LEARNING CHESS BLINDFOLDED

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer language models have made tremendous strides in natural language understanding. However, the complexity of natural language makes it challenging to ascertain how accurately these models are tracking the world state underlying the text. Motivated by this issue, we consider the task of language modeling for the game of chess. Unlike natural language, chess notations describe a simple, constrained, and deterministic domain. Probing for the world (board) state can be done simply via language modeling prompts. Additionally, we have access to a vast number of chess games coupled with the exact state at every move, allowing us to measure the impact of various ways of including grounding during language model training. Overall, we find that with enough training data, transformers can learn to track pieces and predict legal moves when trained solely from move sequences. However, in adverse circumstances (small training sets or prediction following long move histories), providing access to board state information during training can yield consistent improvements.

1 INTRODUCTION

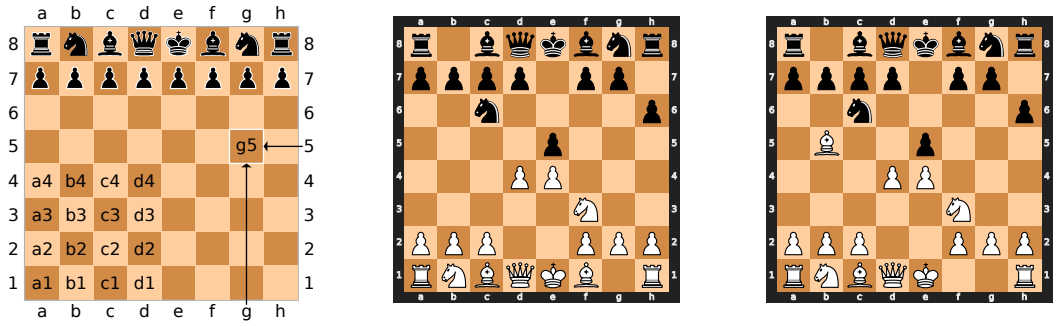
Recently, transformer-based language models such as GPT-3 have stretched notions of what is possible with the simple self-supervised objective of language modeling, becoming a fixture in state of the art language technologies (Vaswani et al., 2017; Devlin et al., 2019; Brown et al., 2020). However, the black box nature of these models combined with the complexity of natural language makes it challenging to measure how accurately these models represent the world state underlying the text.

Motivated by the above issues, we propose training transformer language models for the game of chess. Chess provides a simple, constrained, and deterministic domain where the exact world state is known. Also, chess games can be transcribed exactly and unambiguously using chess notations (Section 2). In fact, the form of chess notations allows us to probe our language models for aspects of the board state using simple prompts (Section 3).

Another benefit of working with chess is that we have access to nearly unlimited data that is coupled with the exact board state at every turn. This board state is a form of grounding for the move sequence and allows us to compare training on move sequences alone to training with access to varying amounts of explicit state. Thus, modeling chess using language models may have implications for the debate surrounding the ability of language models to capture meaning if only exposed to text (Bender & Koller, 2020). To test the impact of chess board grounding on learnability and data efficiency, we can train language models with varying degree of access to the board state (Section 4).

Due to the simplicity and precision of chess, we can evaluate language model predictions at a more fine-grained level than merely comparing it to the ground truth. For example, even if the next move prediction doesn't match the ground truth move, we can still evaluate whether the move is legal given the board state, and if it is illegal, we can determine the reason why. Moreover, since the state can be exactly modeled, we can evaluate models using counterfactual queries as well. The proposed evaluation sets and metrics are described in Section 5.3.

Finally, while chess represents a controlled domain, it is by no means trivial for a language model. To illustrate the challenges of language modeling for chess, consider the left board shown in Figure 1b, where white is next to move. In order to generate a valid next move, the language model needs to (a) infer that it is white's turn, (b) represent the locations of all pieces, both white and black, (c) select one of the white pieces which can be legally moved, and finally (d) make a legal move with the selected piece. Thus, a language model has to learn to track the board state, learn to generate moves according



(a) Square naming

(b) Board state before (left) and after (right) the bishop at `f1` is moved to `b5`. UCI notation represents the move as `f1b5`.

Figure 1: Chess Notation

to the rules of chess, and on top of that learn chess strategies to predict the actual move and not just any valid move.

We find that when given enough training data, transformers can learn to both track piece locations and predict legal moves at high levels of accuracy. However, when testing predictive ability for long move histories or when only given small training sets, predictive ability suffers. In these more challenging settings, we show that providing parts of the board state (during training time only) can lead to significant improvements in accuracy.

Predicting actual moves made by human players, which requires tracking board state but also involves the complexities of chess strategy, represents a much more challenging set of prediction problems. Our results here suggest that only part of this challenge is due to limitations of transformer LMs in representing board state. When provided oracle board state information during inference, the results are only slightly better than models without any state information.

Overall, our results indicate that transformers are able to learn the rules of chess when given enough data, but in adverse circumstances (small training sets or prediction when given long move histories), providing access to board state information during training can yield consistent improvement.

2 Background

Chess Preliminaries. Figure 1a[1] shows how squares are indicated in chess notations via a combination of lettered columns and numbered rows. Chess notations use this square naming convention to denote the movement of pieces. As our notation, we choose Universal Chess Interface (UCI) notation, which combines the starting square and the destination square to represent a move.[2] The move in Figure 1b is represented as `f1b5` in UCI where `f1` indicates the starting square and `b5` denotes the ending square. While another notation, SAN, is the standard choice for gameplay, we prefer UCI (see Appendix A for our motivation for choosing UCI over SAN).

Related Research. There have been several prior efforts in relating simulated worlds to natural language. The bAbI framework simulates a world modeled via templates to generate question-answering tasks (Weston et al., 2015). Other work has developed 3D world simulations and studied language learning (Hermann et al., 2017; Hill et al., 2017). There have been several recent efforts focusing on transformer language models for chess (Presser & Branwen, 2020; Cheng, 2020; Noever et al., 2020). However, these works focus mostly on the quality of game play while we focus on evaluating the board state tracking capabilities of language models.

[1]Source `https://en.wikipedia.org/wiki/File:SCD_algebraic_notation.svg`
[2]For more details see `https://en.wikipedia.org/wiki/Universal_Chess_Interface`

3 LANGUAGE MODEL PROMPTS AS BOARD STATE PROBES

One of the goals of this work is to evaluate the board state tracking capability of language models. A potential solution is to train a probing model on top of a pretrained language model (Adi et al., 2017; Tenney et al., 2019). This setup is time-consuming as it requires training probing models for all tasks. Also, the complexity of the probing model can affect the conclusions (Pimentel et al., 2020).

Chess notations provide a simple alternate solution of using language model prompts as board state probes. For example, the prompt "e2e4 e7e5 g1f3 b8c6 d2d4 h7h6 f1" (the underlined move sequence leads to the left board state in Figure 1b) can be used for next-token prediction with a language model trained on UCI notation. The generated token can be interpreted as the ending square predicted for the bishop at f1. This prediction can then be used to determine the level of board state awareness of the model. A prediction of g1 may indicate that the model does not recognize that the piece type at f1 is a bishop, as such a move is not possible for a bishop. If the model predicts g2, it may mean that the model is not aware that another piece is currently located at g2.

UCI notation is sufficient for assessing ending square prediction. However, it does not allow us to test starting square prediction directly. That is, we would like to give a language model the prompt "e2e4 e7e5 g1f3 b8c6 d2d4 h7h6 N", where N represents knight, and expect it to generate a valid starting position for a knight of the correct color. To allow testing the model with such prompts, we propose randomly including piece types in moves during training with some fixed probability p. We refer to this strategy as "randomly annotated piece type" (RAP) and use the nomenclature "UCI + RAP p" to indicate that with probability p, piece type is part of the move notation during training. Note that for $p = 0$, the notation reduces to UCI. When *testing* with these starting square prediction prompts, we only include piece type for the prompt, not for any moves in the history.

4 LANGUAGE MODELS AND VARIANTS

We use the GPT2-small architecture for our base language model (Vaswani et al., 2017; Radford et al., 2019). GPT2-small is a 12-layer transformer model with 12 attention heads and an embedding size of 768 dimensions. The context size of the model is limited to 512, which is sufficient to cover the longest game in our training set. Note that we only borrow the model architecture; the models themselves are trained from scratch. We use a simple regular expression based tokenizer, which considers a board square symbol such as b1 as a single token. This gives us a vocabulary of 77 tokens which includes the 64 squares, piece type symbols, and some other special symbols (see Table 4 in the Appendix).[3]

We next discuss variants of the base language model that incorporate the board state in various ways.

4.1 RANDOMLY ANNOTATED PIECE TYPE (RAP)

One way of introducing an aspect of board state into move sequences is by randomly annotating piece types (RAP) in training sequences, as introduced in Section 3. We hypothesize that adding piece types during training can aid the model in learning to track the board state by providing additional supervision. Similar ideas have been used when training memory networks for tasks involving entity tracking in natural language (Weston et al., 2014; Hill et al., 2015).

4.2 INCORPORATING BOARD STATE

One of the criticisms sometimes leveled against language models for natural language is that they can never understand the "meaning" of language, particularly given that they do not use any extra-linguistic information, such as visual grounding (Bender & Koller, 2020). The simple and deterministic dynamics of chess means that chess offers nearly-unlimited data with perfect grounding. The ground truth board state can be used to learn a grounded language model, and can also be used in oracle settings which use this board state during both training and inference. We next describe details of the board state representation followed by its incorporation in the base language model.

[3]In initial experiments we used a space token to indicate a move boundary. However, removing the space did not hurt performance and made training much faster due to reduced sequence length.

4.2.1 BOARD STATE REPRESENTATION

Following Oshri & Khandwala (2015), we represent a chess board state as an 8×8 image with six channels (see Figure 2 in the Appendix). The six channels correspond to the six unique piece types in chess, namely, pawn, rook, bishop, queen, king, and knight. The channel for a given piece type has a value of +1 at locations of its white instances, and -1 at locations of black instances, with 0 indicating empty positions. Apart from the piece positions, we add a seventh channel to represent additional information about the board state. In particular, we use one bit to indicate which player moves next, one bit to indicate if the current player's king is in check, and another four bits to represent castling rights of the two players.[4] These six bits are represented as another channel by padding with zeros to make it an 8×8 grid. Thus, we have a $7 \times 8 \times 8$ tensor for the raw board representation.

The raw board representation is fed to a 2-layer convolutional neural network (CNN). Both CNN layers have 128 filters of kernel size 2×2 with ReLU activations. The output of the CNN is passed through a linear layer which outputs a final 768-dimensional board state representation.

4.2.2 MULTI-VIEW TRAINING

We use the board state representation during training to learn a grounded language model. Let $(\boldsymbol{h}_t^i)_{i=0}^{12}$ represent the transformer language model's hidden states at timestep t, where the $\boldsymbol{h}_t^0$ are simply the embeddings for the input symbols plus the position embeddings. Then, the language model's state representation $\boldsymbol{l}_t$ is a learned convex combination of the transformer hidden states:

$$\boldsymbol{l}_t = \sum_{k=0}^{12} \alpha_k \boldsymbol{h}_t^k \qquad \boldsymbol{\alpha} = \text{softmax}(w_0, w_1, \cdots, w_{12})$$

where $\boldsymbol{w} \in \mathbb{R}^{13}$ is a learned parameter vector.

Given a minibatch of tokenized chess games, let K be the total number of moves in this minibatch. Let $(\boldsymbol{l}_i)_{i=0}^{K}$ represent the language model state representations at the end of each move in the minibatch, and let $(\boldsymbol{s}_i)_{i=0}^{K}$ represent the corresponding board state representations. We want the two representations to be close to each other when they correspond to the same state, and far apart otherwise. To promote this behavior, we use a "quadruplet" loss (Chen et al., 2017):

$$\mathcal{L}(\boldsymbol{l}_i, \boldsymbol{s}_i) = \max(0, m + d(\boldsymbol{l}_i, \boldsymbol{s}_i) - d(\boldsymbol{l}_i, \boldsymbol{s}_i')) + \max(0, m + d(\boldsymbol{l}_i, \boldsymbol{s}_i) - d(\boldsymbol{l}_i', \boldsymbol{s}_i))$$

where m represents the margin by which we want the positive pairs to be closer than the negative pairs, $d(.,.)$ represents the distance function (in our case, cosine distance), $\boldsymbol{l}_i'$ is a language model state which corresponds to a board state other than $\boldsymbol{s}_i$, and $\boldsymbol{s}_i'$ is a board state which corresponds to a language model state other than $\boldsymbol{l}_i$. For sampling a negative state given the other view's state, we randomly draw 10 states from the minibatch and pick the one closest to the other view's state. For margin, we use $m = 0.6$ as it gets the lowest validation perplexity among $\{0.2, 0.4, 0.6, 0.8\}$. Finally, the quadruplet loss is simply added to the language modeling loss without any scaling.

4.2.3 ORACLE BASELINE

For the oracle baseline we assume that the language model has access to both the piece type with which moves are made (e.g., `Bf1b5` instead of `f1b5`) and the board state at the end of every move during both training and inference. This setting limits the set of moves to consider, and relieves the language model of the burden of board state tracking. To incorporate the board state, we simply sum the board state with the embedding layer hidden state of the transformer at the end of each move (and not for previous tokens in the move to avoid giving the model enough information to cheat. Since the board state is available during inference, the language model can just focus on local transitions. We therefore limit the attention window to just the previous three tokens.[5] This oracle baseline provides an approximate upper bound in performance for all of the model variants we consider.

[4] Details on castling rights: `https://www.chessprogramming.org/Castling_Rights`

[5] We observe faster convergence by limiting the window size.

Table 1: Examples of each probing task, as well as the corresponding exact move (EM) and legal move (LM) correct answers, are shown below. All examples assume the language model was fed the prefix `e2e4 e7e5 g1f3 b8c6 d2d4 h7h6` (see Figure 1b), and that the actual next move was `f1b5`. While there is only one valid prompt token for both End-Actual and Start-Actual tasks, there are many valid prompt tokens for the other tasks, and we show just one possibility for each. Start-tasks (bottom sub-table) assume the model was trained on games described in UCI+RAP notation.

Task	Prompt Token	Correct Answers (EM)	Correct Answers (LM)
End-Actual	`f1`	{`b5`}	{`e2, d3, c4, b5 ,a6`}
End-Other	`f3`	N/A	{`d2, g1, h4, g5, e5`}
Start-Actual	`B`	{`f1`}	{`f1, c1`}
Start-Other	`N`	N/A	{`f3, b1`}

5 Experimental Setup

5.1 Data

We use the Millionbase dataset which is freely available and has close to 2.9 million quality chess games.[6] After filtering out duplicate games, games with fewer than 10 moves, and games with more than 150 moves (for the complete game to fit into one transformer window), we are left with around 2.5 million games. From this filtered set we randomly select 100K games for training, 15K games each for dev and test, and another 50K games to create prompt-based evaluation sets described in Section 5.3. The dev and test set are used for calculating perplexities on them. The dev set perplexity is used for choosing the hyperparameters for models which incorporate board state. From the 100K training set, we create subsets of size 15K and 30K which we refer to as "Train-S" and "Train-M", while the full training set is referred to as "Train-L". For detailed statistics, see Table 5 in Appendix. All the data processing steps requiring chess knowledge, including parsing chess databases, are carried out using python-chess (Fiekas).

5.2 Training Details

Models are trained for 10 epochs with a batch size of 60. Validation is performed at the end of every epoch and training is halted whenever the validation loss starts increasing. For optimization we use Adam (Kingma & Ba, 2014) with learning rate of 5×10^{-4} and L2 weight decay of 0.01. The learning rate is warmed up linearly over the first 10% of training followed by a linear decay. To accelerate training, we use mixed precision training (Micikevicius et al., 2018). All experiments are carried out using the PyTorch Lightning framework which is built on top of PyTorch (Falcon, 2019; Paszke et al., 2019). We use the GPT-2 implementation from the transformers library (Wolf et al., 2019).

For the UCI + RAP p models, we try $p \in \{0.01, 0.05, 0.15, 0.85, 1.00\}$, and find $p = 0.05$ and $p = 0.15$ get the best perplexity on the dev set. Larger values of p may lead to greater mismatch between training and inference, while small values like $p = 0.01$ likely do not provide enough training signal.

5.3 Board State Probing Tasks

In this section we describe how we probe the language model's board state understanding. In each probing task we feed the model a prefix of a game followed by a single prompt token, and the model is evaluated based on the highest probability next-token under the model given this context. We show an example of each probing task in Table 1 (which we further describe below), assuming the model has been fed the move sequence prefix `e2e4 e7e5 g1f3 b8c6 d2d4 h7h6`, which is visualized as the left board in Figure 1b. The actual next move played in the game is `f1b5`, which takes the white bishop at square `f1` to square `b5`, as shown in the right board of Figure 1b.

To create these evaluation sets, we use the 50K unique games reserved for this task (Section 5.1). We only consider prompts for non-pawn pieces since the dynamics of pawns are fairly limited. We

[6]Download link available at `https://rebel13.nl/rebel13/rebel%2013.html`

ensure that the game prefixes selected are never seen in the training data. To see the effect of game prefix length l (measured in number of moves) on these predictions, we create two classes of this task: (i) Short, where $5 \leq l \leq 50$, and (ii) Long, where $51 \leq l \leq 100$. For both classes we create evaluation sets with 1000 instances. Next we describe the various probing tasks in detail.

5.3.1 Ending Square Tasks

In this set of tasks, the model is given a game prefix and prompted with the starting square of the next move (`f1` in the example of Table 1). The model's next-token prediction represents its prediction for the ending square of this move, which tests the model's ability to track the board state and follow the rules of chess, as well as strategic awareness.[7] We consider two task variants:

1. **End-Actual**: Given a move sequence prefix, the model is prompted with the starting square of the actual piece moved next in the game.
2. **End-Other**: Given a move sequence prefix, the model is prompted with the starting square of any piece on the board that can be legally moved according to rules of chess.

We evaluate End-Actual predictions in terms of both exact move (EM) accuracy (whether the model predicted the true ending square, `b5` in our running example) and legal move (LM) accuracy (whether the model predicted a legal ending square for the piece starting at the square in the prompt). EM accuracy evaluation is similar to the typical evaluation of language models on natural language data, while LM is less stringent and focuses on testing just the model's understanding of chess rules and the board state. Note that for End-Other, only LM accuracy is available. See Table 1 for examples.

5.3.2 Starting Square Tasks

In this category of task, the model is again given a game prefix, but prompted with just the piece *type* of the next move, such as `B` for bishop in the example in Table 1. The model's next-token prediction thus represents its prediction for where the prompted piece type currently is on the board. This task tests the model's ability to track pieces. [8] Note that only models which have seen piece types during training, i.e. "UCI + RAP" models, can actually be tested on this task. Also, no piece types are used in the game prefix (except for the oracle baseline). We again have two variants of this task:

1. **Start-Actual**: Given a move sequence prefix, the model is prompted with the piece type of the actual piece moved next in the game.
2. **Start-Other**: Given a move sequence prefix, the model is prompted with the piece type of any piece on the board that can be legally moved according to rules of chess.

We again evaluate Start-Actual predictions both in terms of EM accuracy (whether the model predicts the starting square of the piece actually moved next in the game), as well as in terms of LM accuracy (whether the model predicts the starting square of a piece with the correct piece type). For Start-Other, only LM accuracy is available; see Table 1 for examples.

6 Results

Due to brevity of space, perplexity results are presented in Table 6 of Appendix. Table 2 shows results when predicting starting squares and Table 3 shows results when predicting ending squares. We will divide our discussion of the results into those that pertain to board state tracking and those that pertain to modeling chess strategy.

6.1 Board State Tracking

There are several observations to note. First, **transformers can learn to identify where pieces are located**. This is shown by the "LM" accuracies in Table 3. UCI + RAP 15 can predict a valid starting

[7] Strategic capabilities of a chess language model are strongly tied to the quality of training games.

[8] In certain cases, this task also tests understanding of chess rules. For example, in Figure 1b only the rook at `h1` can be moved.

Table 2: Accuracies (%) for predicting starting squares ("Start-Actual" and "Start-Other" tasks)

Training Set	Model	Starting Square (Short)			Starting Square (Long)		
		Actual		Other	Actual		Other
		EM	LM	LM	EM	LM	LM
Train-S	UCI + RAP 5	71.3	88.8	83.2	58.6	69.5	65.4
	UCI + RAP 15	80.4	96.7	93.8	72.6	81.6	79.4
	Oracle Baseline	86.2	99.1	97.6	88.0	98.3	97.7
Train-M	UCI + RAP 5	78.8	95.1	92.4	70.9	80.9	78.4
	UCI + RAP 15	84.5	98.7	97.1	83.0	93.3	92.3
	Oracle Baseline	88.5	99.8	98.2	91.3	99.2	98.8
Train-L	UCI + RAP 5	86.8	99.7	97.9	88.6	98.0	98.3
	UCI + RAP 15	87.4	99.7	98.2	89.6	99.3	98.8
	Oracle Baseline	91.0	100.0	99.0	92.2	99.8	99.6

position of a piece at 99.7% accuracy for short histories and 99.3% accuracy for long histories. When asked to identify the location of a piece other than the one selected to be moved next, these accuracies drop only slightly, to 98.2% and 98.8%, respectively. However, this capability requires the large training set. For the medium training set and long histories, accuracy drops to 93.3%, and with the small training set, it drops further to 81.6%.

RAP 5 reaches the same starting position accuracy as RAP 15 for short histories, but for long histories there is a 1.3% gap. At smaller training sizes, the superiority of RAP 15 becomes much clearer, showing the benefit of providing piece type information more frequently during training.

The difference between the location of the piece in the exact move (EM) and the location of either piece of the given type (LM) is substantial, at more than 10% absolute. However, this difference relates to chess strategy rather than board state tracking. When querying a piece type other than the one moved, the accuracies are still high for RAP 15, at 98.2% for short histories and 98.8% for longer histories, though slightly lower than when querying the actual piece type. This trend suggests that piece location tracking is slightly better for the piece type that is actually moved.

Second, **transformers can learn to predict legal moves**. This is shown by the "LM" accuracies in Table 3, for which UCI + RAP 15 exceeds 99% for short histories. We do observe a drop to 95.1% for long histories, which is likely because board state tracking becomes more difficult with longer histories. When decreasing training data, the short history LM accuracy drops by 2% in each case, but the long history accuracy drops by 5%. It may be the case that training on an even larger training set can improve long-history accuracy further.

Overall, when enough training data is available, randomly adding piece types (with RAP 15) is similar to using the complete board state via multi-view training. The differences in LM accuracy are smaller than 1% when using Train-L (Table 3). With small training sets, the multi-view model struggles to use board states effectively, but with the large training set, the results are slightly better than RAP 15. Transformer LMs are robust to various ways of incorporating explicit supervision about the board state when given enough training data.

When predicting ending squares, we can use UCI, which corresponds to RAP 0. We find consistent gains in accuracy when randomly adding piece types during training. UCI + RAP 5 improves over UCI, and UCI + RAP 15 improves further in most settings, with the largest improvements being for the smallest training set. With Train-L, UCI is able to predict a legal ending square in 98.6% of cases when using short histories. This strong result indicates that a transformer without any form of board state supervision can still learn to track board state well enough to predict ending squares if given enough data. Even with long histories, the difference between UCI and UCI + RAP 15 is only 0.6%.

6.2 Chess Strategy Modeling

Overall, our results show that predicting the actual moves made by human players is much more difficult than identifying valid piece positions and legal moves.

Table 3: Accuracies (%) for predicting ending squares ("End-Actual" and "End-Other" tasks)

Training Set	Model	Ending Square (Short)			Ending Square (Long)		
		Actual		Other	Actual		Other
		EM	LM	LM	EM	LM	LM
Train-S	UCI	51.9	90.8	84.3	28.3	69.5	64.8
	UCI + RAP 5	53.4	94.4	86.5	32.5	78.7	73.6
	UCI + RAP 15	57.4	94.8	89.2	36.1	84.6	76.4
	UCI + Multi-view Training	51.4	90.2	85.1	27.7	71.0	65.8
	Oracle Baseline	59.0	96.3	89.9	39.2	89.6	85.2
Train-M	UCI	56.8	95.1	90.7	35.2	82.8	76.9
	UCI + RAP 5	59.8	97.0	91.0	39.6	88.9	81.3
	UCI + RAP 15	61.1	96.9	90.5	43.0	90.2	83.7
	UCI + Multi-view Training	56.8	95.9	91.7	35.7	83.3	78.6
	Oracle Baseline	62.4	97.6	91.3	45.9	93.1	88.8
Train-L	UCI	66.7	98.6	94.9	47.1	94.5	89.9
	UCI + RAP 5	69.0	98.9	94.8	50.3	95.6	91.7
	UCI + RAP 15	68.6	99.1	95.2	48.4	95.1	91.6
	UCI + Multi-view Training	69.7	99.2	95.8	49.3	95.9	92.3
	Oracle Baseline	67.6	98.9	94.4	52.3	96.8	92.4

We can view the exact move accuracy in starting square prediction (Table 2) as the ability to determine which piece to move next given the piece type. The Oracle Baseline EM results show the difficulty of this task even when the full board state is provided at inference time. With the large training set, we see gaps of 2-4% between the RAP models and the Oracle, while in terms of LM, which does not involve strategy, the gaps are much smaller. In general, the gaps increase with less training data and longer histories. This trend suggests that modeling chess strategy requires a more sophisticated board state tracking capability than that required for merely choosing a valid piece type position.

When predicting ending squares (Table 3), the relatively low EM accuracies of the Oracle Baseline show the difficulty of the task. The strongest non-oracle models—-RAP 5, RAP 15, and multi-view—-are able to match or exceed the Oracle Baseline when having access to the large training set and for short histories. This result shows that our models are not limited by their ability to track the board state over short histories, but rather by the inherent difficulty in predicting the next move performed by a human player. However, for long histories, the Oracle Baseline is consistently a bit more accurate than our models. As in starting square prediction, the gap between the oracle and our models increases as training data decreases and histories get longer.

7 Conclusion

We propose the task of language modeling for chess to evaluate how well language models can capture the world (board) state. The simple and precise dynamics of chess allow for (a) training models with varying amount of explicit state, and (b) evaluating model predictions at a fine-grained level. We show that with appropriate choice of chess notation, a language model can be probed for different aspects of the board state via simple prompts. Our results indicate that transformer language models are able to learn the rules of chess, as well as to track the board state when given enough data, but in adverse circumstances (small training sets or prediction when given long move histories), providing access to board state information during training can yield consistent improvement.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. In *ICLR*, 2017.

Emily M. Bender and Alexander Koller. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In *ACL*, 2020.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, 2020.

Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. Beyond triplet loss: A deep quadruplet network for person re-identification. In *CVPR*, 2017.

Ricson Cheng. Transformers Play Chess, 2020. URL https://github.com/ricsonc/transformers-play-chess.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019.

WA Falcon. Pytorch lightning. 2019. URL https://github.com/PyTorchLightning/pytorch-lightning.

Niklas Fiekas. python-chess: a pure Python chess library. URL https://python-chess.readthedocs.io/en/latest/index.html.

Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom. Grounded language learning in a simulated 3d world. *CoRR*, abs/1706.06551, 2017. URL http://arxiv.org/abs/1706.06551.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The Goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*, 2015.

Felix Hill, Karl Moritz Hermann, Phil Blunsom, and Stephen Clark. Understanding grounded language learning agents. *CoRR*, abs/1710.09867, 2017. URL http://arxiv.org/abs/1710.09867.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed Precision Training. In *ICLR*, 2018.

David Noever, Matt Ciolino, and Josh Kalin. The chess transformer: Mastering play using generative language models, 2020.

Barak Oshri and Nishith Khandwala. Predicting Moves in Chess using Convolutional Neural Networks. 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

Tiago Pimentel, Josef Valvoda, Rowan Hall Maudslay, Ran Zmigrod, Adina Williams, and Ryan Cotterell. Information-theoretic probing for linguistic structure. In *ACL*, 2020.

Shawn Presser and Gwern Branwen. A Very Unlikely Chess Game, 2020. URL https://slatestarcodex.com/2020/01/06/a-very-unlikely-chess-game/.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2019.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? Probing for sentence structure in contextualized word representations. In *ICLR*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *NeurIPS*. 2017.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merrinboer, Armand Joulin, and Tomas Mikolov. Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks, 2015.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rmi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

A SAN NOTATION

Standard Algebraic Notation (SAN) combines the piece type moved and the destination square to denote a move.[9] For example, the move in Figure 1b is represented as `Bb5` in SAN where `B` represents the piece type bishop and `b5` represents the destination square.

Standard Algebraic Notation (SAN) Ambiguity SAN notation doesn't use the starting square of the piece in its move representation. This limits the ability to prompt a SAN-based language model with specific piece type instances. For example, given the prompt "`e4 e5 Nf3 Nc6 d4 h6 B`" (the underlined move sequence leads to the left board state in Figure 1b), it's not clear whether the token `B` refers to the bishop at `f1` or `c1`. Due to this limitation on the specificity of probing queries, we do not use SAN for our experiments.

B MODEL VOCABULARY

Table 4: Model Vocabulary

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

Table 4 breaksdown the vocabulary used. Tokens of promoted pawn piece type are used when a pawn gets promoted. For example, `e7e8q` denotes the move where a pawn from e7 moves to e8 and becomes a queen.

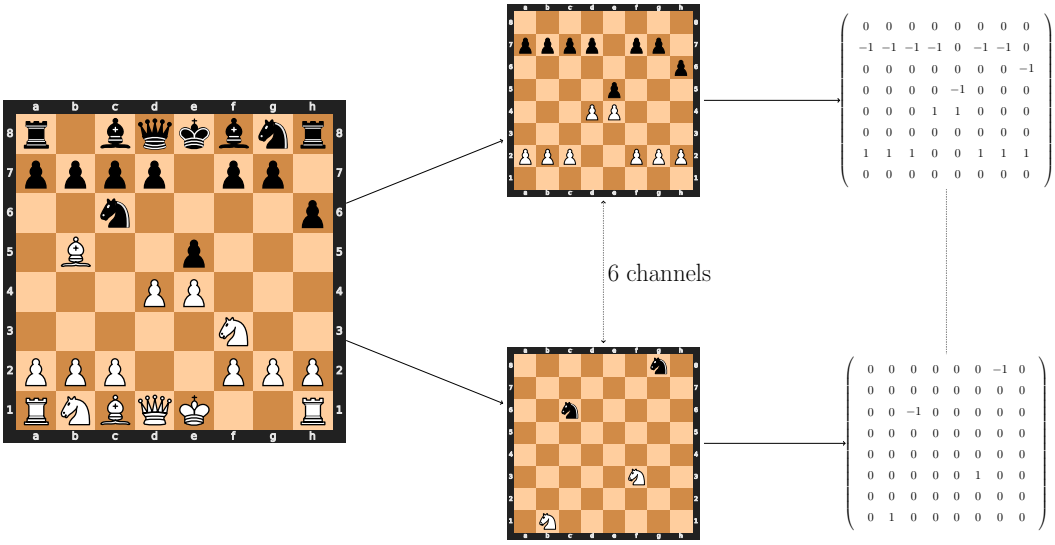


Figure 2: Raw board representation used as input to the board state model.

Table 5: Statistics of language modeling data

Split	# of games (in 10^3)	Total # of moves (in 10^6)	Avg. # of moves/game
Train-S	15	1.1	73.6
Train-M	30	2.2	73.5
Train-L	100	7.3	73.5
Validation	15	1.2	78.5
Test	15	1.2	79.2

Table 6: Canonical Dev and Test Perplexity (calculated by dividing the loss with total number of moves in the sequence)

Training Set	Model	Dev set	Test set
Train-S	UCI	3.21	3.2
	UCI + RAP 5	3.07	3.07
	UCI + RAP 15	3.07	3.08
	UCI + Multi-view Training	3.22	3.22
	Oracle Baseline	2.72	2.72
Train-M	UCI	2.83	2.83
	UCI + RAP 5	2.76	2.75
	UCI + RAP 15	2.79	2.79
	UCI + Multi-view Training	2.80	2.80
	Oracle Baseline	2.50	2.49
Train-L	UCI	2.24	2.24
	UCI + RAP 5	2.76	2.75
	UCI + RAP 15	2.44	2.44
	UCI + Multi-view Training	2.29	2.29
	Oracle Baseline	2.24	2.24

C DATA STATISTICS

D ILLEGAL MOVE ANALYSIS

We categorize illegal moves into three categories described below:

- *Syntax*: The predicted move can't be made by the piece type present at the starting square regardless of the board state. This error indicates that the model thinks that some other piece type is presented on the given starting square prompt.
- *Path Obstruction*: The move can't be executed because there are other pieces present in the path.
- *Pseudo Legal*: This is a very tricky category where the move is not legal because king ends up in check or remains in check if the predicted move is executed.

Table 7 presents results on analysis of illegal moves predicted by the model trained on the full training set for the ending square task for long game prefixes. We observe that most of the gains for the oracle baseline are coming from the pseudo legal error category. The oracle baseline benefits from the input representation explicitly encoding the information that the king is in check. While the models are pretty closely matches on the syntax error category, in the path obstruction category the clear winner is the multi-view training setting which suggests that the multi-view approach is better at modeling the spatial arrangement of the pieces.

[9]For more details see `https://en.wikipedia.org/wiki/Algebraic_notation_(chess)`

Table 7: Error Analysis of Illegal Moves for models trained on Train-L

Model	Ending Square (Long)					
		Actual			Other	
	Path Obst.	Syntax	Pseudo Legal	Path Obst.	Syntax	Pseudo Legal
UCI	21	3	31	34	8	59
UCI + RAP 5	18	1	25	26	6	51
UCI + RAP 15	23	2	24	31	7	46
UCI + Multi-view Training	14	0	27	17	5	55
Oracle Baseline	23	0	12	38	2	27