ANLP - Project Report
Exploring what happens inside a Chess-LLM

Silvano Vento Maddonni
University of Trento - 247370
silvano.maddonni@studenti.unitn.it

Abstract

In this project we test how a LLM performs in computer chess, an area that doesn't directly belong to language, and explore and try to understand how it is internally reasoning. Starting from a gpt2 model trained on a chess datasets, we first adapt the model to predict the next move, making it able to potentially play a game. Then we perform some experiments on whether the model is only memorizing text patterns or actually building a sort of internal representation of the game (while not having any knowledge about it). We find out that the Chess-LLM is able to predict the next move with decent accuracy, while also providing legal moves with very high probability. On the probing tests, we find some interesting insights on how the attention mechanism values the different input parts and how the model knows where pieces are situated.

1 Introduction

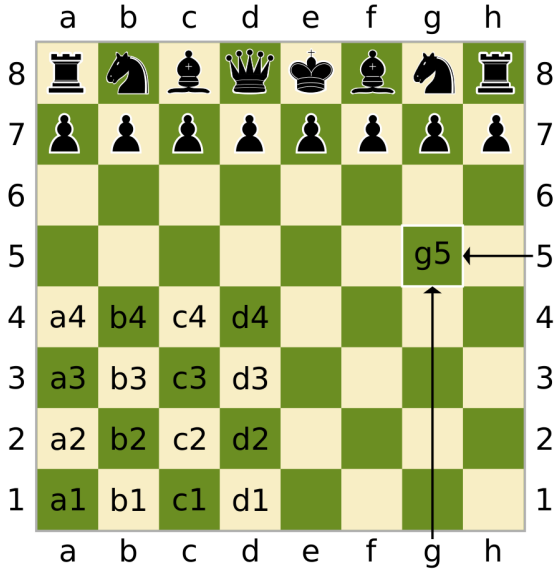
Language models are now used for a variety of applications that are not, or not directly, related to Natural Language Processing tasks. Given the black box nature of the models and the complexity of natural language, it is challenging to measure how accurately they represent the world state underlying the text.

We consider the task of language modeling for the game of chess. We investigate whether a language model is able to learn characteristics of the game of chess only from the records of games played by humans. Unlike natural language, chess notations describe a simple, constrained, and deterministic domain.

The project is composed of three parts. In part one, after handling the conversion and tokenization of the data, the model is set up to be able to predict the next move and a probing is conducted on its ability to keep predictions inside legal moves. In the second part, the focus is shifted towards probing the model by analyzing different attention heatmaps of the layers and heads. In the last part, a dedicated probing classifier is trained on top of the model to check the ability of the model to have an internal understanding of the positions of the pieces.

2 Chess as text

Chess games and moves can be transcribed in many different ways. Algebraic notation is the standard method for recording and describing the moves in a game of chess. It is based on a system of coordinates to uniquely identify each square on the board.



Square Naming

The sequence of moves composing a game are generally encapsulated using the Portable Game Notation (PGN).

```
1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O
   Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8
   10. d4 Nbd7 11. c4 c6 12. cxb5 axb5 13. Nc3
```

We represent moves using Universal Chess Interface (UCI) notation, which combines the starting

square and the destination square to represent a move.

```
e2e4 e7e5 g1f3 b8c6 f1b5 a7a6 b5a4 g8f6 e1g1
f8e7 f1e1 b7b5 a4b3 d7d6 c2c3 e8g8 h2h3 c6b8
d2d4 b8d7 c3c4 c7c6 c4b5 a6b5 b1c3
```

While the SAN notation is the standard choice for gameplay, we prefer UCI as it allows for pieces to be referenced unambiguously via their starting square, and each move is always a string of 4 characters. As drawback, we lose information of the piece type on the text.

2.1 Tokenizer

The games represented in UCI notation are tokenized using a simple regular expression based tokenizer, which considers a board square symbol such as b1 as a single token. This produces a vocabulary of 77 token types, which includes the 64 squares, piece type symbols, and other special symbols.

Type	Examples	Count
Square names	`e4, d1`	64
Piece type	`P, K, Q, R, B, N`	6
Promoted	q, r, b, n	4
Special symbols	BOS, EOS, PAD	3
Total		77

Model Vocabulary

For example, the sequence `"e2e4 e7e5 g1f3"` is tokenized into `"e2, e4, e7, e5, g1, f3"`.

3 Part One: Chess-LLM

The starting model used is a GPT2-small architecture. GPT2-small is a 12-layer transformer model with 12 attention heads and an embedding size of 768 dimensions. The model is pre-trained on 250K chess games taken from a full dataset of 2.9 million quality chess games. The whole training was done using UCI notation.

3.1 Next Move Prediction

To predict the next move, an input sequence representing the game played so far is passed through the model, and then the next two tokens are predicted and glued together. The final move is composed of two prediction because the tokenizer tokenize starting square and final square separately. While it's hard to evaluate how good the predicted moves are or what is the level of this chess-llm-bot, some surface analysis showed that most of the time it was predicting moves that were acceptable.

```
PGN seq:
1.Nf3 f5 2.b4 Nf6 3.Bb2 e6 4.b5 d5 5.g3 Bd7 6.a4
a6 7.c4 dxc4 8.Nc3 axb5 9.axb5
UCI seq:
g1f3 f7f5 b2b4 g8f6 c1b2 e7e6 b4b5 d7d5 g2g3
c8d7 a2a4 a7a6 c2c4 d5c4 b1c3 a6b5 a4b5
LM plays: a8a1
(In the game it was: a8a1)
```

3.2 Board State Probes: Legal Moves

One attractive property of having a language model trained on chess games represented in UCI notation is that the notation itself allows us to probe the trained model's state tracking abilities.

Even if the LM predicts a different move from the one played in the actual game, or a different move from what would be the best, we can still get valuable information from the prediction by checking if it was a legal move.

Instead of asking the model to predict the full next move, we can prompt also the starting square of the next move. The language model's next-token prediction (after consuming the prompt) can be interpreted as the ending square predicted, which can be used to determine the level of board state awareness of the model.

To test this, for more than 4K games, the game was chopped at random points, generated all the possible legal moves from that position, and fed the model with the prefix (first half of the move). The model was then asked to predict the ending square of the moves. The final ratio was an impressive 96.4% over 200K moves, showing that transformers can learn to predict legal moves. Tests on smaller datasets from different ELOs have been conducted, with the ratio being around 95%.

4 Part Two: Attention Heatmap

4.1 Weights, Heads, Layers

To deeply dig into how the LM internally is processing information, we explore the attention weights and how the transformer handles them.

Attention weights represent how much the model is focusing on each part of the input sequence when processing a particular token. In other words, attention weights indicate the relative importance that the model assigns to different tokens when generating the next token or understanding the context.

By visualizing attention, we can see which tokens in the sequence influence the model's prediction for the next token. In our context, analyzing a chess move sequence, attention weights could

show which previous moves the model considers most relevant when deciding on the next move.

Attention weights are computed for each layer and each head in the model, offering a multi-dimensional view of how the model processes the sequence.

- **Head** refers to one of the multiple attention mechanisms (attention heads) used in a multi-head self-attention layer of the transformer model. Each attention head independently focuses on different parts of the input sequence to capture various relationships and features.

- **Layers** refer to the stacked layers in a Transformer model. Each layer consists of multiple components, including multi-head self-attention mechanisms. As we move up the layers in the model, the representations become more abstract. Lower layers capture more localized, syntactic information, while higher layers tend to capture more global, semantic information. Each layer refines the representation of the input based on the information processed by the previous layers.

4.2 Heatmaps

Attention heatmaps are visual representations of where the neural network focuses its attention when processing input data. These heatmaps provide insight into how the model interprets and prioritizes different parts of the input.

The y-axis represents the "query" tokens (the current token being processed), the x-axis represents the "key" tokens (the tokens being attended to). More intense colors (red/dark red) indicate higher attention weights, lighter colors (white/yellow) indicate lower attention weights.

What it is interesting to look for:

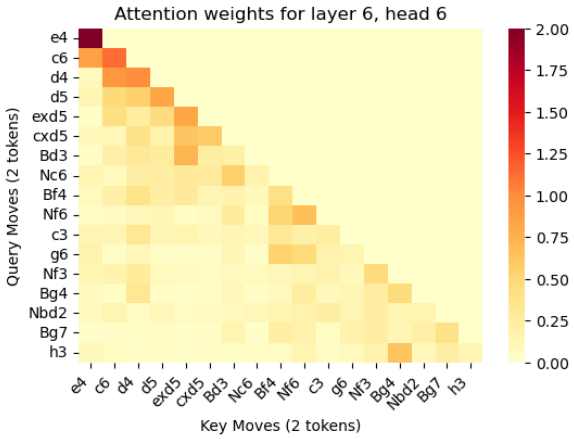
- Patterns of attention (e.g., attention to recent moves, or to specific types of pieces)

- Changes in attention patterns across different layers

- Any unexpected or interesting focus of attention that might give insights into how the model is processing the chess game.

Patterns Analysis

The main diagonal often shows high attention as tokens frequently attend to themselves, however



Attention Heatmap

some interesting different patterns were discovered, some of which are shown in Figure 1.

In theory, lower layers should focus on basic, local patterns and immediate interactions, such as pieces movements and short-term tactics while higher layers should capture more abstract, global concepts, like strategic planning, long-range piece interactions, and game flow.

The model is composed of 12 hidden layers, each with 12 heads. By analyzing each layer and each head, we were able to spot some particular characteristics.

- In the lower levels, layers 1 and 2, the attention is more vague and doesn't focus on anything in particular, neither shows specific motifs. (Fig. 1 - a)

- In the layers 3 and 4, among different heads, we see an alternate of either strong attention only the main diagonal or some sparsity of medium/high on different moves. This could reflect the model trying to catch some relations by exploring connections with various tokens. (Fig. 1 - b)

- In the middle layers (5-6) we see the reinforced diagonal: attention on the main diagonal and the closest few tokens. The model is focusing attention on the last move played and the ones close to it (so not only on the last move). Note: even though this reinforced diagonal, that is also present a lot on higher layers, only looks at the previous few moves, feeding the model with only the last 5 moves rather than the whole sequence drops performance very badly, so the whole sequence con-

cept is still important and captured by others parts. (Fig. 1 - c)

- In layers 6-7-8 the attention focuses on vertical patterns, less definite and polarizing than later layers, but starting to analyze the importance of single moves. (Fig. 1 - d)

- In the last 4 layers, we find a more decisive scheme in the model attention mechanism. It either focuses on the reinforced diagonal or on some specific high vertical targets dominating all the attention space. The latter can be of two kinds: alternate or single. In the alternate, the attention is high every two moves (each of 2 tokens), as this reflects moves of one player. In the single, we have a dense uniform column, where the model is exploring the impact of that move on both player's next moves. (Fig. 1 - e/f)

- These particular combinations of layers-heads very often showed similar behavior in many different samples: [3-11] = sparsity of highs, [5-6] = reinforced diagonal, [8-1] = double vertical dominating interest, [11-9] = single column dominating interest.

What can be inferred from the analyzed patterns?

The model seems to do a more general analysis of the input sequence in the lower layers, where it also doesn't give much importance to tokens in the end of the sequence and instead focuses only on the parts in the beginning. The reinforced diagonal appearing from the middle layers to the end reflects the model predicting new tokens based on the ones in proximity, but treating each token/move in the same way. The vertical patterns, emerging especially on the final layers, are where the model is really exploring particular tokens impact. Note: taken singularly, these wouldn't make too much sense, as the final prediction cannot be made by focusing only on one move/token, but by exploring the various finals layers and heads we found out that there are multiple of these vertical single columns in different parts, suggesting that at the end every single token gets isolated and processed individually (always with regard of the rest of the sequence) and then the results are merged and shaped with the whole general analysis of the early layers and reinforced diagonal (local connectivity of moves) from the middle part.

```
Final Prediction = [1 to 4] + [5 to 7] + [8 to 12]
[layers 1-4] = General and Sparse
[layers 5-7] = Reinforced Diagonal
             (local connectivity of moves)
[layers 8-12] = Single moves attention
```

5 Part Three: Pieces Location

Probing Internal Model Representations

A standard tool for measuring a model's internal knowledge is a linear probe. We can take the internal activations of a model as it is predicting the next token, and train a linear model to take the model's activations as inputs and predict board state as output. A linear probe has little capacity, and the state of the board is not a linear function of the input. If the linear probe accurately predicts the state of the board, then it demonstrates that the model transforms the input into a linear representation of the board state in its activations.

Given the input game sequence that translate into a chess board with 64 squares, we aim to predict the piece-state of each square using a linear probe. For each square, the probe is designed to classify it into 1 or 0 if the analyzed piece is on the square or not.

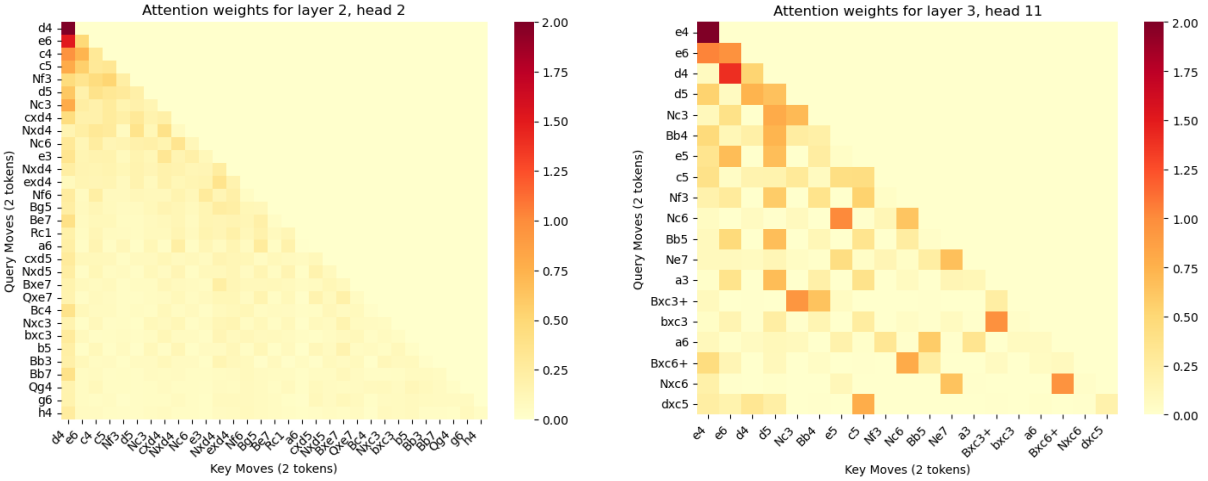
To do that, a PiecePositionProbe is created. It is a linear classifier model with a sigmoid that outputs a 64 elements tensor with probabilities for each square. To use the linear probe, we take the previous model with 'frozen' weights and extract the features from a particular layer (often the last hidden layer). Then, we train the linear classifier on top of these features to see how well it can perform on this task.

The probe is 768x64 because we take the last hidden layer of the model (model hidden size = 768) and not the output.

To train the probe, the input sequence (UCI format) is converted into a board state from which a 64 elements array is extracted with zeros and ones indicating the presence of the selected piece. The loss is calculated between this and the output of the probe.

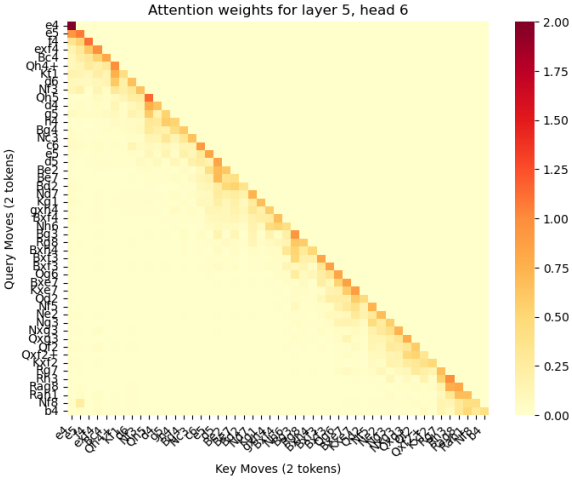
Piece	Base	Trained Probe
Pawn	0.54	0.82
Rook	0.55	0.94
Knight	0.51	0.97
King	0.45	0.93

Classifier Probe

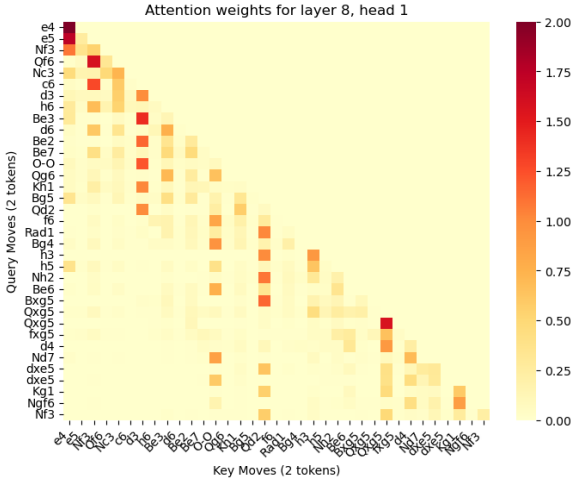


(a) Vague Low

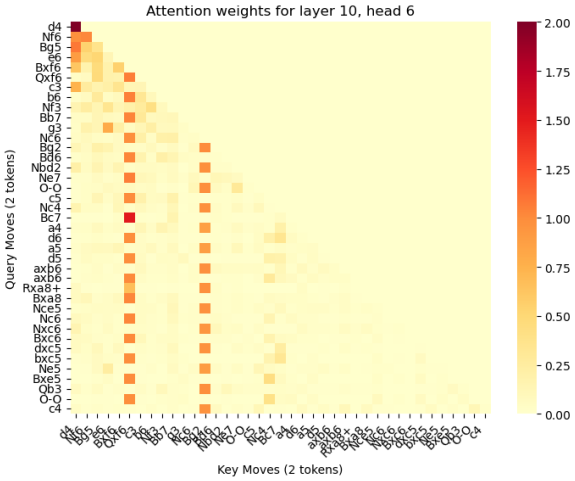
(b) Sparse Highs



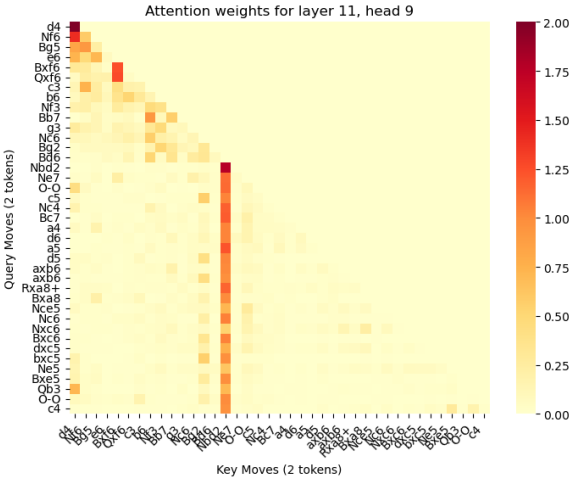
(c) Reinforced Diagonal



(d) Vertical Patters



(e) Alternate Dominating



(f) Single Dominating

Figure 1: Different Attention patterns across the various layers.

References

Xidong Feng, Yicheng Luo, and Ziyan Wang et al. 2023. Chessgpt: Bridging policy learning and language modeling. arXiv:2306.09200v2.

Bowen Jiang. 2022. Building a natural language chess engine with pretraining and instruction fine-tuning.

Adam Karvonen. 2024. Emergent world models and latent variable estimation in chess-playing language models. arXiv:2403.15498v1.

Andreas Stockl. 2021. Watching a language model learning chess.

Shubham Toshniwal, Sam Wiseman, Karen Livescu, and Kevin Gimpel. 2022. Chess as a testbed for language model state tracking. Toyota Technological Institute at Chicago.