Learning Chess from Transformer-Based Language Model

Nuo Wang
Department of Electrical and Computer Engineering
Georgia Institute of Technology
nwang342@gatech.edu

Cheng Wan
Department of Electrical and Computer Engineering
Georgia Institute of Technology
cwan38@gatech.edu

Chunyuan Deng
Department of Electrical and Computer Engineering
Georgia Institute of Technology
cdeng73@gatech.edu

***Abstract*—Transformer-based language models have recently gained popularity in natural language processing tasks due to their ability to process sequential data. Chess, however, which can be represented as a sequence of moves, is an ideal domain for applying transformer models. Motivated by this, we leverage this ability to use these models for acquiring chess-related knowledge by training them on a large corpus of chess games and evaluating their performance on chess-related downstream tasks. The results show that Transformer language models when provided with sufficient amounts of training data, are capable of accurately tracking piece locations and predicting legal moves. The use of transformer models in this context presents an interesting test case for their capabilities and potential for learning complex rules and strategies in other domains.**

***Index Terms*—transformers, chess, language models, deep learning**

I. Introduction

The Transformer model has emerged as a significant advancement in the field of natural language processing due to its ability to effectively process sequence data [1] [2] [3]. This is achieved through the use of a self-attention method, which computes the relationships between elements in the input and output sequences. The mechanism allows the model to capture long-range dependencies in the data, which are commonly found in natural language sentences. This results in improved performance on a variety of NLP tasks, leading to its widespread use and achievement of state-of-the-art results on numerous benchmarks.

Treating chess game input as sequences, it is an excellent tool for learning from transformer-based language models because it presents a complex and challenging environment that requires the application of advanced language processing skills [4] [5]. By playing chess, language models can detect their ability to understand and analyze complex linguistic structures, strategic thinking, and problem-solving abilities. Additionally, the game provides a structured and well-defined set of rules and objectives, which makes it an ideal platform for testing the performance of language models in real-world scenarios [6] [7].

The use of chess as a learning environment for transformer-based language models presents a unique opportunity to advance the study and application of natural language processing techniques. Due to the precision and simplicity of the game, we are able to evaluate model predictions at a highly fine-grained level, assessing not only whether they match the ground truth, but also the legality of predicted moves given the current board state. Our evaluation methods and metrics, enable us to automatically analyze errors and evaluate models using counterfactual queries. Overall, the use of chess in conjunction with language models offers a valuable and robust platform for advancing research in this field.

In this paper, we aim to explore the challenges of language modeling in the domain of chess. Fig.1 shows a chess board in which it is white's turn to move. In order to generate a valid next move, the language model must (a) understand that it is white's turn, (b) represent the positions of all pieces on the board, both white and black, (c) choose a white piece that can be legally moved, and (d) make a legal move with the chosen piece. This requires the language model to not only track the state of the board and generate moves according to the rules of chess, but also to understand chess strategies in order to predict the best move. Overall, while chess may be a controlled domain, it presents significant challenges for language modeling.

The main contributions of our paper are:

- Use chess as a testbed for evaluating the world state tracking capabilities of language models provides valuable insights for the development and analysis of these systems.
- Utilize chess notation to probe the ability of language models to understand and reason about the world state, allowing for a systematic evaluation of their knowledge and understanding of the game and its underlying rules.
- Show that Transformer language models when provided with sufficient amounts of training data, are capable of

Identify applicable funding agency here. If none, delete this.

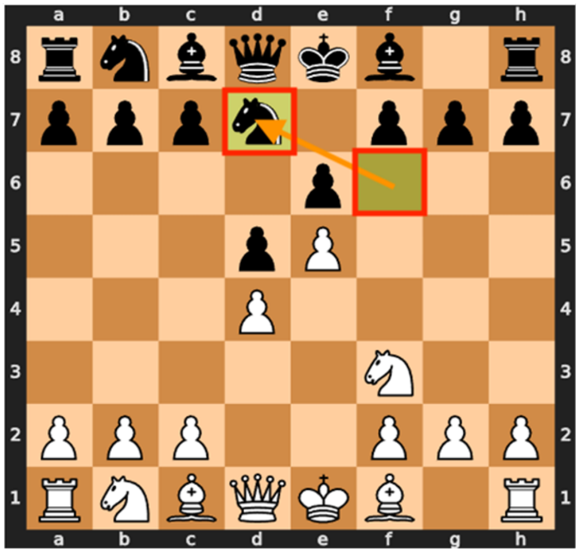

Fig. 1. UCI notation. Move Bishop from f6 to d7.

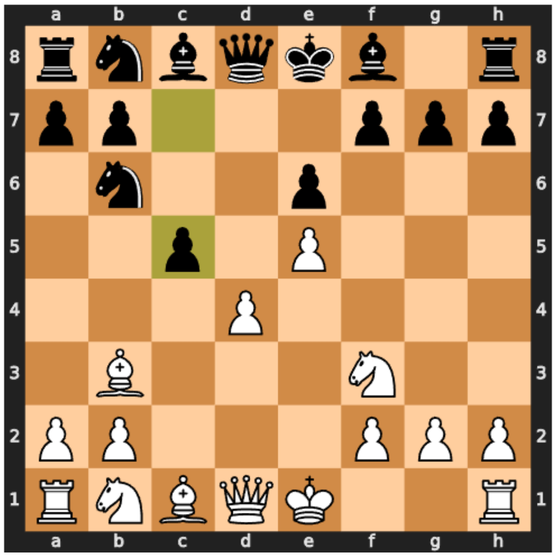

Fig. 2. The next actual move of a valid prompt ”c4b3 c7c5 d4c5 f8c5”

accurately tracking piece locations and predicting legal moves in various board games.

II. Methodology

A. Chess Notations

The standard notion choice for chess games is the Standard Algebraic Notions(SAN), where a move is denoted by a piece type and the corresponding ending square. Since SAN does not use starting square in the move representation, it increases the ambiguity and causes unnecessary complexity for the transformer model. For example, with the given prompt ”e4 e5 Nf3 Nc6 d4 h6 B”, the model has to infer which Bishop is moved. Due to the limitation of SAN, we represent moves using Universal Chess Interface(UCI), where a move is represented by the starting square and the ending square in a coordinate system. The move in Fig. 1 is represented as f6d7. f6 indicates the starting square and d7 indicates the ending square.

For training language models, we used a simple expression-based tokenizer to tokenize chess games represented in UCI notation. In this way, the move sequence ”e2e4 g1f3 c2c3” can be tokenized to a sequence of 6 tokens ”e2, e4, g1, f3, c2, c3”. This generates a total of 77 token types, including 64 squares, 6 pieces types, 4 promoted pawn Piece types and 3 other special symbols. We then train the auto-regressive model on these move sequences.

B. Ending Square Tasks

One intrinsic advantage of using UCI notation to train language models is that it allows us to probe the state-tracking ability of the trained model without an extra probing mechanism. To be more specific, when the model is provided with a prefix of move sequence as a prompt, we use the language model's next token prediction to evaluate how well the model learns the chess rules. For example, considering the prompt ”c4b3 c7c5 d4c5” in II-B, the actual last move is f8c5. Whether the model can predict exact c5 or predict other legal moves is a crucial indicator of the model's tracking board state ability, as well as its rule learning ability.

Two tasks are taken into account:

- End Actual Task: Given a move sequence prefix, the model is prompted with the next actual starting square, and then makes predictions on the ending square.
- End Other Task: Given a move sequence prefix, the model is prompted with any piece that can be legally moved as starting square, and then make predictions on the ending square.

The evaluation of the End Actual task is conducted on both exact move accuracy(ExM) and legal move accuracy(LgM). Exm reflects whether the model can predict the exact same ending square as the actual testing sequence, while LgM concerns about whether the model can perform legal ending squares. For LgM evaluation, we also consider the LgM R-Precision [8], where R means the total number of current state legal ending squares. For example, if there are 2 legal ending squares, only the top 2 predictions will be evaluated for accuracy testing. For End Other tasks, the concept of exact moves is not applicable.

C. Randomly Annotated Piece Type

As we mentioned before, unlike SAN, UCI notation does not tell us what color nor which type of piece is being moved. Therefore, when the model predicts the ending square depending on maximum likelihood, it is true that we can probe its understanding of the board state, but it is difficult to determine whether the model masters the rule and strategies of chess games. For this reason, we consider a more direct approach to probe the trained model's state-tracing ability.

TABLE I
ACCURACY AND PRECISION ON ENDING SQUARE TASKS WITH DIFFERENT PROBING TECHNIQUES

Train Size	Notation	LgM Actual	LgM Actual R-Precision	Lgm Other	LgM Other R-Precision	Exm Actual
200k	UCI	96.8	84.27	88.4	79.5	49.5
200k	UCI+RAP	95.3	84.45	87.3	80.13	48

TABLE II
ACCURACY AND PRECISION ON ENDING SQUARE TASKS

Train Size	Notation	LgM Actual	LgM Actual R-Precision	Lgm Other	LgM Other R-Precision	Exm Actual
200k	UCI	96.8	84.27	88.4	79.5	49.5
150k	UCI	93.9	83.68	86.5	79.35	45.0
100k	UCI	96.1	84.75	86.6	78.66	47.2
50k	UCI	92.3	80.27	82.8	75.89	41.3
15k	UCI	64.6	54.98	60.8	52.5	23.0

TABLE III
ACCURACY AND PRECISION ON STARTING SQUARE TASKS USING RAP

Train Size	Notation	LgM Actual	LgM Actual R-Precision	Lgm Other	LgM Other R-Precision	Exm Actual
200k	UCI+RAP	99.4	99.4	99.3	99.25	88.9

We use randomly annotated piece type(RAP) probing technique to conduct starting square tasks. Before each UCI move, there is a p percent probability that the piece type will also be generated. For example, a 100 percent RAP training sequence looks like " N, g1, f3, N, b8, c6, P, d2, d4, P, h7, h6", where N means next move will be operated on a Knight piece. When it comes to testing data, we use a prompt ended with a piece type and expect the model answers to us with a complete and valid move, composed of a starting square and an ending square. Note that there is no piece type information in the testing prefix, only a piece type token at the end of the prompt for the purpose of testing. Based on the prediction of the language model, we can find out whether the model is aware of the positions of that piece type.

Similarly to ending square tasks, there are also two variants in starting square tasks:

- Start Actual: Given a sequence prefix, the model is prompted with the piece type of the actual piece moved next.
- Start Other: Given a sequence prefix, the model is prompted with the piece type of all piece types that can be legally moved next.

For the Start-Other task, only LgM is applicable.

D. Experiments

1) Data: In our experiments, we collect a large dataset of chess games in UCI notation using the Millionbase dataset, and preprocess it to filter out duplicate games and games with fewer than 10 moves or more than 150 moves. This requires knowledge of chess to properly parse the dataset at this step. The resulting dataset contains 2.5 million games, from which we extract 200K games for training, 15K for validation, and 15K for testing.

However, to explore the effect of different dataset divisions on the GPT2 model's ability to predict end squares, we train additional models by keeping the validation and test set sizes constant and changing the training set size (150K, 100K, 50K, 15K)

2) Model Details: In this work, we employ the GPT2-small [9] [10] architecture as our base language model. The GPT2-small model is a 12-layer transformer model with 12 attention heads and a 768-dimensional embedding size. The context size of the model is limited to 512 tokens, which is sufficient to cover the longest game in our training set. It should be noted that we only utilize the model architecture and all models are trained from scratch for our experiments.

To investigate the impact of varying dataset splits on the performance of transformer-based language models, we employed different data set divisions to train, validate, and test multiple models using the GPT2 model. By observing the ability of different trained models to accurately predict the end square of a given move as the size of the data set varies, we sought to gain insights into the expected error involved in training these models and to identify optimal strategies for addressing the dataset split. Our findings have implications for the use of transformer-based language models in acquiring domain-specific knowledge and may inform future research in this area.

In this work, we also use LSTM language models with different numbers of layers as our baseline. LSTM networks are composed of LSTM cells, which are units that can learn to selectively retain or forget information from the previous time steps. Our LSTM models are trained with an embedding size of 768 and a hidden size of 768. Dropout (0.2) is also used in the training procedure to regularize the model and prevent over-fitting.

TABLE IV
ACCURACY AND PRECISION ON ENDING SQUARE TASKS WITH BASELINE MODELS AND OPTIMAL MODEL

Train Size	Model	LgM Actual	LgM Actual R-Precision	Lgm Other	LgM Other R-Precision	Exm Actual
200k	GPT2	96.8	84.27	88.4	79.5	49.5
200k	RNN(layers=3)	83.5	70.11	78.4	68.06	37.8
200k	RNN(layers=4)	82.9	70.62	76.7	65.93	38.8

TABLE V
COUNTS FOR ERROR SQUARE ESTIMATION

Model	Actual			Other		
	Syntax	Path Obstr.	Pseudo Leg.	Syntax	Path Obstr.	Pseudo Leg.
GPT2	1	28	15	2	36	76
RNN(layers=4)	120	30	20	130	26	76

3) Training Details: In this work, we trained our models for 10 epochs using a batch size of 60. Validation was performed at the end of each epoch, and training was stopped when the validation loss started to increase. For optimization, we used Adam [11]with a learning rate of $5x10^-4$ and L2 weight decay of 0.01. The learning rate was warmed up linearly over the first 10% of training, followed by a linear decay. To improve training speed, we used mixed precision training [12]. All experiments were conducted using the PyTorch Lightning framework and the transformers library.

III. RESULTS

In this paper, we first present language modeling results to demonstrate the effectiveness of RAP method. Next, we evaluate our model on board state probing tasks and show that it can learn to track pieces and predict legal moves with high accuracy when trained on a large training set. Finally, we compare the performance of our model with approximate attention transformer architectures and LSTMs on the probing task, and show that these models do not perform as well as our proposed base model.

A. Language Modeling

Table I presents the accuracy and precision of ending square tasks using different probing techniques. As we introduce piece type information into our model, the accuracy of legal moves and exact moves decreases compared to the non-RAP notation method. However, the R-precision increases when the new piece type information is added to the training set.

B. Board State Probing

Table II and Table III shows error analysis on ending square tasks and starting square tasks.

1) Perform Legal move Ability: The legal move accuracy and legal move actual R-Precision in TableII shows that, given enough data, the transformer model is able to infer legal moves perfectly. While for limited data set, the accuracy drops to an disappointing percent of 64.6. It is also observable that for small data set size, increasing train size yields an significant rise in all aspects of accuracy and precision. The results for Exm accuracy is not as perfect as Lgm. However this fact does not contribute to how well the model processes chess strategy, for the reason that the exact move in testing sequence is not necessarily the only good move in a certain play.

2) Tracking Pieces Ability: The legal move accuracy and legal move actual R-Precision in TableIII indicates that in UCI+RAP starting squares mission, the model's performance is very satisfying, with high accuracies and precisions around 99 %. Such results strongly prove that the transformer model can correctly track pieces locations along the game. Consistently with intuition, R-Precision is significantly lower than LgM accuracy. Since training data emphasizes the essence on the exact moves, it is reasonable for transformer model to underestimate other legal moves.

C. Comparison with Baseline

Table IV presents a comparison of the performance of different models on the end square prediction task. As shown in the table, the RNN model, which serves as our baseline, achieves an LgM Actual of 83.5% (for 3 layers). In comparison, the GPT3 model performs significantly better, achieving an LgM Actual of 96.8%, which is a relative improvement of 13.3%. However, the ExM Actual of the GPT3 model is limited, reaching a maximum of 38.8% (for 4 layers). In contrast, the GPT3 model achieves a higher ExM Actual of 49.5%. These results suggest that the GPT3 model outperforms the RNN models in terms of LgM and ExM Actual.

D. Error Analysis

Table V shows the four errors type that we found in ending square missions.

- Unreachable locations: The testing ending square can never be reached given the starting square.
- Syntax error: The testing ending square can never be reached given the prompt piece type.
- Path Obstruction: The testing ending square can never be reached due to the obstruction of other pieces.
- Illegal move causing King getting checked.

The different number of errors indicates that GPT2 model is better than RNN models in the ability of learning how to perform legal move and 100 times fewer in syntax error counts.

IV. Conclusions

We use the game of chess as a testbed for evaluating the ability of language models to capture underlying world states. By utilizing appropriate chess notation, we can use simple prompts to probe the performance of language models in regard to different aspects of the chess board state. The well-defined and predictable dynamics of chess allow for the training of models with varying levels of explicit state information, as well as the evaluation of model predictions at a fine-grained level. Our results indicate that transformer language models are capable of accurately tracking the state of the chess board when provided with sufficient data, but lacking board state information during training would decrease model performance in cases of limited data availability.

References

[1] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In NeurIPS.

[2] Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In NAACL.

[3] Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165.

[4] Toshniwal, Shubham and Wiseman, Sam and Livescu, Karen and Gimpel, Kevin, "Learning Chess Blindfolded: Evaluating Language Models on State Tracking," arXiv preprint arXiv:2102.13249.

[5] Presser, S.; and Branwen, G. 2020. A Very Unlikely Chess Game. https://slatestarcodex.com/2020/01/06/a-veryunlikely-chess-game/.

[6] Cheng, R. 2020. Transformers Play Chess. https://github.com/ricsonc/transformers-play-chess.

[7] Noever, D.; Ciolino, M.; and Kalin, J. 2020. The Chess Transformer: Mastering Play using Generative Language Models. arXiv:2008.04057.

[8] Manning, C. D.; Raghavan, P.; and Schutze, H. 2008. Introduction to Information Retrieval. Cambridge University Press.

[9] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In NeurIPS.

[10] Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language Models are Unsupervised Multitask Learners. In Technical Report, OpenAI.

[11] Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. In ICLR.

[12] Micikevicius, P.; Narang, S.; Alben, J.; Diamos, G.; Elsen, E.; Garcia, D.; Ginsburg, B.; Houston, M.; Kuchaiev, O.; Venkatesh, G.; and Wu, H. 2018. Mixed Precision Training. In ICLR.

V. Specific contributes per team member

Nuo Wang: data preprocessing, run (15k-150k) experiments, write report

Cheng Wan: data preprocessing, run (RNN+RAP) experiments, write report

Chunyuan Deng: data preprocessing, run (200k+AP) experiments, write report