches only a part not the entire segment. In the general MCTS algorithm, the weight map value is not used. In this paper, we modified the general MCTS method to use the learned information, and confirmed that the winning rate is further improved by combining the MCTS method and the position evaluation method.

3.4 Summary of Chapter 3

We wanted to know a concept of machine learning in simple way. To understand statistic model, using conditional probability, we approached likelihood and posteriori estimation model. We defined the game as a decision problem, and thought it as 'What humans think and how humans think'. We, human beings, calculate the probability fast enough. Sometimes it is called as 'insight'. It also consists of probability model what we think. Then how can modeling a probability model and consist of probability tree. With this probability, we have evaluated each position. The results that were figured out through several ways told us what model is proper, and how to model it. At this time, we could not get a good value from the probability of intuitively thinking and observing (likelihood model). Therefore, we tried to model this probability by Bayes' analysis using a posteriori value. As a result, we were able to achieve some reasonable winning rate. However, we needed to use a normalized value for each position, so we used a step function. In this process, we could obtain a winning rate of about 80%. Also to prevent the local minima problem, we needed decay model (Forget curve). To update the weight map, we needed to define the conditional probability using several methods. We thought it's a simple learning algorithm about Othello or decision problem games. We started with Othello but, it can expand to lots of other one. We thought this approach could be used for other learning model as well. And also we found out about searching algorithm using position evaluation data. This change has helped to make better AI and we thought it could be a good way to combine this method and applied it in other models as well.

Chapter 4

Solving decision problem using local mask filter

The criteria for decision are very important in solving problems. Because when you make good choices, you can solve the problem. In this section, we review the various features in solving these problems, and we want to see the possibility of selecting one of these features as a single criterion even if it is not a good known feature like WPC. This section is being prepared for submission as a scientific paper (Hahn, JeongWon and Kim, DaeEun, 2018c).

4.1 Experimental Setup

4.1.1 Feature Extraction

As before, we tried to find the weight value because we used a Weighted Piece Counter (WPC) method as evaluation function. WPC is a position based feature that uses the well-known characteristic of Othello, which means getting a special position means winning. WPC equation is as equation 2.14 in chapter 2. However, this time, we want to find more various features. We wanted not only Weighted Piece Counter method but also more diverse and intuitive features. First, we could gather the position information as we have done before. But, here, we want to see an depth information that is different from the previous one. It means we want to see additional information according to time. If we acquire a certain position at any point in time, we may be able to collect information that may increase or decrease the probability of winning. Another feature to look at is the feature that uses the number of flips. Because of the characteristics of the Othello game, it is necessary to flip the opponent's discs while playing the game, because it is thought that counting this number can also be a feature. It can be represented as the following equation:

$$\sum_{i,j=1}^{8}[B_{(i,j)} = Black|S_{t+1}] - \sum_{i,j=1}^{8}[B_{(i,j)} = Black|S_t] - 1 \tag{4.1}$$

where : S_{t+1} is after action a_t

For example, as followed in figure 4.1 (a) and (c), when a black disc is placed at point (2,4), it flips white discs at (3,4) and (4,4). So it flips 2 discs. As in the figure 4.1 (c) the summation of black discs is 7 and the number that appears in figure 4.1 (a) is 4. So when calculate $7-4$, it equals 3. But, we actually flipped only 2 discs. That is, we must compare the numbers except for the discs which was placed.

Also, another feature is the difference between the number of black discs and white discs. When action a_t is done in $State_t$, the difference($\sum BlackDiscs - \sum WhiteDiscs$) must be changed. So we will continue to keep track of this difference, to see how the end of the game is determined related to the difference is. It also can be represented as following equation (equation 4.2) :

$$\sum_{i,j=1}^{8}[B_{(i,j)} = Black|S_{t+1}] - \sum_{i,j=1}^{8}[B_{(i,j)} = White|S_{t+1}] \tag{4.2}$$

where: S_{t+1} is after action a_t

As shown in figure 4.1 (b) it has little placeable position and when a black disc is place at point (2,4), it becomes figure 4.1 (c). When placed at point (2,4), the difference is

(a)

1	1	1
1	1	1
1	1	1

(b)

0	0	0
1	1	1
0	0	0

(c)

0	1	0
0	1	0
0	1	0

(d)

0	1	0
1	1	1
0	1	0

(e)

1	0	1
0	1	0
1	0	1

Table 4.1: Example of kernel filter (3 x 3 Size)

$3(=7-4)$. When placed at point (3,2), it will be $5(=8-3)$.

These features provide various viewpoints in a situation where the game is not well known and make a situation where multiple judgments can be made from various points of view. From now we want to think another feature as the kernel filter. We can see that this is also a good feature if we can find out what the surrounding environment is by putting a small filter on the position that can be selected in $State_t$ and clarify the relationship between the confirmed result and the win / loss. This kernel filter method can proceed 3x3 size or 5x5 size. When using this we can create a lot of filters ($(filtersize)^3$) in progress. These kernel filters can be calculated as following equation:

$$f(x,y) = \sum_{i,j=(-\frac{FilterSize}{2})}^{(\frac{FilterSize}{2})} B_{(x+i,y+j)} \cdot Filter_{(i,j)}, (1 <= x+i, y+j <= 8) \qquad (4.3)$$

When: $Board(i,j) = Black$ will $+1$ and $Board(i,j) = White$ will -1

For example when 3x3 kernel filter is put at position (2,4) like figure 4.2 (a) with table 4.1 (a) filter, the result will be -1. But although it is the same position with table 4.1 (b) filter, it becomes $+1$. Using table 4.1 (d) filter, it becomes 0. Same as other position, when filter is put at position (3,2) with table 4.1 (a) filter, it becomes -1 and also using (b) or (c) filter, it becomes -1

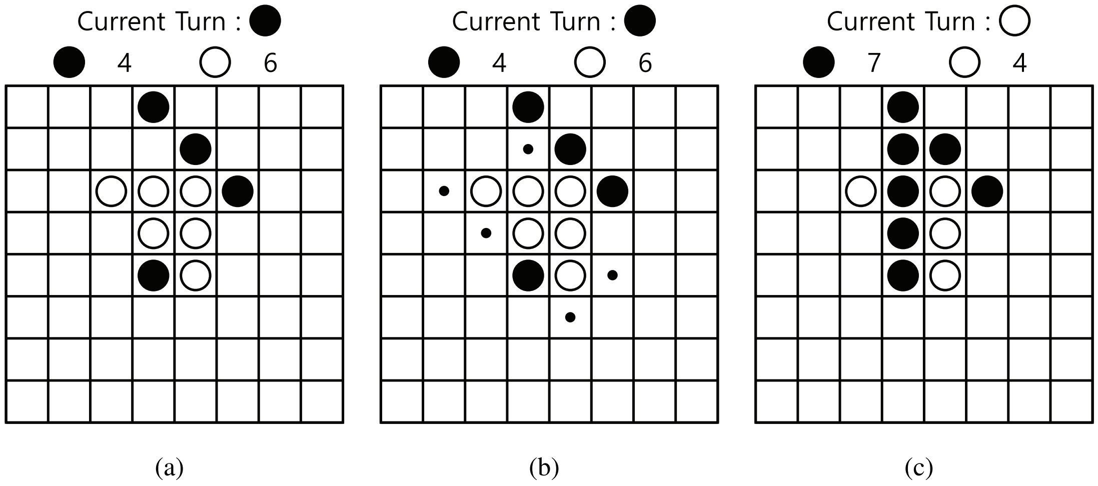


Figure 4.1: (a) Board State = S_t, (b) placeable position when S_t (c) Board State = S_{t+1}

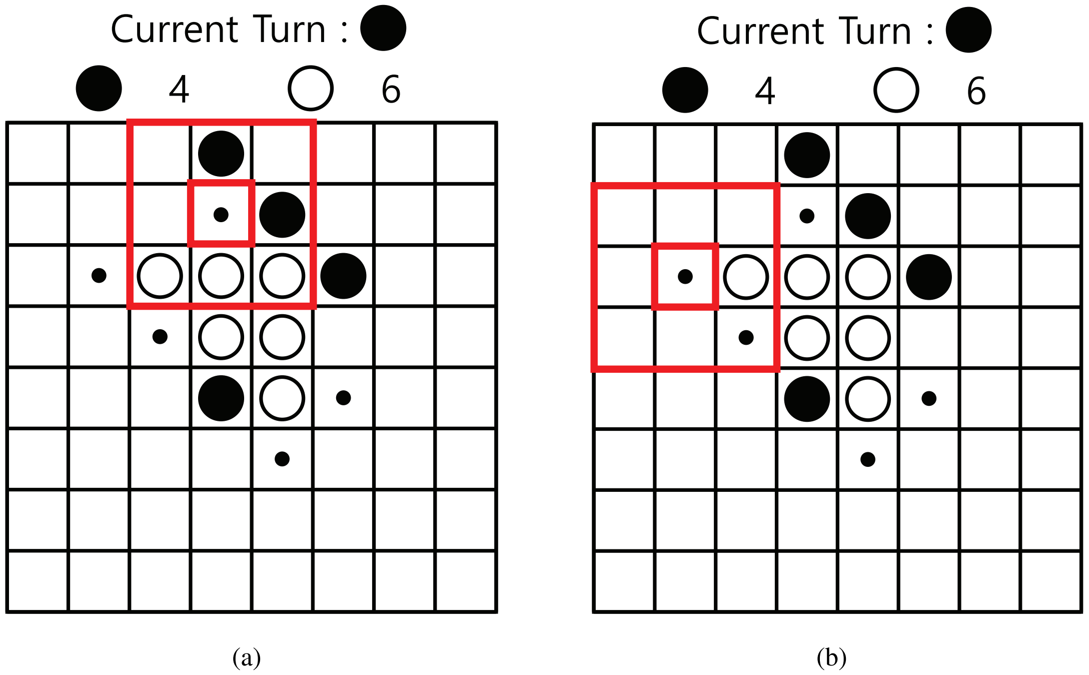


Figure 4.2: (a) Filter with (2,4) position when Board State = S_t (from figure 4.1 (b)), (b) Filter with (3,2) position when Board State = S_t

4.1.2 Evaluate and Learning

In this section, we want to talk about how to estimate the features. In fact, it is a good estimation that using winning rate can control the result of the game. As you know, it is intuition that when using a good feature, winning rate is also good. So, we think we can say it is a good feature or not by calculating winning rate. If it the winning rate increases, as compared with before, it can be called a good feature, and reversibly, it can't be said to be a good feature. However, if you only have the method above that simply use, it is a resultative story, you don't know how to make a choice and how the simulation will be done. Also, we think various evaluation methods, find out that what kind of estimation algorithm has relationship with winning rate. Thus, we propose the first one as below. If the contrast of victory or defeat is certain, it is expected to be a good evaluation method. For example, assume that if you gain position (1,1), you win with 90% probability and lose with 10% probability. In this case, 90%-10% = 80% would be a great contrast, which would be an evaluation criterion. It is thought that it can be evaluated based on how much the contrast point is compared with the whole point. Another method of evaluation may be measuring the uncertainty of a random variable. It is possible to measure the current uncertainty by calculating the time needed to determine a certain value in the situation of uncertainty of victory or defeat. It is similar to the entropy concept but has a different representation. In this case, it is thought that it is possible to measure the amount of the value leading to victory or defeat. The same concept can be applied to the depth measuring the number. However, since these games use reinforcement learning among unsupervised learning methods, it is hard to say that it is a good feature until we see the evaluation process using the feature and the result. Therefore, we compare the method above with the actual evaluation and $P(Win)$ to distinguish between good and bad features, and will use these features for learning.

4.1.3 Simulation with MCTS

In this paper, we will perform simulation using MCTS algorithm. Using the MCTS method has the potential for future development of this experiment and also allows instant evaluation of features during the game. However, the MCTS method described above can only see the final win / lose result at the end of the game. In the early and

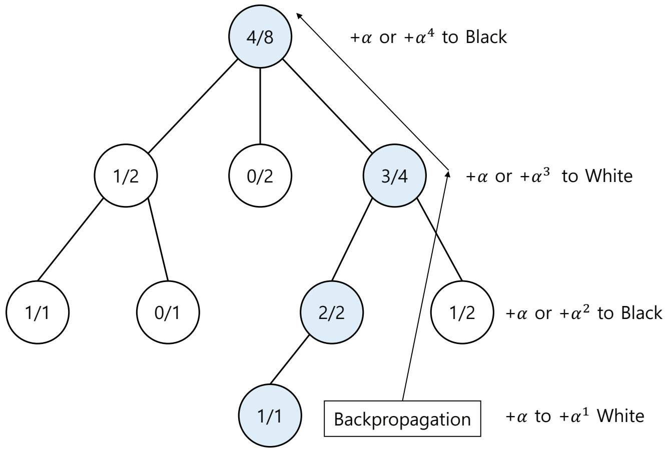

4/8
$+\alpha$ or $+\alpha^4$ to Black
1/2
0/2
3/4
$+\alpha$ or $+\alpha^3$ to White
1/1
0/1
2/2
1/2
$+\alpha$ or $+\alpha^2$ to Black
1/1
Backpropagation
$+\alpha$ to $+\alpha^1$ White

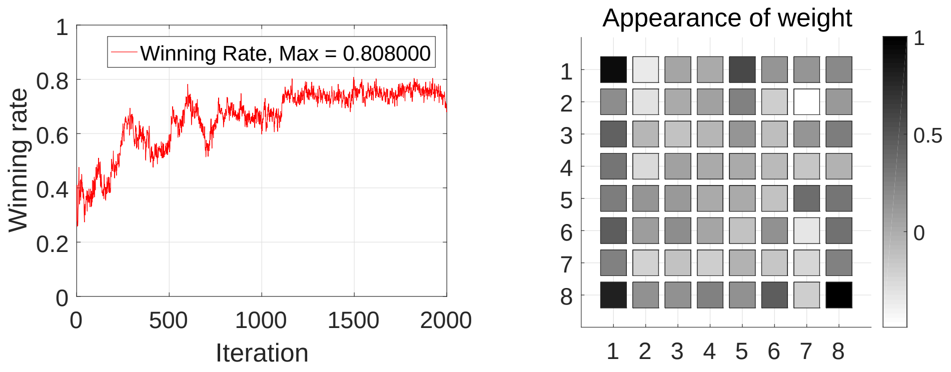
Figure 4.3: Backpropagation of state evaluation using MCTS

middle of the game, it depends on the simulation information. This leads to a problem that in the simulation, the game ends with a victory. But in the real game, a defeat occurs. This is due to the fact that the game will end at the point where the result of the actual game can be evaluated. Therefore, in the UCT method mentioned above, the term reflecting the simulation information and the result of the final game is separately set and reflected. It is possible to learn by using this information because it solves the part that depends only on the simulation result and puts the final result into account by putting the term of this much. We want to use this modified UCT method as we reviewed in the previous chapter. We want to do a learning by using a term for the result part of the actual game. This is like setting a reward in another reinforcement learning, just like learning without decay over time. So, in this time, using this algorithm, we want to simulate the state via various features. Although using the simulate function we need to apply the win or lose information of the real games. In this paper, we are going to conduct feature extraction and learn using MCTS. Unlike the game in which only the position information is collected in the existing single game, the MCTS is used to evaluate the features using a lot of information (the methods shown above) in the middle. Also, by using MCTS, back-propagation by constructing a search tree in a game is easy. In addition, since it is possible to know the next state compared to the existing method, it can be expanded to the form of reinforcement learning in the future.

4.2 Experimental Result

4.2.1 Verification Method

Before proceeding with the experiment, we reviewed the method of using information on other positions to evaluate the usability of the method above. Previously, we used a posteriori estimation model for learning when using the WPC evaluation method. Thus, in order to compare with the information above, we first investigated the possibility of the method above by collecting position information while searching in MCTS. When we collected position information like existing models, we checked how much the winning rate could be increased. In the existing method, about 91% of the 500 opponents won, and the result of current method that acquired the position value is as figure 4.4. It is a result different from the existing method, but it was able to see a similar figure to some extent. Winning rate per iteration and the configuration of weight map are shown in figure 4.4. Although it was not as good as before, we confirmed that it was increased to some extent. We also tried several ways to proceed with this verification. In the backpropagation process, several attempts have been made to achieve similar effect to the Q value decrease caused by the node moves from the Q-Learning to the root node. The method of giving ±1 points to all nodes, the method of multiplying by α as the node goes to the root, the method of Backpropagation to root, and the method of reflecting the data up to the node that starts the simulation. In this way, we proceeded with the review in four cases. The result of figure 4.4 above is completed with ±1 point to node until the simulation site, but we think that we could get better results by adjusting α and reflecting data up to the root. The above concept is shown in figure 4.3. The backpropagation value depends on the value of $+\alpha$ written on the right side. By adjusting this value, you can get smaller or larger values as you go to root. In this case, the backpropagation using α or the backpropagation to the root could reduce the winning rate even more, because it is necessary to turn the value of



(a)

(b)

Figure 4.4: (a) Winning rate graph with 2000 iteration, (b) Weight map when winning rate is 80.8%

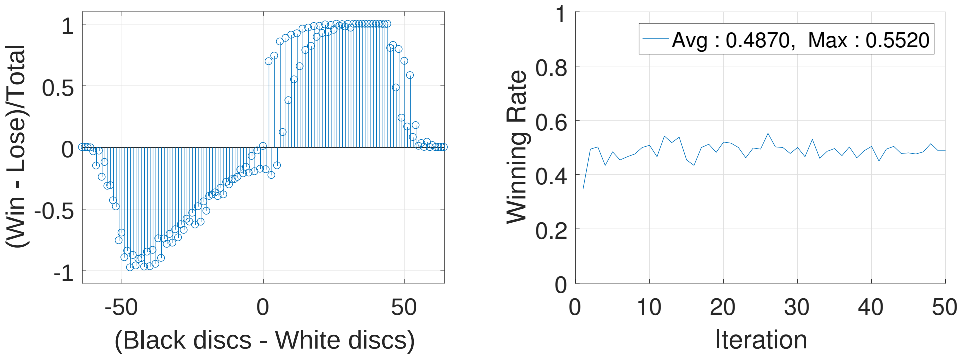
α. However, we have not tuned α because we are primarily interested in reviewing MCTS method and feature extraction.

4.2.2 Comparing Feature Extraction

In this section, we will evaluate each of the various features listed above. In the previous step, we confirmed that learning can be done similarly to the conventional method even if the position is simply collected. Therefore, we will try to collect the results using various actual features. First of all, evaluation of each feature was done in two ways. We examined how the winner or loser of the entire feature can be clearly revealed, and how the winning rate can be increased when the feature is used in two ways. In order to evaluate the winning ratio, we can proceed with the game choosing action(a_t) based on the following equation, and evaluate the winning rate using this result.

$$\underset{a_t}{argmax}\, E_{(S_t,a_t)},\quad E_{(S_t,a_t)} = f(x),\quad x \quad equal \quad equation\ 4.1, 4.2 or 4.3$$

$$from\ equation\ 4.2,\quad x = (\sum_{i,j=1}^{8}[B_{(i,j)} = Black|S_{t+1}] - \sum_{i,j=1}^{8}[B_{(i,j)} = White|S_{t+1}]) \tag{4.4}$$

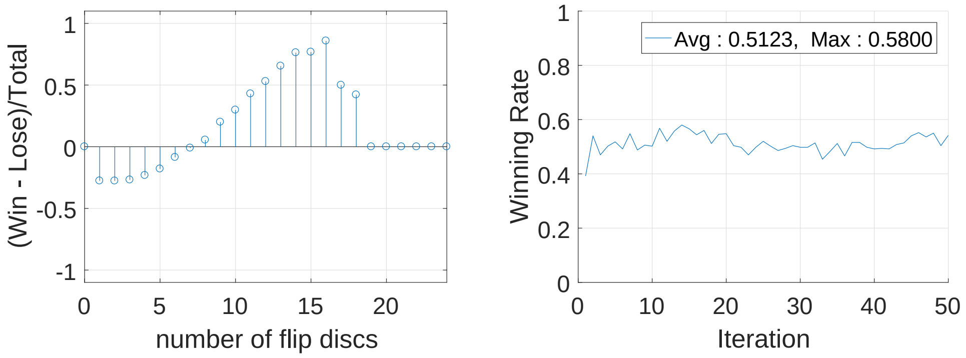


(a) (b)

Figure 4.5: (a) shows the probability of victory or defeat according to the number of discs with difference expected, and (b) shows the winning rate with 500 random opponents when constructing AI using it.

Among the various features, we evaluated the difference first. We use $\sum Black\ discs - \sum White\ discs$ to examine the differences between the two values to see if they can win. The difference can be a maximum of +64 difference (perfect win) or -64 difference (perfect defeat). Therefore, the information using these difference values was obtained. However, in the present process, we did not collect depth information(the order of progress of the game), which is information about time, and we collected only the number of differences that determine winning or losing using equation 4.2. Figure 4.5 shows such information. Figure 4.5 (a) shows the probability of victory or defeat according to the number of discs with differences expected, and (b) shows the winning rate with 500 random opponents when constructing AI using it. Now we want to evaluate the features. First, there are 129 feature points in total, 121 feature points (except for 8 points) of these feature points represent win and loss respectively. This means that 93.8% of the features determine the win or loss ratio. However, when the actual winning rate is evaluated, it is about 50%. When you talk about winning percentage only, this is a 50 point feature. If we use the existing position information and think that the winning rate has risen to about 8-90%, it will not be a good feature.

Next, let's look at the 'flip' feature. Flip means the number that can be flipped at a time,



(a) (b)

Figure 4.6: (a) shows the probability of victory or defeat according to the expected number of discs flipped, and (b) shows the winning rate with 500 random opponents when constructing AI using it.

and we want to use equation 4.1 above. The difference is the problem of using S_{t+1} which changes when action a_t is executed, but Flip is S_{t+1} and S_t. The number of flips that can be done at a time is also determined, so we will use it to find out how many flips you can choose when you win or lose. At this time, the maximum number of possible flips is 24, which is the sum of both up and down, right and left, and diagonal line. This is the same as the above Difference method. The number of possible flips is from 1 to 24, and a total of 24 feature points can be extracted. In this case, figure 4.6 shows that there are 18 total numbers that affect the actual winning percentage, which is 75%. However, this method is considered as a better evaluation to look for actual winning rate as in the previous method. Therefore, the actual winning rate is as shown in figure 4.6 (b). It is a better feature because it has a maximum winning rate of 58%, and the average value is also slightly higher than in figure 4.5. However, this is also not a good feature compared to the previous method(using position).

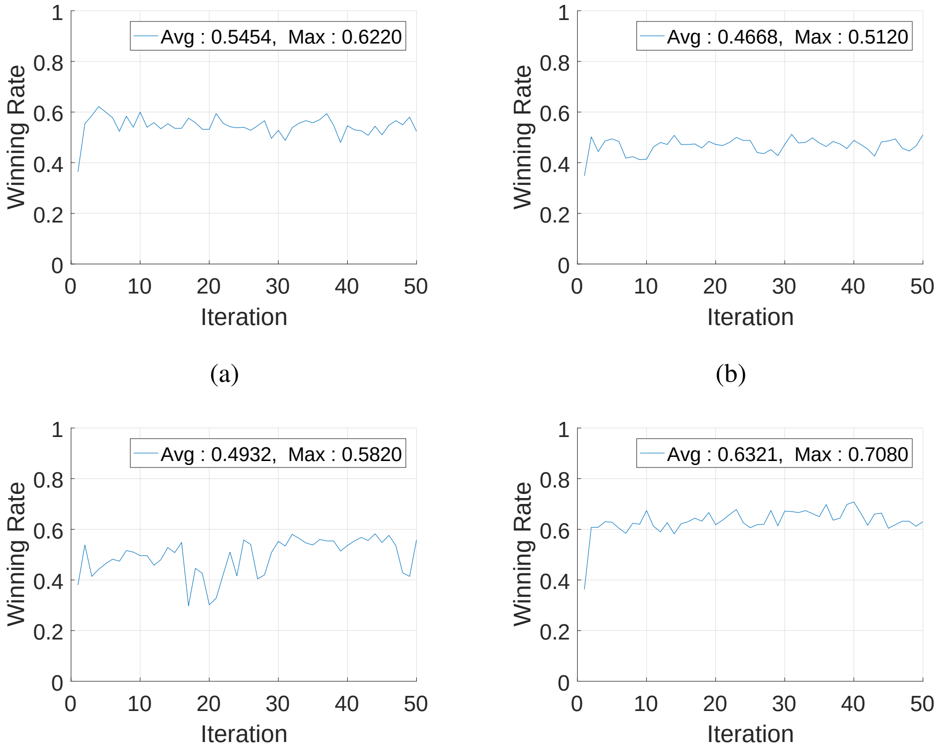
The above two methods, as easily thought, showed some limits. This is because the winning rate did not rise to 8,90% similar to the previous position information. This time, we tried to use Convolution Filter by other methods that do not use existing position information. Convolution filters have 3 x 3 and 5 x 5 cases. In this paper, the

(a)					(b)					(c)					(d)				
1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
1	1	1	1	1	1	1	1	1	1	0	0	1	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0

Table 4.2: The type of filter used in the experiment (5 x 5 Size)

experiment was conducted based on 5 x 5. It was because a larger filter is likely to have higher winning rate due to more information. It has 5 x 5 filter can contain 3 x 3, but not vice versa. The experiment was carried out in the same way as the two cases above. The value of each filter has three values of -1, 0, and +1. As in the evaluation method above, we evaluated the kinds of values that can be obtained for each case and how reliable the contrast is. Figure 4.7 shows the results of using the filters shown in Table 4.2 (a), (b), (c), and (d). Only two of the results above, good and bad, show feature function in figure 4.8. Table 4.3 summarizes the contents of each case. The result of the experiment is as figure 4.7. When each filter was used, the average of winning rate was 46.7 % at the lowest and 63.2 % at the highest. In the two cases above, the result of checking each value is the same as figure 4.8. Each feature point had $\frac{9}{11}$, $\frac{17}{19}$. The number of such feature points does not correlate with the higher results in terms of the number of feature points. Therefore, we think that the direction of comparison with the winning rate is the right direction, and some of the results we have seen so far are the same as table 4.3.

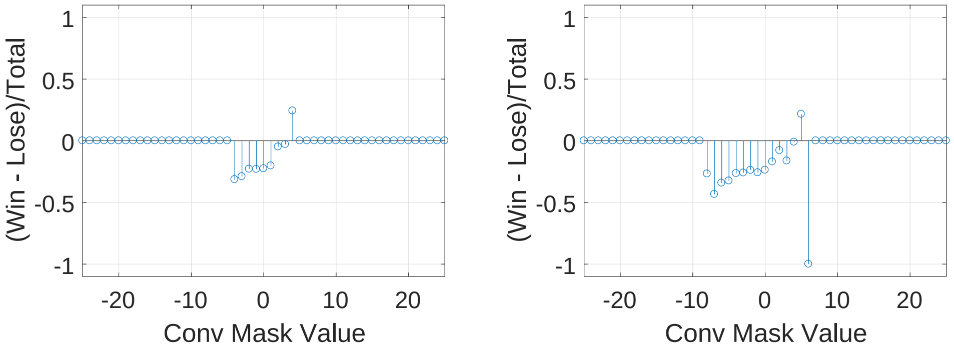
These results show many open results for various features. By using more features, we show the possibility to make evaluation function as different features. We will use this point to make more 5 x 5 filters and for further experiment. There are 5,000 5 x 5 filter types that are generated randomly to avoid duplication. As mentioned above, each filter has a value of -1, 0, and +1, and there are a total of 3^{25} types that can be created without duplication. This can generate 847,288,609,443 branches, and only a



(a) (b)

(c) (d)

Figure 4.7: Winning rate that are using table 4.2 (a),(b),(c),(d). In table 4.3 shows summarized result.



(a) (b)

Figure 4.8: Feature point that is using (a) [table 4.2 (b)] - Worst case, (b) [table 4.2 (d)] - best case.

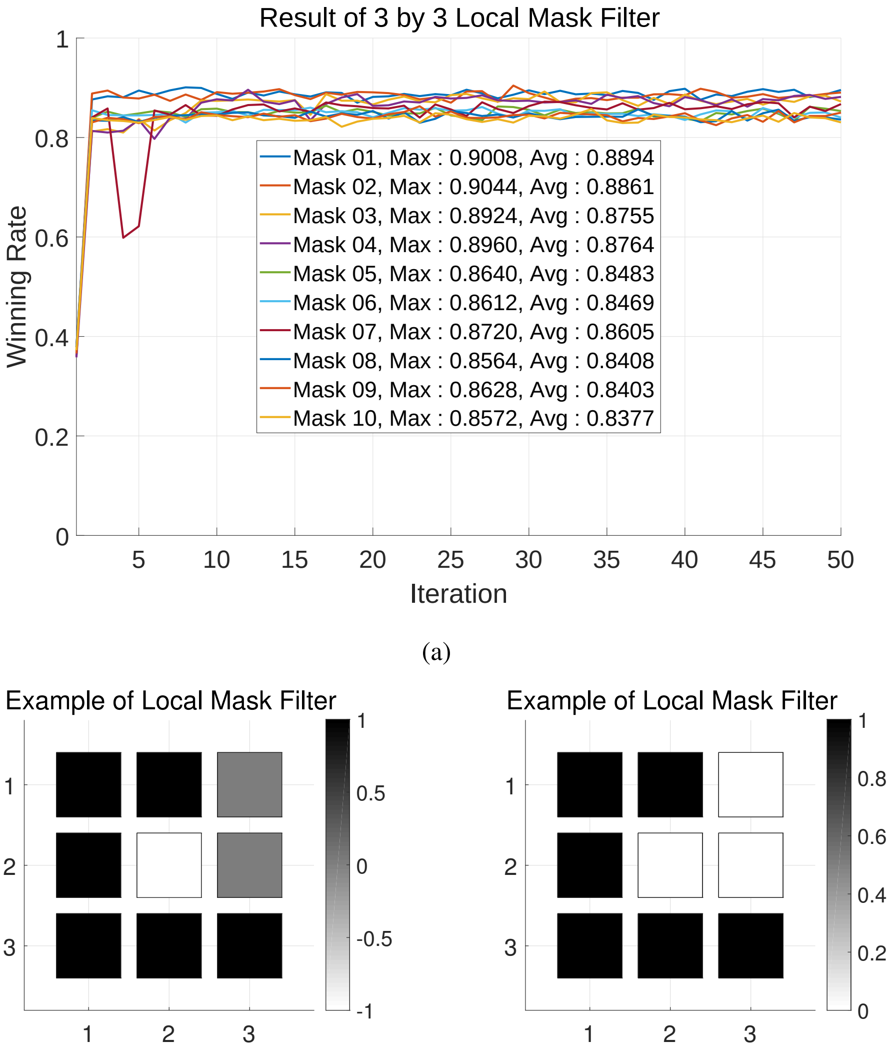
Evaluate Condition	Feature Point	Winning Rate(avg)	Winning Rate(max)
Number of difference	121/129(93.8%)	48.7%	55.2%
Number of flips	18/24(75%)	51.2%	58.0%
Mask From Table (a)	44/51(86.3%)	54.5%	62.2%
Mask From Table (b)	9/11(81.8%)	46.7%	51.2%
Mask From Table (c)	9/11(81.8%)	49.3%	58.2%
Mask From Table (d)	17/19(89.5%)	63.2%	70.8%

Table 4.3: Evaluation results by filter type

few of them show how the winning rate can go up.

We evaluated each of the 5,000 generated filters and obtained the top 10 of them. In this case, the highest peak was 90.8%, and the average peak was 89.5%. Looking at the top 10, we found that there was a feature with somewhat higher winning rate. In the same way, if we look at the bottom 10, the average of 18.1% was the lowest, and were between 18% and 27%. Therefore, it can be confirmed that there is a change in the winning rate depending on selected features, and when the winning rate curve is drawn, the figure 4.9, figure 4.10, and figure 4.11 are shown. We compared filter sizes from 3 by 3 to 7 by 7. As a result, 3 by 3 was more stable than 7 by 7. Rather, it seems to be because the more peripheral information in 7 by 7 lead to a vaguer decision. That is, a consistent choice seemed to make better results. Based on these results, we will use 5 by 5 size in the next trial.

In the case above, the best feature point and mask configuration as in each figure (b) and (c) in figure 4.9, figure 4.10, or figure 4.11. These results show that even if the agent selects based on local view only, there is some possibility to solve the problem. Here, the instantaneous winning rate of 90.8% is similar as previous trial 91.2%.



(a)

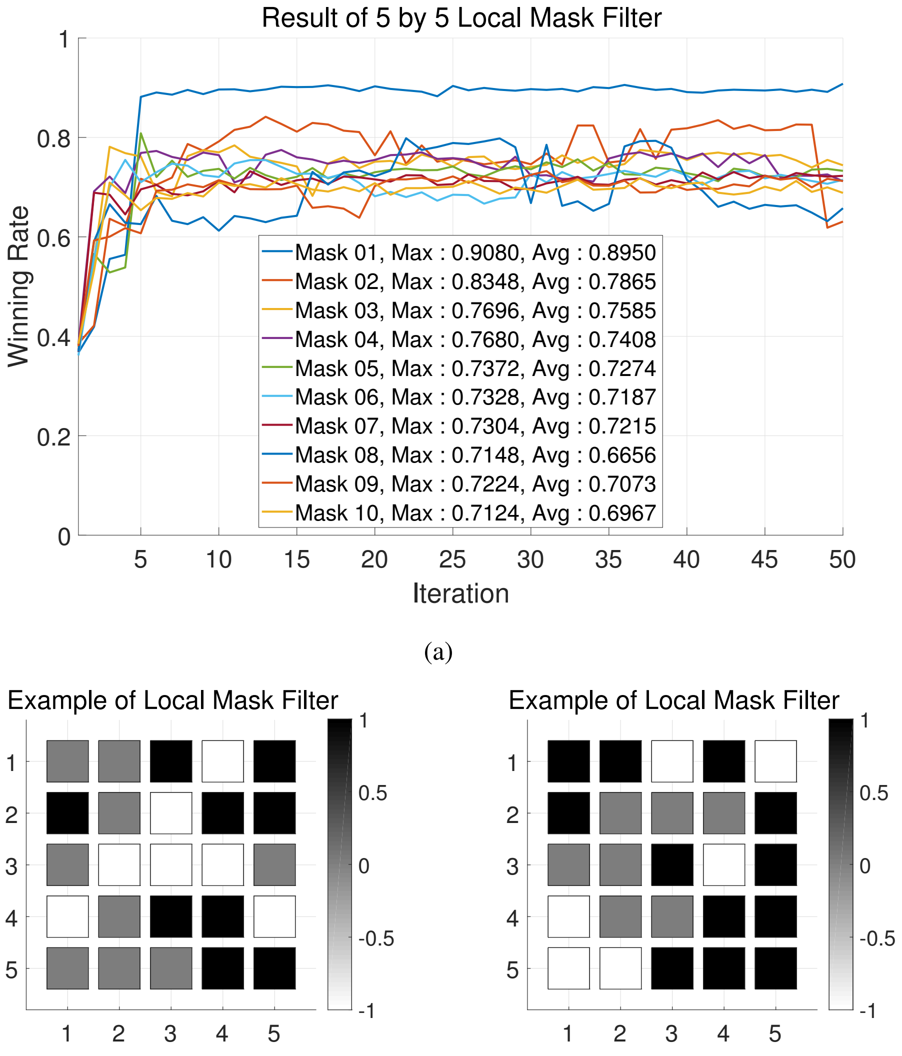
(b)

(c)

Figure 4.9: Best 10 winning rate with 5,000 randomly generated 3 by 3 local mask filter

4.2.3 Multiple Feature Combination

If so, this section will try to figure out how to increase the winning rate by using the various features found above. We will use the MCTS method to evaluate only the 20 most successful mask filters above. Among the above 20 evaluation methods, more than 50% of the methods are expected to be used when the MCTS method is applied and only 20 of the mask filters obtained above are used. If the node is victorious, the other case is evaluated as defeat, backpropagation is performed, and the result is gathered and the MCTS can be configured by selecting the next node. Evaluating



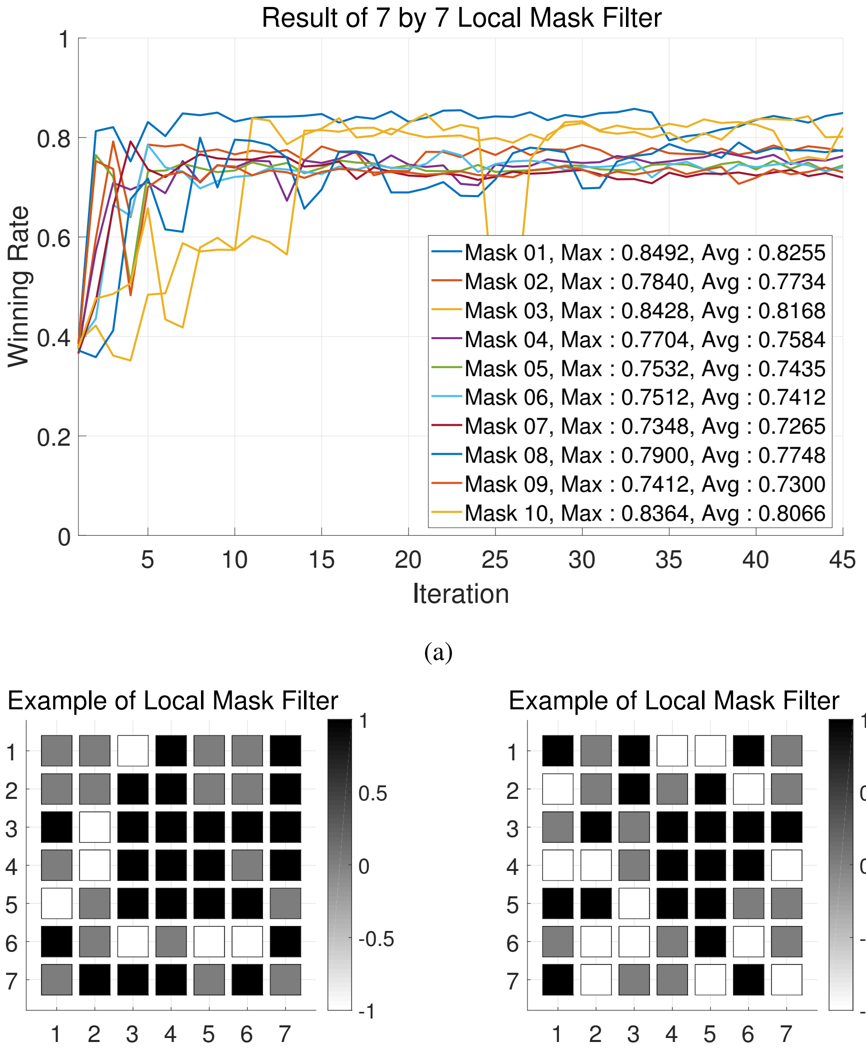
(a)

(b)

(c)

Figure 4.10: Best 10 winning rate with 5,000 randomly generated 5 by 5 local mask filter

these various features on the basis of playing games means viewing the world with various viewpoints, which makes another development possible. It can be expected that if the evaluation is conducted through this method, the winning rate will increase. Reviewing these possibilities using these individual features, we thought that these individual features would be a guide to the value of an action. Thus, if you have more than two guides, we thought that the possibility might be higher. To learn the value of individual features, we use the following equation 4.5. We simply tried to update it based on the possibility of winning through one feature.



(a)

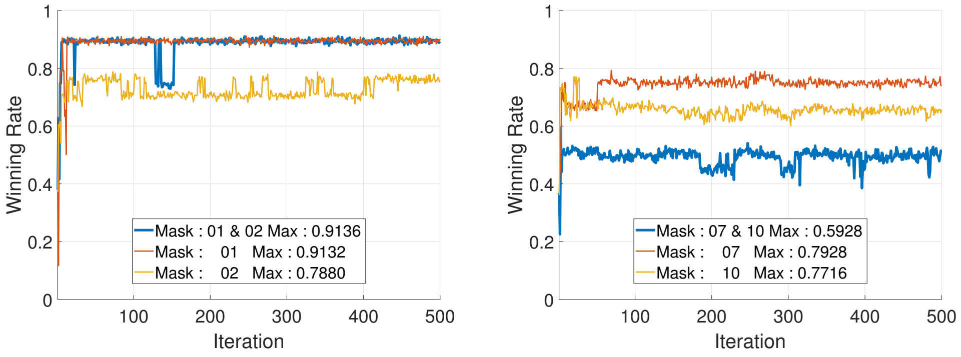
(b)

(c)

Figure 4.11: Best 10 winning rate with 5,000 randomly generated 7 by 7 local mask filter

$$Eval(f(x,y)) = (1-\alpha)Eval(f(x,y)) + \alpha(P(Win|Feature[i])) \tag{4.5}$$

We used two updated features to process learning, and checked the results. We expected to pass through two weak classifiers to get better results, but it was not. Of course, there were cases where better results were produced, but there were cases where the results were worse too. It is thought that aliasing problems will exist in a considerable number of cases because the view through the mask feature only uses



(a)

(b)

Figure 4.12: The result of combining two features with the sum of each probability. (a) represents a successful case, and (b) represents a case of failure. Each result that verse with 2500 opponents using 0.025 learning rate.

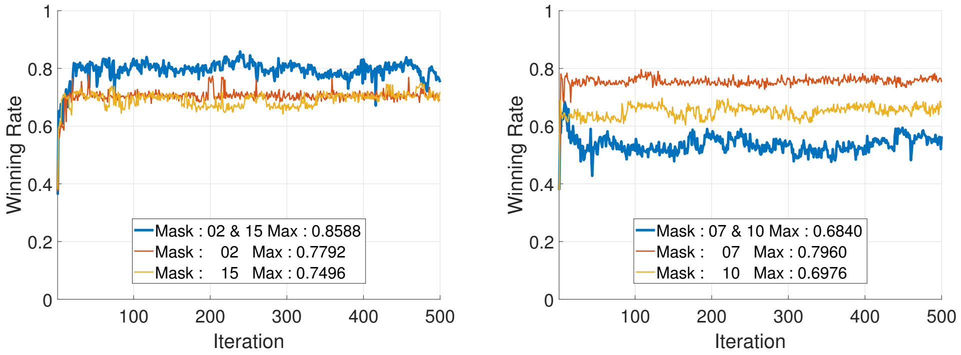
local results. If you use only one feature, you get a good result with consistent selection, but when you use two or more features, a problem is caused by the difference of direction of features. That is, assuming that there are several branches when you are looking for a way, one feature will continue to talk left, and the other feature will continue to talk right. In this case, although the left or the right is the desired destination, the problem is not being able to arrive at the final destination because of the continued turning over between the left and right. Thus, if you have a local view (similar view) that overlaps a feature, you get better results, but if not, you only get worse results.

$$Eval(s_t) = Eval(f(x,y), feature[1]) + Eval(f(x,y), feature[2]) \tag{4.6}$$

We think that the problem is caused by simply summing the probabilities here. Therefore, we tried experimenting using Naïve Bayes classifier. That is, the result of the calculation by equation 4.7 is as follows (Zhang, 2004).

$$P(Win|E[1],E[2]) = \frac{ProbWin}{ProbWin + ProbLose} \tag{4.7}$$

Where:



(a) (b)

Figure 4.13: Result using Multi feature simulation method using equation 4.7. The result verse with 2500 opponents using 0.025 learning rate.

- $ProbWin = P(E[1]|Win) \times P(E[2]|Win) \times P(Win)$
- $ProbLose = P(E[1]|Lose) \times P(E[2]|Lose) \times P(Lose)$

But still, some case makes it better, but sometimes it gets worse. This is thought to be the limit to process with a local view, which is not a representation of the state, but rather an expression of the value of the action. Finally, as an additional verification, we tried to use all 20 of these features. We tried voting with just 20 features, but it only showed worse results. We wanted to simulate MCTS using algorithm 4 above. When we did this, we thought that the winning percentage would increase a lot and proceeded with the experiment. However, when we looked the actual results, we could confirm it was bad (figure 4.14). The results of reviewing 20 features individually showed that the winning rate increased considerably. However, the result of reviewing 20 of them at once to simulate a victory over 10 and a defeat of 10 or less was disappointing. We expected an increase in the winning rate, but the result was only in place, which we think is because if each feature is good in one part, it may be bad in the other part. That is, each feature is thought to be the sum of biased information. There may be situations in which 20 features have a common good value, and some features exhibit fairly good values, but some features may exhibit bad values. Or from another point of view, the MCTS simply compresses better or worse information into winning or

Algorithm 4 Multi feature simulation

```
function EVALUATE BOARD(Board, feature, N)
    Let $N$ be the number of features.
    Let $feature$ be extracted feature[1 ... N]
    Let $Board$ be current state of Board.
    Let $Eval$ be the number of features that have a probability of winning.
    for $i = 1toN$ do
        $Tmp = Evaluate(Board, feature[i])$
        if $Tmp$ is positive then
            $Eval = Eval + 1$
        end if
    end for
    if $Eval$ is bigger than $N/2$ then
        return $SimulationWin$.
    else
        return $SimulationLose$.
    end if
end function
```

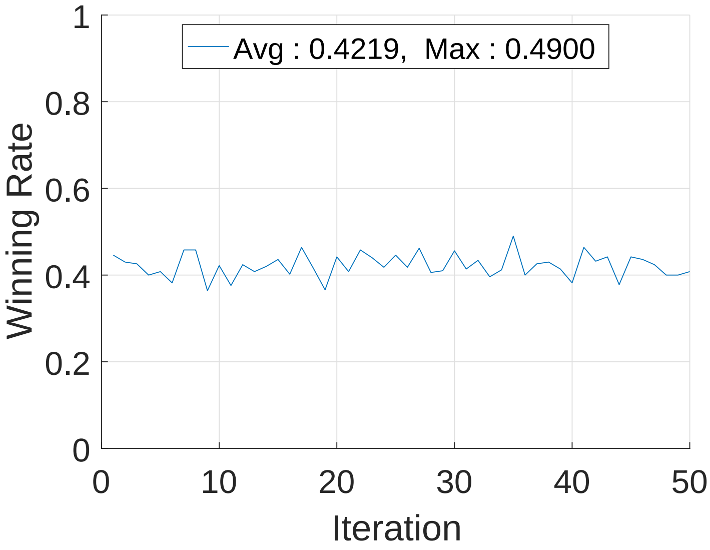


Figure 4.14: Result using Multi feature simulation method using algorithm 4

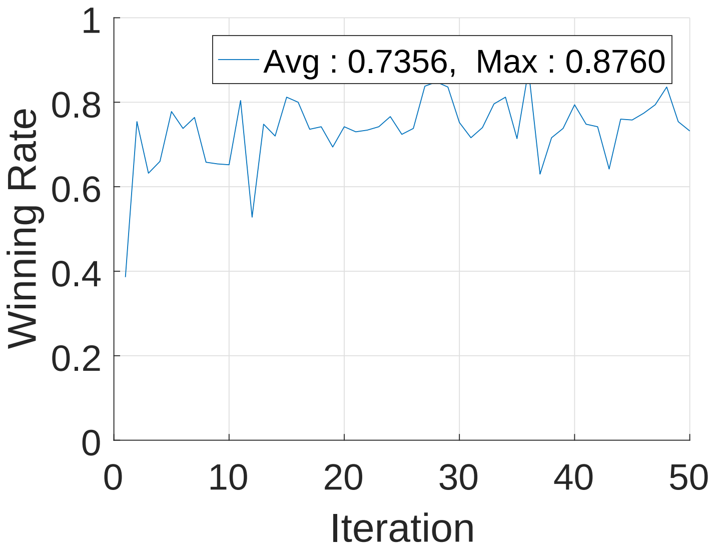


Figure 4.15: Result using Multi feature simulation method, this time, 20 features will be voted and the best action will be selected.

losing information. As seen in the previous co-evolution scheme and the combination of MCTS, the MCTS has compressed information into victory or defeat with each selected activity (a_t) better or worse (future value). So, through the MCTS, it looks like the same chance to win, but in reality, it is not the same victory. It is actually an action that is likely to win more. When evaluating using the algorithm 4 presented above, we can get the same result as figure 4.14, which seems to not have been learned for victory at all. At this time, if the evaluation is performed using the same individual features, there is a slight difference in each feature but an average of 6-70% is obtained.

$$Eval(action) = \sum_{i=1}^{N} Evaluate(Board, feature[i]) \tag{4.8}$$

Then, rather than compressing only the information of victory or defeat as in the MCTS method, we examined what would happen if we chose the probability that has a greater probability individually. That is, we set the value for each $Q(S_t, a_t)$ and proceed to the selection with the largest Q value. This large value is produced from how 20 features are voted on an action and the action that receives the most selected vote. However,

Methods	Winning Rate(Max)
Bayesian probability	91.2%
Local Mask Filter Best in 3 by 3	89.8%
Local Mask Filter Best in 5 by 5	90.8%
Local Mask Filter Best in 7 by 7	84.9%
Combination Mask Filter	91.3%
Combined 20 features with voting algorithm	47.4%
Combined 20 features with normalized summation	87.6%

Table 4.4: Summary combined 20 features result and each convolution mask

doing the voting did not yield good results. As in the case of the MCTS method, the winning percentage remained at 40%s. When we thought about why, We could make the following conclusion. As with the MCTS method, this voting will have a better choice and a worse choice in the choices available for each feature. However, since these options are compressed with victory or defeat in the feature, it is thought that the effect of compressing the probability of features into one is the same as MCTS method. Therefore, we want to normalize the feature point map of each feature according to +1 or -1. In this way, we tried to normalize the feature point map from +1 to -1. We checked that the winning rate rises to some extent through this method. If you see figure 4.15, you can still see the slightest shaking, but you can see the maximum probability of winning rising to 87.6%. This compresses the current situation with only information of victory or defeat in the MCTS system. We can see it as a proof of my thought about compressing the data when using MCTS algorithm. However, even if this is done, it seems that sometimes there is a time when the biased information is generated and the winning percentage changes greatly. This may be because each feature is changing its probability as it changes too much. In other words, it is thought that better results can be obtained by performing filter processing like average filter while accumulating feature statistics.

4.3 Summary of Chapter 4

Through this experiment, we can see that only position information is not a good feature to decide winning and losing of the game. We extracted features by various methods such as viewing the difference of discs rather than position information, performing action (a_t), viewing the number of flips, and using a convolution filter. We were able to know how to extract various features and how to evaluate them, and we were able to try simple learning using them. In addition, we did not simply use only one feature to learn and evaluate, but we looked at the process of creating a better feature by selectively linking various features. In this process, we examined how to select a feature and statistically collected the data and classified it as a feature. The result is the same as table 4.4. By analyzing the Bayesian probability, we were able to achieve a win rate of about 91.4% from chapter 3.. The maximum winning percentage of a single convolution mask was 90.8 %, which was a very encouraging result. In addition, when 20 pieces of features were combined, it does not rise to 90 % when it is tested. There may have been various reasons, but one reason was that each feature can fluctuate as the learning progresses. This was well documented in the figure 4.10, and it is expected that the resulting rate will not increase due to the problem of voting process. In the existing process, only the position evaluation method was used. This is not a more complex and diverse learning method but rather a process that starts from the idea of trying to solve the problem more intuitively. However, this time, we tried to look at one state using various viewpoints. This could be regarded as a fundamental process that goes to deep learning. It could be seen that the process of observing one phenomenon through various features and deriving the best conclusion through weighted combination of features is the same as Convolutional Neural Network. It was a very important way to look at various aspects, rather than to use a single feature, which could be extended to a wider sense rather than being limited to Othello game. For example, we thought it would be possible to do this feature extraction and evaluation, which is currently used in games such as Gomoku and Go. We think this method can also be a reinforcement learning. In this paper, we aimed to examine whether the feature extraction can be done through this method and the problem can be viewed from various viewpoints. In the meantime, we tried to examine how to make more accurate judgements when the perspectives are mixed in various ways. As a result, a simple sum is possible to some extent, but more accurate method such as weighted summation is needed.

Chapter 5

Reinforcement Learning: Application in Othello

In this chapter, we will look at reinforcement learning and examples from Othello. Although reinforcement learning has been studied since the 1980s (Sutton and Barto, 1998), it has not been studied for a while due to lack of data and limitations in computation. However, in recent years, it has been an innovative field and its driving force is the development of big data and deep learning algorithms. The basic idea of these various deep learning methods is reinforcement learning, which means reinforcement learning to enhance certain behaviors. For example, suppose you have an environment with levers and mice in a box. When the mouse pulls the lever, the feed comes out. At this time, the mouse moves around the box, and if the mouse pull the lever accidentally, the mouse will get food. In this case, if the mouse pull the lever, the mouse knows that the food is coming out, and the mouse remembers the way to the lever. The mouse gains compensation by pulling this lever. With these rewards, the mouse will pull this lever whenever the mouse is hungry, and in this case we can say, "The mouse learned the usage of the lever." Repeatedly doing this pulling of the lever will make the reward more certain, which is referred to as reinforcement, and this learning method is reinforcement learning. Thus, in this chapter, we will study

TD-Learning and Q-Learning, which are the basis of this technology, and apply it to game of Othello. This section is being prepared for submission as a scientific paper (Hahn, JeongWon and Kim, DaeEun, 2018b).

5.1 Experimental Setup

5.1.1 State Represent Function

Reinforcement Learning is briefly introduced in chapter 2. However, this reinforcement learning requires state representations. Because without representations all the system has too many state. For example, simply in game of Othello, it is an 8 by 8 size board and there are three type of discs: black disc, white disc, and empty board. So if we want to use this state, we need 3^{64} state. Of course, in the actual Othello game, the number of states is greatly reduced because of the symmetrical nature of the horizontally and vertically, and the placed discs is expanding. However, even so, there are too many cases to represent all these states case-by-case. Therefore, it is necessary to express this state as a single scalar value. Even if it is not such an Othello game, it is the same with other games. In the case of board games like Chess or Go, there are many more cases. For example in Chess, the size of the board is 8 by 8 like Othello, but it has much more cases(13^n) than Othello (3^n). Also, even if it is not such a board game, there are many cases where we need to express the situation as a state in real life. Sometimes it is necessary to check the current situation for a specific choice. Such an act of checking itself is an act of looking for a certain value, ie, if there has ever been a similar experience in the experience. This new representation of the state makes it possible to compress similar situations into one case.

We extend the existing study and try to use the WPC method to express this state. In the past, the value obtain through WPC was used as the evaluated value, but this time, now we want to use it as a scalar value expressing the state simply. In case of the worst

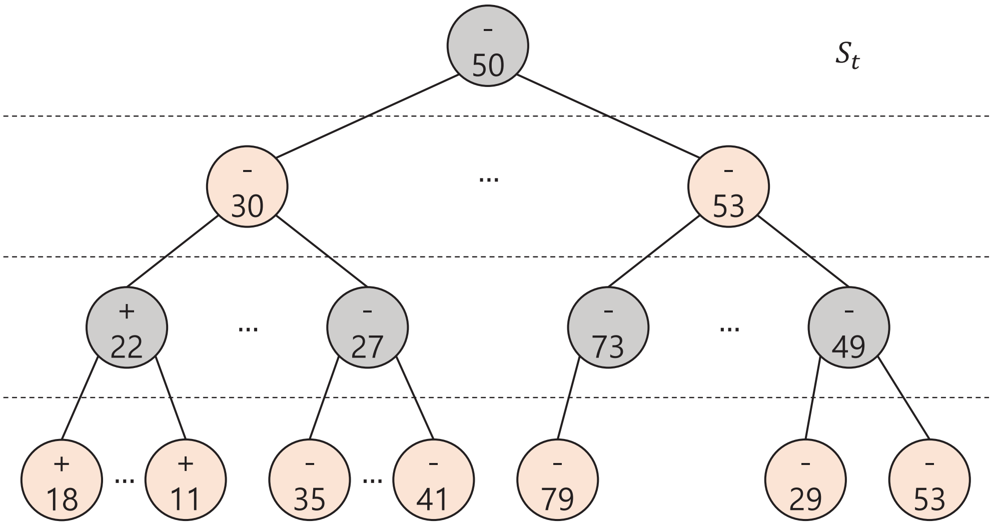
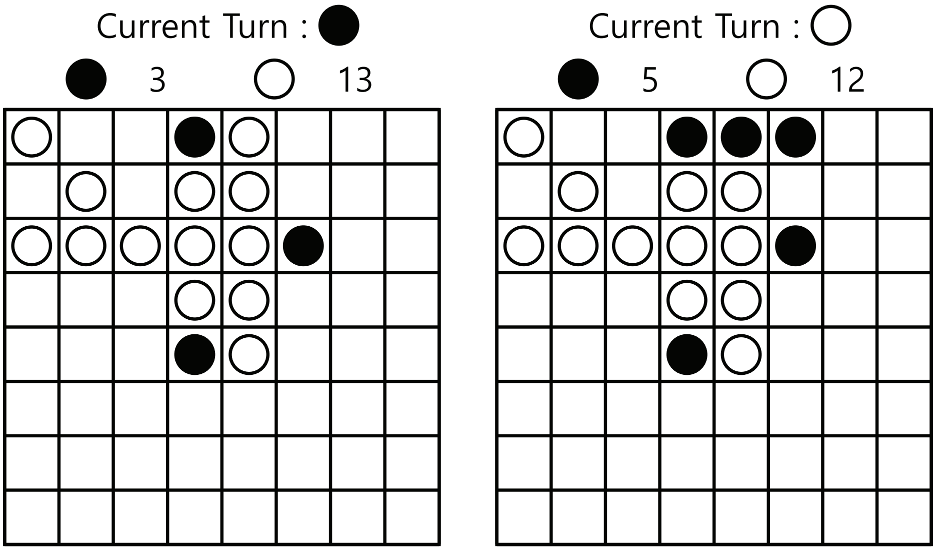


Figure 5.1: A example of state tree, it expand more depth from figure 5.2 (a).

case, the case of 3^{64} is expressed as WPC. By compressing the state using this, it is possible to represent it with a sufficiently small number of states, and to learn the value for each state. The number of states varies depending on how the weight is expressed.

First, I will explain how I learned about TD-Learning and Q-Learning in Othello using a simple example tree. figure 5.1 retrieves the s_t of the state extending from the figure 5.2 (a). Let us consider the process of updating V(s) and Q(s,a) using figure 5.1. For example, suppose that $s_t = +18$ satisfies the terminate condition and Reward is +1. When the state becomes $s_t = +18$, we can calculate the $V(s_t)$ as $V(+18) = V(+18) + \alpha((+1) + \gamma(0) - V(+18)) = (1-\alpha)V(+18) + \alpha(+1)$. Next, when the random visit is repeated and the situation becomes $s_t = +22$, greedy policy will select $s_t = +18$ node. Therefore, the update equation in this case is as follows.
$V(s_t)$ as $V(+22) = V(+22) + \alpha((0) + \gamma(V(+18)) - V(+22)) = (1-\alpha)V(+22) + \alpha(\gamma(V(+18)))$. In this way, $V(-30)$ and $V(-50)$ will be updated as the episode is repeated many times. Other papers have provided basic equations and did not describe the tree as such.

$$S_t = \sum_{i,j=1}^{8} B_{(i,j)} \cdot W_{(i,j)} \tag{5.1}$$



(a) (b)

Figure 5.2: A example state, (a) State (S_t) = -50, (b) State (S_t) = -30

Based on this determined state as in equation 5.1, we calculate the value of the state. Let's take an example of how to express these states. For example, state value(S_t) of figure 5.2 using the weight as figure 3.3 (a) will be as follows:

In case of figure 5.2(a),

$\sum BlackDisc = 5-1-1 = 3$ and

$\sum WhiteDisc = 100+5-50-2-2+10-2-1-1-1-1-1-1 = 53$.

Thus in this case, the S_t becomes $\sum BlackDisc - \sum WhiteDisc = -50$.

Using same way, in case of figure 5.2(b),

$\sum BlackDisc = 5+5+10-1-1 = 18$ and

$\sum WhiteDisc = 100-50-2-2+10-2-1-1-1-1-1-1 = 48$. Thus in this case, the S_t becomes $\sum BlackDisc - \sum WhiteDisc = -30$.

So, in this case, we can estimate the value function as $V(S_t)$ as $V(-50)$ in figure 5.2(a) and $V(-30)$ in figure 5.2(b).

The value of each state is obtained as a value function, where value can be calculated according to the Bellman equation(equation 2.9). Thus, estimating the value according to the Bellman equation has a measure of future value together. However, the present value and the future value are quite different. Also, since not every system has a current immediate reward, we need to predict future values to determine future value. So this, because the value of the future is different from the present, we think of the variable of decay. In other words, there is a reward in the future, but there is no reward at present, so we will continue to decay the future reward and backpropagate to the present. This is part of γ in equation 2.10.

5.1.2 Temporal-Difference Learning

This section looks at temporal-difference learning, a kind of reinforcement learning. Reinforcement learning is originally a theory that originated from the Markov Decision Process(MDP) (Sutton and Barto, 1998).

The Markov Property assumes that the current decision is influenced only by the current state, which is independent of past decisions. This assumption is used in a considerable number of places, and if you ignore this assumption, you have to judge the future by using all the information of the past. So we want to solve the problem based on this assumption. That is, the current choice depends only on the current situation. (Equation 5.2)

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1,\ldots,S_t] \tag{5.2}$$

Based on these assumptions, we try to estimate the value of the future. In the beginning, there are ways to solve the problem with dynamic programming method. However, there is a problem that the amount of information is enormous because the method to solve with this DP is to know about transition state of all states. In order to solve this problem, Monte-Carlo method is introduced in many previous studies. The Monte-

Carlo method used here randomly selects and extends a candidates path (=action), and finally back-propagates the result from the terminate condition. But, this MC method also has the problem of remembering all the paths that propagate backwards. So we want to extend this backpropagation step by step, and the method is the Temporal-Difference method. Now, here if we want to find a path to get good reward, there will be a lot of candidates. If so, what will be good candidates? The guide of this good or bad candidates is policy. This policy is always chosen to maximize the value-function. And the policy looks like equation 5.3

$$\pi(s_t) = \max_{Action} V(s_{t+1}|s_t) \tag{5.3}$$

The contents of the above are summarized as follows (equation 5.4):

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma\, v_\pi(S_{t+1})|S_t = s] \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \quad \text{DP} \\
&= E_\pi\{R_t|s_t = s\} \quad \text{MC} \\
&= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\} \\
&= E_\pi\{r_{t+1} + \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_t = s\} \\
&= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1})|s_t = s\} \quad \text{TD}
\end{aligned} \tag{5.4}$$

Here we look at the value function update of Temporal-Difference method as algorithm 5.

The part of $(\gamma V(s') - V(s))$ is the temporal-difference part, and this method is called the temporal-difference learning because it learns the difference between the values of the next state and the current state.

For example, when looking at the figure 5.2, the state (a) becomes the state (b) when black discs be placed in position (1,5) from (a). In this case, $V(s) = V(-50)$ and

Algorithm 5 V(s) update method using Temporal-Difference

```
Initialize V(s) and π.
repeat
    Initialize s
    repeat
        a ← candidates given by π for s
        Do action a
        Observe reward = r
        Next state = s’
        V(s) ← $V(s)+\alpha[r+\gamma V(s')-V(s)]$
        s ← s’
    until s meet terminate condition
until Saturation
```

$V(s') = V(-30)$ So we can back-propagates and update $V(s)$ as $V(-50|t+1) = V(-50|t)+\alpha(\gamma V(-30|t)-V(-50|t))$. We can update $V(s)$ as above, when we apply it in detail in a game of Othello, problems can arise depending on how we define the state. When you think about it simply, you can think that you can define all of the agent’s turn state and study it according to the difference of the state. However, even if the agent performs the same action, there is a problem in which the agent is confronted with another state depending on the action of the opponents. Therefore, we should update the value function every moment by watching the relative motion as a state transition.

5.1.3 Q-Learning

This time, we want to learn about Q-Learning which is different from TD-Learning. In the existing TD-Learning, we continued evaluation and selection by value-function. In particular, value function(V(s)) was also updated around the node the agent chose. This part is called on-policy, which means that the policy to select and the policy to learn is same(it called online). However, Q-Learning introduced an action-value function.

In other words, it does not mean to choose a good direction based only on the value of the state, but to think about the action together. Therefore, the value of $Q(s_t, a_t)$ is continued instead of the existing $V(s)$. Here, is one more difference between Q-Learning and TD-Learning. Q-Learning basically uses off-policy. This part appears as $\max_{a'} Q(s', a')$ in the Q-Learning algorithm 6. In Q-Learning, if there is a better action-value apart from the selected action, update the Q-value using this value in contrast of TD-Learning use selcted action.

Algorithm 6 Q(s,a) update method using Q-Difference

```
Initialize Q(s,a) and $\pi$.
repeat
    Initialize s
    repeat
        a $\leftarrow$ candidates given by $\pi$ for s
        Do action a
        Observe reward = r
        Next state = s’
        Q(s,a) $\leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s,a)]$
        s $\leftarrow$ s’
    until s meet terminate condition
until Saturation
```

If you think about what you would define as an action, then there is simply a way to define the next state as an action. Starting from figure 5.2 (a), s_t is obtained for some cases and this value becomes figure 5.1. In the previous method, each vertex represents a state, so if you used that value in a vertex, this means that you will have that value in the node connected between the vertices. Here, there is a simple way to express state-action using $a_t = s_{t+1}$.

5.1.4 Update Policy

Both TD Learning and Q Learning are aimed at estimating the value of a state. However, there will be a variety of situations in the game or real world, and there will be several branches to choose from for these various situations. The value of the future depends on which nodes are selected. Therefore, in the existing TD Learning, we used previously selected nodes to guess future value (equation 5.5). The method of determining the future value by using the node selected by the agent as above and updating the value with the current value is called the on-policy. However, there is no need to update only the selected node in order to estimate the value of state. The method separating the policy which makes this selection and the policy which is updating the future value is called off-policy. Therefore, we would like to find a way to increase the winning rate by combining several situations in this off-policy.

$$V(s) = V(s) + \alpha(r_t + \gamma V(s') - V(s)) \tag{5.5}$$

$$V(s) = V(s) + \alpha(r_t + \gamma \frac{1}{N}\sum_{i=1}^{N} V(s'|s,a^i) - V(s)) \tag{5.6}$$

$$V(s) = V(s) + \alpha(r_t + \gamma \max_{a'} V(s') - V(s)) \tag{5.7}$$

The three equations above are the policies we want to compare. We compare the On-Policy method with the Off-Policy method. In Off-Policy, we compare the maximum value and the average value. If learning with Off-Policy method, the agent will have to search more nodes that the agent have not visited, and update it according to the policy, so it may take a relatively long time, but it is expected to be able to show better results. According to the theory of Reinforcement Learning, an agent will always make the choice with the greatest future value. However, due to the nature of this game, future value can vary greatly depending on the choice of enemy (opponents) rather than the choice of agent. If we make this sequence only with the agent's choice, we think the enemy's choice is not reflected in its value and the information is compressed.

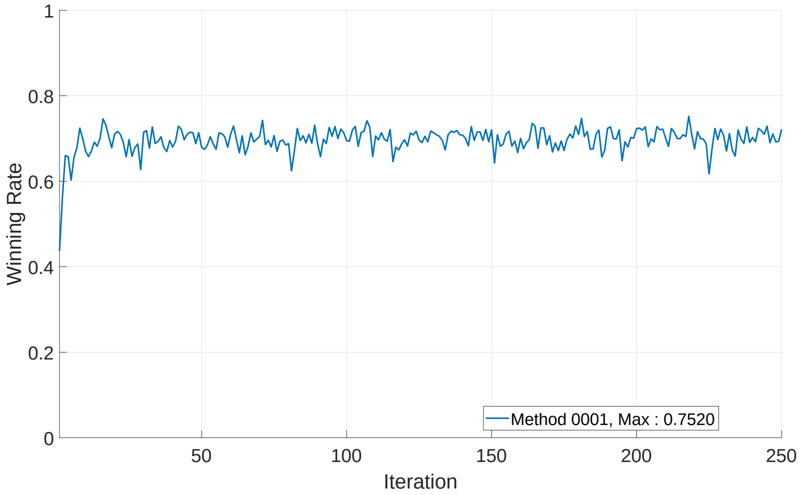


Figure 5.3: The learning result of TD-Learning method from equation 2.10. The result verse with training set and a parameter is learning rate = 0.05, γ = 0.95

Therefore, in order to compare the update, we have also updated V(s) using only agent information and updated V(s) using both agent and opponents information.

5.2 Experimental Result

The result of learning in Othello by TD-Learning method that updates selected node (equation 5.5) every time is figure 5.3. This study shows that learning is possible through TD Learning. However, it showed low winning rate and showed a much worse performance than the linearity of represent.

We also checked the Q-Learning method. At this time, the state and action used are solved by defining the next state as an action as mentioned above. In this case, we can confirm that the final saturation rate is about 5% higher than TD-Learning, but because of the amount of state-action pair is too much, it takes a much longer time to saturation than TD-Learning. However, in this Q learning process, since the value table is updated with one sequence of both the agent and the opponents selection, it is not good to select the maximum value at all times.

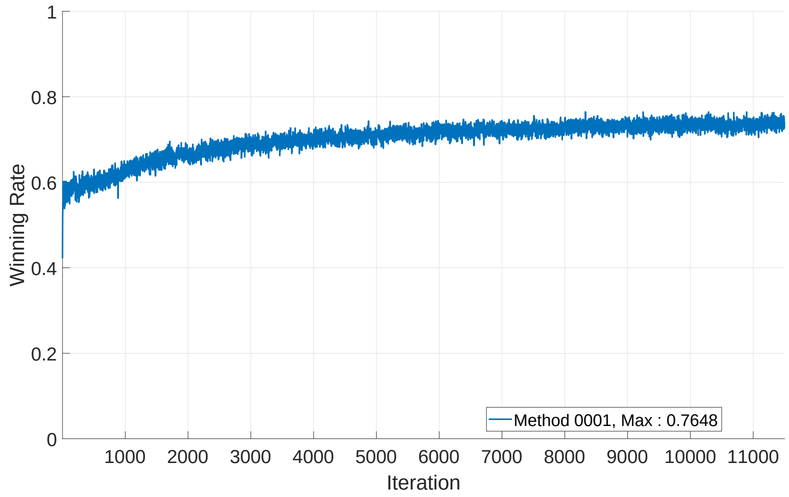


Figure 5.4: The learning result of Q-Learning method. The result verse with training set and a parameter is learning rate = 0.007, $\gamma = 0.95$

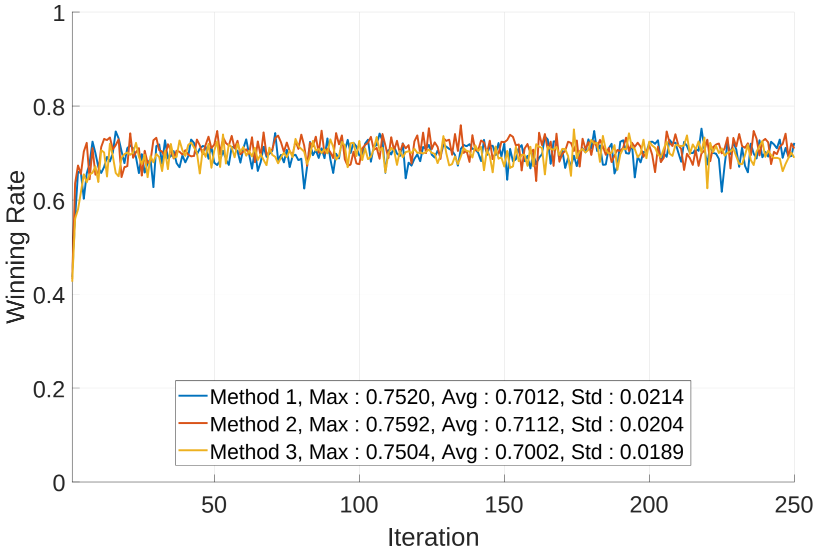


Figure 5.5: The learning result of TD-Learning method using the equation 5.6 and the equation 5.7. Method 1 is original method of TD-Learning, method 2 is using average value, and method 3 is using maximum value. All the result verse with training set and a parameter is learning rate = 0.05, $\gamma = 0.95$

We also compared the use of average value and maximum value in TD learning for policy comparison. The reason for this change in policy is that we have implemented it for confirmation, because there is a part of Sutton and Barto (1998) talking about the limit of on-policy. We have modified the equation of TD-Learning as follows, considering that Q-Learning's off-policy comes from $\max_{a'} Q(s',a')$. In order to distinguish the policy as in Q-Learning, we try again with the method of equation 5.7 which modified $V(s')$ to $\max_{a'} V(s')$. We have tried it, but we have confirmed that there is no difference in winning percentage. In addition, the original Bellman Equation used expectation, so we proceeded to study using the mean of the future. However, we can confirm that there is no difference in the result, and we think that this is the limit of current WPC value. Since the represent what we use, which is pre-defined mapping table, the WPC method has a weighted sum form for each cell, the same state number may be generated even in different situations. For example, if you think about figure 5.1, -53 is appeared two times, and it has different situation. The number of nodes that can be selected is different, and the number of discs placed is different. However, the value of WPC is the same, which seems to be an aliasing problem.

5.2.1 Approximating Unfilled Area

Since we create a value estimation table, there is a problem that it is difficult to express all the various cases. In these various cases, information that is not filled in the learning stage (untrained) leads to a defeat. Therefore, we have categorized the data into several sections in order to reduce the amount of data and approximate it to some extent. In other words, when expressing a state, we used a method of mapping a range from -1000 to -600 as 0, -600 to -550 as 1, instead of -1000 to 1000 which is the range that we have expressed. In this case, both state-based and state-action-based methods showed a fluctuation, but it was confirmed that the tendency was similar to that of the existing one. As a result, this categorized method produces results that are similar to the existing ones, so it has the effect of shortening the learning time. So we used the categorized approach. In addition, since this method is divided into appropriate sections instead of case by case, there is an approximation effect for some sections. Also by applying the Gaussian filter, we were able to achieve a more stable winning

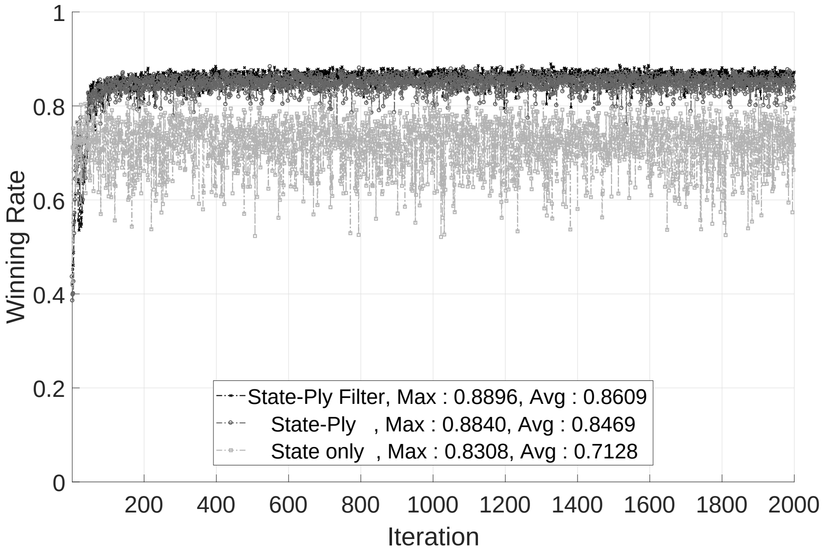


Figure 5.6: Change in the winning rate when adding depth information in TD learning(state based learning). This is the result of adding depth information in method 1 of figure 5.3. And also adding filtering, it approximates other values. $\sigma = 1$ in Gaussian filter. The result verse with training set and a parameter is learning rate = 0.05, γ = 0.99, Categoized 88 section.

rate. This filtering method can obtain a more stable winning rate by approximating using the estimation value of a table that has not been filled yet or the surrounding value when there is a filled but inexperienced value due to lack of experience.

5.2.2 Adding Depth Information

We thought that this aliasing could be eliminated through Q learning but did not meet expectations, so we decided to use additional information to solve aliasing. This attempt tries to use ply (depth) information as additional information. Previously, representations via WPC are one way to represent a board. However, since this method can represent the same value depending on the progress, it can be solved by adding ply information in order to solve this problem. So we want to be able to handle the existing representations at different values as the game progresses. That adds information so that the first case of state x and the second case of state x are different representations.

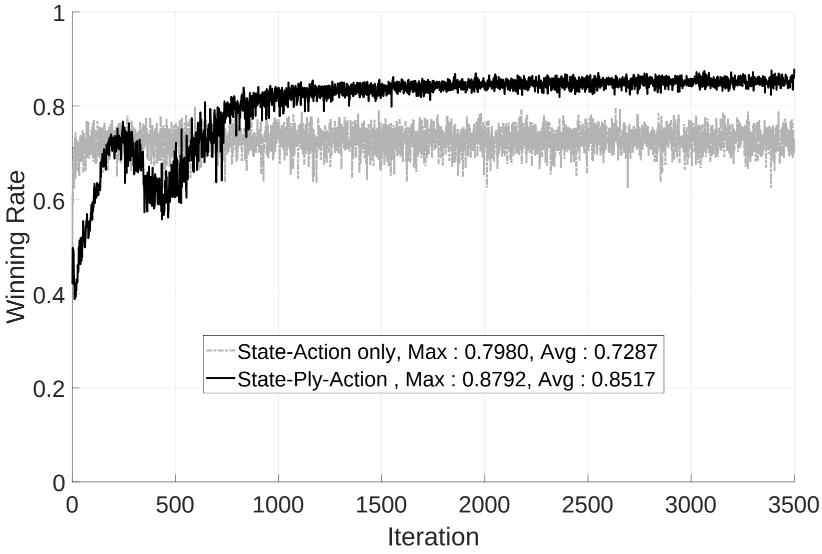


Figure 5.7: In Q Learning, the winning rate when depth information are added. This is the result of adding depth information in method 1 of figure 5.4. The result verse with training set and a parameter is learning rate = 0.05, γ = 0.99, Categoized 88 section.

This is applied to both state-based TD learning and state-action pair based Q-learning.

Figure 5.6 shows how the winning rate changes when applying categorized state method in TD Learning, adding ply (depth) to it, and adding filtering. We were able to see that the winning rate increased steadily in each stage. By adding the ply (depth) information in the TD learning method, the average improvement of about 10% was obtained. At this point, as the ply progresses, it shows a slight increase in the rate at the beginning, and then it rises again. This seems to be because the information according to the ply sparse at the beginning of learning.

In the same way, Q learning (state-action based) has also been compared for policy and for applying filtering. The results are shown in figure 5.8, 5.9, and 5.10. In Q learning, when the ply information and filtering are used together, the winning rate is decreased. This is because the value table is composed non linearly as the game progresses. However, in the Q learning without ply, we could get better results by filtering. This indicates that the value table is somewhat linear in Q learning.

Training Iteration	500	1500	2500	3500
Type I	0.7255(+-0.0261)	0.7246(+-0.0269)	0.7299(+-0.0237)	0.7270(+-0.0239)
Type II	0.7859(+-0.0175)	0.7879(+-0.0181)	0.7895(+-0.0162)	0.7889(+-0.0181)
Type III	0.7856(+-0.0216)	0.7844(+-0.0221)	0.7820(+-0.0219)	0.7852(+-0.0213)
Type IV	0.8515(+-0.0104)	0.8527(+-0.0093)	0.8525(+-0.0088)	0.8528(+-0.0093)
Type V	0.6452(+-0.0451)	0.8367(+-0.0094)	0.8487(+-0.0087)	0.8524(+-0.0083)
Type VI	0.5761(+-0.0177)	0.7224(+-0.0102)	0.7603(+-0.0103)	0.7678(+-0.0103)
Type VII	0.5159(+-0.0151)	0.8111(+-0.0090)	0.8357(+-0.0068)	0.8465(+-0.0072)
Type VIII	0.5092(+-0.0172)	0.6588(+-0.0190)	0.8100(+-0.0358)	0.8455(+-0.0066)

Table 5.1: Performances of the learning algorithms when tested versus randomly movement opponents. Each column shows the performance in the test session where the learning player played best, averaged over a total of five experiments. The standard error (σ / $\sqrt{n}$) is shown as well. Type I shows that result of Q learning using average policy, and II shows average with filter. Type III shows that result of Q learning using maximum policy, and IV shows maximum with filter. Type V shows that result of Type I with ply, and VI shows Type II with ply. Type VII shows that result of Type III with ply, and VIII shows Type IV with ply.

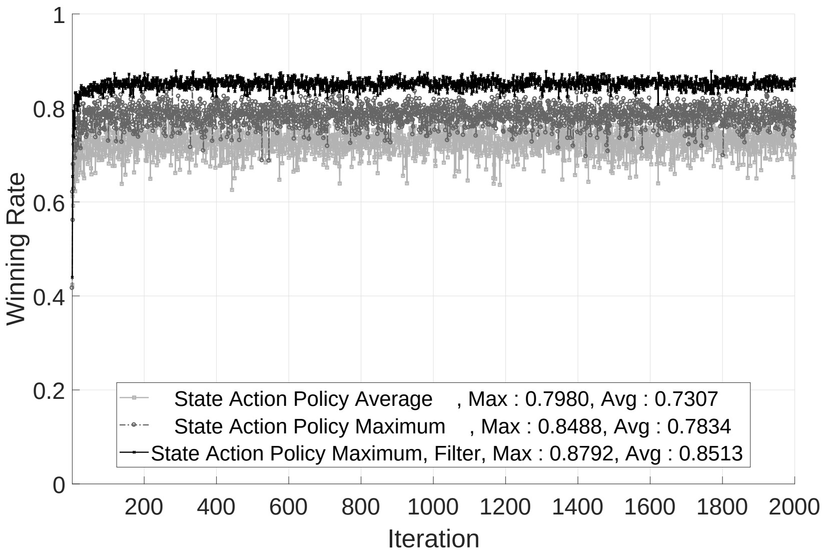


Figure 5.8: In Q-learning, the winning rate according to learning policy and the winning rate when applying a gaussian filter. According to the policy, the average winning rate increases by 5% and when applying filter the average winning rate increases by 7%.

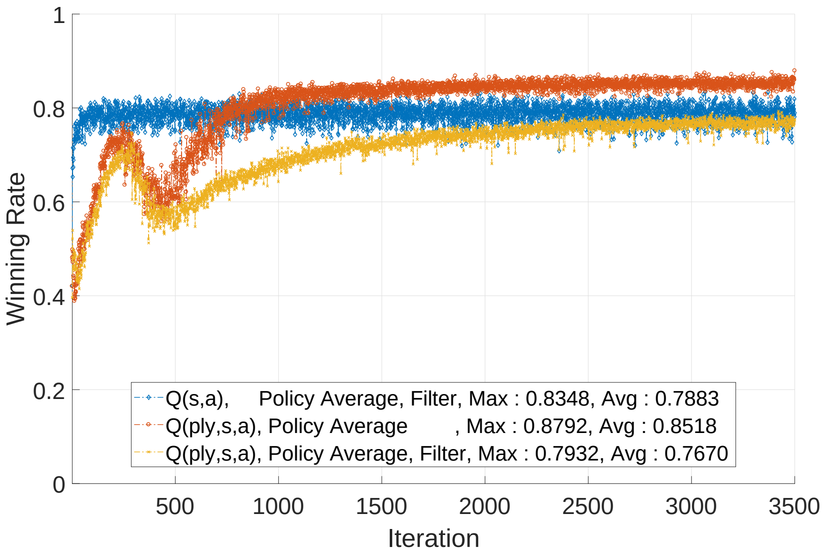


Figure 5.9: In Q-Learning using the update policy as average, compared data previous one with adding ply information. And also compared with applying the Gaussian filter. As a result, we can see that the winning rate is decreased when the ply information is added in Q learning.

Finally, we compared the previous experiments. First, table 5.1 is a comparison table of policy, ply, and filter added in Q learning method of reinforcement learning. When

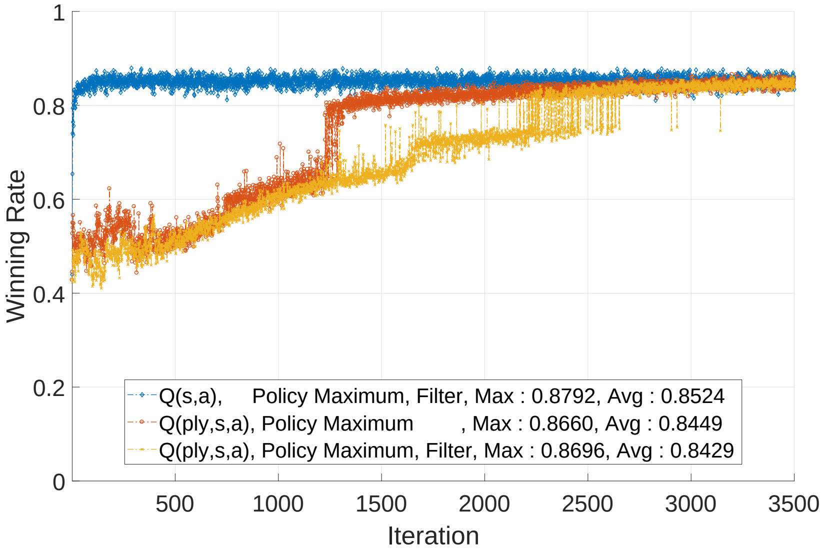


Figure 5.10: In Q-Learning using the update policy as maximum, compared data previous one with adding ply information. And also compare with applying the Gaussian filter. It shows better result comparing with figure 5.9. However, it still can't overcome version of not used a ply information.

the dimension was increased by adding ply information, the poorer results were seen in the early stage due to the lack of information, but the average value converged to about 85% in the latter stage. Also, when Q learning (state - action) was used, it was confirmed that about 6 to 7 % of the winning percentage was increased by the filter. However, when adding ply information, we could see that the filter decreased the winning rate. Also, when using only state(TD), average is better than maximum policy, but it is found that maximum policy is useful when state-action(Q) is used together.

Finally, looking at the table of all the results, it looks like 5.2. The above results are summarized based on the average value. Considering the previous case, the maximum value is higher for the Bayes method, but on the average, the Mask filter gives better results. This may be solved by the change of learning rate in Bayes or Reinforcement Learning. However, because all of the above results are based on random opponents, they may change if you experiment with other opponents in the future. In fact, the

Methods	Bayes Learning Using Posteriori	Masked Filter size = 3 x 3	Masked Filter size = 5 x 5	Masked Filter Combine	TD-Learning	Q-Learning
Case of Learning Average Result	0.859(+-0.017)	0.889(+-0.007)	0.895(+-0.004)	0.895(+-0.006)	0.861(+-0.012)	0.853(+-0.009)
Case of Best Strategy Result	0.908(+-0.014)	0.889(+-0.007)	0.895(+-0.004)	0.895(+-0.006)	0.884(+-0.009)	0.874(+-0.007)

Table 5.2: Table showing the average value and standard variation for all trials until now. From the above results, it can be seen that the mask filter has the best result on the average. Also, the best strategy was extracted and compared too. In this case, Bayes Learning shows the best results.

results of the compilation in Van Der Ree and Wiering (2013) show that the random opponent differs from the opponent using a different algorithm. We also compared the results of repeated experiments with a strategy of maximum value (peak value) during learning rather than average during learning. In this case Bayes learning showed the best result, which is the result of showing the possibility of improving the average by controlling the learning rate during learning.

5.3 Summary of Chapter 5

In this chapter, we briefly studied reinforcement learning. In addition, we checked the problem by using the Markov Decision Process and solving the problem with the Monte-Carlo Method. Finally, we examined Temporal-Difference learning that is based on the state value, and we examined Q-Learning that combines state-action together. In conclusion, Q-Learning showed a slightly better performance, but it required more learning time than TD-Learning. It also shows the potential for further increases depending on how to solve the aliasing problem of state representations. In addition, in the TD-Learning, we tried to use the off-policy learning method a little, but the result showed no significant difference. Rather, using the average expectation according to the bellman equation gave better results. And we thought that Q-Learning could solve the aliasing part when using state representation, but it did not solve well as we expected. However, we tried to solve aliasing by categorizing for approximation and adding depth information, and the result was satisfactory. In particular, when we used depth information together, we could see about 11% higher than the previous one. However, in this case, since the depth information is added to the existing one, the learning takes a long time. These results were applied to both learning methods. However, the difference occurred in the update policy. In the case of Q learning, which shows the state-action together, it is better to take maximize, but in the case of the state-only TD, the best result is obtained when the average is selected rather than the maximum value. This is all due to changes in the agent's winning rate by the choice of opponents. In the case of TD learning, it is more meaningful to express the value of the average because it uses only the state and estimate how much valuable it has when reaching a certain state. However, in the case of Q learning, it is better to select the maximum value of the node rather than the average of each node because both the state and action are used together to determine the value. Here, we can approximate the estimation table by filtering. When this happens, we can see that the winning rate can be increased by 1 or 2 % more. This aliasing problem is caused by the fact that all the representations are fixed, and if it is possible to classify them according to the board state rather than the fixed representations, it seems to be able to produce better results.

Chapter 6

Conclusions

Machine learning is one good way to implement artificial intelligence. The development of the computer makes the agent experience and learn many things which it is difficult for a person to directly try through a lot of data. The learned information is the basis for implementing artificial intelligence. To explore the principles of machine learning, we started from the approach of probability model to reinforcement learning. We have found a way to find the optimal solution from the conditional probability, which shows that learning is possible if there is a lot of experience through the simplest possible method. Recently, it has been studied how to process these data and express it in various representations. The basis for such research is reinforcement learning, and it is called Deep Q Network that solves the problem by using multilayer neural network when expressing state and action in reinforcement learning. In this paper, we have studied Q-learning using the Q-table not DQN, but in the future works, we expect that by expanding the concept it can solve the real-world problem. It also evaluates the value of the action in the Q-table, and if it can be expressed through the local view feature, this may also be a good attempt. In this paper, we have studied how to learn by using a lot of data, how to represent various states through local features, and how to apply learning by reinforcement learning through known methods for expressing global states . In the future, we would like to try to develop this concept further and learn how to mix various features with deep q network or other methods.

6.1 Machine Learning Using Bayes Probability Analysis

We have considered how to express the probability of winning through conditional probability, and also from what perspective we have to see the probability of winning. In other words, we examined how to classify through measurement and class to be successful. At this time, we observed that the probability-based system is sufficient to learn, and confirmed that position evaluation can be done through this method. When learning based on likelihood was started for the first time, the winning rate did not rise to more than about 40% when versed with random weighted opponents. However, when we learned based on a posteriori, we can confirm that it is possible to raise the winning rate to about 90%. This method is expected to be scalable even if we use other measurement and class in the future. Also it is a good possibility if it can be applied to other games in particular. In addition, through a method of MCTS, we have tried to find a way for increasing the winning rate. In the conventional MCTS method, selective depth-first search is performed by simply diversifying the search. At this time, the selection is based on the simulation value. However, these simulation values are compressed into information of victory or defeat, and their value is undermined. If we can maintain this value without damaging it, we can confirm that more valuable decision becomes possible. In the simple minimax tree, although we increased the depth, we could't get better results when using WPC method, but we could confirm that we obtained better probability by constructing game tree in MCTS. It is not performing based on learning,but when comparing between MCTS and minimax in the same weight map, MCTS can show being up to 5% more efficient based on similar time consumption.

6.2 Estimate Local Mask Filter Value

This time, we tried to solve the problem through various features rather than just one feature. There is a phrase “life is a series of choices.”. In the real world, to solve the problem, we continue to make choices. There must be a certain criteria in order to continue the selection every moment. Until now, the criteria has been selected only in the direction of increasing the value through the method called WPC.($max_aWPC(w,B)$) However, this time we looked at several features to diversify the criteria of this choice such as the number of flips, the number of differences between black and white discs, and the value of mask filter. We tried several methods to find a kind of criteria. These various features serve as a basis for selection in solution finding and as a basis for comparing which choices have better value when multiple choices are possible in the same situation. For example, while escaping the maze, there is one way to escape the maze is wall following, and it is a selection criterion. You can solve this problem by creating criteria through various feature extracts and selecting a candidate with a larger value. In this paper, we tried to find out whether selection through these features would be a good choice even if we do not know the whole global situation. Thus, we created a feature with a local view point, and defined the value through extracting from states and tried to learn them by using Monte-Carlo method. The result of this attempt shows that it is possible to solve the problem even if the information is not global. Finally, using this method, we tried naïve bayes classify using multiple features. In this case we just used 2 features. However, the results were not good enough as we expected, and sometimes it was good enough depending on the characteristics of the feature, but in most of the cases, because of the competition between the features, the result was poor. To solve this problem, we thought that there would be a solution by using the value representing the whole situation and the value of the action. This will be our future works.

6.3 Reinforcement Learning

Finally, in this paper, reinforcement learning was examined. Reinforcement learning is one of the unsupervised learning methods that finds out a value of action in an uncertain situation. An important point in reinforcement learning is to read the environment information and express it in the state. This is because the agent learns by valuing these states. Therefore, it is important that the agent knows exactly what the current situation is, and it has the potential to grow further by learning the situation. We defined the represent through the WPC method, and tried to reinforcement learning in the defined state. We examined the temporal-difference method that learns the value of state which is expressed through this represent. This shows that the winning rate is up to about 85% when versed with the random weighted opponents used in the previous chapter. In addition, we studied Q-Learning, which is using state-action pair. It considers the transition from state to another state as an action. In this case, the winning percentage was increased to about 88%. And also this Q-Learning has the possibility to increase performance depending on what action is used and how the state is expressed. In addition, we tried to learn more cases by adding ply (=depth) information in each learning method, or by approximating information that has not been filled yet by filtering. As a result, we were able to win more than the previous simple method. Using this information, we have examined various improvement possibilities by representing the value of the representations in a case by case or approximating a value table. Based on these possibilities, we will be able to do more in the future by using various features.

6.4 Feature Works

6.4.1 More Evaluate Methods

One of many possibilities to try is to introduce various evaluation methods. There have been previous attempts to express this evaluation method with mobility and stability (Rosenbloom, 1982). Thus, in the future, there may be other evaluating methods that are not the evaluation methods presented so far. Experimental data do not use depth information. This means that we ignore information about the time axis. However, the value of information will change over time, and collecting this information will be an additional evaluation method. Also, if you use the time base, you might combine the information from $State_t$ with the information from $State_{t+1}$ or more. For example, the convolution filter used above is currently used in $State_t$, but the difference between the convolution filter value obtained from $State_{t+1}$ and the convolution filter value obtained from $State_t$ depending on the difference, there may be a method of feature extraction. Once you start using the information about time, you can create more features.

6.4.2 Multi Feature Multilayer Model

So far, there have been a few ways to try it out. We used filtering to fill the table that we did not visit while evaluating reinforcement learning by creating state tables. This time we used Gaussian filter, but if we can linearize such a table by dimension reduction method in the future, we think it can be used as one approximation function. If we make the approximation function like this, we think that it will be possible to construct a network model by using together the board information and ply (depth) information found up to this time. If we construct the network model in this way, we think that it will be possible to construct a multi-layer network model and to converge various features. For example, we can configure one layer input for each kind of local mask we used in chapter 4, and another input for the board state to mix the two features. I think that it would be a good idea to construct a model that converge multiple features to obtain a value of future through reinforcement learning. Therefore, I think that fusion of these various features through multi-layer can be a good attempt.

6.4.3 Extend to Another Problem

After that, you can apply the algorithm above to other games. The above algorithm is not for trying to win 100%, but to extract various features and see what results you can get when you view one state from several perspectives. Therefore, we think that it is possible to apply the method of viewing various viewpoints in other games in the same way, and to use the accumulated results to learn. In this game, we looked at the game in various aspects such as difference, flip, convolution, position, depth, etc. and examined the process of integrating and winning the game. Similarly, when we take an action (a_t) in one state (S_t) and apply it to another game (another state) we think it is possible to predict the victory through the process. This processes and results are not as good as simply combining the features with the same weight. Therefore, to process this, the Multilayer Model part mentioned above should be done before.

Bibliography

Abdelbar, A. M. and Tagliarini, G. A. (1998). Using neural network learning in an Othello evaluation function. *Journal of Experimental & Theoretical Artificial Intelligence*, 10(2):217–229.

Alliot, J.-M. and Durand, N. (1995). A genetic algorithm to improve an Othello program. In *European Conference on Artificial Evolution*, pages 305–319. Springer.

Barto, A. (1997). Reinforcement learning. *Neural systems for control*, pages 7–29.

Barua, B. (2013). Provincial healthcare index 2013. *Fraser Institute,'Studies in Health Policy*.

Baudiš, P. and Gailly, J.-l. (2011). Pachi: State of the art open source Go program. In *Advances in computer games*, pages 24–38. Springer.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719.

Binkley, K. J., Seehart, K., and Hagiwara, M. (2007). A study of artificial neural network architectures for Othello evaluation functions. *Information and Media Technologies*, 2(4):1129–1139.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bouzy, B. and Cazenave, T. (2001). Computer Go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–103.

Bouzy, B. and Chaslot, G. (2006). Monte-carlo Go reinforcement learning experiments. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 187–194. IEEE.

Brügmann, B. (1993). Monte Carlo go. Technical report, Technical report, Physics Department, Syracuse University Syracuse, NY.

Buro, M. (1995). Logistello: A strong learning Othello program. In *19th Annual Conference Gesellschaft für Klassifikation eV*, volume 2. Citeseer.

Buro, M. (1997a). Experiments with Multi-ProbCut and a new high-quality evaluation function for Othello. *Games in AI Research*, pages 77–96.

Buro, M. (1997b). Takeshi Murakami vs Logistello. In *ICGA*, volume 20, pages 189–193.

Buro, M. (2002). Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1-2):85–99.

Buro, M. (2003). The evolution of strong Othello programs. In *Entertainment Computing*, pages 81–88. Springer.

Chaslot, G. (2010). Monte-carlo tree search. *Maastricht: Universiteit Maastricht*.

Chaslot, G., Bakkes, S., Szita, I., and Spronck, P. (2008). Monte-Carlo Tree Search: A New Framework for Game AI.

Chen, J. H. and Asch, S. M. (2017). Machine learning and prediction in medicinebeyond the peak of inflated expectations. *N Engl J Med*, 376(26):2507–2509.

Ciancarini, P. and Favini, G. P. (2010). Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*, 174(11):670–684.

Fernández, A. and Salmerón, A. (2008). Bayeschess: A computer chess program based on Bayesian networks. *Pattern Recognition Letters*, 29(8):1154–1159.

Grinstead, C. M. and Snell, J. L. (2012). *Introduction to probability*. American Mathematical Soc.

Hahn, JeongWon and Kim, DaeEun (2018a). Learning with Bayes Probability: A application of Othello.

Hahn, JeongWon and Kim, DaeEun (2018b). Reinforcement learning: A application of Othello.

Hahn, JeongWon and Kim, DaeEun (2018c). Solving a problem with various local mask filter: A application of Othello.

He, S., Wang, Y., Xie, F., Meng, J., Chen, H., Luo, S., Liu, Z., and Zhu, Q. (2008). Game player strategy pattern recognition and how UCT algorithms apply pre-knowledge of player's strategy to improve opponent AI. In *Computational Intelligence for Modelling Control & Automation, 2008 International Conference on*, pages 1177–1181. IEEE.

Heineman, G. T., Pollice, G., and Selkow, S. (2016). *Algorithms in a Nutshell: A Practical Guide*. " O'Reilly Media, Inc.".

Ishii, S. and Hayashi, M. (1999). Strategy acquisition for the game Othello based on reinforcement learning. In *Proc. 5th Int. Conf. Neural Inf. Process.*, pages 841–844. IOS Press.

Jaśkowski, W. (2014). Systematic n-tuple networks for position evaluation: Exceeding 90% in the Othello league. *arXiv preprint arXiv:1406.1509*.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *ECML*, volume 6, pages 282–293. Springer.

Kodratoff, Y. (2014). *Introduction to machine learning*. Morgan Kaufmann.

Korf, R. E. and Chickering, D. M. (1994). Best-first minimax search: Othello results. In *AAAI*, pages 1365–1370.

Krawiec, K. and Szubert, M. G. (2011). Learning n-tuple networks for Othello by coevolutionary gradient search. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 355–362. ACM.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lee, K.-F. and Mahajan, S. (1990). The development of a world class Othello program. *Artificial Intelligence*, 43(1):21–36.

Legg, S. and Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17(4):391–444.

Leouski, A. (1995). *Learning of position evaluation in the game of Othello*. Department of Computer Science.

Liskowski, P. (2012). Co-evolution versus evolution with random sampling for acquiring Othello position evaluation.

Lucas, S. M. (2008a). Investigating learning rates for evolution and temporal difference learning. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 1–7. IEEE.

Lucas, S. M. (2008b). Learning to play Othello with n-tuple systems. *Australian Journal of Intelligent Information Processing*, 4:1–20.

Lucas, S. M. and Runarsson, T. P. (2006). Temporal difference learning versus co-evolution for acquiring othello position evaluation. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 52–59. IEEE.

Maei, H. R., Szepesvári, C., Bhatnagar, S., and Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 719–726.

Marsland, T. A. (1986). A review of game-tree pruning. *ICCA journal*, 9(1):3–19.

McCarthy, J. (2006). Human-level ai is harder than it seemed in (1955). *http://www-formal.stanford.edu/jmc/slides/wrong/wrong-sli/wrong-sli.html).. Retrieved*, pages 12–20.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Moriarty, D. E. and Miikkulainen, R. (1995). Discovering complex Othello strategies through evolutionary neural networks. *Connection Science*, 7(3-1):195–210.

Myerson, R. B. (2013). *Game theory*. Harvard university press.

Narahari, Y. (2014). *Game Theory and Mechanism Design*, volume 4. World Scientific.

Nijssen, J. (2007). Playing Othello Using Monte Carlo. *Strategies*, pages 1–9.

Riedmiller, M. (2005). Neural fitted Q iteration-first experiences with a data efficient neural reinforcement learning method. In *ECML*, volume 3720, pages 317–328. Springer.

Riedmiller, M., Gabel, T., Hafner, R., and Lange, S. (2009). Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73.

Robles, D., Rohlfshagen, P., and Lucas, S. M. (2011). Learning non-random moves for playing Othello: Improving Monte Carlo tree search. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 305–312. IEEE.

Rosenbloom, P. S. (1982). A world-championship-level Othello program. *Artificial Intelligence*, 19(3):279–320.

Ross, S. (2014). *A first course in probability*. Pearson.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Szita, I., Chaslot, G., and Spronck, P. (2009). Monte-Carlo Tree Search in Settlers of Catan. *ACG*, 6048:21–32.

Szubert, M., Jaskowski, W., and Krawiec, K. (2009). Coevolutionary temporal difference learning for Othello. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 104–111. IEEE.

Takeshita, Y., Sakamoto, M., Ito, T., Ito, T., and Ikeda, S. (2017). Reduction of the search space to find perfect play of 6× 6 board Othello.

Tesauro, G. and Galperin, G. R. (1996). On-line policy improvement using Monte-Carlo search. In *NIPS*, volume 96, pages 1068–1074.

Thomassen, A. (1998). Learning and forgetting curves: A practical study. In *33rd Annual Operational Research Society of New Zealand Conference Proceedings, Auckland*, page 89. Citeseer.

Van Der Ree, M. and Wiering, M. (2013). Reinforcement learning in the game of Othello: learning against a fixed opponent and learning from self-play. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*, pages 108–115. IEEE.

van Eck, N. J. and van Wezel, M. (2004). Reinforcement learning and its application to Othello. *Department of Computer Science, Faculty of Economics, Erasmus University, The Netherlands*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Zeng, D., Liu, K., Lai, S., Zhou, G., Zhao, J., et al. (2014). Relation classification via convolutional deep neural network. In *COLING*, pages 2335–2344.

Zhang, H. (2004). The optimality of naive bayes. *AA*, 1(2):3.

국문요약

게임에서의 통계적 확률을 기반으로 학습하는 의사 결정 방법

본 논문은 기계학습의 한 방법을 연구하기 위해 규칙이 존재하는 게임에서 통계적 확률을 이용하는 학습을 연구 해 보았다. 최근 기계학습은 컴퓨터의 발달로 인하여 이론적으로만 연구되던 것들이 실증을 통한 연구로 발전하면서 더 많은 가능성을 보고 다양하게 연구되고 있다. 특히 기계학습은 최근 AlphaGo의 등장으로 기계가 승리할 수 없다고 믿어졌던 바둑에서 컴퓨터의 승리를 보며 많은 관심을 받고 있는 연구 분야이다. 이러한 기계학습은 다양한 방법이 존재하지만 그 중 확률을 기반으로 하는 학습 방법을 연구 해 보고자 한다.

게임에서의 기계학습이 필요한 이유는 게임마다 그 수의 차이는 있지만 대부분의 경우 컴퓨터가 모든 경우의 수를 탐색하기 어려운 양을 가지고 있기 때문이다. 특히 a논문에서 밝히는 바와 같이 바둑의 경우 약 10^360의 탐색범위를 가지고 있고, 이 모든 경우를 탐색하고 다음 선택을 판단하기에는 합리적인 시간 내에 탐색하기가 불가능 하다. 바둑이 아닌 다른 게임의 경우는 체스에서 약 10^123 , 오델로에서 약 10^58 체커 게임에서 약 10^32 정도의 탐색 범위를 갖는다(bouzy2006monte).

따라서, 이런 모든 경우를 탐색하기 어렵기 때문에 우리는 적합한 전략을 이용하여 게임을 하게 되고 이를 하나의 인공지능으로 보고 있다. 이러한 인공지능이 전략을 스스로 학습 할 필요가 있는데 이 때, 우리는 과거의 정보 (통계)를 바탕으로 하는 확률을 이용하여 학습을 진행 해 보고자 한다.

특히 이 논문에서는 3가지 방법을 통해서 이러한 게임을 학습 해 보고자 한다. 첫 번째로는 단순한 확률만을 가지고서 학습이 가능함을 확인하고자 한다. 이때 우리는 Bayes Rule을 이용하여 확률을 분석하고 분석된 정보를 이용하여 학습을 시도 해 보고자 한다. 두 번째로는 우리는 지역적인 정보만을 이용하여 게임의 진행이 가능한지 그 여부를 살펴보고자 한다. 세 번째로는 강화학습을 통해 게임을 학습시켜보고자 한다. 이런 강화학습은 최근 많이 다루어지고 있는 방법이며, 이전에 시도했던 것에서 어떠한 정보를 추가했을 때 더 가능성이 있게 변하는지를 알아보고자 한다.

핵심되는 말 : 기계학습, 인공지능, 오델로, Bayes 확률, 강화학습