APS360 FINAL REPORT - CHESS ENGINE

Michael Akzam
Student# 1007978501
mcihael.akzam@mail.utoronto.ca

Meet Patel
Student# 1007896038
meetd.patel@mail.utoronto.ca

Abhay Walia
Student# 1007447746
abhay.walia@mail.utoronto.ca

Yazan Al Macki
Student# 1007634482
yazan.macki@mail.utoronto.ca

ABSTRACT

The following document is a final report for our chess engine AI using deep learning. It outlines the implementation of our data processing, baseline model, as well as primary model, and goes over the results achieved after training and testing. Qualitative and quantitative results are highlighted to show the effectiveness of the learning model. —-Total Pages: 9

1 PROJECT DESCRIPTION

For centuries, chess has served as a mind-captivating game that has fostered strategic thinking, decision-making as well as problem-solving. The goal of this project is to delve into the world of deep learning to develop a cutting-edge chess engine which can push the boundaries of what machines can achieve in strategic thinking and game analysis.

Our motivation to engage with this classic game is driven by the idea of enhancing it by making it more accessible to beginners, as well as enabling advanced players to further sharpen their skills. By utilizing deep learning, we can address challenges in chess, such as pattern recognition, position evaluation, and endgame analysis, which have not been as accessible as needed. This is a particularly important project since it helps to bridge the gap between individuals of varying skill levels while allowing them to attain a deeper strategic understanding of the different approaches to chess based on their playing style.

Deep learning will be a great approach for this project due to its ability to learn, adapt and generalize well when given large amounts of training data. The existence of vast databases of chess games from players around the globe provides a rich data source which can be used to effectively train deep neural networks. On top of that, deep models are ideal since they continue to learn and improve over time, making them great at adapting and generalizing to new unseen data. Ultimately, this project will create a chess engine using deep learning techniques to provide a platform where players of all levels can improve their skills and strategic thinking.

2 ILLUSTRATION

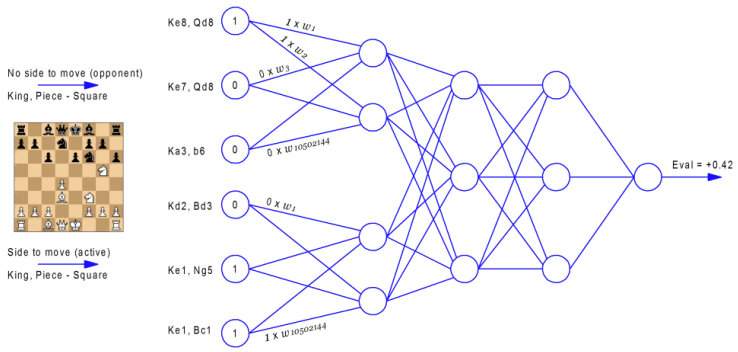


Figure 1: Overview of the model

The image above (Figure 1) provides a snapshot of the project's primary objective. In essence, a user inputs their move, which is then processed by a neural network trained on a multitude of chess games. The network evaluates this input and outputs the best move that as an evaluation of the current state of the chess board.

3 BACKGROUND & RELATED WORK

IBM's Deep Blue was the first built chess computer that defeated world champion Garry Kasparov in a six-game match in 1997 (IBM). Since then, hundreds of different chess AIs were created in the aims of generating the best positional advantages move after move. The following section discusses the different chess engines that are dominating the online chess world currently available, alongside academic breakthroughs within the field.

3.1 CHESS ENGINES

3.1.1 STOCKFISH

Stockfish is currently the highest rated chess engine in the world, winning the Top Chess Engine Championship 14 times (chess.com). Its model uses complex evaluation functions that assesses the quality of a given position taking into account factors like material balance, piece mobility, pawn structure, and king safety. It picks its best moved based on a sophisticated search algorithm called the "Alpha-Beta Pruning", reducing the number of positions the engine has to consider, making it one of the most efficient and advanced engines on the market (wikipedia).

3.1.2 LEELA CHESS ZERO

Soon after the release of AlphaZero, Google's first engine to utilizes a combination of deep neural networks and reinforcement learning, Leela Chess Zero was created as a purely self-learning open-source chess engine based on AlphaZero's contributions, which focuses on other strategic games as well, such as Go and Shogi. Using the same approaches such as Deep Neural Networks, Monte Carlo tree search, and its evaluation function, it mainly differs by its training data, where Leela Chess Zero is purely being fed high ranked chess games, most of them being grand master games, the highest attainable chess ranking globally.

3.1.3 KOMODO

Komodo is one of the most successful chess engines on the market. Bought in 2018 by Chess.com, it has the ability to run at different playing strengths and with different styles and opening books, which is why it has been serving as Chess.com's main engine for its different bots when playing against computers. It selects its moves by win probability and incorporates Efficiently Updatable

Neural Networks to achieve deeper positional understanding possessed by neural network engines such as AlphaZero and Leela Chess Zero.

3.2 Research Work

3.2.1 Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead

This paper presents a novel approach for training Artificial Neural Networks (ANNs) to evaluate chess positions, aiming to mimic the pattern recognition skills of highly rated chess players. It uses about 3 million chess games played by skilled players, labeled with Stockfish chess engine evaluations. The study compares Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) across different datasets and board representations. Findings show MLPs generally outperform CNNs in this context. Additionally, representing chess pieces with numerical values, based on their strength, negatively impacted ANNs' performance in most experiments. The paper concludes that MLPs are more effective than CNNs for learning chess patterns, and deep lookahead in chess can be compensated with significant chess knowledge (Klein).

3.3 DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess

This paper introduces an innovative approach to chess by leveraging neural networks. The architecture involves a deep autoencoder for feature extraction and a artificial neural networks for position comparison. DeepChess is trained to predict the winning position without explicit knowledge of chess rules such as pawn promotions, castling, and illegal moves, which shows the power of such deep training, leveraging the millions of games it was trained on. (Omid E. David & Wolf)

4 Data Processing

4.1 Data Sources

The chess game data was obtained from the LiChess.com database in Portable Game Notation (PGN) format (see figure 2) (Lichess.org). This data includes game information about all the moves played in each game as well as the location, date, tournament (if applicable) and outcome of the game. We also utilized an open-source library called python-chess to help us in the conversion of the PGN data into Forsyth-Edwards Notation (FEN) format (Python-Chess). FEN is a standardized way of representing a single chess position from a chess game. It includes the information of the current positions of the chess pieces on the board, the active color (white or black's move), as well as other information pertaining to certain rules of the game such as castling, En Passant, half-move clock and the full move number (Python-Chess).

```
[Event "Live Chess"]
[Site "Chess.com"]
[Date "2020.03.25"]
[Round "-"]
[White "pdrpnht"]
[Black "ColinStapczynski"]
[Result "0-1"]
[WhiteElo "1543"]
[BlackElo "2241"]
[TimeControl "180"]
[Termination "ColinStapczynski won by resignation"]

1. e4 d5 2. exd5 Qxd5 3. Nc3 Qa5 4. d4 c6 5. Nf3 Bf5 6. Bd3 Bxd3 7. Qxd3 e6 8.
O-O Nf6 9. Bg5 Nbd7 10. Ne5 Qc7 11. Ne2 Nxe5 12. dxe5 Qxe5 13. Bxf6 gxf6 14.
Rfe1 Bd6 15. Ng3 Qd5 16. Rad1 Qxd3 17. Rxd3 O-O-O 18. Red1 Be7 19. Ne4 Rxd3 20.
Rxd3 Rd8 21. Rh3 f5 0-1
```

Figure 2: Portable Game Notation (PGN) data example

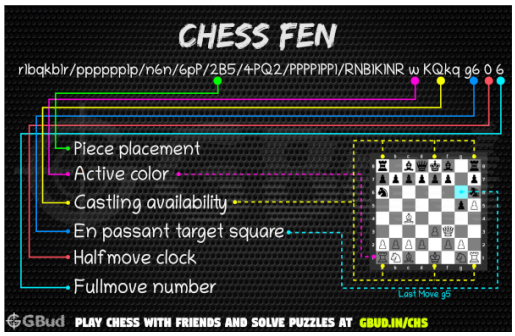


Figure 3: Forsyth-Edwards Notation example

4.2 Data Cleaning and Formatting

Before proceeding, we must filter the games included in the downloaded PGN file. We loop through all the games, and extract the ones that have moves that include Stockfish evaluation ratings, which only made up about 6% of every dataset we got. Of those games, we only select the ones in which a player has an ELO (player) rating of above 2000, which is considered an excellent rating in chess.

Since the data must be fed into an ANN, it must be represented as a numerical array. As the purpose of the ANN is to take in a chess board position as input and output an evaluation of the position, it is necessary to obtain the board position data for each move of the chess game. However, the PGN format only specifies the move made by either player at each step of the game (i.e. PD4 means that a pawn was moved to the D4 square on the board), meaning that the board position must be extrapolated from the sequence of moves made from the start of the game. The python-chess library provides a function that can take in the sequence of game moves from the PGN and convert it into the FEN format which allows us to gain a representation of the current position of the chess game, including the location of pieces, current player and other game-related information that can be used to evaluate the current game position (see Figure 3) (Python-Chess). We then write these FENs (for each move) as well as the Stockfish evaluation previously acquired from the PGN to a CSV file. This allows us to easily share and extract the data without having to recompute it.

Then, the FEN format must be converted into a numerical array to allow it to be used as input to our neural network. We implemented the following steps to format the data into a numerical array.

- Parsing the FEN string into its components, such as the board layout, active color, castling availability, en passant target square, half-move clock, and full-move number.
- One-hot encoding the chessboard using a 3D array (8x8x12) to represent the positions of different pieces (Pawns, Knights, Bishops, Rooks, Queens, Kings) for both white and black.
  - The last dimension of the (8x8x12) array represents the number of different types of chess pieces on the board. In standard chess, there are six different types of pieces for each player (white and black), and each type is represented by a unique one-hot encoded channel. These six types are:
    1. Pawn (P or p)
    2. Knight (N or n)
    3. Bishop (B or b)
    4. Rook (R or r)
    5. Queen (Q or q)
    6. King (K or k)
- Encoding the active color ('w' = 1, 'b' = 0).
- Encoding castling availability, where each type of castling is represented by a binary value (1 for available, 0 for not available) (masterclass).
- Encoding the en passant target square as a one-hot vector. chess.com
- Flattening the one-hot encoded board and other features into a single input vector.
- Combining all the features to create the final input vector.

5 Primary Model

The model we decided to use was an ANN that takes in a numerical array (encoded from the FEN format), representing the chess board's current state, and outputs an evaluation score of this state, within the range of [-15, +15], where -15 represents a clearly winning position for the opponent, and +15 represents a dominant position for yourself.

GeeksForGeeks

5.1 Input

The input layer takes a flattened one-hot array encoded from the FEN format. The bits of this array are represented by the following components:

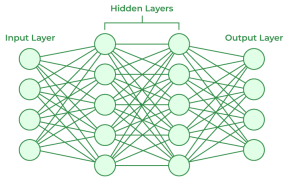


Figure 4: ANN Model

1. Chess Board (One-Hot Encoded): 768 bits (8x8 squares, 12 pieces types per square, 1 bit per piece type)
2. Active Color: 1 bit (1 for white's turn, 0 for black's)
3. Castling Availability: 4 bits (1 bit each for white king side, white queen side, black king side, black queen side castling)
4. En Passant Target Square: 64 bits (8x8 grid, 1 bit per square to indicate en passant target)
5. Halfmove Clock: 1 bit (number of halfmoves since the last capture or pawn advance)
6. Fullmove Number: 1 bit (the number of the full move, incremented after Black's move)

Total: 839 bits

5.2 Architecture

The final model represents a deep Artificial Neural Network designed for the complex task of evaluating chess positions. It consists of an input layer of 839 neurons to accommodate the one-hot encoded chessboard representation. This is followed by four hidden layers with a descending number of neurons: 512, 256, 128, and 64, respectively. Each hidden layer employs a ReLU activation function, batch normalization, and a hyper-tuned dropout rate of 0.5 to promote generalization and mitigate overfitting. The output layer is a single neuron providing the position evaluation score. The model uses Mean Squared Error Loss (MSELoss) and is optimized with the Adam optimizer, with the most optimally tuned learning rate of 0.005. The training was computed on different batch sizes, with 32 being the most suitable one for our model, a choice that balances computational efficiency and the ability to generalize from training data. The architecture, designed with multiple layers and decreasing neuron counts, is intended to capture complex patterns and relationships in the data while maintaining computational efficiency. This detailed description provides clear guidance for reproduction, ensuring similar performance can be achieved in similar settings.

6 Baseline Model

Our baseline model consists of a heuristic chess function that takes into consideration different factors, which, summed up, give us our own evaluation score to rate the 8x8 board state at different timestamps. This heuristic function consists of material evaluation based on piece values, piece mobility, control of the center of the board, etc. This evaluation is then compared to the Stockfish evaluation of the position using the Stockfish API, and the difference between the Stockfish evaluation and our heuristic evaluation is plotted to find the loss of our model (Stockfish).

The baseline model is fairly easy to implement, as it only requires basic chess knowledge with a simple strategic understanding of how to get closer to checkmating your opponent. It also needs minimal tuning, which can easily be made by adding or removing any new functionality to the existing evaluation algorithm.

6.1 Implementation

Our heuristic function takes a FEN (Forsyth-Edwards Notation) string as input, which represents a particular chessboard position. It then creates a chess board from this FEN string using the python-

chess library (Python-Chess). The function proceeds to evaluate the position using various heuristics we found essential.

First, it assigns values to different chess pieces, such as pawns, knights, bishops, rooks, and queens, and calculates the material evaluation. Material evaluation is a simple but crucial heuristic that computes the difference between the total piece values for both sides (white and black).

Next, it considers piece mobility by calculating the number of legal moves each side can make. This mobility evaluation is the difference between the mobility of white and black pieces.

Finally, the function evaluates control of the center, a key area of the chessboard. It counts the number of pieces from each side that influence central squares and computes the difference.

The total evaluation combines these three components: material evaluation, mobility evaluation, and center control evaluation. These values contribute to an overall assessment of the chess position, helping to quantify its quality for each side. The relative weight of each component can be adjusted to fine-tune the evaluation based on the specific characteristics of the heuristic

6.2 Performance

Our heuristic function performed as expected, taking into account the different functionalities that were implemented in the heuristic algorithm. Compared to Stockfish, which uses advanced algorithms that are very computationally heavy such as minimax and alpha-beta pruning for search optimization, our heuristic model is vastly outperformed. Stockfish uses various extensions and optimizations to explore the game tree and calculate these outputs efficiently, while continually updating its evaluation as it explores deeper into the tree, enabling it to analyze positions deeply and accurately (Stockfish).
The testing set was taken from lichess.com, holding billions of games played since the opening of the website (Lichess.org). A small extraction of a few thousand games were taken, each consisting of approximately 30 moves each. Two main plots were graphed (figure 4 & figure 5). The first plot covering a few data points from both functions, and the second plot being the difference in evaluation between both Stockfish and our heuristic, also taken on a couple of data points for better visualization.
By plotting, we can see a big difference in our numerical ratings. When comparing our primary model with Stockfish, we should be able to see a linear graph with a few exceptions, but not as much variance as the difference in evaluation we just computed, proving that our primary model outperforms our baseline model.

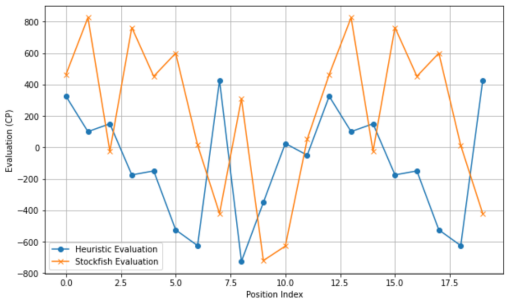


Figure 5: Plots of both the heuristic and stockfish evaluations

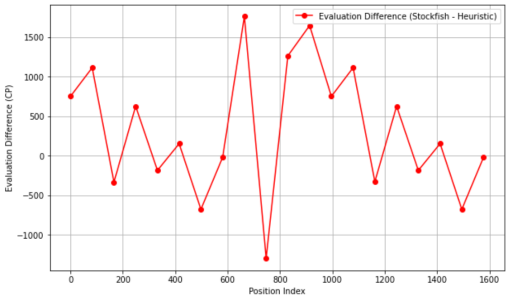


Figure 6: Evaluation difference between stockfish and heuristic function

7 Quantitative Results

Before training, we normalized all evaluation scores to be between -15 and +15, in order to limit the presence of outliers in the data.

During training, our model's performance was evaluated through its training and validation losses. The training loss was recorded at 0.928, indicating that the model learned well from the training

dataset. The validation loss had a value of around 2.0 as seen in Figure 7, which reflects the model's ability to generalize its learning to new data.

The observed discrepancy between the training loss and the higher validation loss can be attributed to the limited scope of our training data, caused by our lack of computational power to process more than 1.2 million positions. Training on a constrained dataset due to the lack of GPU power, especially in a domain as complex as chess, can significantly impact the model's ability to generalize. When a model is trained on a limited dataset, it often learns patterns and correlations specific to that dataset (overfitting), but it may not capture the broader range of variations and scenarios found in the larger population.

Due to this stagnation of the validation loss, we decided to only train for 20 epochs. Training for more epochs did not show any significant improvement with our loss values.

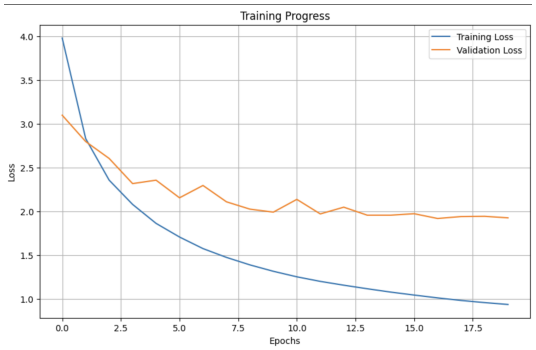


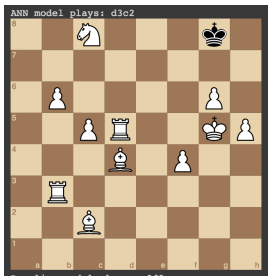
Figure 7: Training vs Validation Loss

8 Qualitative Results

The model is able to play real-time games against both human players as well as artificial players. The model takes in the current position of the board and produces an evaluation score for each possible valid move. It then chooses the move with the highest score and replicates the move on the chess board. We found interesting results which have been reflected through model's interaction in various different games.

The model performs very well against our baseline model. It is able to capture a majority of the pieces the baseline blunders (see Figure 8a) and is clearly outplaying it throughout the game by gaining a strategic advantage. This was expected since the baseline uses simple heuristics while the ANN model is able to learn more intricate features allowing it to make more strategic moves.

Another result which can be highlighted is the model's performance against a 2000 rated AI on Chess.com. This result shows its shortcomings due to the fact that it was not trained extensively on large amounts of data. Given the complexity of chess, and the fact that there are 10^{40} legal possible chess positions (liverpoolmuseum). It is expected to not be able to outperform a player with high ratings. Fewer data points to train on means that there are fewer samples of positions that the network has seen before, making it harder to evaluate unseen positions. Through games with Chess.com' AI trained bot, we can view that the model was playing farily well, but in some positions it has not been able to recognize and handle indirect threats to its king, such as a possible checkmate patterns and positional disadvantages in long-term play, reflected in figure 8b.



(a) Model(white) Vs Baseline(black)

(b) Chess.com AI (white) Vs Model(black)

Figure 8: Comparative Analysis

9 Evaluation On New Data

A new dataset dating back from August 2023 was extracted from lichess.com and used for our testing purposes. We took a date close to the date of the original dataset that we have trained our model on so that the Stockfish evaluation algorithms have not been drastically changed to reflect different evaluation scores on similar positions. We also made sure that the dataset only included games used within August, so that none of these games were showing up in the training data. The data has then been processed and cleaned to make sure that any possible real game can be tested whatever ELO the players were, and these chess positions were then used as our model's input to evaluate our neural network's implementation on unseen data.

In the performance evaluation of our neural network model, we utilized 2 key metrics: Mean Squared Error (MSE), and Mean Absolute Error (MAE), each chosen for their relevance to our specific task and dataset.

We selected MSE, which was recorded at 4.51, due to its effectiveness in highlighting larger errors in evaluation predictions. This metric is particularly relevant to our model, as it emphasizes significant deviations from the actual Stockfish evaluations we anticipate. Our MSE value, which is on the higher-end, reflects the lack of data points.Our training model was composed of 1.2 million chess moves, compared to other well-established chess engines which have trained their networks on hundreds of millions of moves, if not more. This insight pointed out the need for a larger dataset to improve our model's accuracy, with less memorization.

Additionally, the MAE was recorded at 1.28, was chosen as it provides a straightforward measure of average evaluation prediction error, which is less influenced by extreme evaluations, such as checkmating patterns, compared to MSE. This metric was useful as it reflects the average extent to which our predictions differ from the actual evaluations. The moderate MAE value suggests that our model's predictions, on average, are reasonably close to the true evaluations but still indicate the necessity for exposure to a larger variety of chess positions.

10 Results Discussion

The results we got from our model exceeded our expectations, taking into consideration the limited amount of data we were able to process on our own machines' personal GPUs, given the computational limits of the free version of Google Colab. Building a strong chess engine requires substantial computational resources and extensive amounts of data, and with about 1.2 million positions, our model was able to converge with good test and validation accuracy, alongside adequate testing values representing our model's ability to generalize well.

10.1 Interpretation

Taking into consideration that we only took chess games rated over 2000 ELO for black or white, which is considered above the 97th percentile in chess, only 1516 members are rated above this rating system, which makes it very hard to gather any bigger datasets (uschess). Games of lower ELO were not considered, as in general, below this threshold, players would usually be making

their move decisions based on intuition, with no appropriate understanding of chess openings, middle game decision-making, and end game strategies, which we wanted our potential model to base its suggested moves on. Additionally, the number of possible positions in chess is astronomically high, with the number of possible variations of chess games reaching 10^{40} legal moves (liverpool-museum). With the limited amount of data we were able to process, we expected our model to have sub-optimal results.

10.2 Learnings

The results of this project showed us how neural networks attempt to solve such complex games. What was very surprising was the fact that most of the moves made by our computer felt human-like. It demonstrated this in multiple games, with the model waiting for the opponent to make a wrong move to gain a positional advantage, or playing tempting moves that Stockfish did not consider as the most optimal. This is mainly due to the fact that our model was not created by training it to play against itself, which is considered a reinforcement learning approach. Building such model which was able to recognize patterns within opening sequences, identify tactical motifs, and understand strategic principles proved the immense capabilities of such networks.

11 Ethical Considerations

For any project that includes the need to collect vast amount of data, there is always the possibility of ethical concerns rising surrounding the privacy and consent for that data. In the case of chess, it is likely that the data from players that is available on online chess platforms and sources has been collected without the players privacy or consent. The team will ensure to only use the data from the game and not use any private or personal data to ensure there are no ethical violations within this project. In addition, there are also ethical considerations when it comes to the use of the chess engine. There is the possibility of players using the deep learning model to their own advantage to get an edge over their opponents during online chess matches. This can cause disruptions in the online chess community and competitive events due to increased unethical game play. Therefore, to mitigate this risk, the chess engine can be restricted from being used in specific contexts to ensure responsible use.

By addressing these ethical considerations, we aim to ensure the responsible and ethical application of our chess engine.

12 Project Difficulty/Quality

This project proved to be as difficult as expected. This was mainly due to the overall complexity of chess itself. To start, chess has an estimated 10^{40} possible moves (liverpoolmuseum). This posed a significant road block in the training state, since the team lacked the necessary resources such as computational power to account for as many moves as our network needed. Even though chess data was widely available, training a considerable amount of it to improve our model was not feasible, as our laptops kept crashing when larger than usual models were trying to train. Moreover, not being able to train on abundant data meant that a lot of playing styles were not taken into account by the model, meaning there are still many scenarios the model hasn't seen and most likely won't be able to effectively predict the best move for, further emphasizing the significance of being able to train on large data sets. This resulted in subpar results, and could have definitely been mitigated if powerful enough hardware was available or clusters were shared. Chess is a game that necessitates a lot of strategic thinking and requires planning moves ahead, this complex way of thinking is something neural networks struggle with, since the model outputs what it thinks the best move is at the given time, and does not take future moves into consideration. This means that based on the current implementation, it is very hard to design something that can incorporate human level intuition. Overall, we made sure that the quality of the project remained intact, and were very impressed with the results given the limitations that were encountered. Our model far exceeded what we expected it to learn, train, and apply, and the results we got were proof of that. A lot of outside learning was required, which shows our team's persistence and willingness to learn and apply not only what has been learned in class, but all the requirements of building such a complex model from scratch.

REFERENCES

chess.com. Chess.com - en-passant. https://www.chess.com/terms/en-passant//. Accessed: 2023-12-01.

GeeksForGeeks. Geeksforgeeks. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications//. Accessed: 2023-11-03.

IBM. Deep blue defeats garry kasparov in chess match. https://www.ibm.com/ibm/history/exhibits/vintage/vintage_4506VV1001.html. Accessed: 2023-12-01.

Dominik Klein. Neural networks for chess - the magic of deep and reinforcement learning revealed. https://arxiv.org/ftp/arxiv/papers/2209/2209.01506.pdf//. Accessed: 2023-12-01.

Lichess.org. Lichess api. https://lichess.org/api. Accessed: 2023-11-03.

liverpoolmuseum. liverpoolmuseum. https://www.liverpoolmuseums.org.uk/stories/which-greater-number-of-atoms-universe-or-number-of-chess-moves/. Accessed: 2023-12-01.

masterclass. Chess 101: What is castling? learn about the 2 conditions that need to be satisfied in chess before you can castle. https://www.masterclass.com/articles/chess-101-what-is-castling-learn-about-the-2-conditions-that-need-to-be-satisfied-. Accessed: 2023-12-01.

Nathan S. Netanyahu Omid E. David and Lior Wolf. Deepchess: End-to-end deep neural network for automatic learning in chess. http://www.cs.tau.ac.il/~wolf/papers/deepchess.pdf//. Accessed: 2023-12-01.

Python-Chess. Python-chess. https://pypi.org/project/chess/. Accessed: 2023-11-03.

Stockfish. Stockfish. https://stockfishchess.org/. Accessed: 2023-11-03.

uschess. uschess. https://www.uschess.org/archive/ratings/ratedist.php//. Accessed: 2023-12-01.

wikipedia. Wikipedia - stockfish (chess). https://en.wikipedia.org/wiki/Stockfish_%28chess%29#Competition_results//. Accessed: 2023-12-01.