Performance Prediction in Chess using Sequence Models

Stanley Cao
Stanford University
stanley.l.cao@stanford.edu

Arthur Lee
Stanford University
arthurlh@stanford.edu

***Abstract*—We explore the novel problem of using language models to predict chess ratings from the quality of play. Specific models explored in this paper include RNNs, LSTMs, GRUs, and the transformer architecture. While most of the recurrent architectures perform similarly, we find that the GRU and the transformer model achieve the best performance of the aforementioned models according to the L1 loss metric. However, all of our models, still obtain very high loss (≈ 200 rating points of error per prediction), indicating that the problem of performance prediction is an inherently challenging problem.**

I. Introduction

In competitive chess, *Elo ratings* are used to determine the relative strength of players. Proposed by Arpad Elo (1) and formally adopted in the 1970's by the World Chess Federation (FIDE), Elo ratings provide a precise ordinal ranking of the player strength of every player within a "rating pool". The mechanics are as follows: all players have some uniform initial rating. After each game, the winner of a game *steals* points from the loser in proportion to the difference in their respective ratings. If a higher-rated player beats a lower-rated player, a "small" number of points are taken away from the lower-rated player. If a lower-rated player beats a higher-rated player, a "large" number of points are taken from the higher-rated player (in the event of a draw, the lower-rated player gains some points from the higher-rated player).

Formally, the Elo rating model assumes that for a player A with rating R_A and B with rating R_B, we have that the Expected score (where 0 is a loss, 0.5 is a draw, and 1 is a win) of player A is

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

and the expected score of B is $E_B = \frac{1}{1+10^{(R_A - R_B)/400}}$. Let S_A, S_B be the actual scores of players A, B in a game. Then we update their Elo ratings with the rules $R'_A = R_A + K_A \cdot (S_A - E_A)$ and $R'_B = R_B + K_B \cdot (S_B - E_B)$. K_A, K_B are a constant determined by the organization that administers the rating; for the World Chess Federation, they use $K = 40$ for players new to the rating list, $K = 20$ for players rated below 2400, and $K = 40$ for players above 2400.

Outside of chess, Elo ratings have been used for competitive rankings in other games such as Go, Backgammon and Scrabble. They have also been used to evaluate the performance of Machine learning models, for example in the case of multi-agent Reinforcement Learning (2).

Elo ratings are a *relative* metric and tell us little about the *absolute* performance of players in chess. For example, in a rating pool comprised entirely of very weak players, an intermediate player could have an arbitrarily high rating. However, a mapping between Elo ratings and "absolute" playing strength defined by some objective measure could be useful in a number of domains. The following is a sketch of potential use cases:

- **Historical comparison of players from different eras.** Elo ratings do not allow for direct comparison of the playing strength of players from different time periods, who did not face each other and had totally different sets of opponents. It is also thought that over time, Elo ratings have experienced "inflation": a player of constant absolute strength is likely to be higher rated in 2022 than in 1980. Much research has been done to unify ratings on a single scale (3). These typically involve scoring players on some new rating scale (e.g. correlation of moves with chess engine moves). However, our approach allows us to translate the absolute strength of a historical player to *today*'s rating scale, which not only allows ordinal comparison of playing strength but aids interpretability.
- **Player self-evaluation**. Beginners, who do not possess a FIDE or US Chess Federation (USCF) rating, may be interested to know their possible rating were they to acquire one. Using their rating from online sites like chess.com may be unsatisfactory, since folk wisdom typically holds that ratings on chess.com are typically 200-300 points higher than the corresponding FIDE rating.[1] Intuitively, this is because chess.com may have a higher pool of "weak" players that players can acquire points from. Moreover, ratings may take a while to "stabilize", whereas a sophisticated machine learning algorithm could quickly provide a granular estimate of playing strength.
- **Cheating detection**. Cheating detection has recently acquired new significance in professional chess due to high-profile accusations made against American chess grandmaster Hans Niemann by reigning world champion Magnus Carlsen.[2] Having a system that provides a predicted Elo rating could potentially identify isolated

[1] See, for example: https://www.chess.com/forum/view/general/how-does-chesscom-ratings-compare-with-fide-ratings?page=2

[2] https://www.nytimes.com/2022/09/13/crosswords/hans-niemann-magnus-carlsen-cheating-accusation.html

instances of cheating (e.g. where a player's predicted rating from a game or series of games is at odds with their "actual" rating, meaning that they are "playing above their strength"). While resolving the issue of whether Niemann cheated in this specific game likely requires more fine-grained analysis, a model that predicts performance-based Elo ratings is likely useful for settings such that online cheating detection (e.g. if a 1400-rated player plays one game at a 2400 level, it is likely they are using an engine).

- **Calibration of format rules**. Tournament organizers and observers would be able to provide a quantitative answer to questions such as "does reducing time control by X minutes result in a deterioration in the quality of play?" or "how much is a grandmaster handicapped if they are made to play chess with a blindfold?"

II. Related Work and Novelty

Our work is directly related to past attempts to quantify the "intrinsic" strength of players. These typically measure the statistical similarity between an ensemble of engine moves and a player's moves (4) (5). Regan (6) defined two parameters that measure the difference between a player's choice of move and the engine's cardinal ranking of "good" moves. He showed that these parameters had strong correlation with Elo ratings. A more recent paper by Alliot (3) introduces a method that involves computing stochastic transition matrices for each player at each possible move, where entries (i, j) of the matrix represent the probability that a player will transition from a position with engine evaluation i to one with evaluation j. The paper uses the stationary distribution of these transition matrices to determine the win probability for each player in a hypothetical head-to-head match. Our work is differentiated in two keys ways: first, in empirical tests, most of these studies focus only on ranking the world's top players from different periods, whereas we want to provide a comprehensive classification scheme that applies to beginner and intermediate level players. Second, these methods rely on closed-form statistical measures and do not use deep learning. To our knowledge, despite the natural interpretation of chess moves as sequential data, this is the first attempt to use sequence models to interpret the quality of play in a chess game.

Finally, our research builds on modern literature about the use of machine learning in sequence modelling, in particular many-to-one sequence models. We will use methods such as Gated Recurrent Unit (GRU) models (7) and the transformer architecture (8), incorporating recent optimizations and advancements such as sequence packing.

III. Dataset

We make use of the lichess.org game database. lichess.org is a free and open source online chess playing platform. It publishes a monthly public database of over 90 million games (including moves and the lichess.org ratings of the players involved) of which approximately 6% include information about Stockfish analysis. These games are in PGN format (a system of chess-specific notation).

Thus far, we have made use of a dataset comprising 200,000 games played in December 2019 that were scraped from PGN format to CSV files by chess website web.chessdigits.com. These games include engine evaluations. Table I shows the main features of the games in our dataset. The target variable (Lichess ratings) are centered around 1500 with SD of 322.[3]

Our data pre-processing approach was as follows:

- We used R to pre-process the data. First, we pivoted the data such that each row corresponded to a unique pair of (game index, move). For every move, we extracted relevant features such as which piece was moved, the engine evaluation, number of moves to checkmate (if applicable), the amount of time remaining, etc. We also encoded string data (e.g. whether a game was rapid, blitz or bullet) as dummy variables. A sample of the post-processed data from R is included in the github (labelled "for_pandas.csv")
- Next, we removed all moves beyond 150 for all games (we define one "move" as an action by a single player, in contrast to conventional chess notation where a "move" refers to a tuple of a white move followed by a black move). For games with less than 150 moves, we pad missing values with zeros. This is inspired by the approach taken in Natural Language Processing and other sequence models.
- We then normalized columns with large values through either min-max normalization or standardization. We normalized the output vector (a pair of ratings [WhiteElo, BlackElo] by dividing by 2000 (the vast majority of ratings were between $[800, 2000]$).

Due to the high number of examples, we used a 90/10 training/test split for our models in all subsequent sections. Unfortunately, at the time of writing all our models were only trained on 50,000 data points (out of the 200,000 available in our Dec 2019 database) due to computational limitations. We ran out of RAM on both the EC2 instance and our local devices, and also faced long training times (training on 50,000 data points already took 15 minutes per epoch). In the future, a more efficient approach (e.g. use of sparse matrix methods or just asking for more virtual RAM and vCPUs) may improve our training and could generalize our method to the full datset of 200,000, or even the total lichess.org database which includes tens of millions of annotated games.

IV. Baseline

We used a vanilla feedforward neural network with two hidden layers as our baseline. We flattened each sequence into a vector of length (sequence length) $*$ (num feature) $= 150 * 377 = 56550$ and trained it on our dataset. We did not expect this method to yield particularly accurate results, since a flattened version of the game's data does not preserve the

[3]The highest rating of 3110 likely belongs to Magnus Carlsen, who plays under the pseudonym DrNykterstein

TABLE I: Summary Statistics for Data

Statistic	N	Mean	Median	St. Dev.	Min	Max	Pctl(25)	Pctl(75)
White Elo	200,000	1,511.802	1,500	322.308	800	3,091	1,275	1,732
Black Elo	200,000	1,512.332	1,500	322.455	800	3,110	1,276	1,732
Number of moves	200,000	63.101	59	26.048	5	200	45	75
Time Control (seconds)	200,000	385.786	300	371.226	3	10,800	180	600

sequential information of the game. Moreover, having such a large number of features renders us vulnerable to the curse of dimensionality, increasing overfitting.

We experimented with a range of hyperparameters, finding best performance with a network of 128 neurons and learning rate of 0.0001. An important question was the choice of loss function and evaluation metric. Three approaches were considered: first, a regression loss with mean-squared error (MSE) where the outputs were a pair of numbers $y \in \mathbb{R}^2$ representing each player's rating. Second, the same output but with a L1 Loss function. Finally, a classification problem where ratings are classified into rating "bins" (e.g. 1500-1700 strength, 1700-1900 strength, etc) and cross-entropy loss is used to train the model. Between L1 and L2 loss functions, we ultimately opted L1 Loss for two reasons: first, MSE loss had poor ability to learn the data, performing worse on a L1 (absolute deviation) loss basis than just outputting the average. We speculate that this is because the MSE loss function was overly sensitive to the presence of outlier examples, and this dataset contains plenty of outliers (e.g. highly rated players performing purely in blitz format games). Additionally, we felt that minimizing L1 Loss provided a more interpretable output (the mean rating deviation).

We chose regression over classification as the default approach because we were having issues with the convergence of our classification model. Additionally, regression has a more "natural" interpretation since it better captures the ordinal nature of the problem (i.e. the algorithm should be penalized less severely for classifying a 1500 player as 1800 rather than as 2700, however a classification approach with bin width of 200 would penalize both equally). Nevertheless, we still consider the classification approach as a possible optimization and briefly discuss our results.

The baseline model performs relatively well, with L1 Loss of about 210 Elo Points (Table II. This means that despite the problems we highlighted above, it is able to learn nontrivial features of the data. We stopped training after the test loss began increasing, since we noticed the model overfit relatively early in the training process (Figure 2).

V. Methods and Models

We considered various sequence model architectures and optimizations, discussed below. For RNN-based approaches, we tried using both the ReLU and tanh activation functions. We also tried various values in the range $\{32, 64, 128, 512\}$ for batch size and values in the range $\{0.1, 0.05, 0.01, 0.005, 0.001\}$ for the learning rate. We also experimented with sequence lengths of 50, 75, 100, 125 and 150, but found that 150 yielded the best results (200 would be too large as 99 percent of games ended before move 150). Our general approach for hyperparameter selection was to run the model with altered hyperparameters on a base RNN model first, and with a smaller amount of data (5000 examples).

We implement a number of sequence models: first, a standard RNN model with no memory cells. We then study whether LSTM and GRUs can improve on RNNs to encode long-term dependencies in the data. Next, we study additional optimizations: a multi-head architecture, encoder-only transformer network, and other types of optimizations (cross-entropy loss and packing).

A. RNN Model

We implement a vanilla many-to-one RNN model with 3 hidden layers and 256 hidden units. We experimented with values of $\{128, 256, 512\}$ and 2 hidden layers, but there were no significant impacts on performance. The final output of the RNN is fed into a dense layer with 256 neurons with RELU activation followed by a final dense layer with a 2-dimensional output. Since our model overfit quickly, we chose to apply a dropout of 0.2 (we also tried no dropout, and 0.5). We initially had problems with our model predicting the sample average outcome, independent of the input features. This was solved through a mixture of feature normalization and random initialization of RNN hidden units (rather than the default initialization, which is zeros). We also enacted early stopping to mitigate the overfitting that we observed in the baseline model. The RNN model performed poorly, with a L1 loss of 229 Elo points on the test set, about 10 % worse than the baseline.

B. LSTM and GRUs

We suspect that one reason why the RNN model has substandard performance is due to its inability to "remember" long-range dependencies, particularly in long padded sequences. LSTM and GRU models may address this issue. LSTMs incorporate four gates: update gates, relevance gates, output gates, and forget gates. Update gates control how much to update the current cell state; relevance gates control how much the last activation output is relevant to this time-step's output; the output gate controls how much to reveal of the cell state; and forget gates decide how much of the previous cell state should be remembered. A GRU model, introduced by Cho et al (7) is based on a similar architecture except with only update and relevance gates.

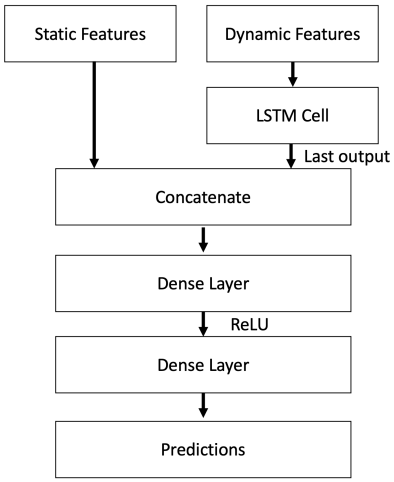
We would expect LSTM models to outperform GRU models given enough data and computational resources, since GRUs

are less complex (no cell state) and have fewer trainable parameters. However, in many empirical applications their performance is roughly equal and GRUs are sometimes preferable due to their computational efficacy.(9) On smaller datasets, this equivalence was observed to hold. However, when trained on our larger (50,000 sample) dataset, we observed that LSTM models performed worse than GRUs. GRUs outperformed the baseline model, whereas LSTMs performed similarly to RNNs. It is unclear why this occurred, and more hyperparameter tuning may be required to study this phenomenon in greater detail. It is possible that LSTMs overfit on the data whereas GRUs provide the right "balance" between long term memory and model complexity.

C. Multi-headed Model

One problem with the sequence models described above is that they cannot distinguish between static and dynamic data. For example, there is static data about the game common to all moves (e.g. the type of opening played, the time control format) and dynamic data unique to each move This poses two problems: first, it leads to computational wastage as we relearn weights for static features at every sequence step. More insidiously, static data may be over-emphasized since the last step of the sequence contains static data, but may not contain the most relevant segments of dynamic data. Even if we zero-pad static data, it may still have outsize influence on our LSTM output relative to its actual importance. Thus we experiment with a *multi*-headed architecture that concatenates the output of the LSTM cell (run only on dynamic data), with the static data features. This model ultimately performed similarly to the standad LSTM model, suggesting that this optimization did not yield substantive improvement and does not solve the issue of learning long-range dependencies.

Fig. 1: Multi-headed Architecture



D. Transformer Models: More than meets the Eye

In order to potentially address longer-range dependencies in our data, we turn to transformer models. Transformers models, proposed in a seminal paper by Vaswani et al. (8), provide a powerful way of tackling sequence problems such as language modeling and machine translation that have been difficult for traditional RNNs. It comprises an encoder stack and decoder stack, both of which implement a multi-headed self-attention mechanism. Through the attention mechanism, we hope that the model can learn to "focus" on the most salient parts of the game through learning an adequate embedding (with the use of positional encoding to tell us "where" in the game these moves are played).

Note that the decoder is required so that the model can piece together an output for a generation task (e.g. translating a sentence into English). Since our output value is a scalar value, not a sequence, we replace the decoder with two dense feedforward layers. Such "encoder-only" models are theoretically justified and have been used in settings such as object detection (10) and particle physics modelling (11). After (8), we use a transformers architecture with 6 attention heads, a 256-dimensional feedforward layer, and 6 sub-encoder layers.

On runs on a smaller dataset (approx. 10,000 samples), transformer models had the best performance of the models discussed here. However, their computational complexity made them difficult to train on larger datasets, requiring more than an hour per epoch on our 50,000 sample dataset. Hence, we were unable to complete training at the time of writing, and report in II the most recent loss number.

E. Sequence Packing

We implement the optimization of *packing* a padded sequence. Packing sequences is an optimization intended to stop the network from computing outputs for the padded layers of a sequence. This is similar to masking padded sequences.[4]. For many-to-many tasks, this appears to be primarily a efficiency optimization: we can drastically speedup training for large datasets with highly variable length sequences. (12) However, for a many-to-one task, where we used the representations from the last LSTM layer, this could potentially have a material effect on the model, since we are using the output of the last *relevant* index rather than the last index of the sequence as a whole. We run the LSTM model on this improved architecture. We achieve a result (table II) that is marginally better than the baseline, and better than the vanilla LSTM model.

F. Cross Entropy Loss

One of the potential problems with all the approaches above was that the model may default to predicting the "average" rating. In order to combat this, we turn the problem into a classification one. Specifically, we created a class for every 100 rating points between 0 and 3000. This means that we have a total of $3000/100 + 1 = 31$ class that the model can choose from. Thus, every label is assigned some class based on the player ratings. Note that there are two players, and thus each example sequence will correspond with two rating classes as

[4]Documentation for PyTorch implementation available at: https://pytorch.org/docs/stable/generated/torch.nn.utils.rnn.pack_padded_sequence.html

its label. Our input is designed to output logits for each rating class, which essentially corresponds to the confidence that the model has that each player is of a particular class. Note that the output of our model corresponds to the labels in that we output two sets of logits, one corresponding to each player. Our loss function is thus a multivariate Cross Entropy loss, which fits our regime nicely. We designed this classification problem so that the model loses the notion of an "average prediction," in order to incentivize it to learn the correct predictions based on the input data.

From figures 4 and 5, we note that the loss is still quite high (in the context of cross entropy loss), and the model is still struggling to learn these predictions, demonstrating that this is still a fundamentally difficult problem for RNNs to learn. One reason this might be the case is that RNNs struggle to learn long-range dependencies, and suffer from both exploding gradients and vanishing gradients. In other words, the error signal is likely not reaching the earlier layers, which hinders the model's ability to perform well in this task. Since the model did not have high accuracy in a basic RNN architecture, we focused our development process on the regression model.

VI. Analysis

From Table II, we note that most of the L1 loss metrics are within the 200 range. We note that the best performing models are the packed LSTM and the GRU model. They are the only sequence models that outperform the baseline on 50,000 sample dataset. However, we did not have enough time and compute power to fully train the transformer architecture on the 50,000 sample dataset. The success of packed LSTM models verifies our intuition that part of the problem comes from the length of sequence padding in certain games. As discussed above, it is unclear why GRU models outperform LSTMs and RNNs, though it could be because they provide the right balance of model complexity and memory ability for this problem. Additionally, overfitting remains a problem. From Figures 2 and 3, it appears that the a minimum of the test loss is found relatively early in the training process. Training loss under LSTM can decrease to around 40 Elo points, suggesting that model complexity is not a problem (rather, the model may be too complex).

On the smaller dataset (e.g. 5000 examples), the transformer model typically performed better than the baseline (at around 218 Elo Points deviation compared a baseline of 225). This is expected due to the transformer's ability to model and track long-term dependencies, as well as learn which parts of the games are very important. The other recurrent architectures (e.g., RNN, LSTM) performed slightly worse than the transformer model on the small dataset, likely due to the fact that predicting an accurate ELO rating for each sequence of moves not only requires a deep understanding of the game, but also the ability to determine which part of the games are actually indicative of a strong or weak player. The transformer model appears to achieve this goal, with its ability to learn the critical parts of a chess game.

Despite using state of the art models, the fact that our performance hovers in 200 range suggests that this particularly problem is fundamentally challenging, even for humans. To predict a player's "true ELO" given only one game is quite difficult, especially considering the various sources of noise in our data, which include the possibility of upsets, the natural variability in a player's performance, and the volatility in unofficial chess ratings from an online platform. Even though we include engine position evaluations as a data source, a low-skilled player can still have a favorable position when playing against another low-skilled player. Due to the unavoidable noise in our dataset, it is quite possible that players with a 200 rating point difference have similar levels of play. This is especially true at the amateur and club levels, where blunders and inaccuracies occur quite frequently, preventing the model from determining a player's ELO with more precision.

VII. Future Work and Conclusions

In conclusion, we showed that predicting chess performance can be modelled as a sequence problem, and sequence models perform similarly to feedforward neural networks when parsing this information on large datasets.

We believe the following are important future directions: first, we could improve feature engineering that brings in more contextual cues about chess (e.g. the amount of material remaining on the board at a given time, evaluation by more than one chess engine, etc). Second, we need a more rigorous framework for model selection and validation. Some of the conclusions drawn on a smaller dataset (e.g. poor performance of baseline NN model relative to RNNs) actually vanished when we moved from a paradigm of 10,000 data points to 50,000 data points. Most of our hyperparameter tuning was done on smaller datasets of 5-10,000 data points. Unfortunately, due to time and compute limitations, we were not able to re-tune the hyperparameters for the larger dataset or more rigorously perform model selection. Finally, we could consider other non-RNN based approaches, such as convolutional neural networks or more traditional statistical methods (such as those used by (3), which provide good performance for predicting the win probabilities of different players in a head-to-head match.

VIII. Team Member Contributions

Stanley Cao contributed the code and architecture for the RNN model and the implementation of both the regression and softmax based approach. Arthur Lee contributed to the sourcing and pre-processing of data and helped to fine-tune the regression based model.

REFERENCES

[1] A. Elo, "The proposed uscf rating system, its development, theory, and applications," *Chess Life*, vol. 22, no. 8, pp. 242–247, Aug 1967.

[2] X. Yan, Y. Du, B. Ru, J. Wang, H. Zhang, and X. Chen, "Learning to identify top elo ratings: A dueling bandits approach," *CoRR*, vol. abs/2201.04480, 2022.

[3] J.-M. Alliot, "Who is the master?" *ICGA Journal*, vol. 39, no. 1, pp. 3–43, May 2017.

[4] M. Guid and I. Bratko, "Using heuristic-search based engines for estimating human skill at chess," *ICGA Journal*, vol. 34, no. 11, pp. 71–81, 2011.

[5] D. Ferreira, "Determining the strength of chess players based on actual play," *ICGA Journal*, vol. 35, no. 1, pp. 3–19, 2012.

[6] K. Regan, "Intrinsic chess ratings," *Proceedings of 25th AAAI Conference on*, pp. 3–19, Aug 2011.

[7] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. [Online]. Available: https://aclanthology.org/W14-4012

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[9] S. Gao, Y. Huang, S. Zhang, J. Han, G. Wang, M. Zhang, and Q. Lin, "Short-term runoff prediction with gru and lstm networks without requiring time step optimization during sample generation," *Journal of Hydrology*, vol. 589, p. 125188, 2020.

[10] Z. Sun, S. Cao, Y. Yang, and K. M. Kitani, "Rethinking transformer-based set prediction for object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 3611–3620.

[11] S. Dutta, T. Gautam, S. Chakrabarti, and T. Chakraborty, "Redesigning the transformer architecture with insights from multi-particle dynamical systems," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 5531–5544. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/2bd388f731f26312bfc0fe30da009595-Paper.pdf

[12] M. Kosec, S. Fu, and M. M. Krell, "Packing: Towards 2x NLP BERT acceleration."

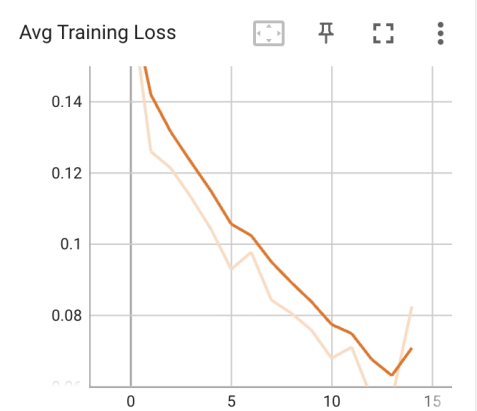
APPENDIX

TABLE II: Model architectures and their corresponding L1 Loss

Model	Hidden States	Units in Fully Connected Layer	L1 Loss on test set (Elo units)
Baseline NN	-	128	209.64
Baseline NN	-	256	211.51
RNN	256	256	229.55
GRU	256	256	203.13
LSTM	256	256	225.85
Packed LSTM	256	256	207.88
Multi-headed LSTM	256	256	224.39
Transformers	-	256	219.80

Fig. 2: Loss Function for Baseline Model

(a) Training Loss (y-axis) vs Number of Epochs (x-axis)



(b) Test Loss (y-axis) vs Number of Epochs (x-axis)

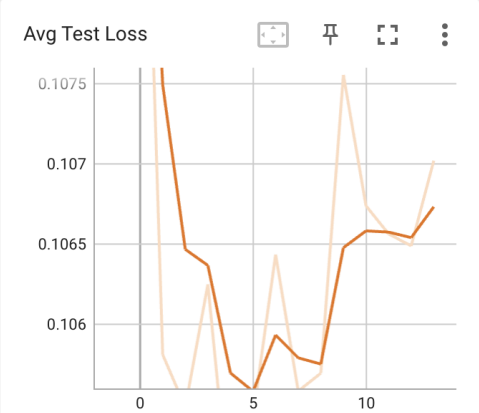


Fig. 3: Loss Function for LSTM Model

(a) Training Loss (y-axis) vs Number of Epochs (x-axis)

(b) Test Loss (y-axis) vs Number of Epochs (x-axis)

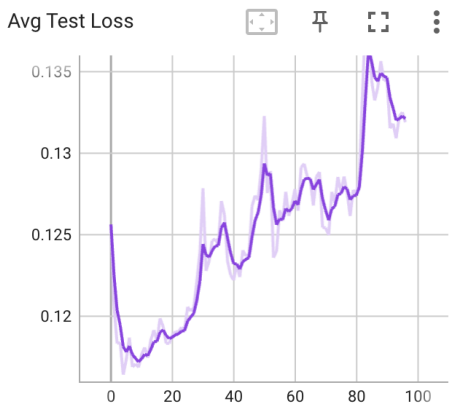


Fig. 4: Cross Entropy Training Loss (y-axis) on 92 Epochs (x-axis)

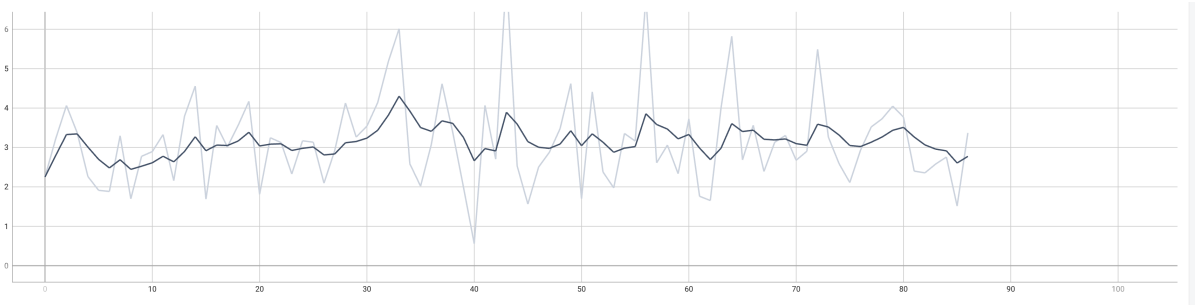


Fig. 5: Cross Entropy Test Loss (y-axis) on 92 Epochs (x-axis)