INTERIM REPORT

PLAYING OTHELLO BY DEEP LEARNING NEURAL NETWORK

NG Argens (3035072143)

ABSTRACT

Computer Go has always been considered a major hurdle for Artificial Intelligence development [1] due to its enormous number of possible moves, which leads to its large degree of freedom. This hurdle was overcome in October 2015 when AlphaGo became the first Computer Go program to beat a professional human Go player without handicaps on a full-sized 19x19 board. On July 18th, 2016, it was even rated by Go Ratings as number one in the world according to its Elo [2]. Its most recent activity was playing on various Go playing websites against the best players in the world at the end of 2016, resulting in a final score of 60 wins out of 60 games.

This project aims to replicate the success of AlphaGo in the game of Othello (a.k.a. Reversi) under the name OthelloBlitz. Using similar algorithms and components, but with smaller board size and degree of freedom, the overall complexity of the problem is simplified. We can then further the research and study the effect of different learning algorithms and neural networks on computer's performance in playing games.

This paper will describe the design and implementation of OthelloBlitz. It will then explain the main algorithms and justify the choices used in the design. Finally, it will showcase our progress of the project, which is the Graphical User Interface, game rule implementation, data collection and value network.

ACKNOWLEDGEMENT

I would like express my greatest gratitude towards Dr. K. P. Chan, my mentor, who has supported and guided me throughout the project. This project would not have such progress without his support and guidance. I would also like to thank the department of Computer Science for their generous support in equipment and organization, Ray Kurzweil who has inspired me to pursue in the field of Artificial Intelligence, and the selfless contribution of the Google DeepMind team in the development of AlphaGo.

TABLE OF CONTENTS

INTRODUCTION

This interim report serves the purpose of reporting the progress of my Final Year Project named "Playing Othello By Deep Learning Neural Network". The project is conducted under the supervision of the Computer Science Department of the University of Hong Kong and in particular, my mentor, Dr. K.P. Chan.

The main goal of my project is to investigate and mimic a highly successful Computer Go program called AlphaGo but with a different game. The game chosen is Othello as its simplicity would allow us to create an AlphaGo-like program without having to use a supercomputer.

Among the various features, our project would be particularly focused in the effect of deep learning neural network and the impact of its performance as its structure is being altered. This is because deep learning neural network is believed by many to be the key feature behind AlphaGo's success. We hope that by using the same principle on different kinds of neural network, we can better understand the strength and weaknesses of them in computer gaming.

In the following sections, I will start by explaining the structure of AlphaGo, followed by the structure hence devised for our program, named OthelloBlitz. Then we will talk about the algorithms chosen and the rationale behind our choices. Finally, I will showcase the progress on Graphical User Interface, game rule implementation, data collection and value network.

PREVIOUS WORK

This project is greatly inspired by AlphaGo and hence it will be discussed in details in the coming sections. AlphaGo is the best Computer Go program in the world and the first to defeat a human professional player in a 5-game series. It was ranked number one in July 2016 by Go Ratings, a website dedicated to ranking Go players, humans and computers alike.

OVERALL STRUCTURE

The main structure of AlphaGo is the Monte Carlo Tree Search (MCTS). It is accompanied by two deep learning neural networks, namely the policy network and the value network [3]. The former one is developed for reducing branching factor, while the latter for reducing depth of search when necessary. While the policy network is accurate, a fast rollout policy is developed for side-by-side comparison [3]. Supervised learning and reinforcement learning are used as standard machine learning approaches.

MONTE CARLO TREE SEARCH

MCTS is a general game tree search without branching at all. Instead, each probe is conducted all the way to the end by moving randomly at each state [4]. The rewards and visit count are then back propagated to better enhance the next probe [4]. This continues until time runs out, which makes it ideal for playing under time constraint. However, as we can easily tell, moving randomly

does not match a rational player (or opponent) and thus policy network is employed to better reflect reality.

POLICY NETWORK

For AlphaGo, 13-layer policy network was trained from 30 million positions from the KGS Go Server [3]. The network is provided game states and the next move by human expert and would gradually learn to predict this move [3]. The result was an accuracy of up to 57%, compared with a maximum of 44.4% for other research groups at that time [3]. However, it takes 3ms for computation, which hence limits the traversed depth of MCTS [3].

ROLLOUT POLICY

Rollout policy is a fast algorithm that analyzes known patterns on the board for the most probable moves. This is also trained with supervised learning of 8 million positions from human games on the Tygem server [3]. The rollout policy achieved a 24.2% accuracy using only 2 μs (0.067 % of Policy Network) [3]. In the end, both rollout policy and policy network are used in a parallel manner, and both results are taken into account [3].

VALUE NETWORK

Value network is used to truncate tree search when deemed necessary. When looking at a game, professionals could understand if the dark or white side was at an advantage. Value network is trying to do exactly the same. At first, full

game histories were provided to train the value network in a similar manner to policy network [3]. This led to overfitting and the machine actually "memorized" the training set [3]. The network was then trained with distinct states from different games played by itself and earlier versions [3]. The problem was then solved.

SUPERVISED LEARNING

Supervised learning is used whenever a program is asked to provide answers to questions. It is supervised in the manner that both answers and questions are provided in the training examples. The program can then predict answers from similar questions later on. It is thus commonly used and as we can see, applied for all three of the policy network, rollout policy and value network [3].

REINFORCEMENT LEARNING

Reinforcement learning basically means practicing. In the case of AlphaGo, the program played games again and again with its random former self, which can be the first or, for example, the 524^{th} version [3]. The program can then identify its mistakes and update its value network and policy network accordingly.

OUTLINE

In light of the above work, an outline for OthelloBlitz was devised. The main structure would remain to be the Monte Carlo Tree Search (MCTS). Whenever we need to truncate the tree, value network would be used to determine which intermediate output is more desirable. It will run alongside the policy network which could suggest generally common moves given current circumstance.

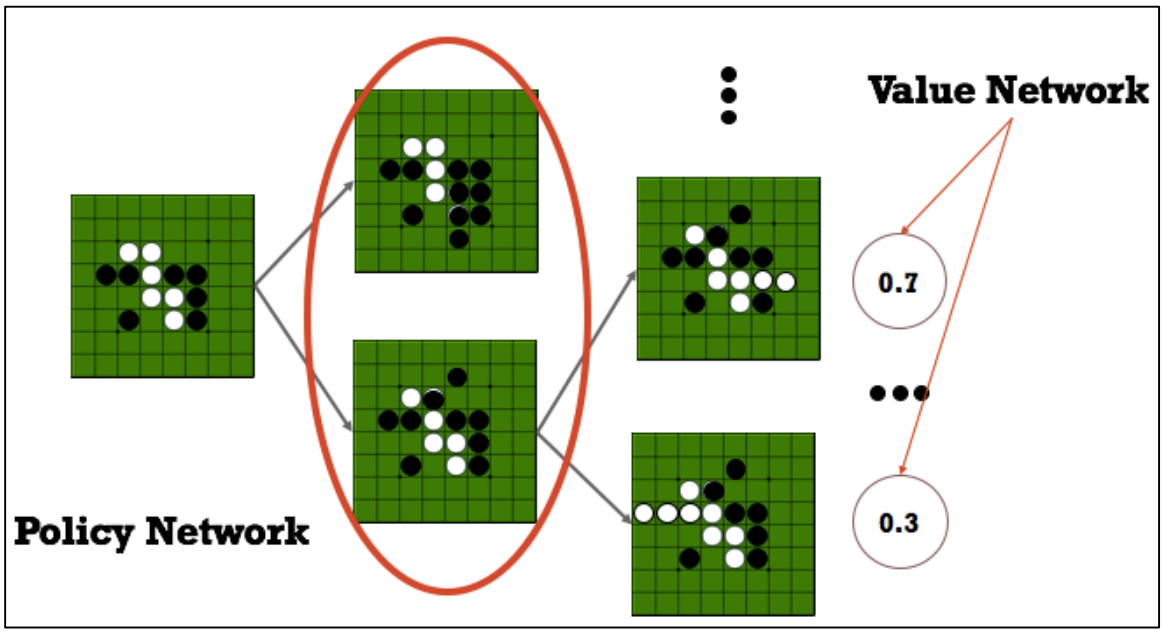

Fig 1. *An illustration of the structure of OthelloBlitz. While conducting a tree search, policy network reduces time taken simulating irrational moves while value network provides evaluation of current scenario whenever needed.*

POLICY NETWORK

For our policy network, we will use supervised learning to do the training. Our network will be provided with a state of game and a corresponding reasonable move made by human players. After going through a large number of such

pairs, the computer is expected to find the move given the state of game and more importantly, to propose a reasonable move given an unseen state of game. This is especially important when the complexity level of our program is low, and it plays at an unreasonably low level.

VALUE NETWORK

For value network, there will be two stages of training. In the first stage, where we would like to experiment on the training of neural networks, it will be trained as a classification network using supervised learning. The training data will consist of (gameState, outcome) pairs, with outcome of 1 representing the winning of the black side, 0 signifying that of white side and gamestate being features extracted from a random instant of the entire game.

This allows us to quickly compare the effectiveness of different structure of neural network such as layer composition and loss function. Also, since value network is to a great extent identical to the policy network, this rapid experimentation could theoretically speed up the training of policy network as well.

In the second stage, the value network would be trained as a regression network using unsupervised learning. The network will be trained by exploring the game tree by itself using policy network, during both gaming and non-gaming time. This should be a continuous progress and thus the network should be saved in between each update. Whenever a win or lose is recorded, all the intermediate steps should be updated and predictions made by value

network should be rewarded and punished according. For example, if a neuron inside the network contributed to a correct prediction, it will be more relied upon, while if it contributed to an incorrect prediction, it will be less relied upon.

DELIVERABLES

At the end of the project, we will submit the following deliverables so that others can test and investigate on our results.

MAIN PROGRAM

The main program is the OthelloBlitz program. It will have Monte Carlo Tree Search and the best performing networks implemented by default. However, it will also allow substitutions on the networks so that its performance can be investigated.

To facilitate the investigation of performance differences, our program would store a handful versions of itself for benchmarking, so that users can conduct a match on the user-modified version and older versions to observe increase or decrease in performance. The main program will also allow users to play against itself through Graphical User Interface.

SUPERVISED LEARNING MODULE

Supervised Learning Module (SLM) is the module we mainly train our policy network with. It will take in parameters such as training mode, iteration, training data as input and output a trained network. It should also provide the test function on test data to observe the predictability of our network. For easy documentation, our module will be able to output to the file system a binary

representation of the networks, from which a graphical representation can be created.

REINFORCEMENT LEARNING MODULE

Reinforcement Learning Module (RLM) would be responsible for the reinforcement learning process used in AlphaGo [3]. It would be able to take in 2 policy networks and make them play against each other. After a game, the network winning would contribute to the adjustment of the latest network parameters and even meta-parameters.

REPORT

Our last deliverables will be our report. After investigating which neural network design is the best under the scenario of computer gaming, we will present our findings, our design and our choices in the final report.

ALGORITHM

In this part, we will discuss some of the algorithm or keywords commonly used in our project.

MONTE CARLO TREE SEARCH

Monte Carlo Tree Search (MCTS) is a very interesting depth-first search algorithm in which there is no specific order. Instead the searching is largely conducted in random order.

At every search, the MCTS chooses a leaf according to its meta-algorithm and simulate until the end-of-game state is reached. The result is then propagated from the leaf back to the root, rewarding and punishing nodes on the way similar to that of machine learning. However, nodes in this case are not neuron nodes and as such has no weights and biases to update. Instead, its win-lose count is recorded and those with a lower winning probability is less often expanded.

DEEP LEARNING NEURAL NETWORK

Before gaining spotlight in the field of gaming, deep learning had already achieved over 90% accuracy in speech and handwriting recognition [5]. By instructing the machines to learn from their mistakes, speech recognition software discovered rules that even linguists are not aware of. This is all due to the structure of neural network, in which all input nodes can affect and receive feedback from all of the next-layer nodes, which can then affect and

receive feedback from all of its next-layer nodes, so on and forth. Layer-by-layer affecting would result in prediction, whereas layer-by-layer feedback simulates human learning. This then allows computer programs to “learn” from mistakes and samples to extract features suitable to computers.

In our project, we need to specify meta-parameters to alter the network under test. This usually means the number of layers in between input and output, as well as the number of nodes in each layer, the training algorithm, etc.

RATIONALE

There may be questions as to why we chose certain decisions when designing the project. Here are the reasons.

MONTE CARLO TREE SEARCH

Monte Carlo Tree Search is chosen for one main reason. Compared with other traditional searching algorithm, its performance does not fall significantly when computation time is limited. However, similar to traditional searching method, as time tends to infinity, MCTS still tends to finding the perfect move. Hence this makes it ideal against other tree searches such as Depth-First Search of Breadth-First Search.

PYTHON

As machine learning is a key feature in our project, at the planning phase, we looked into the popular choices for machine learning, such as R, Matlab and Python. While both R and Matlab are powerful analytical software, ideal for the heavy calculations in Machine Learning, Python is not far behind with its scikit-learn framework and SciPy package. Python also has the advantage for superior Graphical User Interface support which would be ideal for demonstrating our results and finding.

While Java and C can also accomplish similar work as Python while being more efficient, Python was still chosen for its simpler syntax and its large community in Machine Learning. This makes it ideal for rapid prototyping throughout a

year. We can also rely on the online community for the difficulties we encounter. Hence Python is chosen in the end.

PROGRESS

At the time of writing, I have completed the major skeleton in the OthelloBlitz main program. It is now able to play against human players via Graphical User Interface (GUI) using randomly selected moves from the set of legal moves. It can also detect illegal moves and cases when one of the players cannot make a move.

Sep – Oct, 2016	Traditional AI Approach	Tree Search	✖
		GUI	✔
		Game Rules	✔
Nov – Dec, 2016	Value Network	Training Data	✔
		Value Network	✔
Jan – Mar, 2017	Extended Research	Policy Network	✖
		Genetic Algorithm	✖
		Rollout Policy	✖
Apr – May, 2017	Final Product	Main Program	✖
		SLM	✖
		ULM	✖
		Report	✖

Table 1. *Current progress of our project.*

PROGRAM SKELETON

Using PyQt5, a simple GUI was devised. It is able to receive user input and put down pieces accordingly. It is also able to detect illegal moves and provide feedback to the users using the text bar at the bottom of the interface.

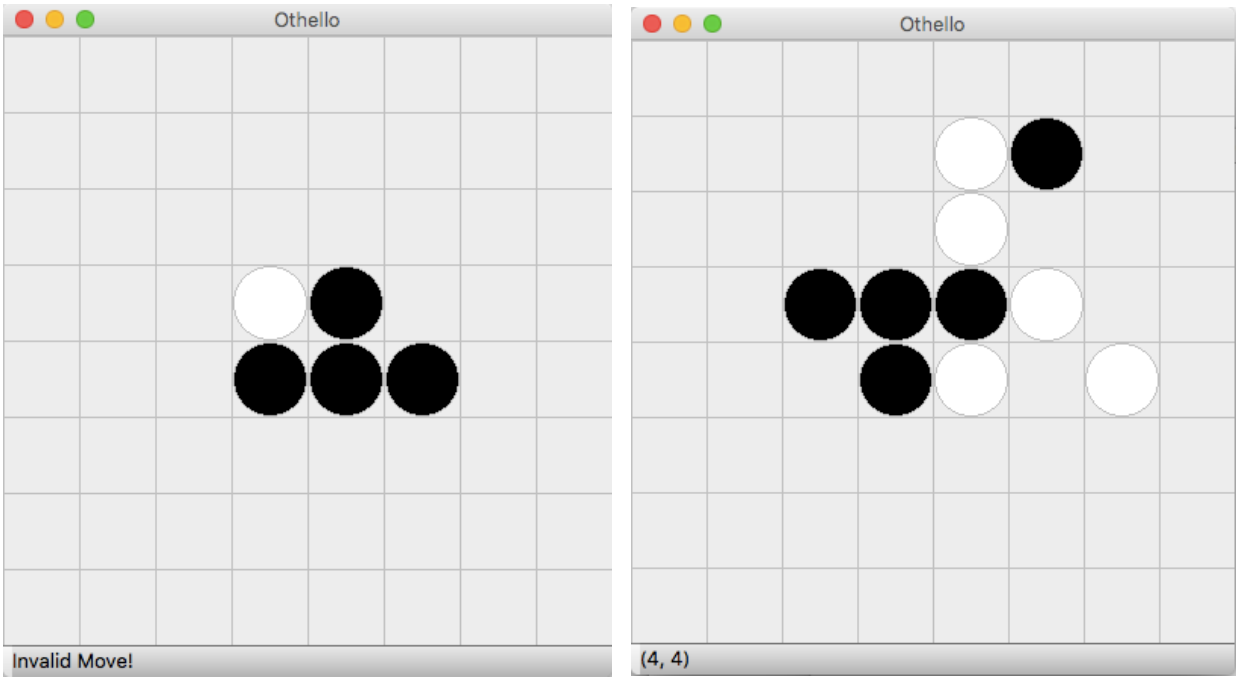

Fig2. *Graphical User Interface of OthelloBlitz developed with PyQt5*

TRAINING DATA

At first I used match data of 360 games from Othello World Cup 2013 to train the value network. However, despite continuous effort of redesigning the neural network structure, the test accuracy remains unsatisfactory, with a maximum of about 53.2%, slightly better than random guessing among two choices. As further adjustment of network only results in symptoms of either undertraining or overtraining, I changed the direction into finding more game data.

After searching for some time online, a large database called WThor was finally discovered on a French Othello website (Federation of French Othello: ffothello.org). It was however using a rather obsolete format called .wtb which was further modified to specifically store Othello match records. In the end, I wrote a code snippet in Python which decodes all the records from 1977 – 2016. This gives us more than 100,000 game records from which we can then train our networks.

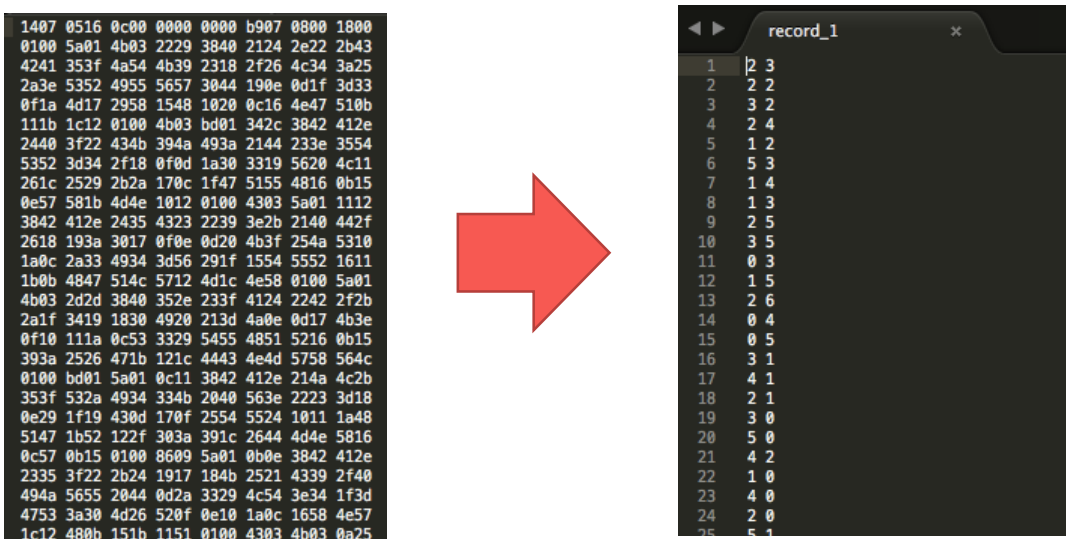

Fig 3. *Decoding of .wtb format* ***(left)*** *into Othello move record* ***(right)***

VALUE NETWORK (1)

As mentioned above, we started by using match data from Othello World Cup 2013 to train our value network. In each game, one random game state was chosen and our network was asked to predict which player eventually won. Using the training algorithm discussed above, our network was able to score an accuracy of 53.2% on unseen data, while standard heuristics of Othello only scored 40.3%.

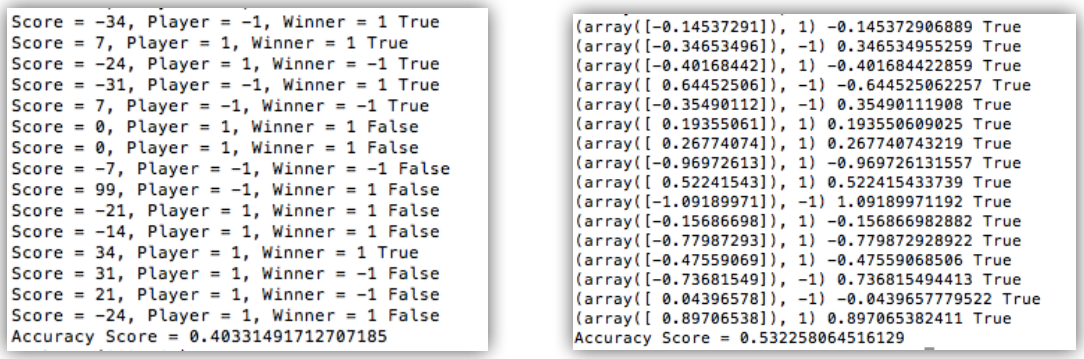

Fig 4. *Accuracy score of prediction of game result of standard heuristics* ***(left)*** *and Othello Blitz* ***(right)***

Fig 5. *Picture illustrating the calculation in standard heuristics. To calculate the score of a particular player, we add up the scores of the grids on which that player's pieces reside. The whole grid is filled using symmetry.*

VALUE NETWORK (2)

Then the match records from WThor were found and used. From the 100,000 match records, the first 2,000 were extracted as our training data due to limited memory space. With this increased data size, I was able to experiment with more network structure meaningfully. At this stage, I also decided to treat it as a regression network similar to that in AlphaGo. In the end, using Keras, a sequential network of 8 Convolution2D layers using Rectified Linear Unit as activation, followed by a fully connected dense network of 512 hidden nodes

were constructed. The input is a 3-channel representation of the gameboard, with the channels being black, white and empty respectively. The output is a single value with 1 being a win by the black side and 0 by the white side. The resulting testing MSE was 0.4947 and training MSE was 0.462. This is still remarkably distant from AlphaGo's result of 0.226 and 0.234, but is nevertheless an improvement from accuracy of 53.2%.

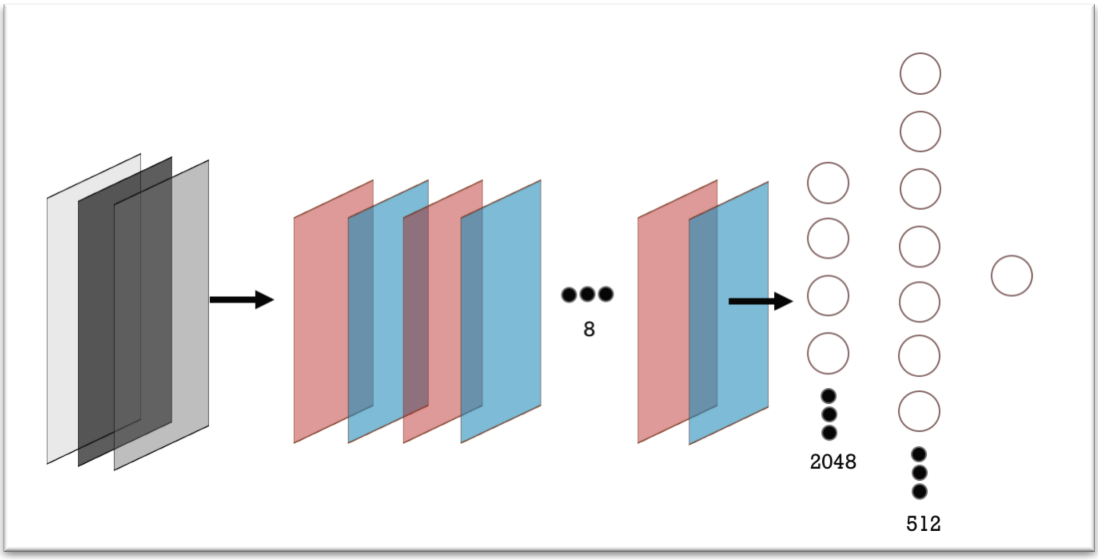

Fig 6. *Illustration of the current layout of Keras neural network*

FUTURE PLAN

In the second semester, the most important task would be implementing the standard tree search. This enables us to convert testing accuracy into playing performance, which is our primary target after all.

After that I would retry the AlphaGo approach of training a policy network, followed by reinforcement learning of policy network, and in the end a value network trained from the game results of well-trained RL policy network. While doing this, I hope that I can modularize the whole process, such that future testing and training can be conducted at a simple manner, ideally at a GUI panel.

LIMITATION

A huge limitation of my Final Year Project was computational power and memory space. They were crucial in any machine learning task but was critically lacking while I was using my MacBook Air 13” to do the task. Throughout the semester break, I have successfully migrated the whole project to my Desktop. This includes installing Linux, Theano, Keras, Tensorflow with GPU backend, and also the most important and time-consuming, NVIDIA driver and compiler. Luckily, these tasks were in the end successfully completed, and hopefully could expedite the process in second semester.

REFERENCES

1. Johnson G. To test a powerful computer, play an ancient game. The New York Times [Internet]. 1997 Jul 29. [cited 2016 Sep 17]; Technology: [about 5 screens]. Available from: http://www.nytimes.com/1997/07/29/science/to-test-a-powerful-computer-play-an-ancient-game.html?_r=0

2. West P. [updated 2016 Sep 07, cited 2016 Oct 26]. Available from: http://www.bnext.com.tw/ext_rss/view/id/1847944

3. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou J, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillibrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. Mastering the game of Go with deep neural networks and tree search. Nature [Internet]. 2016 [cited 2016 Sep 17]; 529. doi:10.1038/nature16961

4. Jeff B. [updated 2015 Sep 07, cited 2016 Sep 18]. Available from: https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

5. Kurzweil R. How to create a mind: the secret of human thought revealed. New York City (US): Viking Penguin; 2012