Pairwise Learning to Rank for Chess Puzzle Difficulty Prediction

Andry Rafaralahy
andry.rafa2@gmail.com

***Abstract*—Machine learning has shown great success in various aspects of chess, particularly in game-playing engines such as AlphaZero. However, predicting the difficulty of chess puzzles remains a relatively unexplored area. In the IEEE BigData 2024 Cup: Predicting Chess Puzzle Difficulty competition, participants are asked to build a machine learning approach to predict the difficulty of chess puzzles. We present an approach that leverages deep learning and pairwise learning-to-rank techniques to estimate the difficulty of chess puzzles. Our method applies pairwise learning to rank approaches to simulate games between puzzles and uses the outcomes to estimate their Glicko-2 ratings.**

***Index Terms*—chess, deep learning, learning to rank, glicko-2**

I. Introduction

Accurately predicting the difficulty of chess puzzles is a relatively unexplored task. To address this challenge, we propose an approach that integrates deep learning architectures and pairwise learning to rank systems. We leverage deep learning to extract positional features and we using the RankNet architecture to learn a pairwise ranking function. This ranking function is then used to simulate games between puzzles, which outcomes are used to estimate puzzle rating using the Glicko-2 model.

The organization of this paper is as follows: in Section II, we review related work in the fields of deep learning applied to chess and pairwise learning to rank models. In Section III we provide a brief description of the competition. Our data preprocessing and feature engineering steps are detailed in Section IV. In Section V, we describe the model architecture along with the training and inference procedures. The experimental results are presented in Section VI, followed by the concluding remarks in Section VII.

II. Related Work

A. Deep Learning in Chess

The application of deep learning to chess has seen significant success, particularly with the introduction of *AlphaZero*, which uses a combination of convolutional neural networks (CNNs) and Monte Carlo Tree Search (MCTS) to surpass traditional chess engines [1].

Building on AlphaZero, the open-source project *LeelaChessZero* (LCZero) initially followed the same CNN-based approach but later transitioned to a *Transformer* architecture to improve performance [2]. Transformers are particularly good when training on large-scale datasets, making them a natural fit for chess, where datasets containing millions of games are available. These models, with their ability to extract meaningful features from raw board positions, serve as a foundation for applying deep learning to the puzzle difficulty prediction task.

B. Ranking and Learning-to-Rank Models

Lichess puzzles are rated according to the *Glicko-2* [3] model, an improvement over the traditional ELO rating system, which incorporates rating uncertainty and allows for dynamic adjustments based on player or puzzle activity. This system is particularly effective for environments with a high variance in participation levels, making it well suited for the task of ranking chess puzzles.

Ranking and learning-to-rank methods have been widely applied to problems where relative comparisons between items are more useful than absolute values. One of the foundational models in this area is *RankNet* [4], which uses a pairwise ranking approach to learn the relative ranking of items through gradient descent. *DirectRanker* [5], a generalization of RankNet, further improves upon this by introducing more flexibility in the model, and using a more modern optimization algorithm. In the context of chess puzzles, these pairwise ranking models are particularly relevant, as the difficulty of one puzzle is often better understood relative to another.

III. Competition Description

A. Task

The goal of this competition was to estimate the difficulty of chess puzzles using the starting position and the solution of the puzzle. The difficulty of a puzzle is quantified by its Glicko-2 [3] rating, as determined by user attempts to solve these puzzles on the Lichess platform.

B. Data

The data provided for this competition consist of roughly 4 millions puzzles from the Lichess database. The dataset includes:

- The starting position of the puzzle, in Forsyth–Edwards Notation (FEN).
- The solution of the puzzle, which consists of the the move leading to the puzzle position followed by the optimal moves for the puzzle taker alternated with the simulated opponent, in Portable Game Notation (PGN).
- The rating of the puzzle.
- The rating deviation of the puzzle, estimating the puzzle's rating uncertainty.
- Additional data such as the number of time the puzzle was played, its popularity, tags, etc.

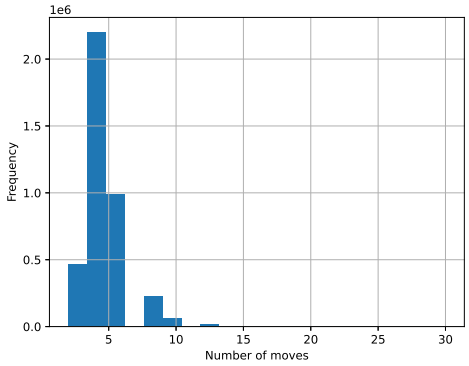

Fig. 1. Distribution of the number of moves in the puzzle solutions

C. Evaluation

The quality of the solutions was evaluated using the Mean Squared Error (MSE). A preliminary score was computed on a small subset of the test records, fixed for all participants. The final evaluation was performed after the completion of the competition using the remaining part of the test records.

IV. Data Preprocessing

We represent the model inputs as a set of 8×8 matrices, called planes, that we stack together, taking inspiration from LeelaChessZero [2]. The first dimension of these planes corresponds to the rows of the chessboard and the second dimension represents to the columns. For each puzzle, $n = 12$ positions are considered. These positions include both the initial puzzle setup and the positions reached after applying the solution moves. The choice of $n = 12$ positions is justified by the observation that 99.7% of the puzzles have a solution with a number of moves of 12 or fewer, as shown in Figure 1. For each position, we generate a total of 52 planes, that encodes information about the position and the legal moves. For puzzles that have less than 12 moves, we apply zero-padding.

TABLE I
Piece types indices

Piece	Index
Pawn	0
Knight	1
Bishop	2
Rook	3
Queen	4
King	5

Following [2], we use the following planes to encode the piece locations and castling rights on the chessboard:

- **Position Planes** These encode the types and locations of the pieces on the board. For each position, they form a $12 \times 8 \times 8$ matrix P_{ijk}. The first dimension corresponds to the type and color of the pieces, with indexing defined as follows: we assign number 0 to White color and 1 to Black color; piece types are indexed according to I; the index of a piece of color c and of piece type p is then defined as $6c + p$. The last two dimensions represent the position on the chessboard.

$$P_{ijk} = 1 \text{ iff piece } i \text{ is at position } (j, k)$$

- **Castling Planes** These encode castling rights for the position. This is a $4 \times 8 \times 8$ matrix A_{ijk}, where:
  - $A_{0jk} = 1$ if White has kingside castling rights, 0 otherwise.
  - $A_{1jk} = 1$ if White has queenside castling rights, 0 otherwise.
  - $A_{2jk} = 1$ if Black has kingside castling rights, 0 otherwise.
  - $A_{3jk} = 1$ if Black has queenside castling rights, 0 otherwise.

We also use additional planes to represent the legal moves available:

- **Move Planes** These represent legal moves for each piece. The move planes are represented as a $12 \times 8 \times 8$ matrix M_{ijk}, where $M_{ijk} = 1$ if piece i can move to square (j, k).
- **Capture and Check Planes** These are similar to the move planes but indicate if a move is a capture or results in check.

These planes are stacked together, resulting in an input matrix of size $52n \times 8 \times 8$. In an attempt to ensure symmetry between White and Black puzzle takers, we apply additional processing when the puzzle taker is Black. However, this was poorly implemented in the code and did not work as expected.

V. Model

A. Glicko-2 Ranking System

On Lichess, puzzle ratings are estimated using the Glicko-2 rating system. Players and puzzles are assigned a rating r, a rating deviation RD and a rating volatility σ. They are initalized as $r = 1500$, $RD = 350$ and $\sigma = 0.09$. The interaction between a player and a puzzle is treated as a game between the two. The outcome of the game—whether the player successfully solves the puzzle—is used to update their rating, rating deviation, and volatility. Rating updates can be done in batches of matches called rating periods [1]. The rating process during a rating period follows these steps:

1) **Initial Setup:** Each player (or puzzle) starts with an initial rating r, a rating deviation RD, and a volatility σ:
$$r = 1500, \quad RD = 350, \quad \sigma = 0.06$$
2) **Convert the rating and rating deviation to the Glicko-2 scale:** To perform the rating calculations on the Glicko-2 scale, the ratings r and RD are converted using the following formulas:
$$\mu = \frac{r - 1500}{173.7178}, \quad \phi = \frac{RD}{173.7178}$$

[1]On Lichess, rating periods include only one match, which means puzzles are updated after every puzzle interaction.

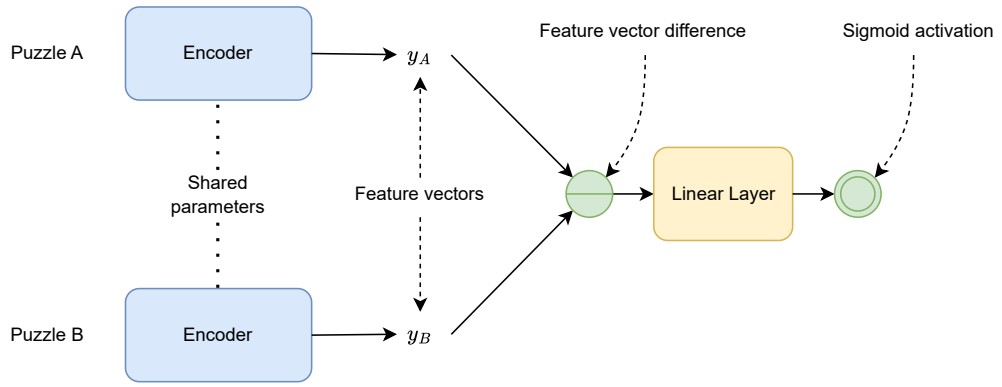

Fig. 2. Diagram of the model architecture.

The volatility σ remains unchanged during this step.

3) **Compute the Variance v:** The variance v represents the uncertainty in the player's rating based on the outcomes of games played during the rating period. It is computed as:

$$v = \left[\sum_{j=1}^{m} g(\phi_j)^2 E(\mu, \mu_j, \phi_j)(1 - E(\mu, \mu_j, \phi_j))\right]^{-1}$$

where $g(\phi_j)$ is a scaling factor given by:

$$g(\phi_j) = \frac{1}{\sqrt{1 + \frac{3\phi_j^2}{\pi^2}}}$$

and $E(\mu, \mu_j, \phi_j)$ is the expected outcome of the match, computed as:

$$E(\mu, \mu_j, \phi_j) = \frac{1}{1 + \exp(-g(\phi_j)(\mu - \mu_j))} \quad (1)$$

4) **Compute the Rating Change Δ:** The change in rating, denoted as Δ, represents how much the player's rating will shift based on the performance during the rating period:

$$\Delta = v \sum_{j=1}^{m} g(\phi_j)(s_j - E(\mu, \mu_j, \phi_j))$$

where s_j is the actual outcome of the game (1 for a win, 0 for a loss, 0.5 for a draw).

5) **Update the Volatility σ':** To update the volatility σ we solve for x the equation $f(x) = 0$ with:

$$f(x) = \frac{e^x(\Delta^2 - \phi^2 - v - e^x)}{2(\phi^2 + v + e^x)^2} - \frac{x - \ln(\sigma^2)}{\tau^2}$$

where τ is a constant that controls the amount of change in volatility. This equation can be solved using an iterative algorithm, such as the Illinois algorithm. Once a solution x^* is found, we set the new volatility as $\sigma' = e^{x^*/2}$.

6) **Update the Rating Deviation ϕ':** The rating deviation ϕ' is updated as:

$$\phi' = \frac{1}{\sqrt{\frac{1}{\phi^2} + \frac{1}{v}}}$$

7) **Update the Rating μ':** The player's new rating μ' is computed by adjusting the old rating μ based on the rating change Δ:

$$\mu' = \mu + \frac{\phi'^2}{v} \sum_{j=1}^{m} g(\phi_j)(s_j - E(\mu, \mu_j, \phi_j))$$

8) **Convert back to the original scale:** Finally, the new rating and rating deviation are converted back to the original Glicko scale:

$$r' = 173.7178 \cdot \mu' + 1500, \quad RD' = 173.7178 \cdot \phi'$$

Our approach centers on the idea that, since simulating games between players and puzzles is challenging, we can instead simulate games between puzzles themselves. By doing so, we can leverage the Glicko-2 model to estimate puzzle ratings using only puzzle data.

B. Model Architecture

The model is a deep neural network based on the RankNet architecture. It takes as input a pair of puzzle (A, B) and outputs an estimate of the probability that A is more difficult than B. It comprises two core parts: a puzzle encoder that extracts features from each puzzle and a ranking model that compares these features to predict a probability, as shown on Figure 2. For the encoder we consider two different architectures:

- A Convolutional Neural Network, based on LeelaChess architecture [6]. We use a smaller variant with 5 blocks and 128 filters.
- A Simple ViT model [7], [8], with a patch size of size 1×1, 6 layers, 16 attention heads and an embedding size of 1024.

The ranking network subtracts the encoded features, then applies a linear layer and a Sigmoid function to produce a probability score.

C. Training

We sample pairs of puzzles (A_i, B_i) from the training dataset. In practice, we can achieve this with the following procedure:

- We sample a batch of size 2N from the training dataset.
- We split the batch into two sub-batches of size N.
- Then we form pairs of puzzle by randomly matching puzzles of the first sub-batch to puzzles of the second sub-batch.

These puzzles have ratings r_{A_i} and r_{B_i}, rating deviations RD_{A_i} and RD_{B_i}, from which we compute scaled values μ_{A_i}, μ_{B_i}, ϕ_{A_i}, and ϕ_{B_i}. The puzzle representations x_{A_i} and x_{B_i} are fed into the encoder to obtain the feature vectors y_{A_i} and y_{B_i}. The output of the model is the estimated probability $\hat{p}_i$ that the puzzle A_i has a higher rating than the puzzle B_i.

$$\hat{p}_i = \sigma\left(W\left(y_{A_i} - y_{B_i}\right)\right)$$

where W is the final linear layer and σ is the sigmoid function. We aim to minimize the Binary Cross Entropy loss:

$$-\frac{1}{2N}\sum_{i=1}^{B} p_i \log(\hat{p}_i) + (1 - p_i)\log(1 - \hat{p}_i)$$

The target probability p^i that A_i has a higher rating than B_i is calculated using the Glicko-2 expected outcome formula, as follows.

$$p_i = \frac{1}{1 + \exp\left(-g\left(\phi_{A_i B_i}\right)\left(\mu_{A_i} - \mu_{B_i}\right)\right)}$$

where $\phi_{A_i B_i} = \sqrt{\phi_{A_i} + \phi_{B_i}}$ and $g(\phi) = \frac{1}{\sqrt{1+3\phi^2/\pi^2}}$.

D. Inference

To estimate the rating of a puzzle A, K puzzles $B_i, i = 1, \ldots, K$ are sampled from the training set and K games (A, B_i) using the model. The Glicko-2 algorithm is then applied to estimate the rating of A. The main variation from the standard Glicko-2 process is the use of continuous outcome probabilities instead of binary outcomes. It is also worth noting that while Lichess pairs players with puzzles of similar ratings, we observed that uniformly sampling puzzles B_i from the training set produced better results than sampling based on proximity in rating. In our experiments, setting $K = 4096$ and a rating period of 5 games yielded the best results.

VI. Experiments

The best models were trained for 10 epochs, with a batch size of 512, using the AdamW optimizer with a learning rate of 1×10^{-5} for the Vision Transformer variant and 1×10^{-4} for the ResNet variant. A 10,000-step warm-up schedule was applied. We also compared the models with their vanilla regression counterparts, composed of a position encoder followed by a regression head. It was observed that regression only models were overfitting the dataset really quickly, so the best results were obtained after only 2 epochs of training. The results are shown in Table II.

TABLE II
Results on the public test set

Model	MSE
Vision Transformer LTR	61381.3812
ResNet LTR	68632.2044
Vision Transformer Regression	77103.7790
ResNet Regression	76651.6022

VII. Conclusions

In this paper, we presented an approach to predicting chess puzzle difficulty by integrating deep learning and pairwise learning-to-rank techniques. Following the RankNet framework, we trained a model to simulate games between puzzles, using the outcomes of these simulated games to estimate their Glicko-2 ratings. This allowed us to efficiently model the relative difficulty of puzzles without requiring player data, instead relying on direct puzzle comparisons.

Our experimental results demonstrated that ranking-based models, especially the Vision Transformer variant, significantly outperformed traditional regression models in terms of prediction accuracy.

While the current approach yields promising results, several potential directions could further improve it. For example, incorporating additional data, such as puzzle themes, into the learning process could prove useful. Additionally, leveraging transfer learning by incorporating pretrained weights from models like LeelaChessZero could further enhance the model's capabilities.

These ideas represent just a few possible future directions, and further progress in applying deep learning to chess could open up even more possibilities for improvement. Continued advancements in deep learning architectures and techniques within the context of chess could lead to significant improvements in both performance and general applicability.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017. [Online]. Available: https://arxiv.org/abs/1712.01815

[2] T. L. Authors, "Leelachesszero." [Online]. Available: http://lczero.org/

[3] M. E. Glickman, "Example of the glicko-2 system."

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd International Conference on Machine Learning*, ser. ICML '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 89–96. [Online]. Available: https://doi.org/10.1145/1102351.1102363

[5] M. Köppel, A. Segner, M. Wagener, L. Pensel, A. Karwath, and S. Kramer, "Pairwise learning to rank by neural networks revisited: Reconstruction, theoretical analysis and practical performance," in *Machine Learning and Knowledge Discovery in Databases*, U. Brefeld, E. Fromont, A. Hotho, A. Knobbe, M. Maathuis, and C. Robardet, Eds. Cham: Springer International Publishing, 2020, pp. 237–252.

[6] T. L. Authors, "Leelachesszero." [Online]. Available: https://lczero.org/dev/backend/nn/

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[8] L. Beyer, X. Zhai, and A. Kolesnikov, "Better plain vit baselines for imagenet-1k," 2022. [Online]. Available: https://arxiv.org/abs/2205.01580