Universiteit Leiden
Opleiding Informatica

Predicting the Outcome
of the Game Othello

Name: Simone Cammel

Date: August 31, 2015

1st supervisor: Walter Kosters
2nd supervisor: Jeannette de Graaf

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

Abstract

Quite a lot of research has been done regarding the game of Othello. Most of the research was done to solve Othello, played on an 8×8 sized board. We study different sized boards of Othello and try to predict the outcome when both players play optimal. For boards sized 4×5 and greater, we use neural networks to try to predict the outcome. We try to train a network to create a strong Othello player for boards of varying sizes.

1 Introduction

In this thesis we study the boardgame Othello, also known as Reversi. Othello is a strategic boardgame played on a 8×8 sized board. The game is strongly solved on 4×4 and 6×6 sized boards. These both result in a white win with perfect play. For 8x8 Othello it is suspected that the game theoretical outcome is a draw.

We investigate different sized boards of Othello, for example $2 \times n$ sized boards and odd sized boards such as the 5×5 board.

In the second chapter more about the game will be explained, including the rules, the board and some background about the origin of the game. In the third chapter we discuss the different sized boards to discover patterns in game configurations. In the fourth chapter we try to predict the outcome of a game using a neural network, optimizing the neural network with different methods. In the last chapters we try to optimize a neural network using temporal difference learning, to make a strong computerplayer.

This thesis was written as a Bachelor project at the Leiden Institute of Advanced Computer Science (LIACS) of Universiteit Leiden, and has been supervised by Walter Kosters and Jeannette de Graaf.

2 The rules

In this section we explain the rules of the game Othello, including the origin of the game.

2.1 The board

The game Othello is usually played on an 8×8 board. It is played by two players. The stones are two-sided rounds that are black on one side and white on the other. The game always begins with the following initial configuration: there are four stones, two of either colour. The stones occupy the middle four squares of the board. A white stone lies on the upperleft corner of the four central squares. The other white stone lies in the lower right corner. The black stones occupy the other diagonal of the center, see Figure 1. There is a variation of the initial variation, where the top row is occupied by two white stones and the lower row by the black stones, see Figure 2. Black is the beginning player.

2.2 Reversi

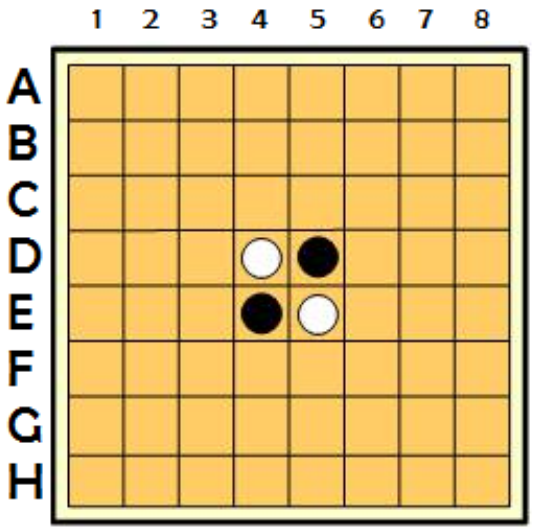
The game Reversi is nowadays synonymous to Othello and has the same rules. However, the original game of Reversi is played a little bit differently than Othello. This game was played in 19th century Europe. The first four pieces were played anywhere on the board as long as they were adjacent to

each other, but still forming a square. There was also a fixed number of pieces for each player. If a player ran out of pieces, the opponent continued making moves until the game ended.

2.3 Making a move

A player makes a move by placing a stone of his/her own colour on a still empty square on the board. The placed stone has to close in, or "sandwich", one or multiple stones of the opponent. This can happen horizontally, vertically and diagonally. The possible moves that can be made by black in the initial configuration are shown in Figure 3.

After a move has been made, the enclosed stones of the opponent switch colour. If, in the starting position, black chooses to place a black stone on square D3, then D4 is replaced by a black stone,



[!b]

Figure 1: The initial configuration of the Othello board.

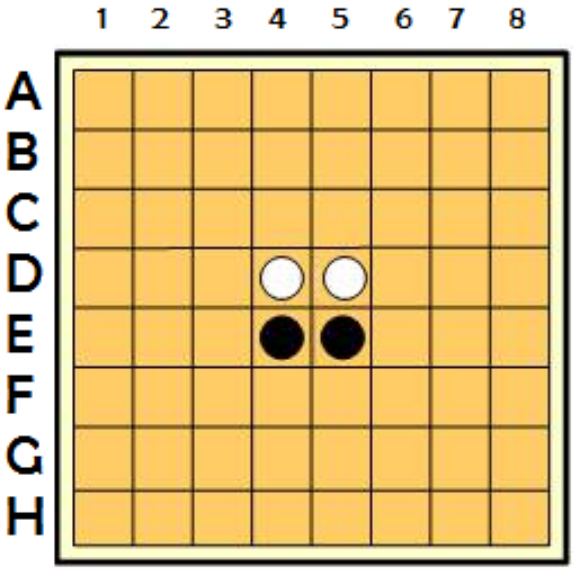


Figure 2: Variation of the initial configuration of the Othello board.

shown in Figure 4. Now it is white's turn to make a move. The possible moves are shown in Figure 5.

When a player has no possible moves to make, the player skips his/her turn and it is the other player's turn.

2.4 Non-iterative flipping of stones

Flipping stones is not done iteratively, as can be seen in Figure 6. Assume it is white's turn. White now has two possible positions where he/she can place a stone, namely C4 and C5.

Both moves would result in three black stones that become white, and a single black stone surrounded by white ones. However, even though this stone is surrounded by stones of the other player it will not

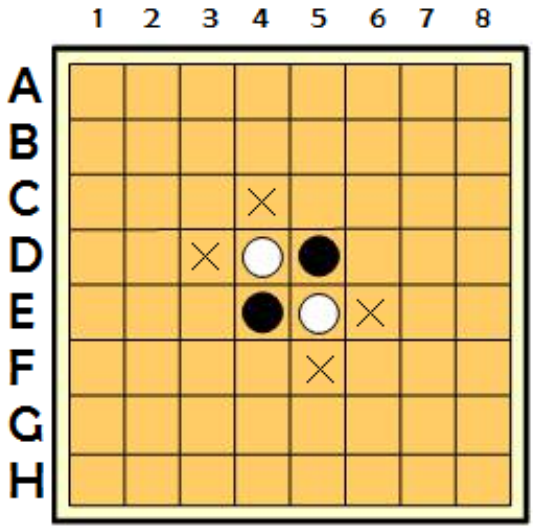


Figure 3: Possible moves that can be made by black.

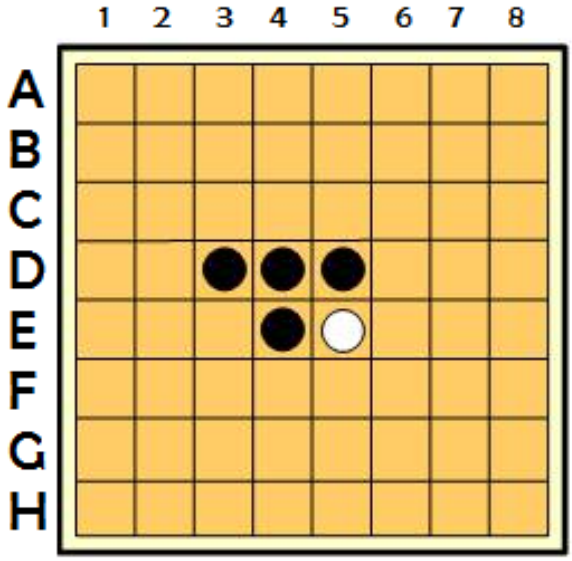


Figure 4: The board after a move made by black.

be flipped to become white. Flipping stones is not done iteratively.

2.5 Objective of the game

The object of the game is to have the majority of one's colour discs on the board at the end of the game.

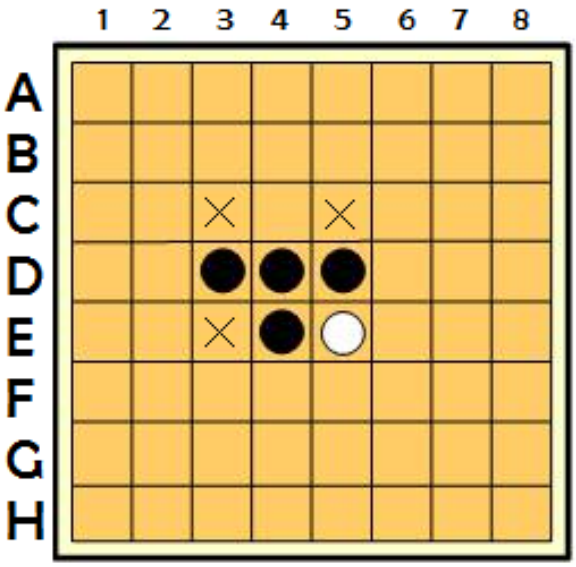


Figure 5: Possible moves that can be made by white.

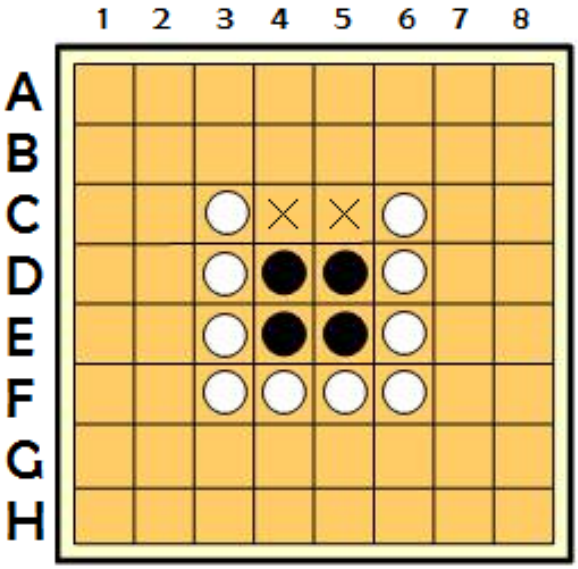


Figure 6: Flipping stones is not done iteratively.

2.6 End of the game

When it is not possible for any of the players to do a move, the game is finished. The stones on the board are counted, and the player with the most stones on the board wins. When both players have an equal amount of stones placed on the board, the game will result in a tie. Notice that it is possible for a game to end before all the squares are filled with stones.

2.7 Important squares

On an $m \times n$ board, with $m > 1$ and $n > 1$, the stones that are already placed at the beginning of the game, lie on the squares $(\lfloor \frac{m}{2} \rfloor, \lfloor \frac{n}{2} \rfloor)$, $(\lfloor \frac{m}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor)$, $(\lfloor \frac{m}{2} \rfloor, \lfloor \frac{n}{2} \rfloor + 1)$, $(\lfloor \frac{m}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 1)$, where the y-value corresponds to the matching letter of the alfabet, e.g., 1 corresponds to A and 3 corresponds to C. These spots are thus always occupied. If m and/or n is odd, there is no one unique square that will be chosen as the middle square. In this case, one can choose from either n or $n-1$ and m or $m-1$ as a replacement for m and n in the coordinates.

The corners of the board are important squares. Those squares, once taken, can not be flipped by the opponent. They are $(1,1)$ (corresponding with $(A,1)$), $(1,n)$, $(m,1)$ and (m,n).

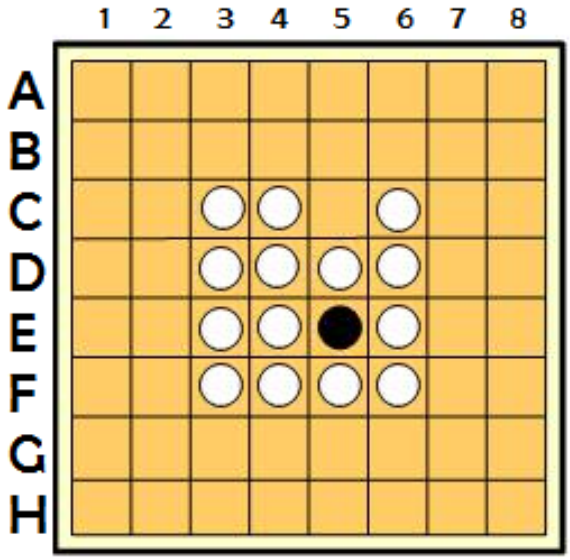


Figure 7: Result of placing a white stone on C4.

3 Gameplay on different sized boards

In this chapter we will evaluate different sized boards to see if there is a pattern to be discovered.

3.1 Definitions

In this section we will define different kinds of play, for example when a move or a game is called good or optimal.

Optimal Play
When a player has to pick a move to make, we want the player to play optimal.

Definition 1. *Optimal play means the following:*

- *If the current player can win, a winning move with the* shortest *path should be taken.*
- *If the current player can not win, all the moves should be checked to be sure.*

The problem in the definition lies within the search for the shortest path.

Shortest path
When looking at the gametree, one could talk about the shortest path. However, there can be numerous definitions for the "shortest path". One could want the path that results in a tree with the lowest width or the lowest height. We have used the following definition:

Definition 2. *The shortest path is defined as the path from the root to a leaf with the smallest distance, meaning that it consists of as little edges as possible.*

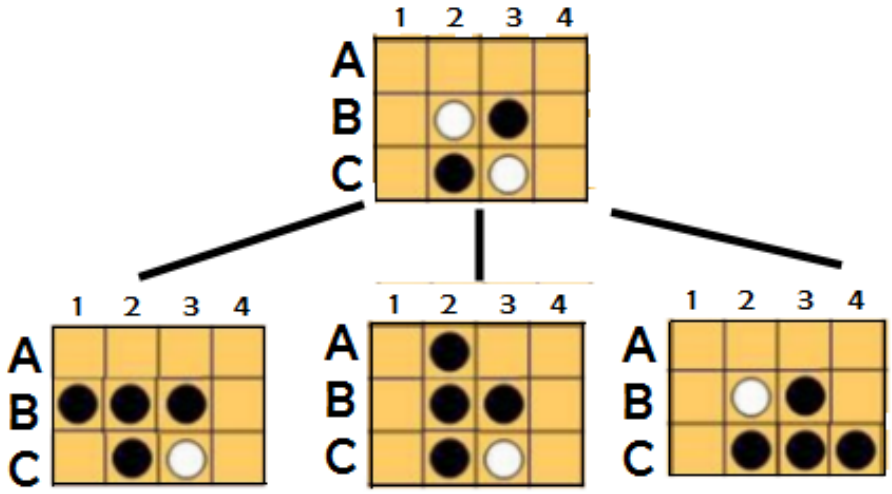


Figure 8: Part of the gametree of a 3×4 sized board that shows the possible moves to make for the black player. After computing the complete game tree bruteforce, we see that only the last, rightmost option results in a black win with optimal play by both players. The other two moves result in a win for white. If black plays optimal, only the rightmost option should be checked.

Also, we do not account for differences in winning. One can have a great win with a big difference in stones, but this is outside the scope of this thesis. We consider every win as a regular win, without considering the "largeness" of the win.

Optimal tree
A gametree is called optimal when only the following nodes are present:

- For every node that has one child that will result in a win for the current player, only this child should be present in the tree.
- If a node has more winning moves, only the child with the shortest path should be present in the tree.
- When there are multiple paths that are the shortest, the one that was found first will be the shortest path for this gametree.
- When there are no winning children, but there are draws, all children should be present in the optimal tree.
- If there are no children that result in a win or a draw, so only losses, all children should be present in the optimal tree.

Optimal outcome
We can now calculate the outcome of a game played in the optimal way. Notice that it does not matter for the outcome if the shortest path was taken or not, if they are both winning moves the result does not differ. The results of games played in the optimal way can be seen in Table 1. This Table was created by computing the outcome of the game played in the optimal way brute force.

It is remarkable that only games played on a 4×4 sized board result in a win for white. It is known that white also wins on a 6×6 sized board [8], however we have not yet been able to verify this.

On a 3×3 sized board it depends on the starting position. Because 3 is odd, there is no fixed middle. The outcome is a win for the player starting with a stone placed in a corner.

height / width	**2**	**3**	**4**	**5**	**6**	**7**	**8**
2	-	draw	draw	draw	draw	draw	draw
3	draw	W or B	B	B	B	B	B
4	draw	B	W	B	B	B	B
5	draw	B	B	B	B	?	?
6	draw	B	B	B	?	?	?
7	draw	B	B	?	?	?	?
8	draw	B	B	?	?	?	?

Table 1: Result of games played in an optimal way on different sized boards (not yet taking the shortest path into account). A ? means that we were not able to determine the winner because of the vast size of the gametree.

Size of the optimal trees
When constructing the optimal tree, it is interesting to know the improvement in size of the gametree that can be made:

- 3×2
  For a 3×2 sized board, the size of the optimal tree is 3 nodes. The optimal path also exists of 3 nodes. There is no improvement possible.

- 3×3
  For a 3×3 sized board, the size of the total optimal tree is 67 nodes. The optimal path exists of 13 nodes. This results in a gametree that is 19.40% the size of the total gametree.
- 3×4
  For a 3×4 sized board, the total tree contains 760 nodes. The optimal path exists of 6 nodes. This results in a gametree that is 0.88% the size of the total gametree, a significant improvement.

3.2 $m \times 2$ sized boards

When looking at $m \times 2$ sized boards, we see that they evolve in a very regular way. The situation where the board size is 2×2 is trivial. There are no possible moves so there is an immediate tie. Boards with a larger width also result in a tie, but in a different way.

Theorem 1. *A $m \times 2$ board with $m \geq 3$ always results in either one of the board configurations in Figure 9.*

Proof. There are two endgames possible for games played on boards sized $m \times 2$. The two possible paths that can be taken can be seen in Figure 10. It results in a draw, with exactly three white stones and three black stones. Even though there are still empty positions above and below the placed stones, there are no more moves left. This means that no matter how big m will get, the result will be the same.

□

3.3 $3 \times n$ sized boards

Games played on $3 \times n$ sized boards behave differently from the ones played on $2 \times n$ sized boards. They behave differently in ways of expanding, there are more choices to be made by each player, and there are no games resulting in a tie. We consider increasing width:

3×2 sized board
The board with size 3×2 will not be considered because this is covered in the previous section. This is a regular $2 \times n$ or $m \times 2$ sized board and thus the gameplay results in a tie after 3 moves.

Figure 9: Board configurations.

3×3 sized board

The 3×3 sized board is an exception to the other $3 \times n$ sized boards. There are essentially two possibible configurations to begin with, see Figure 11. There are not four but only two different starting positions as the boards seen in Figure 11 diagonally from each other are in fact rotations of one another. Choosing a starting configuration has great consequences on the evolution of the gameplay.

The main difference that ensures a different winning player is that in the chosen configuration, one of the players is already in posession of a corner. The evolution of the games, when played optimal, is similar. Both consist of the situation where there is one of the sidecolumns filled with stones from one player, and the middle column filled with stones from the other player. The winner in this case is always the one with the filled sidecolumn. This is not an unexpected result since this player already owns two out of four corners. The final configuration can be seen in Figure 12.

$3 \times n$ boards with $n > 3$

Boards that are sized $3 \times n$ with $n > 3$ show a very repetitive pattern. The following paragraph describes patterns and conjectures of boards that begin in the configuration where the four stones lie as high and as left as possible. This means that the middle square will be at position ($\lfloor \frac{(3-1)}{2} \rfloor$, $\lfloor \frac{n}{2} \rfloor$). This is necessary because there are 2 or 4 beginconfigurations for each of the boards following. Considering this assumptipon, we can make some statements about boards of size $3 \times n$.

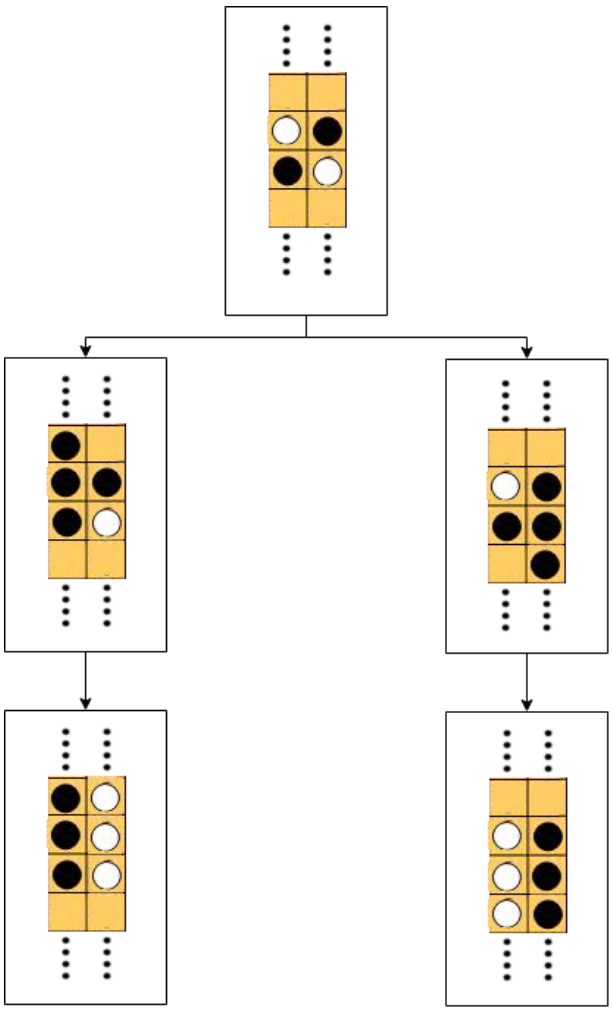


Figure 10: Gametree for a $m \times 2$ tree.

Theorem 2. *A game of Othello played optimal on a $3 \times n$ board with $n > 3$ always ends in the following number of moves:*

$$\begin{array}{ll} n+2 & \text{if } n \text{ is even} \\ n+1 & \text{if } n \text{ is odd} \end{array}$$

We will show why this theorem is true by first looking at the gametree of a board with size 3×4.

As can be seen in Figure 13, the game fairly quickly results in a game dominated by the black player. From the moment that the game reaches the point where the left column is filled with black stones and the middle column is filled with white stones, the white player has no opportunities to place a stone in the rest of the game. The black player is dominant. Notice that the final stone can be placed on three fields: C3, D2 and D3.

Every game with board size $n \times 3$ with $n > 3$ results in this or a similar situation.

Definition 3. *A stable state is a moment in the game from whereon one of the players is the only one that can move. The other player does not have any moves left and has to pass each time it is his or her turn, for the rest of the game.*

Remark that a stable state is not always reached when a player has to pass. It can happen that after

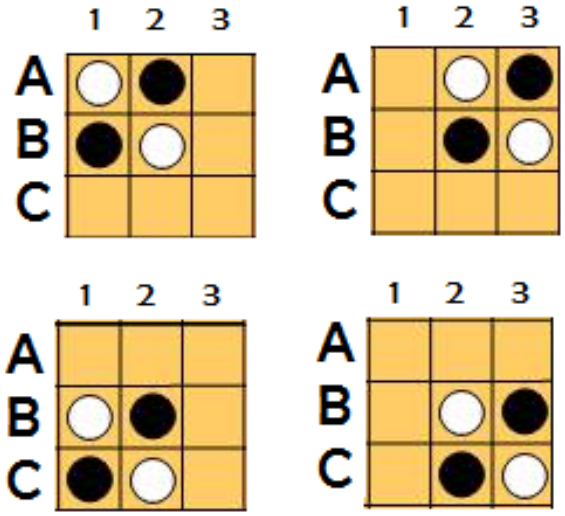


Figure 11: Starting configurations for 3x3 sized board.

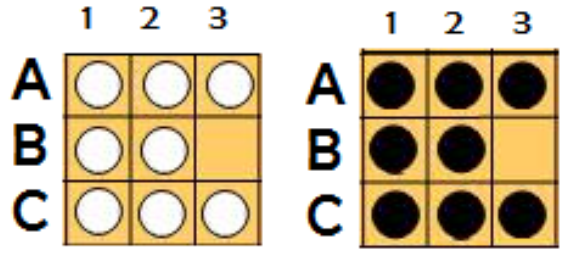


Figure 12: Final configurations for 3x3 sized board.

one or more moves made by the other player, the player is able to place a stone on the board.

The stable state it reaches does vary a little bit with different sized boards but the evolution of all these games has the same structure.

These results, meaning every $3 \times n$ sized board results in a stable state, are obtained because black always tries to conquer the upper right corner first. In this first part of the game, white does get to place stones but always has only one possible move, resulting in a game where black always knows what the next game configuration will look like. It will take m (or if m is odd $m-1$) moves to get to a stable state, where it will only be a matter of placing one extra stone on the position (m, 2), or if m is odd on the position corresponding to ($m-1$,2).

Conjecture 3. *A game of Othello played optimal on a $m \times 3$ board with $m > 3$ always results in a* black win *with 0 white stones remaining, and the following number of black stones on the board:*

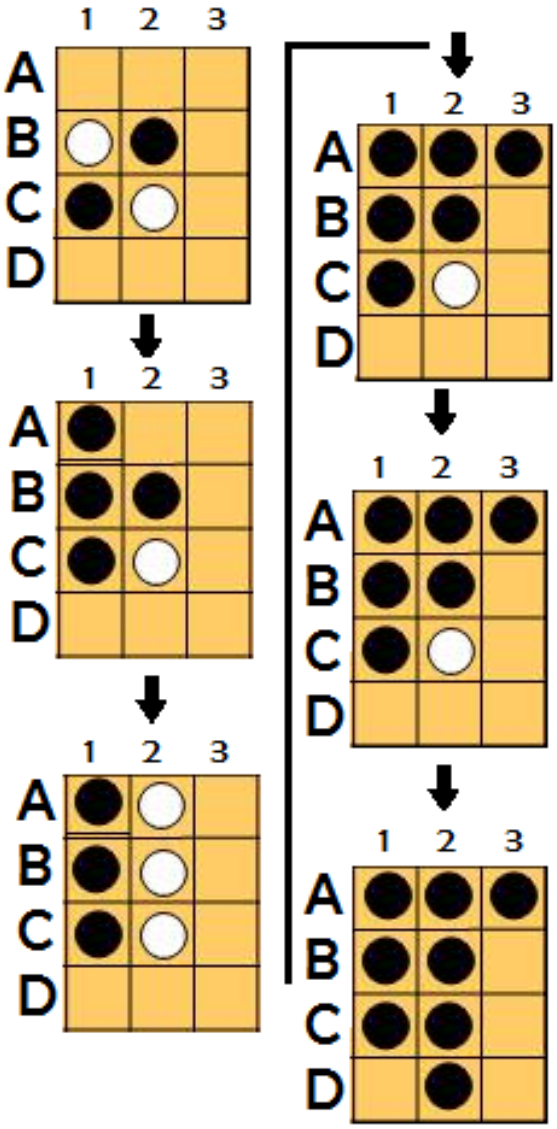


Figure 13: Gametree of a game played on a 4x3 sized board.

$$\begin{array}{ll} m+4 & \textit{if } m \textit{ is even} \\ m+3 & \textit{if } m \textit{ is odd} \end{array}$$

3.4 $m \times n$ sized boards with $m > 3$ and $n > 3$

Bigger sized boards have more patterns than the smaller ones. It is harder to predict the outcome and there are more varying outcomes. For example, a 4×4 sized board has 30 169 endgames in which white wins, 24 611 endgames in which black wins and 5 297 endgames that are a draw.

When we look at even bigger sized boards, for example a 4×5 sized board, the amount of endgames is obviously a lot greater. There are 13 833 538 endgames in which white is the winner, 13 812 384 in which black is the winner and 2 529 346 that are a draw.

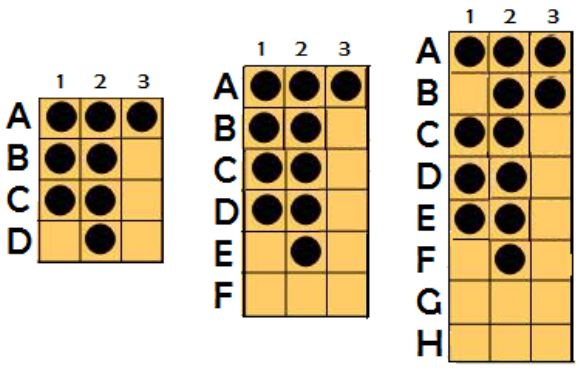


Figure 14: Final configurations of a 4×3, 6×3 and 8×3 sized board.

4 Using a neural network to predict the outcome

We want to be able to predict the outcome of any given game, because we saw in the previous chapter that it is difficult to do this brute force. This prediction can be made by using a neural network.

4.1 Neural network

In an artificial neural network [10], simple artificial nodes, known as "neurons" are connected together to form a network which mimics a biological neural network.

An artificial neural network is an interconnected group of nodes. In Figure 15, each circular node respresents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.

Every input has a certain weight that will represent the matter of importance of this input. The weights will be adjusted during the training of the network. When the training is finished, the resulting weights are used to calculate the output for any given input. The weights are adjusted by using backpropagation, meaning that for every training the error in the output is determined and the weights are adjusted accordingly.

4.2 Definitions

When predicting the outcome (between 0 and 1, a 0 corresponding with a black win, 0.5 corresponding with a draw and a 1 corresponding with a white win) of the game of a given board with a neural network, we consider a neural network *good* for a node when:

Definition 4. *A neural network is called good for a node in a gametree with ℓ (where $\ell \geq 2$) children if for all children $k_1, \ldots, k_\ell$ the following holds for the predicted outcome $N(k_i)$ (where $1 \leq i \leq \ell$) of the neural network:*

- $N(k_i) > N(k_j)$, *with* $1 \leq i \leq \ell$, $1 \leq j \leq \ell$ *and* $i \neq j$

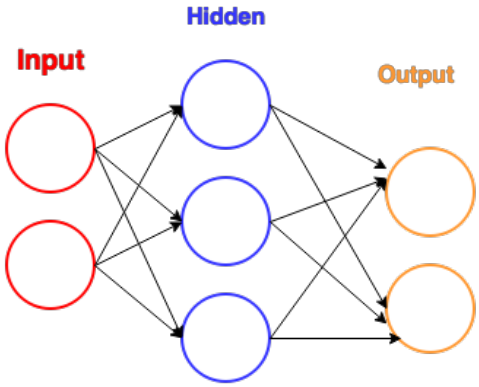


Figure 15: Example of a Neural Network.

- *when k_i results in a winning situation and k_j results in a draw or a loss*
- *when k_i results in a draw and k_j results in a loss*

Some wrong predictions, misses, have greater effect than others. This means they are *consequential misses*:

Definition 5. *A miss is called a* consequential miss *when the child k_j with the highest predicted outcome results in a different winner than the best possible outcome for this situation, whereas the best possible situation means:*

- *If there are one or more winning node(s), one of these nodes should have the highest predicted outcome.*
- *If there are no winning nodes, but there is one or multiple situation(s) that result in a draw: a node resulting in a draw should have the highest predicted outcome.*

4.3 Varying parameters

When making a prediction for a game, there are many network parameters that can be varied. We vary the following parameters:

- the number of used examples
- the number of hidden layers
- the number of input nodes, or more general: the format of the input

The number of used examples
We can vary the number of used examples the network will train on. For example, on a 4×4 sized board the gametree consists of 224 822 nodes. One could decide to let the network train multiple times on all the different game configurations.

The number of hidden layers
One can vary the number of hidden layers, though there are some rules of thumb [3]:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be $\frac{2}{3}$ the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

The number of input nodes
Translating a game configuration to a string of numbers between zero and one can be done in multiple ways. The current player is represented as a 0, when it is black's turn, or a 1, when it is white's turn. The board is represented by an input for every field: a 0 when occupied by a black stone, a 0.5 when empty and a 1 when occupied by a white stone. These will be read starting at the topleft corner, reading horizontally to the right line by line. The output is represented as follows: a 0 for a black win, 0.5 for a draw and a 1 for a white win. Then there are multiple heuristics we can add, increasing the number of inputs nodes. These heuristics can be for example:

- The number of occupied corners
  Represented by a 0 when black has more corners occupied, a 1 when white had more corners occupied and a 0.5 when the amount of corners occupied is equal for both players.

- Current amount of stones on the board
  Represented by a 0 when black has more stones placed on the board, a 1 when white has more stones placed on the board and 0.5 when the amount of stones placed on the board is equal for both players.
- Are there stones placed close to a corner, meaning a corner could possibly be taken easily in the future
  The three neighbouring fields of each corner are regarded. This will be represented by a 0 when black has more stones placed on neighbouring fields of corners, it will be represented by a 1 when white has more stones placed on neigbouring fields of corners and 0.5 when the amount of stones placed on neighbouring fields of corners is equal.
- The number of moves
  Represented by a 0 if black has more possible moves to make in the current situation, a 1 if white has more possible moves to make in the current situation and a 0.5 if the amount of moves to make is equal for both players.
- Easy takeable stones
  Each placed stone is evaluated. If the stone has one direct neighbouring field that is empty, it is valued as a possibly easy takeable stone. In the end this will be represented by a 0 if black has more easily takeable stones placed on the board and a 1 if white has more easily takeable stones placed on the board.It will be represented by a 0.5 when these amounts are equal.

4.4 Predicting the output for games played on a 4×4 sized board

Firstly we want to optimize all the parameters for the 4×4 sized board so we can apply this later on to bigger sized boards.

In these first results, seen in Table 2, we vary the number of used examples for inputs generated by the 16 fields of the board and the current player. The number of hiddens in this situation is 4. The error is the absolute difference between the predicted outcome and the optimal outcome for each game configuration divided by the total number of game configurations. To evaluate moves we need to define what it means for a game to have a *good* result:

Definition 6. *We speak of a good result when a game played choosing moves based on the prediction of a neural network results in the same result as if the game were played optimal.*

Used Examples	Hidden neurons	Inputs	Error	Sorted without misses	Good result
100	4	17	0.444	93.02 %	94.81 %
1000	4	17	0.423	95.47 %	97.05 %
10000	4	17	0.225	96.42 %	97.84 %
100000	4	17	0.182	95.65 %	97.23 %
1000000	4	17	0.174	96.18 %	97.40 %
10000000	4	17	0.167	96.07 %	97.40 %

Table 2: Results of predictions for 4×4 sized boards with 4 hidden neurons. Sorted without misses means that for each node it's children are sorted according to their predicted outcome. A miss occurs when a node in the sorted list does not match with the known perfectly sorted gametree

Now we test the heuristics, but then with a varying amount of hidden neurons. Considering the earlier

presented rules of thumb regarding the number of hidden neurons, the number of hidden neurons in this case should be between 1 and 17, 12 or smaller than 34. The results can be seen in Table 3 and Table 4.

Used Examples	Hidden neurons	Inputs	Error	Sorted without misses	Good result
100	5	17	0.45	95.27 %	96.77 %
1000	5	17	0.41	93.53 %	95.10 %
10000	5	17	0.24	96.33 %	97.88 %
100000	5	17	0.18	95.86 %	97.19 %
1000000	5	17	0.18	96.09 %	97.63 %

Table 3: Results of predictions for 4×4 sized boards with 5 hidden neurons.

Used Examples	Hidden neurons	Inputs	Error	Sorted without misses	Good result
100	6	17	0.44	95.02 %	96.77 %
1000	6	17	0.43	94.35 %	95.95 %
10000	6	17	0.23	95.95 %	97.36 %
100000	6	17	0.18	96.15 %	97.60 %
1000000	6	17	0.17	96.30 %	97.37 %

Table 4: Results of predictions for 4×4 sized boards with 6 hidden neurons.

Considering these results, we can see that the error keeps decreasing, though this does not result in a very good result sorting the children of a node. In the best case, there are still 3 883 misses made which is almost 4% of the predictions.

For further analysis we have added the earlier desribed heuristics to the input. This means we now give the neural network some information we think is important more concrete so we would expect the netwerk to be able to give a better prediction.

Used Examples	Hidden neurons	Inputs	Error	Sorted without misses	Good result
100	4	22	0.46	92.88 %	94.78 %
1000	4	22	0.41	93.78 %	95.28 %
10000	4	22	0.16	97.13 %	98.46 %
100000	4	22	0.12	97.20 %	98.48 %
1000000	4	22	0.11	97.31 %	98.51 %
10000000	4	22	0.10	97.42 %	98.53 %

Table 5: Results of predictions for 4×4 sized boards with 4 hidden neurons and 22 inputs.

Comparing Table 1 to Table 5 we can see that the more input nodes we have, the longer it takes to obtain better results from the neural network. But when we do train with more used examples, this results in a very good prediction. In the best case we now have only 2 709 misses, which means that a little over 2.5% of the instances have a wrong prediction that results in a different sorting of children. But in only 1.5% of the instances this results in a wrong/different possible outcome for the current player. Note that even then, this does not have to result in a different outcome, for there may be made more mistakes, caused by wrong prediction, in the subtree of a taken path.

Intuitively it seems wise to explore the optimal amount of hidden neurons while continuing to have 22 input nodes.

Hidden neurons	Error	Sorted without misses	Good result
4	0.1030	97.42 %	98.53 %
5	0.1015	97.43 %	98.62 %
6	0.1015	97.48 %	98.61 %
7	0.0966	97.42 %	98.53 %
8	0.0959	97.55 %	98.57 %
9	0.0972	97.43 %	98.53 %
10	0.0974	97.37 %	98.51 %

Table 6: Results of predictions for 4×4 sized boards with 22 inputs, 10 000 000 used examples and varying amount of hidden neurons.

Table 6 gives us a good idea of the optimal number of hidden neurons we should use. The number of hidden neurons with the smallest error is 8, with corresponding error 0.0959048. This also results in the highest percentage of sorted children, 97.55%. It is remarkable that even though the error decreases, the percentage of good results does not decrease proportionally but seems to behave quite randomly.

4.5 Evaluating misses on 4×4 sized boards

We can only further enhance the neural network or the prediction if we have some understanding of the errors it does make.

First of all we could check the overlap between the game configurations that result in a miss, a bad prediction.

hidden neurons	1	2	3	4	5	6	7	8	9	10
1	-	90.83	86.55	86.18	78.61	81.69	75.37	78.92	75.23	72.36
2	91.97	-	86.28	86.85	80.04	82.75	75.16	79.53	74.86	73.13
3	88.39	86.96	-	85.97	80.08	82.84	77.87	80.35	76.68	74.06
4	83.68	83.19	81.44	-	76.68	79.31	72.34	75.13	73.41	73.28
5	80.55	80.85	80.14	81.50	-	84.24	76.68	78.32	77.64	76.52
6	81.79	82.63	81.59	82.69	82.73	-	74.29	77.93	74.72	74.69
7	81.93	80.25	82.66	81.05	80.76	80.21	-	79.66	83.98	78.86
8	83.23	82.70	82.98	81.78	80.55	81.22	77.52	-	78.65	74.45
9	77.41	75.87	77.41	78.30	78.20	76.32	79.73	77.07	-	78.81
10	73.92	73.58	74.16	77.65	75.82	75.21	74.16	71.92	78.06	-

Table 7: Percentage of overlap between misses of neural network trained with 22 inputs, 10 000 000 training instances.

As can be seen in Table 7, there are many matches in misses with various neural networks with varying amounts of hidden neurons. It is interesting to take a closer look at the overlapping misses and try to characterize them.

4.6 Analysis

After the observation of the misses, we can conclude that they all follow a certain pattern. There are only two distinctive patterns of misses.

The first is as follows:

1. Black occupies the most corners
2. Black occupies most of the fields on the board
3. Black has more stones placed on positions that are possibly taken easily, in other words black has more stones surrounded by one or more empty fields than white
4. White has more moves left to make
5. It is white's turn to make a move

The second pattern also has the first three characteristics but varies on the last two:

4. Not white but black has more moves left to make
5. It is black's turn in stead of white's

Also, most of the mistakes are made in the final steps of the game when it is almost finished, as can be seen in Figure 16.

When we take a closer look at these mistakes, we can see that most of these mistakes involve the choice between placing a stone on the corner or somewhere else. In these cases it is often the case that occupying the last corner ensures your win. We now need to ensure that the neural network handles corners in a different than it does now. Now, we only have one 0, 0.5 or 1 indicating the player that has the majority of corners occupied. But, choosing this representation, there is no difference in evaluation when a player has one corner more than the other player, or more.

4.7 Enhancing the input neurons

As described before, the neural network does not yet value the occupation of corners as high as we think it should, as these corners are always yours once taken.

We add an input neuron for both the black and the white player:

Figure 16: Distribution of the misses according to the number of empty fields left.

1. We asign the value 0.0 when no corners are occupied by this player
2. We asign the value 0.25 when one corner is occupied by this player
3. We asign the value 0.5 when two corners are occupied by this player
4. We asign the value 0.75 when three corners are occupied by this player
5. We asign the value 1.0 when all corners are occupied by this player

The results can be seen in Table 8.

Hidden neurons	**Error of Table 6**	**Current error**	**Sorted without misses Table 6**	**Sorted without misses**
4	0.1010	0.1052	97.42%	97.45%
5	0.1015	0.1025	97.43%	97.45%
6	0.1015	0.0974	97.48%	97.36%
7	0.0967	0.0962	97.42%	97.41%
8	0.0959	0.0952	97.55%	97.28%
9	0.0972	0.0946	97.43%	97.18%
10	0.0974	0.0949	97.37%	97.17%

Table 8: Results of predictions for 4×4 sized boards with 24 inputs, 10 000 000 used examples and varying amount of hidden neurons.

4.8 Enhancing the testset

We can take the intersection of all the misses with varying amounts of hidden neurons and collect these in a file. We can let the neural network train extra on these instances, hopefully resulting in a better prediction for these game configurations that are reviewed very poorly. The results can be seen in Table 9.

Hidden neurons	**Error of Table 6**	**Current error**	**Sorted without misses Table 6**	**Sorted without misses**
4	0.1010	0.1030	97.42%	97.22%
5	0.1015	0.1109	97.43%	97.02%
6	0.1015	0.1011	97.48%	97.11%
7	0.0966	0.0991	97.42%	97.10%
8	0.0959	0.1026	97.55%	97.26%
9	0.0972	0.0944	97.43%	97.34%
10	0.0974	0.0935	97.37%	97.27%

Table 9: Results of predictions for 4×4 sized boards with 22 inputs, 10 000 000 used examples and varying amount of hidden neurons, using the intersection of all the misses to train 2 times more on than on any other game configuration.

Unfortunately, it seems the predictions are not getting better so this could mean that we are overtraining on the game configurations that were predicted poorly.

5 Temporal difference learning

The previous chapter shows that a neural network can learn to predict the outcome of a game when given all possible game configurations and their optimal outcome. It would be interesting to find a way to train a neural network without having the optimal outcome of all the possible game configurations. This is the case for games played using bigger sized boards than 4×4 sized boards, because determining the optimal outcome of the game brute force will take more than a few days.

5.1 Introduction

We try to learn to distinguish good moves from bad moves and by doing that we hope to make a strong computer player by using temporal difference learning [7]. This involves a neural network that is configured with random initial weights. The neural network plays against itself. The computer will calculate the move to make by predicting the outcome for every possible move the current player can make. Depending on the current player, the program will execute the move with the lowest predicted outcome (when it is black's turn) or the highest predicted outcome (when it is white's turn).

When the game is finished, all the executed moves are added to the training set, with the final result of this game as the target for each of the moves made. The weights are adjusted according to the new information. This means that the network only learns while interacting with the game of Othello.

Initally this is the way the network is learning from the games, but to apply temporal difference learning to the network according to the definition [7] we have to learn *while* playing the game and not at the end of the game. This can be added later on, or maybe be considered for future work.

There are also other ways to implement the temporal learning. We distinguish several variations:

- A network that trains only on the moves of one of the two players, the other player making moves randomly.
- Two networks, one for every player, playing against each other.
- One network for both players, playing against itself, as just described.

5.2 Temporal difference learning on a 4×4 sized board

To measure the performance of the predicted outcome, we first tested 4×4 sized boards because we have the optimal outcome for every game configuration. Comparing each predicted outcome with the optimal outcome, we can monitor the quality of the network over time.

5.3 Training the network

The goal is to obtain a network that plays perfectly, but to obtain this we test the different ways of temporal difference learning as described for 4×4 sized boards. We try to distinguish the best way to train the network.

Adding randomness when making a move and adjusting the testset
When looking at Figure 17 we can see that initially only a handfull of endgames are reached while training the network with the first strategy. The network decides (too) early on what the optimal strategy and thus move must be and does not regard anything else. This can be resolved by adding some randomness in the earlier stage of training. In the first 1000 games, sometimes a random move

is made instead of the best move at that point. This ensures that we do not choose a path too quickly but hopefully research a great enough part of the gametree to ensure the best network possible.

Random

First, we train the network against a random player. To measure the quality of the network, we review the average error (the absolute difference between the predicted outcome and the optimal outcome) per 100 played games. The results can be seen in Table 10. A difference in prediction does not always mean a different outcome of the game, if it still results in the optimal outcome this isn't a problem. Thus we also have to regard the winner of each game and compare this to the winner of the game in case it is played optimal. These results can be seen in Table 11. Reviewing these two tables, we can see that using 5 hiddens, with 1000 used examples each round of training, gives us the best result. Interesting is that in more than 60% of the games are won by black while in a perfectly played game white would be the winner. This suggests that the network does play better than a random player.

Now, it is interesting to see whether the percentage of games resulting in the optimal result will reach 100% over time. We can now train for a longer time with these parameters, so only varying the amount of games played. These results can be seen in Table 12. Unfortunately, we can not see an improvement of the number of games resulting in a win for black.

Part of the played games used to determine the average error	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**5 hiddens, 2000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**
0 - 100	0.2011	0.1739	0.1828	0.1839	0.1788	0.1703
101 - 200	0.1942	0.1793	0.1807	0.2187	0.2036	0.2037
201 - 300	0.2000	0.2122	0.1871	0.1853	0.1910	0.1909
301 - 400	0.2083	0.1930	0.1843	0.2173	0.2090	0.1925
401 - 500	0.2221	0.2084	0.2187	0.1915	0.2009	0.2210
501 - 600	0.1956	0.1817	0.1740	0.1798	0.1912	0.2140
601 - 700	0.1887	0.1837	0.1839	0.2015	0.1909	0.1990
701 - 800	0.1794	0.1775	0.1880	0.2004	0.1750	0.2025
801 - 900	0.2120	0.2037	0.1840	0.1977	0.1966	0.2121
901 - 1000	0.1489	0.1962	0.2156	0.2128	0.1780	0.2093

Table 10: Results of predictions for 4×4 sized boards with 24 inputs and varying parameters, each for 1000 played games. The average error over one hundred games is showed.

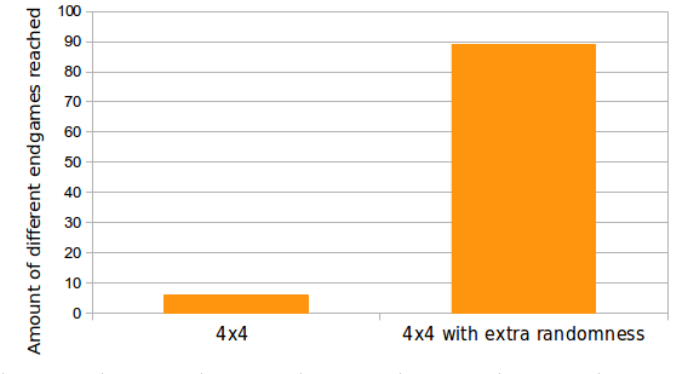


Figure 17: Amount of different endgames reached with or withour doing random moves

Part of the played games used to determine the percentage of black wins	4 hiddens, 1000 used examples	4 hiddens, 2000 used examples	5 hiddens, 1000 used examples	5 hiddens, 2000 used examples	6 hiddens, 1000 used examples	7 hiddens, 1000 used examples
0 - 100	61,0 %	62.0 %	68.0 %	69.0 %	69.0 %	70.0 %
101 - 200	73.0 %	69.0 %	70.0 %	63.0 %	67.0 %	60.0 %
201 - 300	68.0 %	61.5 %	66.0 %	56.5 %	65.0 %	71.0 %
301 - 400	64.0 %	69.0 %	69.0 %	66.5 %	71.0 %	70.0 %
401 - 500	75.0 %	64.5 %	64.0 %	76.5 %	70.0 %	71.0 %
501 - 600	61.0 %	63.5 %	74.0 %	59.5 %	69.0 %	70.0 %
601 - 700	70.0 %	66.5 %	73.0 %	70.5 %	69.0 %	74.0 %
701 - 800	77.0 %	71.0 %	68.0 %	68.0 %	70.0 %	67.0 %
801 - 900	64.0 %	67.5 %	64.0 %	66.0 %	69.0 %	59.0 %
901 - 1000	72.0 %	74.5 %	75.0 %	73.5 %	67.0 %	69.0 %

Table 11: Percentage of games resulting in a win for the black player.

Percentage of played games considered in average	1000 games	2000 games	3000 games	4000 games	10 000 games
0 - 10%	68.0 %	69.5 %	73.0 %	69.8 %	68.2 %
11 - 20 %	70.0 %	67.5 %	68.3 %	63.3 %	67.0 %
21 - 30 %	66.0 %	68.5 %	72.3 %	56.5 %	69.7 %
31 - 40 %	69.0 %	64.0 %	64.0 %	66.5 %	70.3 %
41 - 50 %	64.0 %	70.0 %	65.0 %	69.0 %	68.6 %
51 - 60 %	74.0 %	68.5 %	68.6 %	72.0 %	71.2 %
61 - 70 %	73.0 %	65.0 %	70.0 %	70.3 %	69.0 %
71 - 80 %	68.0 %	67.0 %	71.6 %	68.0 %	68.8 %
81 - 90 %	64.0 %	71.0 %	67.3 %	66.8 %	70.1 %
91 - 100 %	75.0 %	68.5 %	68.0 %	73.3 %	67.2 %

Table 12: Percentage of games resulting in a win for black, while white is the winner in case of the optimal played game, per one hundred played games.

Training against itself

Secondly we can train the network against itself, meaning that we use only one network, but when adding moves to the training set black and white moves are distinguished by either a 0 (for black) or a 1 (for white) as input for one of the nodes. A disadvantage of this method could be that when training the network, it will only be able to play good against similar players but when someone would place a stone on a random place on the board that was never seen during training, the network could react poorly. The results of this method can be seen in Table 13.

Immediately a huge difference in result can be detected. The average error goes to zero almost immediately, meaning the network does learn quite fast. The percentage of games won by black is also almost immediately zero, which is not a weird result considering the fact that white would be the winner when the game is played optimal, and with the average error near zero this could be the case.

Part of the played games used for the calculation	4 hiddens, 1000 used examples Average error	Percentage of games resulting in black win	5 hiddens, 1000 used examples Average error	Percentage of games resulting in black win	6 hiddens, 1000 used examples Average error	Percentage of games resulting in black win
0 - 100	0.1098	17 %	0.078	17 %	0.0816	18 %
101 - 200	0.00	0 %	0.00	0 %	0.00	0 %
201 - 300	0.00	0 %	0.00	0 %	0.00	0 %
301 - 400	0.00	0 %	0.00	0 %	0.00	0 %
401 - 500	0.00	0 %	0.00	0 %	0.00	0 %
501 - 600	0.00	0 %	0.00	0 %	0.00	0 %
601 - 700	0.00	0 %	0.00	0 %	0.00	0 %
701 - 800	0.00	0 %	0.00	0 %	0.00	0 %
801 - 900	0.00	0 %	0.00	0 %	0.00	0 %
901 - 1000	0.00	0 %	0.00	0 %	0.00	0 %

Table 13: Results of predictions and outcome for 4×4 sized boards with 24 inputs and varying parameters, each for 1000 played games.

Training against another network

We can also train the network against another network, one network per player. This can still result in an overtrained network only performing good on stronger players and weak against a random player, but the strength of the network could benefit from the division. The results of this method can be seen in Table 14. What is visasble is that black plays stronger (or white poorer), because the percentage of games resulting in a win for black are a lot higher than in the previous case.

Part of the played games used for the calculation	4 hiddens, 1000 used examples Average error	Percentage of games resulting in black win	5 hiddens, 1000 used examples Average error	Percentage of games resulting in black win	6 hiddens, 1000 used examples Average error	Percentage of games resulting in black win
0 - 100	0.1458	32 %	0.1712	45 %	0.1685	46 %
101 - 200	0.1677	41 %	0.1697	46 %	0.1768	45 %
201 - 300	0.2367	39 %	0.1905	42 %	0.1688	40 %
301 - 400	0.2279	45 %	0.2221	32 %	0.1656	41 %
401 - 500	0.2015	38 %	0.2144	34 %	0.2093	34 %
501 - 600	0.1386	49 %	0.1738	38 %	0.1732	49 %
601 - 700	0.1958	44 %	0.1767	40 %	0.1464	56 %
701 - 800	0.1976	48 %	0.1892	40 %	0.1995	45 %
801 - 900	0.1824	35 %	0.1790	46 %	0.2020	37 %
901 - 1000	0.1730	53 %	0.1977	41 %	0.2137	40 %

Table 14: Results of predictions and outcome for 4×4 sized boards with 24 inputs and varying parameters, each for 1000 played games.

Now it is interesting to see whether one of the players will grow stronger than the other over time. The result of the outcome of the game after training for a long time (10 000 games) can be seen in Table 15. The results vary, in two of the cases white becomes slightly stronger after training the network and when we used 5 hidden nodes, black became a stronger player after training the network. Considering all the obtained results, white is overall the better player.

	4 hiddens, 1000 used examples 10 000 games	**5 hiddens, 1000 used examples 10 000 games**	**6 hiddens, 1000 used examples 10 000 games**
Percentage of black wins in the last 100 games	40 %	48 %	35 %

Table 15: Results of predictions and outcome for 4×4 sized boards with 24 inputs and varying parameters, each for 10 000 played games.

Training against the perfect player

In the case of games played on a 4×4 sized board, we know what perfect play looks like and thus we

can train against an optimal player. However, the result is that the perfect player always wins and the network does not see a lot of different board configurations to learn from.

Training the best network against a random player
We can also test our best network against a random player. In this case we initialize the weights of the network with the weights we obtained from our best network yet. Hopefully, the network will then be a stronger player than a random player in the beginning. The results can be seen in Table 16. The network performs almost the same as it did against random without initializing the weights, the percentage of games resulting in a black win was 63.1% at it is best and now it is 62.8%, which is not really a significant difference.

Part of the played games used for the calculation	4 hiddens, 1000 used examples Using one network Average error	Percentage of games resulting in a black win	4 hiddens, 1000 used examples Using two netwokrs Average error	Percentage of games resulting in a black win	5 hiddens, 1000 used examples Using two networks Average error	Percentage of games resulting in a black win
0 - 100	0.1764	66 %	0.1824	69 %	0.1792	66 %
101 - 200	0.1856	66 %	0.1797	62 %	0.2121	71 %
201 - 300	0.1506	56 %	0.1766	67 %	0.1801	67 %
301 - 400	0.1984	74 %	0.2162	68 %	0.1815	68 %
401 - 500	0.2067	71 %	0.2252	70 %	0.1813	71 %
501 - 600	0.2087	72 %	0.1916	72 %	0.1999	74 %
601 - 700	0.1973	65 %	0.1874	68 %	0.1634	66 %
701 - 800	0.1938	65 %	0.1960	71 %	0.1906	72 %
801 - 900	0.1910	65 %	0.1750	66 %	0.2107	66 %
901 - 1000	0.2001	63 %	0.2071	75 %	0.1848	72 %

Table 16: Results of an initialized network against a random player

Perfect moves
Because we can not really see an improvement while playing against the random player with initialized weights, we can try to measure success in a different way. One way is to check how many times an optimal move was made during the games.

A network using 4 hidden nodes, 1000 examples per round of training and playing 1000 games makes a perfect move 88.60% of the time. A network with the same parameters, but using 5 hidden nodes instead of 4, performs almost the same with perfect moves 88.12% of the time. A network with 6 hidden nodes makes a perfect move 88.29% of the time.

As a comparison: when black is a random player, it makes a perfect move 63.40% of the time. This shows that we can speak of a network that is able to learn, because it is a fairly stronger player than a random player.

5.4 Conclusion

After testing various ways to train the network, we can see that the network is strongest when it was trained against another network, so when each player has its own network. We can now try this out on bigger sized boards and compare the results to the known winner when both player would play optimal. However, because we do not have the complete gametree (these grow exponentially), we can not calculate the average error so we have to find other ways to determine the quality of the network.

6 Temporal difference learning on a board with more squares than a 4×4 sized board

In the previous chapter we studied various methods of temporal difference learning on 4×4 sized boards. We concluded that some methods were better than others, and now we can try to make a network that learns how to play Othello on bigger sized boards. We try to find out if it is possible to make a strong Othello player on bigger sized boards while trying to find a way to distinguish good results from bad ones without having the complete gametree to compare the results to.

6.1 Playing on the 4×5 sized board

The best method we could find was having two different networks, one for each player. When we try this out on a 4×5 sized board, we get very different results than we had on the 4×4 sized board. The results can be seen in Table 17. It is visible that the white neural network is significantly stronger than the black one. Testing this network against a random player, see Table 18, we can see that the black player overall does not really grow as strong as one would expect.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**5 hiddens, 2000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**
0 - 100	22 %	29 %	34 %	26 %	30 %	25 %
101 - 200	11 %	14 %	26 %	18 %	21 %	27 %
201 - 300	3 %	1 %	4 %	3 %	0 %	11 %
301 - 400	0 %	0 %	1 %	0 %	0 %	0 %
401 - 500	0 %	1 %	1 %	0 %	0 %	0 %
501 - 600	0 %	3 %	2 %	0 %	0 %	0 %
601 - 700	0 %	0 %	3 %	0 %	0 %	0 %
701 - 800	0 %	0 %	7 %	0 %	0 %	0 %
801 - 900	0 %	0 %	6 %	0 %	0 %	0 %
901 - 1000	0 %	0 %	5 %	0 %	0 %	0 %

Table 17: Results of predictions for 4×5 sized boards with 28 inputs and varying parameters, each for 1000 played games. Two networks are used, one for each player. The percentage of games resulting in a win for black are shown.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**5 hiddens, 2000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**
0 - 100	37 %	30 %	33 %	32 %	23 %	28 %
101 - 200	34 %	32 %	21 %	20 %	25 %	23 %
201 - 300	21 %	22 %	22 %	28 %	25 %	13 %
301 - 400	23 %	24 %	20 %	18 %	23 %	28 %
401 - 500	21 %	20 %	21 %	23 %	30 %	26 %
501 - 600	32 %	34 %	25 %	22 %	23 %	18 %
601 - 700	21 %	16 %	24 %	29 %	20 %	19 %
701 - 800	23 %	25 %	22 %	17 %	28 %	21 %
801 - 900	20 %	23 %	25 %	24 %	25 %	16 %
901 - 1000	21 %	25 %	25 %	27 %	22 %	19 %

Table 18: Results of predictions for 4×5 sized boards with 28 inputs and varying parameters, each for 1000 played games. One network is used, only for the black player. The white player plays random. The percentage of games resulting in a win for black are shown.

One would expect to see black winning a lot more, especially since black would win the game when both players play optimal (see Table 1). It is interesting to understand why the network doesn't learn as fast or as much as the network for a 4×4 sized board.

Understanding the network
There are multiple reasons why the network does not perform as well as we would expect. One of these might be that even though black is the winner of the game when the game is played optimal, most of the endgames result in a win for white. This would mean that it is harder to find the optimal path and it would be no surprise that white wins a lot more often. On a 4×4 sized board, the game results in a white win 50.14% of the time, a black win 41.01% of the time and a draw in the remaining 8.85% of the time. This is different when looking at games played on a $4 \times 5\%$ sized board. Now, white wins 45.84% of the time, black wins 45.77% of the time and there is a draw in 8.39% of the endgames. These numbers do not give any more clearance about the difference in performance.

6.2 Playing on the 5×5 sized board

We can also look at games played on a 5×5 sized board. The results can be seen in Table 19. In the second column, we can clearly see that the white player has a great advantage over the black player because when both play random one would expect to see a result that would be closer to 50%. Comparing this to the third column, we can conclude that the network does learn to play stronger, still not winning a lot but signifficantly more than when we have a random player. In the last column we have one network playing against another network. As we saw previously on a 4×5 sized board that the white network seems to be learning the game faster, this seems also to be the case on a 5×5 sized board, though the best network here wins only 18.3% of the games, while on the 4×5 sized board black won mostly more than 20% of the games.

Number of hidden nodes	**random vs random**	**network vs random**	**network vs network**
4	6.1 %	18.3 %	3.4 %
5	3.7 %	16.0 %	6.5 %
6	7.9 %	17.5 %	3.9 %
7	3.2 %	17.7 %	3.6 %
8	6.8 %	14.8 %	6.1 %
9	3.9 %	17.4 %	5.3 %
10	3.7 %	18.3 %	5.5 %
11	3.4 %	15.7 %	3.5 %
12	2.9 %	17.4 %	3.6 %

Table 19: Percentage of games ending in a win for black under different circumstances, played using 1000 different examples for every networktraining over a 1000 played games.

6.3 Playing on the 6×6 sized board

It is interesting to examine even greater sized boards. We do not know the result of the game when played optimal from our own program (see Table 1), but it was proven that white wins the optimal game [1]. This is different from the 4×5 and the 5×5 sized boards. The results of a network playing against another network can be seen in Table 20 and the results of a network playing against a random player can be seen in Table 21. Note that they both contain the percentage of games resulting in a draw. These were never present before, all the games resulted in either a win for black or for white. It is remarkable that when there are two networks playing against each other, the games result in a draw most of the time. When a network plays against a random player, the percentage of games resulting in a win is higher but there are way more losses than when playing against a network. It seems that the network focusses on a win and not on a draw in these cases, while you could be more satisfied with almost 100% of the time a draw instead of losing $\frac{2}{3}$ of the time.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**	**8 hiddens, 1000 used examples**
0 - 100	37 %, 3 %	32 %, 2 %	25 %, 4 %	32 %, 2 %	26 %, 4 %	25 %, 3 %
101 - 200	21 %, 19 %	7 %, 58 %	3 %, 83 %	4 %, 87 %	2 %, 82 %	4 %, 79 %
201 - 300	23 %, 12 %	14 %, 45 %	0 %, 93 %	0 %, 100 %	2 %, 91 %	0 %, 93 %
301 - 400	6 %, 63 %	15 %, 33 %	1 %, 88 %	0 %, 100 %	0 %, 96 %	0 %, 91 %
401 - 500	5 %, 45 %	8 %, 20 %	1 %, 91 %	0 %, 100 %	2 %, 89 %	0 %, 94 %
501 - 600	4 %, 31 %	24 %, 11 %	0 %, 93 %	0 %, 100 %	1 %, 85 %	0 %, 92 %
601 - 700	13 %, 11 %	23 %, 9 %	0 %, 97 %	0 %, 100 %	0 %, 98 %	0 %, 79 %
701 - 800	10 %, 32 %	17 %, 9 %	1 %, 92 %	0 %, 100 %	2 %, 93 %	0 %, 88 %
801 - 900	12 %, 22 %	35 %, 2 %	0 %, 93 %	0 %, 100 %	0 %, 98 %	0 %, 68 %
901 - 1000	6 %, 42 %	29 %, 4 %	0 %, 96 %	0 %, 100 %	1 %, 95 %	0 %, 91 %

Table 20: Results of predictions for 6×6 sized boards with 44 inputs and varying parameters, each for 1000 played games. Two networks are used, one for each player. The percentage of games resulting in a win for black are shown left of the comma, and right the number of draws.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**	**8 hiddens, 1000 used examples**
0 - 100	48 %, 9 %	51 %, 2 %	50 %, 5 %	56 %, 4 %	54 %, 6 %	41 %, 9 %
101 - 200	35 %, 6 %	34 %, 6 %	39 %, 6 %	56 %, 4 %	37 %, 11 %	40 %, 6 %
201 - 300	40 %, 2 %	35 %, 1 %	38 %, 3 %	32 %, 4 %	34 %, 2 %	31 %, 1 %
301 - 400	41 %, 4 %	39 %, 4 %	43 %, 4 %	34 %, 7 %	34 %, 5 %	28 %, 7 %
401 - 500	29 %, 3 %	33 %, 3 %	34 %, 7 %	30 %, 5 %	38 %, 11 %	33 %, 6 %
501 - 600	31 %, 4 %	32 %, 4 %	28 %, 4 %	38 %, 6 %	30 %, 4 %	33 %, 5 %
601 - 700	31 %, 6 %	33 %, 4 %	36 %, 11 %	37 %, 2 %	43 %, 3 %	43 %, 6 %
701 - 800	25 %, 5 %	29 %, 7 %	33 %, 4 %	27 %, 8 %	41 %, 1 %	30 %, 5 %
801 - 900	39 %, 4 %	36 %, 3 %	45 %, 2 %	33 %, 8 %	39 %, 2 %	32 %, 4 %
901 - 1000	34 %, 6 %	38 %, 7 %	41 %, 3 %	37 %, 2 %	31 %, 5 %	34 %, 3 %

Table 21: Results of predictions for 6×6 sized boards with 44 inputs and varying parameters, each for 1000 played games. One network is used, only for the black player. The white player plays random. The percentage of games resulting in a win for black are shown left of the comma, and right the number of draws..

6.4 Playing on the 7×7 sized board

The results of games played on a 7×7 sized board are strikingly different from the ones in the previous section. As it seemed the case with the 4×5 sized board, we can see in Table 22 that the white network seems to be learning a lot faster than the black network, winning almost all the games. We do not know who wins the game when played optimal. There is also a difference between the 4×5 sized board and this one. When we look at Table 23, we can see that when a network plays against a random player, the percentage of wins is about 50% of the time, while on the smaller board it is about 25% of the time.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**	**8 hiddens, 1000 used examples**
0 - 100	29 %	29 %	26 %	36 %	21 %	26 %
101 - 200	2 %	1 %	1 %	25 %	1 %	0 %
201 - 300	0 %	2 %	30 %	33 %	1 %	0 %
301 - 400	0 %	5 %	31 %	19 %	1 %	0 %
401 - 500	0 %	6 %	18 %	5 %	0 %	0 %
501 - 600	0 %	6 %	27 %	4 %	0 %	0 %
601 - 700	0 %	10 %	17 %	1 %	0 %,	0 %
701 - 800	0 %	0 %	12 %	6 %	0 %	0 %
801 - 900	0 %	0 %	30 %	5 %	0 %	0 %
901 - 1000	0 %	0 %	25 %	6 %	0 %	0 %

Table 22: Results of predictions for 7×7 sized boards with 57 inputs and varying parameters, each for 1000 played games. Two networks are used, one for each player. The percentage of games resulting in a win for black are shown left of the comma, and right the number of draws.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**	**8 hiddens, 1000 used examples**
0 - 100	59 %	49 %	59 %	61 %	54 %	56 %
101 - 200	43 %	45 %	52 %	53 %	57 %	50 %
201 - 300	58 %	46 %	40 %	39 %	43 %	38 %
301 - 400	41 %	43 %	45 %	46 %	50 %	48 %
401 - 500	45 %	54 %	48 %	44 %	53 %	46 %
501 - 600	52 %	46 %	38 %	52 %	44 %	47 %
601 - 700	55 %	47 %	41 %	46 %	47 %	47 %
701 - 800	51%	48 %	45 %	51 %	55 %	53 %
801 - 900	38 %	53 %	42 %	43 %	43 %	37 %
901 - 1000	37 %	50 %	47 %	51 %	52 %	53 %

Table 23: Results of predictions for 7×7 sized boards with 57 inputs and varying parameters, each for 1000 played games. One network is used, only for the black player. The white player plays random. The percentage of games resulting in a win for black are shown left of the comma, and right the number of draws..

6.5 Playing on the 8×8 sized board

Finally, we consider the standard Othello board, that is sized 8×8. The game is not solved yet, but is expected to result in a draw after perfect play [11]. Note that only the first column sometimes has a draw as a result (Table 24). The results vary significantly depending on the amount of hidden nodes used. Considering Table 25, it is also remarkable that the difference between playing against a network or playing against a random player are not that different than as seen with smaller sized boards.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**	**8 hiddens, 1000 used examples**
0 - 100	48 %, 2 %	42 %	21 %	33 %	35 %	46%
101 - 200	43 %, 5 %	36 %	2 %	28 %	46 %	41 %
201 - 300	42 %, 9 %	45 %	0 %	29 %	41 %	38 %
301 - 400	30 %, 11 %	47 %	0 %	24 %	44 %	45 %
401 - 500	32 %, 11 %	40 %	0 %	30 %	36 %	35 %
501 - 600	55 %, 22 %	36 %	0 %	29 %	34 %	55 %
601 - 700	27 %, 25 %	38 %	1 %	23 %	30 %,	36 %
701 - 800	14 %, 8 %	38 %	0 %	22 %	25 %	53 %
801 - 900	57 %, 8 %	43 %	0 %	18 %	21 %	26 %
901 - 1000	66 %, 11 %	49 %	0 %	21 %	19 %	52 %

Table 24: Results of predictions for 8×8 sized boards with 72 inputs and varying parameters, each for 1000 played games. Two networks are used, one for each player. The percentage of games resulting in a win for black are shown left of the comma, and right the number of draws.

Part of the games used for the calculation	**4 hiddens, 1000 used examples**	**4 hiddens, 2000 used examples**	**5 hiddens, 1000 used examples**	**6 hiddens, 1000 used examples**	**7 hiddens, 1000 used examples**	**8 hiddens, 1000 used examples**
0 - 100	57 %	48 %	40 %	54 %	49 %	56 %
101 - 200	57 %	54 %	58 %	53 %	52 %	56 %
201 - 300	63 %	56 %	65 %	60 %	57 %	42 %
301 - 400	45 %	48 %	57 %	51 %	51 %	46 %
401 - 500	48 %	59 %	59 %	53 %	56 %	56 %
501 - 600	59 %	53 %	59 %	58 %	59 %	43 %
601 - 700	55 %	56 %	56 %	55 %	52 %	49 %
701 - 800	56 %	70 %	54 %	52 %	57 %	59 %
801 - 900	56 %	67 %	55 %	57 %	54 %	50 %
901 - 1000	61 %	56 %	55 %	59 %	51 %	56 %

Table 25: Results of predictions for 8×8 sized boards with 72 inputs and varying parameters, each for 1000 played games. One network is used, only for the black player. The white player plays random. The percentage of games resulting in a win for black are shown left of the comma, and right the number of draws..

6.6 Conclusion

It seems that we are able to learn the computer to play Othello with temporal difference learning. However, it is quite difficult to ensure a great player. Most of the time, the white player is siginificantly stronger than the black player. The reason for this could have multiple causes. In future work, this could be researched to learn from the differences. To enhance the networks, understanding of the differences is needed.

7 Conclusion and future work

In this study we have examined different sized boards of Othello. For smaller sized boards it is not hard to understand the strategy for optimal play. Calculating the winner when a game is played optimal has proven to be more difficult for greater sized boards, because of the huge size of the gametree. We have tried to predict the outcome of the game using a neural network. This went quite well for 4×4 sized boards, where we could compare the results of the predicted outcome with the known outcome.

Because we do not have the known outcome of games played on boards greater than 4×4, we tried to use temporal difference learning to make a strong network that trains on information gained while playing multiple games. After trying multiple variations of elementary temporal difference learning on a 4×4 sized board we could decide what the best method was, because it could be compared to the known outcome. This method was then used for greater sized boards. Using temporal difference learning on greater sized boards gave varying results. More knowledge of the structure of the gametree is needed to enhance the networks.

There are numerous studies that can be done to continue this research. For example, the program used to determine the winner of the optimal game could be enhanced to be able to complete Table 1. Something else is to try to understand the gametrees of the greater sized boards to enhance the neural networks used in the last chapters, to make a really strong computer player. The type of temporal difference learning that was used in this study could be adjusted so that it is temporal difference learning as described by Richard Sutton and Andrew Barto [7]. This would mean that the network would not only train itself after a game is finished, but already during the game.

It could also be interesting to explore the number (and especially kind of) input nodes for $m \times n$ boards with $m > 4$ and $n > 4$. We only studied this for the 4×4 sized board but have not experimented with this on bigger sized boards. However, the bigger the board the more different kind of squares there are in terms of importancy (i.e., closeness to the corner).

In the end, the goal is to solve Othello played on an 8×8 sized board, which has not been done yet.

References

[1] Feinstein, J. Amenor wins world 6x6 Championships! Newsletter of the British Othello Federation (1993) Available from: http://www.britishothello.org.uk/fbnall.pdf. [Accessed: 21/08/2015]

[2] GamesCrafters. *Original Reversi* [Online] Available from: http://gamescrafters.berkeley.edu/games.php?game=othello. [Accessed: 10/06/2015]

[3] Heaton, J. (2008). *Introduction to neural networks with Java. Heaton* Location: Heathon Research. Inc.

[4] Heaton research. *The number of hidden layers* [Online] Available from: http://www.heatonresearch.com/node/707. [Accessed: 24/06/2015]

[5] Iwata, S. and Kasai, T. The Othello game on an $n \times n$ board is PSPACE-complete. Theoretical Computer Science 123 (1994) 329–340.

[6] Rosenbloom, P S. A world-championship-level Othello program. Artificial Intelligence 19 (1982) 279–320.

[7] Sutton, R. and Barto, A. (1988) *Reinforcement Learning*, MIT Press. (1998) 133–140.

[8] Tothello. *Solving* 6×6 *normal Othello game* [Online] Available from: http://www.tothello.com/html/solving_the_6x6_normal.html. [Accessed: 28/06/2015]

[9] Van der Ree, M. and Wiering, M. *Reinforcement learning in the game of Othello: Learning against a fixed opponent and learning from self-play* 2013. Available from: http://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/paper-othello.pdf

[10] Wikipedia. *Artificial neural network* [Online] Available from: https://en.wikipedia.org/wiki/Artificial_neural_network. [Accessed: 24/06/2015]

[11] Wikipedia. *Computer Othello* [Online] Available from: http://en.wikipedia.org/wiki/Computer_Othello. [Accessed: 21/05/2015]

[12] Wikipedia. *Reversi* [Online] Available from: http://nl.wikipedia.org/wiki/Reversi. [Accessed: 21/05/2015].