Deep Neural Network Guided Chess AI

Reese Kneeland, Alex Jonas, Zecheng Qian, Vishnu Ravichandran
University of Minnesota, Twin Cities
kneel027@umn.edu, Jonas060@d.umn.edu, qian0102@umn.edu,
ravic038@umn.edu

Abstract. Deep learning has dominated computer chess games during recent years. It outperforms other search algorithms in artificial intelligence because of its full representation power. In this work, we explore the general pipeline of developing a chess AI using a fully connected neural network and a supervised learning scheme that leverages knowledge from human experts.

Keywords. Deep Learning, Neural Network, Min-Max Searching, Chess Game

1 Introduction

Most chess AIs have intricate designs and thousands of lines depicting strategies and planning. Some use Min-Max searching and Alpha-Beta pruning [1]. Such schemes utilize prior knowledge from human-designed heuristic functions and most of the time the agent cannot outperform an expert in the game. This is particularly true for games with a large state space, as an optimal solution may not be easily found or may not be solvable in polynomial time (which we refer to as NP-hard). Such algorithms usually have deterministic processes for playing a game, meaning adversarial attacks can be easily found if one is familiar with the patterns played by the game agent. Besides, Min-Max and Alpha-Beta require long time intervals to perform a single move particularly with complex states. Therefore our teams goal is to design an agent which can outperform Min-Max implementation, while making decisions quicker then Min-Max and Alpha-Beta.

Related work in the deep learning community such as AlphaZero in 2017 [2] gives us inspiration for developing our agent which can learn from past experience and make optimal decisions for each move based off past experience. There are mainly two categories of deep learning models in chess games. One is model-free, it is usually implemented with deep reinforcement learning and deep Q-networks. Such a scheme does not rely on particular experience from human experts, and it can enhance itself by trial and error. However, training for such a chess AI is extremely expensive, AlphaZero cost $35M to train [3], which is impractical for our team. Since deep reinforcement learning is costly and inefficient, the idea for deep reinforcement learning was quickly scrapped. Instead, an alternative approach was utilized - supervised learning. There are many open source chess evaluation engines, such as Stockfish, can make relatively good evaluations on moves within Chess games. The Stockfish evaluation prior knowledge can be used to bootstrap the learning process of a supervised model with reduced material cost as compared to deep reinforcement learning.

Our team developed a chess AI agent [4] driven by Min-Max heuristic functions which was then integrated with deep neural network creating an ensemble model. The ensemble model enhanced the performance of the Min-Max searching algorithm by using a mixture of weighted predictions from the Min-Max algorithm and the deep neural network.

The Min-Max was developed using a general heuristic guided chess game AI agent. To measure the heuristic we populated a Mini-Max tree with the values of the heuristic of each move, determining the best possible move for each turn. Our heuristic was implemented by researching and determining patterned behavior of how professional players make move choices then applying the same principles to our chess AI heuristic. We quickly learned that AI can choose paths that could mimic a real human strategy or even unorthodox moves that could confuse the opponent. Resulting in a momentary advantage for the AI.

Currently the AI makes the best possible move depending on how favorable (largest possible positive value) the heuristic values of the evaluated board positions are in the move tree. The default configuration allows 5 seconds of computation time per move, using an iterative deepening algorithm (typically reaching a move depth of 3-4 in the allotted time). One of the most important strategies in chess was to learn how to sacrifice a chess piece in a way that you will gain more than you lost thus resulting in a higher overall heuristic value for the board. The sacrificial move remains one of the hardest strategies to implement into our Min-Max which has not been implemented with success. Part of the reason this model can struggle at times is the limited look ahead, causing it to miss how a current move could lead to a worse position in the future.

Due to our models limited capability with only using a heuristics based prediction our team incorporated the power of deep neural networks to enhance our chess AI.

Deep neural networks have high representing power and can theoretically model arbitrary functions [5]. Deep Neural Networks provide a new tool to overcome some problems of traditional chess AI such as deterministic models and scalability issues. We model the whole agent-environment system with a deep neural network and use stochastic gradient [6] to train it on a large-scale data set. There are several well-known works in the area of deep learning in chess game, one is AlphaGo which became well known in 2016 by defeating a human world champion in the game of Go [7]. It also beats Stockfish 28–0, with 72 draws, in a 100-game match. Another is Leela Chess Zero, developed by programmer Gary Linscott, who is also a developer for the Stockfish chess engine. These works are also the first to demonstrate the effectiveness of deep learning on high-dimensional data and close games.

2 Data

The data set for our project is provided by Lichess [8], who provides monthly data sets containing every game played on the website. July 2021, for example, contains $92+$ million games, about 6% of which contain Stockfish evaluations for the moves played, evaluating to $5.5+$ million games. This 6% was the training data and additionally, each game contains on average 80 unique positions, giving us $440M+$ pieces of data to train our network on which we reduced to 37 million moves for ease of training. The data was in the form of a "Portable Graphic Notation" which is a text based format of encoding chess games in algebraic notation. A SQLite Database was created due to the size of the data which was 25 GB.We also created an iterable dataset which allowed us to simulate a real game by seeing each move every iteration. The data itself consisted of a FEN, Binary representations and evaluation of each move. The FEN representation of the chessboard was used as input into our Neural Network (after being encoded into a 808 byte length binary). The Stockfish evaluation (eval) was used as our labels. The binary provided was not chosen as the binary encoding representation was unclear. Training was done on the already given binary representations however the outcome led to erroneous predictions during testing. It was determined the erroneous predictions were due to additional information other then the FEN of the chessboard were encoded in the binary formatting already provided.

Processing our data in a binary-encoded FEN format enabled the Neural Network to optimize evaluation prediction purely on evaluating board positions. The resultant prediction would then be fed into a Mini-Max evaluator along with a heuristic prediction allowing the agent to make the highest positively predicted decision.

3 Methods and Algorithms

Our team employed an integrated ensemble combining Min-Max search and a deep linear neural network utilizing f1 loss as our cost function to build a chess AI.

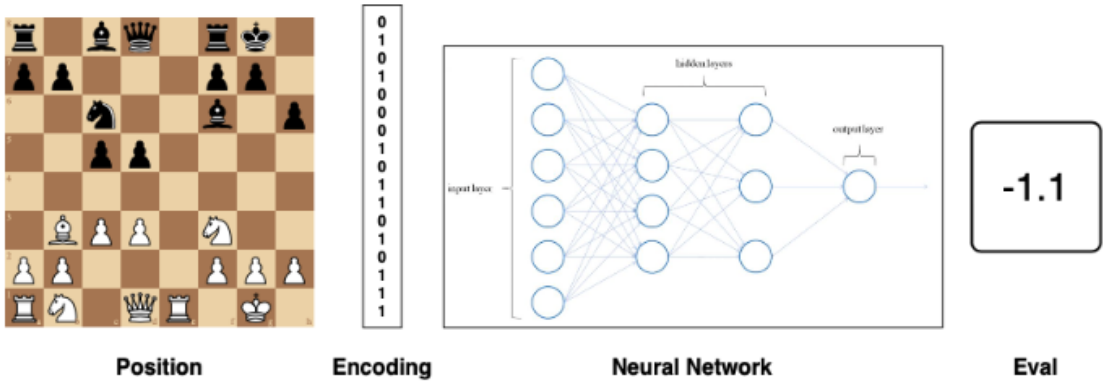


Figure 1: Model Architecture

Each FEN for the training data set had a corresponding Stockfish evaluation score depicting to what degree the chess board was favorable or unfavorable to white. The Stockfish evaluations consisted of a range $[-15, 15]$ when evaluating a given chessboard. Data was inputted into the Neural Network in batches of 2096 binary encoded $(808$ bits$)$ FEN representations of differing chessboards (each FEN corresponding to a chessboard) with a corresponding Stockfish prediction as the label. F1 loss was employed to determine how large a cost our network accumulated when comparing the Neural Network prediction for the FEN board evaluation against the Stockfish evaluation. After our cost was determined for the initial batch our weights were updated then a new batch of 2096 FEN representations were loaded in. Over $17,000$ batches were utilized for training resulting in over 37 million data inputs.Refer to Figure 1 for the architecture of our model. We were able to reduce our overall F1 loss from 5.5 to about 3.01 after training on 37 million data inputs. Refer to figure 2b for overall loss during training. Meaning our model was more capable of determining a chess board Stockfish representation correctly after training.

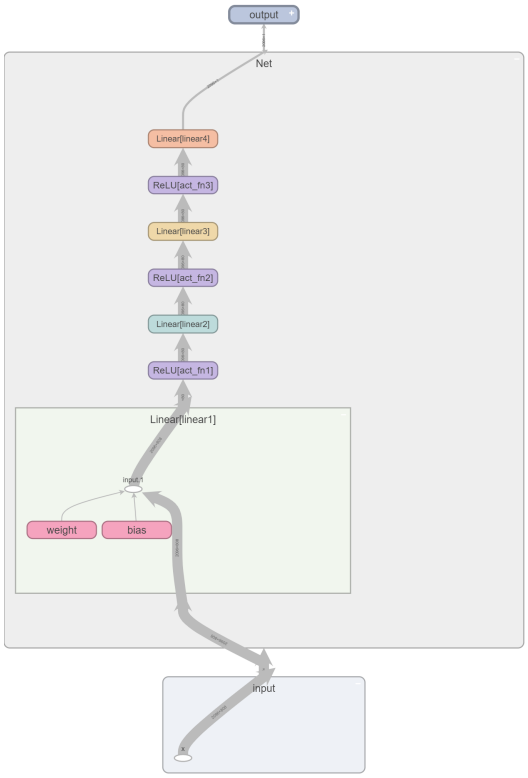
Deep Network Architecture

Our Network architecture consisted of a four-layer Linear Neural Network. Node representation remained uniform throughout the input and hidden layers. Each layer consisted of 808 nodes which coincided with our 808 binary input. Our binary input was the encoded FEN representation of the chess board. ReLU activation functions were utilized for each hidden layer resulting in 3 layers of ReLU activation functions. No activation function was employed for our output as we wanted to keep our networks prediction within the correct range of $[-15, 15]$ rather then manipulating our raw prediction into the ranges of $[-1, 1]$ or $[0, 1]$. Refer to Figure 2a for the structure of our Neural Network.

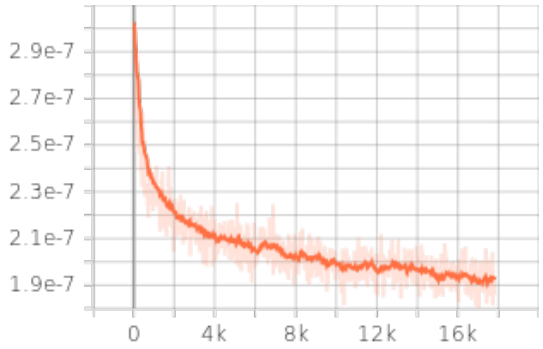
4 Experimental Setup

To evaluate the performance of our engine, we did a number of tests in the ranked games of chess.com, starting with a fresh new account and letting the ELO system rate the engine as a player on the site. This also allowed us to observe the games played from an analytical perspective, to help understand the mistakes the engine would make to help fix them in future iterations.

The Neural Network was trained on +17,000 batches of 2096 binary encoded FEN chessboard inputs. The starting learning rate utilized was 1e-3. A few training iterations helped determine that a learning rate of 1e-2 was to high as the model would initially reduce in cost only to gain after training on around batch 5,000. We also quickly deduced 1e-3 was also too low as our cost function



(a) Network Structure



(b) Loss

Figure 2

line appeared linear after training on all batches. The objective function for the neural network was an F1 Loss function. We incorporated the F1 loss as it would give us an adequate measure of distance between the models prediction and the Stockfish evaluation label which consisted of a range within [-15, 15].

The Neural Network utilized stochastic gradient descent with an Adam optimizer function. The Adam optimizer much like a momentum optimizer made use of exponentially decaying average past gradients when updating our models weights after each batch. The exponentially decaying average gradients are also squared to provide an adaptive learning rate for our model.

5 Results

Testing our ensemble model on chess.com we were able to assign a rough chess ELO rating of $600 - 800$ to our engine, and observe its game-play style. It performs well during the opening and middle stages of the game, with the manual heuristic checks preventing poor moves in which our agent would lose pieces or give up important board positions, and the deep learning model helping to weight common effective human chess strategies higher and perform more complex sequences of moves. This style of play helps the agent maintain a reasonable board position into the middle-game and appear as though it can keep up with quite competent players. This quickly falls apart in the endgame however, as it's manual heuristic is not complex enough to classify detailed endgame positions, in which positioning is significantly more important. The deep learning model also did not help significantly with this stage, as it did not effectively learn checkmating patterns and incentives useful to help close out close games in the ending stages of the game. These late

game weaknesses led to losses against pretty much any competent chess player, and highlighted the most glaring weaknesses of the models chosen for this engine. It is also worth noting that the endgame weaknesses of our ensemble method are pronounced even further when only the manual heuristic is used, but that the opening and early game are helped significantly by this ensemble configuration.

6 Conclusions and Future Work

To summarize, we developed a chess AI ensemble combining Min-Max search with a deep neural network capable of outperforming a chess AI guided solely by a Min-Max search implementation. We compared the performance of two models using several metrics and did a comprehensive analysis of the model performance. For future work, we noticed that the model is not always reliable in making the optimal prediction. This unreliable quality of our model is particularly true in middle to late game as the performance of our ensemble model gradually degrades toward mid-game and late game. Sometimes the moves may not be desirable and the agent loses the game unexpectedly. One of the future directions we would like to explore is how to make the model reliable in these more nuanced game states. Much research has been conducted on reinforcement learning and deep Q-networks [9] and may prove to be a potential solution to our models reliability issue. We recently discovered that Double Q-Learning [10] can rid the bias of the Q value estimator by using an opposite estimator network resulting in further stabilizing the model. If we chose to continue updating our model through supervised training updating our network with complex and more novel deep network architectures would help improve the model performance. To conclude reinforcement learning and Q-networks remain future research directions for our ensemble model.

References

[1] T. S. Anantharaman, *A statistical study of selective min-max search in computer chess*. Carnegie Mellon University, 1990.

[2] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, and J. Wang, "Overview on deepmind and its alphago zero ai," in *Proceedings of the 2018 international conference on big data and education*, 2018, pp. 67–71.

[3] (2020) How much did alphago zero cost? [Online]. Available: https://www.yuzeh.com/data/agz-cost.html

[4] (2022) Github code repository. [Online]. Available: https://github.com/reesekneeland/Guided-MiniMax-Chess

[5] B. C. Csáji *et al.*, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, no. 48, p. 7, 2001.

[6] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[8] (2022) Lichess.org open database. [Online]. Available: https://database.lichess.org/

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.