Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 837–842
DOI: 10.15439/2025F6771
ISSN 2300-5963 ACSIS, Vol. 43

The `bread emoji` Team's Submission to the 2025 FedCSIS Predicting Chess Puzzle Difficulty Challenge

Tyler Woodruff*
Amazon Project Kuiper
hhhtylerw@protonmail.com

Luke Imbing
Independent Researcher
lukeimbing@gmail.com

Marco Cognetta
Google + Institute of Science Tokyo
cognetta.marco@gmail.com

***Abstract*—We detail the `bread emoji` team's submission to the FedCSIS 2025 Predicting Chess Puzzle Difficulty Challenge. Our solution revolved around improving our submission from the previous competition by incorporating a new puzzle metadata feature and optimizing our implementation to allow for larger model ensembles and more stable training. Similar to our submission from last year, our system has two stages: learning a strong predictor for the LICHESS dataset and then rescaling the distribution using an empirically-guided post-processing step to fit it to the smaller and noisier competition dataset.**

Our submission placed second with a $\sim$3.9% gap in mean squared error (MSE) from first place [1].[1]

I. Introduction

THIS paper introduces our team's (`bread emoji`) submission to Predicting Chess Puzzle Difficulty - Second Edition, a competition hosted by FedCSIS. This was an open-to-the-public online machine learning competition that challenged teams to come up with novel solutions for predicting the difficulty (represented by the GLICKO2 rating system [2]) of a chess puzzle, given the initial board state, the winning moves, and a small set of metadata for each puzzle.

We participated in last year's iteration of this competition, placing first by a significant margin [3][4]. Our solution to this year's competition iterated on last year's solution, and we cover our updated methodologies in this paper. Overall, our solution placed second out of 71 teams, narrowly missing out on first place by $2k$ MSE ($<4\%$).

II. Competition Background

A chess puzzle is an initial board state and sequence of moves that lead to one side of the board gaining a significant material or strategic advantage over the other in a chess game. Puzzles are created primarily by algorithms that parse actual human vs. human games on online chess websites and find interesting move sequences. They are then added to a game server that serves players a continuous stream of puzzles to play against. Both the puzzle and the player have a rating, which are both updated by player wins and losses, according to the GLICKO2 rating system.

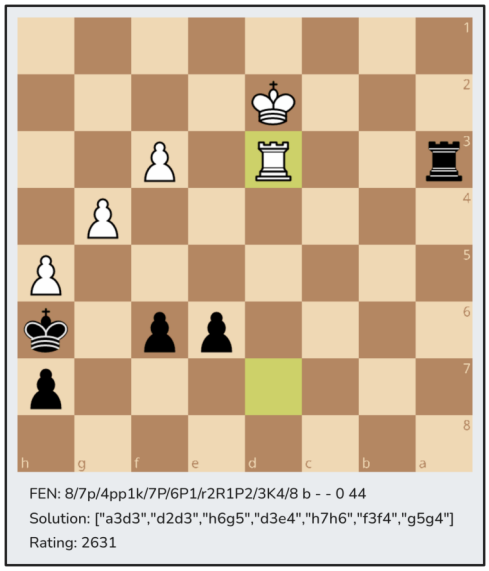

Fig. 1: An example puzzle from the LICHESS database. The most basic information required is the initial position (given in `FEN` format), the solution (from which we can generate a sequence of boards), and the puzzle rating (which we are trying to predict). However, the LICHESS and competition datasets provide some additional puzzle metadata. The theme of this puzzle is to convert white's blunder (capturing the pawn on `d3`, which starts the puzzle) into a winning pawn endgame via zugzwang. The algebraic solution is `♖xd3?? ♜d3+ ♔xd3 ♚g5 ♔e4 ♟h6! ♙f4+ ♚xg4`, after which black has an overwhelming pawn advantage.

Figure 1 gives an example from the LICHESS database.[2]

A. LICHESS *Dataset*

The training dataset given by the competition is a list of over 4.5 million puzzles taken from LICHESS. This is a snapshot of the free-to-download, updated-monthly puzzle dataset taken several months prior to the start of the competition. This dataset contains per-puzzle metadata that is the same as the

*This work is not related to the author's work at Amazon.

[1]Our code can be found here: https://github.com/mcognetta/ieee-chess. We have moved the implementation from the previous iteration of the contest to the `2024-iteration-archive` branch.

[2]https://database.lichess.org/#puzzles

prior iteration of this competition (we point the reader to [3], [4]) with the addition of a set of "success probabilities" computed by the MAIA2 neural chess engine.

B. MAIA2 *Success Probabilities*

This year's competition introduces new metadata in the form of success probabilities generated by MAIA2, a neural chess engine designed to mimic human play (Section III-B). Each value corresponds to the estimated likelihood that a player of a given skill level would successfully complete the puzzle according to the move distributions predicted by 22 different variations of the MAIA2 model [1]. These probabilities were included as scalar features, indexed by their model configurations and were provided for both the LICHESS and competition datasets by the organizers.

III. NEURAL CHESS MODELS

Here, we briefly introduce the families of neural chess models that we use as the backbone for our system. These models differ from classical chess models in that they are trained entirely on data without any hand-crafted features. In particular, we utilize MAIA and MAIA2, which are a family of neural models that are designed to mimic human players of various strengths, and LEELA, an open-source reimplementation of the ALPHAZERO neural chess system. This is not an exhaustive set of neural chess engines, and we point the interested reader to [5] for an overview.

A. MAIA

MAIA is a set of neural chess models that are trained to mimic players of a given rating (for ratings 1100 to 1900 in increments of 100) [6]. Each model has the same architecture—a chain of 6 convolutional blocks with residual connections, followed by a value and policy head—that takes a board representation as input and produces a policy and value score which encode the probability distribution over next moves and the evaluation of the position, respectively.

B. MAIA2

MAIA2 builds upon the ideas of MAIA, but with two key differences. First, the models are no longer segmented by rating; they now take the player ratings as input. And second, the model includes a new rating-board attention layer, which injects rating information into the board representation [7]. The board input and output representations are also different than MAIA, but the training objective is the same.

MAIA2 comes in two flavors, RAPID and BLITZ, denoting the training data used to train it, and the default parameters chunk player ratings into 11 buckets (1050 to 2050).

C. LEELA

LEELA is an open source implementation of the ALPHAZERO neural chess engine [8], [9]. The most modern iteration of LEELA uses a transformer backbone, but we opted to continue using the original model (as we did in [3]), which uses a convolutional backbone. LEELA was trained via self-play to be as strong as possible and has SMALL, MEDIUM, and LARGE variants.

IV. OUR SYSTEM

Our general approach mirrors our implementation from the prior competition [3]. In short, we use neural chess board embedders to extract latent representations of the boards in a puzzle. These are fed into an RNN to produce a puzzle representation, which is then combined with puzzle metadata and passed to a regression MLP. Our solution for this challenge differs from our previous solution in a few details, including an updated, optimized implementation and some architectural and modeling differences. Here, we review the main architectural components of our system and in Section VI-A, we discuss the application of this system to the actual competition challenge.

A. Board and Puzzle Embedders

We again use the MAIA and LEELA neural chess engine family as our neural chess embedding backbone. These are large, convolutional models that ingest board representations (i.e., bitboards representing piece positions and castling metadata) and produce a series of policy and value scalars that are used during min-max tree search in a chess engine. We truncate the models by removing the policy and value heads and take the output of the last convolutional block as the latent vector representation of a given board state.

We point the reader to Figure 10 of [6] (in the Supplemental section) which shows the MAIA architecture. Our latent representation is the vector that is produced by the convolutional block directly before the model is split into the two heads (and we use the analogous vector for the LEELA family, which has a similar architecture [8][9]).

However, puzzles are inherently sequential and variable length. Thus, after extracting latent representations of each board-state in a puzzle, we combine them into a puzzle representation by feeding them into an RNN to produce a single, vector representation.

B. Board Metadata

The LICHESS dataset comes with a large amount of additional metadata other than just the puzzle information. For example, it contains themes (e.g. mate-in-two, pin, etc.), popularity, the game it was drawn from, and more. This metadata may be helpful for modeling purposes, so our architecture allows for injecting representations of the metadata into the latent representation of the puzzle. The competition dataset, however, contains only the explicit puzzle information and the set of precomputed MAIA2 puzzle probabilities (Section II-B), so much of the metadata that is available during training is not immediately helpful since we do not have access to the same metadata during inference.

In our final architecture, we use only puzzle length and MAIA2 puzzle probabilities as metadata features. For the former, we produced a categorical representation of puzzle length (e.g., puzzles of length 2, 4, ..., 16) and used an embedding table to extract the learned metadata representation corresponding to the length of the given puzzle. For the latter, which consists of 22 scalar values, we produce a vector

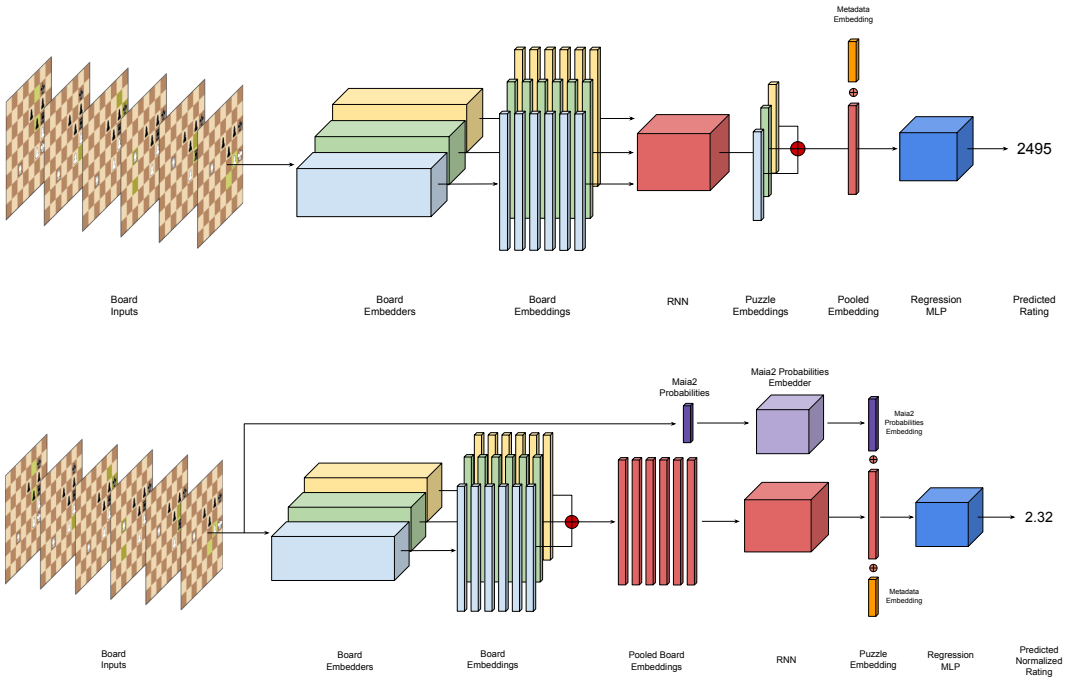

Fig. 2: The model architecture used in the previous competition (top) [3] and the system in this paper (bottom).

embedding by constructing a 22-length vector and passing it through a small MLP to produce a latent representation.

We note one feature, puzzle themes, that is present in the LICHESS dataset but not the competition dataset. However, the LICHESS theme tagging algorithm is open-source[3] and can be used to tag the competition dataset. At least one team in the first iteration of this competition tagged the competition dataset so that the feature could be used during inference [10], however, we did not experiment with including this metadata.

The metadata representations are concatenated to the output of the RNN to produce the final puzzle representation.

C. Rating Prediction as Regression

Once a latent vector representation of the board and its metadata has been formed, we use it to predict the GLICKO2 rating of the puzzle—a simple regression task. We use a three layer, dense MLP with `relu` activations and dropout to learn a mapping from a puzzle representation to its scalar rating.

D. Ensembled Embeddings

Like in [3], we do not restrict ourselves to having a single neural board embedder in a model. Instead, we allow multiple neural board embedders to independently produce board representations which are then combined to form an ensembled board representation. The intuition behind this is that different models have different strengths and weaknesses due to their training objectives and so their latent representations of the board may be complementary and lead to better modeling.

[3]https://github.com/ornicar/lichess-puzzler

Our architecture allows for arbitrarily many board embedders and can mix MAIA and LEELA embeddings.

V. ARCHITECTURAL AND MODELING DIFFERENCES FROM OUR SUBMISSION TO THE PRIOR CONTEST [3]

Despite using the same initial framework as in [3], we implemented a number of architectural and modeling changes (other than the inclusion of the new MAIA2 features) that improved our system's performance. Figure 2 shows the major differences between the two systems.

A. RNN Implementation

In the previous iteration of this competition we manually implemented the RNN in our model architecture. Regardless of puzzle input length, we truncated or padded to 10 boards and then used a hand-written RNN loop. This was slow, had many wasted computations (in that many puzzles have less than 10 boards, but the padded boards were processed anyway), and had some undesirable modeling properties (in theory, models should learn to ignore the padding boards, but they can still "wash out" the signal of non-padding boards, especially when the puzzle is very short).

We replaced our implementation with the PYTORCH RNN implementation, which is faster, uses less memory, and, by using packed tensors, only processes non-padding boards. In a benchmark, we observed a $>4\times$ runtime speed-up when using the PYTORCH RNN compared to our hand-written RNN.

B. Board Pooling

Recall that we have several board embedders that each produce a sequence of latent board representations for each board

(a) Results from the previous competition, taken directly from [3]. These do not include MAIA2 features and use the previous iteration of our architecture. Note that the bottom row is an output-ensemble of embedding-ensemble models.

Model	MSE
MAIA-1300	$56.5k$
MAIA-1500	$57.5k$
MAIA-1700	$57.9k$
LEELA-SMALL	$65.5k$
LEELA-MED	$66.1k$
MAIA-1300 + LEELA-SMALL	$65.4k$
MAIA-1300 + LEELA-MED	$50.8k$
MAIA-{1300, 1500, 1700}	$56.0k$
MAIA-{1300, 1500, 1700} + [MAIA-1300 + LEELA-SMALL] + [MAIA-1300 + LEELA-MED]	$46.7k$

(b) Results for our updated architecture, which includes MAIA2 features. Note that all of these are embedding-ensembled models. The right column is the same model configuration but with an additional LEELA-SMALL embedder.

Model	MSE	
		+ LEELA-SMALL
MAIA-1100	$41.2k$	$41.3k$
MAIA-1500	$41.6k$	$41.3k$
MAIA-1700	$42.2k$	$42.1k$
MAIA-{1100, 1500}	$41.7k$	$38.9k$
MAIA-{1100, 1700}	$40.3k$	$38.7k$
MAIA-{1500, 1700}	$39.7k$	$39.3k$
MAIA-{1100, 1500, 1700}	$39.2k$	**38.3k**

TABLE I: Results on the LICHESS dataset using different model configurations, including from our solution to the previous competition. Each model configuration's MSE is the average of 3 copies of the model. The best performing model is **bolded**.

state in the puzzle and our goal is to somehow combine all of them into a single puzzle representation. In [3], we achieved this by first computing a puzzle vector for each embedder (by processing that embedder's board representations with the RNN) then (mean) pooling the resulting vectors.

In our updated implementation, we reverse this process. We now use the embedders to compute each board state, then (mean) pool all of the analogous board representations before passing it through an RNN.

This has two main benefits. First, it allows for different embedders' board representations to be "mixed" earlier in the model pipeline, which should allow the model to better incorporate the different representations. And second, the board embeddings are processed in parallel in a giant batch, but the RNN inherently involves a non-parallelization, sequential process. Thus, by pooling the embeddings before the RNN layer, we reduce the number of non-parallelizable operations to just a single RNN call, which massively speeds up training and inference runtimes.

Due to the runtime speed up and lower memory requirements of the combination of pre-RNN board pooling and the PYTORCH RNN implementation, we were able to increase our training batch sizes and observed more stable training and faster convergence.

C. Regression Target

Recall that the primary objective of this competition is to predict a puzzle's GLICKO2 rating, which is a non-negative scalar, and try to minimize MSE. In [3], we simply predicted this scalar value directly. Since the ratings can be higher than 3000, the squared errors of predictions can be very large, which leads to training instability due to large gradients.

In our updated version, we instead computed the mean μ and standard deviation σ of the clean LICHESS dataset and scaled the ratings as $r_n = \frac{r-\mu}{\sigma}$ to produce a normalized rating (which was used in the MSE objective). During inference, we reverse the normalization by predicting a normalized rating $\hat{r}_n$ and recovering the predicted rating $\hat{r} = \sigma \times \hat{r}_n + \mu$. This resulted in more stable training and faster convergence.

VI. RESULTS

Overall, our updated model architecture and training scheme, coupled with the addition of MAIA2 probability metadata, led to significantly more accurate models on the LICHESS dataset. Table I shows the results of our prior solution (taken directly from [3]) and a subset of the model configurations we trained for our final submissions.

We note that, across the board, the updated models have substantially lower MSE, despite having similar parameter counts. Most clearly, we can compare models which have the same embedding configurations and thus the salient differences are our new architecture and the MAIA2 metadata (regrettably, we did not perform MAIA2 ablation tests).

For example, comparing the MAIA-1500 model from the previous competition to the MAIA-1500 model from this competition, we observe a $\sim 16k$ reduction in MSE (a 27.7% reduction), despite having similar parameter counts.

Perhaps unsurprisingly, we find that our largest ensemble, MAIA-{1100, 1500, 1700} + LEELA-SMALL, performed the best — achieving $38.3k$ MSE on the LICHESS test set, which is an $8.4k$ (18.0%) reduction in MSE over the best, multi-ensembled model from our submission to the prior competition. Ensembling all of the models that we trained for this competition (i.e., the entire set of models from the right two tables of Table I) reached $36.1k$ MSE corresponding to a $2.2k$ (5.7%) reduction in MSE over the best individual model from this competition and a $10.6k$ (22.7%) reduction over the multi-ensembled model from the last competition.

A. Fitting to the Competition Dataset

Like in [3], we found that models that performed well on the clean LICHESS dataset did not necessarily perform well on the competition dataset, as there are significant differences between the competition distribution and the LICHESS dataset distribution (for example, see Figure 1 of [4]).

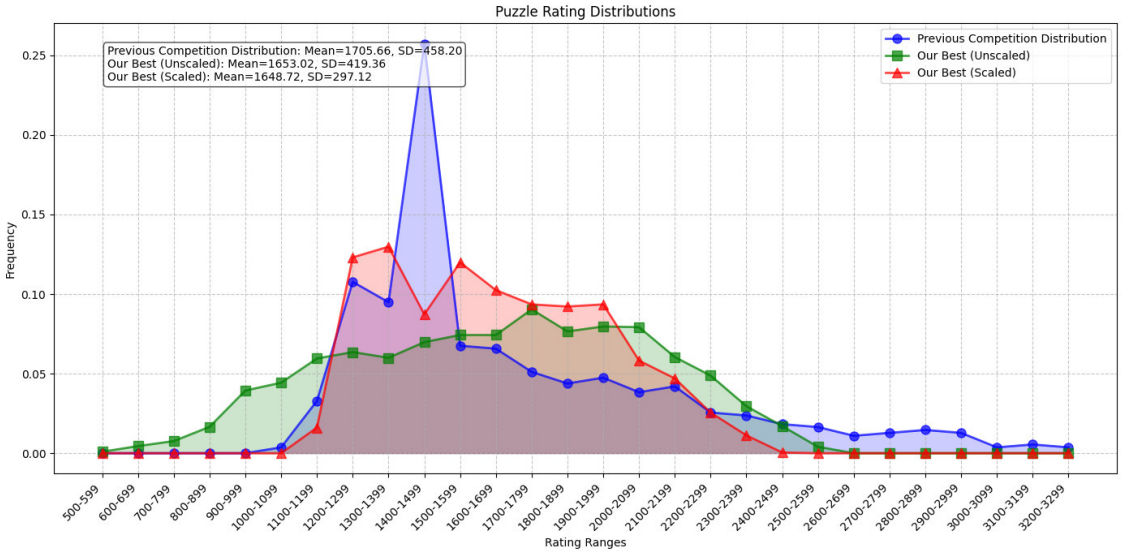

Fig. 3: An overlay of the competition test set's distribution (blue) as reported in the prior competition's writeup [4], our best model's predictions before scaling (green), and our best, scaled result (red). Notice that they have significantly different shapes, implying that accurately modeling one is not a guarantee of accurately modeling another.

We again used the strategy of producing as good of a model as possible on the LICHESS dataset and then rescaling it to fit what we expected the competition dataset to look like.

Previously, we used a simulator that estimated, given a set of players and puzzles for which we had gold-standard rating information, what the distribution of puzzle ratings would look like after each puzzle had been played by a subset of N random players. This was designed to mimic the competition server, which matched players with puzzles randomly (i.e. not matching players and puzzles of similar skill level) and, at the time, did not have many plays-per-puzzle. We used these findings to guide our search for a rescaling function that could map our predicted ratings to what we would expect that puzzle to be rated in the low-play competition-server regime.

We followed the same procedure in this iteration, by using rescaling functions that attempted to reshape our distribution of predicted competition puzzle ratings to be more centered around what we believed the mean competition rating to be.

First, as a baseline, we noted that this year's competition dataset was a subset of the previous year's competition dataset. Thus, we simply reused our winning solution from [3] after filtering out the removed puzzles. On the preliminary competition dataset from last year, our winning solution had $49.1k$ MSE. However, the same solution scored $64.1k$ MSE, indicating that the current iteration of the competition dataset is significantly different than the prior version.

We first performed binary search over the set of possible ratings (between 0 and 3000) for the value that minimized MSE on the preliminary competition dataset when we used it as a constant prediction (i.e., we predicted the same value for all puzzles). The value that minimizes this function gives the mean of the dataset (and we can also recover the standard deviation), approximately $\mu \approx 1650$.

We then used post-processing rescaling steps to try to reshape our predicted output distribution to a distribution that had similar properties to what we expected the competition dataset to have and settled on rescaling functions of the form:

$$\text{SCALE}(r) = \begin{cases} o_u + \frac{\log(c_1 r)}{c_2} & r > 0 \\ o_l - \frac{\log(c_1' r)}{c_2'} & r \leq 0 \end{cases}$$

where r is the normalized predicted rating $\frac{\hat{r}-\mu}{\sigma}$ and o_u, c_1, and c_2 are rescaling hyperparameters that we searched over for when our predicted rating was above the mean (and likewise for o_l, c_1', and c_2' when it is below the mean). We separated these so that we could control the shape of both sides of the output distribution, as we found it was not symmetric. The normalized predicted rating is mapped back into the raw rating space after rescaling.

The differences between the prior competition dataset, our model's (unscaled) predictions on the competition dataset, and our best solution's scaled predictions is shown in Figure 3.

VII. THINGS WE TRIED THAT DID NOT WORK

Like in [3], once our core pipeline was finished, we experimented with several variants but found that none provided gains over our baseline.

A. *RNN → Transformer*

One goal of our updated implementation was to improve the model's runtime performance in order to improve training speed and performance by increasing the batch size and parameter count. A major bottleneck is the RNN layer (particularly in the older, hand-written implementation that pooled boards after the RNN), since it is inherently sequential and cannot

be parallelized. We experimented with replacing this layer with a small transformer model, which has the benefit of increased parallelization and the ability to explicitly inspect the representations of every board in a puzzle simultaneously and learn a weighting for their interactions. We hoped that this would improve both the model's runtime and quality, but we instead found that it massively reduced the modeling performance, even at similar parameter counts.

We experimented with several variations, including stacking several layers of transformers and adjusting the transformer's internal hidden dimension (both within a fixed parameter budget), but we were not able to match the RNN baseline.

B. MAIA2 *Board Embeddings*

We experimented with adding MAIA2 as a board embedder by using the output of the board-rating attention layer as the latent board representation. As the MAIA2 model requires player ratings as input, but the competition dataset does not provide this metadata (the LICHESS dataset does include it), we experimented with several different schemes to select a reasonable rating. For example, we performed a hyperparameter search over all pairs (p_1, p_2) where p_1, p_2 were ratings in the MAIA2 rating buckets to see if some pair worked well. We additionally tried using the ratings as a form of self-ensembling, by simply selecting several rating pairs as hyperparameters and then computing the latent representation of the board for each pair and averaging them.

None of these schemes were effective and we found that models using MAIA2 embeddings were substantially worse than those using MAIA. This is potentially due to the comparatively small output dimension of the rating-board attention layer of MAIA2 (1024) compared to the convolutional block layers of MAIA (4096), which means that it carries less signal when used in the downstream task.

Since MAIA2 also uses a sequence of convolutional blocks as a backbone before the attention layer, we could have used the output of that as well, but we did not experiment with this.

C. Alternative Loss Functions

Our baseline model uses MSE loss as the primary loss function in order to mimic the competition objective. However, as noted in [3], our best performing models had relatively high MSE (40k+ and 50k+ on the LICHESS dataset and the preliminary competition dataset, respectively). Thus, we hypothesized that "small" errors (for example, those with absolute error of less than 180, which would have a squared error of at most 34k) were less important than large errors (e.g., predicting 1800 when the true rating was 2500, corresponding to a squared error of 490k).

We experimented a with piecewise loss, where the model was penalized more harshly for larger errors. For example, we considered loss functions with the basic form:

$$\mathcal{L}(\hat{r}, r) = \begin{cases} (\hat{r} - r)^2, & \text{if } |\hat{r} - r| \leq T \\ (\hat{r} - r)^4, & \text{otherwise,} \end{cases}$$

where r and $\hat{r}$ are the (unnormalized) true and predicted ratings, respectively, and T is some threshold (180, in the example above, though we used normalized rating thresholds in the actual implementation). We also tried more stratified loss functions (e.g. with multiple thresholds and loss scaling functions), but did not observe any improvements.

REFERENCES

[1] J. Zyśko, M. Ślęzak, D. Ślęzak, and M. Świechowski, "FedCSIS 2025 knowledgepit.ai Competition: Predicting Chess Puzzle Difficulty Part 2 & A Step Toward Uncertainty Contests," in *Proceedings of the 20th Conference on Computer Science and Intelligence Systems*, ser. Annals of Computer Science and Information Systems, M. Bolanowski, M. Ganzha, L. Maciaszek, M. Paprzycki, and D. Ślęzak, Eds., vol. 43. Polish Information Processing Society, 2025. [Online]. Available: http://dx.doi.org/10.15439/2025F5937

[2] M. E. Glickman, "Example of the glicko-2 system," http://www.glicko.net/glicko/glicko2.pdf, 2022.

[3] T. Woodruff, O. Filatov, and M. Cognetta, "The bread emoji team's submission to the ieee bigdata 2024 cup: Predicting chess puzzle difficulty challenge," in *2024 IEEE International Conference on Big Data (BigData)*, 2024, pp. 8415–8422.

[4] J. Zyśko, M. Świechowski, S. Stawicki, K. Jagieła, A. Janusz, and D. Ślęzak, "Ieee big data cup 2024 report: Predicting chess puzzle difficulty at knowledgepit.ai," in *IEEE International Conference on Big Data, Big Data 2024, Washington DC, USA, December 15-18, 2024*. IEEE, 2024.

[5] D. Klein, "Neural networks for chess," 2022. [Online]. Available: https://arxiv.org/abs/2209.01506

[6] R. McIlroy-Young, S. Sen, J. M. Kleinberg, and A. Anderson, "Aligning superhuman AI with human behavior: Chess as a model system," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, Eds. ACM, 2020, pp. 1677–1687. [Online]. Available: https://doi.org/10.1145/3394486.3403219

[7] 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," 2024. [Online]. Available: https://arxiv.org/abs/2409.20553

[8] T. L. Authors, "Leela chess zero," https://lczero.org/.

[9] 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

[10] S. Björkqvist, "Estimating the puzzlingness of chess puzzles," in *2024 IEEE International Conference on Big Data (BigData)*, 2024, pp. 8370–8376.