Using an Artificial Neural Network to Evaluate Chess Positions

Niklas Unrau
Computer Science Department
Cal Poly Pomona
Pomona, California USA
unrau73.wu@gmail.com

***Abstract*—Chess is an area in artificial intelligence that has received extensive study over the decades. The most dominant programs in this field utilize a mix of sophisticated search techniques, customized adaptations, reinforcement learning from games of self-play, and evaluation functions that have been manually designed and refined by human experts.**

In this paper, I present a method for training an artificial neural network (ANN) to evaluate chess positions. The ANN is trained using a large data set of chess positions, that are labeled with an evaluation from the strongest chess engine, Stockfish. My approach uses a deep neural network architecture with multiple layers and employs a variety of techniques for data preprocessing and model optimization.

I evaluate the performance of my trained ANN on a held-out test set and compare it to other commonly used chess evaluation functions, including material count and piece-square tables. My experiments show that my ANN outperforms these baselines, achieving a higher accuracy in predicting the outcome of chess positions.

My results demonstrate the effectiveness of using artificial neural networks for chess evaluation and show that they are crucial to improving the quality of chess engines.

***Index Terms*—artificial neural networks, machine learning, deep learning, chess**

I. Introduction

Contrary to popular belief, highly rated chess players are not better because they can calculate several moves ahead. Their strength lies in their ability to quickly understand the current state of the game. They can analyze the situation and determine which moves and lines to consider and how far ahead they should plan before making their move. Grandmasters use their understanding of the board to make wise decisions, leading to their success in chess.

Computer chess has been studied since the dawn of computer science, and both humans and machines have made significant improvements over time. Chess programs, such as Stockfish, which utilizes advanced alpha-beta search, and AlphaZero, which employs a reinforcement learning approach, surpassed human capabilities several years ago. One notable achievement in the field of artificial intelligence occurred in 1997, when Deep Blue defeated the reigning human world champion [1]. Since then, the gap between man and machine has only kept growing.

Many chess engines evaluate positions with functions that have been manually designed and refined by human experts. In this paper, I show how an artificial neural network (ANN) can evaluate chess positions. I trained a deep neural network with multiple layers and used a variety of techniques for data preprocessing to ensure the best possible performance.

The used data is from the free open-source chess server Lichess [2]. The played games are saved using Portable Game Notation (PGN), which lists the moves of the game. I are using Forsyth–Edwards Notation (FEN), which describes a board position, to train the ANN. Using the python library "python-chess" [3]. I fetched all occurred positions in FEN for each game. The data set contains the evaluation that Stockfish gives each position, which is used to train a model. I trained the ANN by converting each FEN record into a binary representation to use as the feature vector and the Stockfish evaluation as the label.

To check if my model is more efficient than simple evaluation functions like material count and piece-square tables, I compared them on a held-out test set.

II. Data set

The data set I used is part of the open database from the free open-source chess server Lichess. It contains all the standard rated games that were played on lichess.org for a given month [2]. I decided to use the most recent file, which contains 98,471,537 games that were played in February 2023. This single file has a rough size of 220 GB.

The data set for each month contains a lot of information about each game. This includes the date, the player names, their ratings, the time control, opening name, result, and much more. The game itself is saved in PGN. About 6% of the games include the needed Stockfish evaluation, meaning about 6 million of the games can be used to train my model.

I prepared the data for the training process in multiple steps. First, I got rid of the unwanted information from the PGN records, as well as appending the FEN representation after each move using the open source command-line program pgn-extract [4]. Afterwards, I filtered out all positions that have not been evaluated. I removed all unwanted characters to be left with a FEN string and the evaluation belonging to it. After filtering out duplicates, I had a list that consisted of about 480 million unique Stockfish-evaluated positions. To ensure that the distribution of the evaluation and features is similar across the data and reduce the risk of overfitting to a specific subset of the data, I also shuffled the data set.

The last step was to convert this data into a format that can be used to train an ANN. I generated a bitboard for each chess piece type. A bitboard contains 64 bits - one for each square of the board - and is either 1 or 0, depending on if the corresponding piece is on that square. Taking piece color into consideration, there are a total of 12 different piece types, which results in 768 bits. I appended 36 bits for a total of 804 bits to this sequence, which contains the additional information offered by an FEN record. This includes the active color and castling availability among other things. After this I were faced with a file sized roughly 500 GB used for training.

Besides manipulating the features, it is also crucial to adjust the labels for each instance. When evaluating a chess board, a number called the centipawn is used to indicate the relative strength or advantage of one side over the other. Centipawns are equivalent to 1/100$^{\text{th}}$ of a pawn and are the prevailing method for assessing a position. When Stockfish's evaluation results in a positive number it indicates that white has an advantage, while a negative number indicates that black has an advantage. For example, if the evaluation of a position is +1.5, it means that white has an advantage equivalent to one and a half pawns.

The evaluations that are provided by Stockfish have two problems that can be detrimental to the learning of my ANN. For one it gives incredibly high advantage evaluations if it is not able to find a forced mate in an extremely winning positions. To eliminate those outliers I bounded the labels to a range of ± 15. And for another Stockfish stops giving evaluations in the form of a centipawn advantage once it found a forced mate. The evaluations for all of those positions are the number of moves until mate including a leading pound sign. All of those positions are also converted to ± 15.

For the final comparison against other evaluation functions I separated a data set with roughly 35 thousand random positions.

III. Methodology

The task of evaluating a chess position based on various features is a complex and nonlinear problem. Traditional methods, such as rule-based or expert systems, may not capture the full complexity of the problem and may require extensive manual tuning. Therefore, I decided to use an artificial neural network (ANN) to tackle this problem.

The final ANN that I used was trained on the entire data set. I have conducted some tests on smaller parts of the data set to find a solid architecture and set of hyperparameters. The input layer consists of 804 nodes for the 804 bit sequence mentioned in Section II. I used five hidden layers with 804 nodes per layer. Increasing the depth did not result in any significant increase in performance. Each layer is activated by the Rectified Linear Unit (ReLU) activation function. The output is a single node that gives the predicted evaluation in centipawns. I used Adam optimization with a parameter of $\eta = 0.001$, as well as a batch size of 1024 for a good trade-off between training speed and generalization. The ANN is trained using the L^1 loss function, because it fits nicely with the centipawn evaluation system. If the absolute difference is one, my model is off by one pawn.

It is important to mention that this architecture is definitely open to a lot of improvement. I were faced with the challenge that training on a small data set does not give enough insight into the performance on the entire data set, but training on smaller sets still took a significant amount of time.

For the evaluation of the resulting model I compared it against two simple evaluation functions. The simplest one is using material value. It works by assigning the values from Table I to each piece and adding up the total. These numbers were also proposed in Claude Shannon's groundbreaking paper called *Programming a Computer for Playing Chess* [4] and has since stood the test of time.

TABLE I
Piece Values

Pawn	Knight	Bishop	Rook	Queen
1	3	3	5	9

This simple function can be improved using piece square tables. The value of each piece gets modified by a constant which is defined for each piece for each square. Knights get a bonus if they are closer to the center of the board and a penalty if they are close to the corners, while the king gets an bonus if he is in one of his own corners, but a penalty if he is on the enemy side of the board. To rule out any errors in my test set I also predicted an evaluation of ± 0 for every position.

IV. Results

The ANN was trained on all 480 million positions. I used a NVIDIA GeForce RTX 3080 GPU and the process took approximately 30 hours to complete. Fig. 1 shows the L^1 loss over iterations. As I can see the loss is decreasing at a slow and steady rate. The ANN finished with a final loss of about 1.45 and due to the huge state-space and game-tree complexity of chess I are not worried about overfitting, so a similar result should be seen for evaluation.

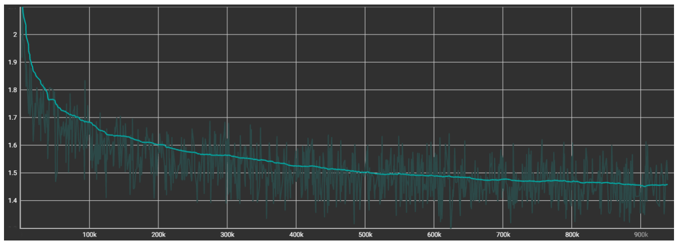

Fig. 1. Training loss over time.

I evaluated my model on the test set with a size of about 35 thousand positions. I compare the different evaluation functions using the L^1 distance, also known as Mean Absolute Error (MAE), for the same reason that I used it as the loss function in Section III. I also take the Root Mean Squared Error (RMSE) into consideration, because it gives equal weight to all errors and penalizes large errors more heavily than small errors. Table II shows the result of my evaluation.

TABLE II
EVALUATION RESULTS

Evaluation Function	Always Zero	Material Count	Piece Square Table	ANN
MAE	3.94	2.47	2.39	1.47
RMSE	6.35	4.24	4.13	2.77

The results of this experiment show that my ANN achieved an drastically better performance in both metrics in comparison to the simple evaluation functions. This suggests that my ANN can accurately evaluate chess positions.

V. Related work

In recent years, artificial neural networks (ANNs) have shown great potential in the field of computer chess. Several researchers have explored the use of ANNs to evaluate chess positions and improve the strength of chess engines.

In 2017 Sabatelli et al. [6] started investigating the strength of Neural Networks for evaluating chess positions. They compared different board representations and model architectures training on 3 million positions. They showed that a representation in bitboards, similar to the way I did, performs better than a saving the piece values of all pieces on a single board.

A new architecture was proposed by Yu Nasu in 2018 for the game Shogi [7]. It is called Efficiently Updatable Neural Network (NNUE). In 2019 and 2020 the architecture was successfully ported to chess and integrated into the Stockfish engine. The training process works similar to the one proposed by me with an improved board representation. The neural network is updated using an incremental learning approach, which allows it to adapt to new positions as they are encountered during gameplay.

The use of NNUE has been shown to significantly improve the performance of traditional chess engines. In a recent benchmark test it was revealed that Stockfish NNUE was stronger than the classical one by at least 80 Elo [8].

Another interesting approach was introduced in 2017 by Silver et al. when they published their paper about AlphaZero [9]. AlphaZero is a computer program that uses a combination of deep neural networks and Monte Carlo tree search to play the games of chess, shogi, and Go. AlphaZero is a significant achievement in the field of artificial intelligence and game playing, as it demonstrated that a single algorithm can master multiple complex games without any prior knowledge beyond the rules.

Overall, these advances in the field of game-playing AI have demonstrated the potential of deep learning and reinforcement learning techniques to tackle complex problems in a variety of domains.

VI. Conclusion

In this paper, I presented a method to evaluate chess positions using an Artificial Neural Network (ANN). I trained the ANN on a large data set of positions and achieved a high accuracy in evaluating chess positions.

My results showed that the ANN was able to outperform traditional evaluation methods, such as the piece square tables, and demonstrated a better understanding of the complex relationships between pieces and their positions on the board.

The ability to quickly and accurately evaluate chess positions can greatly improve the performance of computer chess engines and lead to more engaging and challenging games for human players.

In conclusion, my work highlights the potential of using Artificial Neural Networks to improve the evaluation of chess positions. While my method achieved promising results, there is still room for improvement and further research could explore the use of more advanced neural network architectures and training techniques.

VII. Supplementary material

All code used in this paper, including the implementation of the neural network and the preprocessing scripts, is publicly available on GitHub at the following URL: https://github.com/Niiklasu/chess-ann. The full data set for the training is not in this repository as it has a size of almost 500 GB, but an step-by-step explanation on how to convert the Lichess pgn file can be found.

The LaTeX file to generate this paper can also be found in the repository.

The code and data sets provided here can be used to reproduce the results reported in this paper, as well as to extend the analysis to new data sets and research questions.

References

[1] "Kasparov-Deep Blue 1997," www.chessgames.com. https://www.chessgames.com/perl/chesscollection?cid=1030300 (accessed Mar. 15, 2023).

[2] "lichess.org open database," database.lichess.org. https://database.lichess.org/

[3] N. Fiekas, "python-chess: a chess library for Python," GitHub, May 04, 2023. https://github.com/niklasf/python-chess (accessed May 05, 2023).

[4] "pgn-extract: Portable Game Notation (PGN) Manipulator for Chess Games," www.cs.kent.ac.uk. https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/ (accessed May 05, 2023).

[5] C. E. Shannon, "XXII. Programming a computer for playing chess," The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. 41, no. 314, pp. 256–275, Mar. 1950, doi: https://doi.org/10.1080/14786445008521796.

[6] M. Sabatelli, F. Bidoia, V. Codreanu, and M. Wiering, "Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead." Accessed: Apr. 11, 2023. [Online]. Available: https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/ICPRAM_CHESS_DNN_2018.pdf

[7] Y. Nasu "Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi 1," 2018. Available: https://dev.exherbo.org/~alip/doc/nnue_en.pdf

[8] "Introducing NNUE Evaluation - Stockfish - Open Source Chess Engine," stockfishchess.org. https://stockfishchess.org/blog/2020/introducing-nnue-evaluation/

[9] D. Silver et al., "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," Dec. 2017. Available: https://arxiv.org/pdf/1712.01815.pdf