A Study on Learning Algorithms of Value and Policy Functions in Hex

Kei Takada

Graduate School of Information Science and Technology
Hokkaido University
Sapporo, Hokkaido, Japan

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Masahito Yamamoto for constant encouragement and instruction to get the thesis done. His guidance helped me in all the time of research and writing of this thesis.

I am also deeply grateful to Associate Prof. Hiroyuki Iizuka for his encouragement, insightful comments, and hard questions. I would also like to thank the rest of my thesis committee: Prof. Masahito Kurihara, Prof. Tetsuo Ono, and Prof. Hidenori Kawamura. Their insightful comments incented me to widen my research from various perspectives.

I would also like to offer my special thanks to Prof. Ryan Hayward from the University of Alberta, who provided me an opportunity to stay in his group. Their great advice and technical supports have increased my motivation for my research. Without their precious support, it would not be possible to conduct this research.

I would like to thank all members of my research group, Autonomous Systems Engineering Laboratory. The conversations between them always encouraged me. Particularly I want to thank Masaya Honjo who give me much wholesome advice.

I am grateful to my parents Kaoru and Hisako, who have provided me through moral and emotional support in my life. I am also grateful to my other family members and friends who have supported me along the way. I would not have continued my research without their supports.

Lastly, I thank JSPS for financial support of this research. This work was supported by Grant-in-Aid for JSPS Research Fellow Grand Number JP 16J02092.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The board games have deep strategies and the charms attracting the people, and a lot of people play and enjoy the board games in the world. There are easy-to-understand results such as winning or losing, and it is easy to compare the strength of player by using the game results. The board game is sometimes called mind sport [min97], and there are the board games where human professionals exist. The board games are used not only for entertainment but also for the research, and the board games have been used as a testbed for a long time. Development of the computer player beyond human has been considered as grand challenge, and the studies to create strong computer players have been actively performed. Development of a computer player is a study on a search problem that determines the next move from a large search space. The computer player has to decide the next move in an environment where hostile players exist and the number of positions occurring in the future is huge. The search algorithms that are used to efficiently search the large search space could be applied not only for board games but also for other problems, e.g., planning problems, searching for new materials, and optimization problems. Studies on search algorithms are expected to be applicable to a broader range of artificial intelligence domains.

For an efficient search, two highly accurate evaluation functions are necessary. One is the value function which scores the position, and the other is the policy function which scores the candidate moves. The early period in the development of computer players, the evaluation functions have consisted of the features extracted by hand and their weights [CHhH02, ISR02]. The evaluation accuracy of the evaluation functions mainly depends on the weights. As the number of weights increases, it is difficult to determine the weights manually that improves the evaluation accuracy. To efficiently determine the better weights, a lot of machine learning algorithms that update the weights have been proposed [Tes95, HK14]. These methods have been shown to be able to create highly accurate evaluation functions in chess, shogi, and so on. However, extracting the features of position have to be performed by the human, and it is difficult to apply those methods into the board games where the extracting the features is difficult such as Go.

Recently, a convolutional neural network (CNN) is used for developing the evaluation functions, and it is shown that those evaluation functions have high evaluation accuracy even in Go [MHSS15, CS15]. CNN is a neural network model that has achieved great success in the field of computer vision, and it is shown that CNN can learn abstract features of images [KSH12]. The evaluation functions using CNN treat the position of the board as an image, and CNN can learn the features of the position that are difficult to be expressed manually. The highly accurate evaluation functions can be obtained by the learning that imitates the moves of experts; however, the evaluation accuracy of the evaluation functions created by imitation learning cannot exceed human. Therefore, a reinforcement learning algorithm using game of self-play has attracted attention because it can be expected to acquire the evaluation accuracy exceeding expert. In 2016, the computer player developed by reinforcement learning algorithm beat the top professional players of human in Go [SHM$^+$16]. This event received attention from all over the world, and the effectiveness of the reinforcement learning algorithm which creates evaluation functions using CNN was clarified.

1.2 Objective

I propose the learning algorithms for creating the functions necessary to develop the computer player and show the effectiveness of the proposed learning methods through the development of computer player. In this thesis, two learning methods are proposed roughly. One is the learning method that creates the classifier of the board states, e.g., opening game and end game. The other is a reinforcement learning algorithm using game of self-play to create the value and policy functions using CNN.

The board game to be used is Hex, which is a board game developed by Piet Hein and John Nash independently [Bro00]. Hex is classified as a two-player, zero-sum, and perfect information game. One of the features of Hex is that the first player has the winning strategy [Nas52], and the solvers have been actively developed to solve concrete winning sequences. In Hex, the computer player developed by the proposed method can be compared with the perfect player which is the solver. From the comparison, it can be clarified that whether the proposed method can create the perfect player. Hex is a board game with a relatively large search space in the games that have proved a winning strategy, and it is one of the advantages of using Hex that the proposed computer player can be compared with the perfect player in such a game. Also, the international competition of the computer player is held every year, and there are a lot of computer Hex players using different game tree search algorithms and evaluation functions. By using Hex, it is possible to discuss the performance of the computer player due to the differences in the tree search algorithms and the evaluation functions.

A shift of the strategy according to the board state during the game is often performed in Shogi because it allows the player to perform a better search according to the board state [ISR02]. The board states of the game are roughly classified into three, i.e., the opening game, the middle game, and the end game. The transitions of the phases can be clarified and described by rules to some extent in the long history of the studies. However, it is difficult to design the rules to determine the timing appropriately and handle exceptional positions. For the purpose of flexibly classifying the board state, I propose the learning method to create the classifier of the board state based on a support vector machine (SVM) [CV95]. The classifier determines whether it is better to change the strategy in a given position. In this thesis, at first, I develop a novel value function consisted of the global and local evaluations using the network characteristics calculated from the board networks. The player using the proposed value function can change the strategy by changing the ratio of global and local evaluations. Next, I propose the classifier to change the strategy according to the given position, and the classifier is trained by using the expert moves. To demonstrate that the created classifier can classify the board state appropriately and dynamically changing the strategy is effective, I develop the computer player using the proposed value function and classifier, and I compare it with the previous computer player.

In the above study, the board states are classified by two groups, and the value function is composed of the network characteristics extracted by hand. However, it is impossible to determine the number of board states in a real game, and there may be the features of the board that the used network characteristics cannot express. To create the evaluation functions that evaluate the position more flexibly and accurately, I focus on the reinforcement learning and the CNN. Reinforcement learning algorithm using game of self-play does not need to collect a huge amount of expert's data, and there is a possibility that the highly accurate evaluation functions exceeding human can be created; therefore, a lot of reinforcement learning algorithms have been proposed. Silver et al. proposed AlphaGo Zero algorithm and AlphaZero algorithm to create the value and policy functions using CNN [SSS+17, SHS+17]. In these algorithms, the value function is trained to predict the game result such as win or lose, and the policy function is trained to predict the search probabilities of each move output by monte carlo tree search (MCTS). It is shown that these algorithms can create highly accurate evaluation functions; however, a lot of simulations and high computational cost are necessary to obtain the search probability and train the functions. I propose a novel reinforcement learning algorithm using game of self-play to create the value and policy functions. The primary difference between the proposed algorithm and the previous algorithm is the learning method of the policy function. In the proposed algorithm, the policy function is trained based on the search result by the value function and the game result without using the search probabilities. The policy function makes it possible to appropriately determine the search targets while predicting the best move of the search result by the value function. It can be expected that the computational cost of the proposed algorithm is lower than that of the previous algorithms because it is enough for the self-play player to search the game tree with 1-ply, which means that the candidate move is sufficient to be evaluated only once. In this thesis, I propose the method that applies the CNN to Hex first. I create the input of CNN by focusing on three mutually adjacent cells in order to make it easy to learn the features of the position. From the comparison between the proposed CNN model and the linear model that uses the network characteristics, I demonstrate the effectiveness of the policy functions using CNN. Next, I propose the reinforcement learning algorithm that trains the value and policy functions from the game of self-play. The evaluation functions are trained by the proposed learning algorithm, and I develop the computer player using the trained evaluation functions. From the comparison between the proposed computer player and the previous computer players, I demonstrate that highly accurate evaluation functions can be created by the proposed algorithm.

1.3 Outline of Thesis

This thesis is organized as follows. In chapter 2, I review a board game Hex in detail. The rules, the features, the method of creating the board network, the specific methods of Hex, the game tree search algorithms, the major computer Hex algorithms, etc. are explained. In chapter 3, I describe the study that applies the complex network into Hex and creates the classifier using SVM. I treat the position as the board network and propose the novel value function consisting of the global and local evaluations using network characteristics. From the comparison between the computer player using the proposed value function and the previous computer player, I demonstrate that combining the global and local evaluations is effective. In addition, I create the classifier based on SVM that can classify the board state and develop the computer player who changes the strategy dynamically according to the output of the classifier. To demonstrate the classifier can classify the board state appropriately, I compare the proposed computer player with the previous computer players. In chapter 4, the value and policy functions using CNN are developed, and I propose the reinforcement learning algorithm to train the evaluation functions by using games of self-play. First, I train the policy function using CNN by the supervised learning and compare it with the linear policy function using the network characteristics in order to show that the CNN model can learn the features which are difficult to quantify by using board networks. Next, I train the value and policy functions by the proposed learning algorithm. It is demonstrated that the proposed evaluation functions have the high evaluation accuracy from the comparison between the proposed computer player and world-champion programs 2017. In Chapter 5, I summarize this paper.

Chapter 2

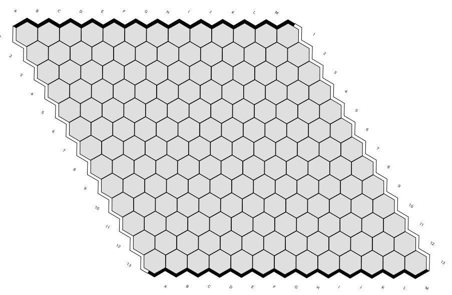
Hex

The game of Hex is a classic board game for two players. It was invented by Piet Hein in 1942, and it is also known that John Nash developed Hex independently in 1948 [Bro00]. Hex is classified as a two-player, zero-sum, and perfect information game.

In this chapter, I describe the features and rules of Hex first. Then, the studies of solving the winning strategy of Hex, the board network created by the h-search algorithm, the game tree search algorithms, and the typical computer Hex algorithms that is the artificial player playing Hex are described.

2.1 Rules and Features

Hex is played on a rhombic board consisting of hexagonal cells. The game was developed for an $n \times m$ board (where n and m are natural numbers); however, an $n \times n$ board is generally used. John Nash advocated that 14×14 board is the optimal size. Figure 2.1a shows a 13×13 board. Two players have uniformly colored pieces(e.g., black and white). In this paper, black was assigned as the first player and white was the second player. The game proceeds with players placing their stones in turn on empty cells. The two opposing black sides of the board are


(a) A 13 × 13 Hex board

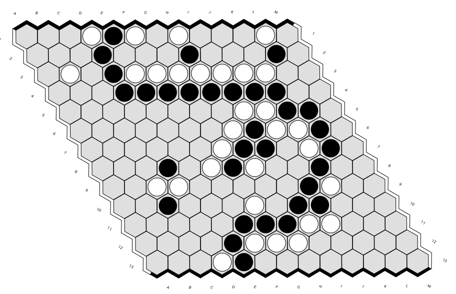

(b) Game board showing a winning configuration for white player

Figure 2.1: 13 × 13 Hex board. The top and bottom sides are assigned to the black player, and the left and right sides are assigned to the white player. The player who connects the assigned sides with their stones wins.

assigned to the black player, and the other opposing sides are assigned to the white player. The goal of the game is to connect the two opposing sides using the player's colored pieces. Figure 2.1b shows a winning configuration for a black player.

It has been shown that the game cannot end in a draw [Gal79], the game is a PSPACE-complete problem [ET76], and the first player has a winning strategy [Nas52]. The winning strategy is the concrete sequence of moves to win. In Hex, the game result is determined by the first move if the winner plays perfectly. However, it is difficult to solve the specific winning strategy because Hex has a large branching factor. The problem of finding the specific winning strategy of Hex is used as a testbed of the method for efficiently searching for a large search space. Currently, a specific winning strategy for all first moves has been demonstrated for 9 × 9 or smaller boards, and only one first move has been demonstrated even in 10 × 10 [PH13]. The details of above are described in the next section.

The international competition of the computer Hex algorithm, which is an artificial player playing Hex, has been held since 2000 [Ans00b, Hay06, AHH09b, AHH09a, AHH10a, Hay12, HAHP13, HPTvdV17, HWY$^+$]. The competition is called Computer Olympiad, and both 11 × 11 and 13 × 13 boards have been used in the recent Computer Olympiad [HW17]. The first computer Hex algorithm was developed by Claude Shannon and E.F. Moore in 1953 [Sha53], and many computer Hex algorithms have been developed since.

2.1.1 Swap Rule

It is known that the first player has an advantage in Hex. Generally, the first player has an advantage in the cells around the center of the board because it makes easy to connect the assigned sides with stones. To reduce the advantage of the first player, the swap rule is usually applied in an official game. The swap rule allows the second player to decide which color the second player plays after the first player plays the first move. For example, when the first player plays the first move advantageous to the first player, it would be better to use the swap rule for the second player. Because the second player becomes the first player, and the second player can continue the game with the first move advantageous to the second player. When the swap rule is applied to the game, the first move is often played near the sides.

2.2 Solving Hex

John Nash has proved that the first player has the winning strategy in Hex [Nas52], but it is difficult to find the specific winning sequences in large board size because Hex has the large branching factor. To efficiently find the winning strategy from the large search space, the search methods have been proposed. In this section, the methods of solving the winning strategy are described.

In 2002, Yang et al. solved some opening moves on 7×7 board by hand [YLP01, YLP03]. They proposed a decomposition method that combines the local games to find the winning strategy for a bigger local game. Small local games are easy to be solved, and it is possible to reduce the computational cost for solving the large local games by combining the small local game. Yang et al. defined over forty local games (patterns). In 2005, Noshita et al. proposed union-connections to solve the game simply in 7×7 [Nos04]. Their method simplified the previous proof and made it possible to find winning move-sequence for central move on 8×8 board by hand. Additionally, Mashima et al. improved the Noshita's method and found a winning sequence for a central move on 9×9 [MSN06].

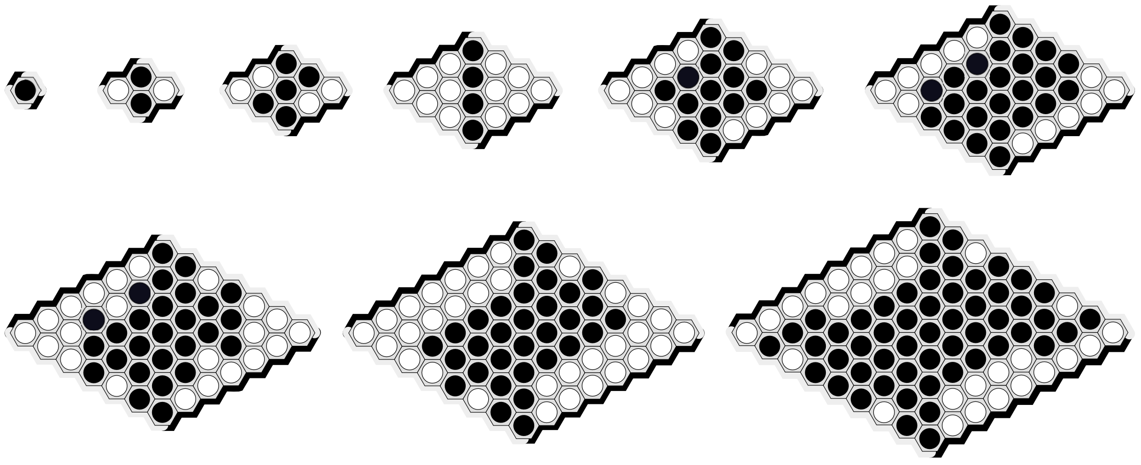


Figure 2.2: These diagrams show which player has the winning strategy for all opening moves on 9 × 9 or less. The black and white players have the winning strategy at the cells occupied by black and white, respectively. These diagrams are drawn with reference to these papers [HAH09, PH13].

The studies that use the computer to find the winning sequence have been also performed. Hayward et al. solved all 7 × 7 openings by using the computer in 2003 [HBJ+05]. They introduced an inferior cell analysis and H-search into the solver of the winning strategy to reduce the computational cost. The inferior cell is a cell whose other cells are better or whose the evaluation value does not change even if a stone is placed in the cell. In many cases, there is no problem that inferior cells are pruned from the search; hence, Hayward et al. prune the inferior cell to efficiently find the winning strategy. When Hayward et al. solved the 7 × 7 board in 2003, only one pattern of the inferior cell was used; however, a lot of inferior cells have been found until now. Björnsson has found new inferior cell patterns in 2007 [BHJvR07], and Henderson et al. solved all 8 × 8 openings by using 273 inferior patterns in 2009 [HAH09]. The solutions of less than 8 × 8 are summarized in [AHH11].

In 2013, all 9×9 openings have been solved by Pawlewicz and Hayward [PH13]. The major difference between the solvers of 8 × 8 and 9 × 9 is the tree search algorithm. The depth-first search was used to solve the 8 × 8 openings, but in 9 × 9, the search algorithm was changed from depth-first search to depth-first proof-number (DFPN) search in order to deal with the large search space. DFPN search was proposed by

Nagai et al. [NI02], and it is the method that changes the search algorithm of the pn-search proposed by Allis et al. [AvdMvdH94]. The pn-search is used in two-player and perfect-information game to solve the game quickly, and it uses and-or tree and best-first search. DFPN search uses the depth-first search instead of best-first search, and this change make it possible to search better than pn-search in terms of memory efficient. Pawlewicz et al. parallelized the DFPN search and proposed the Scalable Parallel DFPN Search (SPDFPN) that scales well. SPDFPN search could find all 9×9 openings and a 10×10 opening.

Figure 2.2 shows the opening moves with a winning strategy at the opening of 9×9 or less.

2.3 Board Network

The board states of the Hex board can be expressed as a network by treating cells as nodes and connecting adjacent nodes with a link (or edge). The board network is used for evaluating the position and the next candidate moves in Hex. In chess, shogi, and reversi, the positions and moves can be evaluated by using the values defined by the piece or cell position, For example, in chess and shogi, each piece have the different value (the king has the highest value in the pieces), and in reversi, each cell position has the different value (the corner cells have the highest values). On the other hand, in Hex, the stones have the same value, and the value of cell positions is not constant. Therefore, using the network characteristics calculated from the board network is effective to evaluate the position.

The board network consists of nodes and links. In the board network, the cell and the connecting adjacent cells are expressed as nodes and links, respectively. Figure 2.4 shows the board network of Figure 2.3. The board networks are created in such a way for each player. The nodes where the stone is placed are excluded from the board network. The links are connected between the nodes adjacent to the node where the own stone is placed because there is an adjacency relationship between those

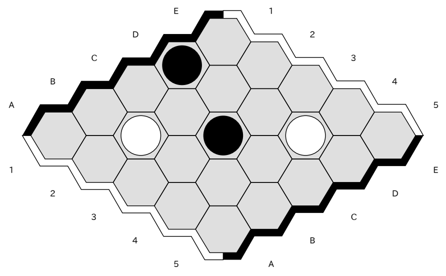
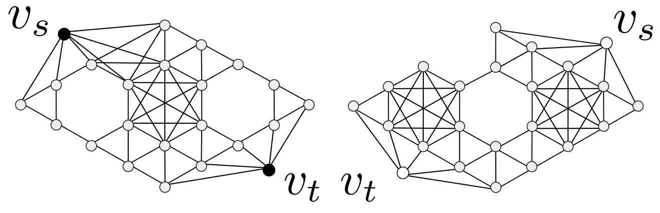


Figure 2.3: An example of 5 × 5 board



(a) Red's network (b) Blue's network

Figure 2.4: The board network of Figure 2.3. The cells and connecting adjacent nodes are treated as nodes and links (or edges), respectively.

nodes. Additionally, the links connected to the opponent stone are excluded. The nodes adjacent to the own side is assumed that there are the adjacency relationship; thus, the links are connected between the those nodes. The player having the link between two own sides wins the game, and this link is called *winning connection*.

In the following, I describe the board game using the graph similar to the board network, a special link including the future information, and H-search algorithm for finding the special link.

2.3.1 Shannon Switching Game

Shannon switching game is a board game for two-player invented by Claude Shannon, and this game is also known as the *Generallized Hex* [Gar61, AQS97]. The board of this game is a graph consisting of nodes and edges, and the graph board includes two special nodes. There are two players and the two players play in turn. One player is called *short player*, and the short player colors one of the edges included the graph by own color. The other player is called *cut player* and deletes one of the uncolored edges in the graph. The short player wins the game when the short player connects two special nodes using own colored edges, and the cut player wins the game when it is impossible for the short player to connect two special nodes. David Gale proposed a game that this game is played on a grid graph similar to the board network of Hex, and this game is known as *Grid* or *Bridge-it* [HvR06].

Shannon has developed a computer that plays Bridge-it, and the method which is used to develop the computer has made a big contribution to the development of computer Hex.

2.3.2 Virtual Connection

The *virtual connection* is a special link between two cells or groups that are not actually adjacent to each other [Ans00a, Ans02]. The virtual connection has the future information that is after playing several moves. It is possible to evaluate the position in consideration of future information by using the network characteristics calculated from the board network including the virtual connection.

There are two types of virtual connection as follows.

Virtual Connection

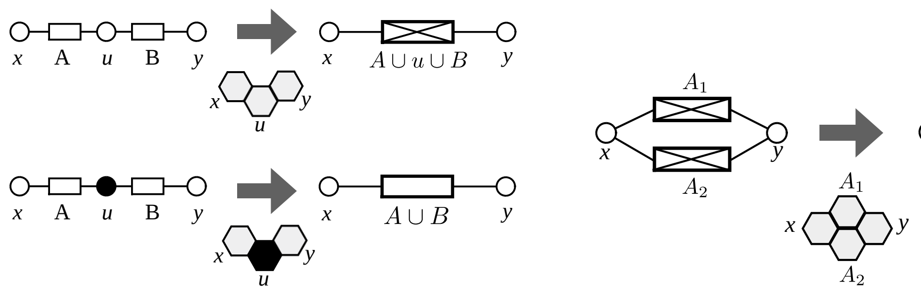
Virtual Connection (VC) is a link between two nodes or groups that is guaranteed to be able to connect after several turns if the player plays the best move. The player can connect two nodes or groups even if the player has the second move. VC is a link included to the board network.

Virtual Semi Connection

Virtual Semi Connection (VSC) is a link between two nodes or groups that is guaranteed to be able to connect after several turns if the player has the first move and plays the best move. The player must have the first player in order to connect the nodes having VSC. VSC is not included in the board network, and it is used for finding VC.

2.3.3 Two Rules for Creating Virtual Connection

VC and VSC are created by repeatedly applying following two rules. VC is defined as $VC(x, \mathbf{A}, y)$ (VSC is $VSC(x, \mathbf{A}, y)$). Both x and y are the nodes or the node sets



(a) AND Deduction Rule (b) OR Deduction Rule

Figure 2.5: Two deduction rules for creating virtual connections. x, y, and u are cells or cell groups, and $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{D}$ are node sets constituting VC or VSC. The square and cross-hatching square show VC and VSC, respectively.

which are connected by VC. $\mathbf{A}$ is a node set that is used for creating VC between x and y, and it is called *career nodes.*

AND Deduction Rule

If there are $VC(x, \mathbf{A}, u)$ and $VC(u, \mathbf{B}, y)$, in addition, $x \neq y$, $x \notin \mathbf{B}$, $y \notin \mathbf{A}$, and $\mathbf{A} \cap \mathbf{B} = \phi$ hold, the following is defined.

- $VC(x, \mathbf{A} \cup \mathbf{B}, y)$ exists if u is own cell or cell sets.
- $VSC(x, \mathbf{A} \cup u \cup \mathbf{B}, y)$ exists if u is empty node.

Figure 2.5a shows a simple example of AND deduction rule.

OR Deduction Rule

It assumes that there are $VSC(x, \mathbf{A_k}, y)(k = 1, 2, ..., n)$. When $\mathbf{A_k}$ is empty node and all following conditions are satisfied for all k, there is $VC(x, \mathbf{D}, y)$.

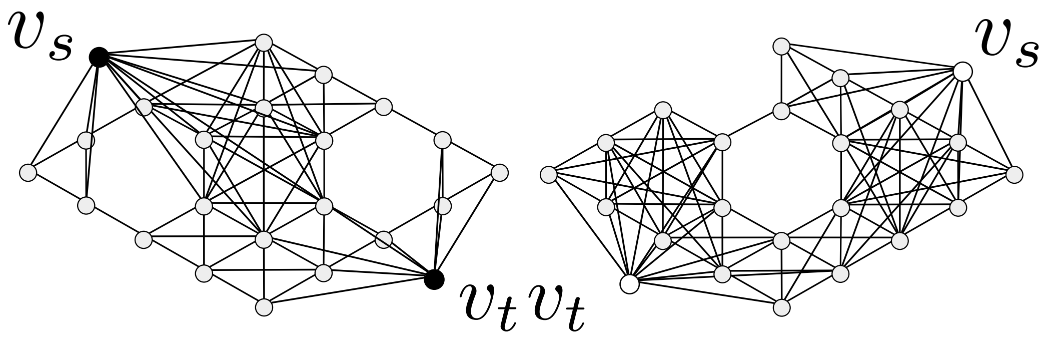
$$\begin{aligned}(x \cap \mathbf{A_k} &= \phi) \text{ and } (y \cap \mathbf{B_k} = \phi) \quad \text{(for all k = 1,2,...,n)} \\ \mathbf{A_k} \cap \mathbf{B_k} &= \phi \quad \text{(for all k = 1,2,...,n)} \\ \mathbf{C_k} &= \mathbf{A_k} \cup u_k \cup \mathbf{B_k} \\ \mathbf{D} &= \bigcup_{k=1}^{n} \mathbf{C_k}\end{aligned}$$

Figure 2.5b shows a simple example of OR deduction rule.

The algorithm that repeatedly applies above two rules is called H-search, and H-search can find many virtual connections [Ans00a]. However, it is known that there are virtual connections which can not be found by H-search. To find those virtual connections, XH-search which is the algorithm for finding more virtual connections has proposed in 2009 [HAH10]. XH-search can find more virtual connections than H-search by adding the Crossing Rule in addition to above two rules. A lot of virtual connections can be found by the XH-search, and part of the patterns that can create the virtual connection is published by King [Kin18]. However, the computational cost of XH-search is higher than H-search, and it takes a time to create the board network. Also, it is known that there are virtual connections which cannot be found by XH-search as with H-search. In order to reduce the time to find the virtual connections, FastVC-search has been proposed in 2015 [PHHA15]. FastVC-search can find the virtual connections quickly more than H-search. It is a trade-off between the number of found virtual connections and the computing cost. It is necessary to consider which is important, whether to find the more virtual connection or to find virtual connection quickly. For example, when monte carlo tree search is used as the game tree search algorithm, finding the virtual connection quickly is important because the number of simulations can be increased. On the other hand, in minimax tree search, it would be effective that finding more virtual connections even if the required time becomes long because it is expected that the evaluation accuracy increases.

2.3.4 The Algorithm for Creating Board Network

I describe how to create the board network including the virtual connections. I define cell i as node v_i, and connect it to adjacent nodes with a link. $\mathbf{V}$ is a set of nodes and $\mathbf{E}$ is a set of links. Function C is defined as the condition of nodes. $C(v_i) = 0$ if v_i is an empty cell, $C(v_i) = 1$ if v_i is occupied by Black, $C(v_i) = -1$ if occupied by White. Further, the two opposing sides belonging to each player are represented by nodes v_s and v_t. $C(v_s) = 1$ and $C(v_t) = 1$ for sides belonging to Black, and $C(v_s) = -1$ and $C(v_t) = -1$ for those belonging to White.



(a) $G^b_B(V_B, E_B)$ (b) $G^b_W(V_W, E_W)$

Figure 2.6: The board network with virtual connections of Figure 2.3

The process to create a board network $G^b_B(V_B, E_B)$ for Black is shown below. The board network for White $G^b_W(V_W, E_W)$ can be obtained in an analogous manner by replacing black with white.

1. Links $e(v_i, v_j)$ are added to $\mathbf{E}$ between all nodes adjacent to v_i and v_j.
2. Links $e(v_i, v_s)$ are added to $\mathbf{E}$ between all nodes v_i adjacent to v_s, and $e(v_j, v_t)$ is added to $\mathbf{E}$ between all nodes v_j adjacent to v_t.
3. Nodes belonging to White v_i $(C(v_i) = -1)$ are removed from $\mathbf{V}$, and links $e(v_i, v_j)$ belonging to the v_i are removed from $\mathbf{E}$.
4. The *H-search* algorithm and pattern matching are applied to the board network, and the VCs yielded are added to $\mathbf{E}$.

Figure 2.6 shows an example of the board network with the virtual connections.

2.4 Artificial Players

In this section, I describe the famous game tree search algorithms called minimax tree search and monte carlo tree search. In addition, I also introduce the major computer Hex algorithms that are artificial players playing Hex and the specific method of Hex used by many computer players. Almost computer Hex algorithms use game tree search algorithm to decide next move. The game tree is a directed graph that the position and move are the node and branch, respectively. Generally, the more the player performs a look-ahead, the better the player can select the move; therefore, the game tree search algorithm that can efficiently look ahead is required.

2.4.1 Minimax Tree Search

Minimax tree search is a method of searching the game tree to decide the move which minimizes the maximum loss for the searching player [KM75]. It is widely used in perfect information games such as chess, reversi, and shogi. To use the minimax tree search, the value function must be defined. In minimax tree search, first, the possible nodes and branches are created from the current node in the game tree. Then, the leaf nodes, which are terminal nodes of the game tree, are evaluated by the value function. Finally, the move with the best evaluation value for the player is selected as the next move. The strength of the player is decided mainly by the evaluation accuracy of the value function because the next move is decided according to the evaluation value output by the value function. Therefore, the evaluation accuracy of the value function is very important in minimax tree search.

Generally, the evaluation accuracy of the value function becomes high at the positions which are near the terminal position because predicting the game result at near the terminal position is easy. Therefore, the game tree is searched deeply. There are two typed of the interior nodes in the game tree. One is the node that the searching player has next play, and the other is the node that the opponent player has next play. The next move is the move with the highest evaluation value when the searching player has the next play. Whereas, the next move is the move with the lowest evaluation value when the opponent player has the next play because it is expected that the opponent player plays the worst move for the player. Minimax tree search selects the move which minimizes the maximum loss by alternately selecting the best move and the worst move for the searching player.

The computational cost is high when the search depth is deep because the number of positions to be evaluated increases. In order to reduce the time required for the search, alpha-beta tree search has been proposed [KM75]. The alpha-beta search is a type of minimax search that introduces the pruning methods called *alpha cutoff* and *beta cutoff*. Pruning is a method of deleting branches in the game tree, which makes it possible to reduce the number of positions to be evaluated by the value function. The alpha cutoff and beta cutoff are performed when it is theoretically found that the branch to be pruned is finally not selected even if the further search is performed. It is guaranteed that the search result of the alpha-beta search and pure minimax tree search is the same. This pruning is called backward pruning [Mar13].

The iterative deepening depth-first search (IDDFS) has been proposed as a search method that is more efficient than the alpha-beta search [Kor85]. IDDFS is a depth-first search that gradually increases the search depth. First, the shallow search is performed. Then, in a deep search, the positions are evaluated in order from the positions having good evaluation values output by the shallow search. Evaluating from the position with the high evaluation value increases the possibility of pruning and consequently speeds up the search.

Policy Function

Policy function scores the next candidate moves, which are branches in the game tree, and it is used to decide the order of evaluating the position by the value function. The value function is applied in order from the move with the high evaluation value of the policy function, and it increases the chances of the pruning. The policy function is also used for the forward pruning [Mar13]. Forward pruning is aggressive pruning, and the result of game tree search which is applied forward pruning is not guaranteed that it is same as the search result of pure minimax tree search. For example, tapered N-best search is the method of searching the top N moves of the evaluation value [GEC67]. Forward pruning can greatly reduce the number of positions to be evaluated, and it makes search more efficient and a deeper search. On the other hand, there is a risk of pruning good moves if the policy function has low evaluation accuracy. Therefore, the evaluation accuracy of the policy function is very important when forward pruning is applied to alpha-beta search.

2.4.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a game tree search algorithm that applies the Monte Carlo method to the game tree [BH04, Cou07b] The position is evaluated based on the random simulations, and the score of the position is winning percentage computed by a lot of simulations from the current position to the terminal position. The MCTS is effective in games where it is difficult to quantify the features of position, e.g., Go, because the value function is not required, unlike the minimax tree search. The position of Go is difficult to quantify the features, and the minimax tree search was not a good choice as the game tree search algorithm. The strength of the computer Go algorithm improved greatly with the appearance of the MCTS.

In the MCTS, it is possible to select a better move by performing a lot of simulations. However, it takes computational costs to perform many simulations. The simulations to the moves which are unlikely to be eventually selected are not necessary. Remi Coulom proposed a UCB1 applied to Trees (UCT) search that can select better move than the pure MCTS by reducing wasteful searching in 2007 [Cou07b]. The UCT search is the search algorithm that applies Upper Confidence Bound I (UCB1) to the MCTS. In the UCT search, the node to be simulated is decided by the current winning percentage and search frequency of the node. UCT search is possible to perform many simulations on the nodes with the high winning percentage, which is the promising position, by increasing the simulating probability of the node. Whereas, the number of simulations to the nodes with low winning percentage decrease because they are bad positions for the searching player. Additionally, the simulating probabilities of the nodes with the low search frequency increases in order to reduce the nodes whose have high/low winning percentage due to luck. UCT search is adopted not only in Go but also in many other games such as Hex and Amazons, and its effectiveness is shown [Lor08].

2.4.3 Computer Hex Algorithms

The computer Hex algorithm is an artificial player playing Hex. The computer Hex algorithms are usually developed by using Benzene framework [AHH12, You13]. Benzene framework is open source project and the some specific methods of Hex are implemented. I introduce the major computer Hex algorithms as follows.

EZO

EZO is computer Hex algorithm developed by Kei Takada et al. and uses the iterative deepening depth-first search as the game tree search algorithm. EZO has participated in all Computer Olympiad from 2013 to 2018, and the competition results are silver, bronze, silver, and silver in order from the oldest [HAHP13, HPTvdV17, HWY$^+$, HW17]. EZO used the value function focusing the local network relating the result of the game in 2013, and then, EZO used the value function which combines the global and local evaluations in 2015. The detail of above is explained in chapter 3. In 2016, EZO used the value and policy functions consisting of 12 network characteristics, and two functions were optimized by using the expert game records. EZO in 2016 and 2018 were called EZO-CNN and DeepEZO respectively, and they used the value and policy functions based on convolution neural networks. EZO-CNN and DeepEZO are described in detail in chapter 4.

Wolve

Wolve is the computer Hex algorithm that uses the iterative deepening depth-first search [Hen10]. Wolve uses shannon's electric circuit evaluation function as with Hexy, Six and Mongoose. This evaluation function is explained in Chapter 3. Wolve has participated in competitions from 2006 to 2011 [Hay06, AHH09b, AHH09a, AHH10a, Hay12]. Wolve won the gold medal in 2008 and the silver medal in the others. The differences between Wolve and other computer Hex algorithms using the same evaluation function are that Wolve introduces the new method for creating virtual connection and inferior cell analysis. Wolve is the strongest computer Hex algorithm in the players that uses shannon electric circuit model and alpha-beta search.

MoHex

MoHex uses Monte Carlo tree search as the game tree search algorithm [AHH10b]. MoHex was the world-champion program at the Computer Olympiad (2009 to 2011). MoHex has made a great success by pruning the move that does not need to be searched and improving the efficiency of the random simulation in the MCTS. In 2013, MoHex2.0, which has further developed MoHex, has proposed by Huang et al [HAH$^+$14]. The quality of MCTS simulation used by MoHex2.0 is improved by learning the board pattern using the expert game records. Additionally, the parameter related to the search was tuned by Confident Local Optimization (CLOP) [Cou12]. MoHex2.0 has been the world-champion program at the Computer Olympiad (2013 to 2017). MoHex-CNN is a computer Hex developed by Gao et al. in 2017 [GHM17]. Gao et al. proposed a move prediction model using Deep Convolutional Neural Network and combine the prediction model with MCTS search of MoHex2.0. MoHex-CNN outperformed MoHex2.0 and won the gold medal in 2017 tournament on 13×13 board. Additinally, MOHEX-3HNN have been proposed by Gao et al. in 2018 [GMH18]. MoHex-3HNN uses the neural network model called Three Head Neural Network (3HNN) and combines it with Policy Value MCTS. MoHex-3HNN won the international tourments in 2018.

2.4.4 Mustplay

Mustplay is the move which is the most obstructive move against the player with the winning connection [HBJ$^+$05]. This method is applied in many computer Hex algorithms. Figure 2.7 shows an example of the mustplay. When the white player has the next move at the position where the black player has some winning VSC, the white player must play the move which can erase the VSC as much as possible. The black player can win the game if the white player cannot erase all winning VSC of the black player because the black player can create winning VC after playing

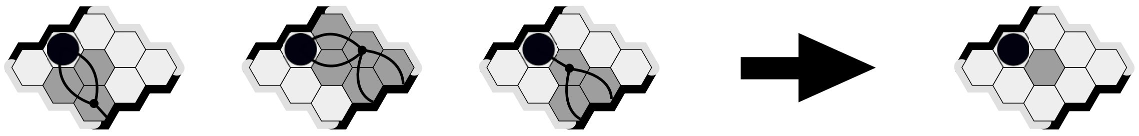


Figure 2.7: An example of *mustplay*. The left diagrams show the winning virtual semi-connection of the black player. The colored cells are carrier cells for building each virtual semi-connection. The right diagram shows the mustplay cell, which is the colored cell, for the white player. The mustplay cell is a common cell of carrier cells for the black player.

next move. If the white player erases all VSC of the black player by playing the mustplay, the game is continued.

2.5 Conclusion

In this chapter, I described the features and rules of Hex first. In Hex, it is proven that the game does not end in a draw and the first player has the winning strategy. A lot of studies for solving the specific winning sequence have been performed, and the winning strategy was proven by hand in early studies. The computer was also used as the board size became large. All 9×9 or less opening moves have been solved by using the DFPN search until now. In the process of solving the winning strategy, the H-search which creates the virtual connections and inferior cell analysis which finds the meaningless move have proposed. These methods improved the strength of the computer Hex algorithms.

The board state can be expressed as the board network by treating cells as nodes and connecting adjacent nodes as a link. The board network can be used to evaluate the positions and moves. The board network can include the future information by adding the virtual connections to the board network. Generally, the evaluation accuracy of the evaluation functions can improve when the board network with virtual connections is used for evaluating the position and moves. Almost computer Hex algorithms are based on the alpha-beta search or the MCTS. In alpha-beta tree search, it is important to develop the value and policy functions with high evaluation accuracy in order to develop the strong computer player.

Chapter 3

Application of Complex Network to Hex

3.1 Introduction

The value function is a very important function because the next move of the computer player is mainly decided based on the value function. In Hex, to define the value function, the board networks and the network characteristics calculated from the board networks have been used. The previous value function, called shannon's electric circuit resistance evaluation function, treats the board network as an electric circuit and uses the resistance value of the electric circuit to evaluate the position [Ans00a]. Although this value function evaluates the position from one perspective, it is shown that this value function has high evaluation accuracy. However, the computer Hex algorithm using this value function cannot win the player based on the MCTS in the recent tournament, and it means that the evaluation accuracy of the value function is insufficient [AHH09a].

I propose a novel value function consisted of the global and local evaluations using the network characteristics. The network characteristic is index proposed in the field of the complex network to quantify the feature of the network [SR07]. The global evaluation evaluates the overall strategy of the position, and the local evaluation evaluates the local strategy directly relating to win or lose of the game. It is expected that the evaluation accuracy is improved by evaluating the positions from two perspectives. I demonstrate that combining the global and local evaluations correctly is effective from the comparative experiments.

I also propose the method that creates the classifier of the board state, e.g., opening game and end game, by using the support vector machine (SVM) [CV95]. The classification of the board state is performed in many games because the strategy of the computer player can be changed according to the board state [ISR02]. The changing strategy allows the player to search the better moves according to the board state. The board state is usually classified based on the rule defined by hand in other game. However, the long experience and the deep knowledge are required to design the better rule, and it is difficult to handle exceptional situations. To flexibly classify the board state, I create the classifier by using SVM, which is known that the classifier created by SVM has high generalization ability. The board state is represented by 12 network characteristics, and the game records of exper are used to train the classifier by SVM. To demonstrate that the proposed classifier can classify the board state appropriately, I develop the computer Hex algorithm called EZO that uses the proposed value function and the classifier, and I compare it with the previous computer Hex algorithms. EZO can change the strategy by changing the ratio of global and local evaluations, and EZO dynamically changes the strategy according to the output of the classifier.

This chapter is organized as follows. First, the method to create two networks, the board network and shortest path network, for the global and local evaluations is described in Section 3.2. Next, I propose the novel value function using two evaluations in Section 3.3. To demonstrate that combining the global and local evaluations is effective, the comparison between MoHex and CH_{E_v}, which is a computer Hex algorithm that uses the proposed value function and alpha-beta tree search, is performed in Section 3.4. In Section 3.5, the classifier using SVM, which determines the ratio of the global and local evaluations, is created. Lastly, the computer Hex

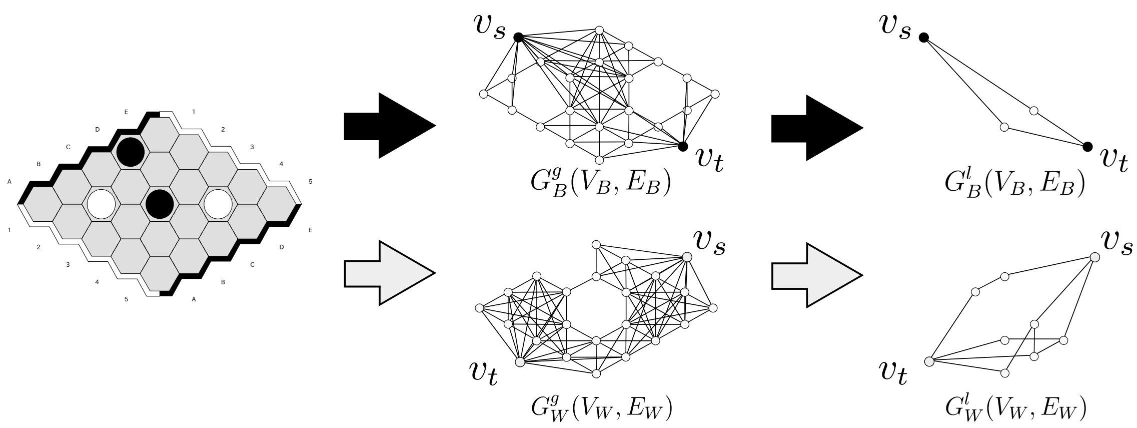


Figure 3.1: The left diagram shows an example of 5 × 5 position. The middle diagrams show the board networks $G^b(V, E)$. The right diagrams show the shortest path networks $G^l(V, E)$ which are the shortest paths between v_s and v_t.

algorithm EZO that uses the proposed value function and the trained classifier is developed, and the effectiveness of the proposed method is demonstrated through the comparison with other computer Hex algorithms in Section 3.6.

The using computer processor was Phenom Ⅱ X6 (6 cores, 2.9 GHz clocks) in this chapter. This chapter is described based on these my papers [THIY14, THIY15b, THIY15a].

3.2 Board Network and Shortest Path Network

To evaluate the position from the global and local evaluations, I use the board network and the shortest path network. The board networks, $G^b_B(V_B, E_B)$ and $G^b_W(V_W, E_B)$, are created by the method described in Section 2.3. **V** and **E** include all nodes and links on the board network, respectively. The board networks are used to evaluate the global strategy considering the whole board. The shortest path networks, $G^l_B(V'_B, E'_B)$ and $G^l_W(V'_W, E'_W)$, are created by nodes and links that are included in the shortest path between side nodes v_s and v_t. The shortest path networks are used to evaluate local strategy related to win or lose. Figure 3.1 shows an example of the board network and the shortest path network.

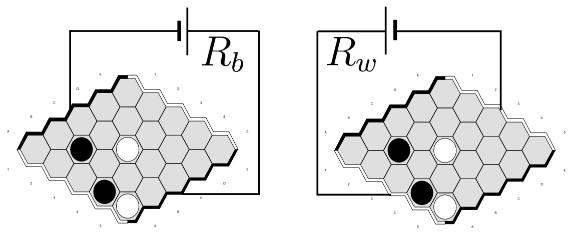


Figure 3.2: The electric circuit applied the voltage for the black and white players.

3.3 Value Functions in Hex

The objective of Hex is to connect two own sides with stones; therefore, the board should be evaluated by qualifying how easy it is possible to connect two sides. In this section, I describe the value function consisting of the global and local evaluations and the previous value function which is called shannon's electric circuit resistance evaluation function. The proposed and previous value functions qualify how easy the player connects to two sides from the variety of the paths and the electric resistance, respectively.

3.3.1 Shannon's Electric Circuit Resistance Value Function

The previous value function, called shannon's electric circuit resistance value function, was created by receiving the inspiration from the machine playing Hex developed by Claude Shannon and E.F. Moore [Ans00a]. The machine developed by Shannon played next move based on the electric resistance between two sides. In the electric resistance model, the board network can be regarded as the electric circuit, and the electric resistance is used to evaluates how easy the player connects two sides.

The voltage is applied to the two sides (see Figure 3.2), and the electric resistance for each player is computed from each electric circuit. The resistance $r(c)$ of the cell

c is defined as follows:

$$r(c) = \begin{cases} 1 & \text{if } c \text{ is empty} \\ 0 & \text{if } c \text{ is occupied by own color} \\ +\infty & \text{if } c \text{ is occupied by opponent color} \end{cases} \tag{3.1}$$

Additionally, the resistance between two cells (c_1 and c_2) are defined as follows:

$$r(c_1, c_2) = r(c_1) + r(c_2) \tag{3.2}$$

Finally, the electric resistance value function E is defined as follows:

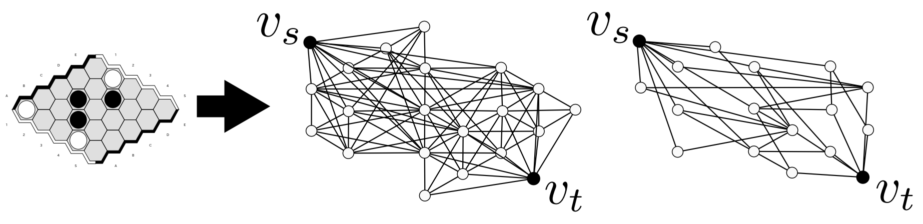
$$E = \frac{R_b}{R_w}, \tag{3.3}$$

where R_b and R_w is a total resistance of the black and white player, respectively. The black player is advantageous when E is small, and $E = 0$ when the black player has the winning connection. The white player is advantageous when E is large, and $E = +\infty$ when there is the winning connection of the white player.

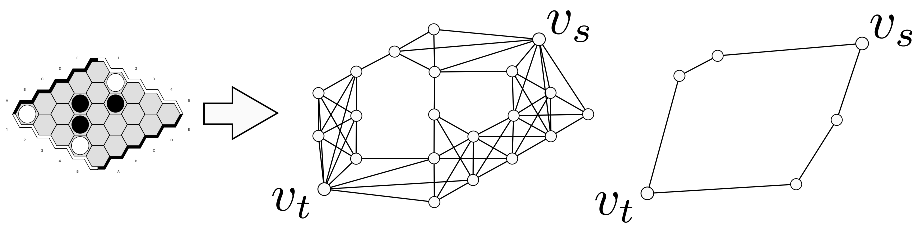
3.3.2 Proposed Value Function

The proposed value function consists of the global and local evaluations. The global evaluation evaluates the overall strategy of the position by using the board network, and the local evaluation evaluates the local strategy directly relating to win or lose of the game by using the shortest path network. The electric resistance model evaluates the position from only one perspective. However, the proposed value function can evaluate the position from two perspectives, and it is expected that the features of the position can be more accurately captured and the evaluation accuracy of the value function improves.

The global and local evaluations are evaluated by using the betweenness centrality, which is one of the network characteristics. The betweenness centrality is an index



(a) For black player



(b) For white player

Figure 3.3: These figures show an example of having high and low betweenness centrality. The board network and shortest path network of the black player have a low betweenness centrality. On the other hands, the two networks of the white player have a high betweenness centrality.

of how the node is contributed to the other shortest paths (see Section 3.3.3). In the previous study, I showed that betweenness centrality is a useful index for evaluating the position by using the expert records [THIY14]. The position where has the variety of the paths has the low betweenness centrality, and the position with low betweenness centrality is better. Figure 3.3 shows the board network and shortest path network for the black and white player in the same position, and they show concretely the relationship between the betweenness centrality and the variety of paths. In the position of Figure 3.3, the black player has an advantageous. In the board network of the black player, the average of the betweenness centrality is low because there are many nodes having direct links with other nodes, and there are variety of paths between two nodes. However, in the board network of the white player, the average of the betweenness centrality is high because a number of nodes are included in the shortest paths between other two nodes. It is also true for the shortest path networks of black and white players.

The total value function E_v consisted of global and local evaluations is defined as follows:

$$E_v = (1 - \alpha)\frac{C_W}{C_B} + \alpha\frac{C'_W}{C'_B}, \tag{3.4}$$

where C_B is the average of betweenness centralities of Black's board network $G^b_B(V_B, E_B)$, and C_W represents this for White's network $G^b_W(V_W, E_W)$. C'_B is the maximum value of betweenness centrality in the shortest path network $G^l_B(V'_B, E'_B)$ between v_s and v_t for Black, and C'_W is that for White. Only shortest paths between v_s and v_t are used to compute C'_B and C'_W in order to consider the paths directly related to win or lose. α is a constant parameter used to adjust the weight of the global and local evaluations. The black player has an advantageous when E_v is high, and the white player has an advantage when E_v is low.

3.3.3 Network Characteristics

Network characteristic is defined to quantify the feature of the network [SR07]. The network characteristics used in this study are described in this section. The centrality of the network is useful for quantifying the variety of paths, so the network characteristics related to the centrality are mainly used in this study.

Degree Centrality

A degree is the number of links connecting the node. The degree of node n is denoted $deg(n)$, and the degree centrality of node $D(n)$ is defined as:

$$D(n) = deg(n). \tag{3.5}$$

Generally, the node with a high degree is important in the network.

Betweenness Centrality

Betweenness centrality quantifies the number of times a node is included in the shortest paths between two other nodes. This concept has introduced by Freeman [Fre77]. The node with higher betweenness centrality can influence the shortest paths between the other two nodes and can be regarded as a highly influential node in the network. Betweenness centrality B_{v_i} of the node

v_i is defined as follow:

$$B_{v_i} = \frac{2}{(N-1)(N-2)} \sum_{k=1}^{N} \sum_{j=1}^{N} g(v_i, v_k, v_j), \tag{3.6}$$

where N is the number of nodes in the network and $g(v_i, v_k, v_j)$ is a function which calculates whether v_i is included in the shortest path network between v_k and v_j. $g(v_i, v_k, v_j) = 1$ for v_i is included, $g(v_i, v_k, v_j) = 0$ for v_i is not included, and $g(v_i, v_k, v_j) = 0$ if $v_i = v_k$, $v_j = v_k$ or $v_i = v_j$.

Closeness Centrality

Closeness centrality is the average length of the shortest path network between the node and all other nodes in the network. The node with lower closeness centrality is centrality node and close to other nodes. Closeness centrality C_{v_i} of the node v_i is defined as follows:

$$C_{v_i} = \frac{N-1}{\displaystyle\sum_{k=1}^{N} len(v_i, v_k)}, \tag{3.7}$$

where $len(v_i, v_k)$ is the length of the shortest path between v_i and v_k, and N is the number of nodes in the network.

3.4 Experiments for Proposed Value Function

In order to demonstrate that combining the global and local evaluations is effective, I developed the computer Hex algorithm CH_{E_v} using the proposed value function and compared CH_{E_v} to MoHex.

3.4.1 Experimental Conditions

The proposed computer Hex algorithm, CH_{E_v}, uses the proposed value function and α is a constant value through the games. CH_{E_v} uses 2-ply alpha-beta tree search, and the searching time using 2-ply is about 35 seconds. The policy function of CH_{E_v} consists of betweenness centrality. CH_{E_v} searches the moves in order from the nodes with the high value of betweenness centrality. MoHex is MCTS player and the reigning Computer Olympiad Hex gold medalist (see Section 2.4.3). MoHex was downloaded from Benzene project site and 2011 version [AHH12]. The search time of MoHex was restricted within 10 seconds per a move. CH_{E_v} was not implemented by using Benzene framework, and CH_{E_v} did not use mustplay and inferior cell analysis. On the other hands, MoHex used mustplay and inferior cell analysis.

The board size was 11×11 and the swap rule was not applied. The first move of the game is decided by the search of each player. The games are performed on 100 trials for each α and each player (the first and second player). The solver for Hex which is implemented in Benzene was used for game judgment. The solver is performed after every move for 10 seconds. The game is finished when the solver finds the winning player.

3.4.2 Experimental Result

Figure 3.4 shows the winning percentage of CH_{E_v} against MoHex for each α. The highest winning percentage is obtained when $\alpha = 0.075$ (79%) for the first player and $\alpha = 0.05$ (18%) for the second player. For the first player, the highest winning percentage ($\alpha = 0.075$) is much higher than the winning percentage of the method that uses only the global value function ($\alpha = 0.0$) and only the local value function ($\alpha = 1.0$). It means that combining the global and local evaluations improve the evaluation accuracy of the value function. It is also true for the second player although the difference is small.

The reason why the winning percentage of the first player is higher than the winning percentage of the second player is that swap rule is not applied and the first move of the game is decided from the search of each player. It is expected that the first move is played at the cell that the first player has an advantageous.

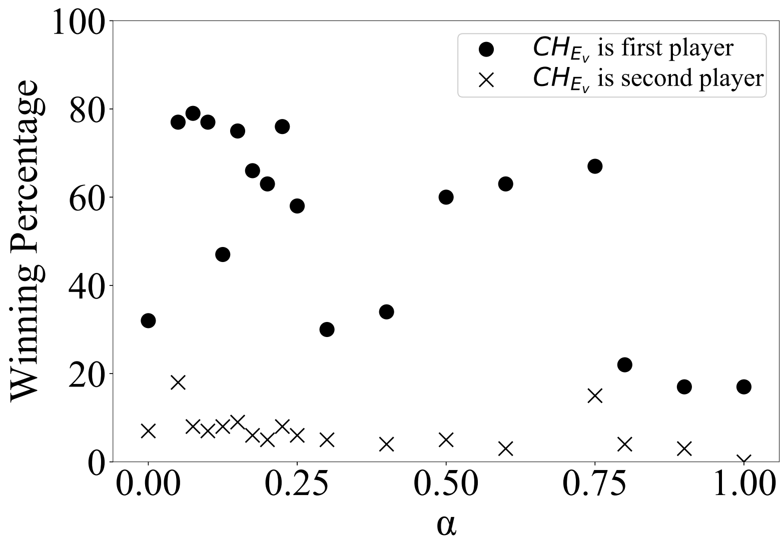


Figure 3.4: The winning percentage against MoHex for each α. 100 trials are performed for each parameter.

It is shown that the value function combining the global and the local evaluations is effective. However, the winning percentage of CH_{E_v} against MoHex is 79% of the first player and 18% of the second player, in other words, the winning percentage of MoHex against CH_{E_v} is 82% of the first player and 22% of the second player. Consequently, CH_{E_v} is weaker than MoHex.

In this experiment, the comparison between the proposed value function and previous value function was not performed. This is because the game result which is played by computer Hex algorithms using value function is uniquely decided; therefore, it is difficult to properly discuss the effectiveness of combining the two evaluations. The search result of the player using the value function is constant in the same position. Once a game is enough and more games are meaningless to compare those computer Hex algorithms. However, the search result of the player based on MCTS may change even in the same position because the search includes the randomness. I did not compare the proposed value function with previous value function because it would be appropriate that multiple games are performed to discuss the effectiveness of combining global and local evaluations.

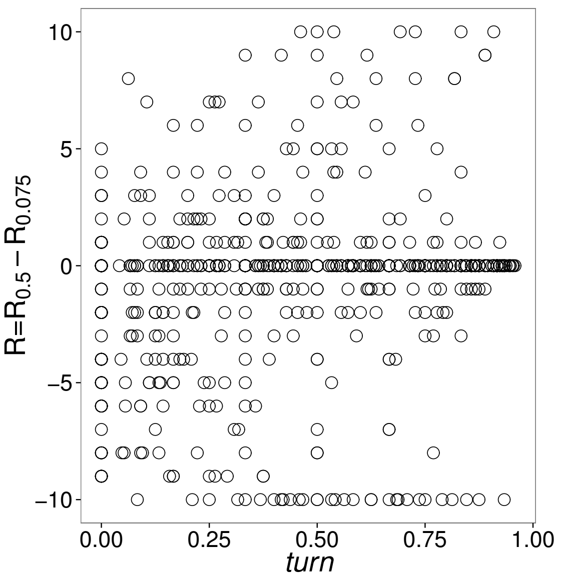
3.5 Changing Strategy According to Board State

To improve the strength of the proposed computer Hex algorithm, I propose a method that changes α in response to the positions. α decides the ratio of the global and local evaluations, and it is possible to change the influence of local evaluation by changing α. It is possible to play the move related to win or lose by increasing the local evaluation because the local evaluation considers the shortest path network between two sides. In Hex, it is expected that it is more effective to increase the influence of the local evaluation and play the move related to win or lose in the latter part of the games. This is because there is a possibility that the game result is decided with one move. In this section, I show that it is effective to increase α in the middle of the game. The timing of increasing α is decided by a classifier using Support Vector Machine (SVM) [CV95].

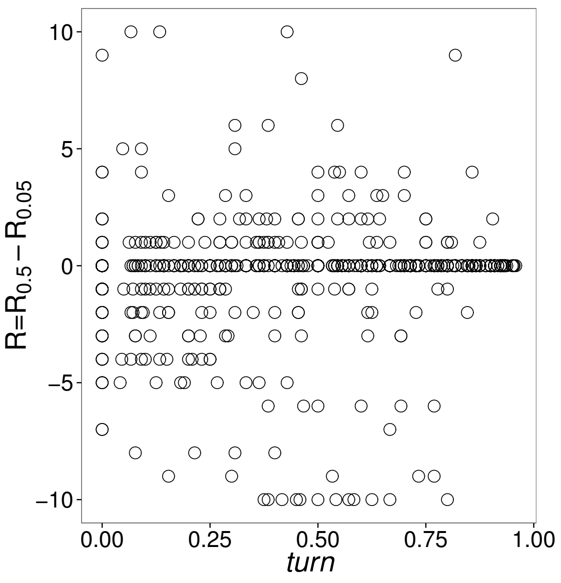
The method which changes the strategy according to the position is a popular method and often used in other games such as Go and Shogi [ISR02]. For example, the phases of the game in Shogi is roughly classified into three, i.e. the opening game, the middle game, and the endgame. The transitions of the phases can be clarified and described by rules to some extent in the long history of the studies. However, changing the strategy based on the rule needs to correspond to many situations, and it is difficult to design the rule appropriately. The originality of the proposed method is to change the strategy using SVM, which is a kind of machine learning, without using rules.

3.5.1 Effectiveness of Increasing Local Evaluation

To demonstrate that increasing α in the latter part of the games is effective, I compared the winning percentages of the value function with high and low α. The high α is set to 0.5 and low α is set to 0.075 for the first player and 0.05 for the second player. The last two values of α are decided from the experimental result in section 3.4.2



(a) CH_{E_v} is the first player



(b) CH_{E_v} is the second player

Figure 3.5: The difference of the number of wins between high and low α. The horizontal axis is normalized number of turn.

I used the same positions to compare the winning percentage of high and low α. The initial positions are given by the game records between CH_{E_v} with fixed low α and MoHex in section 3.4.2. In section 3.4.2, the games were performed 100 trials for the first and second players. There were 659 and 654 positions in the game records for the first and second player, respectively. By comparing the winning percentages, it becomes clear if α should be kept low or should be changed to high α after the given positions. In order to calculate the winning percentages, the games starting from the same positions were performed 10 times.

Figure 3.5 shows the differences between the number of wins of high and low α. The horizontal axis shows the normalized number of turns where 1.0 means the end of the games. The vertical axis shows the difference of the number of wins between high α (α=0.5) and low α (α=0.075 or α=0.05). $R_{0.5}$, $R_{0.05}$, and $R_{0.075}$ are the number of wins against MoHex when α is fixed to 0.5, 0.05, and 0.075, respectively. The meaning of high values of the vertical axis is that the winning percentages increase by changing to higher α after the given positions. When the value of vertical axis is 10, it means that only high α can win against MoHex and low α cannot win after the given positions. The results show that there are some cases where the α should be changed to high α in the latter part of the games.

It becomes clear that there are positions that increasing the influence of local evaluation is effective. However, it is not clear when to increase α. Additionally, the results show that it can be worse when α increases at the wrong timings. α should be changed at the correct timings. In the next section, I will describe a method for appropriately determining the timings where α increases.

3.5.2 Classifier based on SVM

I create a classifier that determines which of high and low α is appropriate for the give position by using SVM. SVM is a supervised learning model with the purpose of solving classification and regression problems. It is known that the model created by SVM has high generalization ability so that the margin with learning data is maximized. A feature vector of the data is generally used for the input to the SVM model. In this study, the feature vector of a position is input to SVM model, and SVM model is the binary classifier and determines whether α is high or low.

In order to classify positions appropriately, it is necessary to use the feature vector that reflects the features of the position in detail. The following 12 network characteristics (6 for each player) are used to capture the feature of the positions.

- the maximum, minimum, variance and average of betweenness centralities over all nodes of the board network $G^b(V, E)$.
- the maximum values of betweenness centrality in the shortest path network $G^l(V, E)$. The shortest paths between v_s and v_t are only considered to calculate the betweenness centrality.
- the shortest path length between v_s and v_t.

The averages of betweenness centrality capture the global strategies, and the others of betweenness centrality capture the biases of the strategy. The maximum values

Table 3.1: The number of positions of learning date and correct answer of SVM

	the number of position (the first/second player)	the number of position correct ansewer (the first/second player)
positive label	42 / 12	34 / 9 (88.1% / 75.0%)
negative label	617 / 642	617 / 642 (100.0% / 100.0%)

of betweenness centrality in the shortest path network capture the local strategies. The shortest path lengths estimate how close to win or lose.

The training data is prepared from the position used in section 3.5.1. The positions that satisfies R>4 in Figure 3.5 are positive labels (the positions where α should be high) and the others are negative labels (the positions where α should be low). For SVM, I use *R*, which is the environment for statistical computing, and "kernlab" library[R C15, KSHZ04]. Learning for SVM is performed by using the radial basis function (RBF) kernel, and the non-linear classifier is created. The learning parameters are determined by the grid search.

Table 3.1 shows the number of positions used for learning and the classification correct answer rate of the classifier for learning positions. For the first player, the number of learning positions is 42 for positive labels and 617 for negative labels. The classification correct answer rates are 88.1% and 100.0% for positive and negative labels, respectively. In addition, the number of learning positions for the second player is 12 for positive labels and 642 for negative labels. The classification correct answer rates are 75.0% and 100.0% for positive and negative labels, respectively. The classification correct answer rate for negative label is high; therefore, it can be said that there are few erroneous classifications for the position where α should not be increased.

3.6 Experiment for Classifier based on SVM

I demonstrate that it is effective to increase α in the game and that the trained classifier can increase α at the appropriate timing.

3.6.1 Experimental Conditions

Four computer Hex algorithms were used in the experiment. One was that the proposed computer Hex algorithm EZO. EZO used 2-ply alpha-beta search, classifier created by SVM in Section 3.5.2, and E_v as the value function. α started at 0.075 and 0.05 for the first and second player, respectively. The classifier was used at the initial positions of each search, and α was changed to 0.5 when the classifier outputs high α. After that, α was kept constant (α=0.5). Another computer Hex algorithm was the CH_{E_v} described in Section 3.4. α of CH_{E_v} were fixed at 0.075 and 0.05 for the first and second player, respectively. The others were Wolve and MoHex. The search times of Wolve and MoHex were 10 seconds per move. EZO and CH_{E_v} were not implemented in Benzene and did not use mustplay and inferior cell analysis. On the other hands, Wolve and MoHex used mustplay and inferior cell analysis. Each computer Hex algorithm used only one thread for the search.

The board size 11×11, and the swap rule was not applied. The game results were determined by using solver, and the solver was performed 10 seconds in each position. The game results between EZO, CH_{E_v}, and Wolve is determined uniquely; therefore, they were compared indirectly thorough MoHex. The number of games against MoHex was 100 for each computer Hex algorithm.

3.6.2 Experimental Results

Table 3.2 shows the game results of each computer Hex algorithm against MoHex. It shows that EZO got the higher winning percentage than the winning percentage of CH_{E_v} in both the first and second player. In other words, changing α in the game

Table 3.2: The winning percentages of each computer Hex program against MoHex for 100 trials (first/second player). ± represents standard error, 68% confidence.

	Win % vs. MoHex
EZO	92 ± 2.7 / 24 ± 4.2
CH_{E_v}	79 ± 4.1 / 18 ± 3.8
Wolve	82 ± 3.8 / 42 ± 4.9
MoHex	83 ± 3.7 / 17 ± 3.7

Table 3.3: The number of games when α is changed and not changed, and the game results

	the number of game (the first/second player)	Win	Lose
changed α	19 / 9	16 / 8 (84.2% / 88.9%)	3 / 1 (15.8% / 11.1%)
not changed α	81 / 91	76 / 16 (93.8% / 17.6%)	5 / 75 (6.2% / 82.4%)

is effective and better than fixed-α. In addition, the winning percentage of EZO is higher than the winning percentage of Wolve for the first player.

Table 3.3 shows the number of games when α is changed and not changed in 100 trials and the game results. The number of games that changed α was 19 for the first player and 9 for the second player. The winning percentage of the games that changed α is high, and it means that the classifier can increase α in position where it is effective to increase α. In addition, I pay attention to the game that α is not change in the game. The number of games that α was not changed is 76, and the winning percentage is 93.8%. This winning percentage is larger than 79.0%, which is the winning percentage of CH_{E_v} where also does not change α. This means that there were games that can win for increasing α even though the games were losing when α was fixed. From these results, it was shown that the developed classifier can determine the timing appropriately to increase α.

Confidence Intervals

The confidence intervals are estimated by the follows:

$$\pm z\sqrt{\frac{p(1-p)}{n}}, \tag{3.8}$$

where, p is the winning rate, n is the number of trials, and z is the $1 - \frac{1}{2}\alpha$ quantile of a standard normal distribution corresponding to the target error rate α . For example, $z = 1$ and $z = 1.96$ for the confidence error rates are 68% and 95%, respectively.

3.6.3 Discussion

The winning percentage of EZO for the first player is better than Wolve but is worse for the second player. According to their original paper [Hen10], they use the same value function for the first and the second player. I also use the same value function basically although the parameter is chosen for each player. Despite the fact that our method has been adjusted for the second player, the winning percentage of the second player cannot overtake Wolve. One possible reason is that the specific methods for Hex are implemented in Wolve and it might be effective for the second player. Wolve uses the mustplay and inferior cell analysis (see Section 2.2). These methods may be useful for the second player but not for the first player because the first player has the winning strategies and the second player starts game with a disadvantage. The winning percentage of EZO may be improved by implementing these methods.

3.7 Conclusion

In this chapter, I proposed a novel value function using network characteristics based on the global and local evaluations and a method to dynamically change strategy by using the classifier developed by SVM. The first experiment showed that it is effective to properly combine the global and local evaluations. In addition, it was showed that it is effective to increase the evaluation ratio of local evaluation in the latter part of the game by using the position appearing in the game records between CH_{E_v} and MoHex. The feature of position was expressed by 12 network characteristics, and the classifier that input is the position features is created to determine the evaluation ratio of global and local evaluations by using SVM. The computer Hex algorithm, EZO, was developed and used the proposed value function and the trained classifier. From the tournament using four computer Hex algorithms, it was showed that it is effective to dynamically change strategies by using the proposed classifier and proposed value function has the higher evaluation accuracy than the previous value function in terms of the first player.

In this chapter, I created the value function using network characteristics extracted by hand, and the board states are classified by only two groups. However, the number of board states cannot be defined in a real game because a lot of situations can be assumed. In addition, there may be the features of the position that the used network characteristics cannot express. In next chapter, I will describe the method using the CNN and the reinforcement learning algorithm to create the evaluation functions that evaluate the position more flexibly and accuracy.

Chapter 4

Learning Algorithm for Creating Value and Policy Functions using Convolutional Neural Network

4.1 Introduction

Traditionally, the evaluation functions have consisted of the features extracted by hand and their weights [HK14]. Also, in the games which are difficult to extract the features of position, e.g., Go, the evaluation value of the position is computed from the winning percentage based on the playout simulations. However, the computer player using these value functions could not win the game against the professional humans, which means that the evaluation accuracy of the evaluation functions was not sufficient. Recently, to create the highly accurate evaluation functions, the evaluation functions using a convolutional neural network (CNN) have been proposed, and it has been shown that the evaluation accuracy of functions that use CNN is greater than that of the traditional evaluation functions [Cou07a, MHSS15]. CNN is a feed-forward artificial neural network that can learn the features of images, and it has made better results in the field of computer vision [KSH12]. The evaluation functions using CNN treat the board position as the image, and CNN can learn the features of the position that are difficult to be expressed manually. It is expected that those evaluation functions can evaluate the positions and the moves based on learned features including the information of the board state.

The methods that create the policy function using CNN by the supervised learning have been proposed [CS15, DNW16]. The policy function is trained to imitate the moves of experts. It is shown that the policy function using CNN can imitate the expert moves more than the traditional policy function using the quantified features, and it means that CNN can learn the features of the position that are difficult to define by hand. However, supervised learning using the moves of experts is difficult to exceed experts, and it is necessary to prepare large amounts of data of experts. For the purpose of developing the evaluation functions with high evaluation accuracy, reinforcement learning algorithms are attracted attention.

In this chapter, I first propose the method that applies CNN in Hex, and then, I proposed a reinforcement learning algorithm using games of self-play to create value and policy functions using CNN in Hex. In order to make it easy to learn the features of positions in Hex, I propose the input of CNN focusing on three mutually adjacent cells. To demonstrate that the proposed CNN model can learn the features which are difficult to quantify by using board networks, I develop the policy function using the proposed CNN model and compare it with the linear policy function using the network characteristics. Two policy functions are trained by using the same expert's game records. In addition, the proposed policy function is compared with the previous CNN model to show that the proposed CNN model has high evaluation accuracy.

Next, I propose a novel reinforcement learning algorithm using game of self-play to create the value and policy functions using CNN in Hex. Many reinforcement learning algorithms have been developed in Backgammon, Chess, and so on [Tes95, Jas18, TKK12, BTW98, VSBU09]. It also has been demonstrated that employing reinforcement learning to train functions with CNN is effective for several board games, e.g., Hex and Go [TIY17, ATB17, SHM^{+}16]. Silver et al. proposed a reinforcement learning algorithm that creates the value and policy functions, and this algorithm has been extremely successful with Go (AlphaGo Zero [SSS+17]), Chess, and Shogi (AlphaZero algorithm [SHS+17]). In these methods, the value function is trained to predict the game result, and the policy function is trained to predict the search probabilities of each move output by the MCTS. To obtain the search probabilities, a lot of simulations are required, and the computational cost of simulations is high. Therefore, I proposed the novel reinforcement learning algorithm without using the search probabilities. The self-play player uses the minimax tree search of depth one with forward-pruning based on the moves determined by the policy function. The primary difference between the proposed method and the previous methods, e.g., AlphaGo Zero and AlphaZero algorithm, is the method to create the policy function, and the proposed algorithm does not use the search probabilities for learning. When the policy function does not use the search probabilities, it has to be trained using the state evaluation values of the evaluation functions, such as DDPG [LHP+15]. The policy always follows the value function. In the case of the game tree search, the policy function can be trained to predict the best moves of the search results by the value function. However, there are bad moves that have resulted in losses in those search results, and learning bad moves may decrease the ability to select the good move for the policy function. Therefore, in the proposed algorithm, the policy function is trained to predict the search result of the minimax tree search in the winner position in the case where the next player won, and to increase the number of moves to be searched in the loser position in the case where the next player lost. In order to demonstrate the effectiveness of the proposed algorithm, I compare the proposed algorithm with the learning algorithm having the policy function that estimates the value functions regardless of the winner or loser positions. Another difference with previous studies is that the value function is trained based on the depth one special case of the TreeStrap algorithm [VSBU09], which can be regarded as a type of Value Iteration [SGG+15]. I developed the computer Hex algorithm using the value and policy functions trained by the proposed algorithm, called DeepEZO, and compared it with world-champion programs in 2017 to demonstrate the performance of DeepEZO.

This chapter is organized as follows. First, the method to create the input of CNN is described in section 4.2. In section 4.3, the policy function using the proposed CNN model is developed, and it is compared with other policy functions in order to demonstrate that CNN model has high evaluation accuracy and can learn the features which are difficult to quantify by using board networks. In section 4.4 and later, the proposed reinforcement learning algorithm is described. I develop the computer Hex algorithm, DeepEZO, using the trained value and policy functions. DeepEZO is compared with the computer Hex algorithms using the evaluation functions trained by the different learning algorithm to demonstrate the effectiveness of the proposed learning algorithm. In addition, the tournaments using DeepEZO and world-champion programs in 2017 are performed to show that the trained evaluation functions have high evaluation accuracy.

This chapter is described based on these my papers [THIY18, TIY17, TIY]. The computer used in this chapter had an Intel(R) Core(TM) i7-7700KCPU (4.20GHz) and a Nvidia GTX 1080 GPU. This computer was used in all experiments. Also, the CNN models, value and policy functions, were developed using Torch [CKF11].

4.2 Representation of Board Position for CNN

In Hex strategy, it is important to consider cell adjacency because the goal of Hex is to connect two opposite sides. To make it easier to learn cell adjacency, I propose the input of CNN focusing on the three mutually adjacent cells [THIY18].

A cell can take three states corresponding to the placement of a player's stone, placement of the opponent's stone, and where no stone is placed. Therefore, the combined states of three adjacent cells yield 27 patterns. Here, the position is represented by 27 channels, and each pattern forms a channel. Figure 4.1 shows an example of creating inputs from a position at which the black player makes a move. In each channel, the number corresponding to a specific channel pattern becomes one, and the others remain zero. When the white player has the next move, I reflect the board in one of the diagonals and swap the black and white cells because the Hex

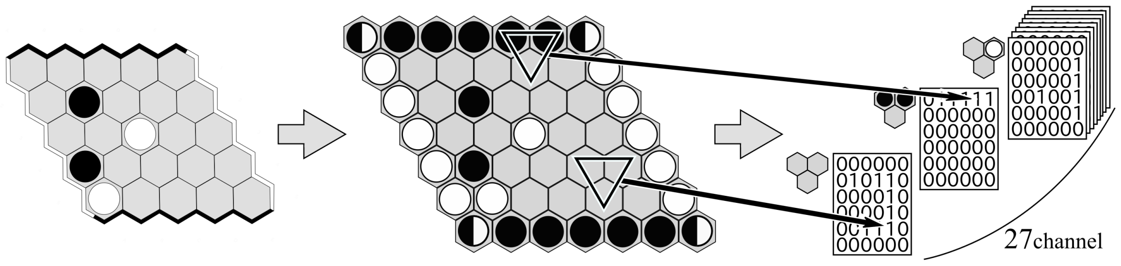


Figure 4.1: Creating position input. The left diagram shows positions on a 5 × 5 board. The middle diagram shows the sides in the left diagram with stones. Here, cells on corners are considered cells in which both black and white stones are placed. The right diagram is the input of the left diagram. Places corresponding to three mutually adjacent cells patterns become one.

board is symmetric. When the board is reflected, the left and right sides become the top and bottom sides, respectively. This configuration is obtained to eliminate the learning cost required to consider that the side to be connected by the player is different.

4.2.1 Other Representations of Board Position

Young et al. have proposed the NeuroHex which is a computer Hex algorithm [YVH17]. NeuroHex plays the next move based on the trained CNN model. Their CNN uses the input consisting of the simple channels expressing the position and the channels emphasizing the local information related to the game result. The number of total channels are six: the present black and white stones, black stone set connected to the edges of black (top and bottom), and white stone set connected to the edges of white (right and left).

Gao et al., 2017 have proposed the CNN model for evaluating the next move [GHM17]. The input features have nine binary channels. This input contains the information of cell (black, white, and empty), two edges (black and white), and simple *bridge pattern* which is the specific pattern in Hex. In addition, Gao et al., 2018 proposed the simple input consisted of four channels, which contain the basic board state information [GMH18].

4.3 Policy Function using Proposed CNN Model

I propose the policy function using CNN and compare it with the policy function $O(n)$ of the linear function based on the network characteristics in order to demonstrate that the proposed CNN model can learn the features of the position that are difficult to express in network characteristics and can obtain a high evaluation accuracy. In addition, the proposed CNN model is also compared with the previous CNN model.

In this section, first, the comparison between the proposed CNN model and the linear function $O(n)$ is performed. It is demonstrated that which model can more imitate the expert's move and which model is superior in the game tree search. Next, the proposed CNN model is compared with the previous CNN model called NeuroHex by using the computer Hex algorithms that decide the next move based on the CNN model.

4.3.1 Proposed CNN Model

The proposed CNN model outputs the evaluation value of all next candidate moves from the input of the current position. The input described in section 4.2 is used.

Table 4.1 shows the structure of the proposed CNN model. Convolutional layers 1 to 7 use 3×3 filters with stride and zero-pad of 1. Convolutional layer 8 uses 2×2 filters with the stride of 1. All activation functions in each convolutional layer are parametric rectified linear units(PReLU) [HZRS15]. The last convolutional layer has shared bias. Except for the first and last convolutional layers, all layers have 128 channels. Our network does not include the pooling layer because the cell location has very important meaning in Hex. In the output layer, I use the softmax function, and the output layer yields a probability distribution of all next candidate moves. I introduced a 20% SpatialDropout to the input layer to increase generalization ability [TGJ^{+}15].

Table 4.1: Network structure of proposed CNN model

Layer type	Image size	Channel	Kernel size
Input	14×14	27	–
Conv 1 - 7	14×14	128	3×3
Conv 8	13×13	128	2×2
Output	13×13	1	–

4.3.2 Linear Policy Function using Network Characteristics

The linear policy function $O(n)$ consists of 12 network characteristics (6 for each player) calculated for each node (cell) for evaluating node n, and it is defined as follows:

$$O(n) = \sum_{i=1}^{12} w_i e_i(n), \tag{4.1}$$

where $e_i(n)$ is the i-th network characteristic of node n, and w_i is the weight of the i-th network characteristic. The six network characteristics used are as follows: betweenness, closeness, degree centrality, the minimum of path length to sides, sum of betweenness centrality between two nodes adjacent to sides, and electrical resistance (see Section 3.3.3). Each network characteristic measures the importance of each cell in the board network from different perspectives. w_i is to be optimized by the optimization algorithm, and details will be described in Section 4.3.3.

4.3.3 Learning of Two Policy Functions

The proposed CNN model and linear model are trained to imitate the expert moves. The training data used for learning is same. The learning conditions, e.g., the objective function and the training data, are described in the bellow.

Objective Function of Proposed CNN Model

The proposed CNN model is trained by using a stochastic gradient descent to minimize the cross entropy objective function *Loss*, and *Loss* in the position-move pairs

(s, m) are defined as follows:

$$Loss = \frac{1}{N} \sum_{(s,m)\in\mathbf{S}} (-\log(out_s[m])), \tag{4.2}$$

where $\mathbf{S}$ is a set of position-move pairs; N are the number of position-move pairs; $out_s[m]$ is the output value (probability value) of node m at position s.

I used a mini-batch size of 32 and Adam optimizer [KB14]. The weights of the model are initialized to random weights.

Objective Function of Linear Policy Function

The weights of each network characteristics w_i are to be optimized by using Minimax Tree Optimization (MMTO) [HK14, Hok06]. MMTO was proposed for Shogi and has been shown to greatly improve the strength of computer Shogi. This method optimizes the weighted parameters in the evaluation function to make the expert moves the highest evaluated values among all candidate moves. MMTO optimizes the weight vector $\mathbf{w}$ to maximize the evaluation value of the expert move m at position s. The objective function $J_{MMTO}(\mathbf{S}, \mathbf{w})$ is minimized and defined as follows:

$$J_{MMTO}(\mathbf{S}, \mathbf{w}) = J(\mathbf{S}, \mathbf{w}) + J_C(\mathbf{w}) + J_R(\mathbf{w}), \tag{4.3}$$

where $J(\mathbf{S}, \mathbf{w})$ measures the degree of coincidence between the best move by $O(n)$ and the expert move; $J_C(\mathbf{w})$ and $J_R(\mathbf{w})$ are constraints and regularization terms, respectively; $\mathbf{S}$ is a set of positions-move pairs.

$J(\mathbf{S}, \mathbf{w})$ is defined as follows:

$$J(\mathbf{S}, \mathbf{w}) = \sum_{s\in\mathbf{S}} \sum_{n\in\mathbf{N_s}} T(h(s.m_s, \mathbf{w}) - h(s.n, \mathbf{w})), \tag{4.4}$$

where $\mathbf{N_s}$ is the set of candidate move sets at position s, excluding the expert move m_s. $s.m$ denote the placement of a stone from position s to move m, and $h(s.m, \mathbf{w})$ is the value of $O(m)$ with weight $\mathbf{w}$ at position s. T is a sigmoid function, and the gain is negative.

$J_C(\mathbf{w})$ is a constraint term, and $J_C(\mathbf{w}) = \lambda_0 g(\mathbf{w})$. λ_0 is a Lagrange multiplier, and $g(\mathbf{w}) = 0$ is the constraint condition. In this study, λ_0 is the median of $\{\frac{\partial J(\mathbf{S},\mathbf{w})}{\partial w_i}|w_i \in \mathbf{w}\}$. $J_R(\mathbf{w})$ is a regularization term, and I use l_2-regularization $J_R(\mathbf{w}) = \lambda_1|\mathbf{w}|^2$.

The weight vector $\mathbf{w}$ is updated using the following equation:

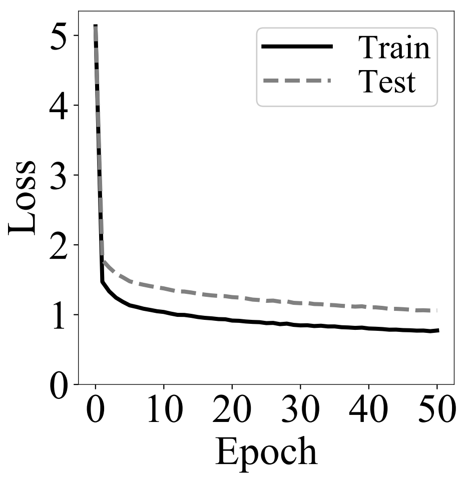
$$\mathbf{w_i}(t+1) = \mathbf{w_i}(t) - c \times sgn(\frac{\partial J(\mathbf{P}, \mathbf{v}(t))}{\partial \mathbf{v_i}}), \tag{4.5}$$

where $sgn(f(x))$ is a function that returns 1 for $f(x) > 0$, 0 for $f(x) = 0$, and -1 for $f(x) < 0$. The constant c is the learning rate, and it decreases gradually.

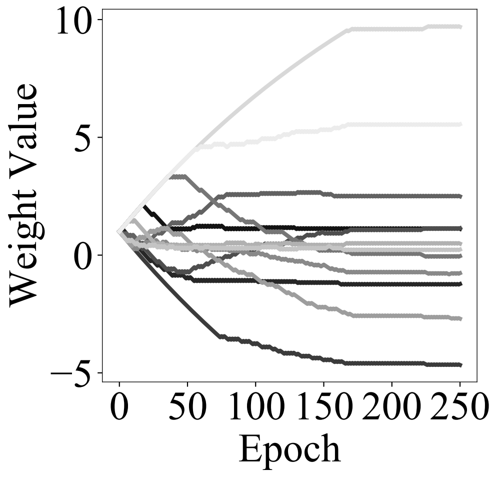
Position-Move Pairs Data for Learning

The position-move pairs (s, m) appearing in game records between the computer Hex algorithms were used to train the proposed CNN model and the linear model $O(n)$. The computer Hex algorithms used were EZO, and MoHex2.0, and Wolve. These programs are medalists in the Computer Olympiad, and it is expected that they can be expected as expert. The games were played between them with different search time and search width on a 13×13 board. The first move of the game was selected randomly from the all first moves. Also, in order to learn the better move, only the position-move pairs from which the winner plays the next move were used as training data. Moreover, if the position coincides with the original position after the board is reflected, I use the reflected position as well for training.

The training data of the proposed CNN model is slightly different from the data used for training the linear model $O(n)$. Some data is not used for training the linear model. In Hex, if a player has the winning connection, the player can win by playing a move optimally. If a winning connection exists, move evaluation is unnecessary because the cell where the next stone will be played is obvious. Hence, the positions of existing winning connections can be excluded from the training data for training the linear model. Notably, training is not disadvantageous for the linear model even



(a) The *Loss* value of train data (solid) and test data (dot) on each epoch



(b) Each weight of the linear model is converged

Figure 4.2: The learning results of the proposed CNN model and the linear model $O(n)$.

if the number of positions to be trained is small. This is because the excluded positions are not necessary to evaluate the move. Also, the network characteristics do not change even if the position is reflected; therefore, the reflected position is not used for training of linear model.

The numbers of training data are 1,915,103 and 475,421 for the proposed CNN model and the linear model, respectively. The positions included in the test data are the positions where the winning connection does not exist, and the test data used by proposed CNN model and the linear model are same. The number of test data is 83,872.

4.3.4 Learning Result

Figure 4.2a shows the translation of *Loss*, and Figure 4.2b shows that each weight of the linear model $O(n)$ is converged.

To clarify which model, the proposed CNN model and the linear model, can evaluate expert moves as the best move from all candidate moves, the two models are compared using common test data. Figure 4.3 shows the cumulative frequency dis-

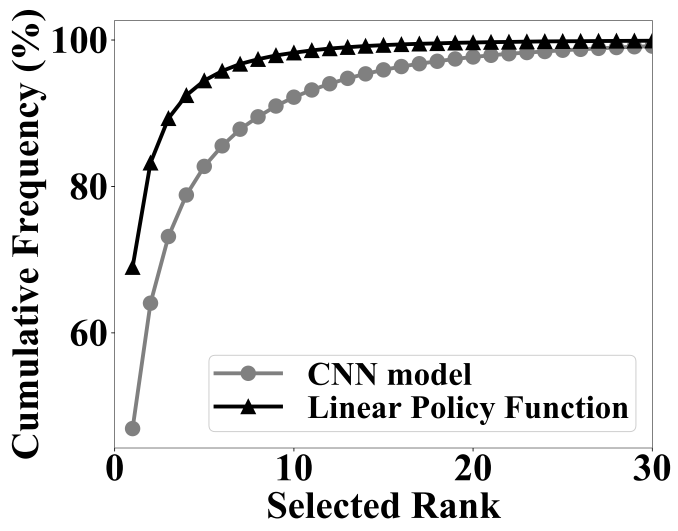


Figure 4.3: The cumulative frequency distribution. The meaning of 1 on the horizontal axis is that the model can make the expert move the highest evaluation value.

tribution of the evaluation value rank of expert moves, and it shows that proposed CNN model evaluates a greater number of expert moves higher than the linear model.

4.3.5 Comparison using Computer Hex Algorithms

It is shown that the proposed policy function using CNN can imitate the expert's move more than the linear policy function. However, the policy function is generally used to determine the search order in the game tree search. The ability required for the policy function is not to determine one good move but to decide the good search targets of game tree search. To demonstrate the proposed CNN model is superior in game tree search, the computer Hex algorithms using each policy function are developed and compared.

The computer Hex algorithm, EZO-CNN, using the proposed CNN model and the computer Hex algorithm, EZO-MMTO, using the linear model were compared to demonstrate that the proposed CNN model is high performance in game tree search. EZO-CNN and EZO-MMTO used the same evaluation function, and the difference between them was only the policy function. The evaluation function consists of the

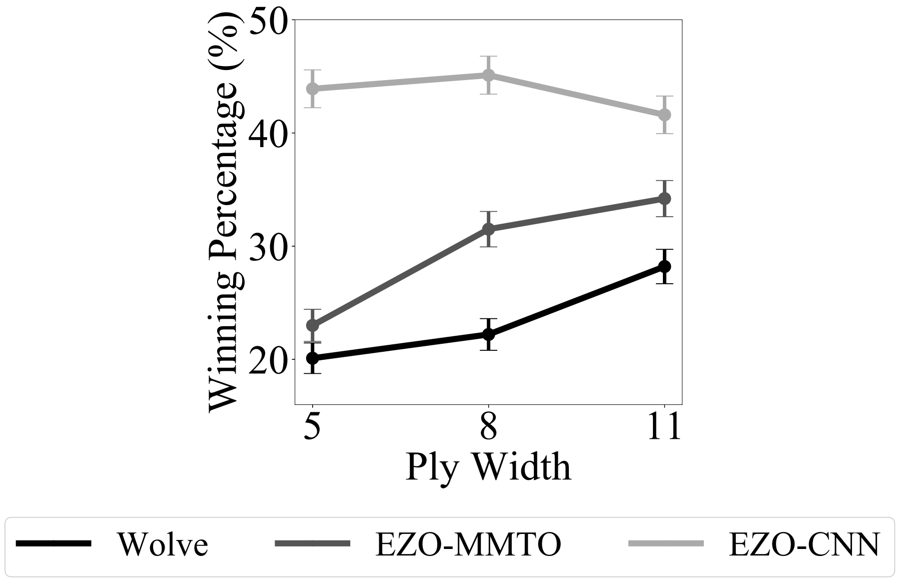


Figure 4.4: Winning percentage of the computer Hex algorithms against the of MoHex2.0 according to the search width of each computer Hex algorithm. The error bar shows standard error (68% confidence)

network characteristics used in the linear model, and the weights were optimized by MMTO. Wolve was also prepared for comparison with the previous method. The games between above three players are uniquely determined; therefore, it is difficult to evaluate the models from the direct comparison. The comparison was performed from the indirect games through MoHex2.0.

The board size was 13 × 13 board. The first player has an advantage in Hex, and this advantage can be reduced by starting the game from the side of the board. The game started at cells within two rows from the side of the board (a1-a13, b1-b13, l1-l13, m1-m13, c1-k1, c2-k2, c12-k12, and c13-k13). There are 88 first moves, and 10 games (five games for the first and second player) were performed from each first moves. The number of total games was 880. All computer Hex algorithms used the two threads for game tree search and the solver. EZO-CNN, EZO-MMTO, and Wolve used the 4-ply iterative deepening depth-first search as the game tree search algorithm, and the search widths at each depth were 5, 8, and 11. The search time per move of these three players was not limited, and MoHex2.0 search time per move was up to 10s.

Figure 4.4 shows the winning percentage of each computer Hex algorithm against

MoHex2.0. The average total search time in the game in the case of EZO-CNN is 247, 526, and 890s in the order of search widths of 5, 8, and 11. Similarly, in the case of EZO-MMTO, it is 212, 493, and 894s, and in the case of Wolve, it is 116, 261, and 472s. The search time of MoHex2.0 is approximately 205s. The results show that the winning percentages of EZO-CNN are higher than EZO-MMTO of the same search width. It means that the proposed CNN model is better than the linear model because the difference between EZO-CNN and EZO-MMTO is only the policy function. Additionally, EZO-CNN can get higher winning percentages than Wolve. However, the search time of EZO-CNN is longer than MoHex2.0, and the winning percentage of EZO-CNN against MoHex2.0 is low. This may mean that the evaluation accuracy of the evaluation function is low because the next move is determined based on the evaluation function.

4.3.6 Comparison of Proposed CNN model and Previous CNN model

In this section, to demonstrate that the proposed CNN model using the input focusing on three mutually cells has high evaluation accuracy, the comparative experiments are performed. The comparison is performed by using the computer Hex algorithms that decide the next move based on the CNN model.

NeuroHex

Young et al., 2016 have proposed the computer Hex algorithm, NeuroHex, using the move prediction model based on CNN to show that it is effective to use CNN in Hex and to use Q-learning for creating the model [YVH17]. NeuroHex decides the next move based on the output of the move prediction model without using game tree search. The model outputs the evaluation value of all next candidate moves from the current position input, and NeuroHex plays the move with the highest evaluation value. The model consists of 10-layer CNN and one fully connected output layer. The shape of the filter used for each CNN layer is unique and is hexagonal in order

Table 4.2: Winning percentage % of the proposed CNN model against NeuroHex.

	NeuroHex is second player	NeuroHex is first player
Proposed CNN model	81.7	79.9

Table 4.3: Indirect comparison of the proposed CNN model and NeuroHex through MoHex2.0. Each value shows the winning percentage % against MoHex2.0 (the first/second player).

	1s	3s	9s	30s
Proposed CNN model	38.5/24.3	35.5/16.0	27.2/17.2	21.3/10.7
NeuroHex	14.2/5.3	12.4/3.0	7.7/1.8	4.1/0.0

to capture the features of local stones. The input to the model is described in section 4.2

The model was trained by supervised learning and Q-learning. The model learned to output the electrical resistance by using a lot of positions, and then, the model is trained by Q-learning over two weeks. In experiments, NeuroHex was compared to MoHex2.0, and the results show that NeuroHex can win some games against MoHex2.0 if the search time of MoHex2.0 is less than nine seconds. However, NeuroHex cannot win the game when the search time of MoHex2.0 is 30 seconds. It was shown that the move prediction model can be created by using Q-learning, but the evaluation accuracy is not high.

Experiments

The comparison between NeuroHex and the proposed CNN model was performed. Each player decide the next move based on the output by the CNN model without the game tree search, and the next move is move with the highest evaluation value. Two models were compared from the direct and indirect games on the 13×13 board. In indirect comparison, the opponent player was MoHex2.0. The first move of the direct and indirect games were all opening moves, and it means that there were 169 moves for the first move. Each player plays as the first and second player from each first move, and the number of total game was 338 games.

Table 4.2 and 4.3 show the results of the direct and indirect comparisons. The proposed model got about 80% against NeuroHex and higher winning percentage than NeuroHex against MoHex2.0. In addition, the proposed model can get the winning percentage of 21.3% for the first player against MoHex2.0 (search for 30 seconds) without the game tree search. These results show that the evaluation accuracy of the proposed model is higher than the previous CNN model.

4.4 Proposed Reinforcement Learning Algorithm for Creating Value and Policy Functions

From previous experiments, it was showed that the proposed CNN model can learn features of position that are difficult to express in network characteristics. However, imitation learning using the moves of experts is difficult to exceed experts, and it is necessary to prepare large amounts of data of experts. To develop the evaluation functions with high evaluation accuracy, I proposed the reinforcement learning algorithm using game of self-play.

In this section, the proposed value and policy functions, the self-play player, and proposed learning algorithm are described. The value and policy functions are composed of the CNN, and they are trained using games of self-play by the self-play player with minimax tree search. The value function is trained to predict the game result, and the policy function is trained to appropriately determine the search targets while predicting the search result by the value function.

4.4.1 Proposed Value and Policy Functions

I explain the structure of the proposed value and policy functions using CNNs. The input using two functions is same, and it is described above (see Section 4.2).

Table 4.4: Network structure of proposed value function.

Layer type	**Image size**	**Channel**	**Kernel size**
Input	14×14	27	–
Conv 1	12×12	128	3×3
Conv 2	10×10	128	3×3
Conv 3	8×8	128	3×3
Conv 4	6×6	128	3×3
Conv 5	4×4	128	3×3
Conv 6	2×2	128	3×3
Conv 7	1×1	128	2×2
Full Connection	–	168	–
Output	–	1	–

Value Function

The proposed value function takes the board position as the input, and outputs a scalar evaluation value of the given position. Table 4.4 shows the network structure for the proposed value function. The value function consists of convolution layers and a fully connected layer, and all activation functions are rectified linear units (ReLU) [XG11]. The stride size is one for all convolutional layers. In the output layer, the output range is 0 to 1 because a sigmoid activation function is used. The value function aims to output 1 if the next player has a winning position, and 0 if the next player has a losing position.

Policy Function

The structure of the policy function is different from previous CNN model in Section 4.3.1. The proposed policy function takes the board position as its input and outputs the probability distribution of all next candidate moves. Table 4.5 shows the network structure for the proposed policy function. The policy function consists of only convolutional layers, and all activation functions in each convolutional layer are ReLUs. Convolutional layers 1 to 7 use 3×3 filters with a stride and zero-pad of one, and convolutional layer 8 uses 2×2 filters with a stride of one. We introduce a 10% SpatialDropout to the input layer to increase generalization ability [TGJ+15]. In

Table 4.5: Network structure of proposed policy function.

Layer type	**Image size**	**Channel**	**Kernel size**
Input	14×14	27	–
Conv 1 - 7	14×14	128	3×3
Conv 8	13×13	128	2×2
Output	13×13	1	–

the output layer, the probability distribution is output using the softmax function.

4.4.2 Self-Play Player with Proposed Selective Search

The self-play player uses a minimax tree search of depth one with forward-pruning. Here, the moves to be searched are determined by the proposed policy function. The output of the policy function is a probability distribution of next moves, and only moves with a high-probability value are searched. Specifically, moves for which the probability value exceeds $1/C$ are searched, where C is the number of cells on the board (e.g., $C = 169$ on a 13×13 board). Exceptionally, if the number of moves that exceed the threshold is less than three, the three highest moves are searched, i.e., the minimum search width is three. When there are no more than three empty cells at the given position, the player searches all empty cells. There are no specific methods available for determining the minimum search width. Here, it was determined by considering the balance between the risk of pruning good moves and the efficiency of the search carried out by reducing the number of moves to be searched. Where the selected moves are played from the root position, each position is evaluated by the proposed value function. The move with the highest evaluation value is selected as the next move.

4.4.3 Proposed Learning Algorithm

Algorithm 1 shows the proposed learning algorithm. Here, positions that have appeared in games of self-play are used to train the proposed functions. We define a position at the t-th turn as s_t, a move at s_t as a_t, and the evaluation value at s_t

Algorithm 1 Proposed learning algorithm.

```
1: function Learning
2:     for $epoch = 0$ to $E$ do
3:         Initialize training data set $\mathbf{D}$
4:         for $game = 0$ to $G$ do
5:             $p1$ uses the latest functions
6:             $p2$ uses the latest or past functions
7:             Randomly select which player is first
8:             Get random number $T$
9:             $t \leftarrow T$
10:            $s_t \leftarrow$ GetStartingPosition($p1$,$p2$,$T$)
11:            while not terminal $s_t$ do
12:                $p \leftarrow$ Next player selected from $p1$ and $p2$
13:                $p$ searches best move $a_t$ at $s_t$
14:                $(s_t, a_t)$ is added to $\mathbf{D}$
15:                $(s_t^{rotated}, a_t^{rotated})$ is added to $\mathbf{D}$
16:                $s_{t+1} \leftarrow$ Position playing $a_t$ at $s_t$
17:                $t \leftarrow t + 1$
18:            end while
19:            $(s_t,\emptyset)$ and $(s_t^{rotated},\emptyset)$ are added to $\mathbf{D}$
20:        end for
21:        Update the proposed functions using $\mathbf{D}$
22:    end for
23: end function
24: function GetStartingPosition($p1$,$p2$,$T$)
25:    $s_0 \leftarrow$ Initial position
26:    $s_1 \leftarrow$ Position playing the random move at $s_0$. The move is selected from
       the cells within two rows from the side.
27:    if $T$ is 1 then
28:        return $s_T$
29:    end if
30:    for $i = 1$ to $T - 1$ do
31:        $p \leftarrow$ Next player selected from $p1$ or $p2$
32:        $p$ searches best move $a_i$ at $s_i$
33:        $s_{i+1} \leftarrow$ Position playing $a_i$ at $s_i$
34:    end for
35:    $s_T \leftarrow$ Position playing random move at $s_{T-1}$
36:    return $s_T$
37: end function
```

as $v(s_t)$. In addition to the position-move pairs in games of self-play, the rotated position-move pairs $(s_t^{rotated}, a_t^{rotated})$ are used as training data because they are essentially the same as the original pairs. The reflection operation is similar to the method described in Section 4.2. The training data set $\mathbf{D}$ is initialized ($\mathbf{D}$ becomes an empty set) for each epoch because high-quality data are required to create highly accurate functions.

Because learning various positions leads to improved generalization ability for the proposed functions, the following techniques are introduced to games of self-play in order to increase the number of training positions.

Player Randomness

Two randomness techniques to a player's search are introduced, one of which is related to the moves to be searched. The player searches moves with a low-probability value. Moves with a probability value that is less than $1/C$ are searched with a probability of 20%. This is done to avoid searching only the currently favored moves. This operation is not performed to play the next move randomly, as with ϵ-greedy, but to determine the moves to be searched that are evaluated by the value function. The other technique is related to the evaluation value output by the value function. A small random number is added to the evaluation value. The value function has an output ranging from 0 to 1 because it uses the sigmoid function. The range of the random number to be added is -0.05 to 0.05. This range was determined empirically; however, by adding this random number, the player can approximate random play if they use an untrained instance of the proposed value function.

Random First Move

The first moves of games of self-play are selected randomly from cells within two rows from the side of the board. In Hex, first moves near the center of the board are advantageous for the first player. When starting a game near the center, it is expected that many favorable positions will appear for the first player. The value function must accurately evaluate positions where it is difficult to determine which player has an advantage. Learning from many difficult positions can be expected to lead to accurate evaluations by the proposed value function; thus, games of self-play begin at cells within two rows from the side of the board. The number of cells at the start of the game is 88 in a 13×13 board ($a1$ to $a13$, $b1$ to $b13$, $l1$ to $l13$, $m1$ to $m13$, $c1$ to $k1$, $c2$ to $k2$, $c12$ to $k12$ and $c13$ to $k13$).

Random Move During Games of Self-play

A random move is played during a game of self-play. The self-play players play a game over T moves until position s_T. Here, T is a random integer ($1 \leq T \leq 60$). The maximum value of T was determined empirically, and it is possible that the game will end within the maximum T value even if the player plays randomly. At position s_T, the next move a_T is played randomly from all empty cells. The self-play players play the game from position s_{T+1} after playing a_T at s_T until the game terminates.

Older Trained Functions

The most recent functions and older functions are used in games of self-play. The proposed functions are stored every 25 epochs. Here, an epoch is the period required to learn positions that have appeared in a given number of games of self-play. One player always uses the most recent value and policy functions. However, another player may use older functions. The functions to be used are selected randomly from the latest, second, or third latest functions, and the selection of these functions is performed randomly for each game.

Determined Move at Position Where Winning Connection Exists

At the position where a *winning connection* exists, the self-play players play move of *mustplay* (see Section 2.4.4). This is because the game tree search is not required in the position where the winning connection exists. At the position where a winning connection exists, the player with the winning connection plays the winning move to connect two opposite sides, and the other player plays the most obstructive move.

4.4.4 Loss Functions

The weights of the proposed functions are updated by the stochastic gradient descent method using the position-move pairs (s,a) in games of self-play. Here, I describe the loss function of each function.

Value Function

The proposed value function is trained by the depth one special case of the TreeStrap algorithm [VSBU09], and predicts the game's result. The reason for which the TreeStrap algorithm is used is that the method that is less sensitive to the strength of the player is necessary because the value function is initialized to random weights in the proposed algorithm. We give the final reward to only terminal positions. The loss function $Loss_{val}$ is minimized and defined as follows.

$$Loss_{val} = \sum_{(s,a) \in \mathbf{D}} loss_e(s,a), \tag{4.6}$$

$$loss_e(s,a) = \begin{cases} -\log(1 - v(s)) & (s \text{ is terminal}) \\ (v(s) - v(s'))^2 & (\text{otherwise}), \end{cases} \tag{4.7}$$

where $\mathbf{D}$ is a set of position-move pairs (s,a) in games of self-play, $v(s)$ is the evaluation value of the position output by the proposed value function at position s, and $s' = m(s,a)$ is the position at which move a was played at position s. In terminal positions, the player who lost the game has the next move. Thus, the proposed value function should output zero at the terminal positions in $\mathbf{D}$.

A log loss is used at terminal positions to evaluate the terminal position accurately. The error given to the value function at the terminal position is larger than that when using the squared temporal-difference error. The value function is trained more strictly to the terminal positions.

Policy Function

To determine the moves to be searched according to the positions, the loss function of the proposed policy function differs depending on whether the position is a "winner position" where the next player won or a "loser position" where the next player lost. If the training position is the winner position, the policy function learns to reduce the search width because the search width was sufficient. However, if the training position is a loser position, the policy function learns to increase the search width. The reason for increasing the search width in the loser position is to search for better moves that are pruned from the tree search. The policy function is used as the deciding search width for the player; therefore, one of the reasons for losing the game is that the policy function has a low evaluation accuracy, and prunes the better moves from the search. To play a good move in the loser position, it is necessary to search widely. By changing the learning policy according to the position, it becomes possible to determine the number of moves to be searched.

The proposed policy function is trained to minimize the following loss function $Loss_{policy}$:

$$Loss_{policy} = \sum_{(s,a)\in\mathbf{D}} loss_p(s,a), \tag{4.8}$$

$$loss_p(s,a) = \begin{cases} -\log \pi(a,s) & (s \text{ is winner position}) \\ -\lambda \sum_{m\in A} \pi(m,s) \log \pi(m,s) & (\text{otherwise}), \end{cases} \tag{4.9}$$

where A is a set of all moves on the board, $\pi(a,s)$ is the output of the proposed policy function for move a at position s, and $\pi(a,s)$ is the probability value. Here, λ is a constant parameter that controls the search width, and I used $\lambda = 10^{-1}$ in this study. As a result, I minimize the cross-entropy loss in the winner position and maximize the entropy loss in the loser position.

4.5 Training the Proposed Functions

I trained the value and policy functions using the proposed learning algorithm, and developed the computer Hex algorithm DeepEZO using the proposed functions. To demonstrate that the learning is properly performed by the proposed learning algorithm, I indicate the winning percentages of DeepEZO against the previous computer Hex algorithm MoHex2.0. The proposed algorithm is also compared with the learning algorithm, where the policy function predicts the moves selected by the search based on the value function in order to demonstrate the effectiveness of the proposed learning algorithm,

The only difference between the proposed algorithm and the learning algorithm to be compared is the update method of the policy function. In the learning algorithm to be compared, the policy function is trained to predict the search results in both winner and loser positions, which means that the loss function uses only cross entropy loss. In the proposed algorithm, the policy function is trained to predict the good moves in the winner position, and to increase the number of moves to be searched in the loser positions. The policy function is used when deciding the moves to be searched; therefore, if I assume that the highly accurate value function is obtained, there is no reason to be lost, except for the cases where the policy function prunes the winning moves, or when it is impossible to win the game. In this sense, the moves in the loser positions should not be entrained, and the function should increase the number of moves to be searched.

In this section, I first describe the computer Hex algorithm used in this section, and then the proposed functions are trained. Next, two policy functions are compared.

4.5.1 Computer Hex Algorithms

The computer Hex algorithms used in this section were implemented using the open-source Benzene framework [AHH12, You13]. All players are allowed to prune provably inferior moves and play the mustplay [HBJ$^+$05]. These methods exclude moves that are meaningless to the search, and it allows the player to play definitive moves to win or lose.

DeepEZO

DeepEZO is the proposed computer Hex algorithm and uses the iterative deepening depth-first search as the game tree search algorithm [Kor85], as well as the proposed value and proposed policy functions. The method employed to determine the moves to be searched is the same as that described in Section 4.4.2. There is not randomness in the search during the evaluation games. DeepEZO only searches moves with a high-probability value from the policy function (no other moves are searched). Here, the minimum search width is three.

I also prepared DeepEZO-Cross, where the policy function is obtained by the learning with only cross entropy. The evaluation functions used for DeepEZO-Cross are obtained by the learning, where only the cross entropy loss is used as the loss function of the policy function. The policy function is always trained to predict the search results regardless of the winner or loser positions. DeepEZO and DeepEZO-Cross use the same algorithms, except for the value and policy functions.

MoHex2.0

MoHex2.0 uses the MCTS as the game tree search algorithm [HAH$^+$14]. MoHex2.0 has been the world-champion program at the Computer Olympiad (2013 to 2017), and is the strongest computer Hex algorithm.

MoHex-CNN was the world-champion program for the 13×13 board at the 2017 [HW17, GHM17]. To demonstrate the effectiveness of the proposed functions, it is important to compare the proposed functions with the currently best program. However, MoHex-CNN is not currently available. Thus, MoHex2.0 was used in this study because it was the world-champion for the 11×11 in 2017, and is a sufficiently strong program.

In this paper, MoHex2.0 uses one thread for the search, and does not use pondering. This means that MoHex2.0 parameters “num_threads,” “lock_free,” and “ponder” are one, zero, and zero, respectively.

4.5.2 Experiments

The proposed functions were trained in reinforcement learning using self-play. I show that the learning is properly performed based on the winning percentages of DeepEZO against MoHex2.0. The effectiveness of the proposed training method is shown compared with that of DeepEZO-Cross in terms of the winning percentages against MoHex2.0.

In order to show that the trained policy function can prune bad moves and keep good moves properly, I indicate the difference in the winning percentages against MoHex2.0 between the two types of DeepEZO with a different search width (DeepEZO and DeepEZO-all). DeepEZO-all searches all next candidate moves at all positions. The search results of DeepEZO and DeepEZO-all are the same unless the policy function prunes the best move because they use the same value function. I also compared DeepEZO-Cross and DeepEZO-Cross-all, and the only difference between DeepEZO-Cross and DeepEZO-Cross-all is also the search width.

Game Conditions Between DeepEZO and MoHex2.0

The games between DeepEZO (DeepEZO-all, DeepEZO-Cross, and DeepEZO-Cross-all) and MoHex2.0 started at all opening cells in the board. There are 169 opening cells on a 13 $\times$ 13 board. One game was played for the first and second players for each opening. A total of 338 games were played. The search depth of DeepEZOs was two, and the search time of MoHex2.0 was up to 30s per move. These computer Hex algorithms used one thread for the search, and they did not use a parallel solver.

Training Conditions

The number of games of self-play in a single epoch G was 1,000, and the total number of epochs was 2,600, which means that 2.6 million games were played. The weights of the value and policy functions were initialized to random weights. I used a mini-batch size of 32 positions and Adam [KB14] as the optimizer to train the proposed functions.

Training Result

Figure 4.5 shows the winning percentages of DeepEZOs against MoHex2.0 in the training process. The average total search time in a game for DeepEZO, DeepEZO-all, DeepEZO-Cross, DeepEZO-Cross-all, and MoHex2.0 was 30s, 295s, 26s, 283s, and 505s, respectively. The training periods were approximately 45 days and 54 days for DeepEZO and DeepEZO-Cross, respectively.

Both DeepEZO and DeepEZO-all had winning percentages greater than 50% against MoHex2.0, and the search time of DeepEZO was less than that of MoHex2.0. These results show that the learning of the proposed algorithm was properly performed, and the evaluation accuracy of the evaluation functions was high. In addition, the results confirm that the winning percentages of DeepEZO and DeepEZO-all are not significantly different, and that the search time of DeepEZO is less than that of DeepEZO-all. It is expected that the policy function can appropriately determine the moves to be searched.

There is no significant difference between the DeepEZO and DeepEZO-Cross algorithms. This result means that both methods can create a policy function that can appropriately determine moves to be searched and the value function with high evaluation accuracy. However, in this experiment, two policy functions may not be compared properly because DeepEZO and DeepEZO-Cross mainly select the next move based on the value function.

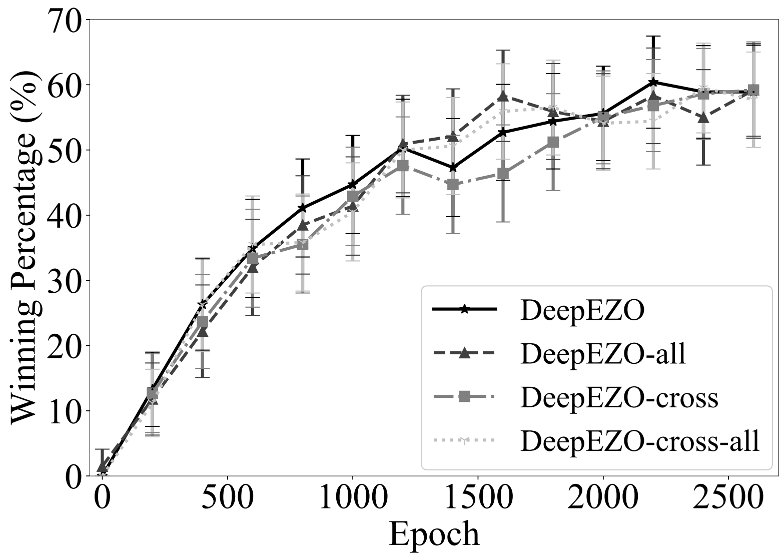


Figure 4.5: Winning percentages of DeepEZO and DeepEZO-all compared with MoHex2.0 in the training process (error bar shows the standard error of games; 95% confidence).

4.5.3 Analysis of Policy Function

I analyze the trained policy function in terms of the number of moves to be searched and direct match performances in order to demonstrate the difference between the trained policy functions.

Number of Moves to be Searched

To show that the proposed policy function can reduce the number of moves to be searched, I compared the number of moves searched by DeepEZO, DeepEZO-all, and DeepEZO-Cross at each position in games against MoHex2.0 in 2,600 epochs. The number of moves to be searched for DeepEZO-all is equal to the number of possible moves. Figure 4.6 shows the number of moves to be searched by DeepEZO, DeepEZO-Cross, and DeepEZO-all. The number of positions of DeepEZO, DeepEZO-Cross, and DeepEZO-all is 8,239, 8,516, and 16,381, respectively. All of the positions were categorized according to the number of turns, and the number of moves to be searched in each category was averaged. The results confirm that many moves were pruned from the game tree search by the proposed policy function, and

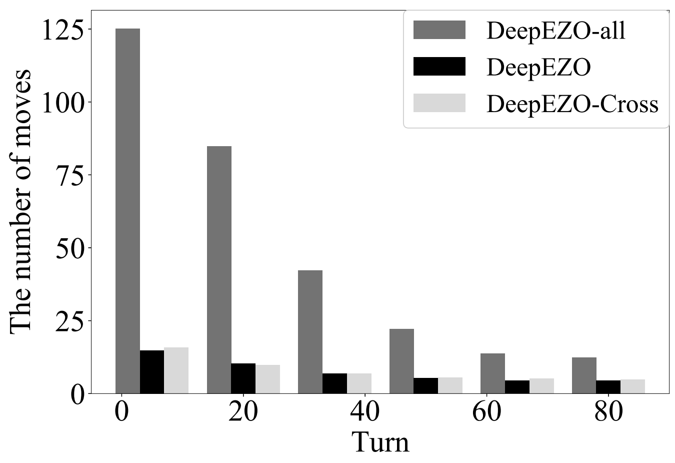


Figure 4.6: Number of moves to be searched at each position classified based on the number of turns from the positions in games against MoHex2.0 in 2,600 epochs.

there is no significant difference between DeepEZO and DeepEZO-Cross algorithms.

Game of Two Policy Functions

To demonstrate which policy function can select good moves, two policy functions were compared directly. I prepared two players who play the move with the highest evaluation value of the policy function, and two players play the games directly. P_{prop} is the player who uses the policy function created by the proposed learning algorithm, and P_{cross} is the player who uses the policy function created by the other learning algorithm. The game conditions are the same as those described in Section 4.5.2, with the exception of the players used.

Figure 4.7 shows the winning percentage of P_{prop} against P_{cross}. The results confirm that P_{prop} can realize a higher winning percentage against P_{cross}, which means that the evaluation accuracy of the policy function created by the proposed learning algorithm is higher than the policy function created by the learning in which the policy function is trained to always predict the search result.

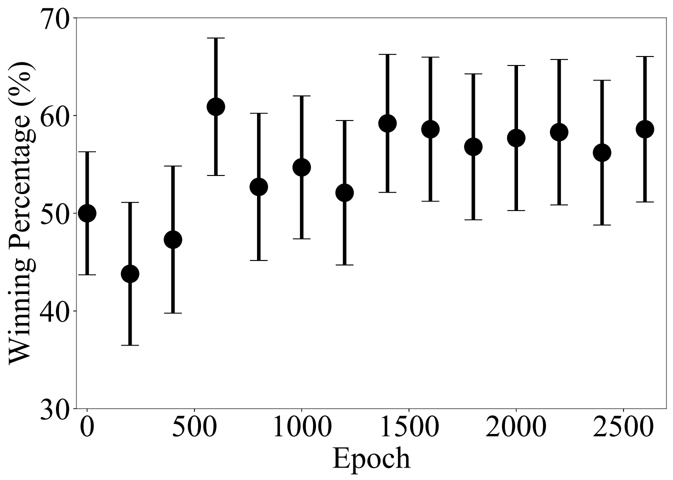


Figure 4.7: Winning percentage of P_{prop} against P_{cross} in the training process (error bar shows standard error of games; 95% confidence).

4.6 Experiment with Computer Hex Algorithms

Here, I discuss the performance of DeepEZO and DeepEZO-Cross when performing the deep search in order to demonstrate that DeepEZO, which uses the evaluation functions created by the proposed algorithm, is superior than DeepEZO-Cross. In Section 4.5.2, the proposed functions were trained, and it is shown that the policy function created by the proposed algorithm has a higher evaluation accuracy. In addition, there is no significant difference between DeepEZO and DeepEZO-Cross with a 2-ply search. There is no problem if the policy function can make the good moves the search target, even if the evaluation accuracy of the policy function is low because it is possible to finally select the good moves by performing the search with the value function. However, in a deep search, the policy function with a low evaluation accuracy may have the negative effect on the search result because the number of positions to be evaluated increases.

4.6.1 Computer Hex Algorithms

To demonstrate the performance of DeepEZO under the same search conditions, I also prepared Wolve and EZO-CNN, which are classical computer Hex algorithms.

EZO-CNN

EZO-CNN uses iterative deepening depth-first search, and placed second at the 2017 Computer Olympiad (described in section 4.3.5). EZO-CNN was the strongest computer Hex algorithm of the programs based on the minimax tree search. The value function employed by EZO-CNN is an optimized function consisting of 12 network characteristics. EZO-CNN uses a policy function with a CNN created by supervised learning. The search width of EZO-CNN is constant at each position, which means that moves are searched in descending order of the probability value of the move obtained by the policy function.

Wolve

Wolve uses iterative deepening depth-first search, and was the second-place computer Hex algorithm at the 2012 Computer Olympiad [Hen10]. The value and policy functions employed by Wolve are based on the electric resistance model, which is used by many other programs, such as Hexy and Six.

4.6.2 Game Conditions

The tournaments were played by DeepEZO, DeepEZO-Cross, MoHex2.0, EZO-CNN, and Wolve. The search depth of EZO-CNN and Wolve was four, and the search width was eight. The search time of MoHex2.0 was up to 30s per move. To ensure equal conditions as the existing computer Hex algorithms, I prepared two types of DeepEZO, i.e., DeepEZO-4ply, which searched at a depth of four and has no time limit, and DeepEZO-30s, which searches for 30s per move and has no depth limit. We also prepared DeepEZO-Cross-4ply and DeepEZO-Cross-30s. In total, seven computer Hex algorithms were used in the tournament.

The games started at all opening moves in the board. Two games were played for the first and second player for each opening, and 676 games were played in total. With the exception of the number of games, the experimental conditions were similar to those described in Section 4.5.2.

4.6.3 Tournament Results

Table 4.6 shows the tournament results. The elo score is computed by BayesElo [Cou10], and the standard error is about ±11 with 95% confidence. As can be seen, DeepEZO-30s has the highest elo score. To show the specific difference of the search between DeepEZO and DeepEZO-Cross, I indicate the number of positions to be evaluated and the search depth. The average number of positions to be evaluated in one position of DeepEZO-4ply and DeepEZO-Cross-4ply is 501.5 and 575.4, respectively. DeepEZO-Cross-4ply searches wider than DeepEZO-4ply, and this difference may be one of the reasons for which the average search time of DeepEZO-Cross-4ply is longer than that of DeepEZO-4ply. In addition, the average search depth in one position of DeepEZO-30s and DeepEZO-Cross-30s is 5.3 and 5.0, respectively. Further, the maximum search depth of DeepEZO-30s and DeepEZO-Cross-30s is 22 and 21, respectively. While the difference is small, it is shown that DeepEZO-30s searches deeply with the same search time.

DeepEZO-30s obtained a winning percentage of 79.3% against MoHex2.0 with the same search time. DeepEZO-4ply obtained a winning percentage of approximately 80.0% against programs based on the minimax tree search with the same search depth. DeepEZO obtained a very high winning percentage against world-champion programs, and it is obvious that the evaluation accuracy of the proposed evaluation functions is very high.

The search time of DeepEZO-30s was greater than that of MoHex2.0 even when the search time per move for each method was 30s because MoHex2.0 stops the search if the best move cannot change, even if it uses the remaining time. The average search time for each game differed; however, both programs required 30s per move.

Table 4.6: Winning Percentage % of each tournament (first/second player) for row player against column player and search time of each program (± is standard error; 95% confidence). Standard error of Elo is about ±11 with 95% confidence.

		A	**B**	**C**	**D**	**E**	**F**	**G**	**Search time in one game**	**Elo score**
A	**DeepEZO-30s**	-	54.1 ± 3.8 (65.1/43.2)	54.4 ± 3.8 (61.8/47.0)	52.8 ± 3.8 (62.4/43.5)	79.3 ± 3.1 (86.1/72.5)	84.6 ± 2.7 (93.5/75.7)	90.4 ± 2.2 (92.0/88.8)	645.3 [sec]	491
B	**DeepEZO-4ply**	-	-	52.1 ± 3.8 (63.0/41.1)	48.2 ± 3.8 (57.4/39.1)	75.3 ± 3.3 (80.5/70.1)	80.6 ± 3.0 (89.3/71.9)	86.2 ± 2.6 (88.2/84.3)	488.7 [sec]	450
C	**DeepEZO-Cross-30s**	-	-	-	52.1 ± 3.8 (64.2/39.9)	79.6 ± 3.0 (83.1/76.0)	84.5 ± 2.7 (89.3/79.6)	89.2 ± 2.3 (93.2/85.4)	655.4 [sec]	465
D	**DeepEZO-Cross-4ply**	-	-	-	-	77.9 ± 3.1 (82.2/73.6)	85.5 ± 2.7 (90.5/80.5)	88.2 ± 2.4 (90.5/85.8)	533.4 [sec]	464
E	**MoHex2.0**	-	-	-	-	-	69.1 ± 3.5 (77.8/60.4)	81.5 ± 2.9 (85.2/77.8)	499.3 [sec]	237
F	**EZO-CNN**	-	-	-	-	-	-	68.6 ± 3.5 (79.3/58.0)	619.7 [sec]	125
G	**Wolve**	-	-	-	-	-	-	-	327.5 [sec]	0

4.7 Discussion

The experimental results indicate that the alpha-beta search using the proposed value and policy functions created by the proposed learning algorithm perform more pruning than the alpha-beta search using the functions created by the learning method that trains the policy function to predict the search result in both winner and loser positions. Figure 4.6 shows that there is not much difference in the number of positions to be searched between two functions at each position. However, the number of positions to be evaluated for DeepEZO-4ply is smaller than that for DeepEZO-Cross-4ply. In addition, even within the same search time, DeepEZO-30s has a deeper search than DeepEZO-Cross-30s. These results indicate that DeepEZO-4ply performs more pruning than DeepEZO-Cross-4ply. It is believed that these results can be obtained because the policy function created by the proposed learning algorithm has a higher evaluation accuracy, as can be seen from Figure 4.7. These are among the reasons for which DeepEZO-30s has the highest elo score from among all computer Hex algorithms. The reason for which the evaluation accuracy of the policy function created by the proposed learning algorithm is high is that only the good moves are learned. The moves at the loser position may be poor moves because they eventually lead to losses. It is considered that learning only good moves consistently improved the evaluation accuracy of the policy function.

The primary differences between the proposed learning method and the existing methods (i.e., AlphaGo Zero and the AlphaZero algorithm [SSS^{+}17, SHS^{+}17]) are the game tree search algorithm and the method employed to create the policy function. In both methods, the policy function is created to increase the evaluation value of good moves and to reduce the value of bad moves. However, with the proposed method, the number of moves to be searched must be less than that of the existing methods. For each method, the number of moves to be searched depends on the evaluation value of the policy function, and moves with high values are more likely to be searched. The proposed method trains the policy function to increase the evaluation value of only the selected move in the winner position; thus, the number of moves to be searched tends to decrease at the winner position. However, the existing method trains the policy function to learn the search probability of each candidate move, and does not learn to increase the evaluation value of only the selected move. Of course, the number of moves to be searched will become small in the position where the winning move is obvious, but in the position where several actions are equally suitable, the number of moves to be searched will not become small. As a result, the proposed method may increase the risk of pruning the good moves by reducing the number of moves; however, there is a possibility that deeper searches and better move selection can be performed if the evaluation accuracy of the policy function is sufficiently high.

In terms of the training cost, the proposed learning algorithm may incur a lower cost than the AlphaZero algorithm. The AlphaZero algorithm requires the search probabilities of each candidate move to train the policy function. To obtain the search probabilities, many simulations are required, and this increases the computational cost. However, the proposed algorithm uses the search results instead of the search probabilities to train the policy function. The search result can be obtained from a 1-ply search, which means that each candidate move is sufficient to be evaluated only once. Because the proposed algorithm can reduce the number of evaluations and the computational cost, it may be possible to create the highly accurate evaluation functions more rapidly with the proposed learning algorithm.

We confirmed that the winning percentage of DeepEZO-30s against MoHex2.0 was greater than that of DeepEZO-4ply against MoHex2.0 (Table 4.6). This shows that the strength of DeepEZO-30s was greater than that of a limited-width 4-ply search, and that the generalization ability of the proposed value function is very high. The search result of the minimax tree search changes if the evaluation of a certain position changes; therefore, the search result will be a worse move if the value function evaluates a certain position incorrectly. Although the number of positions to be evaluated is increased by performing a deep search, DeepEZO-30s can play a better move by evaluating the position appropriately.

Learning for the proposed functions required 45 days; however, this can be reduced easily. Most of the learning time was spent on games of self-play; therefore, the learning time can be reduced by increasing the speed of games of self-play. Each game is completely independent, and parallelization can be implemented easily. In this study, I used only a single CPU and a single GPU, and I expect that the learning time can be reduced considerably by parallelization of additional computational resources.

4.8 Conclusion

In this chapter, I proposed the input of CNN in Hex and the reinforcement learning algorithm to create value and policy functions. The proposed input focuses on three mutually adjacent cells. To demonstrate the effectiveness of the proposed CNN model, I compared the proposed CNN model with the linear policy function using the network characteristics. The experimental results show that the proposed CNN model can imitate the expert moves more than the linear policy function and is superior as the policy function in the game tree search. It means that the proposed CNN model can extract the features that cannot be represented by network characteristics in Hex. In addition, the proposed CNN model was compared with the previous CNN model. Two models were compared from the direct and indirect comparison, and the results show that the proposed CNN model has higher evaluation accuracy.

Next, I proposed a reinforcement learning algorithm to create value and policy functions through games of self-play. The self-play player uses the minimax tree search of 1-ply. The value function was trained to predict the game result, and the policy function was trained to change the number of moves to be searched according to the given position. To demonstrate the effectiveness of the proposed learning algorithm, I compared the proposed learning algorithm with other learning algorithms. In addition, I developed the computer Hex algorithm DeepEZO using the minimax tree search, and compared it to other computer Hex algorithms. The experimental results show that DeepEZO outperforms all of the other programs, which means that the trained value function has a high evaluation accuracy and generalization ability, and that the trained policy function can appropriately determine moves to be searched at each position. I also demonstrated that the policy function trained by the proposed learning algorithm has a higher evaluation accuracy than the policy function that is trained to always predict the search result.

Chapter 5

Conclusion

In chapter 1, the background and objective of this thesis were described. I introduced the evaluation functions to be needed for developing the computer player and the methods to create the evaluation functions. I also described the difference between the proposed method and the previous method, and the originality of this thesis was explained.

In chapter 2, I described Hex which is a board game used in this thesis. I explained the H-search algorithm which is the method to create the board network, the study of solving the winning strategy, the game tree search algorithms, and the major computer Hex algorithms. The board state of Hex can be expressed as the board network by treating cells as nodes and connecting adjacent nodes with a link, and the board network is used for evaluating the positions and moves. Almost computer Hex algorithms are developed based on the minimax tree search or MCTS. There are specific methods of Hex, e.g., the inferior cell analysis and mustplay, to perform an efficient search.

In chapter 3, I described the method that applies the knowledge of the complex network to Hex and creates the classifier of the board state. I developed the value function consisted of the global and local evaluations using two board networks. The previous value function called shannon electric circuit model evaluates the position from one perspective; however, the proposed value function evaluates the position from two perspectives. By evaluating the position from two different perspectives, it is expected that the evaluation accuracy of the value function improves. I developed the computer Hex algorithm using the proposed value function and compared it with MoHex. The experimental results showed that combining the global and local evaluations properly is effective. Next, I created the classifier which determines the timing to change the strategy by using a support vector machine. The timing to change the strategy is determined by rule in the previous study; however, it is difficult to design the rule appropriately. It can be expected that more flexible classification of the position becomes possible by using SVM. To demonstrate the effectiveness of the proposed method, I developed the computer Hex algorithm using the proposed value function and classifier, and the proposed computer Hex algorithm can change the strategy by changing the ratio of the global and local evaluations. I performed the comparative experiments using the MoHex and Wolve, and the experimental results showed that the developed classifier can determine the timing appropriately to change the strategy.

In chapter 4, I described the method that applies the CNN to Hex and the reinforcement learning algorithm to create the value and policy functions from games of self-play. To apply the CNN to Hex, I proposed the input focusing on three mutually adjacent cells in order to make it easy to learn the features of positions in Hex. To demonstrate the effectiveness of the proposed CNN model, I developed the policy function using the proposed CNN model and compared it with the linear policy function consisted of 12 network characteristics. The proposed policy function could imitate the expert's move more than the liner policy function, and it was superior as the policy function in the game tree search, which means that CNN model can learn the features that are difficult to express in network characteristics and can evaluate the moves based on it. In addition, the direct and indirect comparisons were performed between the proposed policy function and the previous CNN model. The results indicated that the proposed policy function has high evaluation accuracy. Then, I proposed the reinforcement learning algorithm to create the value and policy function from games of self-play. The primary difference between the previous algorithm and the proposed algorithm is how to train the policy function. The previous algorithm uses the search probabilities of each move output by MCTS to train the policy function; however, the proposed algorithm uses the search result of the game tree search instead of the search probabilities. To demonstrate the effectiveness of the proposed learning algorithm, I developed the computer Hex algorithm DeepEZO using the trained evaluation functions and compared it with the previous computer Hex algorithms. In addition, the proposed algorithm was compared with the learning algorithm that the learning method of the policy function is different. The result of the comparative experiments shows that DeepEZO outperformed all of the other programs, and it is demonstrated that the proposed algorithm can create the highly accurate value and policy functions.

Finally, this thesis is summarized with some remarks, and some directions toward the future work are described. I proposed the learning algorithm to create evaluation functions necessary for the development of the computer player. In particular, the reinforcement learning algorithm proposed in chapter 4 is a versatile learning algorithm, and it can be applied to other games. It can be expected that the computational cost necessary for the learning is less than the previous algorithms; therefore, it may be effective to apply the proposed method in games with many candidate moves. In this thesis, I demonstrated the effectiveness of the proposed method by using Hex. In Hex, it is shown that the first player has the winning strategy, and the solver has been developed actively. The comparison between the proposed computer player and the solver is interesting and useful in discussing the effectiveness of the proposed method. Development of a learning algorithm that can create perfect players is a very challenging problem.

Bibliography

[AHH09a] Broderick Arneson, Ryan Hayward, and Philip Henderson. Mohex wins hex tournament. In *ICGA Journal*, volume 32, pages 114–116, June 2009.

[AHH09b] Broderick Arneson, Ryan Hayward, and Philip Henderson. Wolve 2008 wins hex tournament. In *ICGA Journal*, volume 32, pages 49–53, March 2009.

[AHH10a] Broderick Arneson, Ryan Hayward, and Philip Henderson. Mohex wins hex tournament. In *ICGA Journal*, volume 33, pages 181–186, September 2010.

[AHH10b] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Monte carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258, December 2010.

[AHH11] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Solving hex: Beyond humans. In *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2011.

[AHH12] Broderick Arneson, Philip Thomas Henderson, and Ryan B. Hayward. Benzene. http://benzene.sourceforge.net/, 2009-2012.

[Ans00a] Vadim V. Anshelevich. The game of hex: An automatic theorem proving approach to game programming. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and*

Twelfth Conference on Innovative Applications of Artificial Intelligence, pages 189–194. AAAI Press, 2000.

[Ans00b] Vadim V. Anshelevich. Hexy wins hex tournament. In *ICGA Journal*, volume 23, pages 181–184, 2000.

[Ans02] Vadim V. Anshelevich. A hierarchical approach to computer hex. *Artificial Intelligence*, 134(1):101–120, 2002.

[AQS97] A.A. Arratia-Quesada and I.A. Stewart. Generalized hex and logical characterizations of polynomial space. *Information Processing Letters*, 63(3):147 – 152, 1997.

[ATB17] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, pages 5366–5376, December 2017.

[AvdMvdH94] L.Victor Allis, Maarten van der Meulen, and H.Jaap van den Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–124, March 1994.

[BH04] B. Bouzy and B. Helmstetter. *Monte-Carlo Go Developments*, pages 159–174. Springer US, 2004.

[BHJvR07] Yngvi Björnsson, Ryan Hayward, Michael Johanson, and Jack van Rijswijck. *Dead Cell Analysis in Hex and the Shannon Game*, pages 45–59. Birkhäuser Basel, 2007.

[Bro00] Cameron Browne. *Hex Strategy: Making the Right Connections.* A. K. Peters, Natick, MA, 2000.

[BTW98] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. TDLeaf(λ): Combining temporal difference learning with game-tree search. In *Australian Journal of Intelligent Information Processing Systems*, volume 5, pages 39–43, 1998.

[CHhH02] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57 – 83, 2002.

[CKF11] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[Cou07a] Rémi Coulom. Computing elo ratings of move patterns in the game of go. In *ICGA Journal*, volume 30, pages 198–208, 2007.

[Cou07b] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin Heidelberg, 2007.

[Cou10] Rémi Coulom. Bayesian elo rating. https://www.remi-coulom.fr/Bayesian-Elo/, 2010.

[Cou12] Rémi Coulom. CLOP: Confident local optimization for noisyblack-box parameter tuning. In *Advances in Computer Games*, pages 146–157. Springer Berlin Heidelberg, 2012.

[CS15] Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play go. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, volume 37 of *ICML'15*, pages 1766–1774. JMLR.org, 2015.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[DNW16] Eli David, Nathan Netanyahu, and Lior Wolf. Deepchess: End-to-end deep neural network for automatic learning in chess. In *Artificial Neural Networks and Machine Learning*, pages 88–96, 09 2016.

[ET76] Shimon Even and Robert Endre Tarjan. A combinatorial problem which is complete in polynomial space. *Joutnal of ACM*, 23(4):710–719, October 1976.

[Fre77] Linton Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 03 1977.

[Gal79] David Gale. The game of hex and the brouwer fixed-point theorem. *The American Mathematical Monthly*, 86(10):818–827, 1979.

[Gar61] Martin Gardner. *The Second Scientific American Book of Mathematical Puzzles and Diversions*, chapter 7, pages 78–88. New York: Simon and Schuster, 1961.

[GEC67] Richard D. Greenblatt, Donald E. Eastlake, III, and Stephen D. Crocker. The greenblatt chess program. In *Proceedings of the November, Fall Joint Computer Conference*, AFIPS '67 (Fall), pages 801–810. ACM, 1967.

[GHM17] Chao Gao, Ryan B. Hayward, and Martin Müller. Move prediction using deep convolutional neural networks in hex. *IEEE Transactions on Games*, 2017.

[GMH18] Chao Gao, Martin Müller, and Ryan Hayward. Three-head neural network architecture for monte carlo tree search. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3762–3768. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[HAH09] Philip Henderson, Broderick Arneson, and Ryan Hayward. Solving 8×8 hex. pages 505–510. Joint Conference Artificial Intelligence, 2009.

[HAH10] Philip Henderson, Broderick Arneson, and Ryan B. Hayward. Hex, braids, the crossing rule, and xh-search. In *Advances in Computer Games*, pages 88–98. Springer Berlin Heidelberg, 2010.

[HAH$^+$14] Shih-Chieh Huang, Broderick Arneson, Ryan B. Hayward, Martin Muller, and Jakub Pawlewicz. Mohex2.0 : A pattern-based mcts hex player. *Computer and Games, Springer LNCS*, 8427:60–71, 2014.

[HAHP13] Ryan Hayward, Broderic Arneson, Shih-Chieh Huang, and Jakub Pawlewicz. Mohex wins hex tournament. In *ICGA Journal*, volume 36, pages 180–183, Seqtember 2013.

[Hay06] Ryan B. Hayward. Six wins hex tournament. In *ICGA Journal*, volume 29, pages 163–165, September 2006.

[Hay12] Ryan B. Hayward. Mohex wins hex tournament. In *ICGA Journal*, volume 35, pages 124–127, June 2012.

[HBJ$^+$05] Ryan B. Hayward, Yngvi Björnsson, Michael Johanson, Morgan Kan, Nathan Po, and Jack van Rijswijck. Solving 7×7 hex with domination, fill-in, and virtual connections. In *Theoretical Computer Science*, volume 349, pages 123–139, 2005.

[Hen10] Philip Thomas Henderson. *Playing and Solving the Game of Hex*. PhD thesis, the University of Alberta, 2010.

[HK14] Kunihito Hoki and Tomoyuki Kaneko. Large-scale optimization for evaluation functions with minimax search. *Journal of Artificial Intelligence Research*, 49:527–568, 2014.

[Hok06] Kunihito Hoki. Optimal control of minimax search results to learn positional evaluation. In *Game Programing Workshop 2006*, pages 78–83, 2006.

[HPTvdV17] Ryan Hayward, Jakub Pawlewicz, Kei Takada, and Tony van der Valk. Mohex wins 2015 hex 11x11 and hex 13x13 tournaments. In *ICGA Journal*, volume 39, pages 60–64, 2017.

[HvR06] Ryan B. Hayward and Jack van Rijswijck. Hex and combinatorics. *Discrete Mathematics*, 306(19):2515 – 2528, 2006.

[HW17] Ryan B. Hayward and Noah Weninger. Hex 2017: Mohex wins hex 11x11 and 13x13 tournaments. In *ICGA Journal*, volume 39, pages 222–227, 2017.

[HWY+] Ryan B. Hayward, Noah Weninger, Kenny Young, Kei Takada, and Tianli Zhang. Mohex wins hex 11x11 and hex 13x13 tournaments. unpublished.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034. IEEE Computer Society, 2015.

[ISR02] Hiroyuki Iida, Makoto Sakuta, and Jeff Rollason. Computer shogi. *Artificial Intelligence*, 134(1):121 – 144, 2002.

[Jas18] Wojciech Jaskowski. Mastering 2048 with delayed temporal coherence learning, multistage weight promotion, redundant encoding, and carousel shaping. *IEEE Transactions on Games*, 10(1):3–14, March 2018.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, December 2014.

[Kin18] David King. Hall of hexagons. http://www.drking.org.uk/hexagons /hex/index.html, 2018. [Online; accessed 1-May-2018].

[KM75] Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta priming. *Artificial Intelligence*, 6(4):293–326, 1975.

[Kor85] Richard E. Korf. Depth-first iterative-deepening:an optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KSHZ04] Alexandros Karatzoglou, Alex Smola, Kurt Hornik, and Achim Zeileis. kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004.

[LHP^{+}15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[Lor08] Richard J. Lorentz. Amazons discover monte-carlo. In *Computers and Games*, Lecture Notes in Computer Science, pages 13–24. Springer Berlin Heidelberg, 2008.

[Mar13] T. A. Marsland. A review of game-tree pruning, 2013.

[MHSS15] Chris J. Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in go using deep convolutional neural networks. In *3rd International Conference on Learning Representations*, 2015.

[min97] Mind sports olympiad. The Times, August 1997.

[MSN06] Ken Mishima, Hidetoshi Sakurai, and Kohei Noshita. New proof techniques and their applications to winning strategies in hex. In *11th Game Programming Workshop*, pages 136–142, 2006.

[Nas52] John Nash. Some games and machines for playing them. Technical report, Rand Corp D-1164, February 1952.

[NI02] Ayumu Nagai and Hiroshi Imai. Proof for the equivalence between some best-first algorithms and depth-first algorithms for and/or trees. *IEICE transactions on information and systems*, 85(10):1645–1653, October 2002.

[Nos04] Kohei Noshita. Union-connections and a simple readable winning way in 7x7 hex. In *Proceedings of 9th Game Programming Workshop in Japan*, pages 72–79, 2004.

[PH13] Jakub Pawlewicz and Ryan B. Hayward. Scalable parallel dfpn search. In *Computers and Games CG2013*, volume 8427 of *Lecture Notes in Computer Science*, pages 138–150. Springer, 2013.

[PHHA15] J. Pawlewicz, R. Hayward, P. Henderson, and B. Arneson. Stronger virtual connections in hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2):156–166, June 2015.

[R C15] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2015.

[SGG$^+$15] Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16:1629–1676, 2015.

[Sha53] C. E. Shannon. Computers and automata. *Proceedings of the IRE*, 41(10):1234–1241, October 1953.

[SHM$^+$16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[SHS$^+$17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv:1712.01815*, December 2017.

[SR07] Cornelis J. Stam and Jaap C. Reijneveld. Graph theoretical analysis of complex networks in the brain. *Nonlinear Biomedical Physics*, 1(1):3, July 2007.

[SSS$^+$17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[Tes95] Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, March 1995.

[TGJ$^+$15] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 648–656, 2015.

[THIY14] Kei Takada, Masaya Honjo, Hiroyuki Iizuka, and Masahito Yamamoto. Development of computer hex strategy using network characteristics. *IPSJ Journal*, 55(11):1–10, November 2014.

[THIY15a] Kei Takada, Masaya Honjo, Hiroyuki Iizuka, and Masahito Yamamoto. Developing computer hex using global and local evaluation based on board network characteristics. In Aske Plaat, Jaap van den Herik, and Walter Kosters, editors, *Advances in Computer Games*, pages 235–246, Cham, 2015. Springer International Publishing.

[THIY15b] Kei Takada, Masaya Honjo, Hiroyuki Iizuka, and Masahito Yamamoto. Developing evaluation function of hex using board network characteristics and svm. *Transactions of the Japanese Society for Artificial Intelligence : AI*, 30(6):729–736, 2015.

[THIY18] Kei Takada, Masaya Honjo, Hiroyuki Iizuka, and Masahito Yamamoto. Computer hex algorithm using a move evaluation method based on a convolutional neural network. In *Communications in Computer and Information Science*, pages 19–33, 2018.

[TIY] Kei Takada, Hiroyuki Iizuka, and Masahito Yamamoto. Reinforcement learning to create evaluation and policy functions using minimax tree search in hex. IEEE Transactions on Games, to appear.

[TIY17] Kei Takada, Hiroyuki Iizuka, and Masahito Yamamoto. Reinforcement learning for creating evaluation function using convolutional neural network in hex. *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 196–201, 2017.

[TKK12] Markus Thill, Patrick Koch, and Wolfgang Konen. Reinforcement learning with n-tuples on the game connect-4. In *Parallel Problem Solving from Nature - PPSN XII*, pages 184–194. Springer Berlin Heidelberg, 2012.

[VSBU09] Joel Veness, David Silver, Alan Blair, and William Uther. Bootstrapping from game tree search. In *Advances in Neural Information Processing Systems 22*, pages 1937–1945. 2009.

[XG11] Yoshua Bengio Xavier Glorot, Antoine Bordes. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323. PMLR, 2011.

[YLP01] Jing Yang, Simon Liao, and Mirek Pawlak. On a decomposition method for finding winning strategy in hex game. In *International Conference on Application and Development of Computer Games in the 21st Century*, pages 96–111, 2001.

[YLP03] Jing Yang, Simon Liao, and Miroslaw Pawlak. New winning and losing positions for 7x7 hex. In *Computers and Games*, pages 230–248, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[You13] Kenny Young. benzene-vanilla. https://github.com/kenjyoung/benzene-vanilla, 2013.

[YVH17] Kenny Young, Gautham Vasan, and Ryan Hayward. Neurohex: A deep q-learning hex agent. In *Computer Games*, pages 3–18. Springer International Publishing, 2017.