Improving CNN Draughts Evaluators using Genetic Algorithms

Student Name: Thien P. Nguyen
Supervisor Name: Stefan Dantchev

Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the School of Engineering and Computing Sciences, Durham University
January 6, 2018

Abstract —

Background

Presently, competitive Draughts AI players are currently designed to play at a fixed ability. While it has produced very competitive and intelligent players, they require manual modifications in order to improve its performance. This is due to their dependency on pre-defined move databases, where optimal moves are pre-calculated, and recalled when necessary. By combining Neural Networks and Genetic Algorithms, this issue could possibly be solved by creating a player that can grow in ability over time, without the dependency on move-banks.

Aims

The purpose of this project is to explore approaches to tackle the game of English Draughts via the use of machine learning techniques. First, we study previous historical successes in the field, and look at the components that helped build their systems. Then, we look at contemporary methods of computer science that could be used to evolve the historical systems. The project will establish whether this approach provides an effective performance on the game.

Method

The initial population will consist of randomly generated AI players, which will play each other to determine the best player out of the population. The performance of championing AI players at every generation of the genetic algorithm are measured against previous champions. Appropiate algorithms are implemented to detect the overall development of the system's ability to play Checkers.

Proposed Solution.

The proposed solution starts with designing a neural network that evaluates the probability of a particular side winning, given a given state of a checkerboard. This is then used in a algorithm that evaluates future moves to predict the best move at a given position. This, alongside a set of weights for the neural network, creates a player that can evaluate potential moves. Finally, the player is then used on an existing Draughts framework that will provide the player with the ability to play Draughts.

Keywords — AI, Neural Networks, Genetic Algorithms, MiniMax, Alpha Beta Pruning, Draughts

I INTRODUCTION

The intention of this project is to explore the effectiveness of genetic algorithms to improve the evaluation of a neural network's probability to determine the performance of two players in a game of checkers. We attempt to use various crossover and mutation strategies to manipulate the weights of the network, and compare their performance relative to the overall performance of the system.

Draughts

English Draughts (or Checkers) is a popular 2-player boardgame played on an 8x8 chessboard. Players begin with 12 pieces each, and they are placed on light-coloured squares. Each player takes a turn to move a piece diagonally in one square. They also have the option to capture their opponments piece by moving two consecutive diagonal squares, where the opponments piece is placed immediately opposite the players piece. Pieces can be captured consecutively in a single turn if the moves afford the scenario. In the event that a piece reaches the opposite side of the board from where the piece started with, they are promoted to a 'King' piece. King pieces have the ability to traverse backwards in the same diagonal motion as pawns.

Genetic Algorithms

Genetic algorithms (GAs) are a group of search techniques used to find exact or approximate solutions to optimisation and search problems. It borrows techniques from Charles Darwin's evolutionism theory; individuals are created by the crossover of the genetic information of their parents. Genetic algorithms are a subset of evolutionary algorithms, where the larger group is also formed of similar strategies not including Evolutionary Programming [Fogel, 1993][McDonnel, 1993], and genetic programming [Koza, 1991].

Neural Networks

Neural Networks are non-linear statistical data-modelling tools, linking inputs and outputs adaptively in a learning process similar to how the human brain operates. Networks consist of units, described as neurons, joined by a set of rules and weights. The units are defined with characterisitics, and appear in layers. The first layer is defined as the input layer, and the last layer being the output. Layers between the two aforementioned are described as hidden layers. Data is analysed by processing them through the layers.

Learning takes place in the form of the manipulation of the weights connecting the units in the layers. This allows it to model complex relationships between inputs and output, and it can also find patterns in the data.

Motivation

Whilst the use of evolutionary algorithms and neural networks have been explored to create draughts players, my intention is to explore a subset of evolutionary algorithms to determine their viability. Can we produce a similarly performant draughts evaluator by using seperate classifiers for the different stages of the game? Can the use of genetic algorithms and neural networks (GANNs) make a competent Draughts player?

Deliverables

Minimum

- Implement a CNN
- Implement a Checkers Game Interface

- Implement a genetic algorithm with an evaluation function that consists of a round robin tournament against the population of CNN Evaluators.
- Implement a mini-max algorithm that chooses moves.

Intermediate

- A user-friendly interface to play against the AI
- A monte-carlo search of the move space.
- Analysis of Crossover methods (within Genetic Algorithms)
- Analysis of Mutation methods (within Genetic Algorithms)

Advanced

- Convolutional Neural Network Layer analysis
- The resulting AI can play to an ELO of at least 1200.

Related Work

Arthur Samuel in the 60's proposed the concept of using AI to evolve checkers, but did not consider the use of neural networks (as it was not conceived at the time). Also, it was dependent on a set of heuristics that he devised.

Blondie24, proposed by Chelipilia and Fogel, built on Samuel's work by using a neural network as the evaluator function, and used an evolutionary algorithm to evolve the agents. However, new agents are made strictly through the use of mutation. Blondie24's neural network structure consists of a 32,40,10,1 set. Spatial awareness is not considered in their paper.

II DESIGN

Requirements

Functional Requirements

F1 A checkers gameboard is created.

F2 Agents are able to harness neural networks to assist in their move decision.

F3 Offspring agents can be created using parents.

F4 The weights and biases of the Agent's neural network are saved to storage.

F5 Humans are able to play against the Agents.

Non-Functional Requirements

N1 Agents only choose valid, legal moves.

N2 Agents always return a valid, legal move.

N3 Agents are able to play against other agents.

Choice of Programming Language

There are several contenders, with each having their inherent benefits. C++ is a notable choice due to it's relatively lower level architecture, support for popular machine learning packages (most notably Google's TensorFlow.) In terns of performance, C++ trumps most languages. C++ has notable parallelised packages, (XXX) which can assist in the overall performance of the system.

However, it would be difficult to write, due to my unfimiliarity with the language. Also, programs written in this language are less portable. It is not suitable for running on university machines without the use of a sandbox. Improperly handled bugs can cause a fatal error on university machines.

Java is also considered due to my familiarity with the language, but it is similarly less portable. Java tends to be less strict

Javascript is a contender; Node.JS is a very powerful and popular package manager, npm. It is difficult to write multi-processed programs as Node.JS runs on a single thread by nature.

I have chosen Python 3.6 due to my familiarity, and the support of popular scientific packages including NumPy and other machine learning tools. Python is also portable with a very wide compatability; for instance it is pre-installed on all popular UNIX machines and also has support from the university machines.

Object Oriented approaches are taken for the majority of the components of the system, ranging from the neural network library to the tournament system. Data structures are implemented using their own classes and methods where applicable.

Players weights (for their neural networks) are stored in two forms, one of which is to be stored on an MongoDB NoSQL instance, and another local copy in JSON. This allows the individual agents to be played against humans.

Tools

The resulting program will be simulated on Durham's MIRA (128-core Intel) distributed system for 4-ply heavy loads, and debugging will occur on a lighter machine (4-core Intel i5 6200u). In order to keep simulations running on MIRA, MOSH is used to maintain a consistent UDP SSH connection to MIRA. The end champion is then hosted online on a Heroku instance as an API.

Architecture

Should include a diagram of the algorithm workflow

Algorithms and Data Structures

.1 Neural Network

In order to evaluate the board, we use a feed-forward neural network in the form of the following node layers 91,40,10,1 where the input layer consists of 32 nodes, with the output node having 1. Our sigmoidal function of choice is the hyperbolic tangent. The input layer is the board, preprocessed to cover all possible subsquares of the checkerboard, ranging from a 3x3 kernel to a 8x8. And here we see figure 1.

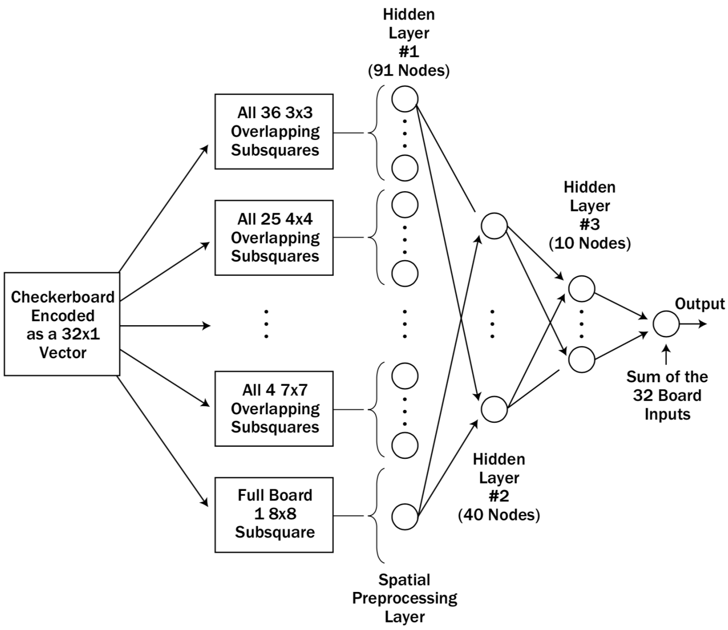

Figure 1: The chosen neural network model. Note that the checkboard is preprocessed.

.2 MiniMax Decision Making

The minimax algorithm revolves around the use of miniMax algorithm to determine the best move to make from a given position. The use of alpha-beta pruning will help to prune unnecessary calls.

The minimax method will have a search depth of 4ply (where the agent will search two moves ahead.) This will allow the agents to have a basic stategy where they can plan their moves in advance. Theere are inherent tradeoffs with having a higher ply count; where the asymptopic complexity is expoential; it being $O(x^y)$ with x being the branching factor, and y being the depth. The branching factor will consist of the moves from each agent in a given game.

Once the initial system is ready, we can migrate to a hybrid Minimax and Monte-Carlo Tree Search proposed by Lorentz, which has shown to dramatically improve performance; especially in the case for games with especially large branching factors.

.3 Tournament Method

The tournament algorithm follows the following pseudocode in Algorithm 1.

Algorithm 1 My algorithm

```
procedure MyProcedure
    stringlen ← length of string
    i ← patlen
top:
    if i > stringlen then return false
    j ← patlen
loop:
    if string(i) = path(j) then
        j ← j − 1.
        i ← i − 1.
        goto loop.
        close;
    i ← i + max(delta_1(string(i)), delta_2(j)).
    goto top.
```

(Budgen 2003)

.4 Genetic Algorithms

The genetic algorithm is the heart of the learning strategy of the system. Here we discuss the various algorithms that form the collection of GA stategies.

.5 Population Generation

The initial population will consist of randomly generated weights and biases of the neural network, with values from -1,1 inclusive. For a population size of 15, the next generation is created using this strategy:

The top five agents from the generation (at the end of the tournament round) are selected for crossover. They will continue to play in the next generation. The next 8 players are generated in the following strategy:

The weights of the 1st and 2nd place agents are used as input to the crossover strategy and will generate 4 offsprings. Two are reciprocal crossover representations from the crossover, and the other two being directly mutated from the parents themselves. Another four children will be created using the same strategy, with the 2nd and 3rd agent's weights.

The remaining two will be direct mutations of the 4th and 5th place agents.

.6 Coefficent Mutation

Each weight of the neural network will be incremented by a random value that is created using the following formula, where $WeightP$ is the current weight, and K represents the number of weights and biases in the neural network:

$$WeightN = WeightP + \frac{1}{\sqrt{2 * \sqrt{K}}}$$

The weights, as explained earlier will have a hard cap of [-1, 1]. This would consequently mean that the mutation is not controlled, and dependent on the number of weights in the system; The more weights in the network implies a less significant mutation.

.7 Crossover Strategy

Two offsprings are created per a pair of parents, with each offspring being the reciprocal crossover of each other. The weights of both parents (now each treated as a 1D array of coefficents), are divided contingent on the number of weights and biases for a given layer. Each layer should be treated separately to reduce the potential dependency on a purely randomly generated neural network. For each set of weights in a given layer, the following algorithm represents the crossover process:

Algorithm 2 Crossover Strategy

procedure CROSSOVER
    stringlen $\leftarrow$ length of *string*
    $i \leftarrow \textit{patlen}$
top:
    if $i > \textit{stringlen}$ **then return** false
    $j \leftarrow \textit{patlen}$
loop:
    if $\textit{string}(i) = \textit{path}(j)$ **then**
        $j \leftarrow j - 1.$
        $i \leftarrow i - 1.$
        goto *loop*.
        close;
    $i \leftarrow i + \max(\textit{delta}_1(\textit{string}(i)), \textit{delta}_2(j)).$
    goto *top*.

Testing and Evaluation

The combined GANN (Genetic Algorithm/Neural Networks) are evaluated under competition style conditions. At each generation, each agent plays 5 games, with their opponment being from randomly chosen from the generation pool. Point scores are measured by 2,0,-1 where 2 is a win, 0 is a draw, and -1 is a loss. There is a hard cap of 50 moves, where the game is considered a draw if after the game hasn't ended after 50 moves from each player.

At the end of a given generation, we measure growth of performance using the champion of the generation. Presently we will use the mean of means approach. When a new chapmion is generated, it is played against the previous 5 champions from earlier generations. 6 games are played for each previous champion, with 3 being as Black, and 3 being White. A mean score is calculated from those 6 games. The overall performance of the current chapmion is the mean of the 5 sets of games. A positive improvement is when the mean of means are greater than 0.

Point Score for the champion games are measured by 1,0,-1 where a Win counts as 1 point and -1 for a loss. The weights are scaled differently to the regular tournament in order to accurately portray the difference between previous champions.

At the end of the generation run, the end player will be used to compete against human players on various online multiplayers checkers websites in order to determine an accurate ELO rating of the system.

A Future Work

We could potentially build a neural network model that

The use of ELO could have a precise reprentation of the agents progress. Arpad Elo's algorithm runs as such:

This could be used to categeorise neural networks, and helps us to understand the potential thinking position of opponments. This allows us to optimise our minimax algorithm in order to exploit the players weakness.

III REFERENCES

References

Budgen, D. (2003), *Software Design*, 2nd edn, Addison Wesley.