Master's Thesis

Bad Move Detection and Playstyle Production using Deep Learning Go Programs

FAN Tianwen

Supervisor Kokolo Ikeda

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Master)

September, 2020

Abstract

Computer Go programs have already exceeded top-level human players by using deep learning and reinforcement learning techniques. In 2016, AlphaGo developed by DeepMind defeated Lee Sedol, In 2017, its successor, AlphaGo Zero, defeated Ke Jie, the world's first-ranked human player at that time. The chasing for strength is enough for human players.

On the other hand, "Entertainment Go AI" or "Coaching Go AI" are also exciting directions that have not been well investigated. Several kinds of research have been done for entertaining beginners or intermediate players. "Bad move detection" and "Playstyle Production" are important tasks in the area of education and entertainment Go. In the previous researches, they are proposed and evaluated using a traditional Monte-Carlo tree search program. In this research, we try to evaluate the approaches using Leela Zero and KataGo. There are some critical differences between the previous program and the new programs. For example the new program does not use random simulations to the ends of games, then the previous method for producing various playstyles cannot be used. Also, we want to check how well previous approach performs in strong Go programs. Leela Zero is designed under the structure by AlphaGo Zero, while KataGo tries to improve AlphaGo Zero's process and architecture.

Bad move detection is an important task in educational Go programs, which can help human players improve their play by pointing out the program's good/bad moves. In the previous researches, data with bad labels are used in machine learning. The learning result showed that it is useful for intermediate players. In this research, we try to evaluate the machine learning again by KataGo, as the strength of KataGo has already surpassed human top players, also, some new features, such as "territory difference" and "prior selection probability", are much preciser than the previous program, the calculated new features may help to produce better models of machine learning. The result shows that the weighted average F-measure of good/bad moves is better than the previous research.

Playstyle Production is another vital task in entertainment Go programs, Which can entertain human players by using some specific playstyle while without apparent strength loss. In the previous researches, a Monte-Carlo Go Program used an online method to produce some playstyles, human players can feel some of these playstyles clearly. As the searching mechanism of Leela Zero is totally different from the previous Monte-Carlo program, in Monte-Carlo Go, online method is used for playstyle production, it is worth

to produce playstyles by new offline method. In this research, we try to train models to produce playstyles by Leela Zero. Human projects are used for evaluation, which claims that human players could identify the produced playstyles (center and edge/corner) with a high probability.

Acknowledgement

Above all, I would like to express deepest gratitude to my advisor Professor Kokolo Ikeda of the School of information Science at Japan Advanced Institute of Science and Technology. During the time I study and research under his guidance, I have received many supports, encouragements, and many valuable comments from him. Even though I have made many mistake, he was always patient and steered me out with his experience and immense knowledge.

I would like to thank to my parents for supporting me all the time. I love you! I also want to express my very profound gratitude to my sister, who is strong and independent. She always guides the right way in my life. I love you!

Finally, I would like to say thanks to my friends in Lab. I lived a very happy and joyful life here. Thank you very much!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The game of Go requires an outstanding decision-making ability and has been treated as an essential goal of Artificial Intelligence(AI) for a long time. In recent years, the longstanding challenge for Computer Go at a professional human level has been successfully tackled. In 2016, AlphaGo, one Go programs developed by DeepMind, defeated a legendary human professional player Lee Sedol [1]. In 2017, AlphaGo Zero [2], a revised and advanced version of AlphaGo, defeated Ke Jie, the world first ranked human player at that time. The chasing for muscular strength of Computer Go has already been enough.

Besides the area of researches about how to make a strong Computer Go Program , the researches about how to make progresses in the topics of education and entertainment Computer Go Programs is also in the process and becomes worth taking. There are many tasks in the researches of Education and Entertainment Go Programs. One regular but important task in the educational area is to point the bad moves to show players' mistakes and then explain how to improve them. Another task in the entertainment Go area is to use some strategies when playing with human players.

In the past research works, researches about bad move detection and playstyles production have been taken in Monte-Carlo tree searching Computer Go Programs with strength about amateur 3-dan.

As the development of Computer Go, two Go programs, Leela Zero and KataGo, which are two strong Computer Go AI designed according to AlphaZero[2] are easy for using. Their strength have already surpassed human top players.

For bad move detection, 1) we want to check and evaluate how the previous approaches work by using KateGo; 2) some new features, such as "territory difference" and "prior selection probability", are much precise than the previous program, the calculated new features may help to produce better models of machine learning.

For various playstyles production, 1) we want to evaluate the original methods in one different strong Go program; 2) one offline method, which is different from online method by using Monte-Carlo Go program, is approached.

In this paper, there are mainly two tasks we are facing. The first is aiming at evaluating the original methods in Bad Move Detection using KataGo. The second is producing various playstyles using Leela Zero.

In more detail, in the previous Bad Move Detection task, some machine learning methods are used to classify bad and good moves. A Monte-Carlo Go Programs Nomitan [11] calculates many features extracted from matches generated by fight with human players. We will re-evaluate the matches while calculating the features again by KataGo. The new evaluated features will be used for machine learning. In the previous playstyle production, an online method is used for producing specific playstyles. As the searching mechanism in KataGo is totally different from Monte-Carlo Go program used in the previous research, we will try to produce these playstyles by training models, in which deep learning will be applied.

In this paper, chapter 2 is used for introducing the game of Go and the development of Go Programs in recent years. In chapter 3, the related works and previous researches about Bad Move Detection will be introduced. In chapter 4, the related works and previous researches about Playstyle Production will be introduced. In chapter 5, our new approaches in Bad Move Detection and Playstyle Production will be claimed and explained. In chapter 6, experiments about Bad Move Detection will show how well the result will be compared to the previous research. Experiments on Playstyle Production will show how well human players can felt playstyles. In the last chapter, some conclusions and future work will be introduced.

Chapter 2

Go Programs

In this chapter, introduction before section 2.1 is written and modified based on the wikipedia of Go [22].

Go is a complex board game that requires intuition, creative and strategic thinking. Go has been considered as a difficult challenge in the field of artificial intelligence (AI) and is considerably more difficult to solve than chess. Many researches in the field of artificial intelligence consider Go to require more elements that mimic human thought than chess.

Prior to 2015, the best Go programs only managed to reach amateur dan level[3]. On the small 9×9 board, the computer fared better, and some programs managed to win a fraction of their 9×9 games against professional players.

In 2016 and 2017[1] [2], as the AlphaGo and AlphaGo Zero defeated professional human players, the chasing for 30 strong strength of Computer Go temporarily come to an end. At the same time, the researches of entertainment and education Go Programs became necessary topics. How to use strong Computer Go to help human players to get fun and advance their skills becomes another topic worth to making researches.

2.1 The Game of Go

Go is a complete information game for two players, in which the aim is to surround more territories than the opponent. The game was invented in China more than 2,500 years ago and is believed to be the oldest board game continuously played to the present day.

The rules of the game Go are easy to understand, it is played on a 19×19 grid using black and white stones.

2.1.1 Basic Rules

This section is written based on introduction of Go from britgo [23],

A game of Go starts with an empty board. Players take stones to put on the intersections on the board, usually "19x19" for formal matches, other board sizes, such as "9x9" and "13x13" are used more for excise and entertainment. The main object of the game is to use your stones to form territories by surrounding vacant areas of the board. It is also possible to capture your opponent's stones by completely surrounding them. The winner of one match decides by the territories or scores which are surrounded or captured by players, moreover, black will compensate some specific scores when taking the last score calculation for black always w plays first, the compensated scores are called "komi".

Liberties and Capturing Stones

One basic and important concept is Liberties of moves. As Figure 2.1 shows, when one stone is placed on the board, the empty points which are horizontally and vertically adjacent to a stone, or a solidly connected string of stones, are known as liberties, such as Black A has 4 liberties marked with "X", White B has 3 liberties and Black C has 2 liberties.

If all the Liberties of one stone of a string of stones are occupied by opponent's stones, the surrounded stone(s) will be captured and removed from board. As Figure 2.2 shows, if White plays at position A, all the Liberties if Black 1 are occupied by White, so Black 1 will be removed if White plays at position A. Moreover the White A is called "Atari". Black B and White C are "Atari" too.

2.1.2 Life and Death Problem

Life and Death Problem is one common but important problem when playing Go. As the Figure 2.3 shows that, if black wants to make the block alive, Black needs to play at position A, or black will have only one eye when white plays A and becomes dead.

2.1.3 Basic Stages of One Game

One complete match of Go can be roughly divided into three periods of stages.

1. Primary Stage (fuseki)

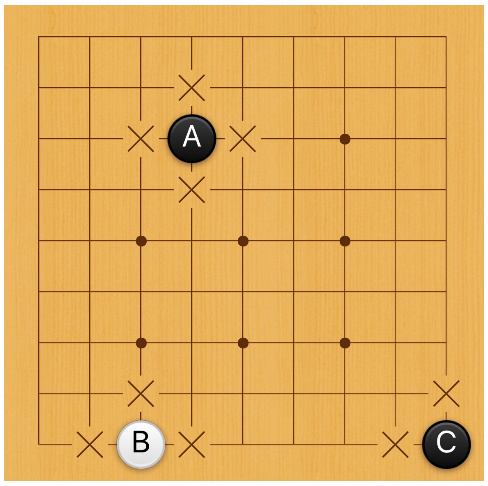

Figure 2.1: Liberties in Game of Go

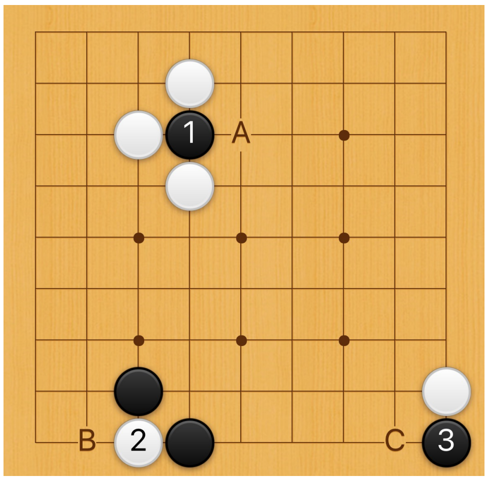

Figure 2.2: Capturing Stones

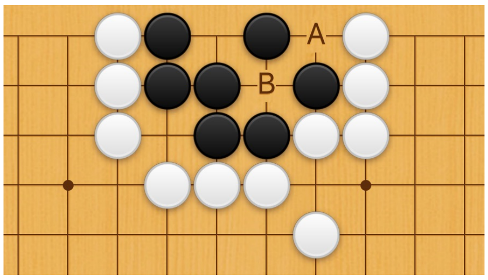

Figure 2.3: Life and Death Problem

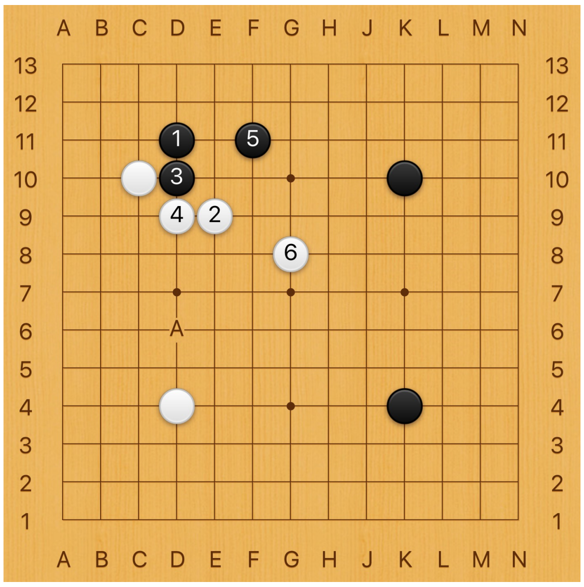

Figure 2.4: Primary Stage

2. Middle Stage

3. Terminal stage (yose)

The primary stage of Go is also called by “fuseki” in Japanese. In this stage, both sides of the players capture open space on the board, and try to prevent the other side from occupying the territories, which leads to the middle battle. Usually during the primary stage, players tend to play with some strategies, such as try to play moves near to the edge/corner to get more territories or play moves near to the center for influence, it is also called “playstyle” which will affect throughout one entire match.

As the Figure 2.4 shows that, this is one episode of primary stage, which decides the basic and vague boundaries of Black and White.

In the middle stage of Go, players use kinds of skills and techniques and try to defeat the other player, or gain advantages until entering into the terminal stage. Figure 2.5 is one example from FAN Yunruo(Black) versus. YANG Dingxin(White) played on 17th, March, 2020 . It shows that the vague boundaries near to the edge and corner of white and black are determined, while the territories near to the center are not determined. The last black move A try to surround the center by attacking white stones marked with “x”. The result of the fight in the center, how white “x” become safe, plays an important role during the whole match.

In the terminal stage of Go, the territory and Life and Death prob-

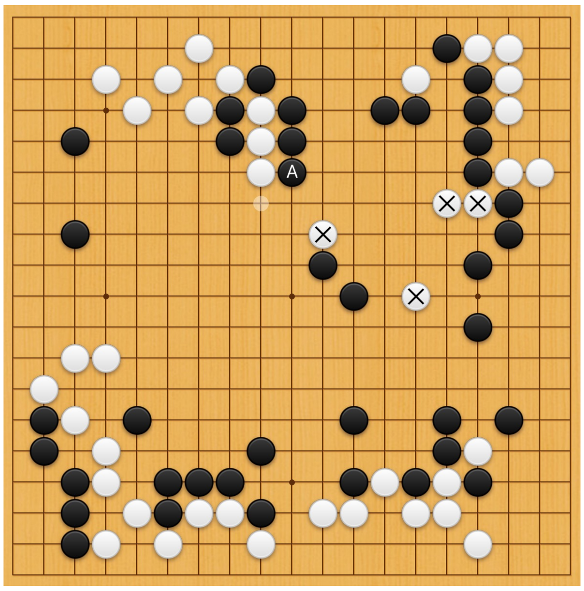

Figure 2.5: Middle Stage

lem(referred in section 2.1.2) have been roughly determined after fighting in the middle stage, and the competition boundary is established roughly. Furthermore, terminal stage of Go is usually relatively static compared to Shogi. In the case of Shogi, player must attack with risk to win, but in the case of Go, peaceful moves are possible.

Figure 2.6 shows one terminal stage of one Go match played by Lee Chang-ho (Black) versus Cho Hun-hyun (While). It is very clear that the boundaries for black and white are determined.

2.2 Monte-Carlo Go Programs

This section is written based on “Monte Carlo Tree Search in Go” by A. Couetoux [21] and Sensei’s Library [24].

Before the appearance of AlphaGo and AlphaGo Zero, the main structure of a Go program used Bradley-Terry Model for policy selection and Monte-Carlo search under the guidance of the Upper Confidence Tree (UCT) to decide the next move.

The main idea of Monte-Carlo Tree Search (MCTS) is to construct a search tree step by step repeatedly, choosing the most promising child node of the already searched tree, expanding it, and then evaluating the new leaf with random self-play simulations. The result of the simulation (win or loss value), is back-propagated in all the parent nodes up to the root, and a new

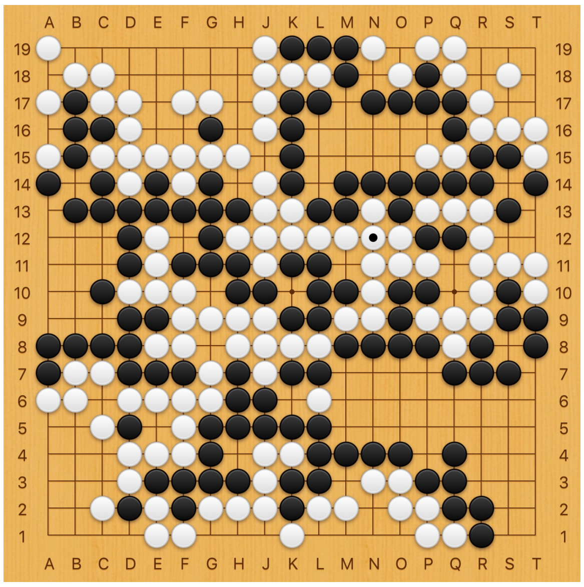

Figure 2.6: Terminal stage

most promising leaf-node is chosen [7].

Initial Monte-Carlo methods used random simulation, but it soon became clear that if heuristic knowledge can be used as a policy to obtain realistic moves, the strength of Go can advance a lot. In 2007, Coulom introduced the Bradley-Terry model to learn automatically the weights of features associated to the moves the game [8]. Several strong Go programs, using this kind of model have been designed, various detailed features are used for improving accuracy when entering into MCTS. Bradley-Terry model is very important for the Go Programs before the appearance of AlphaGo and AlphaGo Zero. But in this research, we will not focus on how Bradley-Terry Model works in Monte-Carlo Go programs, you can refer it in the Appendix B.

Main Loop of MCTS

In this section, how MCTS works will be introduced and explained here. The algorithm can be described as follows: One initial node is created from the information gathered during the simulation of one game, which will be the root for a Tree, then the child node is added to the tree by searching. By repeating the simulation and searching process, the tree will be expanded. MCTS then repeats the main loop, until the computation time is over. It then uses the information stored in the Tree to select a move according to a selection(action) function. Normally, the function is a heuristic function related to the game itself, which helps to select the best node.

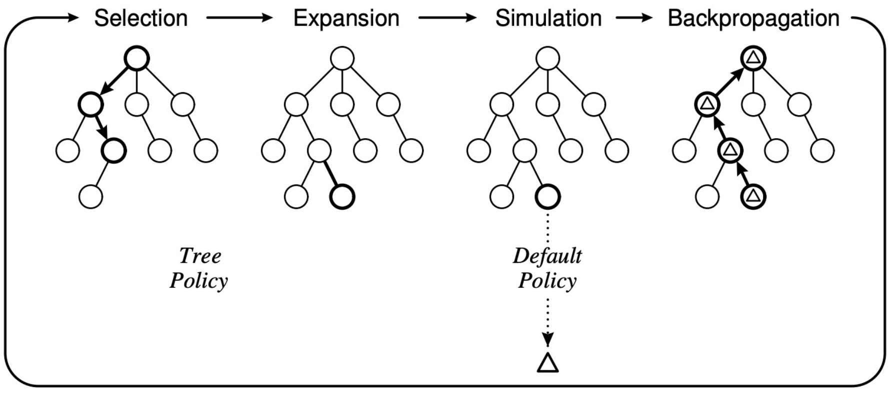

Figure 2.7: The four steps of the main loop of MCTS. Each iteration of the loop corresponds to one possible sequence of states and moves, and adds information to the tree.

A commonly used final selection function is to pick the most simulated move. Such for the game of Go, it will select the candidate move with the most simulation time, here we call one simulation a “Playout”. The main loop of MCTS can be divided in four steps, as illustrated by Figure 2.7, borrowed from [9]:

- Selection: It starts from the root node R and select a node that needs to be expanded most urgently till the leaf node L.
- Expansion: If the leaf node L is not a node with terminal state, the it will be added to the tree, and one node C will be selected for simulation.
- Simulation (playout): It starts from the new node C, one simulates a possible process of the game until a final state is reached, where the result of the game is known;
- Back Propagation: If the the last simulation's result is given, the tree will be updated typically by statistics in all the nodes visited during this last simulation.

We formally describe this loop in Algorithm 1. The transition function is specific to the problem, and is assumed to be known. In the case of Go, this function models the rules of the game and is deterministic. This loop clearly shows three essential parts of MCTS: (i) the selection policy, (ii) the simulation procedure, and (iii) the update of the tree. As for the simulation procedure, a standard choice is to apply a default policy until a final state is

Algorithm 1: MCTS algorithm

```
Result: add next move to the tree $T$
Initialize $s \leftarrow root(T)$ ;
while $s \in T$ AND $(s! = final\ state)$ do
    $m$ = Selection(s) ;
    $s$ = Transition(s,m) ;
end
$result = Simulation(s)$ ;
$T \leftarrow Update(T, result)$ ;
```

reached, where the result of the game can be computed. An easy choice for that default policy is to choose random moves, but some default policies can improve the overall performance.

Using UCT-method for simulation

UCT is MCTS using the Upper confidence Bound algorithm in each node [14]. The Upper Confidence Bound is the deterministic algorithm for solving a Multi-armed bandit problem. It is used to find the bandit with the biggest expected value in a probability distribution. UCT chooses always the maximum value of an action in the selection step.

As mentioned in the previous section, the MCTS program searches for moves with a higher winning rate, which are derived from playing at least a few hundred random games. MCTS will only collect all the statistics of all root nodes, but it is difficult to converge to the best strategy through random selection during expansion.

The UCT method can be naturally applied to the MCTS process of the Go program, because the branching factor of each node may be too large. The first few steps of each game are selected by searching the tree growing in memory, and once the terminal node is found, the new step/sub is added to the tree, and the rest of the game is played randomly. Then, the evaluation of the completed random game will be used to update the statistics of all moves in the tree that belong to the game. Theoretical results were presented showing that the UCT enables selecting the optimal action(move) converges as the number of samples(simulations) grows to infinity [15].

The UCT method solves the problem of selecting actions in the tree, so that important actions are searched more frequently than seemingly bad actions by random selection.

UCT also chooses the best action all of the time, but also explores other actions in a more complex way. It does this by adding a number to the winning percentage of each candidate move that drops every time the node is visited. However, every time you visit your parents and choose other actions, this number also increases a bit. This means that the not only the winning rate, but also the visited count of unexplored moves will increase. This enables that:

- If the node (candidate move) can help to win the game, it tends to be selected more with bigger winning rate.
- If the node (candidate move) may cause to lose the game, it tends to be selected less with smaller winning rate.

In the MCTS of game of Go, node with bigger playout times will be easily selected, while not the winning ratio.

The sum of the winrate and the number of playout for each candidate move n is computed with:

$$UCT_j = X_j + C * \sqrt{\frac{ln(n)}{n_j}} \tag{2.1}$$

where:

- X_j is the win ratio of the child
- n is the number of times the parent has been visited
- n_j is the number of times the child has been visited
- C is a constant to adjust the amount of exploration and exploitation

The first component of the formula above corresponds to exploitation, as it is high for moves with high average win ratio. The second component corresponds to exploration, since it is high for moves with few simulations.

2.3 AlphaGo and AlphaGo Zero

As introduced in the previous section 2.2, the Bradley-Terry model is used for policy prediction, the resulted candidate moves will be used as the root in the next Monte-Carlo Searching Tree. Although the MCTS with the guidance of both heuristic UCT and RAVE make MoGo became the first program to achieve human master level in competitive play in the domain of

9x9 Computer Go [13], in the domain of 13×13 or 19×19, such traditional MCTS program could not compete with top human professional player.

AlphaGo is the first computer Go program to defeat a professional human Go player, the first to defeat a Go world champion, and was the strongest at the time in history. The policy network trained from human players' game records help the node expansion in MCTS more precisely and efficiently [?].

AlphaGo Zero is the advanced version of AlphaGo, there are two new features comparing to the AlphaGo. 1). with only basic knowledge of the game of Go, it is trained by self-playing. 2). it combines policy network and value network into one. The two features had made AlphaGo Zero a Go program with stronger strength [2].

2.4 Nomitan, Ray, Leela Zero and KataGo

As have noticed in section 2.2, Nomitan is a Go program, which is an implementation of Bradley-Terry model and MCTS algorithm. It reached a rank of 3 dan on the KGS server under the account nomibot.

Ray is another Monte-Carlo Go Program in this research [6], it is also built by the model, which is about 2 dan on the KGS server.

At the same time, as having been noticed in section 2.3, there are mainly two computer Go program used in this research. They are Leela zero and KataGo.

Leela Zero is a free and open-source computer Go program released on 25 October 2017. The original Leela is also a Go program built by a lot of human knowledge and heuristic information. As the release of the AlphaGo Zero [2] from DeepMind in 2017, new project called “Leela Zero” started according to the structure of AlphaGo Zero, which used deep learning to train a model by self-playing. It has been shown that the strength of Leela Zero has already exceeded the top-human players. In this research, Leela Zero will be called “Leela” for short. As the structure presented in AlphaGo Zero's paper, Leela uses one network combining policy network p and value network v together as the output (p, v).

KataGo is another Go program according to AlphaGo Zero's structure, but the original amount of computation in training is very large. KataGo introduced several new techniques to improve the efficiency of self-playing learning [17]. One of the major improvements in KataGo's training process over AlphaZero comes from adding auxiliary ownership and score prediction targets. They are many features in KataGo,

- Support territory and score estimation. Rather than only estimating "winrate", it can also helpe to analyse the territory and score prediction.

- Supports alternative values of komi (including integer values) and good high-handicap game play.

- Supports board sizes ranging from 7×7 to 19×19, and as of May 2020 may be the strongest open-source bot on both 9×9 and $13\times$ as well.

- Supports a JSON-based analysis engine that can batch multiple game evaluations efficiently and be easier to use than GTP. In this research, scripts writing by using analysis mode helps to evaluate all the moves in matches more efficiently.

In this research, these new features will be used for researches.

Chapter 3

Bad Move Detection

In this chapter, the introduction of “Bad Move Detection” before section 3.1 is modified and written from the article “Detection and labeling of bad moves for coaching go” which is published in the 2016 IEEE Conference on Computational Intelligence and Games (CIG).

There are many ways to improve skills and techniques in the Game of Go, such as replaying professional games, solving local life and death problems (tsumego), or reading books about common tactics and patterns. But it is often considered that one of the best way to improve is to play a game with a stronger player, and to review the game with him.

Coaching Go (Shido-Go in Japanese) is a special type of game, where amateur players pay some professional or semiprofessional player to play and review a game. There is a strong demand from amateur players for such games, but it can be expensive. Also, intermediate-level players are often reluctant to invest the money or the time in such coaching games, because they feel that their level is still too low for that. If a computer player could perform the same kind of coaching, it would be of great help for many amateur players, especially from beginner to intermediate level.

The authors of the paper [11] investigated the strong players of the Japanese Go club Teach intermediate players their bad moves. It usually follows the process shown in Figure 3.1, which is also borrowed from previous work of [11].

1. Detection: Bad moves are detected;
2. Reason: Give some reasons to label what is kind of the bad move;
3. Explanation: More detailed explanation is given as the result of the bad move;

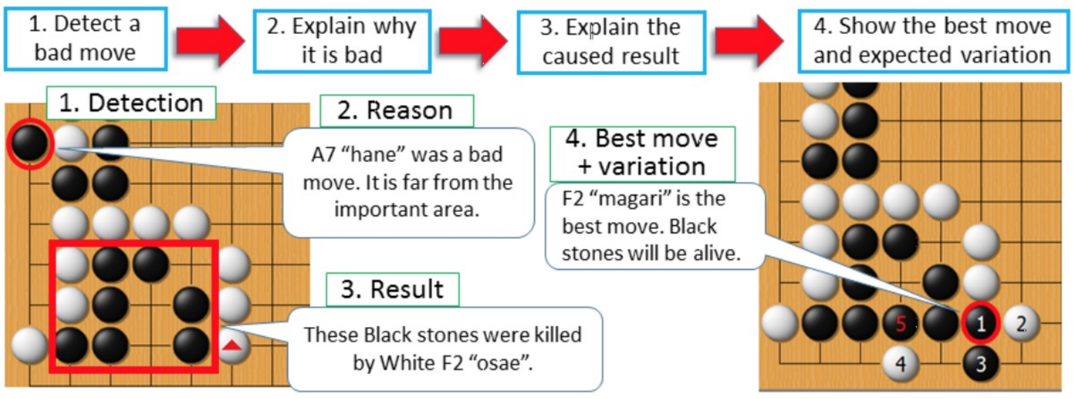
1. Detect a bad move
2. Explain why it is bad
3. Explain the caused result
4. Show the best move and expected variation
1. Detection
2. Reason
A7 "hane" was a bad move. It is far from the important area.
4. Best move + variation
F2 "magari" is the best move. Black stones will be alive.
3. Result
These Black stones were killed by White F2 "osae".
1
2
3
4
5

Figure 3.1: Complete process of helping players to correct their bad moves

4. Recommendation: Better move is shown with the best variation.

As the example in Figure 3.1. In step 1, The position "A7" will be detected as the bad move. In step2, it is far from the importance area marked by red rectangular will be shown as the reason. In step 3, "The black will be killed" will be the explanation for the bad move. In the last step 4, "black should play at "F2" to make alive would be the recommendation as a better move.

3.1 Related Works

As the researches in the area of Coaching board game is developing to be another interesting topic. In 2015, Kameko et al. used machine-learning to generate comments in natural language about Shogi positions [18]. Also in 2015, Ikeda et al. used machine-learning to learn the natural language names usually used by humans to refer to moves in the game of Go [19]. This is an important part for a coaching Go program, since moves are usually refered by shape names and not by coordinate positions in the game of Go. For example, in Figure 3.1, A7 is called "Hane".

In 2016, Ikeda et al. proposed approaches to train a classification model by using machine learning from data of played matches [11]. The details will be shown below:

1. Many handicap games are done. Game records are collected
2. Bad moves played by Black (weaker players) are selected by strong human players, with only 5 to 10 moves selected per game.

3. Many features are calculated by a computer program, for each move. We obtain a set of items (feature1, feature2, ..., bad/good, type) as features for training in step 4.

4. A supervised learning is executed by using all items where "bad/good" is the output. The result is the "Detection System".

The procedure of the approach is clear and easy to understand. But the important parts are: 1). how to definite "bad moves" in step 2 ; 2). how to select proper features for machine learning; 3). By using which kind of machine learning method.

For the problem 1). The definition of "bad move" is not trivial. If one set of best moves can be defined and calculated, though theoretically it is possible, it may be possible to say that the other moves are all bad moves. However, for coaching intermediate players, usually *fairly* bad moves are pointed out because such players will be confused or depressed if too many moves are corrected.

For the problem 2). After having made the standard of selecting a "bad move", it is natural to think out of "winning rate" is connected closely with "bad moves", other than such kinds of features, some features can also be calculated by Nomitan. The totally used features in the previous work are explained in Appendix A. The finally used in the machine learning are 9 features: **wrdiff**, **wrbefore**, **wrafter**, **shapelog**, **trdiff**, **trbefore**, **trafter**, **move**, **owndiff**.

For the problem 3). As the good/bad prediction can be viewed as classification problem. In the previous work, MP(Multilayer Perceptron) was used for the binary-classification problem.

In the previous work, totally 3963 good move instances and 873 bad move instances are collected from game records by human players. Since unbalance among the numbers of instances is not preferable in classification, 2000 good move instances are randomly removed in this experiment. Finally, good moves (1963) and bad moves (873) were used to train by Multilayer Perceptron (MP) with 10-folding self validation.

The learning result showed that the achieved F-measures were 0.826/0.444/0.709 for good moves/bad moves/average.

Chapter 4

Playstyle Production

In the area of education and entertainment Go, not only human players can enjoy playing against with strong Go players by coaching Go, but also some playstyle playing can help to get lots of fun in the game of Go.

In the area of coaching Go using strong Go program, position control is used for proper move selection, which tries to play evenly and smoothly with amateur human players [12].

In the area of various playstyle production, if human players can play against some Go programs with specific playstyle. There are some clasical playstyles in real world, such as offensive/defensive, optimistic/pessimistic, it will also be a seed of entertainment. So it is valuable to produce various playstyles in computer Go programs.

4.1 Related Works

In 2011, Takise and Tanaka proposed a method by adding some specific features to the Shogi program and learned from a specially selected part of game records again, this method successfully produced a specific playstyle without decreasing winning rate [10].

In 2013, Ikeda introduced approaches in Monte-carlo Go program for producing various playstyles [12], and three methods were proposed for various playstyles production.

In this research, we only focus on production of center-oriented and edge/corner-oriented playing, we will introduce the previous approach simply about how to produce kinds of playstyles.

4.1.1 previous approaches: edge/corner-oriented versus center-oriented Play

In the previous research, Nomitan, which is referred in section 2.4, a Mote-Carlo Go Program, is used as the experiment tools for implementing their approach.

Usually, count 1 point for each open intersection is used for calculation after the simulation has reached the final position,some other weights were used, such as, counting intersection in the corner 1.5 point, while 0.5 for the intersections in the center. This change will affect the calculation result of territories for two players. The details of the method are given below:

1. Set the parameter α that indicates the relative importance of the center, and the parameter n_{max} that limits the influence when the game advances.
2. At the position of move $n (n < n_{max})$, use the following weights when counting the territory result of a simulation:
   (a) $1 - \alpha \cdot (1 - \frac{n}{n_{max}})$ from the first line to the third line of the board (usually considered the main place for territory in the game of Go);
   (b) 1 for the fourth line;
   (c) $1 + \alpha \cdot (1 - \frac{n}{n_{max}})$ for lines above the fifth one (territory in the center).

By doing this changes for weight calculation in different intersections on the board, it is possible that what should have been counted as a loss will be in fact be counted as a win.

As the result in simulation in computer Go will directly affects the selection for root node in MCTS. For example, if we set α to be a negative number, which will prefer the intersection close to the edge/corner to have more point(greater than normal 1), while intersections close to the center will have a weaker point less than 1. The calculation will guide the trend to play moves more nearer to the edge/corner.

Figure 4.1 is an example in the previous research. Black uses $\alpha = +0.2$ (center-oriented), white uses $\alpha = -0.2$ (territory-oriented), and $n_{max} = 80$. We can see clearly that black prefers the center area while white prefers the corners and the edges.

The previous approach also showed that the new Go programs with edge/-corner and center-oriented playstyles are not so weaker than the original program, Nomitan, for playing gently against intermediate level human players with a winning rate about 56%.

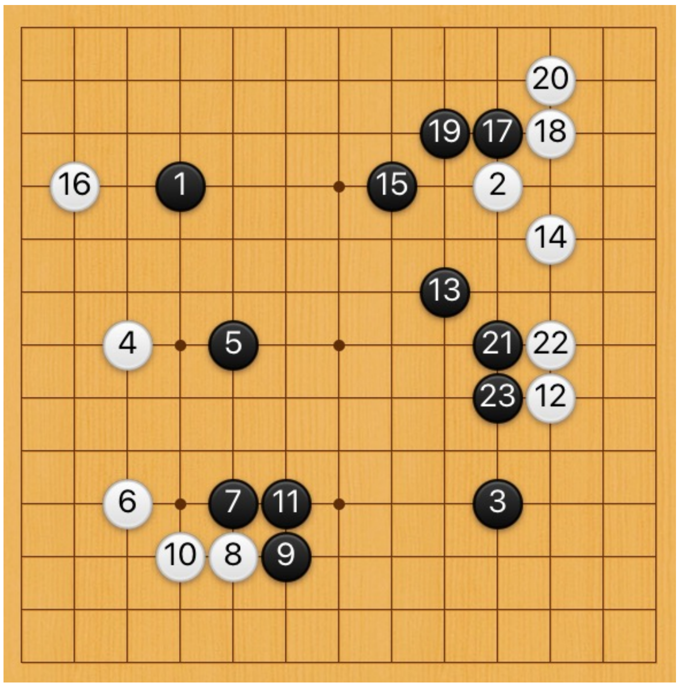

Figure 4.1: Example of real game between territorial settings (white) vs influence settings (black).

Human subjects also clearly declared that the edge/corner and center-oriented playstyle productions can be felt by human players [12].

In the real Go playing, some common playstyles are:

1. center-oriented vs edge/corner-oriented playing
2. aggressive vs defensive playing
3. pessimistic vs optimistic playing

For playstyle 1, it is often used in primary stage (fuseki) in section 2.1.3. As Figure 2.4 shows that, after moves from 1 to 6, the white move tends to play towards the center, we can judge the white a center-oriented playstyle, while if the while move 6 changes to be position A, which has more interest in palying near to the edge/corner for territory capture.

For playstyles 2, aggressive players tend to fight with opponents to gain profit, while defensive players fear to fight, who choose to defense firmly for territories.

For playstyles 3, as the necessary condition for winner is to get more territories than the opponent. Intermediate players will estimate their territories while playing. Pessimistic players tend to estimate less than the real territories while optimistic players will take the idea that they are having much more territories than actually captured. In such cases, pessimistic players will tend to play moves defensively while optimistic players prefer some moves more offensively.

Chapter 5

Approaches

In this chapter, section 5.1 and 5.3 are written and modied based on the article "Position Control and Production of Various Strategies for Deep Learning Go Programs" which is published in the 2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI 2019) in Taiwan in 2019.

In this Chapter, firstly differences between Go programs based on AlphaGo Zero model and traditional MCTS Go programs. These differences may positively/negatively affect the quality of bad move detection or playstyle production. In fact, better performance of bad move detection is reported in section 5.2. Also, since there are some known problems in playstyle production, we propose a new method in section 5.3.

5.1 Differences in The New Programs

The existing methods of bad move detection [11] and producing various playstyles [12] were proposed for traditional MCTS programs. Recent programs based on AlphaGo Zero are much stronger than traditional ones, and have different mechanisms. Then, some of the existing methods cannot be directly applied, or may produce bad performance. In this section, we summarize the differences and several expected or known problems. In this paper, we employ two open-sourced programs, Leela [4] as a recent program, and Ray [6] as a traditional program, and 13×13 board is used for experiments.

The main difference lies in the state evaluation mechanism. To evaluate a leaf node, traditional MCTS programs run some (biased) random simulations to the ends of games. Win or loss is judged by the rules of Go. On the contrary, recent programs use value networks to evaluate leaf nodes. As a result, our previous method for producing various playstyles (described

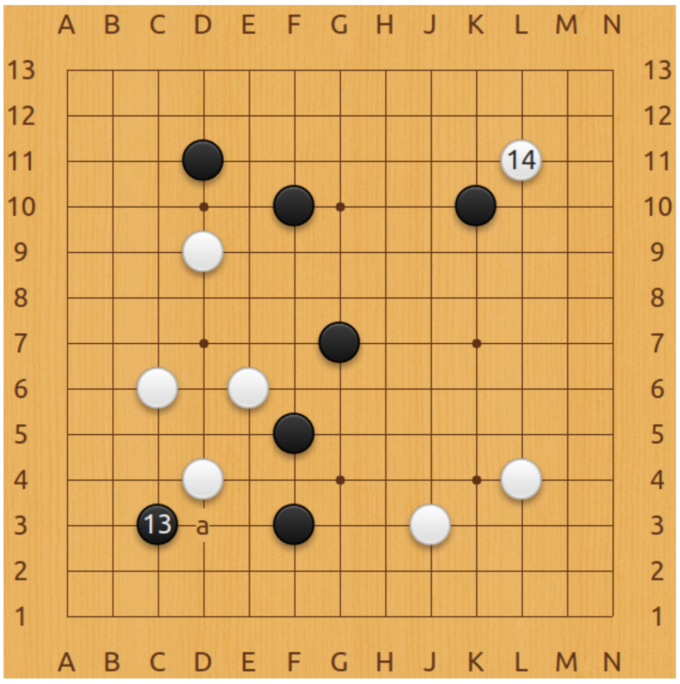

Figure 5.1: A move far from the opponent move.

in Section 4) cannot be used, because the trick modifies the definition of wins/losses at the ends of games.

Another difference is whether human game records are used. Recent programs do not involve human game records, then their trained policy networks are sometimes different from human players' senses. Especially, recent programs tend to play far from the opponent moves more frequently. For example, when Leela and Ray play against each other, the average Euclidean distance between a Leela's move and the last Ray's move (100 games, from the second to the 60th moves) is 3.16 ± 0.10. This is significantly bigger than that between Ray's moves and the last Leela's moves, 2.65 ± 0.08. Usually it seems natural to react directly when the opponent tries to attack or invade. This tendency is strong especially for beginners. So, playing far from the last move is sometimes risky for entertainment, because beginners may think "my move was ignored, it's a strange play". Figure 5.1 is an example. Black played C3, which aims to invade the territories of the White corner. For beginners, it is natural to play at D3 (A). However, here Leela selected L11 (1). In fact, this move is not bad, but may seem to be strange from beginners' viewpoints.

5.2 Bad Move Detection

In this research, we try to reproduce the bad move detection system by using data analysed from KataGo [5].

In the previous research [11], good/bad moves were labelled by strong human players, then 9 features were calculated by using Nomitan, a Monte-Carlo Go program.

In this research, we use KataGo to analyse the candidate 29 features in Appendix A of the previous labelled data again, and try to show how feature combination affect the final training result.

5.2.1 Data Preparation

In the previous work, all the data used for machine learning are evaluated by Nomitan during the game playings or after the end of games, while when we want to evaluate the data again by KataGo, we need to use the "analysis mode" to evaluate all the moves in game records. It should be noticed that the game records are the same as what in the previous research [11].

The basic information of the game records are referred in [11]:

- 8 intermediate-level human players (from about 7k to 1d) were asked to play against Nomitan;
- handicapped games (2 to 4 stones) were played on 13×13 board, totally 108 games were collected;
- 3 strong human players (about 4d to 7d on KGS) were asked to select bad **BLACK** moves;
- 5 to 10 bad moves are selected for one game.

Notice that in this research, these same game records are used without any modification.

The next step is to use KataGo to analyse useful information and calculate features.

Analysis Mode in KataGo and feature calculation

KataGo contains an engine that can be used to analyze large numbers of positions in parallel (entire games, or multiple games). By using this mode, it is easy to make tools for game records analysis.

In analysis mode, the kataGo engine accepts queries written in JSON on stdin and outputs analysed result on stdout. One query example is:

```
{
    "id": "Kat01_cho_0323_1.sgf",
    "initialStones": [
        ["B", "k10"],
        ["B", "d10"],
        ["B", "d4"],
        ["B", "k4"]],
    "moves": [
        ["W", "f11"],
        ["B", "c11"]],
    "rules": "tromp-taylor",
    "komi": 0.5,
    "boardXSize": 13,
    "boardYSize": 13,
    "maxVisits": 6000,
    "includeOwnership": true,
    "analyzeTurns": [1]
}
```

In each field of one JSON string,

- id, identification string that was provided on the query.
- initialStones, moves that have already existed on the board before start of one game. As our games are almost handicapped games, handicapped stones can be written here.
- moves, moves played in order.
- rules, the "tromp-taylor" a common rule used in Computer Go such as "AlphaGo Zero" [2].
- komi, set to 0.5 for handicapped game.
- boardXSize and boardYSize, as the original games are played on 13×13 board.
- includeOwnerShip, ownership prediction will be included in the result.
- analyzeTurns, which turns of the game moves to analyze, for the example above, the "analyzeTurns:[1]" means the board state after playing the first move "f11" of white will be evaluated by KataGo.

Notice that “includeOwnerShip” is a new feature of KataGo. For a 13×13 board, ownership of one intersection is a number from -1 to 1. If we set that the analysis in KataGo insights from **BLACK**’s (default) viewpoint. If the number of one intersection is nearer to 1, it means the intersection has a bigger probability of been occupied by black, and if is nearer to -1, the intersection has a bigger probability of been occupied by white.

Respectively, the corresponding analysis result will be:

```
{
        "id":"Kat01_cho_0323_1.sgf",
        "moveInfos":[
            {
                "lcb":0.998273635765558,
                "move":"E9",
                "order":0,
                "prior":0.150713190436363 22,
                "scoreLead":48.04539764154197,
                "scoreMean":48.04539764154197,
                "scoreSelfplay":48.04539764154197,
                "scoreStdev":23.17215707835482,
                "utility":1.101951159742821,
                "utilityLcb":1.1047525838698247,
                "visits":2661,
                "winrate":0.997273127148771
            },
            ...
           ],
        "ownership":[
            0.584422664140057,
            0.5599006744552762,
            0.40138626156590357,
            ...],
        ...
        "turnNumber":1
}
```

The mainly used fields are:

- winrate, The winrate of the candidate moves, as a float in [0,1].
- scoreLead, The predicted average number of points that the current side is leading by before searching. It is given to each candidate moves.

- scoreStdev, The predicted standard deviation of the final score of the game after searching.
- prior, The selection probability before searching.
- ownership, The predicted ownership of all the intersections on the board. This field is independent of “moveInfos”, which means it just has relationship with one specific board state, has no relationship with the selected moves.

As all the 29 features in Appendix A can be calculated by these variables. The next step is to use these features to build a proper model by some machine learning methods.

Feature Selection and Machine Learning

It is a binary classification problem for detecting good/bad for one move. Since there are many methods for binary classification, we employ Multilayer Perceptron (ML), which is the same method applied in the previous work, in a free machine learning platform, Weka version 3.8.4 [20].

The same method (ML) using can help to show how features calculated by two different program Go (Nomitan and KataGo) affect the final model of ML.

5.3 Playstyle Production

In this research, we try to reproduce the edge/corner and center-oriented playstyles by using Leela [4].

In the previous research [12], the production of edge/corner and center-oriented playstyles are implemented by changing the definition of wins/losses for random simulations of MCTS, which needs to play games to the ends. However, since random simulations to the ends of games are not done in programs such as Leela, the same method cannot be applied. So, we propose to train new networks to produce various playstyles in an offline manner, while the previous method is done in an online manner.

Since Leela open-sourced the codes [4], we can customize the training of new Leela networks, e.g., with our own definition of wins/losses, or in different board sizes. Usually, when training value networks, the pairs (game state, result) are saved as training data. The result is determined by the rules of Go, by comparing the territories of Black and White (+Komi).

In this research, we tried to make a center-oriented network, by giving higher weights to center territories:

- From the first line to the third line (i.e., corner or edge), the weight is $1 - \beta$.
- On line 4, the weight is 1.
- Starting from line 5 (i.e., center), the weight is $1 + \beta$.

When β is positive, center-oriented network are trained. With negative β, which gives lower weights to center territories, edge/corner-oriented networks are trained. Note that if such networks are used solely, they may not play well for standard games of Go.

Simply by modifying the definition of wins/losses, networks with different preference can be trained, such as optimistic, pessimistic, offensive or defensive in the previous research [12]. For example, if one captured stone is counted as -2 points, defensive networks may be trained. We remain such experiments as future work.

5.3.1 Pure Network Production

In section 5.3, a new approach for producing various playstyles was proposed. For each playstyle or preference, some period of self-training is needed. We trained center-oriented Leela network (Net_{center}) and edge/corner-oriented Leela network (Net_{edge}) about two weeks from scratch on GeForce GTX TITAN X. The parameter β was set to $+0.2$ for Net_{center}, and -0.2 for Net_{edge} respectively.

Net_{center} and Net_{edge} were trained not for playing solely. Both won no game against the standard Leela in 60 games for checking their behaviors. Figure 5.2 shows a game between Net_{center} (Black) and the standard Leela (White). Moves 1, 9, and 11 are very strange from the viewpoint of the standard rules of Go, but does fit the goal of Net_{center}. Figure 5.3 shows a game between Net_{edge} (Black) and the standard Leela (White). Moves 13 and 15 are far from hot area, and finally the left-bottom black stones are in danger. The results confirmed that Net_{center} and Net_{edge} had clear preference to the center and the edge/corner territories respectively.

5.3.2 Mixed Approach

Both center- and edge/corner-oriented network had clear preference, but too weak when used solely. In this section, we introduce a way to combine the trained network, Net_{center} or Net_{edge}, with the original Leela, in order to balance the strength and strategy preference. The decision making procedure is as follows:

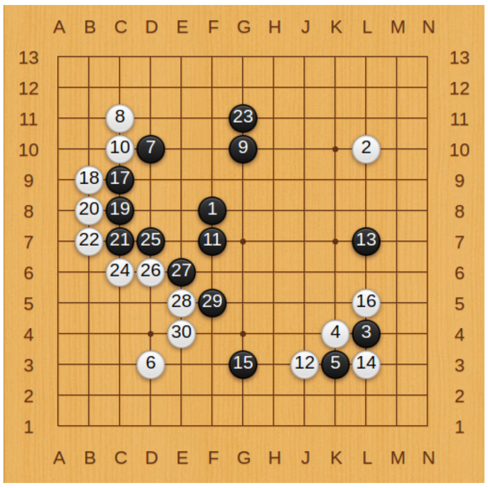
Figure 5.2: Net_{center} (Black) vs standard Leela

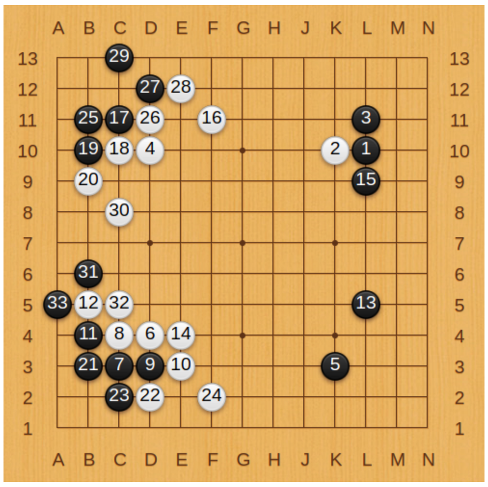
Figure 5.3: Net_{edge} (Black) vs standard Leela

- *Search*. The current board status is given to both networks (the original Leela and the biased one), and two lists of candidate moves, M_o and M_b, are obtained. Let $w_o(m)$ be the winning ratio of move m from the original Leela, and $p_b(m)$ be selection probability of the biased Leela .
- *Mix*. From the move list M_b, some moves are removed: (1) when its winning ratio $w_o(m)$ is less than $\max_i\{w_o(i)\} - param_{gap}$, or (2) when its selection probability $p_b(m)$ is less than $param_{sp}$. After that, if M_b is not empty, the most visited move in M_b is selected. Otherwise, the most visited move in M_o is selected.

This mix procedure means that when and only when there are several acceptable moves from the viewpoint of the original Leela, the best one from the viewpoint of the biased (center- or edge/corner-oriented) Leela is selected. $param_{gap}$ was set to 0.05, and $param_{sp}$ to 0.1. We call the mixed players $Leela_{center}$ and $Leela_{edge}$ respectively. Figure 5.4 shows a game between $Leela_{center}$ (Black) and the original Leela (White). Figure 5.5 shows a game between $Leela_{edge}$ (Black) and the original Leela (White). While there are still several unexpected moves, we can recognize the preference.

We conducted experiments to evaluate the strength of the mixed versions. Each of $Leela_{center}$ and $Leela_{edge}$ played 300 games against the original Leela. The number of wins were 135 and 129 respectively. We can say that the mixed versions were not so weak, because they only selected acceptable moves from the viewpoint of the original Leela.

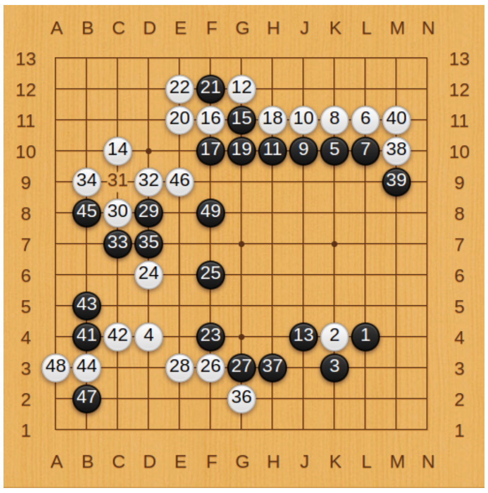
Figure 5.4: *Leela*$_{center}$ (Black) vs standard Leela

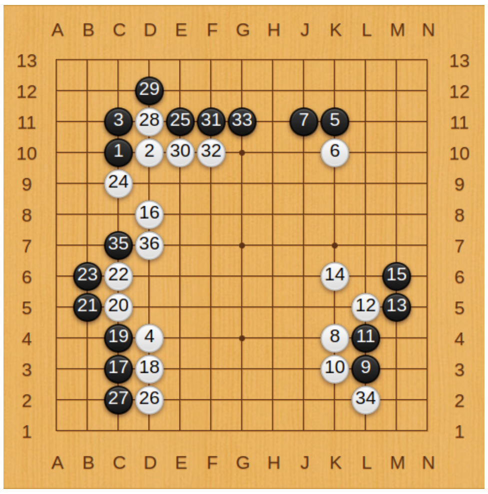
Figure 5.5: *Leela*$_{edge}$ (Black) vs standard Leela

Chapter 6

Experiments

In this Chapter, the experimental results of bad move detection (described in section 5.2) and playstyle production (described in section 5.3) are shown.

6.1 Bad Move Detection

In this section, bad move detection performance is evaluated. The training/test data is the same, used in the previous research [11], 873 bad moves and 1963 good moves are used. Also, the employed supervised learning method is the same, Multilayer Preceptron (ML) is used for machine learning. Only the difference is the employed feature values. All values are recalculated by KataGo, a new program Go based on AlphaGo Zero model, also some new features are tried.

For bad move detection, territory score is very important for human players. These features were also used in the previous work, but the estimation was sometimes inaccurate. So, by using very strong computer program, these features became the core features.

The experiment result is shown on below by table 6.2, in the table, we try to compare F-measure values for different feature selections and make comparisons.

As the table 6.2 shows, when using the same features in the previous work,

- The total F-measure of Good move in the previous work is better than the new approach;
- The features of **trdiff**, **trbefor** and **trafter** help to build a better model for Bad move detection by upgrading F-measure of average 0.026 comparing to the previous work.

Table 6.1: Good/Bad detection results. Used features in the previous work and new approach. F-measures and Comparisons

features	F-measures (previous)	F-measures (new)	gain (new)
wrdiff only	0.812/0.299/0.654	0.809/0.195/0.621	-
+wrbefore, wrafter	0.815/0.361/0.675	0.812/0.251/0.640	0.019
+shaperate	0.814/0.326/0.664	0.811/0.241/0.637	0.016
+shapelog	0.812/0.357/0.672	0.831/0.359/0.687	0.066
+trdiff	0.809/0.381/0.677	0.798/0.492/0.704	0.083
+trbefore, trafter	0.817/0.389/0.685	0.800/0.509/0.711	0.090
+handi	0.810/0.333/0.663	0.809/0.192/0.620	-0.001
+move	0.812/0.378/0.678	0.813/0.189/0.622	0.001
+dist1b	0.813/0.322/0.662	-	-
+owndiff	0.812/0.330/0.664	0.805/0.215/0.624	0.003
+own2diff (new)	-	0.813/0.242/0.639	0.018
+own2before, own2after (new)	-	0.808/0.282/0.647	0.026
+8 features	0.826/0.444/0.709	0.821/0.521/0.729	0.108
+feature group A	-	0.822/0.536/0.735	0.114
+feature group B	-	0.817/0.535/0.731	0.110

- The F-measure when using adding 8 features by new approach is better than the previous work about 0.020.

Furthermore, we added one group of new features, **own2before**, **own2after** and **own2diff** to the original feature group. When just adding **own2diff** to **wrdiff**, the F-measure advanced by 0.018. When adding **own2before** and **own2after** to the **wrdiff**, the F-measure advanced by 0.026. So we try to add these three features to the original feature group and evaluate some new feature groups.

- feature group A: **wrbefore**, **wrafter**, **shapelog**, **trdiff**, **trbefore**, **trafter**, **move**, **owndiff**, **own2diff**.

- feature group B: **wrbefore**, **wrafter**, **shapelog**, **trdiff**, **trbefore**, **trafter**, **move**, **owndiff**, **ownbefore**, **ownafter**, **own2diff**, **own2before**, **own2after**.

Still as showed in table 6.2, the totally F-measure is upgrading to 0.736 when employ feature group A, while feature group B is a little small than previous feature group.

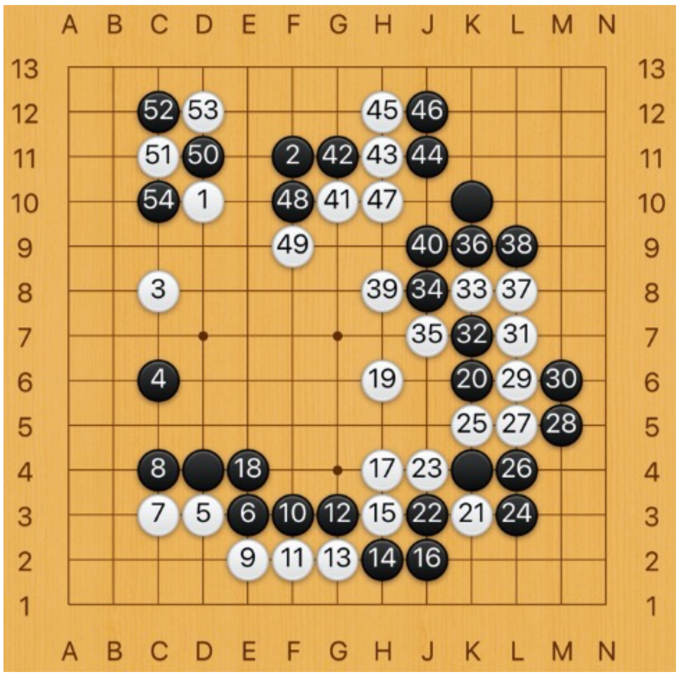

Figure 6.1: Game for bad move detection, up to 54th moves.

From the result, we can claim that one Deep Learning Go Program helps to build a better model of Bad Move Detection with a advance of 0.026 F-measure.

The new approach by using KataGo to build bad move detection advanced and performed better than the previous work by using Monte-Carlo program [11].

6.1.1 An Example of Bad Move Detection

We will perform the new approach to predict bad moves in the same game record used in previous work [11].

The Figure 6.1 shows the 1st to 54th moves of one game record, evaluated in both previous work and new approach.

The prediction result of previous work and new approach is shown in Table 6.2.

It is claimed from professional players that,

- 24th. This move is not good, but not detected by the previous work, now new approach can detect it.

Table 6.2: Detection of Bad Move in previous approach and new approach

move number	previous approach	new approach
8	detected	detected
14	detected	detected
16	not detected	detected
18	detected	detected
20	not detected	detected
24	not detected	detected
30	detected	not detected
32	detected	not detected
38	detected	not detected
46	detected	detected
48	not detected	detected
54	not detected	not detected

- 30th. This move is detected by previous work, but actually not so bad, it is not detected by new approach.
- 38th. This move is detected by previous work, but actually not so bad, it is not detected by new approach.

At the same time,

- 54th is not detected by both previous and new approach. Actually it is bad.
- 16th, 20th and 48th moves are detected by the new approach.

About 16th move, the shaperate is 0.000866, which is very low then detected by the system. But actually, this can not be said to be a "bad move". About 20th and 48th move, the trdiff is -3.5 and -5.5 respectively, which is very low, then detected. But actually, the best move is difficult to select, the best move may be unexpected move from intermediate player's side. Maybe they will not be selected by human coaches as bad moves.

The result shows that the detection by new approach can detect some bad moves while not detected by previous work. While some moves are newly detected by new approach.

6.2 Playstyle Production

We mentioned that mixed Leela seems to have a playstyle or preference, and not so weak. Finally, we conducted an experiment using human subjects, to

Table 6.3: Evaluation result of center, edge/corner, and standard Leela

Actual Evaluated	a	b	c	d	e	Total	Average
$Leela_{center}$	36	19	20	3	2	80	-1.05
$Leela_{edge}$	3	4	13	14	46	80	1.20
Original	37	22	48	25	28	160	-0.09

evaluate how the playstyle can be identified from the viewpoints of human players.

We tested three versions of Leela, (A) the original Leela, (B) $Leela_{center}$, and (C) $Leela_{edge}$. Four groups of game records were generated, A vs A, A vs B, A vs C, and B vs C. In each group, five game were played. Totally, 20 game records were given to eight human subjects (ranks 6k to 8d) in random order and in a blind manner. Each person was given one hour to review the records.

Human subjects were asked to judge which preference can be identified for Black player and White player respectively in each game. Options were (a) center-oriented, (b) slightly center-oriented, (c) nothing, (d) slightly edge/corner-oriented, and (e) edge/corner-oriented. For averaging the results, we assigned -2 to (a), -1 to (b), 0 to (c), $+1$ to (d), and $+2$ to (e). The number of answers and average values are listed in Table 6.3.

We can claim that both center- and edge/corner-oriented playstyles could be produced and recognized with a high probability. In 115/160 answers, the strategies of programs were correctly judged, and there were only 12 opposite answers. Average scores were also close to expected, and indeed different from each other. It is interesting that when the original Leela played against center- (or edge/corner-) oriented Leela, the original Leela tended to be judged as edge/corner- (or center-) oriented.

In this paper, the training and experiments were done on 13×13 board. The early stage is shorter than 19×19 board case, and thus human players had less chances to identify strategies. So, we expected it to be easier to produce various playstyles in 19×19 board case, while the required training cost will be much higher.

Chapter 7

Conclusion and Future Work

In this research, we used Deep Learning Go model, Leela and KataGo to evaluate Bad Move Detection again and production of center- and edge/corner-oriented playstyles.

In the Bad Move detection, the result shows that it can provide a much better model from the data analysed from KataGo. The F-measure of Good/Bad move increased from 0.709 to 0.735. One example to compare detection by previous and new approach show that new approach can detect moves not detected by previous approach, while some moves are predicted bad newly. If explanation and recommendation are available, maybe proper explanations for newly detected moves are more convincing. A complete coaching Go system based on Bad Move Detection not only contains Bad Move Detection, but also other 3 steps should be taken researches.

In the Production of Various Playstyles. By self-training with using a modified definition of wins/losses, it is possible to obtain center-oriented or edge/corner-oriented network. Experiments on human subjects successfully showed that the mixture of such biased networks and the original networks was effective to produce specific playstyles while keeping the strength. Some promising future researches include (1) training on 19×19 board and comparing to other existing work, (2) implementing optimistic/pessimistic/offensive/defensive strategies, and (3) solving remaining problems about naturalness.

Bibliography

[1] Silver, David, et al. ”Mastering the game of Go with deep neural networks and tree search.” nature 529.7587 (2016): 484-489.

[2] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of go without human knowledge[J]. nature, 2017, 550(7676): 354-359.

[3] Wedd, Nick. ”Human-Computer Go Challenges”. computer-go.info. Retrieved 2011-10-28.

[4] “LeelaZero,” https://github.com/leela-zero/leela-zero, last accessed: 2019-02-10.

[5] ”KataGo,” https://github.com/lightvector/KataGo, last accessed: 2020-01-18

[6] “Ray,” http://computer-go-ray.com, last accessed: 2019-02-10.

[7] Browne, Cameron B., et al. ”A survey of monte carlo tree search methods.” IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.

[8] Coulom, Rémi. ”Computing “ELO ratings” of move patterns in the game of Go.” ICGA journal 30.4 (2007): 198-208.

[9] Browne, Cameron B., et al. ”A survey of monte carlo tree search methods.” IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.

[10] Ryuji Takise and Tetsuro Tanaka, “Development of entering-king oriented shogi programs”, 16th Game Programming Workshop, pp. 25-31 (2011)

[11] Ikeda K, Viennot S, Sato N. Detection and labeling of bad moves for coaching go[C]//2016 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2016: 1-8.

[12] Ikeda, Kokolo, and Simon Viennot. "Production of various strategies and position control for Monte-Carlo Go—Entertaining human players." 2013 IEEE Conference on Computational Intelligence in Games (CIG). IEEE, 2013.against

[13] Gelly S, Silver D. Achieving master level play in 9 x 9 computer go[C]//AAAI. 2008, 8: 1537-1540.

[14] Olivier Teytaud Adrien Couetoux, Jean-Baptiste Hoock, Nataliya Sokolovska and Nicolas Bonnard. Continuous Upper Confidence Trees. LION 5, page 433–445, 2011.

[15] Adrien Couetoux David Auger and Olivier Teytaud. Bandit based monte-carlo planning. ECML 2006, page 282–293, 2006.

[16] Gelly S, Wang Y. Exploration exploitation in go: UCT for Monte-Carlo go[C]. 2006.

[17] Wu D J. Accelerating self-play learning in Go[J]. arXiv preprint arXiv:1902.10565, 2019.

[18] Hirotaka Kameko, Shinsuke Mori and Yoshimasa Tsuruoka, "Learning a Game Commentary Generator with Grounded Move Expressions". Proceedings of the 2015 IEEE Conference on Com- putational Intelligence and Games (CIG), pp. 177-184 (2015)

[19] Kokolo Ikeda, Simon Viennot and Takanari Shishido, "Machine Learning of Shape Names for the Game of Go", 14th International Conference Advances in Computer Games, (2015)

[20] J. R. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993

[21] A. Couetoux, M. Müller and O. Teytaud, "Monte Carlo Tree Search in Go", from https://webdocs.cs.ualberta.ca/-mmueller/ps/2013/2013-gochapter-preprint.pdf, 2017.

[22] Go wikipedia: https://en.wikipedia.org/wiki/Go_(game)

[23] British Go Association: http://britgo.org/

[24] Sensei's Library: https://senseis.xmp.net/

Appendix A

Features Explanation

All the 29 features used in chapter 3 are explained here. They all can be calculated by analysed results from KataGo.

- **handi**, the number of handicap stones.
- **move**, the number of moves played.
- **wrbefore**, **wrafter**, **wrdiff**, expected winning ratio before the move, after the move, and its difference.
- **trbefore**, **trafter**, **trdiff**, expected territory advantage before the move, after the move, and its difference.
- **shaperate**, **shapelog**, shape goodness calculated from policy network, the prior selection probability and absolute log value.
- **dist1b**, Euclidean Distance between the actual Black move and the estimated best move.
- **ownbefore**, **ownafter**, **owndiff**, ownership of the position before the move, after the move, and its difference. High ownership means that the area is occupied by Black, i.e. the Black stones in the area are strong, or the White stones in the area are weak.
- **trstdbefore**, **trstdafter**, **trstddiff**, standard deviation of territory advantages, before/after the move and its difference. They are calculated with trbefore, trafter, trdiff, and representing how unclear the game result is.
- **dist01**, **dist02**, **dist21**, **dist0b**, **dist2b**, Euclidean distances between two of the last White move (0), the next White move (2), the actual Black move (1), and the estimated best move (b) .

- **own2before**, **own2after**, **own2diff**, averaged ownership of Black stones on 3×3 area neighboring the Black move. Values before/after the move, and its difference.
- **bdecav**, **wdecav**, average ownership decreasements of all Black/White stones, by the next White move. When bdecav is high, it means Black stones are weaken by the next White move, because of losing chance to defend.
- **bdec30**, **wdec30**, the number of Black/White stones which their ownerships are decreased by 0.3, by the next White move.

Appendix B

Bradley-Terry Model

The Bradley-Terry Model are usually applied in paired comparison. For a team consists of some individuals. γ_i stands for the strength of individual i in the team with n elements. The competition result between two individuals i and j can be predicted by the following probability:

$$P(i\ wins\ against\ j) \ = \ \frac{\gamma_i}{\gamma_i + \gamma_j}. \tag{B.1}$$

If we want to predict individual i wins in one team, the competition result can be presented as $P(i\ wins) \ = \ \frac{\gamma_i}{\sum_1^n \gamma_k}$.

In the most general case, we consider a competition between terms of individuals. The strength of one team is the product of all the team members. For example, the strength of team 1-2, which consists of member individual 1 and 2, is presented as $\gamma_1\gamma_2$. In this case, the prediction of teams 1-2 wins against team 3-7-9 and team 4-8 can be presented as:

$$P(1, 2\ wins) \ = \ \frac{\gamma_1\gamma_2}{\gamma_1\gamma_2 + \gamma_3\gamma_7\gamma_9 + \gamma_4\gamma_8} \tag{B.2}$$

Bradley-Terry Model in Go Programs

When applying Bradley-Terry Model in Go Programs, Bradley-Terry model can help determine possible moves according to one board position. For one specific board state, many features such as $\gamma_(distance\ to\ previous\ move)$, $\gamma_(distance\ to\ boarder)$, $\gamma_(distance\ to\ move\ before\ previous\ move)$ can be viewed as individuals in one team. So Each legal move can be considered as one team competing with other moves.

The selection probability P_{m_j} of one legal move j can be then presented as a team of features composition wins against all the teams. Which can be presented as:

$$P(m_j) \; = \; \frac{\prod_{features\ i \in m_j} \gamma_i}{\sum_{legal\ move\ m} \prod_{feature\ i\ \in m} \gamma_i} \tag{B.3}$$

From data of strong professional players' matches, the γ can be learned by maximizing the likelihood estimation $L(\gamma_i) \; = \; \prod_{j=1}^{n} P(m_j)$.

Therefore, moves calculated by various features learned in Bradley-Terry model can help MCTS more precisely and efficiently.