Human-Chessbot: Training a Neural Network to Play Chess Like a Human

Gee Stott

Abstract— This report presents the development and evaluation of Human-Chessbot, a neural network-based chess engine designed to play chess in a human-like manner. While conventional chess engines have achieved superhuman performance by prioritizing optimal play, their non-human-like strategies limit their effectiveness as training tools for human players. This project addresses this gap by creating an AI that mimics human decision-making, including plausible errors and stylistic nuances. By training a neural network on a large dataset of human games from Lichess, our model learns to predict the moves of human players at various skill levels. This work builds upon the MAIA project but aims to create a more computationally efficient model, comparable in size to Stockfish, making it accessible on standard hardware. The significance of this research lies in the exploration of human-aligned AI, with potential applications in creating more effective and engaging AI-based training systems for a variety of domains.

***Index Terms*—** Chess, Neural Networks, Machine Learning, Human-like AI, Deep Learning

Introduction

Since the 1940s, chess-playing programs, or “chessbots,” have evolved to surpass the capabilities of the most skilled human players. These bots are now integral to modern chess training, with players globally using them to refine their skills. However, a significant disparity exists between the strategies employed by traditional chessbots and human players. Human chess is characterized by the use of heuristics, intuition, and pattern recognition. In contrast, conventional chessbots rely on brute-force computation, analyzing vast numbers of potential future board states to select the move that maximizes their probability of winning.

This computational superiority creates an imbalance in human-computer matches, where the bot’s expansive memory and processing power provide a decisive advantage. To mitigate this, lower-level bots are often programmed to introduce deliberate errors, resulting in an unnatural and often confusing experience for the human player. This presents a paradox: while chessbots dominate the game, their non-human-like mode of play limits their effectiveness as a learning tool for humans.

This project addresses the challenge of creating a more “human-like” chessbot. Our work builds upon the research of McIlroy-Young et al. on the MAIA models, which were trained using supervised learning to emulate human decision-making, a departure from the reinforcement learning approach common to engines like AlphaZero. While MAIA demonstrated strong performance in predicting human moves, its reliance on the large and resource-intensive AlphaZero architecture limits its practical application. Conversely, Stockfish, a significantly smaller engine, achieves comparable performance to AlphaZero, indicating that model size does not scale linearly with performance.

The primary objective of this project is to develop a chessbot that not only plays like a human but is also computationally efficient enough to run on standard consumer hardware. We aim to achieve a level of human-like play comparable to the MAIA models but with a model deployable to edge devices. To this end, we will utilize the Lichess Dataset, a comprehensive collection of human chess games and player ratings, which is the same dataset employed in the training of MAIA. This work contributes to the growing field of human-aligned AI by exploring the trade-offs between model size, performance, and human-like behavior in the domain of chess.

Background and Related Work

II.A. Chess Engines

While early chess engines relied on traditional search algorithms like minimax and alpha-beta pruning, the field has been transformed by neural network approaches. These early engines, including Deep Blue, used handcrafted evaluation functions that required extensive manual tuning of positional and material factors.

The introduction of deep neural networks marked a paradigm shift in computer chess. Modern engines employ sophisticated architectures: AlphaZero uses a deep residual network with 20 residual blocks, each containing convolutional layers and batch normalization. This network processes the 8x8x73 board representation to output both a position evaluation and a policy distribution over possible moves. Leela Chess Zero (LC0), an open-source implementation of similar principles, utilizes a network with 24 residual blocks and generates 256 channels in its intermediate layers.

MAIA, specifically designed for human-like play, employs a modified version of LC0′s architecture. Instead of learning through self-play, MAIA is trained on millions of human games using supervised learning. Its network consists of multiple convolutional layers followed by a policy head that predicts move probabilities and a value head that evaluates positions. MAIA processes the board state through 12 input planes (6 piece types × 2 colors) and incorporates additional features like player ratings. Recent hybrid approaches, such as Stockfish's NNUE (Efficiently Updatable Neural Network), combine traditional search with a smaller neural network containing roughly 40,000 parameters that can be efficiently updated during search.

II.B. Human-Like Chess AI

While the pursuit of optimal play has been the primary focus of chess engine development, there is a growing interest in creating AI that plays in a more human-like manner. The MAIA project, by McIlroy-Young et al., represents a significant step in this direction. Instead of using reinforcement learning to find the best possible move, MAIA is trained on a large dataset of human games to predict the move a human player would make in a given position. This supervised learning approach results in an engine that exhibits more human-like characteristics, including making mistakes that are typical of human players at a certain skill level instead of the optimal move. This leads to a much more human learning experience instead of the current method of leveling the playing field by purposefully choosing non-optimal moves.

II.C. Dataset and Training Data

The Lichess database is a massive, publicly available collection of chess games played on the Lichess.org platform. It contains billions of games played by humans of all skill levels, making it an invaluable resource for training human-like chess models. The games are stored in the Portable Game Notation (PGN) format, which includes the moves of the game, the players' ratings (Elo), and the game's outcome.

Methodology

III.A. System Architecture

We primarily focused on three separate aspects of the human chess bot. Firstly we converted and processed the data into usable chunks, then we trained the model before finally deploying the model into a playable environment.

III.B. Data Collection and Processing

III.B.1. PGN File Processing

Lichess data is stored in the form of PGN files. PGN or Portable Game Notation is made up of 2 parts the tag pairs, which stores game metadata about the game and players, and the move strings of the game stored in UCI.

To make this data usable we stored the players elos, which is a measure of their skill. Then we iterate through the game storing "snapshots" of the board at every state, and the move taken by the current player at that position. The board states are then represented as 3 dimensional tensors that are 8 by 8 (due to the size of the board) with 12 channels with 6 channels for each of the white pieces and 6 for the black.

III.C. Dataset Statistics

We trained our model off of 823,000 games from January to May of 2013. These games were played by a wide breadth of players in a variety of states. The original maia model was split into 9 separate models split apart by elo ranges of 100 (e.g. 1100-1200) this limited the models generalizability and so we trained our model on one complete dataset to increase access. Exanding from 1100-1900 to 700-2500 to further increase access.

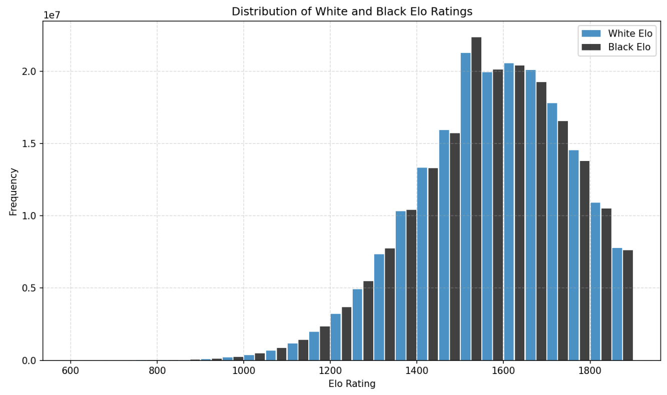

Figure 1: Spread of the elos present in the training set.

One other key difference is the fact that both iterations of the maia paper are unable to play chess openings as they don't have any knowledge of the first 10 moves. This is a profound limitation as the opening is one of the key areas where players struggle within a game. So knowing this we choose to include the first 10 moves to increase the understanding of openings.

We also didn't limit the model to just the blitz games as we wanted to increase the overall variability in the games that the model was capable of representing. We introduced this noise as it gives a far more holistic view of human players. This was useful as blitz games only show up a small portion of the data and we want to show all human play.

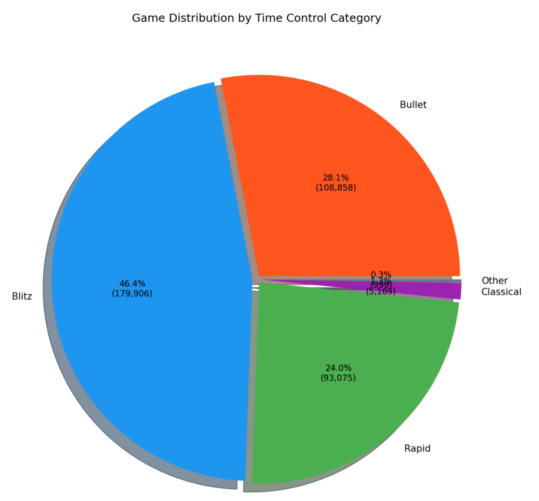

Figure 2: Time Control Splits of the games from Jan-May of 2013

III.D. Neural Network Architecture

The neural network architecture consists of three main components: a convolutional backbone for processing board states, a fully connected network for feature extraction, and dual output heads for move prediction and auxiliary tasks.

The convolutional backbone processes the 8x8x12 board representation through six convolutional layers. Each layer uses 64 filters with 8x8 kernels, maintaining the spatial dimensions through "same" padding. ReLU activation functions are applied after each convolution to introduce non-linearity.

The fully connected portion begins with a 4100-dimensional input (flattened board features plus metadata) and progressively reduces dimensionality through six layers ($512 \rightarrow 32 \rightarrow 32 \rightarrow 32 \rightarrow 32 \rightarrow 32$) with ReLU activations. This creates a compact 32-dimensional representation of the game state.

The network splits into two parallel output heads:

1. Move prediction head: Maps the 32-dimensional state to 2104 move probabilities
2. Auxiliary prediction head: Produces additional chess-relevant predictions

Input representation:
- Board state: 8x8x12 tensor (6 piece types × 2 colors)
- Metadata: Player ratings and game state information
- Combined input: 4100 dimensions (4096 from board + 4 metadata features)

Output representation:
- 2104-dimensional probability distribution over legal moves
- Softmax activation ensures valid probability distribution
- Auxiliary head provides additional game state predictions to augment learning process

The architecture balances computational efficiency with sufficient complexity to capture chess patterns and human play styles. The dual head design allows simultaneous learning of move prediction and auxiliary chess concepts.

III.E. Training Process

III.E.1. Implementation Details

The training implementation utilizes PyTorch's ecosystem for deep learning. The training pipeline is encapsulated in a Trainer class that handles data loading, model training, evaluation, and checkpointing. Key components include:

- Data loading with PyTorch DataLoaders for efficient batch processing
- Adam optimizer with configurable learning rate and weight decay
- Cross-entropy loss for both move prediction and valid moves
- Automatic checkpointing and performance logging
- GPU acceleration when available

III.E.2. Training Parameters

The model was trained with the following configuration:
- Batch size: 512
- Initial learning rate: 0.001
- Weight decay: 1e-4
- Beta parameters: (0.9, 0.999)
- Number of epochs: 50
- Data split: 80% training, 10% validation, 10% test

III.E.3. Optimization Strategy

We employed random search for hyperparameter optimization, exploring combinations of:
- Learning rates: [0.1, 0.01, 0.001, 0.0001]
- Weight decay rates: [1e-3, 1e-4, 1e-5]
- Beta values: [0.9, 0.95, 0.99]
- Momentum values: [0.9, 0.95, 0.99]

The best performing configuration was selected based on validation loss.

III.E.4. Hardware Configuration

Training was conducted on embedded AI hardware:

- Platform: NVIDIA Jetson Orin Nano 8GB Developer Kit
- GPU: 1024-core NVIDIA Ampere architecture GPU
- CPU: 6-core Arm Cortex-A78AE v8.2 64-bit CPU
- RAM: 8GB 128-bit LPDDR5
- Storage: 64GB eMMC 5.1

This setup processed approximately 10,000 positions per second during training.

Implementation

IV.A. Convert Package

IV.A.1. PGN to CSV Conversion

IV.A.2. Combining PGN Files

IV.B. Train Package

IV.B.1. Dataset Module

IV.B.2. Training Module

IV.C. Play Package

IV.C.1. Game Engine Integration

IV.C.2. Player Implementations

IV.C.3. User Interfaces

IV.D. Testing and Quality Assurance

Results and Evaluation

V.A. Training Results

V.A.1. Training Metrics

V.A.2. Model Performance

V.B. Playing Strength Evaluation

V.B.1. Games Against Baseline Engines

V.B.2. Elo Rating Estimation

V.C. Human-Likeness Evaluation

V.D. Example Games

Discussion

VI.A. Key Findings

VI.B. Challenges and Limitations

VI.B.1. Technical Challenges

VI.B.2. Model Limitations

VI.C. Comparison to Project Goals

VI.D. Lessons Learned

VI.E. Broader Implications

Conclusion and Future Work

VII.A. Summary

VII.B. Future Work

VII.B.1. Model Improvements

VII.B.2. Training Enhancements

VII.B.3. Feature Additions

VII.B.4. Evaluation Extensions

VII.C. Concluding Remarks

References

References

[1] R. McIlroy-Young, S. Sen, J. Kleinberg, and A. Anderson, “Aligning Superhuman AI with Human Behavior: Chess as a Model System,” in Proc. 26th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), 2020.

[2] D. Silver et al., “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, pp. 484–489, 2016.

[3] D. Silver et al., “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” Science, vol. 362, no. 6419, pp. 1140–1144, 2018.

[4] Z. Tang, D. Jiao, R. McIlroy-Young, J. Kleinberg, S. Sen, and A. Anderson, “Maia-2: A Unified Model for Human-AI Alignment in Chess,” in Proc. 38th Conf. on Neural Information Processing Systems (NeurIPS), 2024.

[5] R. McIlroy-Young, S. Sen, J. Kleinberg, and A. Anderson, “Aligning Superhuman AI with Human Behavior: Chess as a Model System,” in Proc. 26th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), 2020, pp. 1667–1677.

[6] T. McGrath et al., “Acquiring human-like concepts from a superhuman AI,” arXiv:2210.13432, 2022.

[7] R. McIlroy-Young, R. Wang, A. Anderson, and J. Kleinberg, “Detecting Individual Decision-Making Style: Exploring Behavioral Stylometry in Chess,” in Proc. 35th Conf. on Neural Information Processing Systems (NeurIPS), 2021.

[8] A. Paszke et al., “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[9] N. Moskopp, “python-chess: a chess library for Python,” GitHub repository, 2014. [Online]. Available: https://github.com/niklasf/python-chess

Appendices

X.A. Appendix A: Source Code Repository

All of the source code for this project is a available at https://github.com/EthanDGee/ryleeeeeeeeeeee

X.B. Appendix B: Dataset Details

The Lichess dataset used in this project consists of over 1 billion chess games played on the Lichess.org platform between 2013-2023. Key characteristics:

- Total size: 2TB of compressed game data
- Game format: Portable Game Notation (PGN)
- Rating range: 800-2900 ELO
- Game types: Classical, Rapid, Blitz, and Bullet time controls
- Key fields per game: **Player ELO ratings** Move sequences in UCI format **Game result** Time control **Opening classification** Timestamps

For training, we took data from January to May of 2013 consisting of 860,000 games from players rated 797-2412 ELO to focus on typical club-level play patterns. Games were validated to remove incomplete or corrupted entries.

Data processing pipeline:
1. PGN parsing and validation
2. Feature extraction (board states, moves, metadata)
3. Train/validation/test splitting (80/10/10)
4. Normalization of numerical features

X.C. Appendix C: Additional Results