THE UNIVERSITY OF HONG KONG

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE

FINAL YEAR PROJECT

Playing Othello by Deep Learning Neural Network

INTERIM REPORT

Team Members

CHAN Lok Wang : 3035072777

YIP Tsz Kwan : 3035068685

Supervised by

Dr. K.P. CHAN

22nd January, 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.

At this stage, data have been collected for the training process. Values has been assigned to the board configurations collected. A computer Othello with a predefined evaluation function has also been implemented. A simple fully-connected network has been implemented and tested for once. The result was quite close to one of our intermediate goals. The immediate next step is to redesign the neural network for training. This is the major challenge we encountered at this stage. Overall, we are slightly lagging behind the schedule. The intermediate goal should be achieved by the end of January but it is unlikely that the intermediate goal can be met by the deadline. The new targeted deadline to complete this task is mid-February.

Acknowledgement

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

Abbreviations

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

List of Figures

List of Tables

1. Introduction

Over the years, people have been trying to exploit the ability of a computer. One of such abilities is to make decision as if we human do. A way to show this capability is to play board games as it involves the evaluation of the current board settings and the selection of the next best moves constantly throughout the game. It is also a good way to evaluate how well a computer can perform compared with human.

Traditionally, a computer plays a board game against the opponent by the searching through the game tree, a tree containing all the possible moves with the corresponding weights. The best way to assign the weights is by propagating the final result given by the leaf nodes back to the upper layers.

However, due to the limited storage of a computer and the huge complexity of the game search tree, in practice, only a few lower layers will be constructed and stored in the memory. As a result, instead of propagating the final results to the upper layers, an evaluation function is used to assign the weight to the possible moves in a tree. [1] The problem of how accurate an evaluation function can assign the weight to the given board configuration arises, and this is where (deep learning) neural network proves to be useful.

Inspired by the human brain, neural network is a way of information processing. Several layers of nodes in the neural network are connected together. As the system is trained, the connections between the nodes will change. A large amount of training data is used to fine-tune the connection during the training process so that the network can produce a specific output corresponding to the given input. A deep learning neural network is a network with many hidden layers. [2] More features can be captured and studied in each hidden layer to increase the accuracy of the result.

This project aims to demonstrate how powerful deep learning neural network can be for gameplay by developing a computer Othello with board size being 8x8 using this technique. Othello is chosen as the size and the complexity of this game is suitable for a one-year long project with limited resources. The technique of deep learning neural network can also be applied to this game. More information on the rules of Othello can be found in *Appendix I*.

It is worth noting that when both sides play the game perfectly, it appears that the game will very likely end with a draw. [3] Therefore, the targeted winning

rate when playing against a moderate opponent is 50% and that when playing against a strong opponent is 35%. Moderate and strong computer opponents are yet to be discussed and defined with our supervisor during the neural network training, in around January.

There are two main types of neural networks for the game Othello in this project - the policy network and the value network. The value network, which is also known as the evaluation network, is used to evaluate the probability of winning given the board configuration. [4] The policy network, which outputs a probability distribution over the possible moves, is used for selecting the moves. [4] The value network for the game will be implemented first as the focus on this project is to develop the evaluation function without human logic. The policy network will only be implemented if time is allowed. More discussions on the software development cycle and the algorithms used will be included in *Section 5 Design and Procedure* of this report.

At this stage, 200 sets of game move records were collected, while 100 sets of them were collected by our team. Board configurations were extracted from the game move records collected. A value was assigned to each board configuration as its label for the input training data. A computer Othello was built with the traditional method with game tree and a pre-defined evaluation function. A simple fully-connected neural network was constructed and trained with data. Testing was carried out with the trained neural network. The result was quite close to one of the intermediate goals – playing against an "easy" computer opponent with a winning rate of 25%. Detailed intermediate goal can be found under *Section 2 – Objective and Benefits*. The immediate next step is to improve the neural network for training, which is also the current major challenge. Overall, we are slightly lagging behind the schedule as the intermediate goal should be achieved by the end of January. It is believed that the intermediate goal will not be reached by the targeted deadline. The new deadline for achieving this goal is set to be mid-February.

2. Objectives and Benefits

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 again 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 key of this project is to allow the evaluation function of the program being constructed by the program itself via the value network without human logic.

An intermediate goal is set to evaluate the interim results of the training. The intermediate goal 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 25% when playing against an easy (computer) opponent;
- A winning rate of 10% when playing against a moderate (computer) opponent; and
- A UI for easy playing

The intermediate goal is originally targeted to be achieved by the end of January. However, due to some previous delay in schedule, the new targeted deadline of achieving the above intermediate goal is set to be mid-February.

3. 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;
- Evaluation function being constructed by a value network to increase its accuracy;
- Selection of the next best move being done by the traditional game search tree; 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;
- Monte Carlo tree search algorithm being implemented for estimating the value of each state in a search tree; and
- Selection of the best next moves being implemented with policy network.

The following features will not be included in this project:

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

4. Deliverables

A program with a user interface (UI) that plays the game Othello with the following features will be delivered:

- Game of board size being 8x8;
- Winning rate of 50% when playing against a moderate opponent;
- Winning rate of 35% when playing against a strong opponent;
- Evaluation function being updated with the result of the deep learning neural network training; and
- Next best move being selected from the game search tree.

A report of the results obtained with different parameters and the corresponding analysis will be presented.

5. 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, evaluation functions, and neural network, will be introduced. The choice of the implementation language and the test specification can also be found in this section.

5.1 Software Development Cycle

As the main focus on the project is to optimize the evaluation function, the software development cycle adopted for this project should allow flexibility for modifying the evaluation function. A combination of the waterfall and agile model will be adopted for this project. (See Figure 1).

Requirements will be collected and thorough study and analysis on the current solution and strategy will be done at the inception phase of the project. 100 sets of training data will also be collected for the use in deep learning.

The game Othello will first be developed using the traditional method, in which the game is built with a game search tree and pre-defined evaluation function. This is to ensure that there is a computer Othello that can run and allow us to debug the program when new modules are added.

Then, an evaluation function for evaluating board configuration will be developed by deep learning neural network. It will also be incorporated into the existing game developed. After constructing a program with the evaluation function being implemented with the deep learning neural network, testing will be started with different sets of parameters. Each set of parameters will be used in the deep learning training for one week. Results will be evaluated and if necessary, the values of the parameters will be re-chosen to improve the performance. Agile model will be used to construct the evaluation function by neural network.

During the test, Monte Carlo tree search will also be implemented and incorporated into the program to improve the tree search performance. Test on the computer Othello will be carried out with the Monte Carlo tree search is being incorporated. Details of the Monte Carlo tree search will be provided under the section *Algorithm* (see Section 5.3.3.3).

If time is allowed for both implementation and testing, the policy network of the game will be implemented and incorporated in the program for the next best move selection.

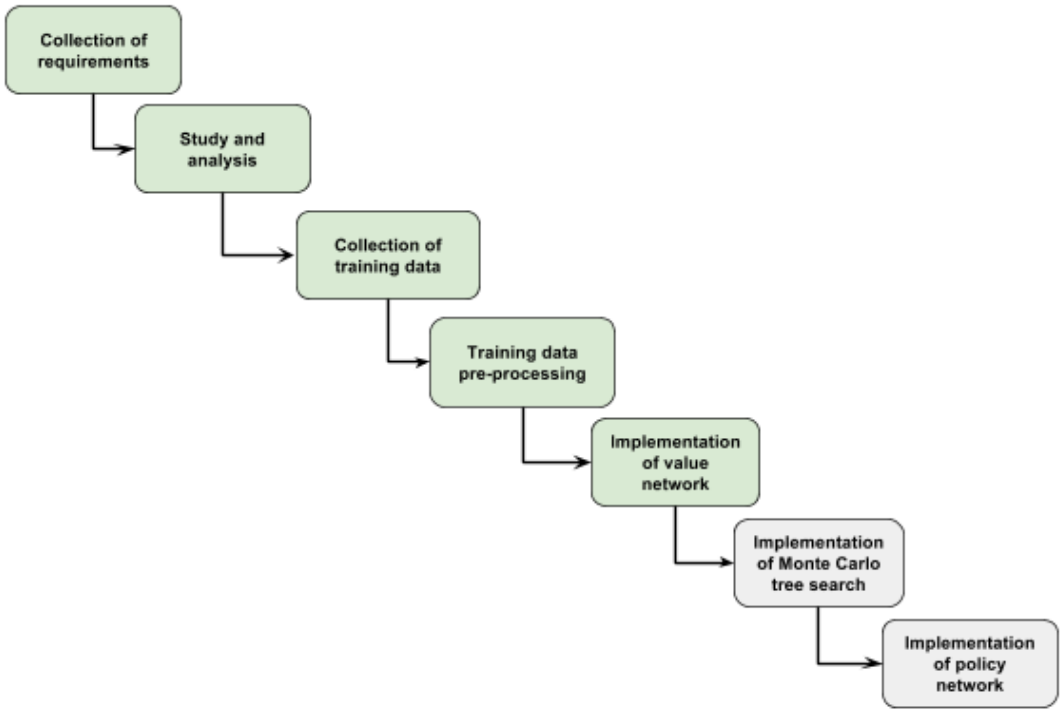

Figure 1: The waterfall model for the development process of this project. Requirements of the project will first be collected, followed by thorough study and analysis of the current solution. Board configuration of the game moves will be collected as the training data. Values will be assigned to the board configurations as labels for the training purpose in the training data pre-processing step. Value network will be implemented to obtain an evaluation function for the game. Agile process will be adopted for this stage. If time is allowed, Monte Carlo tree search algorithm and the policy network will be implemented for the tree search and move selection respective.

5.2 Training Data Collection and Processing

5.2.1 Training Data Collection

To facilitate the deep learning neural network training, 100 sets of training data are to be prepared from our group with the format as follows:

$$c_i x_i y_i$$

where c_i is an integer representing the color of the disc in the i^{th} move , with 1 being black and 0 being white. X, where $0 \leq X \leq 7$, is an integer representing the x-coordinate of the disc position in the i^{th} move, while Y, where $0 \leq Y \leq 7$, is an integer representing the y-coordinate of the disc position in the i^{th} move.

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

As these data are collected for training the neural network, the goal is to collect different board configurations for a game such that the computer Othello we develop can learn how to play the game better by learning to evaluate different board configurations.

With human playing against a computer, the game player can make a more flexible move than a computer as the move made by the computer is based on the computation of the evaluation function and select the best moves. When the evaluation function is not close to optimal, the move made by the computer may not favor winning the game. In this case, the flexibility of human playing based on one's personal experience can be an advantage in playing the game. Therefore, half of the games were played by human player against a computer opponent.

At the same time, an ordinary player may not be able to make the best move due to the limitation of one's experience in playing the game and one's vision on the effect on that move for the whole game. On the contrary, a computer can build a game tree that includes all the possible game moves and can make a

better move when searching through the game tree. As a result, half of the games recorded were played by computer against another computer opponent.

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 Othello, 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.

5.2.2 Training Data Processing

After collecting the game move records, a program is written to translate each move in a record to the corresponding board state (board configuration). The board state is represented by a string of 64 characters, with “0” representing an empty cell, “1” representing a cell occupied by a white disc and “2” representing a cell being occupied by a black disc. For example, the initial board state representation is as follows:

```
0000000000000000000000000002100000012000000000000000000000000000
```

After converting the moves in all the game records, depending on the result of the game, a value is assigned to each game state of that game. If white player wins the game, +1 score will be added to all the board states in that game record. If the white player loses the game, -1 score will be added to all the board states in that game record. If it is a draw, 0 will be added to all the board states in that game record.

5.3 Algorithms

Three types of algorithms - evaluation function, search algorithms and neural network, will be used in this project.

5.3.1 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 neural network is to minimize the difference between the input layer's label and output layer's result.

The neural network accepts a number of input nodes with their corresponding labels in the input layer and gives out the training result from the output node(s) in the output layer. The nodes in the input layer and the output layer are connected to the nodes in one or more hidden layers, where the training takes place (see Figure 2). A deep learning neural network is neural network with more than one hidden layer.

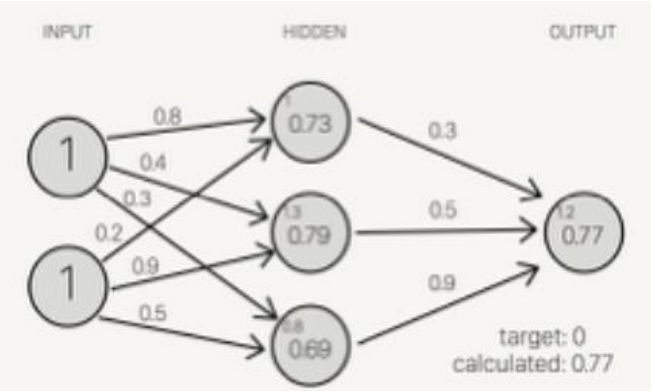

Figure 2: An example of a neural network with three layers. There are two input nodes in the input layer. The input nodes are connected to three nodes in the hidden layer with assigned weight. The hidden layer's nodes are connected to the only output layer's node. The activation function of this example 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 is not desirable, the set of weights and bias will be adjusted by the process "back-propagation". In back-propagation, the calculation for each bias is similar to that in forward propagation. Though, the calculation starts from the output layer instead of the input layer. The adjustment of the weights and biases are determined by the error function and the learning rate of the neural network.

5.3.2 Evaluation Function

In this project, the evaluation function will be implemented with a value network. The simple flow of building the evaluation function is demonstrated in the figure below (see Figure 3).

The training data collected in the pre-processing phase are the sets of board configurations of different gameplays. During the learning phase, the learning model will be configured with suitable parameters and learning rate so that it can evaluate the different board states. The final model will be constructed and delivered in the evaluation phase.

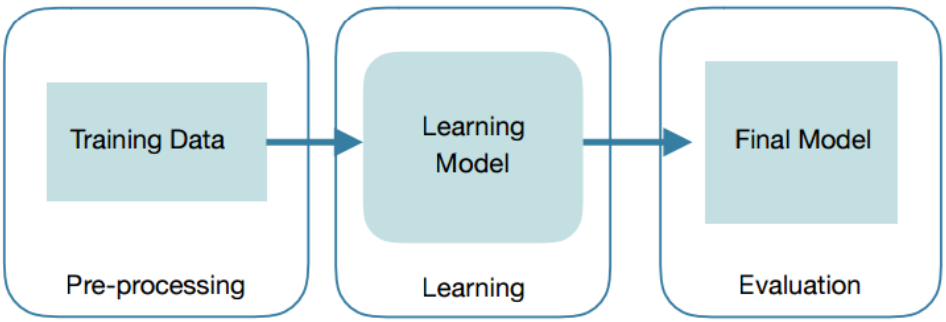

Figure 3: The flow of building an evaluation function with value network. There are three phases in building an evaluation function, namely pre-processing, learning and evaluation. Training data will be collected in the pre-processing phase. A learning model will be built and trained with the data collected in the first phase during the learning phase. The final model will be obtained at the evaluation phase, which is the last phase of building an evaluation function with value network.

5.3.3 Search Algorithms

The search algorithms, minimax algorithm and alpha-beta pruning will be implemented in the computer Othello. Monte Carlo tree search algorithm will also be implemented if time is allowed.

5.3.3.1 Minimax Algorithm

When developing the computer Othello with traditional method, a brute-force approach of minimax algorithm will be applied in searching the game tree to find the best next move. This algorithm can be applied to two-player games in which both players know everything about the next possible moves.
Examples of such games are tic-tac-toe and Othello.

Weights are assigned to all the board configurations in 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.

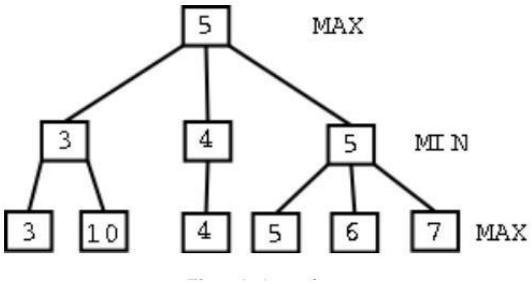

Figure 4: A game search tree with minimax algorithm. The direction of applying the minimax algorithm is from the bottom to the top of the game tree. At the bottom level, since it is the MAX level, the MAX player should choose the node with the maximum value. Therefore, the MAX player will choose the move evaluated to the weight of 10. Then, for the MIN player's turn, the MIN player will choose the move with node having the smallest weight, which is 3 in this game tree. At last, since there is one node that the MAX player can choose, the MAX player has to choose the node with weight evaluated to 5.

5.3.3.2 Alpha-beta Pruning

The alpha-beta pruning algorithm will also be implemented to eliminate the unfavorable moves in the game tree. The full minimax search explores some part of the tree that is not necessary to explore. Alpha-beta pruning is an algorithm to avoid searching the parts of a tree with nodes' value not lying in the range [min, max]. [5] This algorithm can increase the efficiency of searching the game tree and is usually used together with the minimax algorithm for game tree search.

5.3.3.3 Monte Carlo Tree Search

Monte Carlo tree search (MCTS) is another search algorithm to be applied in this project. It is a heuristic algorithm that uses random simulations to selectively grow a game tree. [6] It uses Monte Carlo rollouts to estimate the value of each state in a game search tree. With more simulations, the search grows larger and the value of each state becomes more accurate, allowing the game to converge to an optimal play and the evaluation converge to an optimal evaluation function. [4]

5.4 Implementation language

The implementation language for this project should support both functional design and GUI design. For functional design, there must be some machine learning libraries or frameworks that support deep learning neural network for the programming language. Python is chosen as the programming language for this reason.

Python, first released in 1991, is a general-purposed, interpreted, dynamic programming language. It supports different programming paradigms, including object-oriented, imperative and functional programming, which facilitates both the GUI design and the functional design of the game. [7]

There are a number of machine learning libraries available in Python. In particular, Keras, a neural network library written in Python, is used in our project. It can use Theano or TensorFlow as the backend library, where both are common libraries used in neural network training. This can help write the deep

learning model easily. [8] A bonus of Python is its simplicity, which can help simplify our code and make it clear and easy to understand.

5.5 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. 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 six levels of difficulties as shown in Table 1.

Level 1 - Level 5	Level 6 - Level 10	Level 11 - Level 15	Level 16 - Level 20	Level 21 - Level 25	Level 26 - Level 30
Very easy	Easy	Moderate	Above Average	Strong	Extremely strong

Table 1: Level of difficulties of The Othello. Level 1 to Level 5 was considered as "very easy". Level 6 to Level 10 was considered as "easy". Level 11 to Level 15 was considered as "moderate". Level 16 to Level 20 was considered as "above average". Level 21 to Level 25 was considered as "strong" and Level 26 to Level 30 was considered as "extremely strong".

The two types of opponents will be tested with 50 games each. 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%.

There are two reasons for the decision on the winning rate.

Firstly, it is believed that when both sides play the game perfectly, the game will very likely end with a draw. [3] Hence, when the winning rate of 50% is achieved, the computer Othello we developed can be said to be comparable with the existing computer Othello developed with the traditional method. Therefore, the winning rate of 50% when playing against a moderate opponent is chosen.

Secondly, the evaluation function obtained from the neural network may not be optimal due to the limited time for training and hardware availability.

To obtain an optimal evaluation function, the neural network usually needs to be trained for some time to obtain the optimal set of bias and weights. The training parameters, e.g. the learning rate, will also affect the time of training. If the training parameters are not set well, the problem of over-fitting will arise.

Therefore, more time is required for training the neural network to get the optimal evaluation function.

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. The relatively low computational power of the CPU results in longer training time. With limited time for this project, the evaluation function obtained from the neural network may not be optimal.

Besides, the number of distinct board configurations we obtained as the training data may not be large enough for constructing a good evaluation function. These factors will affect the performance of the computer Othello we are going to develop, and hence, a lower winning rate of 35% is set when our computer Othello plays against a strong computer opponent.

6. Division of Work

The responsibilities and the proportion of work are listed in Table 2 below:

Member	Responsibilities	Proportion
Chan Lok Wang	Preparation of the training data	50% (50 games played)
	Design and implementation of the model in building the evaluation function of the game tree	50%
	Design and implementation of the decision making algorithm	100%
Yip Tsz Kwan	Preparation of the training data	50% (50 games played)
	Design and implementation of the model in building the evaluation function of the game tree	50%
	Front-end development, including the UI design and the graphical user interface (GUI) gameplay environment	100%

Table 2: Division of work among the team members

7. Schedule of Tasks

The schedule and overview status of the project is shown as Table 2 below:

Date	Task(s)	Deliverables	Remarks
Sep 2016	1 Self-study on 1.1 Game tree 1.2 Deep learning neural network 1.3 Strategies of playing Othello 2 Preparation of training data		Completed
2 Oct 2016		1. Detailed project plan 2. Webpage 3. 100 sets of training data	Completed
Oct – mid-Nov 2016	1. Assignments of values for the board configurations obtained from training data 2. Implementation of the game with traditional method with UI	1. Delivery of the computer Othello implemented with traditional method	Completed
Dec 2016	1. Implementation of the value network for the evaluation function	1. Delivery of the computer Othello with evaluation function implemented with value network	Completed
9-13 Jan 2017	1. Interim presentation		Completed
22 Jan 2017		1. Delivery of the interim report	Completed
31 Jan 2017	1. Intermediate goal achievement		In progress
Jan - Feb 2017	1. Implementation of the GUI of the game 2. Implementation of the Monte Carlo tree search algorithm	1. Delivery of the computer Othello with evaluation function implemented with value network and MCTS implemented	
Mar 2017	1. Finalized optimization		
16 Apr 2017		1. Delivery of the final report	
02 May 2017		1. Project exhibition and presentation	

Table 3: The schedule and current status of the project

8. Current Progress

100 sets of game move records were prepared by our team with the format specified in Section 5.2. The records were uploaded to a server and shared with other groups doing the final year project with the same topic. In total, 200 sets of game move records were available for training. Board states obtained from the 200 sets of move records were assigned with a value according to the procedure described in *Section 5.2.2*. These board configurations with their corresponding values will be used in the neural network training. A computer Othello was implemented with a game tree and a pre-defined evaluation function. A simple fully-connected neural network was constructed and trained. It was tested against a random player, a "very easy" player and an "easy" player. The design, specification of the test environment and the test results are analyzed in section 8.1.

At this stage, the immediate next step is to improve the neural network for training by re-designing the neural network. At the same time, a GUI will be implemented for better user experience. Overall, we are slightly lagging behind the schedule.

8.1 Result of the First Implementation

8.1.1 Design of the Neural Network

The first version of the neural network was implemented to (purpose). The neural network built was a fully-connected neural network with 5 layers in total, including an input layer and an output layer. The activation function used was the rectified linear unit, which its approximation is the analytic function $f(x) = \ln(1 + e^x)$. The loss function used was the mean-squared error. Detailed modal structure and training environment can be found in Table 4 and Table 5 respectively. The neural network was implemented with a Python interface Keras. Keras can use Theano or TensorFlow as the backend library [10], which facilitates the implementation of deep learning neural network. The total number of board configurations used for training was 10487. 10% of the total number of board configurations (1048) was used as the validation set. The first trial of training was carried out on a MacBook Pro. Detailed specification can be found in Table 6.

Model Structure	
Number of inputs in the input layer	64 inputs
Number of outputs in the output layer	1
Number of hidden layers:	3
Activation function	Rectified linear unit

Table 4: Model structure of the first training. 64 inputs were taken in the input layer and one output was obtained in the output layer. Three hidden layers were constructed in the fully-connected neural network. The activation function used was rectified linear unit, with the approximate analytic function $f(x) = \ln(1 + e^x)$.

Training Parameters	
Initial weight method	Normal
Loss function	Mean squared error
Batch Size	100
Epoch	5000

Table 5: Training parameters of the first training. The initial weight method is set to be normal. The loss function used was mean square error. The batch size selected was 100. The epoch set was 5000. The parameters for the trial training were chosen with some guess to test the performance of training.

Running Environment	
Machine	MacBook Pro (Retina, 13-inch, Early 2015)
Processor	2.7GHz Dual-core Intel i5
Memory	8GB 1867MHz DDR3

Table 6: The running environment specification for the test. The test was carried out on a MacBook Pro with 2.7GHz Dual-core Intel i5 for processor and 8GB 1867MHz DDR3 for memory.

8.1.2 Result of the First Training

The training time taken was about 4 minutes and 30 seconds. The training result was satisfactory. The accuracy was around 97%, but the validation data accuracy was only 73%. Detailed result is shown in Figure 5.

Epoch 4999/5000
9438/9438 [====] - 0s - loss: 0.4778 - acc: 0.9704 - val_loss: 0.9585 - val_acc: 0.7350
Epoch 5000/5000
9438/9438 [====] - 0s - loss: 0.4755 - acc: 0.9720 - val_loss: 0.9543 - val_acc: 0.7312

Figure 5: Statistics of the first trial. “loss” and “acc” are the average loss and accuracy for the training data set respectively at each epoch while “val_loss” and “val_acc” are the average loss and accuracy for the validation set at each epoch.

The model and the weights were then saved and used as a utility function in evaluating board configuration of the Othello program.

8.1.3 First Testing with the Value Network

An AI using the trained value network as the evaluation function was built. Simple minimax search with depth level 1 was applied. The built AI (Oneply) was tested with a player picking moves randomly (Rand), a very easy player (Lvl 5) and an easy player (Lvl 10) for 10 games each. The result is shown in Table 7.

	Rand	Lvl 5	Lvl 10
Oneply (Black)	4,6,0	9,0,1	2,8,0
Oneply (White)	5,5,0	9,1,0	3,7,0

Table 7: Result of the testing games with value network. The three numbers in each cell represents the number of game won, lost and draw for “Oneply (Black)”, “Oneply (White)” respectively.

Oneply performed very well in playing against a very easy player. However, its strength was no better than a player picking a move randomly and the winning rate when playing against an easy player dropped to around 20% to 30%. More details of the gameplay result can be found in Appendix II. The results will be analyzed in Section 8.1.4.

8.1.4 Limitations and Solutions of the First Training and Testing

The sudden drop in winning rate of the built AI when playing against an easy player may be due to 3 reasons. The first reason is the insufficient and inadequate training data set in reflecting different board states accurately. The second reason is the problem of over-fitting in the value network. The last reason is the insufficient depth of the search algorithm.

8.1.4.1 Training Data Set

The training data may not be sufficient for the neural network training in terms of quantity and quality. As for quantity, in the first training, only 10487 of the board states in 200 sets of game move records were used to train the neural network, which was insufficient for the training. As for quality, about 50 sets of the game move records used were from the unprofessional human playing and computer playing. This might also contribute to the performance of the training result. Besides, the source of game records provided from other teams are unknown. Hence, the quality of the training data cannot be guaranteed, which may hinder the performance of the training result.

To address the quantity problem, more training data will be collected during the process of the training and from external playing. Moves will be recorded during each game during the training process and will be processed and used as the training data for future training. Besides, external game records will be extracted from the Othello World Cup 2013. The record will be processed and used as the training data for the neural network. Since the records were the games played by professional Othello players, using these record in the training can increase the reliability and ensure the quality of the training data, and hence the performance of the training.

8.1.4.2 Value Network

The relatively lower validation data accuracy may imply the occurrence of over-fitting. For this reason, the network may not evaluate different board state and make correct prediction of its value correctly. Network model will be re-designed again

with educated guess to address this problem. Furthermore, hyper-parameters will also be fine-tuned until a more satisfactory result is obtained.

8.1.4.3 Search Algorithm

Only one level down was explored in the tree search using Minimax algorithm, which was insufficient for accurate result. To address this problem, the depth of the tree search will be increased to 4 layers. This can increase the accuracy for the evaluation using Minimax algorithm during the tree search process for selecting the next best move.

9. Challenges

At this stage, the two major challenges of this project is the optimization of the deep learning training.

Parameters are set for the deep learning process to determine the rate of learning and adjustments. Regardless of the default setting of the rate of learning and the update of the parameters, the time taken for each learning process with a set of pre-defined parameters may take longer than expected due to the limited computational power of an ordinary CPU.

In practice, GPUs are used for deep learning as it favors matrix multiplications, which is one of the most essential and important operations in deep learning algorithms. Compared with a CPU, the training time required for a GPU can be 8.5 to 9.0 times faster. [9] However, due to limited resources, we are not able to use a GPU for the project. Instead, less powerful CPUs will be used. The time taken for each deep learning training process will therefore be lengthened, and hence the overall optimization process, which may result in the slippage of schedule.

To address this problem, the game and the evaluation function will be implemented earlier in December and more time will be reserved for optimization.

10. Future Plans

Improvement in performance and user experience are the focus of this project in the second semester.

In terms of performance, the main focus is to redesign the value network. Different parameters will be set during the training. The level of game tree to be explored during the tree search will be increased from one to four to improve the accuracy of the training result.

If time is allowed, Monte Carlo's tree search algorithm will be implemented for a more efficient tree search. The policy network will also be implemented for the move selection.

In terms of user experience, in addition to the existing command line program, a graphical user interface will be implemented for easier playing and better user experience.

11. Conclusion

This project aims to develop a computer Othello of size 8x8, where its evaluation function being constructed without any human logic involved. The machine learning technique to be used to achieve this purpose is deep learning neural network. The targeted winning rate is 50% when playing against a moderate opponent, and 35% when playing against a strong opponent.

At this stage, 200 sets of game moves have been collected as the training data. The board configurations from the game move records have been assigned with a value as the label for the neural network training. A computer Othello has been built with a game tree and a pre-defined evaluation function. A value network has also been constructed, trained and tested, with a quite satisfactory result. The immediate next step is to redesign the value network for the performance improvement. This is the current major challenge.

12. References

[1] David Eppstein, ‘Minimax and negamax search’, David Eppstein. 17-Apr-1997 [Online]. Available: https://www.ics.uci.edu/~eppstein/180a/970417.html. [Accessed 15-Sep-2016]

[2] Emerging Technology from the arXiv, ‘Deep Learning Machine Teaches Itself Chess in 72 Hours, Plays at International Master Level’, MIT Technology Review. 14-Sep-2015 [Online] Available: https://www.technologyreview.com/s/541276/deep-learning-machine-teaches-itself-chess-in-72-hours-plays-at-international-master/. [Accessed 15-Sep-2026]

[3] Othello Sky, ‘Chapter 11 Book openings’, Othello Sky. [Online] Available: http://www.soongsky.com/en/strategy2/ch11.php [Accessed: 17-Sep-2016]

[4] D. Silver, Mastering the game of Go with deep neural networks and tree, Nature, Vol. 529. 28-Jan-2016 [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html. [Accessed 10-Oct-2016]

[5] Cornell University, Minimax search and alpha-beta pruning, CS312 Recitation 21, Cornell University. Spring-2002 Available: https://www.cs.cornell.edu/courses/cs312/2002sp/lectures/rec21.htm [Accessed 15-Oct-2016]

[6] M. Magnuson, Monte Carlo Tree Search and Its Applications, Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal: Vol. 2, Iss. 2, Article 4. 2015. [Online] Available: http://digitalcommons.morris.umn.edu/cgi/viewcontent.cgi?article=1028&context=horizons [Accessed 13-Oct-2016]

[7] A.M. Kuchling, Functional Programming HOWTO – Python 3.5.2 Documentation’, A. M. Kuchling. [Online] Available: https://docs.python.org/3/howto/functional.html. [Accessed 13-Sep-2016]

[8] LISA lab, University of Montreal, ‘Deep Learning Tutorial’, LISA lab, University of Montreal. 01-Sep-2015 [Online]. [Accessed: 03-Sep-2016]

[9] Larry Brown, ‘Deep Learning with GPUs’, Larry Brown. Jun-2015 [Online]. Available: http://www.nvidia.com/content/events/geoInt2015/LBrown_DL.pdf. [Accessed: 18-Sep-2016]

[10] Dylan Drover, ‘Keras: An Introduction’, Dylan Drover. 02-Dec-2015 [Online]. Available: https://uwaterloo.ca/data-science/sites/ca.data-science/files/uploads/files/keras_tutorial.pdf [Accessed: 29-Dec-2016]

Appendix I - The game of Othello

Othello is a strategy game played by two players in black and white discs. The standard board size is 8x8. The game starts from the board configuration with the four centered squares occupied as follows:

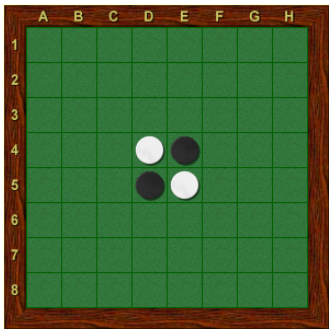

Figure 6: Initial board position of Othello

The black player makes a move first, and the white player follows. A legal move requires players to place a disc on an empty square adjacent to his opponent's disc. Opponent's discs sandwiched between the newly placed disc and another disc of the same color along a line in any direction (horizontal, vertical, or diagonal) are flipped to match with the color of the newly made move. An example of a legal move is shown below:

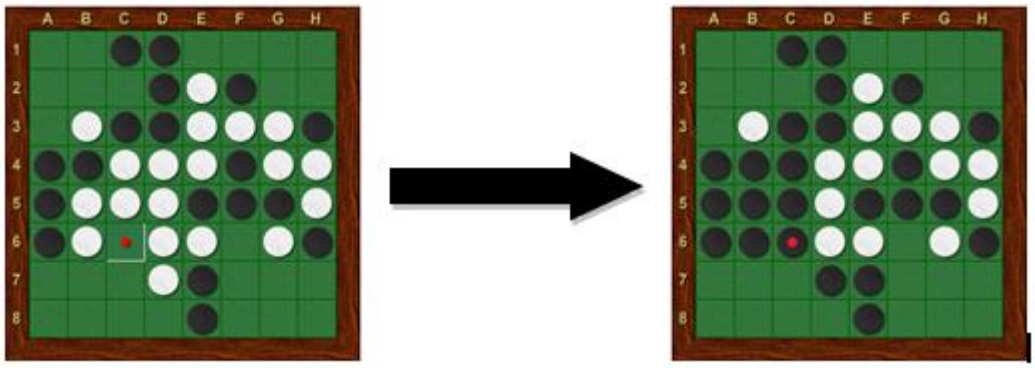

Figure 7: A legal move in the game Othello. Black player makes a move at the empty space C6, flipping the disc at positions B5 across the horizontal line, B6 across the diagonal line, and C4 and C5 across the vertical line.

The game continues until no legal moves are available. When the game terminates, the player who has more discs on the board is the winner. When both players have the same amount of discs on the board, they tie the game.

Appendix II - First Gameplay Testing Result

The following tables show the number of discs of each player when the game terminates:

	Rand (Black)	Oneply (White)
#1	21	43
#2	37	27
#3	24	40
#4	36	28
#5	37	27
#6	42	22
#7	30	34
#8	19	45
#9	29	35
#10	34	30

	Oneply (Black)	Rand (White)
#1	28	36
#2	26	38
#3	44	20
#4	38	26
#5	34	30
#6	35	29
#7	31	33
#8	17	47
#9	15	49
#10	20	44

Table 8 (left) & Table 9 (right): Results of the game played by the AI trained with deep learning neural network against a random player.

	Level 5 (Black)	Oneply (White)
#1	27	37
#2	34	30
#3	27	37
#4	21	43
#5	14	50
#6	25	39
#7	19	45
#8	26	38
#9	20	44
#10	27	37

	Oneply (Black)	Level 5 (White)
#1	32	32
#2	45	19
#3	45	19
#4	54	10
#5	44	20
#6	34	30
#7	40	24
#8	54	10
#9	48	16
#10	50	14

Table 10 (left) & Table 11 (right): Results of the game played by the AI trained with deep learning neural network against a “very easy” player.

	Level 10 (Black)	Oneply (White)
#1	19	45
#2	40	24
#3	14	50
#4	40	24
#5	49	15
#6	38	26
#7	30	34
#8	37	27
#9	44	20
#10	49	15

	Oneply (Black)	Level 10 (White)
#1	16	48
#2	22	42
#3	36	28
#4	26	38
#5	15	49
#6	17	46
#7	43	21
#8	19	45
#9	21	43
#10	18	46

Table 12 (left) & Table 13 (right): Results of the game played by the AI trained with deep learning neural network against an “easy” player.