Training a Neural Network for Checkers

Daniel Boonzaaier
Supervisor: Adiel Ismail

June 2017

Thesis presented in fulfilment
of the requirements for the degree of
Bachelor of Science in Honours
at the University of the Western Cape

1 Declaration

I, Daniel Boonzaaier, declare that this thesis Training a Neural Network for Checkers is my own work, that it has not been submitted before for any degree or assessment at any other university, and that all the sources I have used or quoted have been indicated and acknowledged by means of complete references.

Signature: . . . . Date: . . . .
.
Printed Name: Daniel Boonzaaie

2 Abstract

This project will attempt to create a learning program that teaches itself how to play checkers using a neural network. Learning will be done by having the program play checkers numerous times and then evolve based on the outcomes of each game played.

Contents

3 Introduction

Machine learning is not a new concept in computer science. Arthur L. Samuels Some Studies in Machine Learning Using the Game of Checkers was originally published in July 1959.

Machine learning is about computer algorithms that allow a computer to learn and is a related to a branch of statistics called computational learning theory. Neural Networks are but one type of Machine Learning methods that use input nodes that are connected to a hidden layer via different weights which are in turn connected to an output layer via more weights.

Checkers is a good game to train as it provides complete information. Which means that the complete status of any game is known to the players at all times throughout the game.

4 Proposal

This project will attempt to create a learning program for the game of checkers using a neural network to autonomously learn to play checkers from scratch. In other words, the program will teach itself how to play checkers.

The main focus of this project will be training of a neural network by having the program play the game of checkers a multitude of times. By having the program play against itself the goal is that this program will learn from a predetermined set of rules how to play. Consideration will have to be made in order to make sure that the neural net for this program is not over trained.

5 Project Plan

This project will have four main parts that will take place throughout the year of 2017 with each part of the project taking place in each quarter of the year.

The first part of the project which is the analysis of the project is the research into what the project will require and the analysis of said requirements from the stand point of the user and software. Researching past works related to the project as well as technologies and software related to the project will assist in guiding the projects development.

The second part of the project is the projects design and development. This entails the creation of a User Interface Specification and prototype. From this an Object Orientated Analysis and then an Object Orientated Design can be done. This will take a closer look at the setup of the neural network and other related software.

The third part of the project is the projects implementation where the design previously done will be used to create the projects learning program. The implementation will need full documentation.

The final part of the project will be the projects testing, evaluation and presentation. Here the created program will be tested to determine whether it works according to expectations and refined if needed.

6 User Requirements

The program shouldnt require much from the user. The user simply needs to determine whether or not the program has developed in its playing abilities.

The program will need to play checkers in some way and learn the best moves to make in order to win.

How the neural net will work, its layout and application will need to be well thought out. The interface be it graphical or otherwise will need to be thought.

The program simply needs to show that it has learned how to play checkers without outside help from a user. Playing against the program should be possible and could be done further but is not the goal.

7 Requirements Analysis

There are previous works done on autonomous game playing systems that involve various games, checkers included. One such work, which was mentioned in the introduction, is Arthur L. Samuels implementation.

Another implementation was done by Nelis Franken and Andries P. Engelbrecht. Particle Swarm Optimisation was used in the implementation of their game playing program.

Thus one implementation of the checkers game learning/playing program may be done using the neural net in conjunction with gradient decent in its back propagation and the second using the particle swarm optimisation technique in its back propagation.

All possible moves need to be analysed and the system should determine which move is the best one to make in order to win. The program will need a look ahead to determine possible moves.

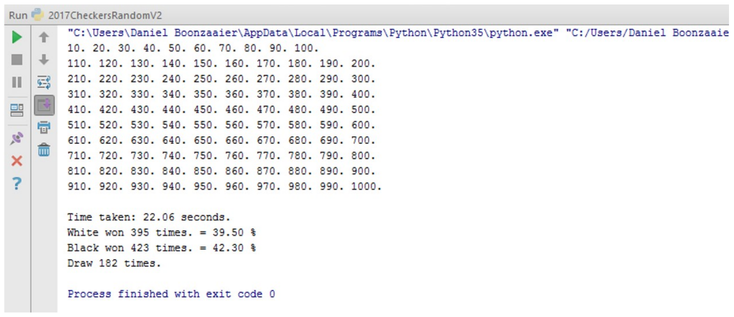
Testing of the program should be as simple as seeing whether or not the program follows the rules set out for the game of checkers and if it has learned to play the game properly and the a decent level of competency.

8 Interface

This program will be interacted with in two different ways. The first is when the training of the neural network is taking place. The second is that of the checkers game.

8.1 Interface for training

There will not be a graphical user interface for the training portion. The interaction that takes place during the training will not be accessed by a user and will only have output to show what is happening during the training process. This output will be visible from within an integrated development environment (IDE) such as PyCharm.



```
"C:\Users\Daniel Boonzaaier\AppData\Local\Programs\Python\Python35\python.exe" "C:/Users/Daniel Boonzaaie
10. 20. 30. 40. 50. 60. 70. 80. 90. 100.
110. 120. 130. 140. 150. 160. 170. 180. 190. 200.
210. 220. 230. 240. 250. 260. 270. 280. 290. 300.
310. 320. 330. 340. 350. 360. 370. 380. 390. 400.
410. 420. 430. 440. 450. 460. 470. 480. 490. 500.
510. 520. 530. 540. 550. 560. 570. 580. 590. 600.
610. 620. 630. 640. 650. 660. 670. 680. 690. 700.
710. 720. 730. 740. 750. 760. 770. 780. 790. 800.
810. 820. 830. 840. 850. 860. 870. 880. 890. 900.
910. 920. 930. 940. 950. 960. 970. 980. 990. 1000.

Time taken: 22.06 seconds.
White won 395 times. = 39.50 %
Black won 423 times. = 42.30 %
Draw 182 times.

Process finished with exit code 0
```

Figure 1: Example of output for checkers games played.

8.2 Interface for the game

The game of checkers that a user can play, will have a graphical user interface. It will have a simple start button and quit button on the first screen.

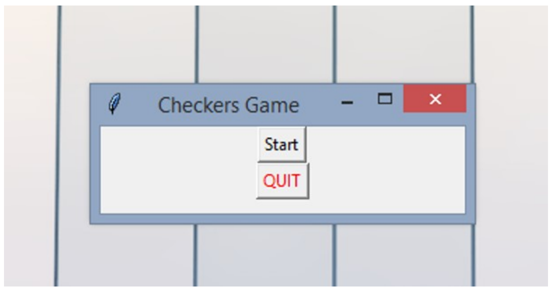


Figure 2: Simple Checkers Game GUI.

Once the start button is pressed, a game of checkers will begin where the user will play against the trained neural network. The quit button will end the program.

The game board will be shown and the user can click on a piece to move and then on the tile the player wishes to move said piece to. If the move is valid then the piece will move otherwise an error message will be shown telling the user that it is an invalid move.

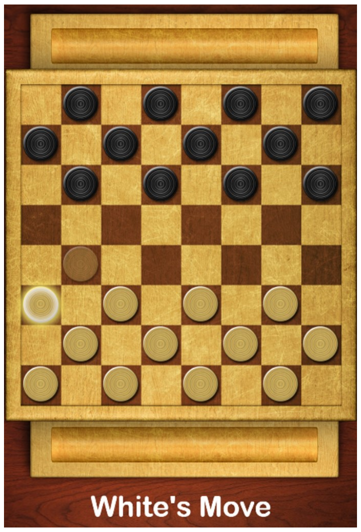


Figure 3: Example of Checkers Game Board GUI.

When it is the programs turn to play, the board will be evaluated by the neural network and its selected move will be made. The player may exit the game at any time.

9 High Level Design

This high level design will attempt to give an explanation of the components that will make up this project. A brief explanation of each component will be given and how these components will interact.

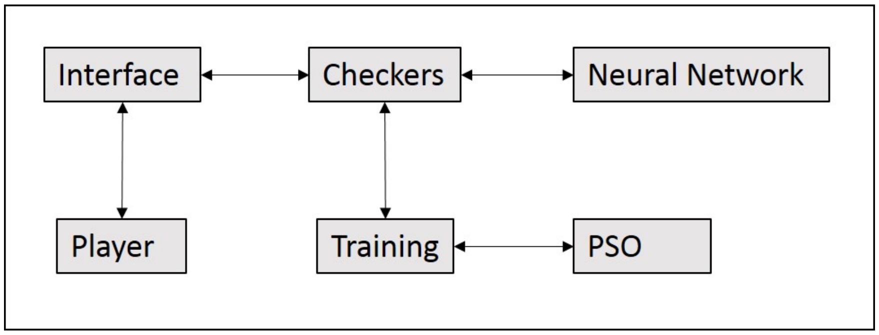


Figure 4: Interaction of Components.

9.1 Interface

The interface will be the playable game that a user will interact with. This Interface will interact with the checkers game which controls the game being played and the rules for checkers.

9.2 Player

The Player here refers to any user which will play the game of checkers. The player interacts with the interface which in turn interacts with the checkers game. The player will indicate to the interface what the player wants to do and the interface will interact with Checkers to determine whether whatever the player wants to do is valid for the game.

9.3 Checkers

Checkers refers to the rules and computations behind the interface for the game to work. It will be responsible for the rules of the game and for the neural network that plays against a Player. Checkers will interact with the Neural Network when any game is being played, during training as well as during Training. Checkers also interacts with Training.

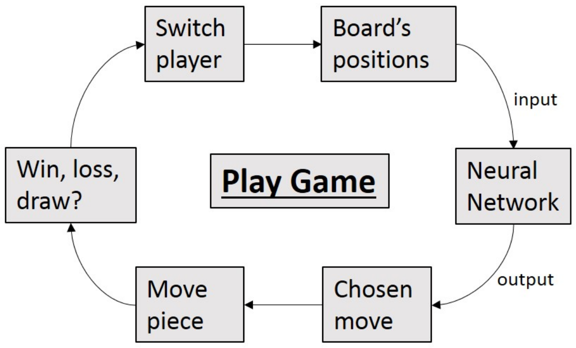


Figure 5: Basic Depiction of program playing checkers for Training.

9.4 Training

Training will be done before any user can play a game of checkers. Training involves playing a game of Checkers where the neural network for each agent in the training phase determines which moves should be done. Once a number of games have been played this way a score, or victory score, is calculated from the number of wins, losses and draws. Plus one for a win, minus two for a loss and zero for a draw. Training interacts with the PSO to update the weights of all the agents Neural Networks after each agents score is calculated.

9.5 Neural Network

The Neural Network will work for any board by taking in a thirty two vector input of all board positions. The Neural Network will then output a score based on the positions of the board. This score is obtained for each possible move and the move with the greatest score should be chosen as the best move for that Neural Network.

9.6 PSO (Particle Swarm Optimization)

The Particle Swarm Optimization will work on all the vectors made up of all the weights for each agents Neural Network. The victory scores of the agents will be used to determine the global best. The weights in the vectors will be updated according to a velocity function that takes into account the global best and each agents personal best.

10 Low Level Design

The low level design will provide more specific details on the components discussed in the high level design. It will attempt to provide a clear definition of the algorithms used in the creation of the program.

10.1 The Checkers Game

The core of this project is the neural network which will be trained to play checkers. However, to do this, one must first have a checkers game to play. The algorithm with which a game will be played is as such:

10.1.1 Play Checkers Game Algorithm for random moves

1. Run through all current players checkers pieces.
2. If Piece is a normal piece.
   - (a) Check positions left forward diagonal and right forward diagonal.
   - (b) If position is open then add to list of possible moves.
   - (c) If position contains opponents piece.
     - i. Check positions diagonally behind opponent piece position.
     - ii. If position contains any piece ignore.
     - iii. If position is open then check for further jumps until all jumps have been exhausted, steps 2(a) and 2(c), then add to list of possible jump moves.
3. If Piece is a King.
   - (a) Check positions left forward diagonal, right forward diagonal, left back diagonal and right back diagonal.
   - (b) If position is open then add to list of possible moves.
   - (c) If position contains opponents piece.
     - i. Check positions diagonally behind/in front of opponent piece position.
     - ii. If position contains any piece ignore.
     - iii. If position is open then check for further jumps until all jumps have been exhausted, steps 3(a) and 3(c), then add to list of possible jump moves.
4. If there are possibilities to jump and take opponents piece one of these must be chosen at random.
5. Else choose random move from all possible non jump moves.

6. If all opponents pieces have been removed from the board. Current player wins. Stop if game has been completed.
7. Change player. If Player 1 change to Player 2. If Player 2 change to Player 1.
8. Continue above steps until game reaches a conclusion or after 100 moves resulting in a draw.

10.1.2 Finding Checkers Moves

A list of all possible moves will be created for the 32 possible positions on the board where pieces can occur. Then a list of the possible moves for a certain board can be extracted from this list by comparing the positions that are open and which contain pieces to determine which moves are valid. The algorithm to create said list of all possible moves is created as such:

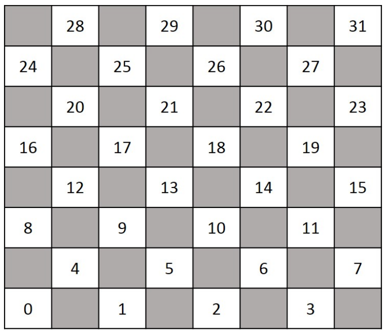


Figure 6: Checkers Board with positions 0-31.

List of possible moves:

- P = `[0, 1, 2, ..., 31]`
- moves = `[]`
- for all n in p
  - if `(n % 8) >= 0` and `(n % 8) < 4` then
    - right move = n+4
    - if `n % 8 = 0` then left move = no move
    - else left move = n+3
  - if `(n \% 8) >= 4` and `(n \% 8) < 8` then
    - left move = n+4
    - if `n % 8 = 7` then right move = no move
    - else right move = n+5
- append `[n, [left move, right move]]` to list moves

10.2 Neural Network

The neural network that will be used as the brain of the checkers playing program will first be trained. This training will be done to ensure that the neural network used will be able to play the game of checkers and play the game to a decent competency. At minimum the neural network should play better than a game played against that of randomly chosen moves.

In the training of the neural network, the neural network will be used to evaluate board positions and then output a score of the board. These scores will be attributed to each of the available moves and the move with the highest score should be chosen as the best move that should then be carried out. This step will continue throughout the entirety of each played game. This part of the training is referred to as feed forward.

10.2.1 Neural Networks use within a game of checkers

The move that should be played will be obtained by using the neural network to evaluate all the possible valid moves as such:

- maxScore = Set very large negative number
- movesList = Obtain all possible valid moves
- index = will contain the index of selected move
- For each move in MovesList do:
  - boardCopy = create a copy of the games board. The game board consists of a vector of size 32. Each element in the vector represents a place on the board, as can be seen in figure 6.
  - Perform the current move and update the vector boardCopy
  - value = Use boardCopy as input for the neural Network and obtain output. The output being a scalar value.
  - If value is greater than maxScore then
    - Make maxScore equal to value
    - Set index to equal the index of the current move
- Return index as the move that should be played

10.2.2 Details on the Neural Network

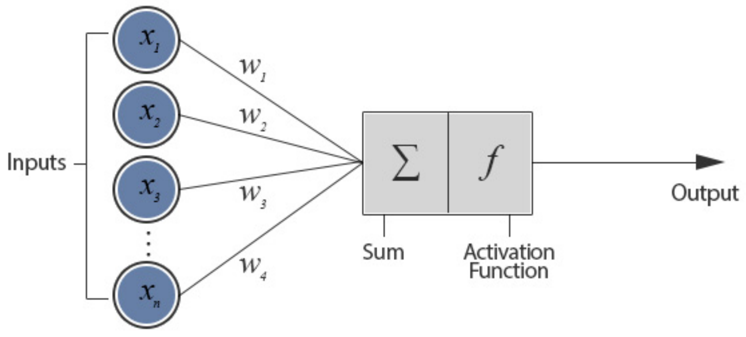


Figure 7: Neural Network Example.

The vector of 32 board positions work as the input for our Neural Network. Then at each node in the hidden layer its activation value is calculated via a summation function and sigmoidal evaluation function.

The summation function works, given figure 7 above, as such:

```
Sum = X1W1 + X2W2 + X3W3 +  + XnWn,
```

where X stands for the values of the inputs and W stands for the values of the weights.

The sigmoidal evaluation function is as such:

```
f (n) = 1 / 1 + e-n,
```

where n stands for the value of the node.

10.3 Particle Swarm Optimization (PSO)

In order to improve the neural networks, that will be played against each other during training, some back propagation needs to be done in order to update the weights of the neural networks. Updating the weights iteration after iteration will lead to the neural networks evolving and eventually produce one or more networks that will be used in the final checkers game.

A PSO algorithm will be used to update the weights of the neural networks. The PSO will work on all the weights of each neural network in a population of a certain size. The weights will make up a vector with each vector corresponding to a particle in the algorithm. Therefore if the swarm is of size n, there will be n particles representing the weights of each neural network, plus a copy of each particle which represent its personal best.

Each particle is updated according to two best values. The first is the best value that the particle has achieved so far, which is the personal best or pbest. The second is the best value is that of the best out of all the particles in the swarm, which is the global best or gbest.

10.3.1 PSO Algorithm

- Initialise particles from weights of neural networks
- For each particle
  - Calculate the fitness value
  - If the fitness value is better than the best fitness value, pbest, then set the current value as the new pbest.
- Choose the particle with the best pbest and set it as the gbest
- For each particle
  - Calculate the particle velocity
  - Update the particles position
- Repeat above steps for certain number of iterations or until minimum error criteria is met.

10.3.2 Particle velocity function

```
V = V + ( C1 * rand(0,1) * ( pbest - present ))

      + ( C2 * rand(0,1) * ( gbest - present )),
```

where c1 and c2 are learning factors, rand(0,1) is a random value between zero and one and present is the particle as it currently is.

10.3.3 Function for particles position

Present = Present + particle velocity.

10.3.4 Fitness value

The fitness value is calculated for each particle, which is a vector of each neural networks weights. The fitness value is calculated after each neural network has played a number of games against other randomly selected neural networks. After each game the fitness value is updated by +1 for a win, -2 for a loss and 0 for a draw.

References

[1] T. O Ayodele *Types of Machine Learning Algorithms.* University of Portsmouth, United Kingdom (2010)

[2] A. L. Samuel *Some Studies in Machine Learning Using the Game of Checkers.* IBM Journal, Vol 3, No. 3, (July 1959)

[3] Cranfill, R., Chang, C. *What is Facebook's architecture?.* Quora.com/What-is-Facebooks-architecture-6 (12/2014)

[4] K. Chellapilla, D. B. Fogel *Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge.* IEEE Transactions on Neural Networks, Vol. 10, No 6, (Nov 1999)

[5] N. Franken, A. P. Engelbrecht *Evolving intelligent game-playing agents.* Proceedings of SAICSIT, Pages 102-110, (2003)

[6] N. Franken, A. P. Engelbrecht *Comparing PSO structures to learn the game of checkers from zero knowledge.* The 2003 Congress on Evolutionary Computation. (2003)

[7] A. Singh, K. Deep *Use of Evolutionary Algorithms to Play the Game of Checkers: Historical Developments, Challenges and Future Prospects.* Proceedings of the Third International Conference on Soft Computing for Problem Solving, Advances in Intelligent Systems and Computing 259, (2014)