Final Year Project Plan

Project Title: Playing Othello by Deep Learning Neural Network

Author: Kuai Yu (David)

Supervisor: Prof. K.P. Chan.

UID: 2013507514

Abstract

On March 15th 2016, the Go-playing computer program known as AlphaGo took the world by surprise as it won its fourth and final game against international Go champion Lee Sedol[i]. In light of Google DeepMind's recent achievement, this project aims to create a board-playing artificial intelligence using similar approaches as AlphaGo but instead of Go, the program will play the much simpler board game Othello using a combination of tree search algorithms and neural networks. The program will be implemented in Python and will offer a simple graphical user interface to play against human challengers.

Project Background

The AlphaGo program combines Monte-Carlo tree search with deep neural networks and is trained by datasets of both human-played Go games as well as a series of games in which AlphaGo played against other iterations of itself[ii]. This unique architecture allows AlphaGo to effectively "learn" how to play Go overtime and improve its odds of winning with every game it analyses. Furthermore, unique properties inherent to any deeply-layered neural network enables the program to "predict" the odds of winning for any particular board layout based off data extrapolated from past games and grants AlphaGo flexibility in playing.

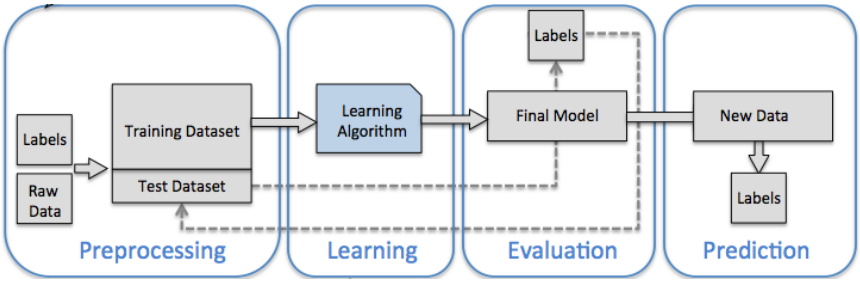


fig.1: The fundamental working principle behind any machine learning algorithm is its ability to adjust a matrix of weight values over time based on an input set of training data and the desired outputs. These weight values are then used in combination with new data to produce a prediction.

This project aims to create an artificial intelligence much like AlphaGo in that multiple neural networks will be optimized based on a set of training data and then used to predict the winning moves in a completely independent game. A concise description of this process is shown in figure. 1. However, instead of the game of Go, my program (henceforth named "IagoBot") will specialize in the game of Othello.

The layout of Othello is similar in appearance to that of Go. Except instead of a 19x19 board, Othello is played on a much smaller 8x8 board. The rules of Othello are also somewhat different: Although the two players still place alternating pieces of black or white stones on the board, a player may only place a stone if it forms a continuous horizontal, vertical or diagonal line with another stone of his color while sandwiching his opponent's stones in the process. Once this happens, all sandwiched stones are "captured" and transforms into the player's color. The game ends whenever the entire board is occupied by stones or neither player can make a legal move. At the end of the game, the player with the greater number of stones on the board wins. Otherwise it is considered a draw.

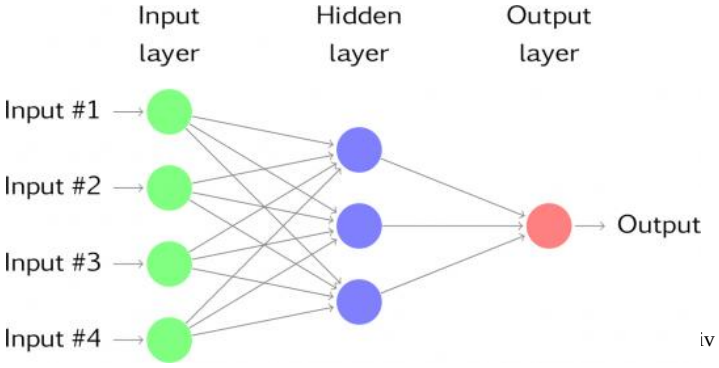
Owing to the small size of the 8x8 board, it is entirely possible to construct a strong Othello program by utilizing only tree search algorithms that are capable of analyzing every potential outcome produced by any legal move from a given board position[iii]. In fact, most existing

Othello programs do not require any neural networks or subsequent "training" to attain a near-perfect level of mastery over the game. Nevertheless, I feel that a deep-learning neural network approach to the game warrants my further investigation for two reasons: Firstly, it offers me valuable insight into the world of machine-learning, in which I have no prior experience. Secondly, it would be interesting to see whether a deep-learning Othello program holds any significant advantages over traditional tree-search programs, namely whether it is possible to produce a program that is less exacting on physical resources such as CPU / RAM usage, as well as intangible resources such as time.

Project Objectives & Deliverables

Toward the end of my project timeline, I aim to produce a single piece of portable program that will satisfy the following objectives and criteria:

Firstly, the key technology behind the program should be two neural networks that work in tandem to evaluate a game of Othello. Although the project specification stipulated the use of deep learning neural networks, I still analyzed the advantages and disadvantages of many other potential algorithms for machine learning such as the perceptron model, the support vector machine, decision tree and linear regression to see how they compared to the neural network model. My findings conclusively point to a neural network being the superior candidate for modeling IagoBot owing to the fact that it allows for a 64 dimensional input space (corresponding to the number of cells on an Othello board), which is where the perceptron and linear regression models fail, as well as its ability to consist of n-numbers of hidden layers in between the input and output layers so I can adjust the model on the fly as opposed to the support vector machine which is a fixed parametric model.



[iv]

fig.2: The value network. Each node/neuron in this network takes a rudimentary value such as (-1. 0. 1), to indicate the state of that particular cell. This value is combined with an initialized weight value pertaining to that node and then passed into an activation function. Depending on the output of the activation function, nodes in the hidden layer(s) may or may not be activated. The output would be a single double value.

The two neural networks consist of a value network and a policy network. The former is responsible for evaluating the position of a board in any given game for a selected player (in the form of a double) while the latter makes decisions on which path to take in the game tree based on the value network. The value and policy networks must then be combined in a tree-search algorithm. An evaluation of this part of the project will be detailed in the methodology section.

Another constraint which IagoBot must abide by is that it must be a multi-threaded program. Due to the fact that neural networks are much more costly to run than traditional search heuristics, a multi-threaded approach which takes advantage of both CPU and GPU power is a must. The exact number of threads to use will be determined at a later stage of the project.

A further objective which I aim to achieve is an effective approach to evaluating a board position at any given time during game play. This evaluation function must be able to take in the current board position, evaluate the positional advantages of both players and output a single double for a given player to be fed into the value network. Furthermore, the evaluation function must be able to project several layers beyond a particular node in the game tree to analyze the value state of future board positions and return those values to the current layer so as to reflect an accurate prediction of the current state of the board after adjustment.

The final deliverable at the end of the project shall provide a graphical user interface for any human player to play Othello against IagoBot. Depending on how much time is left over at the end, I will also consider discussing with other groups to develop a unified API by which different versions of Othello programs may play continuously against one another and come up with a ranking list determined by the relative strength of each program. In terms of target play-strength, I aim for IagoBot to be able to beat me (a relatively good player) at the majority of games we play.

Project Methodology

This project will comprise two phases: the construction phase, in which I create IagoBot as well as its neural networks using Python, and the training phase, in which I feed the program sets of training data over a period of several days to allow it to optimize its neural network weight parameters.

For the construction phase, I have chosen Python as the language to code IagoBot due to several reasons. Firstly, it is a dynamically typed language which allows for rapid-prototyping development techniques. Secondly, it provides rich and diverse sets of library functions and APIs developed by the open-source community, making it possible for me to plug rudimentary or generic functions into my code, which in turn cuts down my overall development time. Such libraries include but are not limited to: numpy for easy matrix manipulation, matplotlib to quickly visualize the output of neural networks, pandas for data analysis and scikit-learn, which is a very popular machine learning library. Moreover, Python is widely popular as a machine learning language and there exists an active online community specializing in Python machine learning which I could turn to should I require any help during the process.

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=lr, test_idx=test_idx,
                      xlabel = 'petal length [standardized]',
                      ylabel='petal width [standardized]')
```

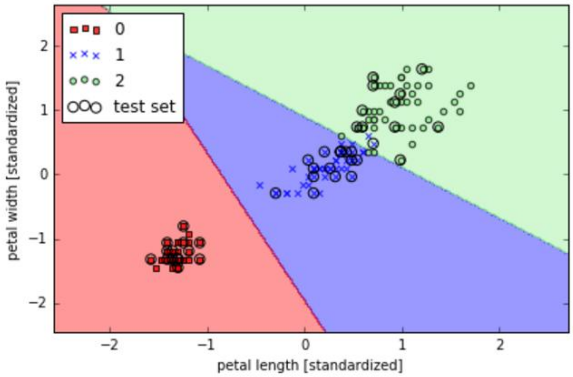


fig.3: An illustration of the various library functions available to Python. Here, a logistic regression model is imported and trained and new test data classified in just a few lines of code using scikit-learn and matplotlib.

IagoBot will utilize a form of tree-search algorithm to help it decide which move to make at each turn. Ideally, the tree-search algorithm would be able to combine both the value and policy networks in a single heuristic that selects what move to make via a lookahead search. I have yet to decide which algorithm will be most suitable for Othello, but it will likely be a variation of Monte-Carlo tree search which is used by AlphaGo.

In the second phase of this project, I will begin to train IagoBot by means of supervised learning. To be sure, there are other methods of training, for instance reinforcement learning which is popularly used to train AI programs to play classic Atari games such as Mario or Breakout. However, I have decided that a reinforcement learning approach is not necessary for IagoBot because it is overly complex and totally unnecessary for a relatively simple game (simple meaning computationally inexpensive) like Othello. A supervised learning method would stipulate feeding the program sets of training data which consist of sample games of Othello as well as an analysis (target outcomes) of every player's strength/likelihood to win within each of those games. IagoBot would take these values as the "golden standard", compare them with its actual output at the end of each iteration, and adjust its weight matrix accordingly. The policy network would be trained in a similar manner.

During this phase, I would be responsible for feeding IagoBot a series of sample games as training data. These sample games will be stored as plain text documents, and under a unified format. This is so that every group currently taking on this project can produce around 50 games worth of data which are then to be shared with every other group. I plan to construct all my training data by playing against an Othello program currently installed on my phone and recording each step.

Risks, Challenges & Mitigation

Two potential issues must be addressed during the training phase of this project, namely: oscillation of the loss function due to an incorrect learning rate and overfitting during regression.

The loss function L is the difference between the target value and the actual output from the neural networks at the end of each iteration of training. Since L is expressed in terms of the weight vectors, minimizing L would allow the neural network to know how much to adjust its weight values. The speed at which the loss function is being reduced is controlled by the learning rate μ. A problem arises when an improper μ is selected as illustrated below:

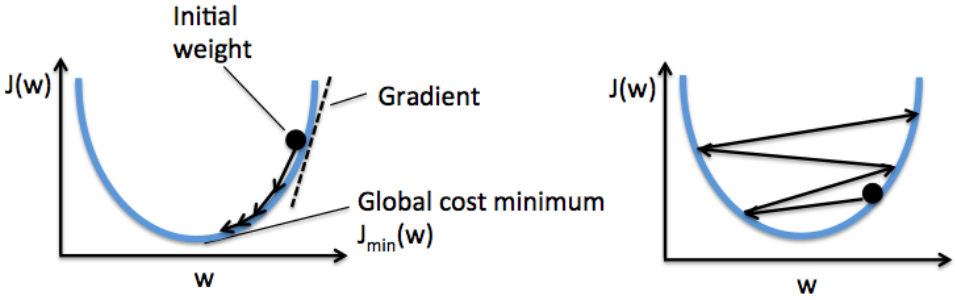


fig.4: Objective/loss function J(w) against weight w for different learning rate values.

If μ is too large as in the graph on the right, then there is a chance that the weight values would overshoot the global minimum. Since the neural network has no knowledge of the actual location of the optimal w value, and instead only knows its relative direction, there would be a tendency for the direction to oscillate with ever increasing magnitudes. On the other hand if μ is too small, then it would take a long time for weight to optimize, leading to wasted time and computational resources. This is a potential pitfall that must be avoided during the training stage.

Regression is a category of machine learning that will be used to model the two neural networks used in this project. It simplifies down to a problem of curve-fitting. Each point in figure 4 may be seen as a data point from the set of training data in the 64^{th} dimensional space. Here, only 2 dimensions are shown for ease of illustration where x_1 and x_2 correspond to two cells on the Othello board. The aim of IagoBot would be to adjust its weight matrix so that a curve is produced that models the trend of the data as accurately as possible without overfitting (shown in figure 4). The problem with overfitting is that sample data often contains noise, and an overfitted curve rarely reflects what the actual model would look like. Furthermore, an overfitted curve would introduce unnecessary complexity and slow down the program. On the other hand we would also like to avoid underfitting by producing an overly simplistic curve.

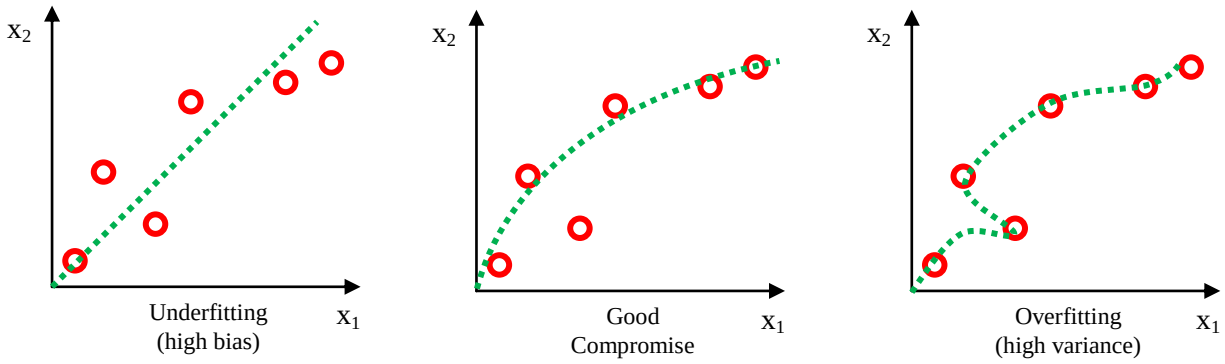


fig.5: An illustration of the curves produced in a regression problem by selecting different regularization values

The solution to this problem is to attach a penalty function or regularization function P(θ) to the loss function which we are trying to minimize. P(θ) would serve to favor a simple curve over a complex one by evaluating to a higher value for higher order curves, where θ represents the weight value parameters.

Tasks and Project Schedule & Milestones

Deadline	Deliverables
September 30th, 2016	Familiarize with Monte-Carlo tree search algorithms. Familiarize with linear regression and how it can be used to minimize the loss function for a neural network using supervised learning.
October 8th, 2016	Record 50 sets of sample games to be shared with other groups.
October 31st, 2016	Design a viable valuation scheme that correctly assigns a numerical value to any player during any given board position. Design and begin coding deep neural network based on value scheme.
November 30th, 2016	Familiarize with minimax algorithms with alpha and beta pruning. Finish Deep Learning Tutorial (by LISA lab, University of Montreal)
December 31st, 2016	Design and implement policy algorithm based on minimax algorithms.
January 31st, 2017	Interim report Begin Testing on first version of IagoBot.
February 28th, 2017	Begin training stage, run continuous training data to optimize neural network loss function. Adjust codebase according to findings.
March 31st, 2017	Begin final round of human testing, collaborate with other groups to produce standardized API for inter-group tournament if time allows.
April 21st, 2017	Deliver finalized implementation. Submit final report for review.

Works Cited

[i] Go Game Guru [Internet]. [updated 2016 May 20; cited 2016 Sep 19]. Available from: https://gogameguru.com/tag/deepmind-alphago-lee-sedol/

[ii] DeepMind [Internet]. USA: DeepMind. [nd; cited 2016 Sep 19]. Available from: https://deepmind.com/research/alphago/

[iii] Buro M. Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello. New Jersey (USA): NEC Research Institute; nd. Available at: https://skatgame.net/mburo/ps/improve.pdf

[iv] Neural Network Models [Internet]. [nd; cited 2016 Sep 19]. Available from: https://www.otexts.org/fpp/9/3

(fig.1 & fig.4) Raschka S. Python Machine Learning. 1st ed. UK: Packt Publishing; 2015