國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

master thesis

# 大規模線性增強式學習與競賽式學習以五子棋爲例

# Large Scale Linear Reinforcement Learning and Tournament Learning Illustrated with Gomoku

莊凱閔

Kai-Min Chuang

指導教授：林智仁 博士

Advisor: Chih-Jen Lin, Ph.D.

中華民國 101 年 6 月

June, 2012

# 國立臺灣大學碩士學位論文
# 口試委員會審定書

大規模線性增強式學習與競賽式學習以五子棋為例

Large Scale Linear Reinforcement Learning and Tournament Learning Illustrated with Gomoku

本論文係莊凱閔君（學號 R97922144）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 101 年 6 月 4 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

林智仁

（指導教授）

李育杰　　林軒田

系 主 任　許永真

# 中文摘要

增強式學習曾經是一個熱門的研究主題，但至今增強式學習依然停留在原始階段，即便最知名的例子，TD-Gammon，一個西洋雙陸棋代理人，仍需要藉由搜尋的技巧來提升棋力。在本篇論文中應用了監督式學習中的兩個重要技術，大幅提升增強式學習的能力，分別是大規模資料與線性支持向量回歸。此外，我們也討論了兩個獨立代理人是否能藉由不斷地競爭去增強彼此能力，我們稱這新的學習模式爲競賽式學習。以上兩個概念將會以五子棋演示，結果顯示所產生的代理人不需藉由搜尋的技巧，也具有可與人類匹敵的能力，意味著這兩個概念是實際可行，對於棋類遊戲或是特定應用將會有很大的幫助。

關鍵詞：大規模資料、線性支持向量回歸、增強式學習、競賽式學習、五子棋。

# ABSTRACT

Reinforcement learning has been a promising research topic. However, until now it almost stays at an initial stage. Even the most successful case, TD-Gammon also needs assisted by searching techniques. We design a new method to improve the learning ability of reinforcement learning. The core contributions are using large-scale data and linear support vector regression. In addition, we discuss that could two agents improve their abilities by competing with another. We call this framework as tournament learning. These two concepts would be illustrated with gomoku. The results, our agents, achieve a competitive level, and it implies our concepts are available and practical.

KEYWORDS: Large-scale data, Linear support vector regression, Reinforcement Learning, Tournament Learning, Gomoku.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

CHAPTER I

# Introduction

Reinforcement learning (Kaelbling et al., 1996; Sutton and Barto, 1998) has been a promising research topic, because its purpose is very general. The framework is to train an agent in an environment, and then the agent can take proper actions while facing different situations. TD-Gammon by Tesauro (1992, 1995) could be the most successful case. It is a computer backgammon trained by a reinforcement learning method, and achieved a level closing the human world-champion. People hope such successful cases of reinforcement learning also can be applied in other purposes, like automatic driving, robot or other things human can do. However, until now, reinforcement learning almost stays at an initial stage. Although it is widely used in many fields, it only has puny ability and usually must be assisted by other methods. Even TD-Gammon also used searching techniques. Actually, we do not see any real strong cases yet so we have doubts about its ability.

Unlike reinforcement learning, supervised learning has been well developed. One useful tool that we will use is support vector machines (Boser et al., 1992), which use kernel tricks to map data to high dimensional spaces for separating data. Recently, two properties, large scale data and linear kernel are promoting. Many studies such as Hsieh et al. (2008); Joachims (2006); Keerthi and DeCoste (2005); Shalev-Shwartz et al. (2007) showed in large scale data linear kernel can achieve almost the same accuracy as nonlinear kernels, but much reduces training time and prediction time.

Past reinforcement learning has been helped by supervised learning. The issue of representation is a hard problem of reinforcement learning. Markov decision process (MDP) Bellman (1957) is a typical model to record the information, but it is impossible to build a huge lookup table for complex problems. TD-Gammon using neural networks to simulate the situations in backgammon, but the performance of neural network may indirectly affect the ability of reinforcement learning.

In this thesis, we invent a new representation method for the difference between states. It is very simple, and can record a lot of information in tiny storage. With profuse information we use the modern concepts, large-scale data and effective linear regression to greatly increase the ability of reinforcement learning and reduce the training time. We will illustrate our new method with a board game, gomoku. We also discuss a learning framework, tournament learning. It is a framework used in the situation without a teacher knowing how to improve the ability of competitors. Finally, experiments show our method is stable and fast. More importantly, without any searching technique our gomoku agents can achieve a competitive level.

## 1.1 Gomoku and Backgammon

Two board games, gomoku and backgammon will be discussed in this thesis. To clearly understand this thesis we introduce them briefly here.

### 1.1.1 Gomoku

Gomoku, also called Five in a Row, is a board game. Two players compete on a $15 \times 15$ board, shown in Figure 1.1. One side owns black stones while another owns white stones. The player owning black stones plays first. Then both sides alternately put a stone on an empty intersection. The player that first achieves an unbroken row of five or more stones horizontally, vertically, or diagonally is the winner. For convenience, we call the player who plays first Black, while the other White. Because Black plays first, it has a big advantage. To make a fair game, there are many extra rules, like Renju International Federation (RIF), Sakata rule, Yamaguchi rule and others. To keep easy we use the original rule, and restrict that the first stone must be put at the center of board, $(8, H)$.

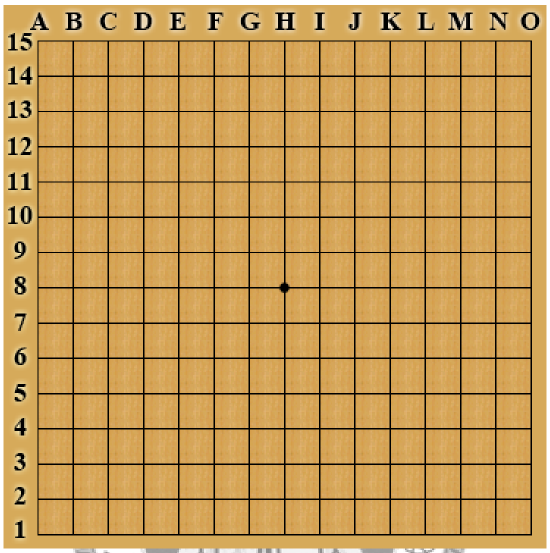


Figure 1.1: A $15 \times 15$ gomoku board.

We choose gomoku because it is easy to learn. A new player can quickly understand the rule, and become familiar in few games. Although the rule is simple, gomoku is very profound. For common human players, defeating current high-level gomoku software is very hard. It means gomoku is not an easy problem. Past reinforcement learning have not successfully solved a difficult problem like gomoku, so we use it to demonstrate the ability of our method.

### 1.1.2 Backgammon

There are many variants backgammons. Here we introduce the simplest version. Figure 1.2 is a backgammon board, and it is also the initial setting at the opening stage.

Each player has 15 stones, and there are 24 triangular cells on the board. Two players roll a die to determine who plays first. White side moves stones counterclockwise and red side moves stones clockwise. At every step a player rolls two dices to decide the number of moves. The special case is player can move stones four times for the same number. For example, if the player rolls a 5 and a 3, he can move his any stones 5 steps to a legal position and repeat the action for 3 steps; if the player rolls two 3, he can move any stones 3 steps four times.

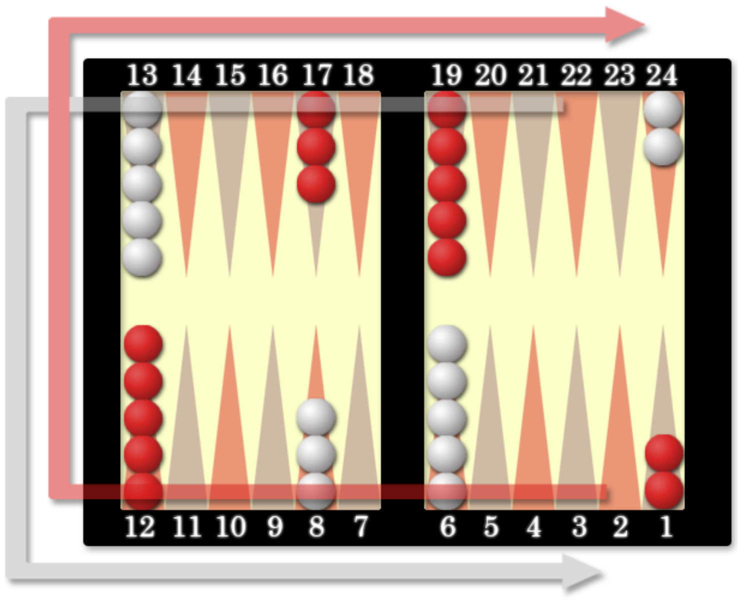


Figure 1.2: A backgammon board.

A legal position is an empty cell or a blot. A blot is a cell occupied by one opponent's stone. If a player moves his stone to a blot, opponent's stone will be removed out of the board, and this stone must restart from beginning. Notice that if there are any stones out of board, we must move them into the board first or we can not move any stones on the board. Therefore, sometimes we can not move any stones if the opponent occupies our aim.

It is needed to move all stones to the last six cells, and then allowed removing the stones out of the board. The player who first removes all stones out of the board is the winner. The winner would get different scores for different cases:

**Backgammon**: The opponent does not remove any stones out of the board, and lefts some stones at the first six cells.

**Gammon**: The opponent does not remove any stones out of the board.

**Normal victory**: Other situations.

The winner can get 3 scores for **Backgammon**; 2 scores for **Gammon** and 1 score for **normal victory**.

Because of rolling dice, luck is an important factor in backgammon. Compared to other board games strategies may not be that effective. Even an expert could be defeated by a rookie.

CHAPTER II

# Large-scale Data and Linear Reinforcement Learning

## 2.1 Background

### 2.1.1 Reinforcement Learning

Reinforcement learning is a trial and error procedure. It uses an agent to explore an environment. When an agent takes an action in a state, it would receive the reward. According to the reward it reconsiders the action. Next time the agent can make a better decision by this experience. Markov decision process (MDP) is used to record the information (Bellman, 1957; Kaelbling et al., 1996; Puterman, 1994; Sutton and Barto, 1998; White, 1991). An MDP consists of four tuples $(S, A, P.(\cdot,\cdot), R.(\cdot,\cdot))$, where

1. $S$ is a finite set of states.
2. $A$ is a finite set of actions.
3. $P_a(s, s')$ is the probability that action $a$ in state $s$ at time $t$ will lead to state $s'$ at time $t+1$.
4. $R_a(s, s')$ is the immediate reward.

A policy $\pi(s,a) = Pr(a_t = a|s_t = s)$ is a mapping function from states to actions. There are two types of the value function. The first one is the state-value function: $V^\pi(s) = \mathbb{E}_\pi[R_t|s_t = s]$, which means that from state $s_t$ following the policy can get expected reward $R_t$. $R_t = \sum_{k=t}^{T} r_k$ is the total reward from time $t$ until reaching the terminal state at time $T$. In order to avoid that future feedback affects the immediate reward aggressively, there is a discount factor $\gamma$, $0 < \gamma \leq 1$, so the expansion of the state-value function is

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \tag{2.1}$$

where $r_t$ is the reward at time $t$. The second value function is the action-value function:

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\pi}[R_t|s_t = s, a_t = a] \tag{2.2}$$

It differs from the first one by emphasizing the action to take.

The goal of reinforcement learning is finding the best policy $\pi^*$ to maximize the value of every state. That is,

$$V^*(s) = \max_\pi V^\pi(s), \forall s \in S \tag{2.3}$$

or

$$Q^*(s,a) = \max_\pi Q^\pi(s,a), \forall s \in S, a \in A \tag{2.4}$$

Two famous reinforcement learning methods, temporal difference (TD) learning by Barto et al. (1989) and Q-learning by Watkins (1989), solve these two types of optimization problems respectively. The basic TD learning, TD(0), uses (2.5) to update the value of the current state by the value of the next, where $\alpha$ is learning rate.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2.5}$$

The update rule in (2.2) is very similar.

$$Q(s_t.a_t) \leftarrow Q(s_t, s_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.6}$$

Although there are many research works about reinforcement learning, successful cases are rare. Most applications are simple toys like maze (Dayan and Hinton, 1993), soccer (Littman, 1994), elevator dispatching (Sutton and Barto, 1998), cart-pole swing-up (Doya, 2000) and crawler robot (Tokic et al., 2009). TD-Gammon may be the most successful case in reinforcement learning, even though it used searching techniques for prediction. Therefor, the research of reinforcement learning is still at an early stage.

### 2.1.2 Supervised Learning

Assume we have a set of training data $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_l, y_l)\}$, where every instance consists of features $\boldsymbol{x}_i$ and label $y_i$. Supervised learning finds a function $g : \boldsymbol{X} \to \boldsymbol{Y}$, where $\boldsymbol{X}$ is the feature space and $\boldsymbol{Y}$ is the label space. If the label is discrete, it is called classification; if the label is continues, it is called regression. One popular supervised learning tool is support vector machine (SVM) (Boser et al., 1992). For classification the following optimization problem is solved.

$$\min_{w} f(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{j=1}^{l}\xi(\boldsymbol{w}; \boldsymbol{x}_j, y_j), \tag{2.7}$$

where $\boldsymbol{x}_j \in \mathbf{R}^n$, $y_j \in \{-1, +1\}$, $\xi(\boldsymbol{w}; \boldsymbol{x}_j, y_j)$ is a loss function, and $C \in \mathbf{R}$ is a penalty parameter.

Feature space can be mapped into a high-dimensional space by a nonlinear function. Using the technique usually can increase the accuracy. Recently researches on linear classification without using mapping functions have been promoted. For large-scale sparse data the performance of linear kernel is close to nonlinear kernel, but training time and predicting time both are much reduced.

The great ability of supervised learning is collecting fractional information to predict. It has been used in classification, regression, and ranking successfully. If training data is proper and enough, currently supervised learning tools can get excellent results. However, existing research is often restricted these three applications. We think that supervised learning can be extended to other applications, where reinforcement learning may be a possible choice. Using modern supervised learning techniques, the ability of reinforcement learning may be improved.

### 2.1.3 The High-Level Idea

The ability of past reinforcement learning is restricted by the representational issue (Tesauro, 1992). The simplest way is recording every MDP discretely, but there is no sufficient memory to store the huge lookup table. Further, it is impossible to go through every MDP, so the agent can not take a correct action while encountering unknown states.

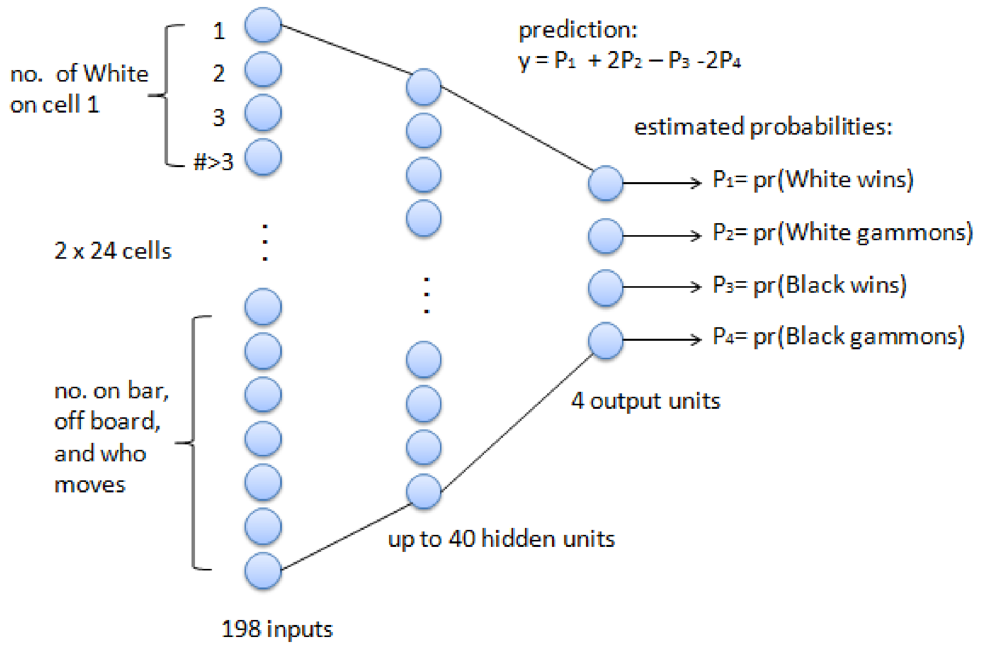


Figure 2.1: The TD-Gammon network. Hidden and output units are sigmods, and learning rate: $\alpha = 0.1$; initial weights are set randomly between $-0.5$ and $0.5$.

In earlier works, neural networks have replaced with the lookup table. Using neural networks can simulate MDPs by fewer inputs. We briefly introduce how TD-Gammon used Backpropagation Networks (Nilsson, 1996) to implement reinforcement learning. Figure 2.1 shows the rough network structure. The updating rule of weights $\boldsymbol{w}$ which combines TD($\lambda$) with backpropagation is

$$\Delta \boldsymbol{w}_t = \alpha(y_{t+1} - y_t) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial y_k}{\partial \boldsymbol{w}} \tag{2.8}$$

where $\lambda$ is the discount factor, $\alpha$ is the learning rate, and $y_t$ is the prediction at time $t$. With $\lambda = 0.7, \alpha = 0.1$ after about $200,000$ games, TD-Gammon can win 66.2% of $10,000$ games against SUN Microsystems Gammontool (Nilsson, 1996).

Although neural networks can reduce the storage, however, the slow convergence is another issue. Equation (2.8) shows that the update of weights is just depended on few data. It is not reliable because the agent could change its policy aggressively after just few games. Hence, even TD-Gammon still needs to use searching techniques. It also implies neural network may not be the best way to implement reinforcement learning.

We propose a new and simpler reinforcement learning system which tries to solve these issues. We use Support Vector Regression (SVR) by Vapnik (1995) to implement reinforcement learning. We treat each MDP as a feature and each reward value as a target value. The framework is shown in Figure 2.2. It includes seven independent parts:

1. **Extract the features**: Extracting the features is like observing the environment for the agent; more detailed information is more beneficial, but we also need to consider the issue of storage.

2. **Predict**: The agent has its weights. In linear regression, the weights are coefficients of the corresponding features. Therefor, the prediction is just the inner product of features and weights. We call last part and this part as the playing stage. The agent extracts features, and then uses its weights to determine what

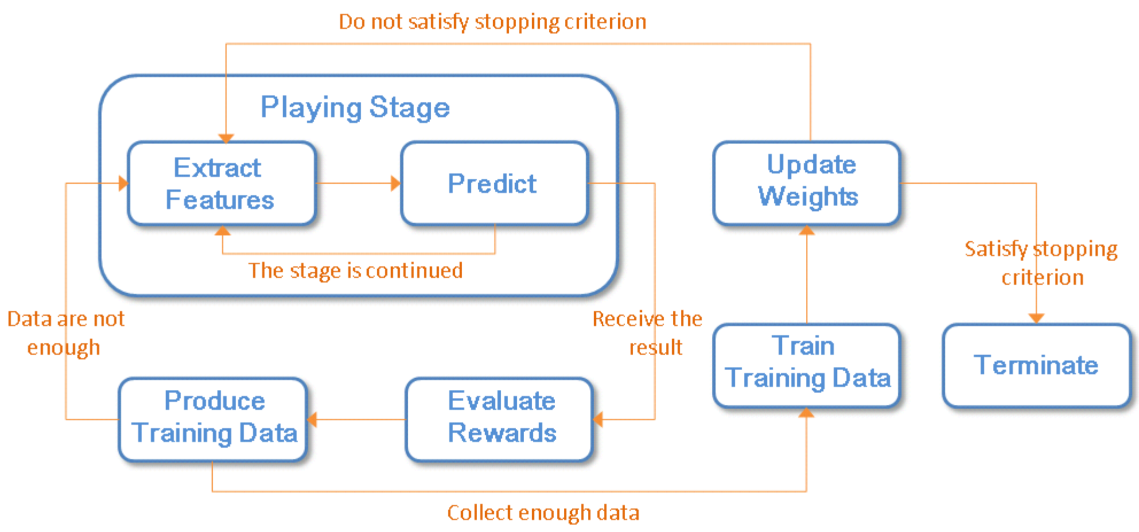


Figure 2.2: The framework of the reinforcement learning.

action it should take in the environment.

3. **Evaluate the rewards**: After the playing stage the agent would receive a result from the environment. According to the result it revises the original predictions to be new rewards. For example, in the board games, the agent would receive a victory or a losing. If it receives a victory, the original predictions would be increased. We use a temporal difference approach to revise the predictions.

4. **Produce the training data**: An instance includes features and target values. We have already extracted the features in the playing stage, and get rewards at the previous step. However, a contest can provide only a limited amount of instances, so the playing stage would be repeated many times for more instances.

5. **Train the training data**: We choose LIBLINEAR (Fan et al., 2008) to train the training data. LIBLINEAR is designed for large-scale classification and regression. Two properties, large-scale data and linear kernel, are our core contributions, and we will explain them later.

6. **Update the weights**: After training we get the new weights. Because of using a linear regressor we can update weights in linear time.

7. **Terminate**: The new agent has the new action model. If the reinforcement learning is proper, the ability of the agent should be better. We set the stopping criterion to control when the whole procedure should be terminated. For example, in the board games, we can use the winning ratio as a stopping criterion. However, the agent may never reach the targeted winning ratio, so a maximal number of attempts is also used as a stopping criterion.

We explain more about core contributions of using large-scale data and linear support vector regression. Both TD-gammon and our method need large-scale data to train the agent, but the key point is how to update the weights. We suggest to train large-scale data and then update original weights by new weights. This setting is very different from updating weights many times based on one instance. We think that considering many different situations can get more reliable results, and more data is more beneficial. However, collecting large-scale data is time consuming. While collecting data the agent uses its weights to predict what action should take. Thus, fast prediction is also important. This is achieved by using linear kernel. With the help of these two properties, our reinforcement learning system can deal with difficult problems in acceptable time.

Because of these two properties, we call our method large-scale-data linear reinforcement learning. We would illustrate it with gomoku and discuss more detailed issues in the following sections.

## 2.2 Extracting Features

We mentioned that the representation issue is important. It is needed to design a new representation which can represent various states in acceptable storage. TD-Gammon just used stones' positions as input, and claimed that it is a "knowledge-free" approach. The reason is that besides positions it did not add any knowledge. However, for increasing the ability, it still added some special features as inputs in latter versions. Undoubtedly, more meaningful features are very helpful for learning. Without meaningful features the situation is like we want an agent to walk but do not give it any sensor.

### 2.2.1 Conditions and States

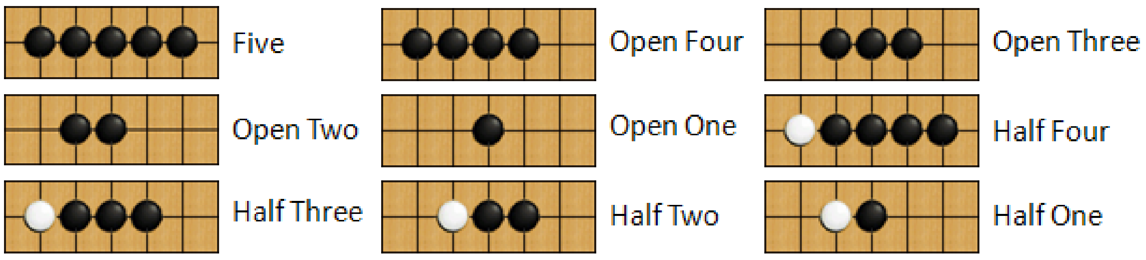


Figure 2.3: Some terminologies about gomoku.

Figure 2.3 shows some terminologies about gomoku. These terminologies are adopted through common practice. Five means that five stones are in a row and victory. Open Four means that four stones are in a row and both sides are empty; Half Four is like Open Four but there is an opponent's stone on one side. We also call these terminologies conditions, and define a vector, state $\boldsymbol{S}$, to record the conditions of a board.

Intuitively, we can record the number of the conditions in the state, but it is not proper. For example, an Open Three just represents closing to win, but two Open Three represents almost winning, so the weight of the latter should be larger than twice of the former. Some sample states are shown in Table 2.1. Every element indicates whether one condition is happening or not.

For example, if there is an Open Four on the board, the value of the 5th element is 1; if there are a Half Four and two Open Three on the board, the value of the 8th and the 12th element are 1. If a condition is not happening, the value is 0. Because there are two players, we also need to record the opponent's states.

| **Index** | **Element meaning** | **Index** | **Element meaning** |
|---|---|---|---|
| 1 | #Five is 0 | 10 | #Open Three is 0 |
| 2 | #Five is 1 | 11 | #Open Three is 1 |
| 3 | #Five is 2 | 12 | #Open Three is 2 |
| 4 | #Open Four is 0 | 13 | #Half Three is 0 |
| 5 | #Open Four is 1 | 14 | #Half Three is 1 |
| 6 | #Open Four is 2 | 15 | #Half Three is 2 |
| 7 | #Half Four is 0 | 16 | #Open Two is 0 |
| 8 | #Half Four is 1 | 17 | #Open Two is 1 |
| 9 | #Half Four is 2 | 18 | #Open Two is 2 |

Table 2.1: Sample states.

### 2.2.2 A New Representation: Difference between States

We design a new representation method which is suitable for supervised learning. We try to represent the action in the state as features. In last subsection, we define a state $\boldsymbol{S}$ to represent a board, so in a contest, there are $t$ states $\boldsymbol{S}^1, \boldsymbol{S}^2, \ldots, \boldsymbol{S}^t$ where $t$ is the step index, and all elements' values are 0 or 1. We use the difference between three states as features. In step $j$, we make features by differences between $\boldsymbol{S}^{j-2}, \boldsymbol{S}^{j-1}$,and $\boldsymbol{S}^j$. Notice that $\boldsymbol{S}^{j-1}$ is controlled by the opponent. Because all elements' values are 0 or 1, the combinations of differences could be $\{0,0,0\}, \{0,0,1\}, \ldots, \{0,1,1\}, \{1,1,1\}$. Thus, if there are $N$ conditions in a state, the number of features is $8N$.

By some examples, we explain the advantages of our representation method. There are three advantages:

1. It does not need to define actions, but can capture more slight changes.

2. It has deep consideration by using the information of previous states.

3. It reduces the storage but keeps profuse information.

First, in our example there is no need to define actions. In gomoku, the transition from one state to the next state could change many conditions. Figure 2.4 is an example, where from state $\boldsymbol{S}^6$ to state $\boldsymbol{S}^7$ many conditions are changed: Black achieves an Open Three and a Half Three, and also defends White's Open Three. Therefore, many changes could happen at the same time, and it is hard to define an action including so many changes. In our method, the difference between states exactly means action. Table 2.2 shows that some features can be captured by our method. We can see that every change has its meaning. We just focus on what conditions could happen, and make more different conditions. Based on profuse conditions, we can capture slight changes.

Table 2.3 shows the rough meaning of differences between three states. For example, $\{0, 1, 0\}$ means that a condition did not happen on my last turn, and the opponent made it happen, but we cancel it again now. It implies the opponent want this condition happen but we do not permit. We consider that preventing the opponent's will is defense.

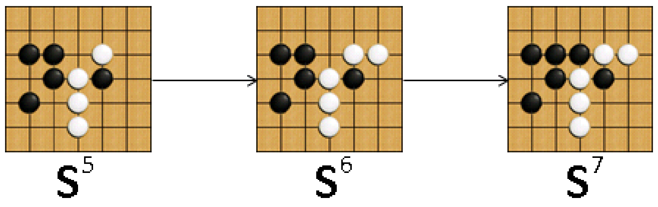


Figure 2.4: An example of board transition.

Second, deep consideration. A disadvantage of an MDP model is that it considers

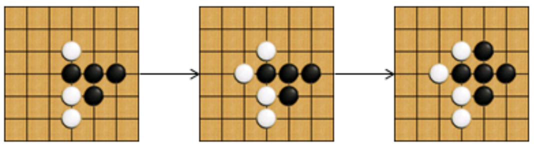
| Condition | $S^5$ | $S^6$ | $S^7$ | Meaning for Black |
|---|---|---|---|---|
| Black's #Open Three is 1 | 0 | 0 | 1 | Achieves Open Three |
| Black's #Half Three is 1 | 0 | 0 | 1 | Achieves Half Three |
| White's #Open Three is 1 | 1 | 1 | 0 | Defends opponent's Open Three |
| White's #Half Three is 1 | 0 | 0 | 1 | Lets opponent achieve Half Three |
| White's #Open Two is 1 | 1 | 0 | 1 | Lets opponent achieve Open Two |
| White's #Open Two is 2 | 0 | 1 | 0 | Defends opponent's Open Two |

Table 2.2: Some examples of three states of Figure 2.4.

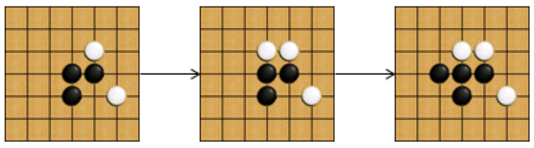
| $S_i^{j-2}$ | $S_i^{j-1}$ | $S_i^j$ | Meaning |
|---|---|---|---|
| 0 | 0 | 0 | Nothing change |
| 0 | 0 | 1 | Attack |
| 0 | 1 | 0 | Defense |
| 0 | 1 | 1 | Agree opponent's action |
| 1 | 0 | 0 | Agree opponent's action |
| 1 | 0 | 1 | Defense |
| 1 | 1 | 0 | Attack |
| 1 | 1 | 1 | Nothing change |

Table 2.3: Rough meaning of the change of three states.

only the current state, but previous states may reveal important information. Figure 2.5 shows that the meaning of $\{1, 0, 1\}$ and $\{0, 0, 1\}$ are different. The conditions of Black's Open Three in case 1 are $\{1, 0, 1\}$, and in case 2 are $\{0, 0, 1\}$. Because White must defend Open Three, Black still can attack by Half Four in both cases. However, if we consider more deeply, Black has two choices of Half Four in case 1 but only one in case 2. Thus, for Black, case 1 is more beneficial than case 2. A good strategy usually needs many steps to achieve. Without the information of previous states it can hardly be realized.



(a) Case 1: The difference is $\{1, 0, 1\}$.



(b) Case 2: The difference is $\{0, 0, 1\}$.

Figure 2.5: Two cases of the differences of Black's Open Three.

Third, our approach reduces the storage but keeps profuse information. Our method only needs three boolean arrays to record three states. The size of training data for LIBLINEAR is determined by the number of instances. We restrict the number of instances to be less than $15,000$, and average number of non-zero features of an instance is less than 50, so the training data is less than 10MB. The training result, a set of weights, is also small and less than 1MB.

### 2.2.3 Trivial and Fractional Information or Strategy

Two famous strategies for gomoku are Victory of Continuous Four (VC4) and Victory of Continuous Threats (VCT). VC4 is the continuing attack by Four until victory. Because the opponent must defend Four, we can know the position where the opponent will put stone. If we still can achieve Four again in our next turn, we can know the next two actions of the opponent. It means that we can control opponent's action and develop a beneficial situation for us at the same time. VCT includes Double Three and the step before VC4. The effect is very similar to VC4, and it also implies that we can partially control the opponent. Both VC4 and VCT are excellent strategies for searching. Currently high-level gomoku software also use them and can make a decision quickly. However, the strategies are contributed by human but not computers.

In most cases we may not have a good strategy, but can get some trivial and fractional information. This is a situation that we hope reinforcement learning can help us. We want to show that even depending on trivial and fractional information, reinforcement learning may already give good performance.

Of course we can add the features about VC4 and VCT just by spending more learning time. More informative features are more helpful for learning. From our viewpoint we do not add features such as VC4, VCT, or others. We hope our method is a purely reinforcement learning method without any searching technique, and we want to show that combining many pieces of trivial information also can achieve a competitive level.

## 2.3 Reward Value Evaluation

After a contest we collect a set of instances without reward value $y$. Notice that half instances are opponent's reactions so we discard them. Algorithm 1 is a temporal difference reward algorithm. Here we write down it in a supervised learning form, where $\boldsymbol{w}$ is the current weights of the agent and $\boldsymbol{x}$ is the whole instance, so $\boldsymbol{w}^T\boldsymbol{x}_i$ is the prediction of instance $\boldsymbol{x}_i$. If the result of the contest is a victory, we set a positive value for the final reward; if the result is losing, we set an inverse value. $k$ step means examining the action by the rewards of next $1, \ldots, k$ step. The discount rate $\gamma$ controls the future reward effect, and lets the effect of further reward be decreased exponentially.

**Algorithm 1** $k$-step temporal difference reward algorithm

```
• Given a set of one contest instances, assume the number of instances is $L$.
• Set the value of the final reward $y_L$, the step of consideration $k$ and the discount
  rate $\gamma$.
• For $i = L, \ldots, 1$
      $y_i \leftarrow \boldsymbol{w}^T\boldsymbol{x}_i$
      $y_i \leftarrow (1-\gamma)y_i + \gamma y_{i+1} + \ldots + \gamma^k y_{i+k}$
```

In our experiments we set $k$ to be four. Because the effect is exponential decreased by steps, after four or five steps the effects are closed to zero. Figure 2.6 are two updating examples of different $\gamma$. Black-font values are original prediction values. Red-font values are updated with $\gamma = 0.3$, and blue-font are updated with $\gamma = 0.1$. We can see that red-font values are changed aggressively, but with large-scale data, an

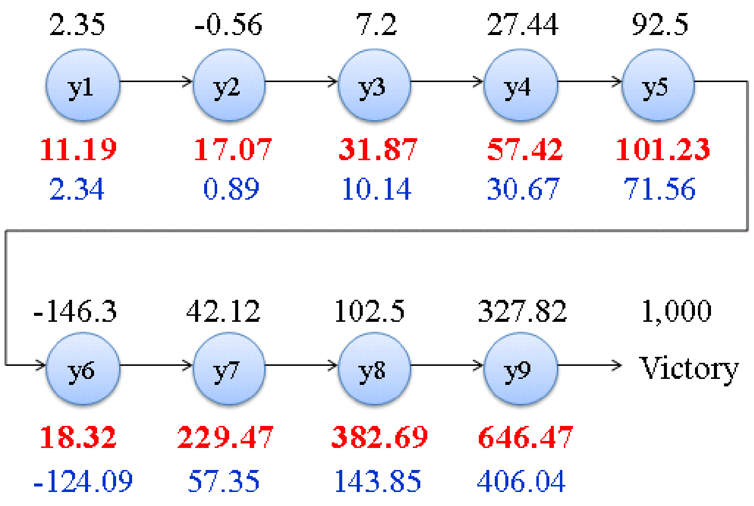


Figure 2.6: An example of updating the reward value. The final reward of a victory is $1,000$. Red-font values are updated with $\gamma = 0.3$, and blue-font values are updated with $\gamma = 0.1$.

aggressive update can quickly determine which features are beneficial and which are harmful.

## 2.4 Training and Update weights

### 2.4.1 Settings of LIBLINEAR

Support vector regression (Vapnik, 1995) (SVR) is extend from SVM. The detailed formula of linear SVR is

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w}) \tag{2.9}$$

where

$$f(\boldsymbol{w}) \equiv \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_\epsilon(\boldsymbol{w};\boldsymbol{x}_i,y_i),$$

$C > 0$ is the regularization parameter, and

$$\xi_\epsilon(\boldsymbol{w};\boldsymbol{x}_i,y_i) = \begin{cases} \max(|\boldsymbol{w}^T\boldsymbol{x}_i - y_i| - \epsilon, 0) & \text{L1-loss} \quad (2.10) \\ \max(|\boldsymbol{w}^T\boldsymbol{x}_i - y_i| - \epsilon, 0)^2 & \text{L2-loss} \quad (2.11) \end{cases}$$

is the $\epsilon$-insensitive loss function related with $(\boldsymbol{x}_i, y_i)$. Thus, if $|\boldsymbol{w}^T\boldsymbol{x}_i - y_i| \le \epsilon$ the loss is zero.

The latest LIBLINEAR provides some solvers for linear SVR. We have tried many parameters and solvers, and find out two special viewpoints in our case.

1. Regularization term is not beneficial for regression.

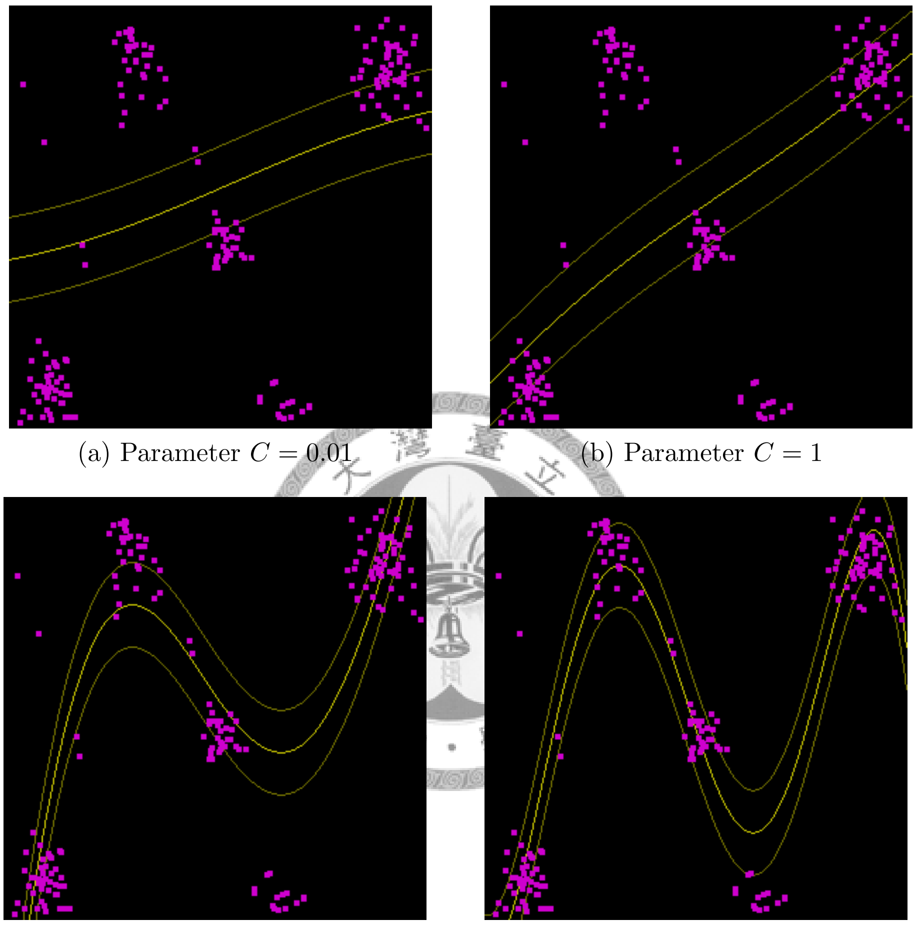
2. Absolute loss is not proper and unfair.

These two viewpoints are very different from our former experience. However, they do not relate to reinforcement learning, so we did not research them deeply.

In SVM, regularization term enlarges the margin between instances with different labels. Undoubtedly the concept is very successful, but it does not imply it is suitable for SVR. We have tried many different values for $C$, and find out large $C$ always is better. Figure 2.7 are four examples of regularization term affects SVR. Large $C$ means the effect of regularization term is less, and with large $C$ the curve is more fit to data. If the curve is too smooth, like Figure 2.7a, it almost loses the purpose of regression because of erroneous predictions. To avoid underfitting, we prefer selecting a large $C$.

It also can be explained by the weight of features. We hope SVR helps us to determine which features are beneficial and which are harmful. If the differences of weights are large it is more clear to distinguish. Regularization term is like gravity. It attracts the value of every weight to zero, so every weight becomes close to each other and contravene the purpose of SVR.

In our experiments, we set parameter $C$ as a large number to decrease the effect of the regularization term, but a large $C$ may derive overfitting. For some features which rarely appear in the training data, their corresponding weights could become very large to fit target values. For example, assume there is an instance whose non-zero features indexes are $\{2, 5, 7\}$ and the target value is $1,000$. If feature indexes 5 and 7 appear in many instances which have target values close to zero, the weights of these two features are possibly small. If feature index 2 just appears once in the whole data, its value



(a) Parameter $C = 0.01$ (b) Parameter $C = 1$

(c) Parameter $C = 100$ (d) Parameter $C = 10,000$

Figure 2.7: An example of regularization term affects the SVR. The pictures are captured from LIBSVM (Chang and Lin, 2011) graphic interface, and the parameters are -$s$ 3 -$t$ 2.

will be close to $1,000$ because of large $C$. Therefor, the weights of rare features are unreliable. Thus, we delete the features which rarely appear in the training data.

| Instance | Target value | Prediction $\boldsymbol{w}_1^T\boldsymbol{x}_i$ | Prediction $\boldsymbol{w}_2^T\boldsymbol{x}_i$ |
|---:|---:|---:|---:|
| $\boldsymbol{x}_1$ | 1 | 1.2 | 200 |
| $\boldsymbol{x}_2$ | 20 | 25 | 500 |
| $\boldsymbol{x}_3$ | 500 | 600 | 1,200 |
| $\boldsymbol{x}_4$ | 10,000 | 12,000 | 10,000 |
| $\boldsymbol{x}_5$ | 50,000 | 65,000 | 50,000 |
| L1-loss | | $17,150.2$ | **1,379** |
| L2-loss | | $229,012,500.04$ | **760,001** |
| Relative-loss | | 6.15 | 229.4 |

Table 2.4: An example of different losses.

The lose term plays an important role in our case. We have tried L1-loss and L2-loss SVR, but neither got good results. A strange phenomenon is that agents can take a right action at the final stage, but at the primary stage they seem randomly play. We checked the value of predictions and found out that, the predictions at the primary stage are always many times greater than its target value.

Because the target values increase exponentially, the differences of them are very large. The effect of absolute loss is very different for different scales of target values. For example, a target value $y_1 = 1,000$ can endure a loss $\xi = 10$, but for a target value $y_2 = 0.0001$ a loss $\xi = 10$ is a disaster.

Table 2.4 shows four examples. There are five instances $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_5\}$ and two models, $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. Notice that we prefer large $C$ in SVR, so we omit the value of the regularization term. If we use L1 and L2 losses to evaluate these two models, $\boldsymbol{w}_2$ is better than $\boldsymbol{w}_1$. It means we just take care of the instances which have large target values, and other instances can be discarded.

We consider that the loss should be proportional to the target value, so we design a new loss function, relative loss.

$$\xi_\epsilon(\boldsymbol{w}; \boldsymbol{x}_i, y_i) = \max(|\frac{\boldsymbol{w}^T\boldsymbol{x}_i - y_i}{y_i}| - \epsilon, 0) \tag{2.12}$$

Using relative-loss the effect of every instance becomes equal. In Table 2.4, we can see that by evaluating the relative loss, $\boldsymbol{w}_1$ is better than $\boldsymbol{w}_2$. Further, parameter $\epsilon$ becomes more meaningful. For L1-loss and L2-loss SVR, setting a proper $\epsilon$ is difficult. Now it becomes $\epsilon$-percentage-insensitive loss function. For example, if $\epsilon = 0.1$, a target value 1000 can tolerate the prediction in $[1100, 900]$; a target value 1 can tolerate the prediction in $[1.1, 0.9]$. Besides, using a relative loss reduces the training time.

One issue of using the relative loss is that target values can not be zero. The loss will be infinity, so we would replace the zero target value with a small value.

### 2.4.2 Update Weights

We define a terminology "generation" which represents how many times the weight has been updated. Because of linear kernel, the new weight $\boldsymbol{w}^*$ can be linearly combined with the old weight:

$$\boldsymbol{w}^{t+1} \leftarrow \alpha\boldsymbol{w}^* + (1-\alpha)\boldsymbol{w}^t \tag{2.13}$$

where $\alpha$ is the learning rate, and $t$ is the generation index. Linear combination is also an advantage of linear kernel. If using a nonlinear kernel, combination will not be so easy.

## 2.5 Prediction

Computer board games usually use searching techniques to simulate possible developments. After many plies searching must be stopped, and there must be a judgment function to evaluate the score of the board. We completely only use the prediction of SVR, so our prediction algorithm is simple and quick.

The $\epsilon$-greedy policy (Watkins, 1989) is used to explore unknown situations. It randomly chooses a choice with probability $\epsilon$ and chooses the best with $1-\epsilon$ probability. We slightly modify an $\epsilon$-greedy policy, that is to set a boundary to discard too bad choices. We define $y^*$ as the value of the best choice, and accept a choice $j$ which satisfies $|y^* - y_j| < 0.3|y^*|$. Algorithm 2 is our method.

**Algorithm 2** Boundary $\epsilon$-greedy policy algorithm

```
• Assume $\boldsymbol{p}$ is the vector of all empty positions, and $\boldsymbol{w}$ is the current weights of
  the agent.
• For $j$ : every empty position of $\boldsymbol{p}$
        Put the stone at $\boldsymbol{p}_j$ and extract the feature $\boldsymbol{x}_j$.
        Calculate prediction $y_j = \boldsymbol{w}^T\boldsymbol{x}_j$ and remove the stone at $\boldsymbol{p}_j$.
• Discard too bad choices.
• Choose the best choice with probability $1-\epsilon$, or randomly choose an acceptable
  choice.
```

Because calculating prediction $y_j = \boldsymbol{w}^T\boldsymbol{x}_j$ is very fast, time complexity of prediction is determined by the time for extracting features. If extracting features takes $O(F)$, time complexity of prediction is $O(F)$.

# CHAPTER III

# Tournament Learning

Collecting large-scale data is hard. From a supervised learning viewpoint, we would collect data via playing gomoku with human, but it is impractical.

First, it needs a lot of manpower and spends too much time. If we need only thousands or tens of thousands of instances, we may collect data by playing with human. However, reinforcement learning is a trial and error procedure, so it needs new training data to revise the model. Totally it could need millions of instances, and nobody can play gomoku so many times.

Second, it is hard to keep the quality of training data. A thought is that we can design a web site and let many human players play with our agents. By this way we may collect more data, but players may consist of experts, common players and rookies. Results of our experiments show that competing with weak opponents can not improve the ability of our agents. If we can not ensure human players are better than our agent, more data will be useless.

In reinforcement learning, it often assumes that the environment would respond to the agent's actions. That is, we can unlimitedly collect data from the environment. If the environment is real world or can be simulated, the concept is practicable. However, in competitive games, like gomoku, the environment can not be simulated, because the environment is exactly the opponent, another agent. If we simulate an agent which can play gomoku, it absolutely violates our purpose.

We consider the agent can improve the ability by competing with another agent. Like we can learn by competing with peers, classmates, or opponents, and then they also can learn by competing with us. By virtuous circle, we and our opponents all become strong. If this concept works, we just wait and the agent will become an expert without humans' help.

A similar concept has been used in TD-Gammon. They called it self-teaching, or self-play. In a self-teaching framework, there is only one agent to compete with itself, but using an agent to represent two sides may not be appropriate. For example, in gomoku, playing first and playing second are very different. Black can attack more aggressively but White should focus on defense. Moreover, in some extra rules Black can not win the game by achieving Double Three or Double Four, but White can.

We use two agents to represent two sides respectively, and design a framework so that two agents can stably improve. Because the procedure continuously plays games, we call it tournament learning.

## 3.1 Tournament Learning

Tournament learning is suited for competitive games with two competitors. It is the result tested by many experiments. The framework is shown in Algorithm 3. First, both agents randomly play with opponents and update weights. Second, choosing one side to challenge its opponent until defeating the opponent. We define defeating by a winning ratio greater than 0.75. If one side can defeat its opponent, we then choose another side to challenge its opponent. Third, repeating the last two steps until one side can not defeat another in enough attempts. The maximal number of attempts is thus used as a stopping criterion. If one side can not defeat opponent in a maximal number of attempts, it may never defeat the opponent forever and we terminate the whole procedure.

**Algorithm 3** Tournament Learning for Gomoku

```
Given two agents $\{w_B^0, w_W^0\}$, one represents Black and another represents White.
Initial $\{w_B^0, w_W^0\} = \{\mathbf{0}, \mathbf{0}\}$, set maximum attempt $M$.
Choose $w_B^0$ randomly play with the opponent, then update $w_B^0$ to $w_B^1$.
Choose $w_W^0$ randomly play with the opponent, then update $w_W^0$ to $w_W^1$.
While true
    For $p =$ Black, White
        Choose agent $w_p^t$ to challenge the opponent.
        While iteration $< M$
            Challenge the opponent and update $w_p^t$ to $w_p^{t+1}$.
            If defeat the opponent, break.
        If the iteration reaches $M$, break whole procedure.
```

We use Figure 3.1 to illustrate the concept of tournament learning. There are two agents Blue and Red, and from ape to human there are five levels, $\mathsf{Blue}^1$ to $\mathsf{Blue}^5$ and $\mathsf{Red}^1$ to $\mathsf{Red}^5$. The bold arrows mean their evolutions, and the thin arrows mean the opponent they compete with. At initial stage, they are weak. The procedure starts from that $\mathsf{Blue}^1$ challenges $\mathsf{Red}^1$. After a few generations, $\mathsf{Blue}^1$ becomes stronger and evolves into $\mathsf{Blue}^2$. $\mathsf{Blue}^2$ has a good ability to overwhelm $\mathsf{Red}^1$, and then $\mathsf{Red}^1$ turns to challenge $\mathsf{Blue}^2$. After a few generations, $\mathsf{Red}^1$ also becomes stronger and evolves into $\mathsf{Red}^2$. By repeating these steps, finally, Blue and Red both have great abilities and evolve into humans.

## 3.2 Chaos Learning, Degeneration and Huge Gap

There are four parameters related to tournament learning. They are

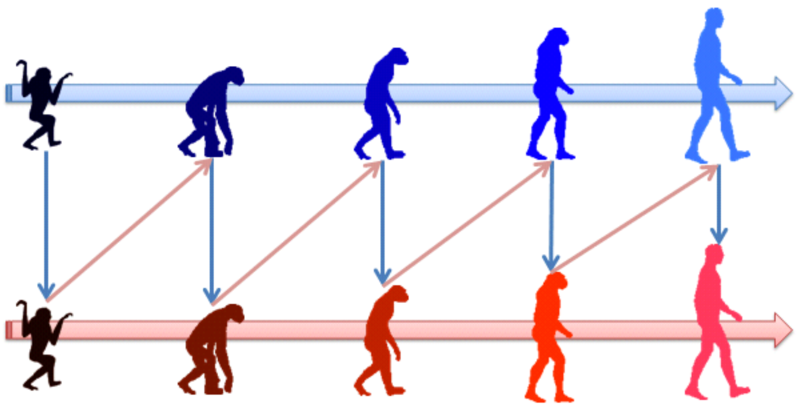


Figure 3.1: Tournament learning.

$N$: Number of instances.

$R$: Alternant ratio.

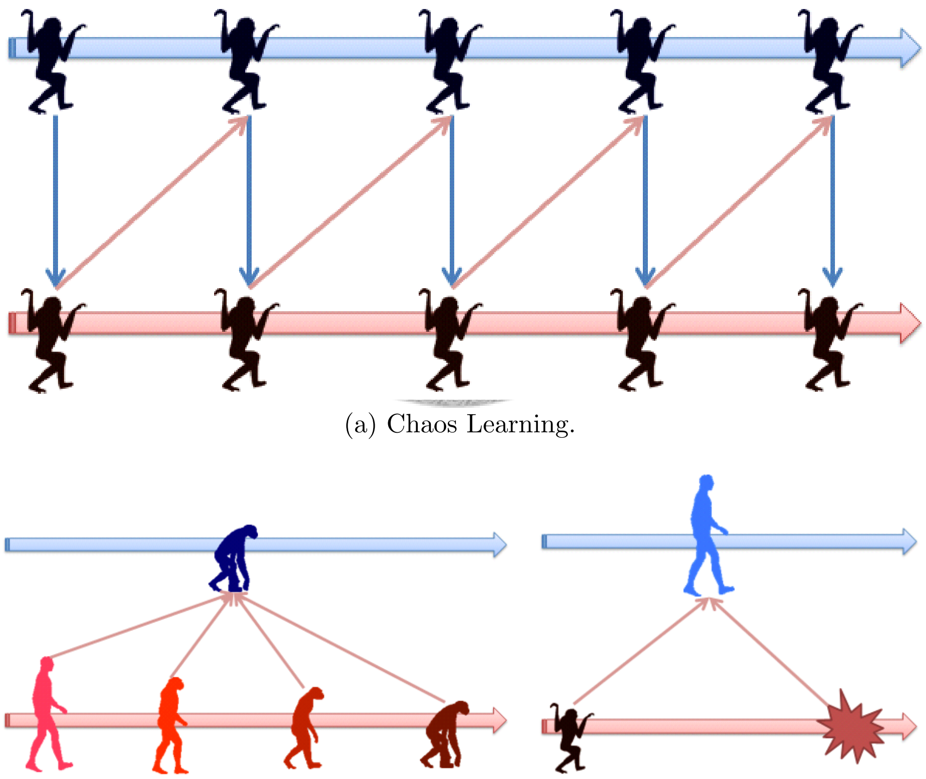
$M$: Maximal number of attempts.

$\bar{n}$: Number of non-zero features.

These four parameters are very important in tournament learning, and we will discuss them in this section. In fact, it is not easy to realize tournament learning, and after many experiments we find out three failed cases. They are chaos Learning, degeneration and huge gap.

The first case is chaos learning, illustrated with Figure 3.2a. For tournament learning, it must guarantee that the agent has a reliable learning ability. If the agent does not, it even can not defeat current opponent. In our reinforcement learning method, a reliable learning ability depends on detailed features and $N$. Features like sensor, eye or ear, provide the information in the environment; instances like experiences, and consider more experiences together is better. Thus we need to make profuse features and set a large $N$ to guarantee the learning ability.

Even with a large $N$, a low alternant ratio $R$ can lead to chaos learning. For a low $R$, like $R = 0.3$, when the winning ratio of one side is 0.5, the winning ratio of another side also is about 0.5. Because both winning ratios are greater than 0.3 , they train alternately. We consider that learning from an opponent with the same ability as ourself can give us only limited knowledge. The reason is that if the opponent is not strong, winning the game may not mean we make good choices. It is very possible that the opponent makes mistakes, so the data of this game are noise. The noise will let the agent take bad actions but still can win the game. Next turn the opponent will also learn wrong information from the agent. It is a vicious circle, and let the learning become chaos and slow.



(a) Chaos Learning.

(b) Degeneration.

(c) Huge gap.

Figure 3.2: Two special cases of tournament learning.

The second case is degeneration, illustrated with Figure 3.2b. While we defined the defeating opponent, we had considered that the agent should defeat more than one opponent. Otherwise, overfitting may occur. However, we found out the performance may become worse, because we play with some weak opponents. To confirm it, we design an experiment that let a strong agent always learn from a weak agent. The result is, although the strong agent always can defeat the weak agent, its ability does not improve. For example, a professional basketball player always plays with a child. He can easily defeat a child, but after a long time he will forget his skills. Thus, keeping the quality of opponents is very important. Bad opponents are harmful for learning.

The third case is huge gap, illustrated with Figure 3.2c. It is a reversed case of degeneration. In the beginning of this chapter, we have mentioned that collecting data via playing with human is not proper. When the agent is weak, defeating human is very difficulty and it would collect a data which all instances with a negative reward. Next generation, it still collects a data which all instances with a negative reward. No matter what actions it takes, it always receives bad rewards, so it can not find out a good model. After a few generations the model will collapse, because all weights are negative and close to the final reward.

Nonetheless, if the agent has a very powerful learning ability, it could avoid the situation. However, a powerful learning ability depends on a large $N$, so collecting data via playing with human is still hard to realize.

# CHAPTER IV

# Experiments

## 4.1 Conditions and Settings of Experiments

The ability of an agent is very related to features. To defeat human expert, the agent needs large and detailed conditions. Because we make hundreds of conditions, we just describe them roughly. There are five types of conditions:

1. **Fundamental conditions**: Fundamental conditions are shown in Figure 2.3. However, we slightly revise it. We prefer a condition which is greater or less than a number not equal to a number. For example, we would like use a condition like the number of Open Three is greater than zero, but not equal to zero.

2. **The threats of one occupied intersection**: The threats mean some threatening conditions, like Open Three, Half Four and Open Four. An intersection has four directions, so some intersections may have more than two threats, like Double Three, Double Four and Four Three. These intersections are critical.

3. **The threats of one empty intersection**: Some empty intersections where we put a stone can achieve the conditions of type 2, They are also critical, because we must prevent that the opponent achieves these conditions.

4. **The positions during primary stages**: Most intersections are empty during the primary stage, so the conditions are fewer. Position is the rare information we can use. It also called "chess opening", but ours is simpler. In first three steps, the stone in the square consist of $(6, F), (6, J), (10, F), (10, J)$ is a condition, and else the square consist of $(4, D), (4, L), (12, D), (12, L)$ is another condition, and else is another.

5. **Manual link above conditions**: Some conditions could change its weights if other specific conditions happen. For example, Open Three is a positive condition, but if the opponent has already achieved Half Four, we must defend Half Four first. Thus, the weight of Open Three would decrease in this case. If we add a new condition which includes these two conditions, the new condition will help Open Three to keep its positive weight. This method is very like low-degree polynomial data mappings (Chang et al., 2010), but to restrict the number of features we only link the important conditions manual.

To avoid overfitting, after collecting data, we delete the features which rarely appear. We delete the features whose numbers of appearances is less than 5. Because the agent has different strategies at different generations, the numbers of non-zero features $\bar{n}$ are different. Average $\bar{n}$ are between $2,000$ to $2,500$. Number of instances $N$ and alternant ratio $R$ are variables, and we will try some combination in experiments. The reward function is fixed. We set $k$ to be four and $\gamma$ to be 0.3. In order to make observation, we do not set the maximal number of attempts and the procedure is stopped manually when achieving our purpose.

The parameters of LIBLINEAR are fixed. Although different parameters could affect the performance, we would like to focus on reinforcement learning rather than supervised learning. We use relative loss and the parameters are -p 0.1 -c $100,000,000,000$

-B 0 -e 50.

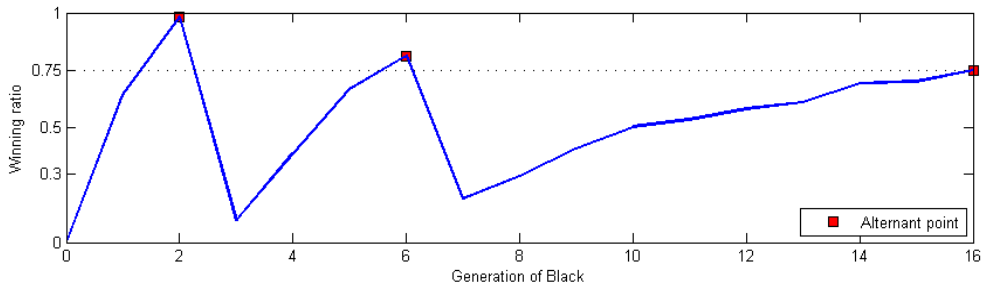
## 4.2 Episode of Tournament Learning

In this section, we examine the episode of tournament learning. Figure 4.3 is the results. The $x$-axis is the generation and $y$-axis is the wining ratio. The alternant point means that one side defeats the opponent and take turns to train, so after an alternant point the agent should face a stronger opponent.

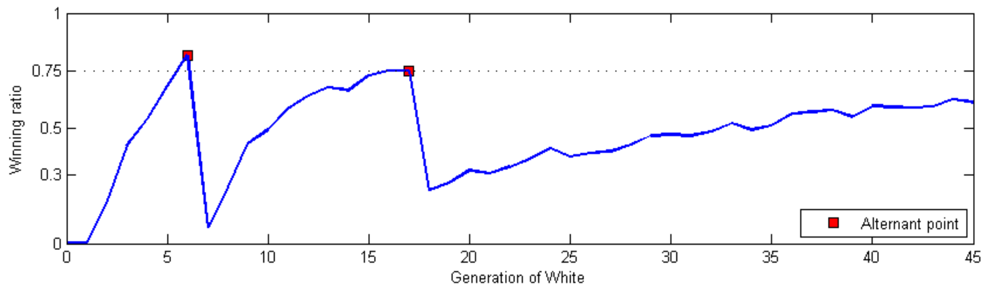
Figure 4.1a and Figure 4.1b use $N = 15,000$ and $R = 0.75$. They respectively represent result of Black and White. It is an ideal tournament learning. We can see, for a fixed opponent, our reinforcement learning works very well. Both sides can increase their ability very stably. During the whole procedure, there are only three alternant points on Black's side, and after Black's 16th generation White can not defeat Black. The learning rate will be slower after a few alternant points, because the ability of the opponent is stronger. It also means that the set of features reaches its limitation. If we want to increase the ability of the agent, we need to add more useful features.

We examine the variable $N$. We have mentioned that large-scale data is one of our core contributions. Figures 4.1c and 4.1d are the evidences. The parameters are $N = 500$ and $R = 0.75$. We can see the figure is very zigzag, and it means the learning is unreliable. When one side defeats the opponent, we are unsure if it happens by chance or not. Therefore, for stable learning, large-scale data is essential.

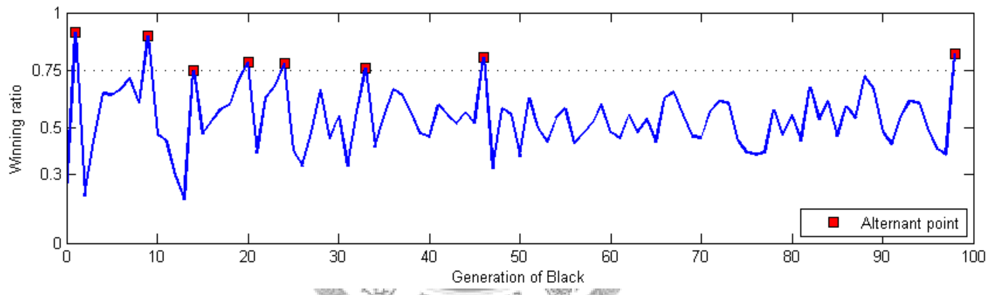
Figure 4.2a and Figure 4.2b are another pair of tournament learning. The parameters are $N = 1,500$ and $R = 0.3$. The result shows that after few alternant points they train alternately. It could lead to chaos learning. Another disadvantage is that with lower $R$, it is difficult to decide $M$, the maximal number of attempts.



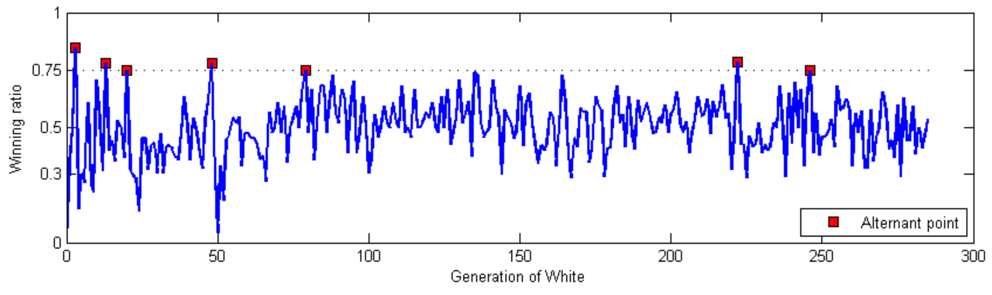
(a) $N = 15,000$ and $R = 0.75$, Black's episode.



(b) $N = 15,000$ and $R = 0.75$, White's episode.

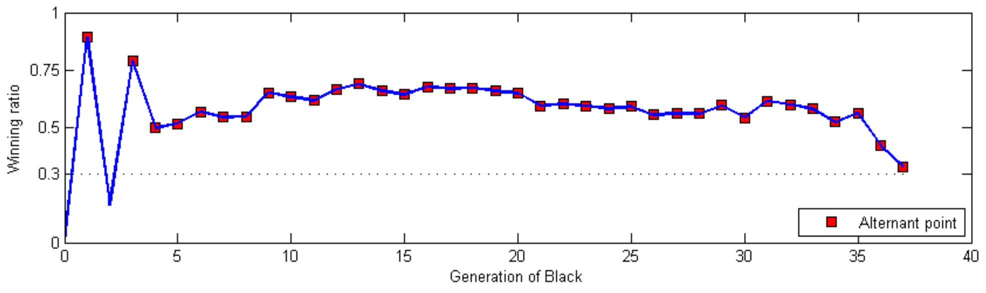


(c) $N = 500$ and $R = 0.75$, Black's episode.

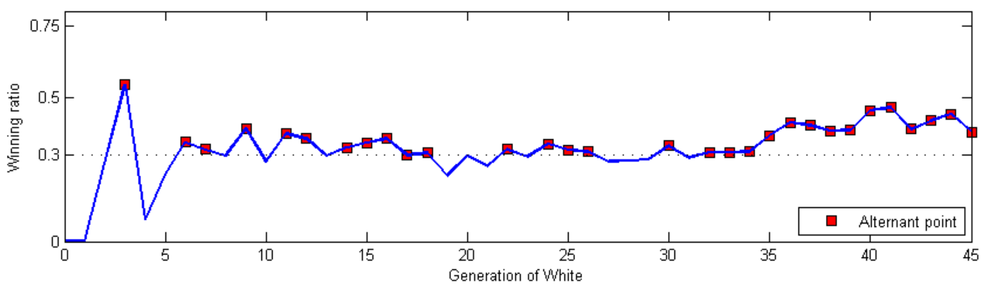


(d) $N = 500$ and $R = 0.75$, White's episode.

Figure 4.1: Episode of tournament learning.



(a) $N = 15,000$ and $R = 0.3$, Black's episode.



(b) $N = 15,000$ and $R = 0.3$, White's episode.

Figure 4.2: Episode of tournament learning.

## 4.3 Competing against Unfixed Opponents

Notice that in tournament learning, the agent faces to a fixed opponent, and we want to ensure that the ability still increases stably when facing different opponents. Unfortunately, there is no standard to evaluate the level of gomoku, so we choose four Black agents to be targets. They are trained by tournament learning with parameters $N = 15,000$, $R = 0.75$, $k = 4$, $\gamma = 0.3$, and the generations of $5, 7, 10, 15$.

We examine the winning ratio of every generation of White, and every generation competes with four targets 50 games respectively. Figure 4.3 shows the results of comparison between different $N$ and $R$. The $x$-axis is the generation of White and $y$-axis is the wining ratio.

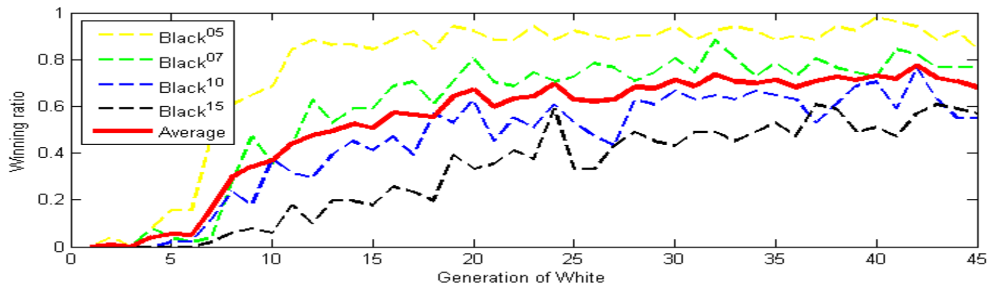
We can see that when facing different opponents the figure fluctuates more, but the ability still stably increases. Figures 4.3a, 4.3b and 4.3d are corresponding to Figure 4.1b, 4.2b, 4.1d. We can see the agent trained by ideal tournament learning gets the best performance (Figure 4.3a). Although the final average winning ratios in Figure 4.3a and Figure 4.3b are almost the same, the accumulated value of generations are $16 + 45 = 61$ and $37 + 45 = 82$, so actually Figure 4.3a is 1.3 times faster than Figure 4.3b. It implies that with a lower $R$ and a large $N$, the agent still can improve its ability to a strong level, but it takes longer time.

Comparing the winning ratio against $\mathsf{Black}^{05}$ between Figures 4.3a and 4.3b, we find out something interesting. $\mathsf{Black}^{05}$ is a weak agent, so after few generations White should overwhelm it. It happens in Figure 4.3a but dose not in Figure 4.3b. Because in Figure 4.3a, the Black agent learns stiff basic concepts in the primary stage, White agent can get correct information without noise. Actually Figure 4.3b is close to chaos learning. White agent always gets noisy data, but with large $N$ it still can increase the ability stably.

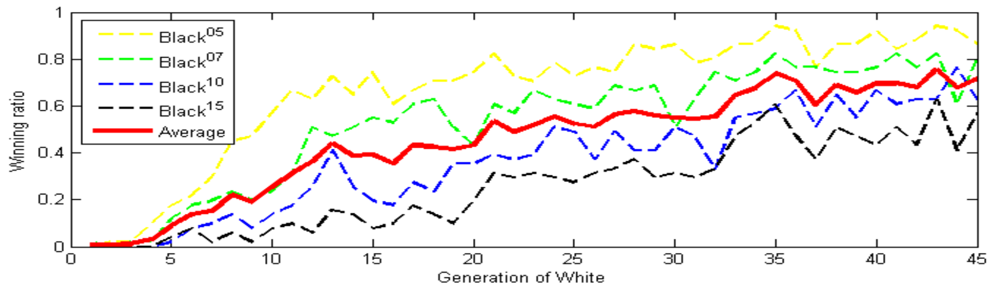
With a larger number of training data $N$ the winning ratio increases more stably, but it seems unnecessary to set a too large $N$. The agent trained by parameters $N = 5,000$, $R = 0.75$ can achieve 0.7 average winning ratio at 40 generations, but spending the same time the agent trained by parameters $N = 15,000$ and $R = 75$ is just at generation 13, and winning ratio is just about 0.5. It is a trade-off between time and stabilization. Empirically, the parameter $N$ at least should be large than $3\bar{n}$.

## 4.4 Experiments of Degeneration and Huge Gap

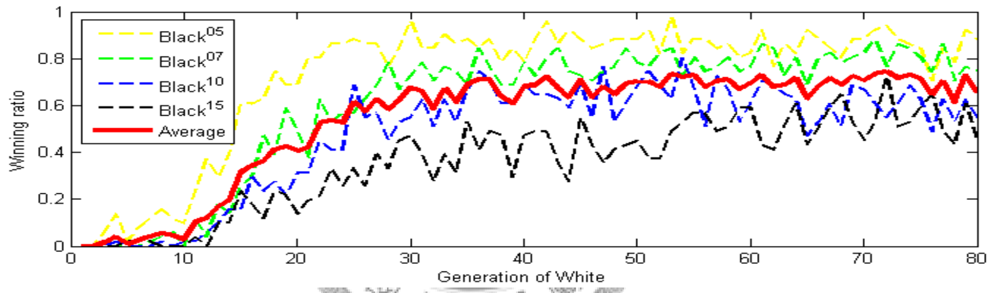
Figures 4.4a, 4.4b, 4.4c are three experiments of huge gap. Three agents directly learn from a strong opponent, $\mathsf{Black}^{15}$. The learning ability depends on the number of instances $N$, so the agent in Figure 4.4a has the best learning ability, and the agent in Figure 4.4c has the worst. Because their learning abilities are different, huge gap only happens in Figure 4.4c. We can see the model absolutely collapses after 100th



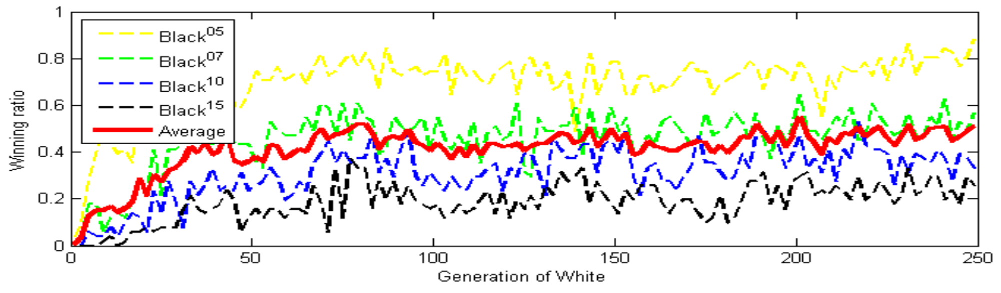
(a) $N = 15,000$ and $R = 0.75$,



(b) $N = 15,000$ and $R = 0.3$,



(c) $N = 5,000$ and $R = 0.75$,



(d) $N = 500$ and $R = 0.75$,

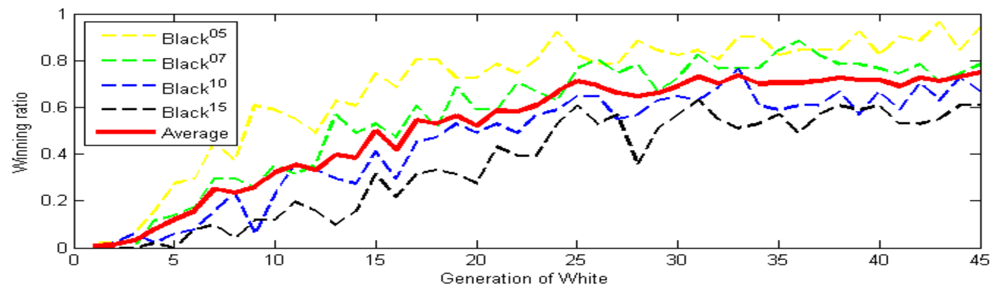
Figure 4.3: The performance when facing different opponents.

generation in Figure 4.4c.

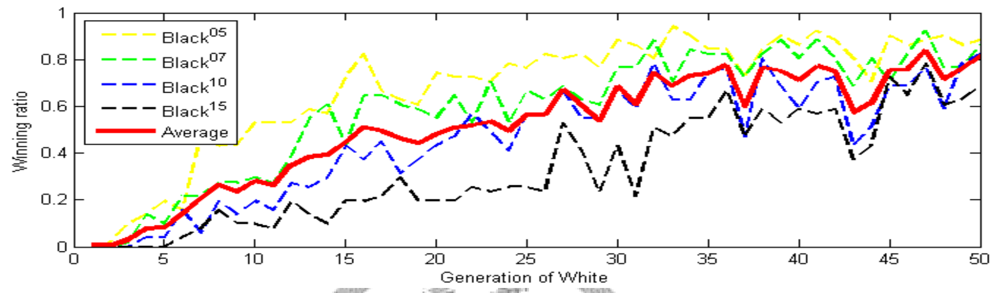
Figure 4.4d is the experiments of degeneration. To distinguish from chaos learning, we set the agent to have a powerful learning ability, $N = 15,000$. In the first 45 generations, it directly learns from Black$^{15}$, and in the rest 45 generations it learns from Black$^{05}$. We can see that after 45th generation, the agent overwhelms Black$^{05}$ and the winning ratio almost reaches 100%, but the winning ratio against Black$^{15}$ and Black$^{10}$ decreases. Fortunately, although the degeneration really happens, it dose not happen suddenly.

## 4.5 The Gomoku Game

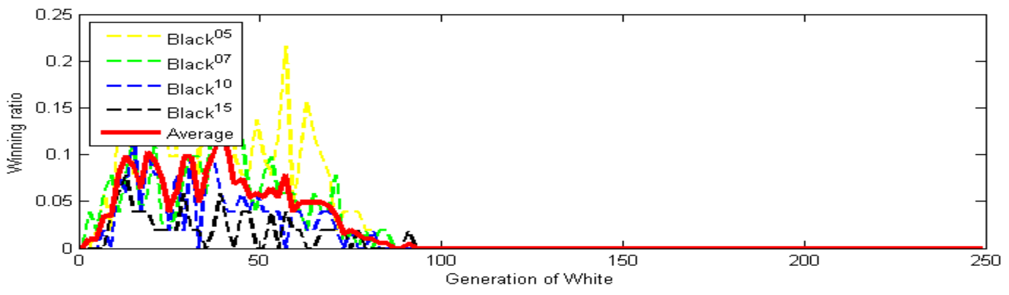
We design a Java applet to demonstrate the abilities of our agents. It is available at `http://www.csie.ntu.edu.tw/~r97144/TL/TL_v01.html`.



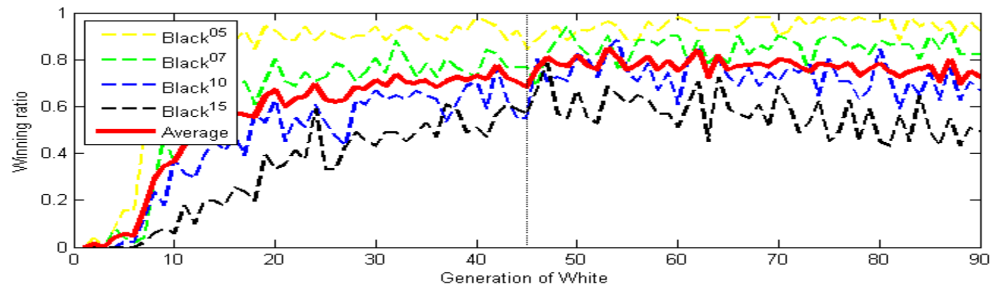
(a) $N = 15,000$, White agent directly learn from Black$^{15}$.



(b) $N = 5,000$, White agent directly learn from Black$^{15}$.



(c) $N = 500$, White agent directly learn from Black$^{15}$.



(d) $N = 15,000$, White agent directly learn for Black$^{15}$ in first 45 generations, and learn from Black$^{05}$ in rest generations.

Figure 4.4: Degeneration and huge gap

CHAPTER V

# Conclusions and Future Work

In this thesis, we use the modern techniques of supervised learning to improve the ability of reinforcement learning. Past reinforcement learning just can deal with easy problems. Our method can deal with complex problems like gomoku, and without any searching technique it achieves a competitive level. It implies our method has a powerful learning ability. We also realize tournament learning, and by experiments we see some interesting observations.

Our work is just a preliminary study. There are many future works which have great potentials.

1. Add short searching feature. In the thesis, to proof our method is a pure reinforcement learning, we discard any search features. However, if we add some such features, the agent's ability will be greatly increased.

2. Solve the problem that can not be solved by searching. In board games, go is the hardest problem. Using searching techniques becomes impractical because the search tree is too huge. Our method is more like human brain, because it estimate every step by conditions or trends.

3. It can be parallelized easily. The bottleneck of our method is collecting data. Fortunately, collecting data can be parallelized. Figure 5.1 is the framework. The root

computer transits two agents' model to leaf computers, and then every leaf computer starts to collect data. Finally, the root computer collects data from all leaf computers and trains the whole data. After updating the agent's model, we repeat the above steps. If we can collect more data in acceptable time, we can add more features, and the learning ability will be greatly increased.

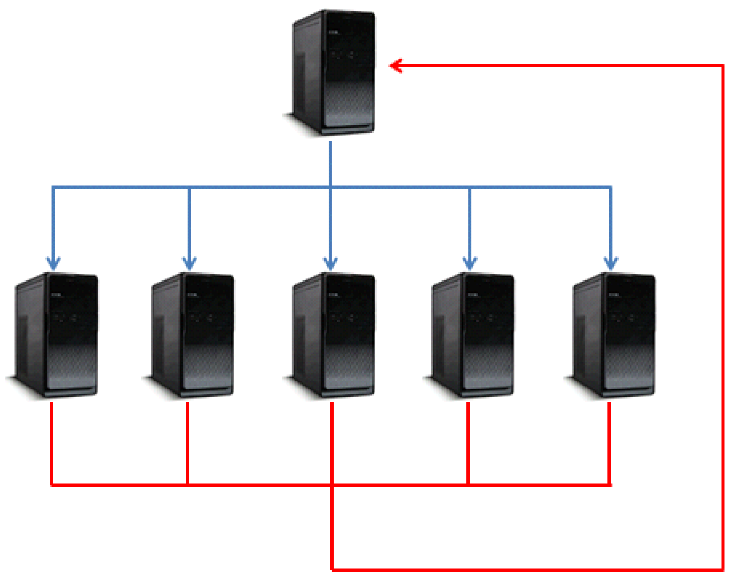


Figure 5.1: The framework of parallel processing.

# BIBLIOGRAPHY

A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and sequential decision making. In *LEARNING AND COMPUTATIONAL NEUROSCIENCE*, pages 539–602. MIT Press, 1989.

R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490, 2010. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/lowpoly_journal.pdf`.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 271–278, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-274-7. URL `http://dl.acm.org/citation.cfm?id=645753.668239`.

K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:219–245, 2000.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf`.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

N. J. Nilsson. Introduction to machine learning. an early draft of a proposed textbook, 1996.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In *Proceedings of the Twenty Fourth International Conference on Machine Learning (ICML)*, 2007.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetts, London, England, 1998.

G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38:58–67, 1995.

M. Tokic, W. Ertel, and J. Fessler. The crawler, a class room demonstrator for reinforcement learning. In *Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference (FLAIRS 09)*, Menlo Park, California, 2009. AAAI Press.

V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, NY, 1995.

C. J. C. H. Watkins. Learning from delayed rewards. In *PhD thesis, Cambridge University*, 1989.

C. White. A survey of solution techniques for the partially observed markov decision process. *Annals of Operations Research*, 32:215–230, 1991. ISSN 0254-5330. URL `http://dx.doi.org/10.1007/BF02204836`. 10.1007/BF02204836.