Using Deep Learning to Construct a Chess Evaluation AI

Jonathan Shi
jshi037@ucr.edu

Matthew Hunt
mhunt023@ucr.edu

Nathan Romanelli
nroma009@ucr.edu

Abstract

We build a shallow neural network and a deep neural network to evaluate chess board positions on data obtained from the FICS dataset. We obtain an evaluation using three data augmentations: context, worldview, and dropout which are ran on the shallow neural net. For the shallow neural net, we achieve a baseline training accuracy of 82.5% and test accuracy of 80.5%. For the deep neural net, the training accuracy and test accuracy are 90.7% and 87.3% respectively. For each data augmentation, context gives 89.9% training accuracy with 76.2% test accuracy, and worldview has 89.0% training accuracy and 86.1% test accuracy. In the case of dropout, worse accuracy is obtained the higher the dropout rate, dropping as low as 20% training accuracy and 40% test accuracy.

1 Introduction

Chess is a deceivingly complex games with simple enough rules, making it an attractive game for people to test their wits and strategy against one another. As with any other game, many try to “solve" chess the same way that Tic-Tac-Toe or Connect 4 have been solved. This naturally leads to extensive theory about chess openings and endgames, and countless principles to understand in the billions of possible positions over the board.

When Stockfish 15 is developed, its developers have arguably come extremely close to that goal, completely changing the approach to the game. Nowadays, top grandmasters use Stockfish 15 as a reference to improve their own game, and try to emulate computers during competition to get the biggest edge possible.

We have built two different types of models, under two different training methods, and three additional data augmentations with the baseline data.

In Section 2, we discuss the architecture of Stockfish and its impact. In Section 3, we discuss the models and architecture choices used for chessAI. In section 4, we discuss how the data is obtained, processed, and augmented to obtain results. In section 5, we discuss and analyze the results obtained from the models. Finally, we propose some conclusions and future directions for advances in sections 6 and 7.

2 Stockfish

The main source of inspiration comes from Stockfish, the most powerful engine publicly available. Stockfish processes data to find the best move in any given position and provides an evaluation. Stockfish implements an efficiently-updatable nerual network (NNUE) that utilizes an extremely large input layer of 10.5 million parameters, three much smaller layers, and a scalar output providing an evaluation on the position [2].

Part of the cleverness of the NNUE is that making a move on the chessboard alters very few entries of the vector. This makes computation vastly more efficient, and is able to be incorporated into a move-searching engine to deliver the best move in a very reasonable amount of time.

Our models takes some inspiration from Stockfish’s design in data preprocessing and model architecture. However, our models runs on a significantly smaller scale with 11 million available positions in our dataset compared to the 100 million positions that Stockfish uses for evaluation. Our models are also not as computationally efficient as Stockfish's from a software perspective and not optimized for real-time evaluation as Stockfish is.

There is another large distinction between Stockfish's model and our model. Stockfish's model outputs a single scalar as an output to be interpreted as an evaluation of the position. For the interpretation of this evaluation, a large positive number represents a significant advantage for White, a large negative number represents a significant advantage for Black, and a number that is close to 0 represents an even position. Our models follow a classification model, which will be discussed further in sections 3 and 4.

3 Models and Architecture

We have two different models that we use to create our deep state board evaluation. The three models are the simple neural network, the deep neural network, and the transformer. Each of these models takes in the data consisting of the boards and pieces for the games played by the skilled chess players and then tries to predict good moves based on if the moves were played by the winning or losing player. We then take these predictions and calculate the accuracy with test boards never seen before by the model to get an idea of the effectiveness of our chess artificial intelligence.

3.1 Simple Neural Network

The Simple Neural Network consists of one hidden layer that it puts the data through and predicts with the Adam optimization and the Cross Entropy loss used in pytorch. Some hyperparameter values we used for this model was a learning rate of 0.001, a hidden layer size of 770 and 80 total epochs. Running the data through this model gives about 80 percent test accuracy which is pretty decent for a simple model.

3.2 Deep Neural Network

The Deep Neural Network consists of ten hidden layers that the data goes through with consistently decreasing sizes of the hidden layers. The Adam optimization and Cross Entropy loss is used again to predict between the three classes of win, loss, and draw. The hyperparameters are also the same with a learning rate of 0.001 and 80 total epochs. The overall test accuracy of this model is about 90 percent which is better than the Simple Neural Network.

4 Data Preprocessing

The skill level of the players are a relevant factor to consider when selecting games to use for our dataset. The skill level of a player can be determined using an ELO rating system which assigns an integer to every player representing their strength. Today, the average grandmaster's rating ranges between 2400-2700 ELO. For reference, the strongest chess engine to date is Stockfish 15 with an ELO of 3620.

We train the model on games that are obtained from the FICS Games Database from 2019 to 2022, restricting the games to those which use standard time controls and have an average rating of over 2000. This ensures the games represent higher quality moves, since running out of time is a viable winning option when playing on lower time controls such as blitz (3-5 minutes per player) or bullet (1 minute per player). Under faster time controls, a player is way more likely to play many blunders at a fast rate, but still win the game due to their opponent running out of time.

Each game is encoded with Portable Game Notation (PGN), which describes the entire move order of a game. However, we must examine a game position by position to properly receive an evaluation.

Each board state is encoded using Forsyth-Edwards Notation (FEN), which uses a string to describe the position of each piece at any given move. The white pieces are denoted with capital letters, and the black pieces with lowercase letters. The string uses forward slashes as delimiters, and describes the board row by row from top to bottom. As such, since FEN only encodes one board state, a game can be expected to have on average 40-50 FENs representing each move made by both players. This means our dataset has approximately 11 million data points, but with hardware limitations, the models are trained on 1 million data points instead.

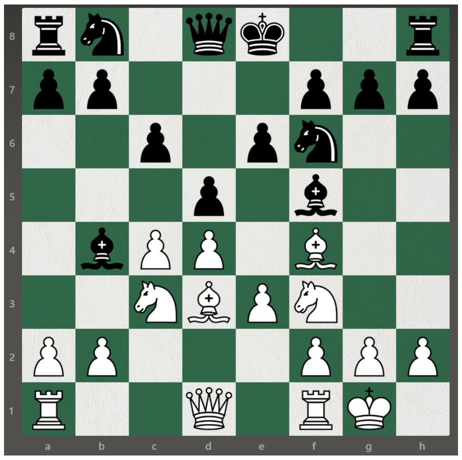

Figure 1: Board state with FEN encoding rn1qk2r/pp3ppp/2p1pn2/3p1b2/1bPP1B2/2NBPN2/PP3PPP/R2Q1RK1 w kq - 0 1

Each FEN encoding is converted into an $8\times8\times12$ one-hot tensor. The 8×8 axes of the tensor describe the location of a piece, and the third axis encodes the information of the piece type for the 12 unique pieces on the chessboard. The tensor is then flattened into a tensor of length 768.

We label each position by the result of the game with $\{1, 0, -1\}$ representing a White win, draw, and a Black win, respectively. This is fundamentally different from the Stockfish evaluation, since we use a classification model to evaluate a position.

The flattened one-hot tensor and its corresponding label are now the inputs that we can feed into the classification model and receive an classification.

4.1 Data Augmentation

We explore three data augmentations to see whether the model improves its performance.

4.1.1 Context

The Context data augmentation provides context for the board state. Each data point is augmented to include the board states of two previous moves. Intuitively, this means that there is a sliding window with a width of 3 board states that is moving over the dataset for the model to train on.

For the first and second moves of each game, the data is padded with the starting board state to make the data 3 boards long.

4.1.2 Worldview

FEN encoding implicitly encodes the board state from White's "worldview", placing the White pieces on the bottom of the board. Pawns move "up" the board, and so Black's pawns from White's perspective would be backwards. We alleviate this by padding each data point with Black's worldview. This also emulates how Stockfish preprocesses the data to improve its evaluation.

The reason for padding the player-relative coordinate systems is so that the neural network can be used regardless of whether it is evaluating for White or Black's position. Both worldview's are highly relevant, since the goals of both sides are completely opposite one another (namely you want to protect your own King and capture the enemy's King) [2].

4.1.3 Dropout

Dropout removes nodes randomly with probability p from each layer during the training process. For training, we train the model dropout with probabilities $p = \{1.0, 0.5, 0.1\}$, where p represents the probability that a node does not get removed. As we are training on positions and some positions may appear multiple times with different labels, we want to see whether omitting some nodes will improve the performance of the model.

5 Results

5.1 No Augmentation Shallow network

For our simple neural network, with a single hidden layer, we achieved a training accuracy of 82.3% and a testing accuracy of 81% with the training curves show below.

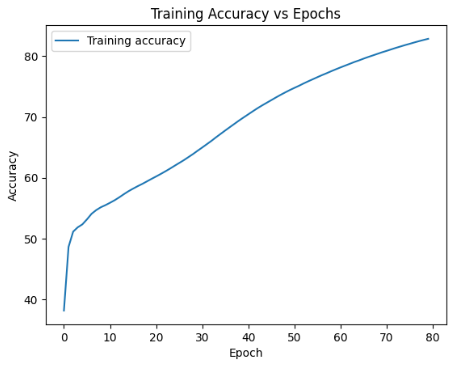

Figure 2: Training accuracy over training time for shallow neural net

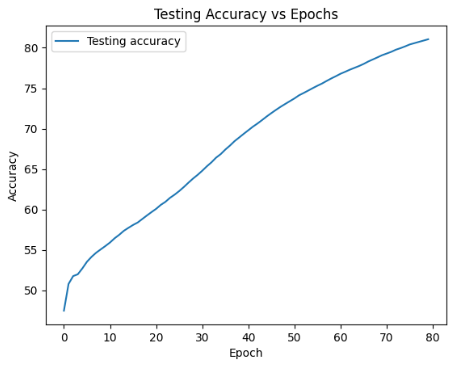

Figure 3: Testing accuracy over training time for shallow neural net

Our deep Neural network achieved a training accuracy of 90.7% and a testing accuracy of 87.3%.

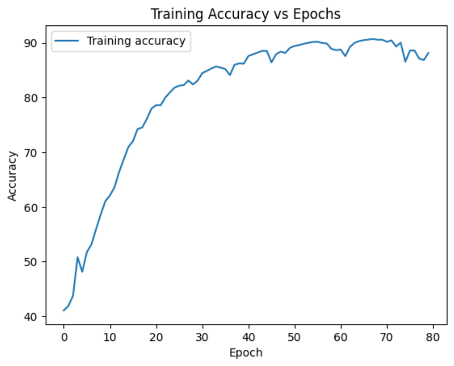

Figure 4: Training accuracy over training time for deep neural net

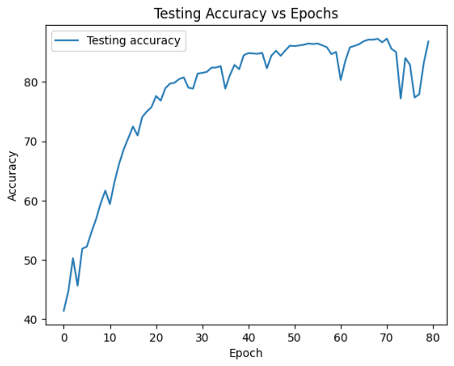

Figure 5: Testing accuracy over training time for deep neural net

5.2 World View Augmentation

Using the world view augmentation, our simple neural network achieved a training accuracy of 88.6% and a testing accuracy of 86.6%.

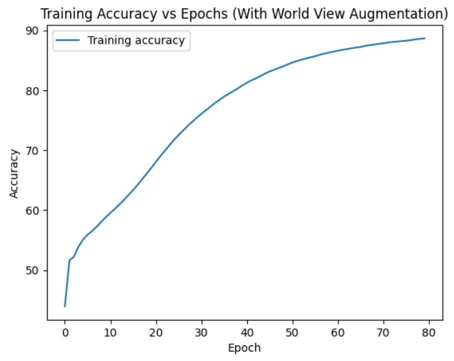

Figure 6: Training accuracy over training time for shallow neural net with world view augmentation

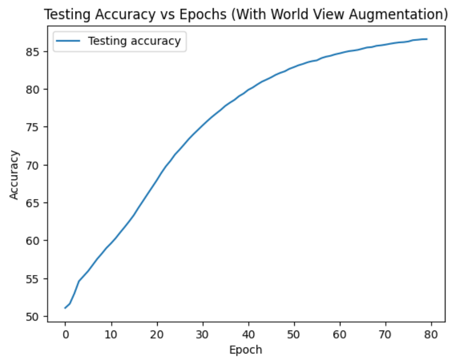

Figure 7: Testing accuracy over training time for shallow neural net with world view augmentation

5.3 Context Augmentation

Using the context augmentation, our simple neural network achieved a training accuracy of 89.9% and a testing accuracy of 76.6%.

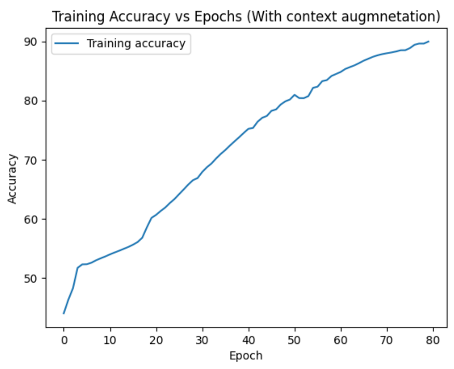

Figure 8: Training accuracy over training time for shallow neural net with context augmentation

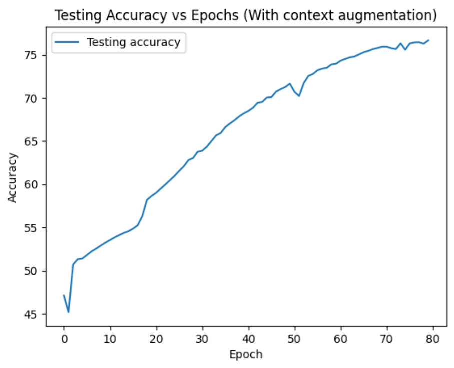

Figure 9: Testing accuracy over training time for shallow neural net with context augmentation

5.4 Dropout Augmentation

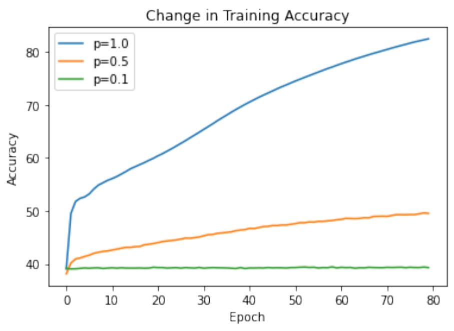

Figure 10: Training accuracy over training time for shallow neural net, with dropout

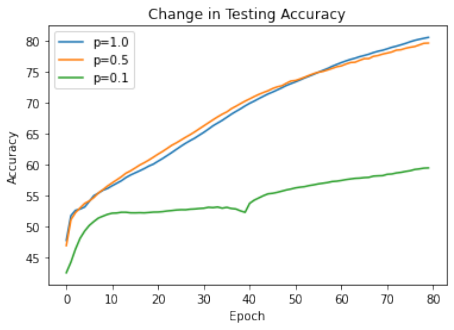

Figure 11: Testing accuracy over training time for shallow neural net, with dropout

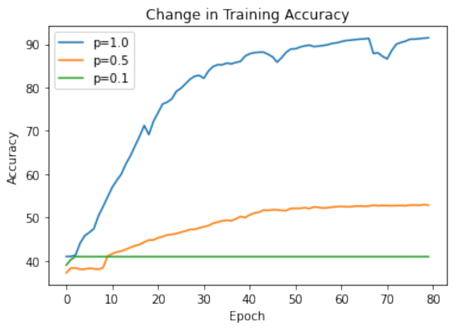

Figure 12: Training accuracy over training time for deep neural net, with dropout

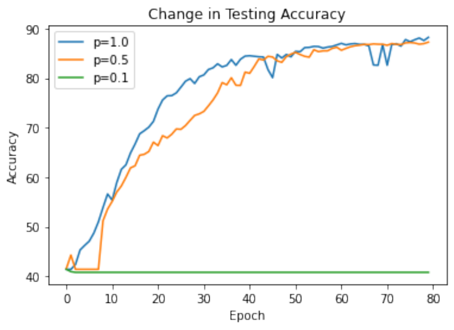

Figure 13: Testing accuracy over training time for deep neural net, with dropout

6 Discussion

When Stockfish evaluates a position, it scales the output number by some heuristic number which was altered by grandmasters working with the developers over the course of the research [1].

6.1 Interpretation of Results

The shallow network with no augmentation has sub-par results (as we somewhat expected). The deep network out-preforms it by almost 7% improved accuracy. While this is a great case for the deep network, the data augmentation also give optimistic results. The world-view augmentation increased the accuracy of our model by 6%, almost as accurate as the deep network. Our context augmentation provides a worse training accuracy, and we believe it is because learning context may not provide a benefit for state evaluation as one move can drastically change the game state. For instance, a game could be winning for white until white blunders a piece, which context would make hard for our model to detect. As such, we think the best model we could train would be a deep model with the world view augmentation.

While our results classify a position as winning for White, winning for Black, or a draw, we can extend this to an evaluation by taking some sort of heuristic function for the probabilities that the models output. For example, if it is White's turn to play, we can evaluate a position as

$$\text{Evaluation} = \max(P(W) - P(B), P(D) - P(B))$$

where $P(W)$ denotes the probability of a White win, $P(B)$ the probability of a Black win, and $P(D)$ the probability of a draw.

7 Conclusion

Our model provides us insight into the strength of Stockfish and chess evaluation models as a whole. The context of a move (context being the moves leading up to a board state) does not provide beneficial information for the model to determine the evaluation of a board. Additionally, providing both angles of the board (as the board is flipped vertically and horizontally for black) allows the model to perform better. Intuitively, this makes sense, since our models are not immune to rotations/mirroring. Finally, a deep network out preforms our shallow networks, most likely due to the fact that it is more expressive.

References

[1] `https://www.chessprogramming.org/Score`.

[2] apgoucher. The neural network of the stockfish chess engine. *Complex Projective 4-Space*. `https://cp4space.hatsya.com/2021/01/08/the-neural-network-of-the-stockfish-chess-engine/`.