Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Application of Game Theoretic Algorithms to Gomoku

Muzika Martin

Supervisor: Ing. Jiří Čermák
Field of study: Open Informatics
Subfield: Computer and Information Science
May 2017

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Martin M u z i k a
Study programme:	Open Informatics
Specialisation:	Computer and Information Science
Title of Bachelor Project:	Application of Game Theoretic Algorithms to Gomoku

Guidelines:

The student will learn the state-of-the-art approximation algorithm for solving large perfect information games - Monte-Carlo tree search and devise its modification suitable for the game Gomoku. He will further enhance this algorithm using heuristic learned by machine learning methods on data containing matches of human players. This algorithm will be used to create the first competitive player of the game Gomoku.

Bibliography/Sources:

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

[2] Levente Kocsis, Csaba Szepesváry and Jan Willemson - Improved Monte-Carlo Search – Univ. Tartu, (2006).

[3] Viliam Lisy, et al. - Convergence of monte carlo tree search in simultaneous move games – Advances in Neural Information Processing Systems. (2013)

Bachelor Project Supervisor: Ing. Jiří Čermák

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, December 15, 2016

Acknowledgements

Thank your for everything that I have learned during my studies. I will keep my memories safe and I will do my best in live to convince the school to not be ashamed of me.
Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures"

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses. Prague, date 24. May 2017

Abstract

In this thesis, we focus on solving the game *Gomoku-swap2*. The challenges of this domain are the size of the branching factor and the depth of the game. For this reason, the game cannot be solved exactly and so the evaluation function estimating the quality of a game state is the most important factor. Because it is impossible to solve the game exactly, we use Monte Carlo Tree Search (MCTS) where we use large number of simulations as an evaluation function. To guide the search in the MCTS we use the Neural Network learned on the human-played games. In experimental evaluation, we show that the Neural Network heuristic significantly enhances the performance of play. This leads to the first algorithm capable of human-like performance in the game *Gomoku-swap2*.

Keywords: MCTS, Gomoku, Gomoku-swap2, Neural Network

Supervisor: Ing. Jiří Čermák

Abstrakt

V této práci se zaměřujeme na řešení hry *Gomoku-swap2*. Problémem této hry je obrovský množství možných akcí na daný tah a poměrně velká hloubka hry.Z toho důvodu ohodnocovací funkce pro ohodnocení úrovně herního stavu je nejdůležitější část programu hry. Z důvodů velkého prohledávacího prostoru jsme použili algoritmus Monte Carlo Tree search (MCTS). Pro usměrňování prohledávání jsme použili Neuronovu Síť naučenou na hrách, hraných hráči, místo random heuristiky. V experimentech jsme prokázali lepší výsledky MCTS s neuronovou sítí než základní MCTS. Toto řešení vedlo k prvnímu algoritmu schopnému hrát hru Gomoku-swap2.

Klíčová slova: MCTS, Gomoku, Gomoku-swap2, Neural Network

Contents

Figures

Tables

Chapter 1
Introduction

In this thesis, we focus on solving the game *Gomoku-swap2* which is the modification of old strategy game Gomoku (tic-tac-toe) for two players [8][7].

The *Gomoku-swap2* is a perfect information game where the action to be played can be found in every state independently. The main challenge of this game is the size of branching factor and depth of the game (the depth is 225 and branching factor is $225 - x$ where the x is the depth of a state in the game tree). For this reason appropriate algorithm need to be used because the evaluation function for estimates the quality of a game state is the most important factor.

There are many recent advancements in the basic Gomoku using the Alpha-Beta pruning and Monte Carlo Tree Search (MCTS) with domain specific heuristics [9][6]. The main disadvantage of these methods is slow performance (approximately few minutes to choose the right action for every move). Therefore these approaches cannot be used in the game *Gomoku-swap2* where the search space is large and strict rules make the game complex. In this work we provide the first algorithm able to play *Gomoku-swap2*.

The algorithm MCTS got positive results in the similar board games like *GO* [16], *chess* [10], *othello* [14], video games like *Total War: Rome II's Campaign* [4] where the enormous size of branching factor and small amount of time are challenging or even in the imperfect games with complete information like *Poker* [18].

To solve the game we use the MCTS with advanced heuristic guiding the search. Similarly to Deepmind's AlphaGo (is the first AI who beat the best professional human player Lee Sedol in the game GO) we use the neural network as the heuristic for guiding the search [13][16]. We are interested in applying the similar configuration of the Neural Network with less hidden layers and a smaller number of filters because we want to use it on the personal computer not on the compluter's cluster like Deepmind's AlphaGO.

I trained the Neural Network using 1,5 million games played by the human players downloaded from the online server *playok.com* and offline server *piskvorky.cathedral.cz*.

The experimental evaluation shows that the MCTS with the Neural Network dominates in the *Gomoku-swap2* with 90% winning rate against basic MCTS even though the basic MCTS did one thousand times more iterations of

MCTS algorithm than MCTS with Neural Network in the given time limit. Which means that basic MCTS is one thousand times faster than the MCTS with the Neural Network.

Chapter 2

Technical Backgroud

In this chapter I will explain basic parts of the *Game Theory* needed for understanding how the *Game Theories* are applied in game *Gomoku-swap2*.

2.1 Agents and Utility

Assume we have agents in some environment. This environment is built from states. The transition between states is realized by taking an action. To evaluate an action we will use utility function, which maps actions from agents states to real numbers. It helps us to choose the best action because these numbers give us information about how good an action is for an agent.

Example 2.1.1. *Student and School*
We begin with a simple example to demonstrate how utility function works and how we will use it. Consider an agent student with name Gerry who is in the fourth year of high school in last week before graduation. He has tree options go to school immediately (i), sleep a few hours more and go to school later (l), or not to go to school at all (a). If he is on his own, Gerry has a utility of 5 for i, 30 for l and 15 for a. However, Gerry is also interested in the activities of two other agents Veronica and Jane. If he will be late Veronica will be sad and she will stop talking to him for 2 days but only if she will be at school. Jane, on the other hand, wants him to stay at home to be able to sit with Veronica in school and she will let her materials for graduation tests to Gerry but only if Veronica is at school. Unfortunately, there is only 50% chance that Veronica will be at school (s_V) and only 30% that Jane will be at school (s_J). Gerry needs materials so if he got them his utility will be increased by 200% but two days of not talking with Veronica got him sad and the utility will decrease by 50% (utility decreases after the increase due to materials from Jane). Now it will be easier to determine what choice Gerry should choose. There are only 12 outcomes. Because Gerry has 3 options and two other agents both have only 2 options. Lets describe outcomes, where u is utility:

$$
\begin{aligned}
u_G(i) &= 5\\
u_G(l) &= u_V(\neg s) \cdot 30 + u_V(s_V) \cdot [30 \cdot 0.5]\\
u_G(l) &= 0.5 \cdot 30 + 0.5 \cdot [30 \cdot 0.5 = 22.5\\
u_G(a) &= u_V(s_V) \cdot 15 + u_V(\neg s) \cdot [(u_J(\neg s_J) \cdot 15) \cdot u_V(s) + u_J(s_J) \cdot (15 \cdot 2) \cdot 0.5]\\
u_G(a) &= 0.5 \cdot 15 + 0.5 \cdot [(0.7 \cdot 15) \cdot 0.5 + 0.3 \cdot (15 \cdot 2) \cdot 0.5] = 12.375
\end{aligned}
$$

Now we can see that if Gerry gets up and will go to school immediately he will reach $u_G(i) = 5$. However, if he changes his mind and will go to school later he will get the highest result $u_G(l) = 22.5$ even if there is a chance that Veronica won't be here.

2.2 Games in Normal Form

The normal form is the most common representation of a game. The game in normal form is usually represented with an n-dimensional matrix. This representation does not include any notion of time or sequence, of actions of the player which mean that the players are assumed to move simultaneously without any information about the opponents.

Definition 2.2.1. Games in normal form [15]

Normal-form game is tuple (N,A,u), where:

- *N is finite set of n players, indexed by i.*
- *$A = A_1 \times \cdots \times A_i$ where A_i is a finite set of actions available to player i. Each vector $a = (a_1, \ldots, a_n) \epsilon A$ is called an action profile;*
- *$u = (u_1, \ldots, u_n)$ where $u_i : A \to \mathbb{R}$ is a real-valued utility (or payoff) function for player i.*

Assume game of *choosing the color* in $Table$ 2.1 where you need to choose the black or the white color. The N is 2, A is {Black, White}×{Black, White} and $u = (u_{1bb}, u_{1bw}, u_{1wb}, u_{1ww}, u_{2bb}, u_{2bw}, u_{2wb}, u_{2ww})$ where u_{1bb} *is* -1, u_{1bw} *is* 2, u_{1wb} *is* 1, u_{1ww} *is* -2, u_{2bb} *is* -1, u_{2bw} *is* 1, u_{2wb} *is* 2 and u_{2ww} *is* -2 u_2.

Choosing color			
		Player 2 b	Player 2 w
Player 1	b	-1, -1	2, 1
Player 1	w	1, 2	-2, -2

Table 2.1: We depict a game where two players simultaneously choose a color. The b is the black color and the w is the white color.

2.2.1 Zero-sum Games

Zero-sum games are two-player games where for any action profile the player 1 has opposite utility value to player 2.

Definition 2.2.2. Zero-sum Games [15]
A two player normal-form game is zero-sum game if for each strategy profile $a \epsilon A_1 \times A_2$ *it is the case that* $u_1(a) + u_2(a) = 0$.

Assume the player one will play an action rock in the game $rock, scissor, paper$ in $Table$ 2.2 and player two will play scissor. Then the player one gains the utility 1 while the player two gains utility 0.

Rock, Scissor and Paper				
		Player 2 Rock	Player 2 Scissor	Player 2 Paper
Player 1	Rock	0, 0	1, -1	-1, 1
Player 1	Scissor	-1, 1	0, 0	1, -1
Player 1	Paper	1, -1	-1, 1	0, 0

Table 2.2: A rock-scissor-paper game.

2.2.2 Strategies in Normal-form Games

Best strategies are used to describe behavior of players.

We have two basic strategy representations in normal-form games.

First one is the pure strategy, which represent deterministic play. The pure strategy in normal-form corresponds to actions of players.

The second one is a mixed strategy, which represent non-deterministic play.

Definition 2.2.3. Mixed strategy [15]
Let (N, A, u) *be a normal-form game, and for any set* X *let* $\Pi(X)$ *be the set of all probability distributions over* X*. Then the set of mixed strategies for player* i *is* $S_i = \Pi(A_i)$.

Assume the game depicted in $Table$ 2.2 rock, scissor and paper. In this game, one of the pure strategies is to play always rock.

The mixed-strategy means that you will play rock with probability p_1, scissor with probability p_2 and paper with probability p_3 where $\sum p_i = 1$.

The pure strategy is a special case of mixed strategy where we use only one possible action with probability 1 (it means all of them have probability 0 except one). Fully mixed strategy means that there is for every action probability bigger than zero. In multiple agents games is hard to find optimal strategy because the outcome of player 1 depends on the outcome of player 2.

2.2.3 Nash Equilibrium

Now we will look at games from an individual agent's point of view, rather than from the vantage point of an outside observer. This will lead us to the most influential solution concept in game theory, the Nash equilibrium [15].

Our first observation is that if an agent knew how the others were going to play, his strategic problem would become simple. Specifically, he would be left with the single-agent problem of choosing a utility-maximizing action. Formally, define $s_{-i} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$ a strategy profile s without agent i's strategy. Thus we can write $s = (s_i,\ s_{-i})$ [15]. The first player expects the best strategy from the second player and the second player expects best strategy from the first player. This leas to Nash equilibrium because it expects from each player the best strategy.

Definition 2.2.4. Best response [15]
Best response of player i to the strategy s_{-i} is a mixed strategy s_i^* such that $u_i(s_i^*,\ s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \epsilon S_i$.

From the definition we can see that there can exist more best responses because there can be two or more best responses with the same result.

Definition 2.2.5. Nash equilibrium [15]
A strategy profile $s = (s_1, \dots s_n)$ is a Nash equilibrium if, for all agents i , s_i is a best response to s_{-i}.

Assume the example *rock, scissor, paper* in *Table* 2.2. The Nash Equilibrium is playing *rock* with $\frac{1}{3}$, *scissor* with $\frac{1}{3}$ and *paper* with $\frac{1}{3}$.

2.3 Extensive-form Games

Extensive-form is another representation of the game. In this thesis we focus only on perfect information extensive-form games which are commonly visualized as game trees as show in *Figure* 2.1. Opposite to normal-form, it includes time, or sequence, of actions of the players. Informally speaking, the node of the tree is state for the player in the environment and the edge from one node to another node is an action of the player. The terminal states correspond to leafs of the tree. The utility function set values to terminal states (nodes).

Definition 2.3.1. Extensive form game [15]
A perfect information (finite) game(in extensive form) is a tuple $G = (N,A,H,Z,\chi,\rho,\sigma,u)$, where:

- *N is a set of n players;*

- *A is a (single) set of actions;*
- *H is a set of nonterminal choice nodes;*
- *Z is a set of terminal nodes, disjoint from H;*
- *χ: $H \rightarrow 2^A$ is the action function, which assigns to each choice node a set of possible actions;*
- *ρ: $H \rightarrow N$ is the player function, which assigns to each nonterminal node a player $i \epsilon N$ who chooses an action at that node;*
- *σ: $H \times A \rightarrow H \cup Z$ is the successor function, which maps a choice node and an action to a new choice node or terminal node such that for all $h_1, h_2 \epsilon H$ and $a_1, a_2 \epsilon A$, if $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ then $h_1 = h_2$ and $a_1 = a_2$; and*
- *$u = (u_1, \ldots, u_n)$, where $u_i : Z \rightarrow \mathbb{R}$ is a real-valued utility function for player i on the terminal nodes Z.*

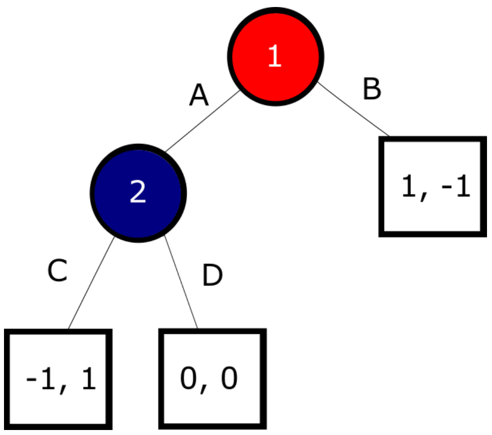

Figure 2.1: The circle nodes (red-player 1, blue-player 2) are nonterminal nodes. The square nodes are terminal nodes with utility values (-1,1 says that the first player lost and the second player won). The edges {a,b,c,d}=A are actions of the player.

Chapter 3

Gomoku

Gomoku [8] also know as *tic tac toe* is a *two player* game. It is played on a board with black and white stones on 15x15 intersection made of wood. The starting position of a game is empty board. The player who has black stones always starts the game. Stones need to be put on intersections, not on the squares. The stones already put on the board cannot be removed or changed until the end of the game. The winner of the game is the first player who get unbroken exactly five stones in a row either horizontally, vertically, or diagonally. More stones than five in a row is called *overline* and it is not considered as win.

Example 3.0.1. [7]

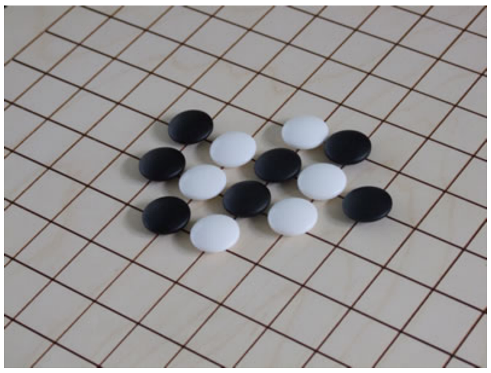

Figure 3.1: Gomoku board and stones.

3.1 Development of Gomoku Rules

The Gomoku itself is old few thousands of years.

Basic rules are really simple. The first player puts the first black stone on the board then the second player puts the first white stone on the board then the first player puts the second black stone on the board and they continuously put their stones until one of them has win or board is full.

In the 19th century people started to realized that the first player can always guarantee to win on the start of the game so called *surewin*. This was proven with *L. Victor Allis* in 1994 who comes with the algorithm of proof-number search and dependency-based search [2]. He even proved that restriction to exactly five stones in a row doesn't change the guarantee to win of first player.

The problem with a huge advantage of the first player brings several variations of Gomoku. Most know are $gomoku-pro$, $gomoku-long-pro$, $swap$ and $swap2$.

3.1.1 Gomoku-pro

The starting player (black) puts the first stone to the middle intersection of the board (H8), this move is compulsory. The second player can put the second move anywhere on the board. Now it's black's turn and the third move has to be outside a 5x5 square from the centre of the board (H8) [8]. Even after this restriction shown in *Figure* 3.2 [8], black player has huge advantage maybe even sure win too but it was still not mathematically proven.

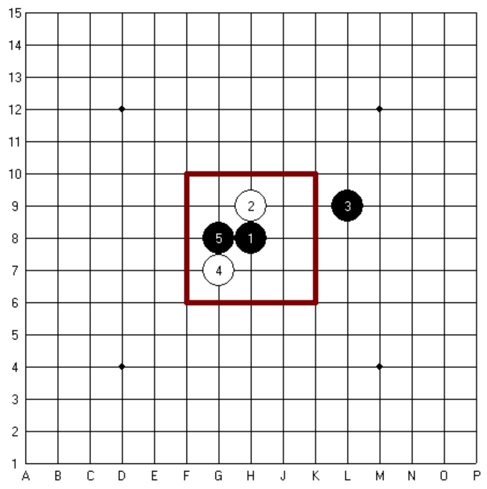

Figure 3.2: The second player is on the move. Black stones have advantage even when the 3th and 5th moves were played poorly.

3.1.2 Gomoku-long-pro

The first move of the starting player (black) is compulsory to be put to the middle intersection of the board (H8). Then the second player (white) can put the second stone anywhere on the board. The 3rd move must be put outside a 7x7 square. The centre of the square is the first black stone on H8 [8].

These rules (shown in *Figure* 3.3 [8]) improve the balance between players than $Gomoku-basic$ or $Gomoku-pro$. Unfortunately, only a few variants in these rules were possible to play for the first player due to restrictions and possible chances of the second player.

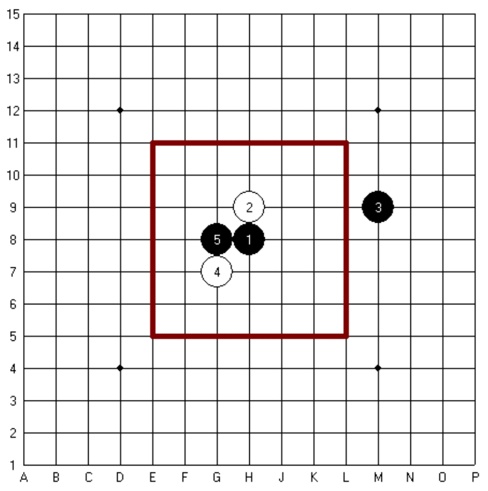

Figure 3.3: This opening was used a lot in the swap but last few years is considered as a sure win for white with top world's players. We can compare with *Figure* 3.2 how situation change when only one stone is shifted from *L9* to *M9*.

3.1.3 Gomoku-Swap

The starting player puts the first three stones anywhere on the board (two black stones and a white one). The second player can decide whether s/he wants to keep white stones and put a fourth stone or s/he can swap and control the black stones. After this start the players keep on playing till someone gets five in a row or board is full [8].

These rules had only one problem. It has to much weight on the decision of the choosing color. The second player needs to calculate the whole game to choose correctly or he will risks to choose wrong color.

3.1.4 Gomoku-Swap2

Gomoku-swap2 rules [8]:The first player puts three stones (two blacks and one white) on any intersections of the gomoku board. The second player has three options now:

- *s/he can choose white and puts the 4th stone*
- *s/he can swap and controls the black stones*
- *s/he can put two more stones (one black and one white stone) so there will be a position composed of five stones on the board and s/he passes the opportunity to choose color to the opponent.*

- *If the second player choose to add two more stones the first player is forced to choose the color.*

- *After this start the players keep on playing till someone gets five in a row or board is full.*

These rules shown in ($Figure$ 3.4) change Gomoku from the view of players to fair game. It allows the first player to prepare strong opening (first three moves) as in $Gomoku - swap$ where he knows best moves until the end of game. But in *Gomoku-swap2* second player has the opportunity to add two more stones to prevent first player to win with only prepared best moves. This makes the game more complex and more interesting. It leads to strategies like to choose the complex openings or preparing complex 4^{th} and 5^{th} moves to highly expecting openings from the first player.

This rule is adapted and used with professional players in the world.

In this thesis we used *Gomoku-swap2* rules in our artificial intelligence.

Example 3.1.1. *Gomoku-swap2*

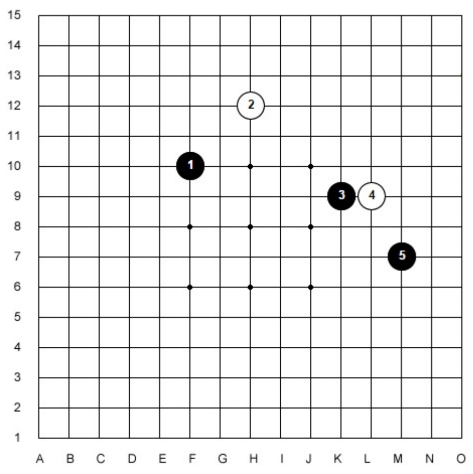

Figure 3.4: The position of the game after the second player added two stones. The first player is choosing the color of stones now.

Chapter 4
Algorithms

In this chapter, we focus on Minimax and Monte Carlo Tree Search (MCTS). These algorithms are commonly used to solve perfect information games like *Gomoku-swap2*.

4.1 Minimax

Minimax is a recursive algorithm for finding the best action in a given state of perfect information extensive form game.

The algorithm is applied to the tree of the game. *Figure* 4.1 demonstrates the search. The arrays show the way how the algorithm search the game tree. The algorithm propagate the values in the bottom up fashion.

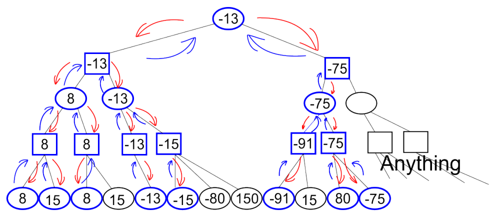

Figure 4.1: The circles represent maximizing player (the player who start) and the squares represent the opponent (minimizing player). The red and the blue arrows show the path of the algorithm where the algorithm went always to the right in the direction of dart. The black circles and squares represent the nodes that algorithm didn't visit due to pruning.

In the *Algorithm* 1 we provide the pseudocode of the minimax algorithm. When evaluating node, the algorithm just go recursively in the left most child of the node of the tree until we reach the leaf. After that, if the parent node has more children we will continue to the next child and recursively we will get the best value for the root of the game tree. if we maximize we compare the value of leaf with the max value that we got from the parent node and return the bigger value to the parent node. If we minimize we compare the min value with the leaf value and return the smaller value.

We assume the positive infinity to be the win of the maximizing player, negative infinity to be the win of the minimizing player and zero the draw of the game. Next values assume the evaluation of the states of game where we do not reach the end of the game due to limited resources.

Algorithm 1 minimax(node, depth, maximizingPlayer)

```
if if depth = 0 or node is a terminal node then
  return the heuristic value of node
end if
if maximizingPlayer then
  $bestValue \leftarrow -\infty$
  for all child of node do
    $V \leftarrow minimax(child, dept - 1, FALSE)$
    $bestValue \leftarrow max(bestValue, V)$
  end for
else
  $bestValue \leftarrow \infty$
  for all child of node do
    $V \leftarrow minimax(child, dept - 1, TRUE)$
    $bestValue \leftarrow min(bestValue, V)$
  end for
  return bestValue
end if
```

Definition 4.1.1. *Evaluating tree for Minimax* [19]
The values of nodes comes from a rule that if the result of a move is an immediate win for player 1 it is assigned positive infinity and, if it is an immediate win for player 2, negative infinity. The value to player 1 of any other move is the maximum of the values resulting from each of player's 2 possible replies. For this reason, player 1 is called the maximizing player and player 2 is called the minimizing player, hence the name minimax algorithm. The above algorithm will assign a value of positive or negative infinity to any position since the value of every position will be the value of some final winning or losing position.

In the large games like Gomoku, chess or GO it is not possible to reach all leafs of the game tree due to computations resources.

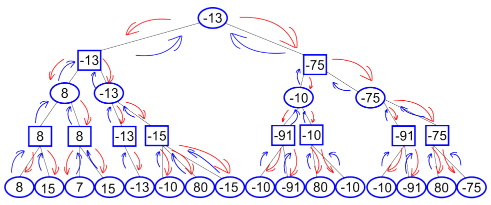

Figure 4.2: The circles represent maximizing player (the player who start) and the squares represent the opponent (minimizing player). The red and the blue arrows show the path of the algorithm where the algorithm went always to the right in the direction of dart. We can see that algorithm visited all nodes of tree.

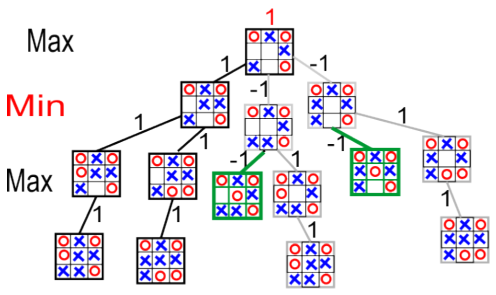

Figure 4.3: Assume a game where two players play tic-tac-toe against each other. In own turn, you choose the action with max value and second player will play his max value in next action. So you need to minimize the maximum loss. The dark black nodes show the visited nodes with maxmin, the light grey shows the skipped nodes with maxmin and the green nodes show the nodes which would be selected with maxmin algorithm if the first branch would be not winning branch.

The essential attribute of minimax is pruning. The algorithm prunes a branch if there is no chance to get better result from this branch. I will show three examples. First: Assume example in *Figure* 4.1. The minimax algorithm didn't visit 3 leafs from 8 leafs in the left branch of the root. In the right branch of root it even skipped few nodes which were not leaf at all because the root of the tree send the -13 value to the minimizing child as the lower bound and the child get the value -75 from own children as upper bound (-13, -75), which is the empty interval. For this reason is not need to search next children because minimizing node has guarantee that the best move is smaller than -13 and hence, the maximizing root will always prefer -13.

Second: Assume example in *Figure* 4.3. The minimax algorithm visited only the left-most branch of the root.

Unfortunately, it can happen that minimax will prune nothing and that it will search whole game tree (it is the worst case) as we can see in *Figure* 4.2.

Games with the huge game tree cannot be solve with *minimax* due to limited computational resources and it is difficult to get good heuristics. This reason leads to approximation algorithm such as Monte Carlo Tree Search.

4.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a best-first search technique algorithm used for the games with the finite length like *GO, chess, poker, Gomoku* [16][5] and many others. The basic *MCTS* is illustrated in *Figure* 4.4.

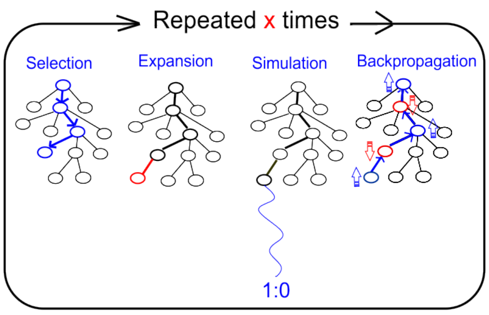

Figure 4.4: Monte Carlo Tree Search.

The main idea of the algorithm is to build a tree of the most promising nodes chosen with some heuristics. The leafs of the Monte Carlo tree are evaluated based on random simulations of the game starting in the state corresponding to the leaf. The algorithm balances between expanding nodes of most promising branches (exploitation) and expanding nodes of less promising branches (exploration).

The node of Monte Carlo tree used in Gomoku store from: a position of stone, a color of the stone, black stone's owner, an array of possible children, a number of visits and score reached from sum of how many times the node was in *win*, *lost* or *draw* games.

Definition 4.2.1. Monte Carlo Tree Search [5]
The algorithm iteratively uses and builds the Monte Carlo tree of the most promising future game states, according to the following mechanism:

Selection
While the state is found in the Monte Carlo tree, the next action is chosen according to the statistics stored, in a way that balances between exploitation and exploration. On the one hand, the task is often to select the game action that leads to the best results so far (exploitation). On the other hand, less promising actions still have to be explored, due to the uncertainty of the evaluation (exploration).
Expansion
When the selection reaches the first state that cannot be found in the Monte Carlo tree, the state is added as a new node. This way, the tree is expanded by one node for each iteration of MCTS (selection, expansion, simulation, backpropagation).
Simulation
For the rest of the game, actions are selected at random until the end of the

game. Naturally, the adequate weighting of action selection probabilities has a significant effect on speed of convergence of MCTS to best actions. We can use heuristic knowledge to give larger weights to actions that look more promising to increase convergence of MCTS.

Backpropagation
After reaching the end of the simulated game, we update each Monte Carlo tree node that was traversed during that iteration. The visit counts are increased and the win/loss ratio is modified according to the outcome.
The game action finally executed by the program in the actual game, is the one corresponding to the child which was explored the most.

It was proven that the evaluation of moves in MCTS converges to results of minimax algorithm [12][]. Unfortunately, the basic MCTS converges very slowly.

4.2.1 Types of MCTS

The most know type of MCTS is the MCTS using UCT (Upper Confidence Bound 1 applied to trees) function in the *selection* part of MCTS.

Definition 4.2.2. UCT [3]

UCT is the evaluation function $f = \max_{j \epsilon 1,2,\ldots,k} V(j)$, *where* k is number of children, *and* $V(j)$ *is value of* j^{th} *child:*

$$V(j) = \bar{X_j} + 2\times C_p \times \sqrt[2]{\frac{2\times ln\ (N)}{n_j}}$$
where :

- $\bar{X_j}$ *is* $\frac{\sum 1\cdot win, 0\cdot draw, -1\cdot lost}{number\ of\ simulations}$
  - *win - how many times the node was in the winning sequence*
  - *draw - how many times the node was in the drawing sequence*
  - *lost - how many times the node was in the loosing sequence*
- C_p *is the constant representing the exploration/exploitation tradeoff*
- n_j *is the number of times the* j^{th} *child has been visited*
- N *is the number of times current (parent) node has been visited*

We experimented with additional selection functions (Exp3 [3]), however UCT had the best performance in the game Gomoku.

4.2.2 Problem with Simulation

The simulation part affects how the Monte Carto tree converges to the results of the minimax algorithm. The random simulation converges slowly due to the large branching factor. The random simulation needs a high amount of simulations to reach the right results.

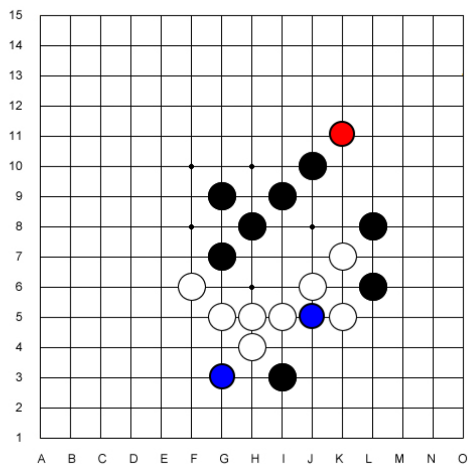

Figure 4.5: Assume this position to be evaluated by a random simulation. The player with black stones is on the move and has the opportunity to win with only one move(k11). If he choose different move than the white color will have the chance to win the game with j5 or g3.

We can see in *Figure* 4.5, that black player is on the move and he has only one action (K11) how to win and white player will than has two actions (j5, g3) how to win if the black player will play anything else then K11. It points out that only specific configurations are winning and the random simulations have a small probability of reaching them, so extremely high amount of simulations is need which is time consuming.

4.2.3 Improving Simulations

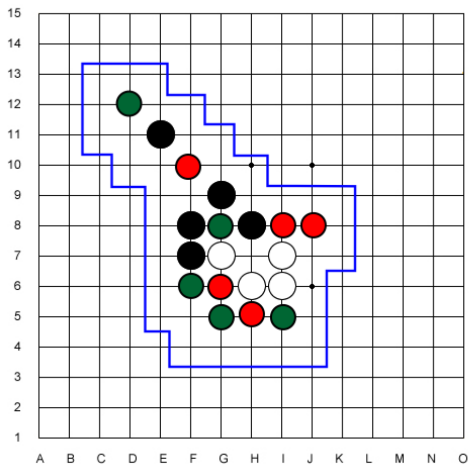

Figure 4.6: Improved random simulation

Assume the position in *Figure* 4.6 to be the start of a random simulation. The blue polygon restricts the possible moves for random playing where each intersection inside of the blue polygon is the possible move for the random player. The black stones and big white stones represent the part of the real game. Green stones (white stones) with red stones (black stones) were taken with the selection. The green stones assume the white stones.

This version of simulations we use in the MCTS Basic and the same principle we use in expansion of the MCTS Basic. This improvement increased the efficiency of MCTS Basic.

The bad results of MCTS when playing the game *Gomoku-swap2* convinced us to search for a heuristic to guide the selection and simulation. We inspired with *AlphaGo* [16] - the first program in game *GO* who beat human. They used Neural Network to guide the search of MCTS.

Chapter 5

Neural Network

In this chapter, we explain the idea of the Neural Networks and describe the Neural Network used as a heuristic in MCTS applied to *Gomoku-swap2*.

5.1 Idea of Neural Network

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated [1].

Definition 5.1.1. *Neural Networks* [11]
A neural network is a sorted triple (N, V, w) with two sets N, V and a function w, where N is the set of neurons and V a set $\{(i, j)|i, j \epsilon N\}$ whose elements are called connections between neuron i and neuron j. The function $w : V \rightarrow \mathbb{R}$ defines the weights, where $w((i, j))$, the weight of the connection between neuron i and neuron j, is shortened to $w_{i,j}$. Depending on the point of view it is either undefined or 0 for connections that do not exist in the network.

5.1.1 Multi Layer Network

A multilayer network is a network consisting of several layers of neurons which are interconnected. The input layer is stacked onto the first-layer neural network and a feed-forward network (A neural network that takes the initial input and triggers the activation of each layer of the network successively, without circulating. Feed-forward nets contrast with recurrent and recursive nets in that feed-forward nets never let the output of one node circle back to the same or previous nodes). Each subsequent layer after the input layer uses the output of the previous layer as its input [1].

The use of the convolution Neural Network was motivated by its sucess in *AlphaGO* [16].

Definition 5.1.2. Convolution [17]
For simplicity we assume a grayscale image to be defined by a function

$$I : \{1, ..., n_1\} \times \{1, ..., n_2\} \rightarrow W \subseteq \mathbb{R}, (i, j) \rightarrow I_{i,j}$$

such that the image I can be represented by an array of size $n_1 \times n_2$ *Given the filter* $K \epsilon \mathbb{R}^{(2h_1+1)\times(2h_2+1)}$*, the discrete convolution of the image I with filter K is given by*

$$(I \cdot K)_{r,s} := \sum_{u=h_1}^{h_1} \sum_{v=h_2}^{h_2} K_{u,v} I_{r+u,s+v}$$

where the filter K is given by

$$K = \begin{pmatrix} K_{-h_1,-h_2} & \dots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \dots & K_{h_1,h_2} \end{pmatrix} \tag{5.1}$$

Note that the behavior of this operation towards the borders of the image needs to be defined properly. A commonly used filter for smoothing is the discrete Gaussian filter $K_{G(\sigma)}$ *which is defined by*

$$(K_{G(\sigma)})_{r,s} = \frac{1}{\sqrt[2]{2\pi\sigma^2}} \exp(\frac{r^2 + s^2}{2\sigma^2})$$

where σ *is the standard deviation of the Gaussian distribution.*

Definition 5.1.3. Convolution Layer networks [17]
Let layer l be a convolutional layer. Then, the input of layer l comprises $m_1^{(l-1)}$ *feature maps from the previous layer, each of size* $m_2^{(l-1)} \times m_3^{(l-1)}$*. In the case where* $l = 1$*, the input is a single image I consisting of one or more channels. This way, a convolutional neural network directly accepts raw images as input. The output of layer l consists of* $m_1^{(l)}$ *feature maps of size* $m_2^{(l)} \times m_3^{(l)}$*. The* i^{th} *feature map in layer l, denoted* $Y_i^{(l)}$*, is computed as*

$$Y_i^{(l)} = B_i^{(l)} + m_1^{(l-1)} \sum_{j=1} K_{i,j}^{(l)} \cdot Y_j^{(l-1)}$$

where $B_i^{(l)}$ *is a bias matrix and* $K_{i,j}^{(l)}$ *is the filter of size* $2h_1(l)$ *+ 1* $\times$ $2h_2^{(l)}$ *+ 1 connecting the* j^{th} *feature map in layer (l - 1) with the* i^{th} *feature map in layer l.*

For mathematical purposes, a convolution is the integral measure of how much two functions overlap as one passes over the other. Think of a convolution as a way of mixing two functions by multiplying them [1].

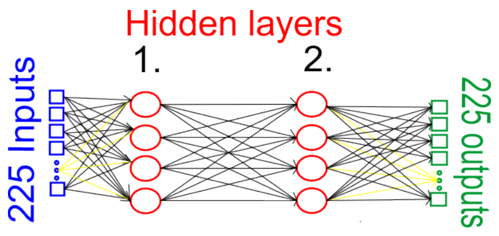

Figure 5.1: My Neural Network has 225 inputs (blue squares), two hidden convolution layers of kernel size 5×5 both with four filters(red circles) and 225 outputs . The black arrows show connections.

I trained Neural Network to be able to predict next move of player. I passed in the board position as a 15×15 matrix. I used convolution layers due to efficiency in the game *GO* [16]. As you can see in *Figure* 5.1 my Neural Network has 225 inputs (blue squares), two hidden convolution layers of kernel size 5×5 both with four filters (red circles) and 225 outputs (values of output are probabilities of the board positions where we accept only these values which are not positions of the stones on the board). The black arrows show connections. For the activation, I used rectifier activation function [20]. The number of layers and the number of filters are small due to the speed of Neural Network because we need it for the simulation games.

5.2 Data

In the beginning the data look like

1. h8 g7 2. e11 white 3. – i7 4. h6 h7 5. j7 g6 6. f7 i6 7. i8 g8 8. g9 f9 9. g9 e10 0-1

where strings *1.,2.,3.* assume the number of the round. One round is formed from two actions *(h8, g7)*. One action of player 1 *(h8)* and one action of player 2 *(g7)*. Strings like *h8* assume the position on board (the matrix (15×15) indexed by the set $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\} \times \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$, $a1$ assume the intersection fulfill the condition first row and the first column), the *h8* is the center of the board, the string *white* says that the player 2 choose the *white* color, the string – says that the player 1 is not allowed to play in the round and the string *0-1* says that player 2 won the game.

We take the original data (above) and transform them to the configuration of the board and the last action of the player. Because there were several actions for each configuration which complicated the training of the Neural Network, we change the data to the configuration of the board and the relative frequency of the moves (from data).

The process of changing data:

We transformed data from matrix (15×15) to the vector (the length 225). For the example the *h8* changed to 112. This vector we used as input for the

Neural Network.

We separate the game to subgames where the subgame is first $x-1$ actions of the game and the x^{th} action is the next expected action. The x is from range $2-k$ where the k is the number of moves of the game. For this reason, the each game gives us $9-223$ positions (the input data for the Neural Network).

To increase the performance of training the Neural Network we merge the different expected actions for the same inputs together where the probabilities of the expected actions of the new output sum to *1*.

We have used 1,5 million games (25 million positions) of real players from piskvorky.cathedral.cz and playok.com.

We separated the data for the Neural Network to test and train data where the test data served for controlling how good the Neural Network is. 100000 positions were used for the test data. The rest (24900000) were the training data.

5.2.1 Training of Neural Network

To measure how fast the Neural Network is learning we measure score at iteration and score of multiplying output from the Neural Network with the expecting output.

Score at iteration [1]
This is the value of the loss function on the each 2000 input data. We used the Multiclass Cross Entropy [1].

In the *Figure* 5.2 we can see that the loss function decreased (the positive sign of training).

Score of output
Predicted output (the vector $p = [p_0, p_1, p_2, \ldots, p_{224}]$*) multiplied by the expected output (the vector* $e = [e_0, e_1, e_2, \ldots, e_{224}]$*),* $SO = \sum_{j\epsilon 0,1,\ldots,224} p_j \cdot e_j$.

In the *Figure* 5.3 we can see that the score continuously increased on test data but the score of training data oscillate due to different inputs of training data (each next point is a different input of training data).

5.3 Final Neural Network

The best final Neural Network was trained for 15000 *epoch* on *MetaCentrum*. One epoch contained 10000 positions which were *fit* to the Neural Network.

The *Figure* 5.2 shows that the *Multiclass Cross Entropy* loss function continuously decreased. After 50000 iterations we can observe huge differences. It can be caused by the large amount of data compared to the small Neural Network.

The *Figure* 5.3 shows that the Neural Network continuously increased in the score on the testing data but the score of the training data oscillate due to different files of the training data (each next point is a different file of the training data).

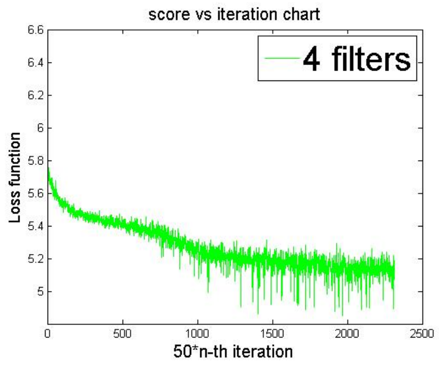

Figure 5.2: We can see that the inputs fitting to Neural Network has huge differences and that it let score oscillate too much.

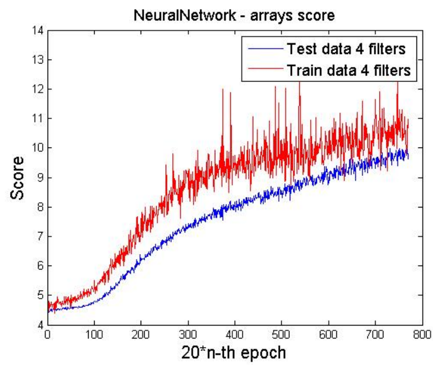

Figure 5.3: We can see that the score continuously increased on test data but the score of training data oscillate due to different files of training data (each next point is a different file of training data).

5.3.1 Application of Neural Network to MCTS

We use the Neural Network as the heuristic in the simulation ($\max_{j\epsilon 0,1,\ldots,224}(I(j))$, where $I(j)$ is the j^{th} position of the board (need to be empty)) and in the expansion (empty positions with the value bigger than 0.013) parts of the Monte Carlo Tree Search (MCTS).

The average time of query to the Neural Network is 3.8ms on 1.6GHz processor.

Now we can compare the Neural Network simulation with the random simulation from *subsection* 4.2.3.

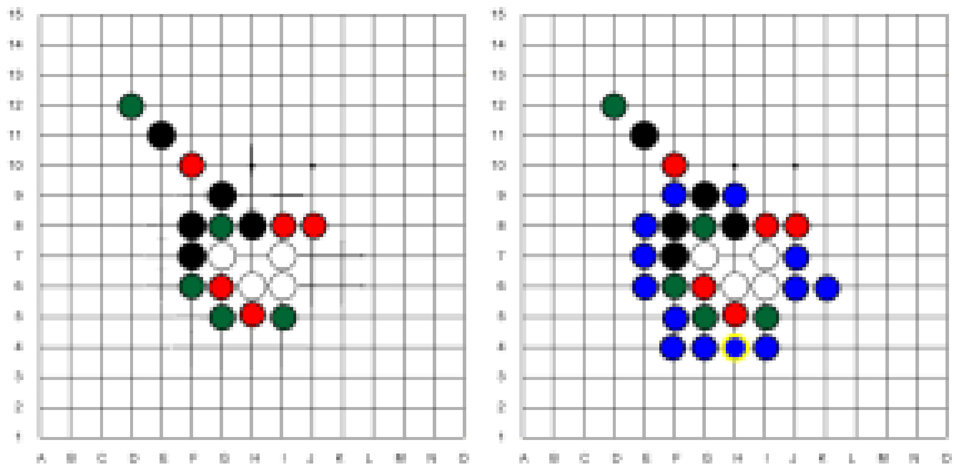

Figure 5.4: The left picture is the input for the Neural Network. The right picture contains the input with the output of the Neural Network (the blue circles).

The position in the left picture in *Figure* 5.4 is the input for the Neural Network. The blue stones in the right picture in *Figure* 5.4 are the possible actions from the Neural Network output for playing with probability bigger than 0.015. The blue stone with the yellow circle is the move with the highest probability. The black stones and the white stones represent the part of the real game. The green (white) stones with red (black) stones were chosen with the selection.

Now we can compare random simulation in *Figure* 4.6 with the Neural Network simulation in *Figure* 5.4 . As we can see the Neural Network prefer only a few stones. This can be used in selection because Neural Network detects the area of the desk where is wise (from the point of the human) to play the next action. The second advantage of Neural Network is searching for patterns like *position moves* or *threes* (three stones in a row) and *fours* (four stones in a row). Unfortunately, Neural Network has one disadvantage. From the experiments in chapter 6 we know that the MCTS basic (with improved random simulation) does *1000 times* more iterations of MCTS than the MCTS with the Neural Network simulation. However the MCTS with the Neural Network is still better than MCTS basic.

Chapter 6

Experiments

In this chapter, we present the comparison of experiments.

We compared basic MCTS with the improved simulations (MCTS basic) versus MCTS with the Neural Network (MCTS NN), basic MCTS versus Neural Network and MCTS NN versus Neural Network.

6.1 MCTS Basic vs. MCTS NN

First experiment compares MCTS Basic and MCTS NN where both have the same time to next move. The time was set to 1 minute for one move. The MCTS basic made approximately 20 million of iterations and the MCTS NN only 2 thousand of iterations in the given time .

The experiment run 7 days on 11 cores on the $MetaCentrum$.

The MCTS NN lost only 203 games from 2100 and reached 90,3% winning rate. The standard deviation is 0.59.

The second experiment compares algorithms where both algorithms have the same number of iterations of MCTS. We set the number of iterations to 30000. The experiment runs only 7 days on two cores. The experiment completed 74 games. MCTS NN won 67 games and got 90,5% winning rate. We need to say that NN lost 6 games when it starts the game and only 1 game when the opponent start the game. The small amount of games is due to first experiment where the MCTS NN proves dominant. For that reason we expect that decreasing the amount of iterations of MCTS Basic and increasing the amount of iterations of MCTS NN not change the domination of MCTS NN.

6.2 MCTS Basic vs. Neural Network

This experiment was about to compares the Neural Network with the basic MCTS.

The experiment runs only a few hours because the Neural Network needs only a few *ms* to return the move.

In this experiment we set number of iterations of MCTS to 1000. I let the half of games start the Neural Network and the second half of games the MCTS Basic.

The result was 95% of winning rate for MCTS Basic from 1000 games.

6.3 MCTS NN vs. Neural Network

We did only one experiment. We limited the time to 1s on move for MCTS NN and it made approximately 50 iterations of the MCTS in the given time. The half of games start MCTS Alien and second half of games start the Neural Network.

The amount of games was set to 1000. The MCTS Alien lost only one game and this game start the MCTS Alien itself. The winning rate of MCTS Alien was 99,9%.

6.4 Conclusion of Experiments

The experiments show us that the basic MCTS and the simple Neural Network are bad but the combination of these two works fine. The experiments proved that basic MCTS is 1000 times faster than MCTS NN but it is quite inefficient.

Chapter 7

Conclusion

We create the first player for *Gomoku-swap2*. We use Monte Carlo Tree Search (MCTS) algorithm with the Neural Network as the heuristic for the simulation and the expansion parts (instead of the random heuristic).

We collect the data from human players. To train the Neural Network we modified these data to be able to use them for our machine learning methods - the convolution multilayer Neural Network. We experimented with data to increase the learning speed of the Neural Network.

We show that the Neural Network significantly improves the performance of MCTS. It proves that the Neural Network is acceptable as the heuristic for the MCTS. The only problem is slow speed of the Neural Network.

We show that the basic MCTS with random heuristic is not acceptable for the solving the *Gomoku-swap2* because of needs a huge amount of iterations to reach the right outcomes (it cost time).

As a future work, we need to increase the performance of the Neural Network. For increase the performance we need to use a different data for training the Neural Network. For example, the data played with professional players on the long time games (live tournaments).

Bibliography

[1] Chris Nicholson Adam Gibson. deeplearning4j. `https://deeplearning4j.org/`, 2014 (accessed April 30, 2017).

[2] Louis Victor Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen, 1994.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[4] Alex J. Champandard. Monte-carlo tree search in total war: Rome ii's campaign ai. `http://aigamedev.com/open/coverage/mcts-rome-ii/`, 2014 (accessed May 17, 2017).

[5] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*. Universiteit Maastricht, 2008.

[6] Daniel Ford. Gomoku ai player. 2016.

[7] Gomoku. Gomoku czech website. `http://www.piskvorky.cz/`, 2006 (accessed April 22, 2017).

[8] Gomoku. Gomoku world website. `http://gomokuworld.com/`, 2013 (accessed April 22, 2017).

[9] Jun Hwan Kang and Hang Joon Kim. Effective monte-carlo tree search strategies for gomoku ai. 2016.

[10] Andraž Kohne. Monte-carlo tree search in chess endgames. 2012.

[11] David Kriesel. A brief introduction on neural networks. 2007.

[12] Viliam Lisy, Vojta Kovarik, Marc Lanctot, and Branislav Bosansky. Convergence of monte carlo tree search in simultaneous move games. In *Advances in Neural Information Processing Systems*, pages 2112–2120, 2013.

[13] CADE METZ. Go grandmaster lee sedol grabs consolation win against google's ai. `https://www.wired.com/2016/03/go-grandmaster-lee-sedol-grabs-consolation-win-googles-ai/`, 2016 (accessed May 2, 2017).

[14] Archer Ryan. *Analysis of Monte Carlo Techniques in Othello.* PhD thesis, BS Thesis, The University of Western Australia, 2007.

[15] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2008.

[16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[17] David Stutz. Understanding convolutional neural networks. In *Seminar Report, Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr-und Forschungsgebiet Informatik VIII Computer Vision*, 2014.

[18] Guy Van den Broeck, Kurt Driessens, and Jan Ramon. Monte-carlo tree search in poker using expected reward distributions. In *Asian Conference on Machine Learning*, pages 367–381. Springer, 2009.

[19] wikipedia. minimax. `https://en.wikipedia.org/wiki/Minimax/`, 2017 (accessed May 3, 2017).

[20] wikipedia. Rectifier (neural networks). `https://en.wikipedia.org/wiki/Rectifier_(neural_networks)/`, 2017 (accessed May 3, 2017).

Appendix A
CD content

The CD contents three packages:

SourceCode - it contents the source code of the bachelor thesis.

GomokuSwap - this package contents jar file of program and the Neural Network (both need to be in the same folder to ensure working).

BachelorThesisText - it contents the pdf file of bachelor thesis.