Evaluating The Effectiveness of Neural Networks Against Basic Decision-Making Algorithms Through the Creation of A Chess Engine

Abstract

In order to understand the benefits and limitations of Neural Networks[1], I have programmed two chess engines and will be comparing the effectiveness of each. One engine has been programmed using the Minimax[2] algorithm along with Alpha-Beta Pruning[3], while the other engine has been created using a two layer Feedforward Neural Network as part of its static evaluation function.

My results have found that neural networks are certainly capable of advanced logical thinking and have a few benefits over more traditional game playing techniques. However, they are severely limited by the hardware upon which they are run which has led me to conclude that deep-learning networks may be limited to specialised applications for the foreseeable future.

1. Introduction

From image recognition to scientific analysis, neural networks have been shown to be successful in a wide variety of applications. Recent research within the field has also demonstrated how powerful they can be. For example, DeepMind, a research division within Google, has had massive success with deep-learning neural networks in the context of chess engines; using deep-learning, they were able to create a program which taught itself how to play at a level of mastery within 24 hours (DeepMind, 2017). This program beat the world's most powerful chess engine which relied on more conventional game-playing algorithms. Such promising results beg the question of whether we have now reached a point where neural networks can effectively simulate complex human behaviour.

In order to answer this question, I will be creating two chess engines: one which relies on conventional Artificial Intelligence algorithms and another which uses a neural network. I have chosen to create these programs in the context of a chess engine for a number of reasons. Firstly, computational chess engines have been researched since the 1960s (PCWorld, 2013) which means there is an abundance of resources I can use to guide my project. Chess is also a non-trivial game which means that both engines will require a certain level of competence in order to play effectively.

I will be comparing the effectiveness of these different algorithms through how the programs compare on four different factors. These include win percentage, move time, ease of extension, and ease of implementation.

I chose this project for two reasons. Firstly, computer science is a subject I am very passionate about and I plan to study it in university. Therefore, I feel it is important for me to understand the theory behind AI as it is an extremely promising field within the subject. Secondly, in the future I hope to go into academia and become a researcher. This is why I wanted to engage with the academic process and conduct independent research. These goals are also the reasons why I chose my two final aims

[1] A neural network is a piece of software modelled on the human brain. It can be used to predict values given a set of inputs.

[2] Minimax is a simple algorithm used to play moves in games based on the principle of minimizing your maximum loss. It uses a game tree structure in order to find the best move.

[3] Alpha-Beta Pruning is a way to reduce the number of positions the minimax algorithm has to evaluate. This is done by removing branches of the game tree.

for the project: to learn more about Artificial Intelligence, and to understand the academic process better.

2. Research

As stated earlier, this project was inspired by DeepMind through their research paper titled 'Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm'. Through CRAAP testing this source, I feel it is very reliable. I believe two most useful parts of this source are its provenance as it is an academic article from a reputable source; therefore it will be extremely objective and factual. It is also useful as it is very recent, having only been published half a year ago, so many of the methods they discuss in the paper are still relevant. The main drawback with this article is how it is only tangentially relevant to my project - this is as their implementation of a neural network is different to my implementation.

Another key source I used was a book called 'Algorithms To Live By' by Brian Christian and Tom Griffiths. In terms of this sources Currency, the book was published in 2016 by Holt House, a reputable New York City-based publishing firm, and won numerous awards. Therefore I believe it's Currency is useful. It is also extremely relevant to my topic as it discusses various mathematical algorithms and how they relate to everyday life; I especially found Chapter 7 on Machine Learning useful as this was directly related to my EPQ. Both of the authors are also prize winning journalists and highly respected in their fields so I find there is little fault with the Authority of the book. Additionally, throughout the book they mention numerous other studies done by different researchers so it is obviously well sourced and informative. The only fault I can find with this source is in its purpose as it was a commercial endeavour created primarily to sell books - this means some of the information may be watered down which isn't very useful for me as I want to write an academic paper. Overall though, I feel it is very useful.

In terms of websites I used to guide my research, I relied heavily on a website published on code-spot titled '15 Steps to Implement a Neural Network'. This was a guide written to help beginners create their first neural network and I found it very useful when programming my chess engine. I feel the two most distinct aspects of this source are its purpose and accuracy. In terms of its purpose, I feel it is limited as it is a commercial website with a primary goal of attracting views. This means some of the information may be skewed or less detailed than necessary. However, I feel the source is still very accurate as it discusses the fundamental mathematics behind neural networks and goes into detail. Additionally, Herman Tulleken, the author of the article, is a senior author at code-spot and is a reputable source when it comes to this subject. Therefore I feel this source was useful for my EPQ.

In terms of other academic papers I have read, I found a number of postgraduate thesis' from various universities discussing neural networks in the context of chess. One of these was written by E. Robertsson from Lund University, and another was from B. Oshri from Stanford University. I found these extremely useful as they discussed methods of implementing neural networks without requiring the advanced infrastructure that DeepMind had in their paper. Since these papers are also academic research papers, they are accurate and are a good source of information for my project. These papers ultimately changed how I would implement my network - I have described this process in section 3.

I have also used a range of different sources to guide my project. For example, I found a number of documentaries on neural networks online extremely useful for explaining the basics behind the concept. I also found magazines such as 'PC World' contained a lot of useful articles on this subject. These were very useful for showing me new sources I could read.

3. Evolution of Product

My original plan was to create two chess engines: my first engine was to be created using traditional AI algorithms and my second engine was to be made with a neural network. For my first engine, I was able to implement it without any issues. This was as it was a reasonably simple project and had been doing numerous times by other programmers so I could read their articles and go from their work (Hartikka, 2017).

Since my second engine was far more theoretical, I changed my implementation a few times. Originally, I had planned to use a single neural network to teach itself how to play chess and decide what moves to make - this was based on the work done by the Deep Mind group. However, I quickly realised that this would not be feasible for me as I lacked the hardware and resources the Google team had. This is why I instead changed my neural network to only evaluate the board given a position (similar to the work done by E. Robertsson and B. Oshri in their master's thesis). This meant that I would still use a game tree which Minimax would search through, but the evaluation of each board position would be done by a neural network (E. Robertsson, 2002). This implementation was far more feasible for me to accomplish as it could be run on my laptop.

4. Chess Engine A: Static Evaluation Function

This engine relies on a basic minimax algorithm using a hand-built evaluation function and provides a control for me to test my neural network against. I programmed it from scratch in C++.

The bulk of the work with this implementation was in programming the rules of chess. This was a relatively complex task as chess has a host of special cases (e.g pawns can only move one square except at the start when they can jump two). In order to program this, I had a base class called Piece which stored details about a given piece. All other pieces inherited this class and override properties to reflect the piece values.

Programming the actual AI was a relatively simple task. I had a function built into my Board class which generated a list of legal moves from the current position. This allowed my minimax algorithm to search through it and find the best move. Each move would be passed through a static evaluation function which accounted for factors such as number of pieces on the board in order to generate a numerical value. In order to speed up tree search time, I used Alpha-Beta pruning to cut out branches of the tree which were unnecessary. Building this entire program took roughly 6 months.

5. Chess Engine B: Neural Network Evaluation Function

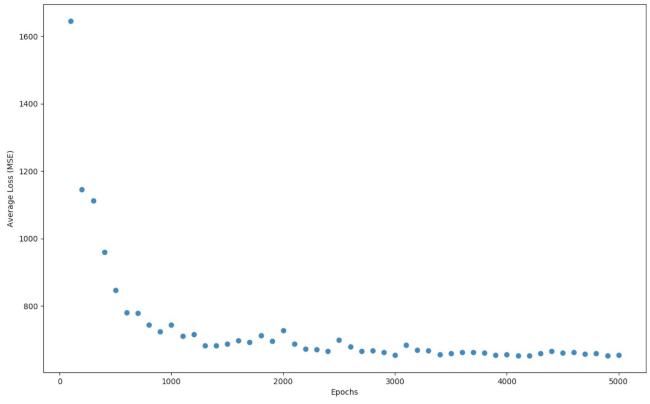
As stated in section 3, this engine will use minimax with Alpha-Beta pruning in order to search the game tree. However, it will incorporate a neural network in the evaluation function.

In order to program this neural network, I implemented a deep neural network in

TensorFlow. This was made up of 65 input neurons, 2 hidden layers with 65 hidden neurons each, and 1 output layer of a single neuron. This configuration allows me to pass in the piece values of each square of the board plus a binary bit indicating whether it's white or black to move, and it outputs a floating point number from ∞ to -∞ indicating how strong the current board position is.

The data which I will use to train my neural network will be generated using a Python script I wrote, in conjunction with a chess library called python-chess. Using the script, I generated 10,000 chess positions by moving random pieces. I then used a chess engine, Stockfish 7, to evaluate the board positions for me. I did this by writing a bash script to automatically iterate through the board positions and write the evaluations to a text file. This allowed me to train the neural network on real, potential chess positions.

Training the neural network took roughly 12 hours on my laptop. By the end of it, my network had converged on an optimum set of values, as is shown by how the graph below plateaus over time.



Loss (Mean Square Error) of my neural network against the number of epochs

6. Problems Encountered During Process

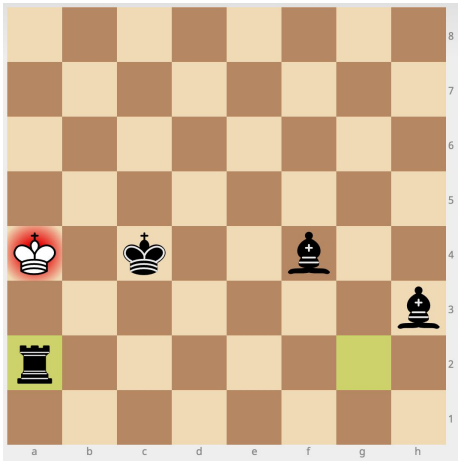
When I was programming my first chess engine, I found this to be a relatively smooth process as I had a solid understanding of the algorithms behind the AI. However, I did encounter a few problems. My biggest problem was to do with programming the rules of chess in the first place. Since my program needed to account for all the nuances of the game, this was an extremely time consuming process and took up most of my time. Furthermore, I was doing this program in C++, a very complex language which I had little experience with; this made my task harder. In order to solve this, I took an online course on C++ to understand the language better. Additionally, I consulted numerous online sources such as the chess-programming wiki to assist me as I programmed my chess board.

Another big problem I faced was with programming my second chess engine. At the start of my project, I had planned to program my neural network in C++ and use the same code for my chess board. However, half-way into my project I realised that the infrastructure for machine learning was not present in C++ as it is a relatively old language. I therefore had to choose whether I would switch to Python and TensorFlow, which had extensive resources for AI, or stick with C++ and program my neural network from scratch. Ultimately, I decided to program my neural network in TensorFlow as the aim of my project was to see how effective neural networks were at making decisions rather than programming neural networks themselves. Therefore, I felt it would be more beneficial for me to implement my neural network quickly and achieve this aim rather than waste time reinventing the wheel.

7. Comparison of Engines

I will be evaluating the two engines I have created based on the following factors: win percentage, move time, ease of extension, and ease of implementation. In order to make this a fair test, I will run 14 games between both engines and have them alternating sides. I will set both engines to search to a depth of 4 plies in order to ensure that the evaluation function is the only component being judged.

Regarding win percentage, I found that Engine A won 4 games, Engine B won 2 games, and 8 games were drawn (either due to a 3-move repetition or a lack of pieces on the board). In order to ensure that games were unique, I introduced a random multiplier into the evaluation function of Engine A to choose unique moves. Even though Engine A won more games than Engine B, the fact that the majority of games were drawn indicates to me that both engines were evenly matched in terms of their evaluation functions. Another observation I made was the high number of moves per game - on average a game was finished after 67.3 moves. This indicates to me that, while the evaluation functions were effective in stopping obvious attacks, they were unable to effectively progress the game.



An example checkmate which black (Engine A) won. This game took 89 moves.

In terms of move time, Engine A was clearly superior to Engine B. For a search depth of 4 plies, Engine A would reliably play a move within 10 seconds regardless of what stage the game was at. Engine B however required a few minutes in order to search through the game tree and find the best position - this was despite the fact that Alpha-Beta Pruning had been implemented. This huge lag came from the fact that it takes a substantial amount of time for TensorFlow to instantiate a model and process the input data; this must be done for every position being evaluated.

For ease of extension, I will be discussing how easy it would be for me to make changes to the code in order to substantially increase the effectiveness of gameplay. With Engine B, this is a reasonably simple process as I can simply generate more chess positions and leave the neural network running in order for it to train. While this would be hardware intensive, it would not require much work on my end which means it is relatively easy to extend. With Engine A, this is a much more complex task as I would need to analyse what specific features of a chess board make a strong position and translate this into code. This can become very complex very quickly which is why I believe Engine B is far easier to extend.

Finally, for ease of implementation, I feel the converse is true. This is as while Engine A is difficult to extend, it is extremely easy to implement as you can simply start by introducing heuristics such as piece values. This can be done using if statements. On the other hand, Engine B requires a substantial amount of code in order to set up the neural network. You would also need to write additional code to generate board positions and train the network. Finally, training the network itself requires a

great deal of intensive computation. Therefore, Engine A is far easier to implement.

8. Conclusion

In conclusion, I believe that my evaluation of these two chess AIs has revealed some of the benefits and drawbacks behind neural networks.

The fact that my neural network chess engine was roughly of equal ability to my conventional chess engine shows that they can be effective in complex applications. Moreover, the fact that Engine B was able to generate interesting positions that a typical chess engine would not see indicated to me that neural networks are certainly capable of some high-level thought. Finally, the fact that neural networks can easily be extended is another major benefit over conventional game-playing algorithms as it means there is no limit to their potential.

However, my study has shown that there are a few drawbacks behind neural networks. The primary issue is the requirement for specialised hardware. Given that I was only able to train my network with a limited dataset due to my lack of dedicated hardware, and the long move time for Engine B, I believe that neural networks will only be limited to specialised applications for the foreseeable future. This is due to the technological requirements of running a neural network. Coupled with the huge amount of initial work necessary to create a neural network, I believe such obstacles will be prohibitive in the usage of this technology.

Concerning the future for deep-learning neural networks, I believe my project has shown that they are extremely promising due to their ability to make complex decisions. However, due to considerations such as hardware and development time, the barrier to entry may be prohibitive. This is why I believe they will only be limited to specialised applications until further development is made in these fields.

9. Evaluation

In terms of my two original aims of wanting to learn more about AI, and wanting to understand the academic process better, I felt I was successful to some degree. For example, I feel I fully achieved the first aim as I understand the nuances of neural networks and machine learning far better than I did a year and a half ago. Furthermore, this project exposed me to new languages such as Tensorflow which I am comfortable with now having created a project from scratch using then. I also feel I partially achieved my second aim as I understand how certain parts of the academic process work such as research and analysing data. Skills like CRAAP testing sources were new to me at the start of this EPQ so having experience with them in an independent project was definitely useful.

I do feel though that I could have done more to achieve this second aim. I feel that I could have spent more time on the analysis section of my EPQ essay as it was less academic than I was hoping it would be. In other scientific papers I read, I noticed they used a lot of quantitative comparisons whereas my comparisons were more qualitative and descriptive. Had I spent more time on the analysis of my two chess engines, I felt I could have tightened up my essay and fully achieve my second aim.

10. Reflection

I thoroughly enjoyed the EPQ process and feel I learnt a lot over the last year and a half. During my project, I firstly feel I massively expanded my knowledge of computer science and Artificial Intelligence. For example, my first chess engine was roughly 10,000 lines of C++ code and my second chess engine was roughly

6000 lines of Python and TensorFlow code, so I feel taking an extended project such as this made me very comfortable with programming. I also learnt about new algorithms in AI and how to build a neural network which was something I had never done prior to my EPQ.

In terms of the skills I have improved, I feel my time management abilities have become better as a result of this project. This is due to the massive amount of work I had to do in this product. Because of this, I needed to be strict with myself on how I would spend my time. I felt one of the reasons my project was successful was because I had broken it down into small chunks over the course of Year 12 and was basically done by the start of summer. I feel this attitude towards my work has improved my time management skills massively.

I also feel my academic research skills have improved as I now have knowledge of resources such as JSTOR and EBSCO host.

In terms of my main challenges, one of the biggest problems I faced was a lack of planning on my part. This is as I dived right into programming without any real thought as to how I would do it. As a result of this, I ended up programming 9 different chess AIs over the course of my EPQ, only 3 of which worked. This process would have been a lot more efficient had I done the research beforehand.

Ultimately, I enjoyed the EPQ process and felt I learnt a lot of important skills such as academic research and referencing. I believe that these skills will serve me well in university and beyond.

11. Bibliography

1. Robertsson, E, 2002. Using Neural Networks in the Static Evaluation Function of a Computer Chess Program. Master's Thesis. Sweden: Lund University.

2. Oshri, B, 2014. Predicting Moves in Chess using Convolutional Neural Networks. Master's Thesis. USA: Stanford University.

3. code-spot. 2009. 15 Steps to Implement a Neural Network. [ONLINE] Available at: http://code-spot.co.za/2009/10/08/15-steps-to-implemented-a-neural-net/. [Accessed 1 July 2018].

4. Int8. 2016. Chess position evaluation with convolutional neural network in Julia. [ONLINE] Available at: https://int8.io/chess-position-evaluation-with-convolutional-neural-networks-in-julia/. [Accessed 1 July 2018].

5. Erik Bernhardsson. 2014. Deep learning for… chess. [ONLINE] Available at: https://erikbern.com/2014/11/29/deep-learning-for-chess.html. [Accessed 1 May 2018].

6. Kyle McDonell. 2018. NeuralChess. [ONLINE] Available at: https://github.com/KyleMcDonell/NeuralChess. [Accessed 1 August 2018].

7. PCWorld. 2013. A brief history of computer chess. [ONLINE] Available at: https://www.pcworld.com/article/2036854/a-brief-history-of-computer-chess.html. [Accessed 1 April 2018].

8. DeepMind. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. [ONLINE] Available at: https://arxiv.org/pdf/1712.01815.pdf. [Accessed 1 January 2018].

9. Christian, B., Griffiths, T., 2016. Algorithms to Live By: The Computer Science of Human Decisions. 1st ed. UK: HarperCollins.

10. Yoskowitz, J., 1991. Chess versus Quasi-Chess: The Role of Knowledge of Legal Rules, University of Illinois Press.

11. Stockfish. 2018. [ONLINE] Available at: https://stockfishchess.org/. [Accessed 1 August 2018].

12. Ripley, B. D., 1994, Neural Networks and Related Methods for Classification, Royal Statistical Society.

13. Curram. S, Mingers. J, 1994, Neural Networks, Decision Tree Induction and Discriminant Analysis: An Empirical Comparison, University of Warwick

14. Reynolds. R, 1982, Search Heuristics of Chess Players of Different Calibers, Nasson College

15. Ripley. B, 1994, Neural Networks and Related Methods for Classification, University of Oxford

16. Peterson. I, 1983, Playing Chess Bit By Bit, Science News Vol. 124, No. 15, pp. 236-237

The codebase of Engine A is publicly available at *https://github.com/nirvikbaruah/ChessEngineC-*

The codebase of Engine B is publicly available at https://github.com/nirvikbaruah/ChessNeuralNetwork