Deep learning with Othello
Application and analysis of deep neural networks and tree search on Othello

Sun Peigen (3035084548)
Worked with Nian Xiaodong (3035087112) and Xu Chaoyi (3035084328)
Under supervision of Prof. Kwok-Ping Chan

Department of Computer Science
The University of Hong Kong

Submission Date: Apr 16, 2017

Project Website: i.cs.hku.hk/fyp/2016/fyp16017
Contact Information: sunbacon@hku.hk
Abstract

Recently, deep learning is becoming prevalent in the AI field. However, currently most of the game AI are still using the manually extracted features. What if we apply the technology of deep learning to game AI? This report is inspired by AlphaGo and going to discover the potential of deep neural network (DNN) to be the evaluation functions of the game Othello. In this report, design, implementation as well as findings of our program will be discussed in detail. We used the winning rate with other AIs to measure the strength of evaluation functions. By comparing the different AI based on DNN and other methods, the applicability of using DNN for evaluation has been verified. However, the effectiveness and efficiency of using DNN is not Satisfactory due to the size of the problem. This finding may have an enormous impact on game AI design.
Acknowledgement

We would like to express our special thanks of gratitude to our supervisor Prof. Kwok-Ping Chan as well as our principal Peter Mathieson who gave us the golden opportunity to do this wonderful project on the topic deep learning, which also helped us in doing a lot of Research and we came to know about so many new things we are grateful to them.
Table of Contents

Abstract ............................................................................................................................. 1

Acknowledgement .......................................................................................................... 2

Table of Contents ............................................................................................................ 3

Abbreviations .................................................................................................................. 5

Figures and Tables .......................................................................................................... 6

1 Introduction .................................................................................................................. 7
  1.1 Rules of Othello ....................................................................................................... 7
  1.2 Analysis of Othello .................................................................................................. 7
  1.3 Deliverables ............................................................................................................. 8
  1.4 Scope ....................................................................................................................... 8
  1.5 Contribution to this project .................................................................................... 8

2 Previous Works ............................................................................................................. 11

3 Theoretical Background ............................................................................................... 12
  3.1 Problem setting ...................................................................................................... 12
  3.2 Evaluation in Game ............................................................................................... 12
  3.3 Game Tree Searching ........................................................................................... 13

4 Methodology ................................................................................................................ 15
  4.1 Development Environment ................................................................................... 15
  4.2 Algorithms ............................................................................................................. 15
    4.2.1 Minimax Search and Alpha-beta Pruning ....................................................... 15
    4.2.2 Weighted Square Strategy ............................................................................... 16
    4.2.3 Evaluation Network ......................................................................................... 17
    4.2.4 Monte Carlo Tree Search ............................................................................... 17
    4.2.5 Policy Networks ............................................................................................... 19
4.2.6 Value Networks .............................................................................................................. 20

5 Results ................................................................................................................................... 21

5.1 Training Data Set .................................................................................................................. 21

5.1.1 Overview of Data Set ........................................................................................................ 21

5.1.2 Symmetry augmentation .................................................................................................. 22

5.2 Evaluation Networks ............................................................................................................ 23

5.2.1 Training ............................................................................................................................. 23

5.2.2 Evaluation on the Playing Strength .................................................................................. 24

5.3 Policy Networks with MCTS ............................................................................................... 24

5.3.1 Training ............................................................................................................................. 24

5.3.2 Evaluation on the Playing Strength .................................................................................. 25

5.4 Value Networks with MCTS ............................................................................................... 26

5.4.1 Training ............................................................................................................................. 26

5.4.2 Evaluation on the Playing Strength .................................................................................. 27

5.5 Random policy with MCTS ............................................................................................... 27

5.6 Discussions .......................................................................................................................... 28

6 Conclusions ............................................................................................................................. 30

References ................................................................................................................................... 31
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphic Processing Unit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>MCTS</td>
<td>Monte Carlo tree search</td>
</tr>
<tr>
<td>PUCT</td>
<td>Polynomial Upper Confidence Trees</td>
</tr>
<tr>
<td>SL</td>
<td>Supervised Learning</td>
</tr>
<tr>
<td>tanh</td>
<td>hyperbolic tangent function</td>
</tr>
<tr>
<td>UCT</td>
<td>Upper Confidence Bounds for Trees</td>
</tr>
</tbody>
</table>
Figures and Tables

Figure 1 | Illustration of Othello rules ................................................................. 7
Figure 2 | Home page of the project website .......................................................... 9
Figure 2 | Illustration of how an evaluation function will give score to moves ........ 13
Figure 3 | Game tree in opening of a round in Othello ............................................ 14
Figure 4 | Weighted square strategy in Project Tempo ............................................ 16
Figure 5 | Architecture of deep neural networks used in Project Tempo .................. 17
Figure 6 | Monte Carlo tree search in Project Tempo ............................................. 18
Figure 8 | Categorical score distribution ................................................................. 22
Figure 9 | Symmetry of Othello chess board ............................................................ 23
Figure 10 | Accuracy of value networks with epochs .............................................. 23
Figure 11 | Training result of policy networks ......................................................... 25
Figure 12 | Training result of value networks ............................................................ 26

Table 1 | Result of battles  \( v_z \) against random choice ........................................ 24
Table 2 | Result of battles  \( v_z \) against weighted square strategy ........................... 24
Table 3 | Result of battles  \( p_{SL} \) against random choice ...................................... 25
Table 4 | Result of battles  \( p_{SL} \) against weighted square strategy ....................... 25
Table 5 | Result of battles MCTS (policy) against random choice .............................. 26
Table 6 | Result of battles MCTS (policy) against weighted squared strategy ............. 26
Table 7 | Result of battles MCTS (policy + value) against random choice .................. 27
Table 8 | Result of battles MCTS (policy + value) against weighted squared strategy .. 27
Table 9 | Result of battles MCTS (random) against random choice .......................... 27
Table 10 | Result of battles MCTS (random) against weighted squared strategy .......... 28
1 Introduction

In the first half of 2016, AlphaGo became rather well-known due to a victory against Mr. Lee Se-dol. It is the first time that, on a full-sized board, a computer Go program defeated a top professional human Go player. The core techniques that AlphaGo used is deep neural networks (DNN) and Monte Carlo tree search (MCTS) algorithm. [1] As a field developing astonishingly in recent years, deep learning benefits from the huge improvement of computational capability of modern processors and becomes one of the most popular research topic of artificial intelligence. Motivated by AlphaGo, the objective of this project - Tempo, is to develop a similar game artificial intelligence (AI) program, applying the same technologies of neural network and tree search algorithm as AlphaGo, to play another chess game, Othello (also known as Reversi).

1.1 Rules of Othello

The basic rule of Othello is that players take turns placing discs to bound opponent’s ones and reverse them into his own. As shown in Figure 1, after placing a new disc, opponent’s discs bounded in straight line by the newly placed disc and other disc(s) of current player will be turned into the current player’s color. Each move must have at least one opponent’s disc flipped, otherwise the player will be skipped until he can make a move to flip opponent’s disc(s). Thus, for each turn, there will always be limited valid moves that could be chosen to be next move, and the amount is usually not more than 10. When the chess board is full-filled by discs, or any party have no disc left (all discs were flipped by the opponent), or neither of players can make a valid move, the current game round ends, and the player with more disc wins the game.

![Figure 1 | Illustration of Othello rules.](image)

The game will start with an initial board of 4 discs in the middle, as shown in left most board. This figure shows a simple opening of 2 steps of one round, and the opponent’s discs bounded by current own discs and new disc are flipped.

1.2 Analysis of Othello

The board size of Othello is relatively small, which is only 8×8, and the number of legal moves
during each step is also limited. Thus, both the total number of steps and the number of possible moves during one step are much smaller than that of Go, and this is one of the reasons why we choose Othello: a simpler game will be easier for us to handle.

However, Othello is estimated to have the number of legal positions up to the level of $10^{28}$, and the game-tree searching complexity of approximately the level of $10^{58}$. On the other hand, Othello remains unsolved\footnote{A mathematically “solved game” is a game whose outcome can be correctly predicted from any position, if both players play perfectly.} mathematically. [2] Thus, the further study to make stronger game AI programs on Othello is still meaningful to solve the game.

### 1.3 Deliverables

As the outcome, an online Othello battle AI program with interactive graphic user interface (GUI) is available at [i.cs.hku.hk/fyp/2016/fyp16017/demo.html](http://i.cs.hku.hk/fyp/2016/fyp16017/demo.html), which can play Othello against other player by calling APIs on cloud computing backend to get computer’s move.

Different AIs were developed in this project, including weighted square strategy with minimax search, DNN with minimax search and MCTS.

### 1.4 Scope

This project mainly focused on the software implementation of the Othello AI program, including the game tree searching algorithm, preprocessing of game data, structure design of policy networks and value networks, as well as discussion of the results. Studies and research about these fields were carried out to build up a strong enough AI program for this project. Since this is an individual report of a group project, the report will mainly focus on my parts in this project but still include relative parts.

### 1.5 Contribution to this project

This section will specify my work within the scope of this project. Since all team members have studied the topic and made mutual contributions to the project, it is hard to precisely separate the work in this project, so I will include all individual works and cooperative works I have participated in.

In the beginning, I built an Othello game engine and implemented Minimax search with Alpha-beta tuning to generate training samples, perform AI battles and record each move into a game
file in a standard recording method. Also, I develop a JavaScript version of Othello game engine to enable the visitors of our website to play Othello with our AI. The website is on [i.cs.hku.hk/fyp/2016/fyp16017](http://i.cs.hku.hk/fyp/2016/fyp16017). Figure 2 is the home page of the website. The UI design is developed by another group member Nian Xiaodong.

![Figure 2 | Home page of the project website](image)

Click the button “Play” will let the visitor be able to play with our AI.

To generate the categorical data and numerical training data for neural networks, I wrote several python scripts to read from existing game books and combine the data into an array. Together with Nian Xiaodong, we cleaned the duplicate data out of our set and did the symmetry augmentation to enhance the robustness of our model. During the model construction and tuning, we together tried different input features of the game board and distinct model architectures of neural networks.

In the latter half of the project, we found that the original Othello engine was not fast enough to run a large scale of testing. Thus, Nian Xiaodong and I learned the algorithms from an open-source game engine, namely 'paip-python' [3], and reconstructed our Othello engine. Also, we refined our weighted square strategy to make it more powerful.

Also, during the development of MCTS, Nian Xiaodong and I collaboratively adapted the implementation in ‘MuGo’ engine [4] to be compatible with our game engine and developed a battle bot for different AI to play against each other.
In the following sections, the report will introduce some previous works done by others in the field of Othello AI in Section 2, more theoretical background about this game in Section 3, the theories used and algorithms applied in the project in Section 4, the total results and assessment of the project in Section 5.
2 Previous Works

Even though Othello is unsolved, computer scientists still devoted themselves in developing stronger Othello programs. *Iago* developed by Paul S. Rosenbloom in 1981 became the first program which beat the human world champion. But later in 1986, it was defeated “consistently” by *Bill*, which was developed by Kai-Fu Lee and Sanjoy Mahajan, adopting the concept of machine learning (quite shallow, though). [6] *Bill*, of course, is also surpassed in a few years. In 1992, Michael Buro started the Othello program *Logistello*, which used human-defined features to abstract useful information from the game board. [7] In 1997, it turned out that *Logistello* can beat the greatest human player and achieved remarkable success. And similarly, *Logistello* is far surpassed by later stronger programs. Nevertheless, the main ideas behind *Iago*, *Bill* and *Logistello* are worth studying and all have been patient teachers and qualified opponents of our program. In our future research and development, they will still be of significant help.
3 Theoretical Background

The objective of our project is to develop a game AI. In the following parts, the game will first be abstracted into a simple problem. Then comes with other definitions and tools used in this project.

3.1 Problem setting

Based on the rules of Othello, it is a game where both players have perfect information about the whole game and can be defined as an alternating Markov game [6]. Thus, the general problem setting for alternating Markov games is also suitable for Othello. Here, we follow the descriptions in the way that AlphaGo used to abstract Go: there is a state space $S$, an action space $A(s)$, and a state transition function $f(s, a)$. The major differences between Go and Othello are the size of $S$ and $A(s)$: the state space and action space of Othello are far smaller than those of Go.

Based on the setting, if we have different probability to choose among moves, we can define this prior probability as a policy $p(a|s)$, which is a probability distribution over the legal moves $a \in A(s)$. Specially, we can regard the random strategy as a policy which has uniform distribution over the legal moves.

3.2 Evaluation in Game

To obtain advantages over the opponent, one player needs to have a clear knowledge on the game, has capability to evaluate the current state and find the most valuable move. For AI, it needs an evaluation function to help make decisions.

Here, we define the evaluation function as a map from board configurations to values. If we define the function as $v$, the board features as $s$ and the outcome score as $G$, the equation is

$$G = v(s)$$

Combined with the problem setting, we can have that

$$G = v(s') = v(f(s, a))$$

where $s' = f(s, a)$. Thus, to find the best move for board $s$ is equivalent to find $a^*$ s.t. $v(f(s, a^*)) = \max(v(f(s, a)))$ for any possible $a \in A(s)$.

Obviously, the strength of an AI is mostly constrained by the accuracy of its evaluation function. A good evaluation function should never be worse than the random strategy, as the random strategy can be considered as a constant function.

Take an example from Othello: if we have three possible moves on the board, marked as A, B
and C in Figure 3, mostly an ideal evaluation function should give the result as:

\[ v(s, A) > v(s, B) > v(s, C) \]

This is because in Othello, the “unchangeable discs” are the much more valuable than other discs. The corner, A, is a common unchangeable disc as the opponent can never regain it according to rules. The edge, B, is less possible to be changed as it can only be sandwiched in two directions while normal discs may suffer attack from at most four directions. Thus, in most situations, the corner is the best choice, followed by the edge and the middle.

![Figure 3](image)

**Figure 3** | Illustration of how an evaluation function will give score to moves. If the evaluation function of Othello agrees with the experience that usually corners are more important than edges and edges are more important than the middle, it will evaluate different moves on board \( s \) ordering in:

\[ v(s, A) > v(s, B) > v(s, C) \]

There are different ways to build strong evaluation functions. One way is basing on the experience of human players to design the equation used for calculating the score from the board. *Iago* and *Bill* mentioned above used this way and achieved astonishing strength. However, it requires the developer to have deep insights and rich experience with the game. Another way of constructing an evaluation function is to let the computer give a score by learning or simulations. In our project, the second method is adopted: neural networks are used to learn from existing samples and Monte Carlo Tree Search is used for simulation.

### 3.3 Game Tree Searching

In game theory, a game tree is a directed graph whose nodes are positions in a game and whose edges are moves. As shown in Figure 4, by listing all possible moves and corresponding results, a thorough analysis can be obtained to help find the best move for current step. If one can write down the whole game tree, he will be able to find a way to maximize his rewards all the time, which is the “win strategy” for some games. However, due to that the space of search trees usually grow exponentially, it is hard to exhaust all possible leaf of a game tree.
Figure 4 | Game tree in opening of a round in Othello. A game tree is a directed graph that represents the game theory logic. Each edge denotes a possible move and each node denotes the possible position corresponding to the move of the edge.

To find the best move in a limited size of a game tree, different strategy can be used: one is to make the evaluation function as precise as possible; another is to try to discard those nodes that has little value to expand and spare the resources to exploit the useful nodes. A good AI should combine both strategies to maximize its possibility of finding the best move.
4 Methodology

In this section, the scope of this project will be first introduced, then different algorithms implemented for the game AI used in the project will be discussed, including weighted square strategy, alpha-beta pruning based on Minimax search, convolutional neural networks (CNN) and Monte Carlo Tree Search (MCTS). The implementation details and other trials will be described in the next section.

4.1 Development Environment

In our project, Python is used as the main language to build the game engine to play Othello, the neural networks and the search trees. We chose Python for its high cost-effective value and wide supporting packages related to deep learning and mathematical calculations. Among deep learning frameworks designed for Python, Keras is chosen for its rapid develop cycle, light scale and high-level integrations as these features fit the size and duration of our project. Other packages like scikit-learn are also used for the simplicity of hyper-parameter tuning of models.

Other languages are also used during the development. For example, HTML, CSS and JavaScript are used in the construction of our website and GUI.

4.2 Algorithms

In this project, different algorithms are used to help enhance the performance of AI. To compute the optimal value function, minimax search can be used recursively. However, if efficiency is taken into consideration, the performance of normal tree search drops quickly as the search space grows. In the prior work like Bill, minimax search with alpha-beta pruning was widely used combined with an elaborately designed value function. In our project, minimax search with alpha-beta pruning will used as a tester of our other AI.

Another algorithm used in our project is the Monte Carlo Tree Search (MCTS), which can be considered as an alternative to minimax search. The use of MCTS has achieved success widely in other chess game, including Go.

4.2.1 Minimax Search and Alpha-beta Pruning

Minimax search is a way to select the best move based on a game tree. Its core idea is to predict the counter strategy of the opponent and avoid the worst situations. Alpha-beta pruning is an effective pruning algorithm based on Minimax search [7], and Alpha-beta pruning together with Minimax search is widely used in the game AI design. By implementing the alpha-beta pruning to the Minimax search, the AI program can prune the useless nodes if the algorithm find that the value of current subtree is already equal to or worse than other subtrees, and save the time to evaluate other nodes.
4.2.2 Weighted Square Strategy

In this project, a simple traditional AI program based on alpha-beta pruning and weighted square strategy was built to serve as a baseline. Weighted square strategy is one of the widely-used strategies in Othello [8]. This strategy is abstracted from the observations that occupying different squares on the Othello game board has distinct influences on the game result. From earlier experience, the outer places such as the four sides, play much more important roles than those at the inner board. Especially, the corners are the most influential places as once been taken, they cannot be re-occupied by the opponent, thus they provide unimpeachable stability for the player who occupies them and can help to possess the sides and the inner board afterwards. According to the theory of this strategy, a scoring matrix storing the different importance of places is needed to evaluate the board. If we denote the scoring matrix as $M$, the evaluation function should be

$$v(s, a) = \sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij} \times s'_{ij}$$

where $s$ is the current game board and $s'$ is the board after an action $a$ is taken. Here, a three-way representation is used to encode the game board. $s'_{ij}$ is 1 if the place at $i^{th}$ row $j^{th}$ column is occupied by the current player, and is $-1$ if that place is occupied by the opposite player. If that place is not occupied by either, $s'_{ij}$ is 0. As the game board of Othello has the size of $8\times8$, $n = 8$ in this function.

![Game board $s'$](image1)

![Weighted squares $M$](image2)

![Evaluation](image3)

**Figure 5** | Weighted square strategy in Project Tempo. Let $s'$ be the board after black taking move $a$ based on $s$, and $M$ be the weighted squares. An intuitive way to evaluate the board is superpose $M$ on $s'$, which makes it easy to see the weight of each disc, and sum all disc's weight as positive for player while negative for opponent. For $s'$, $v(s, a) = 40$, which indicates an advantage.

The score matrix of weighted square strategy is usually pre-defined. Thus, it is an evaluation function designed manually and its accuracy depends on the designer’s experience.
4.2.3 Evaluation Network

CNN (Convolutional Neural Network) is a kind of feed-forward artificial neural network; whose artificial nerve units can respond to the surrounding patterns in detection fields. A CNN model consists of one or multiple convolutional layers and fully-connected layers, and can also include pooling layers and relevance weights. This kind of structure enables CNN make use of the 2-dimensional structure of the input data. Due to this, compared to other deep learning models, CNN could give a better result on processing 2-dimensional data, such as the image and game board processing. In our project, an evaluation CNN is used as an evaluation function combined with Minimax search to calculate the best move.

CNN evaluation networks were constructed to automatically learn how to evaluate the game board. As shown in Figure 6, the neural network consists of 2 convolutional layers and 2 fully-connected layers with a $tanh$ activation function. It is used to predict the evaluated numerical score of the game board. We did not apply max-pooling layers because the game board is relatively too small.

![Figure 6](image)

**Figure 6 | Architecture of deep neural networks used in Project Tempo.**

Evaluation network $v_z(s)$ (z for WZebra) was trained by supervised learning. And the training data of the network were from the self-playing games of another Othello AI program - WZebra, which is one of the strongest Othello AIs in the world. This AI provides various levels of search depths and evaluation scores of moves. We generated training games with six search steps, considering the balance of search strength and generating efficiency. Currently, over 4000 self-playing games with evaluation scores of each step were recorded as the training set.

This value network $v_z(s)$ was used as evaluation function in alpha-beta pruning searcher, which provide a deep learning AI program. The assessment of this AI is available in result section.

4.2.4 Monte Carlo Tree Search

MCTS [9] is a heuristic search algorithm, which makes move based on results of copious self-gaming. AlphaGo has implemented an asynchronous policy and value MCTS algorithm, which combined both policy network and value network into MCTS. [1] Based on this idea, we
constructed a similar MCTS (as shown in Figure 7) algorithm using the policy network $p_{SL}(s)$ and value network $v_p(s)$, whose details will be discussed later.

**Figure 7 | Monte Carlo tree search in Project Tempo.** Each loop in a typical MCTS consists of 4 steps: selection, expansion, simulation, and backpropagation (backup), but the MCTS here is modified slightly. 

- **a.** Select the leaf node along the edges with maximum action value $Q + u(P)$ positively correlated to the stored probability $P$ in each edge, which is a kind of variant UCT. 
- **b.** Expand the selected leaf node, generate the probabilities for next move on the board fitting by $p_{SL}$ and store the probabilities as the priority $P$ for valid moves. 
- **c.** Simulate the game with $p_{SL}$ by self-play to the end of the game, also evaluate the leaf node board by value network $v_p$. 
- **d.** Update the action value along the backup path for each node, the action values $Q$ are the mean of simulation winning rate and score given by value networks.

Each node of MCTS has the following fields for $s$ in state space $S$ and $a$ in action space $A(s)$

$$\{P(s,a), N(s,a), W_r(s,a), W_p(s,a), Q(s,a)\}$$

$P(s,a)$ is the prior probability that generated by the policy; as stated before, a random policy will generate same prior probability for all children of a node. $N(s,a)$ is the total times that this node is visited or simulated to the end. $W_r(s,a)$ is the winning times when the simulations starts with this node with the rollout policy. $W_v(s,a)$ is the value evaluated from the value network. $Q(s,a)$ is the final score of this node, which is also called as the "action value". When the MCTS is asked to give a best move $a^*$ for a board state $s$, it will return

$$a^* = \arg\max_a Q(s,a)$$

Thus, $Q$ can be taken as the evaluated score generate from the MCTS.

**Selection** At the beginning of simulation, a node should be selected as the starting node. To balance exploration and exploitation, we use PUCT algorithm [10] to determine which node to be selected:

$$a^* = \arg\max_a (Q(s,a) + u(s,a))$$

where
Policy network is a 64 except the last layer uses softmax as its activation function. The output of the policy network is a 64-length 1-D vector representing the probability of each move on the board.

Policy network \( p_{SL}(s) \) (SL for supervised learning) was trained by supervised learning. The training set was the same as evaluation network, but using moves as the data label instead of

\[
u(s, a) = c_{\text{puct}} \sqrt{P(s, a) \times \sum_b N(s, b)} / (1 + N(s, a))
\]

\( b \) is the possible values of \( a \), and \( \sum_b N(s, b) = N(s', a') \) where \( f(s', a') = s \) (i.e. \( \sum_b N(s, b) \) is equal to the \( N \) value of this node’s parent). The \( c_{\text{puct}} \) is a constant to adjust the balance of exploration and exploitation. This algorithm will prefer nodes with high prior probability and low visit counts, and gradually shift to exploit nodes with high action values.

**Expansion** When a node \((s, a)\) is visited, it will be expanded by all possible moves. All its children \((s', b)\) will be initialized as

\[\{ P(s', b) = p(b|s'), N(s', b) = 0, W_r(s', b) = W_v(s', b) = 0, Q(s', b) = Q(s, a) \}\]

where \( s' = f(s, a) \). \( p(b|s') \) is based on the prior policy that MCTS is used. A good prior policy should be able to inhibit the expansion of useless nodes.

**Evaluation** When a node is going to be evaluated, its action score comes from two parts: one is directly from the value network \( v_p \), and the other is from the quick simulation following the rollout policy \( p_r(a|s) \) where each move \( a^* = \text{argmax}_a(p_r(a|s)) \). When the game reaches the end, a score \( z \) indicating whether the current player of this node wins or loses will be returned as the evaluation value from the rollout policy.

\[z_r = \begin{cases} 
0 & \text{win} \\
0.5 & \text{draw} \\
1 & \text{lose}
\end{cases}\]

**Backup** When backing up the value from evaluation to the root, we let \( N(s, a) \leftarrow N(s, a) + 1 \), \( W_r(s, a) \leftarrow W_r(s, a) + z_r \), \( W_v(s, a) \leftarrow W_v(s, a) + v_p \) and \( Q(s, a) \leftarrow \frac{(1-\lambda)W_v(s, a) + \lambda W_r(s, a)}{N(s, a)} \).

Thus, the final action score \( Q(s, a) \) of this node is a mixture of the results from the rollout policy and value networks.

### 4.2.5 Policy Networks

Policy networks are used for calculating the prior probability of a board \( s \). It worked as the prior policy \( p(a|s) \) in MCTS. Also, it can also be used as the rollout policy in simulations.

Policy networks were also constructed by CNN, but with a different structure. In policy networks \( p_{sl} \), we used six convolutional layers before output: the first layer adds zero paddings to the input and convolves 128 filters of kernel size 5*5 with stride 1, second to the fifth layer adds zero padding and convolves 128 filters of kernel size 3*3 with stride 1, and the last layer convolves 1 filter of kernel size 1*1 with stride 1. All layers use the activation function ‘RELU’ except the last layer uses softmax as its activation function. The output of the policy network is a 64-length 1-D vector representing the probability of each moves on the board.
scores provided by WZebra. Since these over 4000 game transcripts were generated by WZebra with human-style randomness, these training samples can have some degree of variety on chess playing routine, which is good for neural network training to avoid overfitting.

After processing the initial data, we have 660,000 training samples for our policy network. The loss function for the network is the categorical cross entropy and a stochastic gradient descent update is applied to minimize the loss with learning rate as 0.04 and momentum set as 0.

4.2.6 Value Networks

Value networks are used to help evaluate the state $s$ together with the simulations. It is an evaluation function mentioned in the theoretical background.

The structure of value networks ($v_s$) is almost the same as policy networks, except that a fully connected layer with 128 hidden units and one output using $tanh$ as the activation function is added after the policy networks.

The training data of value networks are generated from 500 self-playing games of MCTS using the policy networks $p_{sl}$. We stored all nodes searched in MCTS with its $Q(s, a)$ value. After processing the raw data, we have 6,000,000 training samples. The loss function for this model is mean square error. The optimizer is the same as that of policy networks.
5 Results

This section will firstly describe the processing methods of training data set, then show the
details of training tendency of neural networks, and provide the training accuracy and battle
winning rate of Project Tempo as the assessment result at last.

In the evaluation of AIs’ strength, we use two indicators: one is the random strategy, and the
other is the weighted square strategy with searching depth 3 based on Minimax search with
Alpha-beta tuning. To make the comparison standardized, we let each AI play with these two
testers for 100 games, with 50 playing as black and 50 playing as white.

5.1 Training Data Set

5.1.1 Overview of Data Set

As mentioned in section 4.2.3 (Page 19), the training data of evaluation network and policy
network are generated by self-playing of WZebra. The games are evaluated using search depth
of 6 with last 14 perfect moves, and played with high-level randomness. Using search depth of
6 is under the consideration of both time efficiency and evaluation qualifications. By setting
randomness as median, the training data can cover more situations of chess board
configuration but remain reasonable moves. And the neural network takes the evaluation score
provided by WZebra as the label for supervised training. The training set of Project Tempo has
more than 4,000 games in total.

In this project, each board configuration (namely the board after each step) is treated as a data
sample for evaluation, and generally each game contains about 60 steps, which means each
game can be transferred into about 60 board configuration samples. After extension by rotation
and flipping, and deduplication, the total size of the data set is over 660,000.

As training data, each input sample of board configuration is encoded into 10-layer 8*8
matrices, which is a 10*8*8 3-densiontional matrix. Each 8*8 matrix represents a certain feature
or specific information about the game-board. In total, we have 10 layers: there are 3 layers
representing the current discs on the board, 2 constant layers with all ones and all zeros, 1 layer
of valid moves for current players, and 4 layers to mark the internal discs and external discs for
own discs and opponents’ discs. (More details about input features are in Appendix I)

Another source of our training set was from the self-playing games of MCTS with policy
network \( p_{\text{sl}} \), which were used to train out value network used in MCTS. The extension of
rotation and flipping as well as deduplication were also applied on this set, forming a set of
more than 6,000,000 training samples.

As to labels, the evaluation networks \( v_{\text{e}}(s) \) used categorical labels of 17 classes. The original
scores provided by WZbera followed the normal distribution. After rescaling, these scales were
transferred into 17 classes, denoting values from -8 to 8. The processed data roughly followed
uniform distribution as shown in Figure 8, which decreased the risk to be overfit.

![Figure 8](image.png)

**Figure 8 | Categorical score distribution.** The distribution of the scores given by WZebra roughly follows the uniform distribution.

The labels for policy networks $p_{SL}$ were the moves made by WZebra. As there are 64 squares on the game-board of Othello, labels are represented by 64-dim vectors.

The labels for value networks $v_p$ were the $Q(s, a)$ values from the MCTS. These float value ranged from 0 to 1.

### 5.1.2 Symmetry augmentation

In the early stage, every single board configuration was treated as a data sample, and the total size of the training set was over 250,000. However, the neural network model trained based on such a data set was not ideal. The model even predicted every input to be same class. This problem puzzled our team for quite a long time and we tried many probable solutions on neural network structure, while the solutions didn’t work. After that we realized that the problem may existed in training data: 1. there existed too many same sample in the data set especially in first few steps, and 2. the square chess board is symmetrical (as shown in Figure 9) but samples in data set are not the same in different rotation direction. Both too many duplicate data and unbalanced data affected the classification accuracy.

To eliminate the shortage, we extended the whole data set by rotating the board by 180° and flipping over two diagonals as in Figure 9, thus the training set is 4 times as large as before. And then remove duplicate scenarios in the new data set by removing the same board configuration and take the average of the score as the new label. As a result, the size of final training data set is 660,000 as mentioned earlier in this subsection.
Project Tempo - Deep Learning with Othello

Figure 9 | Symmetry of Othello chess board. a, the basic board $G = \{s[x][y] \mid x, y \in \text{range}(8)\}$. b, the mirror board along the diagonal from top left to bottom right, $G' = \{s[y][x] \mid x, y \in \text{range}(8)\}$. c, reverse board, $G'' = \{s[8-x][8-y] \mid x, y \in \text{range}(8)\}$. d, the mirror board along the diagonal from top right to bottom left $G''' = \{s[8-y][8-x] \mid x, y \in \text{range}(8)\}$.

5.2 Evaluation Networks

5.2.1 Training

The structure of CNN evaluation networks in Project Tempo consists of one input layer, four hidden layers, and one output layer. Within hidden layers, there are two convolutional layers and two fully connected layers and dropout layers. Limited by the board size, more hidden layer is workable. And we do not apply max-pooling layers because the game board is relatively too small. The output layer has 17 neurons stands for 17 classes (-8 ~ 8) defined by input data. The figure of training and test accuracy is shown below.

Figure 10 | Accuracy of value networks with epochs. This figure shows how the training and test
accuracy changed with the iterations of training. Each Epoch has 10 iterations. We stopped at 150 iterations before it becomes overfit. The batch-size of training is 2000.

### 5.2.2 Evaluation on the Playing Strength

We used the evaluation networks together on the Minimax search with Alpha-beta pruning. To make the duel more balanced, we set the search depth of it also be 3. Each game needs around 4 seconds. The results are shown below.

<table>
<thead>
<tr>
<th></th>
<th>(v_z) wins</th>
<th>RC first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v_z) wins</td>
<td>47</td>
<td>42</td>
<td>89</td>
<td>89%</td>
</tr>
<tr>
<td>Random choice wins</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>7%</td>
</tr>
<tr>
<td>Draw</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4%</td>
</tr>
</tbody>
</table>

**Table 1 | Result of battles \(v_z\) against random choice**

<table>
<thead>
<tr>
<th></th>
<th>(v_z) first</th>
<th>WS first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v_z) wins</td>
<td>20</td>
<td>18</td>
<td>38</td>
<td>38%</td>
</tr>
<tr>
<td>Weighted square wins</td>
<td>28</td>
<td>27</td>
<td>45</td>
<td>45%</td>
</tr>
<tr>
<td>Draw</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>7%</td>
</tr>
</tbody>
</table>

**Table 2 | Result of battles \(v_z\) against weighted square strategy**

From these results, we can conclude that \(v_z\) have certain intelligence and do help in analyzing the game board, but it is still a little bit weaker than the carefully designed weighted square strategy. This result pushed us to find more effective algorithms.

### 5.3 Policy Networks with MCTS

#### 5.3.1 Training

The structure of CNN policy networks was discussed in detail in the methodology part. The training accuracy with iterations is shown in Figure 11.
5.3.2 Evaluation on the Playing Strength

To test the evaluation of policy network, we first let it battle directly with our two testers without any search. Each game needs less than 1 second.

<table>
<thead>
<tr>
<th></th>
<th>( p_{SL} ) first</th>
<th>RC first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_{SL} ) wins</td>
<td>34</td>
<td>33</td>
<td>67</td>
<td>67%</td>
</tr>
<tr>
<td>Random choice wins</td>
<td>14</td>
<td>12</td>
<td>26</td>
<td>26%</td>
</tr>
<tr>
<td>Draw</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>7%</td>
</tr>
</tbody>
</table>

Table 3 | Result of battles \( p_{SL} \) against random choice

<table>
<thead>
<tr>
<th></th>
<th>( p_{SL} ) first</th>
<th>WS first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_{SL} ) wins</td>
<td>17</td>
<td>8</td>
<td>25</td>
<td>25%</td>
</tr>
<tr>
<td>Weighted square wins</td>
<td>33</td>
<td>42</td>
<td>75</td>
<td>75%</td>
</tr>
<tr>
<td>Draw</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 4 | Result of battles \( p_{SL} \) against weighted square strategy

The above results show that the policy networks are significantly stronger than the random choice strategy. However, its strength is still weaker than the weighted square strategy.

Then we combined the policy network with the MCTS, using the policy network both as prior prediction and rollout policy. The \( \epsilon_{pred} \) is set as 5. The maximum search time of each step is set as 5 seconds. Each game consumes more than 120 seconds.
### Table 5 | Result of battles MCTS (policy) against random choice.

<table>
<thead>
<tr>
<th></th>
<th>MCTS first</th>
<th>RC first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCTS wins</td>
<td>50</td>
<td>49</td>
<td>99</td>
<td>99%</td>
</tr>
<tr>
<td>Random choice wins</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1%</td>
</tr>
<tr>
<td>Draw</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

### Table 6 | Result of battles MCTS (policy) against weighted squared strategy.

<table>
<thead>
<tr>
<th></th>
<th>MCTS first</th>
<th>WS first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCTS wins</td>
<td>24</td>
<td>24</td>
<td>48</td>
<td>47%</td>
</tr>
<tr>
<td>Weighted square wins</td>
<td>19</td>
<td>24</td>
<td>43</td>
<td>43%</td>
</tr>
<tr>
<td>Draw</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>9%</td>
</tr>
</tbody>
</table>

From the result in Table 6, it can be said that MCTS and weighted squared strategy evenly matched. However, the time used for each step is too long. Considering the efficiency, this algorithm is not as good as weighted square strategy with Minimax search.

### 5.4 Value Networks with MCTS

#### 5.4.1 Training

The structure of CNN policy networks was discussed in detail in the methodology part. The training accuracy with iterations is shown in Figure 12.

After 20 iterations of training, the ASE dropped to 0.11 where the value range of targets is [0, 1].

![Figure 12 | Training result of value networks.](image)

In each epoch, the model is trained for two iterations. Further training has little effects in decreasing the MSE.
5.4.2 Evaluation on the Playing Strength

We used the value function together with MCTS to calculate the \( Q \) value as a mixture of value networks evaluation and results of simulations. We set \( c_{puct} \) as 5 and the mix parameter \( \lambda \) as 0.5. The maximum search time of each step is still 5 seconds.

<table>
<thead>
<tr>
<th></th>
<th>MCTS first</th>
<th>RC first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCTS wins</td>
<td>44</td>
<td>46</td>
<td>90</td>
<td>90%</td>
</tr>
<tr>
<td>Random choice wins</td>
<td>6</td>
<td>3</td>
<td>9</td>
<td>9%</td>
</tr>
<tr>
<td>Draw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 7 | Result of battles MCTS (policy + value) against random choice.

<table>
<thead>
<tr>
<th></th>
<th>MCTS first</th>
<th>WS first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCTS wins</td>
<td>25</td>
<td>20</td>
<td>45</td>
<td>45%</td>
</tr>
<tr>
<td>Weighted square wins</td>
<td>24</td>
<td>25</td>
<td>49</td>
<td>49%</td>
</tr>
<tr>
<td>Draw</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6%</td>
</tr>
</tbody>
</table>

Table 8 | Result of battles MCTS (policy + value) against weighted squared strategy.

There is no significant improvement of winning rate and even leads to a bit decrease. We tried to analyze the reasons, and suggested that it might because the results from simulations with rollout policy are more accurate when the game is close to the end, as its result is obtained from brute force which can exhaust all possible endings. Based on this assumption, we adjusted the algorithm to update our \( Q \) value as

\[
Q(s, a) \leftarrow \frac{(30 – depth)(1 – \lambda)w_p(s, a) + depth \cdot \lambda w_r(s, a)}{N(s, a)}
\]

where depth increases from 0 to 30 as the game goes. However, this adjustment did not influence the winning rates of the AI.

5.5 Random policy with MCTS

We also try the basic MCTS with random policy as both prior policy and rollout policy. As illustrated in the previous section, random policy is a constant function \( p(a|s) = \frac{1}{k} \) where \( k \) is the total number of possible values of current state \( s \). We also set the maximum search time as 5 seconds for each step and the \( c_{puct} \) as 5.

<table>
<thead>
<tr>
<th></th>
<th>MCTS first</th>
<th>RC first</th>
<th>Sum</th>
<th>Winning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCTS wins</td>
<td>50</td>
<td>50</td>
<td>100</td>
<td>100%</td>
</tr>
<tr>
<td>Random choice wins</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Draw</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 9 | Result of battles MCTS (random) against random choice.
Surprisingly this algorithm overwhelmed the weighted square strategy. It seems that the application of deep learning even hold back the performance of MCTS. After analyzing, we found several reasons that can help explain this phenomenon.

1. From the battle results of policy networks without search against random choice strategy, we can conclude that policy networks are much stronger than random choice strategy. However, when doing simulations, as the possible valid moves of Othello is restricted, the policy network may always choose the same move for the board and follow the same route of game tree, which makes the accuracy of the simulations biased.

2. The differences between prior probabilities of the policy network may hinder its exploration to some nodes. Once the simulation keep gave several bad results of a promising node with low prior probability, the MCTS may abandon this node and stop exploring it. However, in random strategy, all nodes have the same prior probabilities, so priors will not be a barrier to stop exploration. Also, due to the randomness, it is hard for random strategy to always give the same result, which always happens for policy networks as it is too inflexible.

3. The time to make a random choice is much less than the prediction of a CNN. Thus, within the same amount of time, random policy is able to do much more simulations than policy networks do. The imbalance between the simulation numbers may directly cause the huge differences in strength.

5.6 Discussions

We used the similar algorithms that implementing AlphaGo; however, the strength of the AI is not as strong as expected. There may exist problems in the dataset we used to train and the structures of our models. If larger sets and more delicate models are used, the strength of AI may have a breakthrough.

The disappointing performance may also result from the differences between Othello and Go: while random strategy is extremely bad for Go as the board is too large and the disc can be placed anywhere on the board, the rules of Othello guarantee that even random choice can flip at least one of opponent’s disc and the number of choices of each move is much less than that of Go. Also, the search space of Go is overwhelmingly larger than Othello. Thus, pruning is especially important for Go but not necessary for Othello.

Based on the analysis above, deep learning is useful to evaluate the game state or make
simulations based on current state. However, when it comes to problems of which sizes are small, it is too expensive to use such a strong tool and even brings about drawbacks.
6 Conclusions

This report has described the idea and implementation of our project, whose objective is to adopt the technology of deep learning neural networks to play Othello. Our expectation is that with the help of recent technologies, the program developed by us can achieve, if not transcend, the level of traditional algorithms. Regrettably, this aim has not been accomplished yet.

However, our DNN AI using MCTS and our enhanced traditional AI being well-matched in strength is quite inspiring. It proved itself that the trained deep neural network is indeed intelligent on the game and does have a strong potential (although still not tremendous enough) power in playing Othello as expected.

More importantly, this project might slightly discourage the hope to use deep learning on small problems: not only because it consumes more computations and time, but also the accuracy may not be comparable with manually designed evaluation functions.

This report is not intended to prove that deep learning is not suitable for Othello. The methods we tried are only a tiny part of possible ways to apply deep learning on Othello. In the future, we will keep trying other algorithms to explore other effective approaches to use deep learning with Othello.
References


Available: https://github.com/brilee/MuGo.
### APPENDIX I | Input features for neural networks

<table>
<thead>
<tr>
<th>Feature</th>
<th># of planes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disc color</td>
<td>3</td>
<td>Player disc / opponent disc / empty</td>
</tr>
<tr>
<td>Ones</td>
<td>1</td>
<td>A constant plane filled with 1</td>
</tr>
<tr>
<td>Zeros</td>
<td>1</td>
<td>A constant plane filled with 0</td>
</tr>
<tr>
<td>Valid moves</td>
<td>1</td>
<td>Valid moves for current player</td>
</tr>
<tr>
<td>Internal discs</td>
<td>2</td>
<td>Internal discs of player and opponent</td>
</tr>
<tr>
<td>External discs</td>
<td>2</td>
<td>External discs of player and opponent</td>
</tr>
</tbody>
</table>
### APPENDIX II | Structure of evaluation network $v_x(s)$ of Project Tempo

<table>
<thead>
<tr>
<th>Layer</th>
<th>Detailed Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input layer</td>
<td>64 kernels, each size 4*4, with border type = “same”, activation as sigmoid</td>
</tr>
<tr>
<td></td>
<td>128 kernels, each size 3*3, with border type = “same”, activation as sigmoid</td>
</tr>
<tr>
<td></td>
<td>Dropout layer with a dropout rate of 0.3 (optional)</td>
</tr>
<tr>
<td></td>
<td>Fully connected layer with 256 neurons, activation as $tanh$, initialization as uniform</td>
</tr>
<tr>
<td></td>
<td>Fully connected layer with 128 neurons, activation as $tanh$, initialization as uniform</td>
</tr>
<tr>
<td>Output layer with 17 neurons, activation as SoftMax, initialization as uniform</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX III | Structure of policy network $p_{SL}(a|s)$ and of Project Tempo

<table>
<thead>
<tr>
<th>Input layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ 128 kernels, each size 5*5, with border type = “same”, activation as ReLU, stride = 1</td>
</tr>
<tr>
<td>↓ 128 kernels, each size 3*3, with border type = “same”, activation as ReLU, stride = 1</td>
</tr>
<tr>
<td>↓ 128 kernels, each size 3*3, with border type = “same”, activation as ReLU, stride = 1</td>
</tr>
<tr>
<td>↓ 128 kernels, each size 3*3, with border type = “same”, activation as ReLU, stride = 1</td>
</tr>
<tr>
<td>↓ 128 kernels, each size 3*3, with border type = “same”, activation as ReLU, stride = 1</td>
</tr>
<tr>
<td>↓ 1 kernels, each size 1*1, with border type = “same”, activation as Softmax, stride = 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatten to 64 linear neurons</td>
</tr>
</tbody>
</table>
APPENDIX IV | Structure of value network \( v_p(s) \) and of Project Tempo

---

**Input layer**

↓

128 kernels, each size 5\*5, with border type = “same”, activation as ReLU, stride = 1  
↓

128 kernels, each size 3\*3, with border type = “same”, activation as ReLU, stride = 1  
↓

128 kernels, each size 3\*3, with border type = “same”, activation as ReLU, stride = 1  
↓

128 kernels, each size 3\*3, with border type = “same”, activation as ReLU, stride = 1  
↓

128 kernels, each size 3\*3, with border type = “same”, activation as ReLU, stride = 1  
↓

1 kernels, each size 1\*1, with border type = “same”, activation as linear, stride = 1  
↓

**Fully connected layer with 128 neurons, activation as linear, initialization as uniform**

↓

**Output layer**

**Fully connected layer with 1 neurons, activation as \textit{tanh}, initialization as uniform**

---

*The blue parts in Appendix III and Appendix IV are the same.*