THE UNIVERSITY OF HONG KONG  
FACULTY OF ENGINEERING  
DEPARTMENT OF COMPUTER SCIENCE  
FINAL YEAR PROJECT  

Playing Othello by Deep Learning Neural Network  

FINAL REPORT  

Team Members  
CHAN Lok Wang : 3035072777  
YIP Tsz Kwan : 3035068685  

Prepared by  
Chan Lok Wang  

Supervised by  
Dr. K.P. CHAN  

15th April, 2017
Abstract

Board game implementation has long been a way of showing the ability of AI. Among different board games, Othello is one of the popular subjects due to its simple rules and well-defined strategic concepts. Today, majority of the Othello programs at a master level still require expert knowledge from their programmers. The decision on moves of a computer Othello is based on the game tree with the values of the nodes. The value of each node is evaluated by a pre-defined evaluation function, which is hard-coded by the programmers.

In this project, we aim to use a machine learning technique, deep learning neural network, to play Othello. With this technique, the computer can be trained to play Othello without any human logic. The targeted winning rate of the computer Othello to be developed is 50% when playing against a moderate opponent, and 35% when playing against a strong opponent. The project is yet to achieve its goal but expected to obtain a better outcome with more data collection.
Acknowledgements

I would like to thank our supervisor, Dr. K.P. Chan, and our English course instructor, Dr. Ken Ho, for their advice and guidance. I would also like to thank Ng Argens and Yu Kuai for sharing the game move records for the neural network training.
Acronyms

AI: artificial intelligence
CPU: central processing unit
GPU: graphical processing unit
GUI: graphical user interface
MCTS: Monte Carlo tree search
UI: user interface
# Table of Contents

Abstract ........................................................................................................................................... ii

Acknowledgements (optional) ........................................................................................................ iii

Acronyms (optional) ....................................................................................................................... iv

List of Figures ...................................................................................................................................... vii

List of Tables .................................................................................................................................... viii

Chapter 1 - Introduction  .............................................................................................................. 1

1.1 Motivations .............................................................................................................................. 1

1.2 Objectives and Scope ................................................................................................................ 1

1.2.1 Objectives ................................................................................................................................ 1

1.2.2 Scope ...................................................................................................................................... 1

Chapter 2 - Project Background .................................................................................................. 3

2.1 The game of Othello ............................................................................................................... 3

2.2 Computer game algorithm ....................................................................................................... 3

2.3 Data-driven developed evaluation function ............................................................................. 3

Chapter 3 - Design and Procedures .............................................................................................. 5

3.1 Software Development Cycle ................................................................................................. 5

3.2 Training Data Collection and Processing ............................................................................... 6

3.2.1 Data Collection ................................................................................................................... 6

3.2.2 Data Processing .................................................................................................................. 7

3.3 Algorithms .............................................................................................................................. 8

3.3.1 Neural Network .................................................................................................................. 8

3.3.2 Search Algorithm ............................................................................................................... 9

3.4 Game program implementation ............................................................................................... 10
3.5 Testing ...................................................................................................................... 10

3.5.1 Gameplay testing ................................................................................................... 10

Chapter 4 - Results ......................................................................................................... 12

4.1 Network Performance ............................................................................................... 12

4.2 Gameplay Performance .............................................................................................. 26

4.3 Limitations ................................................................................................................ 27

   4.3.1 Slower computational time for CPU ................................................................. 27

   4.3.2 Size of training data .......................................................................................... 27

Chapter 5 - Conclusion and Recommendation for Future Work .................................... 28

References ....................................................................................................................... 29
List of Figures

Figure 1 An example of a neural network with three layers ........................................... 8
Figure 2: A game search tree with minimax algorithm ...................................................... 9
Figure 3 Loss trend for N0-1 .......................................................................................... 16
Figure 4 Loss trend for N0-2 .......................................................................................... 16
Figure 5 Loss trend for N0-4 .......................................................................................... 16
Figure 6 Loss trend for N0-3 .......................................................................................... 16
Figure 7 Loss trend for N0-7 .......................................................................................... 18
Figure 8 Loss trend for N0-8 .......................................................................................... 18
Figure 9 Loss trend for N0-9 .......................................................................................... 19
Figure 10 Loss trend for N1-1 .......................................................................................... 21
Figure 11 (Left): Loss trend for N1-2-1 ................................................................. 21
Figure 12 (Right): Loss trend for N1-2-2 ................................................................. 21
Figure 13 Loss trend for N2-1 .......................................................................................... 24
Figure 14 Loss trend for N2-2 .......................................................................................... 24
Figure 15 Loss trend for N2-3 .......................................................................................... 25
Figure 16 Loss trend for 2-4 ............................................................................................ 25
List of Tables

Table 1 - Level of difficulties of The Othello ................................. 11
Table 2 Model Architecture used in the project ................................ 14
Table 3 Parameters for training Networks N0-1 to N0-4 ....................... 15
Table 4 Parameters for training Networks N0-7 to N0-9 ....................... 17
Table 5 Parameters for training Networks N1-1, N1-2-1 and N1-2-2 .......... 20
Table 6 Parameters for training Networks N1-1, N1-2-1 and N1-2-2 .......... 23
Table 7 Win-loss-draw ratio for selected networks playing against level 10 and level 15 opponent ................................................................. 26
Chapter 1 - Introduction

1.1 Motivations
This project aims to apply the technique of deep learning neural network to produce an evaluation function for the game Othello, such that the produced evaluation function will be solely data-driven without human interpretation. Othello is chosen as the size and the complexity of this game is suitable for a one-year long project with limited resources.

1.2 Objectives and Scope

1.2.1 Objectives
The objective of this project is to develop a desktop computer Othello with the technique of deep learning neural network and the following attributes:

- The game board configuration size being 8 x 8;
- A winning rate of 50% when playing against a moderate (computer) opponent;
- A winning rate of 35% when playing against a strong (computer) opponent; and
- A user-friendly user interface (UI) for easy playing

While the ultimate outcome is to deliver the program mentioned above, the focus of this project is to construct the evaluation function via the value network without human logic.

1.2.2 Scope
In this project, we plan to develop a program playing Othello with the following specifications:

- The board size of the game being 8 x 8;
- Data-driven evaluation function being constructed by a value network;
- Selection of the next best move using minimax algorithm; and
- Central processing units (CPUs), instead of graphical processing units (GPUs), being used for the deep learning process
The following features will be included if time is allowed:

- A graphical user interface for the game;
- Selection of the best next moves being implemented with policy network.

The following features will not be considered in this project:

- A mobile application of the game; and
- Improvement in speed

1.3 Contribution of the project

The contributions of each member in this project is listed below:

Chan Lok Wang:

- Design of the CNN model structure for training process
- Neural Network training process and performance review
- Implementation of the move search algorithm to be used in the Othello game program
- Bug fixing of the game program
- Early contribution to the data collection process

Yip Tsz Kwan:

- Data Collection and Processing
- Gameplay Performance testing
- Code review
Chapter 2 - Project Background

2.1 The game of Othello

Othello is usually played on an 8x8 board, and the players are traditionally labeled black and white. The game starts from the board configuration with the four centered squares occupied. The black plays first, and the game continues until there are no legal moves left to play. A legal move requires players to place a disk on an empty square and flipping disks with another color to their color. Opponent disks are flipped, when they are sandwiched between the newly placed disk and another disk of the same color, along a line in any direction (horizontal, vertical, or diagonal).

2.2 Computer game algorithm

A classical way of building a computer game program has two main parts, which are the search algorithm and evaluation function. [1] The idea of search algorithm is to search a game tree of possible legal moves of the current state so that it goes to a move with the best score to win. If the tree is small, all nodes can be examined so that the score can be calculated from the leaf nodes using minimax algorithm.

However, it is often that the computational power is not enough to search the entire tree in a given time and space. In such case, an evaluation function approximates the scores of the current game state and returns the prediction of the scores. The evaluation function can either be hand-coded or constructed by the program itself.

2.3 Data-driven developed evaluation function

Hand-coded evaluation function obviously requires human efforts in analyzing different board state of the game. This limits the performance of the program if the hand-coded algorithm is based on an incomplete analysis of the game. Most state-of-the-art Othello programs of champion level, such as Saio, Cyrano, Cassio, Wzwbra and Heracles, requires expert knowledge from their programmers. [2]

Self-constructed evaluation function is another approach in tackling the limitation. In this project, we aim to construct an evaluation function by just using a set of game records without human analysis over the strategies of the game. Deep learning neural
network is a tool enabling the approach. If this is successful, it is hoped that the similar approach can be applied to other kinds of turn-based game and thus reduce the effort of developing an evaluation function specifically for a particular kind of game.
Chapter 3 - Design and Procedures

The approach and methodology of this project will be discussed in this section. The software development cycle will first be discussed, followed by the training data collection. The algorithms used, including the search algorithms, and neural network, will be introduced. The test specification can also be found in this section.

3.1 Software Development Cycle

For this project, a combination of the waterfall and agile model will be adopted to allow flexibility for modifying and adjusting the development process of evaluation function by deep learning neural network training.

A program for the game Othello will first be developed with a pre-defined evaluation function. The pre-defined evaluation function has a simple logic with its only mission of ensuring the developed program is fully functional and the program can allow new modules of evaluation function integration.

The evaluation function for the game Othello will then be developed by deep learning neural network. Networks trained under different set of hyperparameters and training data will be tested by incorporating into the existing game program developed. The test result will be used for reviewing the existing setting of the training process. Adjustments and modification to the training process will be made after each round of testing.
3.2 Training Data Collection and Processing

3.2.1 Data Collection

At the beginning stage of the project, 100 sets of training data are to be prepared from our group with the format as follows:

Each move in a game is represented by the string in the format below,\n\[ c_i \ x_i \ y_i \]
where \( c_i \) is an integer representing the color of the disc in the \( i \)th move, with “1” as black and “0” as white. \((x_i, y_i)\) represents the position of the disc on the board in the \( i \)th move, with \( 0 \leq x_i \leq 7 \) as the column position and \( 0 \leq y_i \leq 7 \) as the row position.

The game move records are collected for obtaining different board configurations regarding the winning probability of the game. Detailed process on processing these game move records into suitable form for neural network training is described in the following Section 3.2.2.

Training data were obtained through the following ways:

- 25 games with human (black) playing against computer (white)
- 25 games with human (white) playing against computer (black)
- 50 games with computer playing against computer

The 100 sets of game were obtained through the mentioned three ways such that the both ordinary human players’ logic and computers’ evaluation when playing the game Othello can be captured.

The computer Othello chosen is “The Othello”, a free application offered in both Apple’s App Store and Google’s Play Store. As there are 30 thinking levels for the computer, it allows us to get more board configurations in different games by selecting different levels, achieving the main goal of the training data collection process.

An addition of 100 sets of game move records were obtained from other groups working
on the same project in “Playing Othello by deep learning neural network”.

At the middle stage of neural network training, 685 more sets of game were collected from “2013 Othello Word Cup” [3] and “World Othello Championship 2016” [4] and added to the training data set to eliminate the effect of overfitting observed.

3.2.2 Data Processing

Each step in a game move record will be transformed to the corresponding board configuration. The board configuration is represented by a string containing 100 characters, with “0” representing the appended border cell, “1” representing a cell occupied by a white disc, “2” representing a cell being occupied by a black disc and “3” representing an empty cell.

Additional border cells with value “0” are appended to the board to help the network to capture the features at the border area.

Each board is assigned a value depending on the result of the games.

i. If the white player wins a game, every boards from that game move record will be awarded +1 score.

ii. If the white player loses a game, every boards from that game move record will get a penalty of -1 score.

iii. If the game ends with a draw, no effect will be made to the boards for that game.

Repeat the steps for each game move records and a list of board configurations will be assigned with a total score. The total score of each configuration will be divided by the total number of game move records to give out the probability of winning chance at each configuration.

To improve the calculation of winning probability for board configurations, the symmetric property of the board is taken into consideration. It is achieved by applying the above steps with the following rule:

i. For each board configurations assigned with a value v, the other 3 equivalent rotated board configurations will also be assigned with the value v.

The rest of the processing process is the same as mentioned above.
3.3 Algorithms

3.3.1 Neural Network

Neural network is a way of information processing. There are three types of layers in a neural network - input layer, hidden layer, and the output layer. The goal of training a supervised neural network is to minimize the difference between the input layer’s label and output layer’s result.

The input layer with defined number of nodes accept values with their corresponding labels and calculates and produces the result at the output node(s) in the output layer. The input layer and output layer is connected by nodes in the hidden layer(s) (see Figure 1). A deep learning neural network is neural network with more than one hidden layer.

![Figure 1 An example of a neural network with three layers.](image)

There are two input nodes in the input layer. The input nodes are connected to three nodes in the hidden layer with assigned weight. The hidden layer’s nodes are connected to the only output layer’s node. The activation function of this example is

\[ y = \frac{1}{1 + e^{-x}}. \]

A bias, \( b_i \), is assigned to every node \( i \) in the hidden layer and the output layer as the threshold. A weight, \( w_{ij} \), is also assigned to each edge connecting the two nodes \( i \) and \( j \) in different layers. The value of each node, \( v \), is calculated by the formula

\[ v = f \times \sum b_i w_{ij} \]

where \( f \) is an activation function. This value assignment step repeats until the output layers’ nodes are reached. This process is known as forward propagation.
If the final output does not align with the labelled value, the set of weights and bias will be updated by the process “back-propagation”. The adjustment of the weights and biases are determined by the loss function and the optimizer.

A loss function will be defined to evaluate the performance of the neural network. In the case of this project, the loss function needs to calculate the difference between the labeled value and the predicted value. Since it is network for regression, mean squared error will be used as the loss function.

### 3.3.2 Search Algorithm

#### 3.3.2.1 Minimax Algorithm

A brute-force approach of minimax algorithm will be equipped with the game program in searching for the best next move, where the algorithm is commonly applied to turn-based games without any random elements.

Weights are assigned to all the board configurations at the leaf nodes of the game tree. At each round, the two players, known as MAX and MIN, need to select the moves that favors them winning – the MAX player should select the node with the highest weight in the game tree at the MAX level, while the MIN player should select the node with the smallest weight in the game tree at the MIN level. An example is shown in Figure 4.

![Figure 2: A game search tree with minimax algorithm with final evaluation as 5](image)
3.3.2.2 Alpha-beta Pruning

The alpha-beta pruning algorithm will be implemented for better efficiency in searching for the next best move. Alpha-beta pruning is able to skip examining some part of the game tree in which these branches are confirmed having moves leading to unfavorable states. [5]

3.4 Game program implementation

We decided to obtain a command line program for Othello written in Python from Github contributed by JaimieMurdock [6] to allow more time in developing the evaluation function using neural network. The program package comes with two Engines – human mode and random mode, which allow a user to select the mode of action for the two players in the game.

There are bugs in the package that can lead to wrong result of a game. Bugs identified and fixed are as follow:

1. Incorrect legal moves detected for border discs
2. Incorrect flips for some moves, leading to a wrong board

A new engine that implement the minimax algorithm is developed by our team for gameplay performance testing.

3.5 Testing

3.5.1 Gameplay testing

The computer Othello developed will be tested against two types of opponents, moderate computer opponents and strong computer opponents. The application “The Othello” was chosen for testing. This is an application available on both Google’s Play Store and Apple’s App Store. The level of difficulty for this application depends on the number of playouts of the Monte-Carlo tree search algorithm according to the reply email from the developer of the application. 30 levels of difficulty were set in this application, with Level 1 being the easiest and Level 30 being the most difficult. These levels were grouped to five levels of difficulties as shown in Table 1.
Table 1 - Level of difficulties of The Othello. Level 1 to Level 4 was considered as "very easy". Level 5 to Level 8 was considered as "easy". Level 9 to Level 14 was considered as "moderate". Level 15 to Level 22 was considered as "strong" and Level 23 to Level 30 was considered as "extremely strong"

In this project, level 10 and level 15 has been chosen as moderate computer opponents and strong computer opponents for gameplay testing respectively. The duration for each round of game is estimated to be around 15 minutes. The number of gameplay test was set to be 5 for each types of opponents such that there would be enough time allowed for performance review and adjustment to the project.

When playing against a moderate computer opponent, the targeted winning rate is 50%, while the targeted winning rate for playing against a strong computer opponent is 35%.
Chapter 4 - Results

As stated in Chapter 3.4, an executable command line program has been implemented for the use of gameplay performance testing. Trained networks were selected to perform gameplay testing by examining the training logs. Since the value network is trained for regression, “loss” and “validation loss” are the two indicators that we will observe.

A total of 13 networks for estimating the winning probability of the game have been trained using different sets of data under a few variances of network model structures. Despite their statistics figures measuring the quality of these networks, among the 13 networks, only one of them is considered having a similar strength with a moderate player when performing test with selected Othello game application.

4.1 Network Performance

Three kinds of model structure have been defined during the process of exploring and training a suitable network as an evaluation function for the game program. The structure of the three model is listed in Table 2.

<table>
<thead>
<tr>
<th>CNN architecture (Layers in order)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 1. Convolution Layer</td>
<td>- 5x5 filters are applied before 3x3 filters to capture favorable features at border and unfavorable features at second row and column of the board</td>
</tr>
<tr>
<td>- input layer, shape 10 x 10</td>
<td></td>
</tr>
<tr>
<td>- 32 5x5 filters, 0 padding to give output size as input size</td>
<td></td>
</tr>
<tr>
<td>- Activation function: reLU</td>
<td></td>
</tr>
<tr>
<td>2. Convolution Layer</td>
<td></td>
</tr>
<tr>
<td>- 16 5x5 filters, 0 padding to give output size as input size</td>
<td></td>
</tr>
<tr>
<td>- Activation function: reLU</td>
<td></td>
</tr>
<tr>
<td>3. Pooling Layer</td>
<td></td>
</tr>
<tr>
<td>- Size 2x2, strides 2x2</td>
<td></td>
</tr>
<tr>
<td>4. Convolution Layer</td>
<td></td>
</tr>
<tr>
<td>- 16 3x3 filters, 0 padding to give output size as input size</td>
<td></td>
</tr>
<tr>
<td>- Activation function: reLU</td>
<td></td>
</tr>
<tr>
<td>5. Convolution Layer</td>
<td></td>
</tr>
<tr>
<td>- 16 3x3 filters, 0 padding to give output size as input size</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Pooling Layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Size 2x2, strides 2x2</td>
</tr>
<tr>
<td>7. Convolution Layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16 3x3 filters, 0 padding to give output size as input size</td>
</tr>
<tr>
<td></td>
<td>Activation function: reLU</td>
</tr>
<tr>
<td>8. Pooling Layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Size 2x2, strides 2x2</td>
</tr>
<tr>
<td>9. Fully connected Layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16 modes</td>
</tr>
<tr>
<td></td>
<td>Activation function: reLU</td>
</tr>
<tr>
<td>10. Fully connected Layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output layer</td>
</tr>
<tr>
<td></td>
<td>1 nodes</td>
</tr>
<tr>
<td></td>
<td>Linear activation function</td>
</tr>
</tbody>
</table>

M2  
1. Convolution Layer  
   - input layer, shape 10 x 10  
   - 32 5x5 filters, 0 padding to give output size as input size  
   - Activation function: reLU  
2. Convolution Layer  
   - 16 3x3 filters, 0 padding to give output size as input size  
   - Activation function: reLU  
3. Convolution Layer  
   - 16 3x3 filters, 0 padding to give output size as input size  
   - Activation function: reLU  
4. Dropout Layer  
   - Rate: 0.25  
5. Fully connected Layer  
   - 16 modes  
   - Activation function: reLU  
6. Fully connected Layer  
   - Output layer  
   - 1 nodes  
   - Linear activation function  

- 5x5 filters are applied before 3x3 filters to capture favorable features at border and unfavorable features at second row and column of the board  
- Dropout layer is added to control the problem of overfitting

M3  
1. Convolution Layer  
   - input layer, shape 10 x 10  

- 5x5 filters are applied before 3x3 filters to capture favorable features
<table>
<thead>
<tr>
<th>2. Pooling Layer</th>
<th>Size 2x2, strides 2x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Convolution Layer</td>
<td>16 3x3 filters, 0 padding to give output size as input size</td>
</tr>
<tr>
<td></td>
<td>Activation function: reLU</td>
</tr>
<tr>
<td>4. Pooling Layer</td>
<td>Size 2x2, strides 2x2</td>
</tr>
<tr>
<td>5. Convolution Layer</td>
<td>16 3x3 filters, 0 padding to give output size as input size</td>
</tr>
<tr>
<td></td>
<td>Activation function: reLU</td>
</tr>
<tr>
<td>6. Pooling Layer</td>
<td>Size 2x2, strides 2x2</td>
</tr>
<tr>
<td>7. Fully connected Layer</td>
<td>16 modes</td>
</tr>
<tr>
<td></td>
<td>Activation function: reLU</td>
</tr>
<tr>
<td>8. Fully connected Layer</td>
<td>Output layer</td>
</tr>
<tr>
<td></td>
<td>1 nodes</td>
</tr>
<tr>
<td></td>
<td>Linear activation function</td>
</tr>
</tbody>
</table>

Table 2 Model Architecture used in the project

Our team started working on fitting data to M1 at the beginning of the phase of neural network training, using only the 200 sets of gameplay records collected (D0) from our team’s and other teams’ playing.

The neural network was implemented with a Python interface library Keras. Keras can use Theano or TensorFlow as the backend library [7], which facilitates the implementation of deep learning neural network. In this project, Theano was chosen as the backend library for neural network training.

It produced a network with serious problem of overfitting when analyzed with the “loss” and “validation loss”. We suspected that it was caused by insufficient data and had our first data expansion (D1) by collecting game records from different world competitions. [3][4]

The total number of game move records became 885 and the number of board
configurations increased to 43924.

4 networks were trained under M1 model structure using different variance of D1. The details configurations and results for these 4 networks are shown in Table 3, Figure 3 - 6.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Optimizer</th>
<th>Data used</th>
<th>Number of boards for training</th>
<th>Shuffle data after each epoch?</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N0-1</td>
<td>SGD</td>
<td>Training set - Data set D1, with total score as the label - express border cells as “3” and empty cells as “0” Validation set - 10 % of the total boards</td>
<td>Training set - 39532 Validation set - 4392</td>
<td>True</td>
<td>Exclude D0 set to experiment if data set with high quality gameplay records can produce better results</td>
</tr>
<tr>
<td>N0-2</td>
<td>SGD</td>
<td>Training set - Data set D1, with average score as the label - express border cells as “3” and empty cells as “0” Validation set - 10 % of the total boards</td>
<td>Training set - 39532 Validation set - 4392</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>N0-3</td>
<td>SGD</td>
<td>Training set - Data set D1, with total score as the label - express border cells as “0” and empty cells as “3” Validation set - 10 % of the total boards</td>
<td>Training set - 39532 Validation set - 4392</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>N0-4</td>
<td>SGD</td>
<td>Training set - Data set D1 excluding D0, with total score as the label - express border cells as “0” and empty cells as “3” Validation set - 10 % of the total boards</td>
<td>Training set - 30462 Validation set - 3385</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 Parameters for training Networks N0-1 to N0-4
From the figures, we observed that these networks suffered from different degrees of overfitting. While the significant factor leading to overfitting could not be confirmed by reading the patterns from figures 3-6, we suspected that the data was still insufficient for the network and also for reflecting the true winning probability of the game. Therefore, consideration of the symmetric property of the game boards was added in processing the data for training (D2), details of this implementation is written in Chapter 3.2.2.

3 networks were then trained under M1 model structure using different variance of D2. The optimizer was switched to Adam because two networks using SGD were found to be trapped by a local minimum. The details of the two networks were omitted from this report.
The details configurations and results for these 4 networks are shown in Table 4, Figure 7 – 9.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Optimizer</th>
<th>Data used</th>
<th>Number of boards for training</th>
<th>Shuffle data after each epoch?</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N0-7</td>
<td>Adam</td>
<td>Training set - Data set D2, with total score as the label - express border cells as “0” and empty cells as “3” Validation set - 40 % of the total boards</td>
<td>Training set - 103687 Validation set - 69125</td>
<td>True</td>
<td>-</td>
</tr>
<tr>
<td>N0-8</td>
<td>Adam</td>
<td>Training set - Data set D2 excluding D0, with average score as the label - express border cells as “3” and empty cells as “0” Validation set - 50 % of the total boards</td>
<td>Training set - 66785 Validation set - 66784</td>
<td>True</td>
<td>- Exclude D0 set to experiment if data set with high quality gameplay records can produce better results</td>
</tr>
<tr>
<td>N0-9</td>
<td>Adam</td>
<td>Training set - Data set D2 excluding D0, with normalized score as the label - express border cells as “0” and empty cells as “3” Validation set - 50 % of the total boards</td>
<td>Training set - 66785 Validation set - 66784</td>
<td>True</td>
<td>- Exclude D0 set to experiment if data set with high quality gameplay records can produce better results</td>
</tr>
</tbody>
</table>

Table 4 Parameters for training Networks N0-7 to N0-9
Figure 7 Loss trend for N0-7

Figure 8 Loss trend for N0-8
As the problem of overfitting persisted by observing figures 7 - 9, we switched to M2 model structure, in which it has a smaller size that requires less data than M1 model structure. 3 networks were trained under M2 model structure using different variance of D2. N1-2-1 and N1-2-2 were to be tested together with N1-2-1 being responsible for predicting the winning probability of the board for the first 30 rounds of the game and N1-2-2 responsible for the last 30 rounds of the game. The details configurations and results for these 2 networks are shown in Table 5, Figure 10 – 12.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Optimizer</th>
<th>Data used</th>
<th>Number of boards for training</th>
<th>Shuffle data after each epoch?</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1-1</td>
<td>SGD</td>
<td>Training set - Data set D2 excluding D0, with normalized score as the label - express border cells as “0” and empty cells as “3” Validation set - 50 % of the total boards</td>
<td>Training set - 66785 Validation set - 66784</td>
<td>True</td>
<td>- Exclude D0 set to experiment if data set with high quality gameplay records can produce better results</td>
</tr>
</tbody>
</table>
| N1-2-1 | SGD  
- learning rate: 0.001  
- decay: 1e-7  
- momentum: 0.9 | Training set  
- Data set D2 with constraint accepting only boards for the first 30 round of the game, with normalized score as the label  
- express border cells as “3” and empty cells as “0”  
Validation set  
- 25 % of the total boards | Training set  
- 51118  
Validation set  
- 17040 | True |
| N1-2-2 | SGD  
- learning rate: 0.001  
- decay: 1e-7  
- momentum: 0.9 | Training set  
- Data set D2 with constraint accepting only boards for the last 30 round of the game, with normalized score as the label  
- express border cells as “3” and empty cells as “0”  
Validation set  
- 25 % of the total boards | Training set  
- 78490  
Validation set  
- 26164 | True |

Table 5 Parameters for training Networks N1-1, N1-2-1 and N1-2-2
From the Figures, less degree of overfitting was observed from network N1-1, N1-2-1 and N1-2-2 when compared to the previous networks. However, their gameplay performance did not improve as much. As it was difficult to figure out whether overfitting or the incompleteness of data on reflecting the winning probability was the real cause of poor performance in gameplay test, we tried to eliminate the effect of overfitting first by adding pooling layers to M2 model, which was the M3 model structure as stated in Table 2.
4 networks were trained under M3 model structure using different variance of D1. D1 was used because we were on a tight schedule and the time estimated to finish training using D2 was not sufficient within the project’s schedule. The details configurations and results for these 4 networks are shown in Table 6, Figure 13 – 16.

<table>
<thead>
<tr>
<th>Networks</th>
<th>Optimizer</th>
<th>Data used</th>
<th>Number of boards for training</th>
<th>Shuffle data after each epoch?</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2-1</td>
<td>SGD</td>
<td>Training set - Data set D1, with total score as the label</td>
<td>Training set - 39532</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- express border cells as “3” and empty cells as “0” Validation set - 10 % of the total boards</td>
<td>Validation set - 4392</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2-2</td>
<td>SGD</td>
<td>Training set - Data set D1, with average score as the label Validation set - 10 % of the total boards</td>
<td>Training set - 39532</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- express border cells as “3” and empty cells as “0” Validation set - 10 % of the total boards</td>
<td>Validation set - 4392</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2-3</td>
<td>SGD</td>
<td>Training set - Data set D1, with total score as the label Validation set - 10 % of the total boards</td>
<td>Training set - 39532</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- express border cells as “0” and empty cells as “3” Validation set - 10 % of the total boards</td>
<td>Validation set - 4392</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2-4</td>
<td>SGD</td>
<td>Training set - Data set D1 excluding D0, with total score as the label Validation set - 10 % of the total boards</td>
<td>Training set - 30462</td>
<td>True</td>
<td>Exclude D0 set to experiment if data set with high quality gameplay records can</td>
</tr>
<tr>
<td>Validation set</td>
<td>- 10 % of the total boards</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>produce better results</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6 Parameters for training Networks N2-1 – N2-4
Figure 13 Loss trend for N2-1

Figure 14 Loss trend for N2-2
N2-2 was the best among other networks in which both its loss and val_loss dropped in a similar trend without much fluctuations. It is also the model that stand out from the Gameplay Performance test in Chapter 4.2. We believe that M3 model structure will be a suitable choice for further improvement of evaluation function if more data is collected for training.
4.2 Gameplay Performance

Selected networks were imported to the game program as a utility function predicting the winning probability of a player with a given board. The program used minimax algorithm as the search algorithm and was configured to look forward 4 steps ahead.

For each network, it was tested by playing with a moderate opponent and a strong opponent 5 times each as a white player. The playing results are listed in Table 7. More details of the gameplay performance testing are recorded in the Document “GameplayPerformanceTest.xlsx”.

<table>
<thead>
<tr>
<th>Tested Networks</th>
<th>Win-loss-draw ratio against level 10 opponent</th>
<th>Win-loss-draw ratio against level 15 opponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>N0-8</td>
<td>0-5-0</td>
<td>0-5-0</td>
</tr>
<tr>
<td>N0-9</td>
<td>1-4-0</td>
<td>1-4-0</td>
</tr>
<tr>
<td>N1-1</td>
<td>1-4-0</td>
<td>1-4-0</td>
</tr>
<tr>
<td>N1-2-1, N1-2-2</td>
<td>2-3-0</td>
<td>1-4-0</td>
</tr>
<tr>
<td>N2-2</td>
<td>3-2-0</td>
<td>1-4-0</td>
</tr>
<tr>
<td>N2-3</td>
<td>2-3-0</td>
<td>0-5-0</td>
</tr>
<tr>
<td>N2-4</td>
<td>1-4-0</td>
<td>0-5-0</td>
</tr>
</tbody>
</table>

Table 7 Win-loss-draw ratio for selected networks playing against level 10 and level 15 opponent

As shown in Table 7, N2-2 had the best performance in playing against the two opponents when compared with other networks. It was the only network that had around 50% chance of winning a level player, which implied that it had a similar strength as a moderate opponent. As for the performance on playing against a strong opponent, a few of the networks could win one game out of five games. It shows that none of these networks were of equal strength as a strong opponent.

In addition to the gameplay win-loss ratio, as recorded in the Document “GameplayPerformanceTest.xlsx”, it was observed that most networks were not sensitive to advantageous positions such as the corners and borders. After a simple examination over the prediction results and the expected probability from the data set, the prediction results generally matches with the data set. This implies the probability that our current data set cannot reflect the true winning probability of different boards for the game of Othello.
4.3 Limitations

4.3.1 Slower computational time for CPU

In practice, GPU is used for training the neural network due to its high computational power. However, due to limited funding, only CPU can be used for the training. When compared with a CPU, the training time required for a GPU can be 8.5 to 9.0 times faster. [8] The relatively low computational power of the CPU results in longer training time. With limited time for this project, the neural network training process may not be complete and thorough enough, leading to undesirable results.

4.3.2 Size of training data

Firstly, the number of distinct board configurations obtained from the collected game move records may not be large enough for training a neural network with a larger size, which may hinder the process of capturing different board features.

Secondly, the calculated winning probability for each board configurations as the training data may not reflect the true winning probability for the game Othello. This may due to the reason of insufficient number of game move records. It is expected the more game move records can provide a more accurate estimation of winning probability of different board configurations.
Chapter 5 - Conclusion and Recommendation for Future Work

This project aims to develop a data-driven evaluation function for the game of Othello by using deep learning neural network. The evaluation function developed is able to predict the winning probability of a board of size 8x8 for a player. 13 networks have been trained using different set of training data under three major kinds of neural network model structure. Majority of them has suffered from different degrees of overfitting. One of them can achieve the targeted winning rate is 50% when playing against a moderate opponent, while none of them can achieve the targeted winning rate of 35% when playing against a strong opponent. More game records are expected as input data for improvement of neural network training and for better gameplay performance.
References


