CSCI 49362 Final Project
Chess by Proximity: Playing Chess using Embeddings and N-grams

Sruly Rosenblat
Hunter College

Abstract

In this paper, I explore the capabilities of embeddings and N-grams to learn the rules and playstyles of chess. To do this I created two models each trained on the same corpus provided by Kaggle, I then cleaned the dataset, removed the outliers, and made the moves readable by Chess.py to aid in testing.

1 Introduction

1.1 Traditional chess playing programs

Chess-playing programs like Stockfish can play at levels human grandmasters can not. This is traditionally accomplished by looking at possible future moves and using something like the min-max algorithm to find an optimal move given an allowed maximum depth set by the programmer; this is often based on an elo rating.

This is very effective at finding the optimal move however it is less effective at simulating a human opponent. This is the case because its goal is not to find the most likely move but instead the best move within its look ahead range. While shrinking how far it could look ahead is somewhat effective in simulating someone of a lower elo, it's not ideal if the player's elo is not known.

1.2 Model Information

In this paper, I will explore the feasibility of an alternate system using just nlp tools. Specifically, I used a 5-gram and an L2 embedding search using a window of 32 for the word2vec model, and getting the mean of all moves embeddings to get the search vectors. I had both the N-gram and embedding models return a dictionary of moves and the amount of time it was seen following the moves in the sequence.

1.3 PGN notation explanation

Chess game moves are commonly represented in a format called Portable Game Notation or PGN. Each space has a unique name in this format that identifies its unique position on the board. The names are computed by taking their horizontal position where the leftmost square is ‘a’ and the bottom vertical position is ‘1’.

Additionally to deal with any ambiguity moves are often preceded by a letter representing the piece that is moved and sometimes part of the position it's moved from. In addition to storing just the raw move the PGN format also stores additional information about the state of the game; most notably a check is notated with a “+” at the end of the token and a checkmate is notated by appending ‘#’ to the end of the last move

PGNs are an especially useful format for NLP compared to other formats as every move could be easily broken up into its own token. In contrast, another popular format is called Forsyth-Edwards Notation commonly abbreviated as FEN, it greatly condenses the chess state into just a few characters on a single line; however, this comes at the cost of not storing how the state of the board was reached making it impractical to use to predict the next move as it would lose all the information

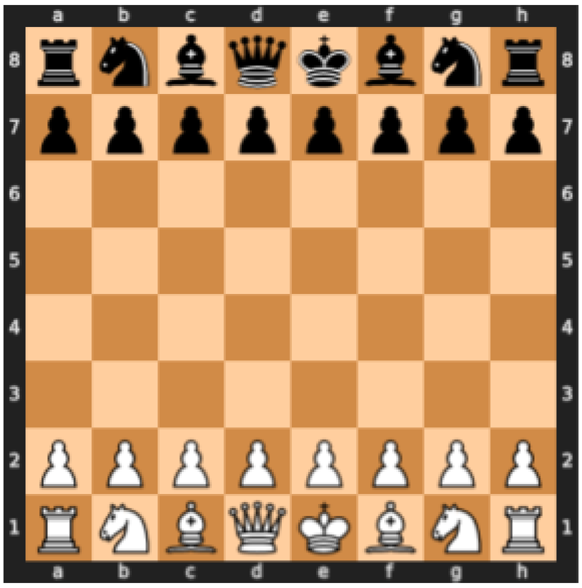


Figure 1a: a chess board with no moves taken

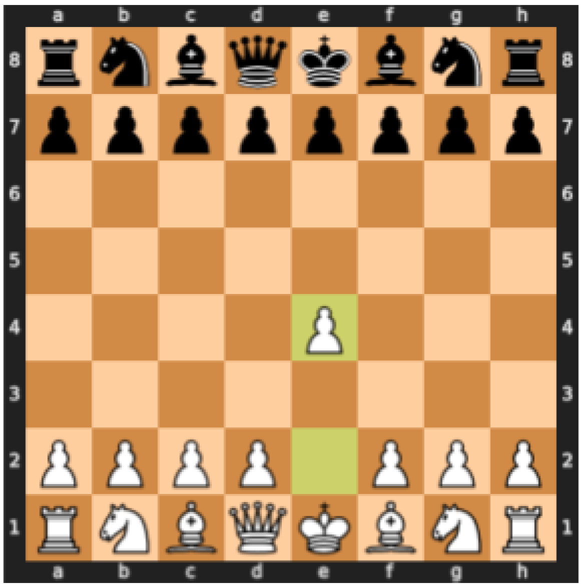
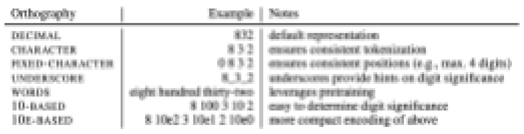


Figure 1b: a chess board with 'e4' taken

that could be gained from what moves were played throughout the different games.

2 Previous research

2.1 Investigating the Limitations of Transformers with Simple Arithmetic Tasks



Orthography	Example	Notes
DECIMAL	832	default representation
CHARACTER	8 3 2	ensures consistent tokenization
FIXED-CHARACTER	0 8 3 2	ensures consistent positions (e.g., max. 4 digits)
UNDERSCORE	8_3_2	underscores provide hints on digit significance
WORDS	eight hundred thirty-two	leverages pretraining
10-BASED	8 100 3 10 2	easy to determine digit significance
10E-BASED	8 10e2 3 10e1 2 10e0	more compact encoding of above

Figure 3: chart of results taken from *Investigating the Limitations of Transformers with Simple Arithmetic Task*

Chess has a lot of similarities to Arithmetic, as such I found this paper informative in the process of tokenizing the dataset. Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin discovered in their research that the way numbers are tokenized could significantly affect the reasoning skills of their model. Specifically, they found that by tokenizing in a way that the digits are separated they significantly improved their results. *s*

The chess equivalent of a digit seems to be a move as like with decimal digits each move could be seen as a single unit of information. While you could theoretically break a move down into further parts, it will greatly complicate the generation process for the models I'm using. With a transformer model, this may make more practical sense.

2.2 Watching a Language Model Learning Chess

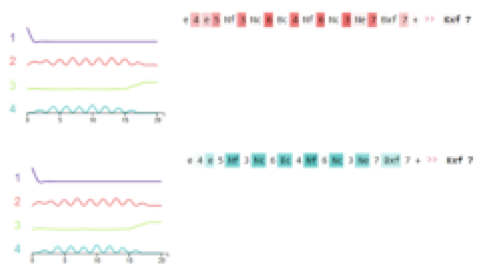


Figure 4: Image from *Watching a Language Model Learning Chess* depicting how moves are represented in a transformer model

In his paper Andreas Stockl looked at how a transformer model picked apart previous moves to decide what to play. This seems promising for my research even though the model architecture is quite different. He found that some layers of the transformer focused exclusively on the columns and some on rows see Figure 4 giving it a more complex representation than I am using for my embedding and N-gram model.

3 Data and Methodology

3.1 Data gathering

For the data I used the Lichess Chess Game Dataset from Kaggle. It included roughly 20 thousand games of different skill levels. It also included additional information about the games but I ultimately decided to just focus on the actual PGN game contained for each game.

20058 games	Total	Complete games	Incomplete Games
Checkmate	6325	6325	0
Draw	906	0	906
Resignation	12827	0	12827

Table 1: the data composition of the original dataset.

Because of a slight variation in how moves on a PGN could be represented, I first fed the games into chess.py to standardize the notation for the entire dataset. Additionally, to eliminate outliers, I limited the games to only those that had between 6 to 129 moves. Leaving longer games in the dataset skewed the probabilities because some of the longest games tended to repeat a lot.

3.2 Data Exploration

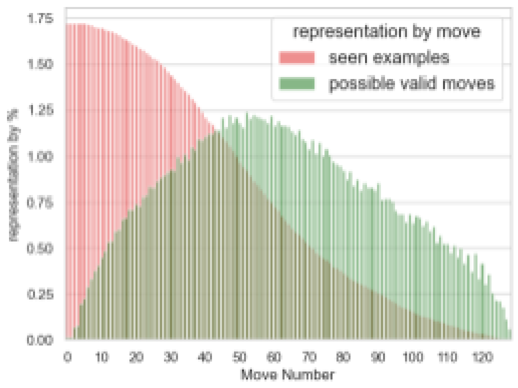


Figure 5: a graph of the dataset by move count and the number of possible moves

Because of the linear nature of chess games the first move is the most represented in the dataset followed by the second then the third leading to a curve of knowledge where the further into a chess match the fewer examples are available. The dropoff curve of the seen examples is represented in Figure 5 by the red part of the chart. Also represented in Figure 5 is the diversity of possible next moves at each move number.

The possible moves seem to peak at around 50-60 moves into the game; one reason for this could be that until this point in the game pieces are moving from their starting positions, opening up new options for the players to move. It also seems like after the peak more pieces start to get taken simplifying the board and making the valid moves trend down over time.

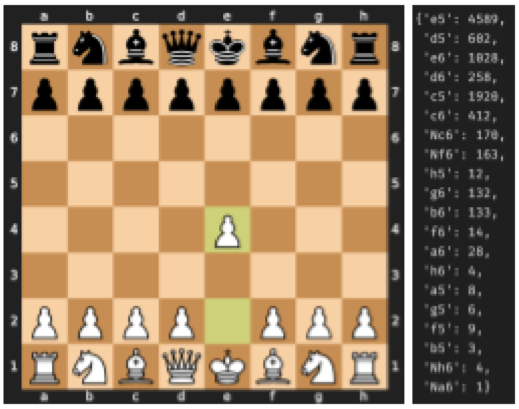
3.3 Methodology

18891 games	Train (80%)	Test (20%)
Games	15113	3778
Moves	874506	219527

Table 2: the train test split.

For this paper I have built two models, one is a 5-gram that I have used as a baseline to compare against. The second model is a set of two embedding indexes, one for black moves and one for white moves. Before training either of the models I split the dataset into two parts, one training, and one testing. The exact breakdown of the split is shown in table 2. The 5-gram is fed the games with the token “<s>” repeated 4 times so that the model could ‘learn’ how to predict the start moves. For the embedding model, the PGN is fed unmodified.

3.4 N-gram Implementation



```
{'e5': 4589,
 'd5': 682,
 'e6': 1028,
 'd6': 258,
 'c5': 1920,
 'c6': 412,
 'Nc6': 170,
 'Nf6': 163,
 'h5': 12,
 'g6': 132,
 'b6': 133,
 'f6': 14,
 'a6': 28,
 'h6': 4,
 'a5': 8,
 'g5': 6,
 'f5': 9,
 'b5': 3,
 'Nh6': 4,
 'Na6': 1}
```

Figure 6: an example of a dictionary item from the n-gram implementation

I implemented the 5-gram as a dictionary where the key is the moves leading up to the predicted move. The value is a dictionary containing all the moves that were seen following the 4 moves in the key along with their count. To train it I simply made a function that increased the count of a value linked to by a key in the dictionary. This is useful because then I just had to loop through the training games to ‘train” the model using the 4 preceding moves and incrementing the current move in the dictionary.

3.5 Word2Vec Embeddings Implementation

The model I used to make the embeddings was the word2vec model. I fed the model the training data plus a window size of 5 and a vector size of 32. In order to represent full games and not just individual moves I also added an average vector function that finds the mean of all the vectors in the game so far; note while the average vector function accounts for every move in the game it cant distinguish move order so ‘e4 e5 d4’ will be indistinguishable from ‘d4 e5 e4’ to the model.

3.6 Index implementation

In order to capture the differences between white and black moves I used 2 different indexes; one index that just has white moves and one for just black. I found this to be an acceptable solution due to the fact that in actuality the computer would need to play either white or black but not both. Separating them allows searching half the amount of vectors and should increase performance as there will be fewer places for error.

For the indexes, I used 2 flat L2 indexes from Facebook's Faiss library. For each, I added all moves in the training data corresponding to the move color. In the end, the white index saw 360269 moves, while the black move index saw 360452.

4 Baseline

	Probabilities
Probability Valid	0.15031562579979116
Probability Correct	0.0923156958230946 7

Probability Correct if Valid Percentage > 0	0.787501560672794

Table 3: probabilities of 5-gram model

For my baseline I used an N-gram model trained on the same data that my embedding model was. Specifically, I used a 5-gram model with the following stats, 15% probability the move is valid, 9.2% correct predictions, and a 78.7% correct rate when the move has a greater than 0% chance of validity. I define a valid move as a move that if played in a chess game would be legal. The correct prediction percentage is the probability that the move has if it's in the dictionary; if it's not in the dictionary the probability is considered 0 in my stats.

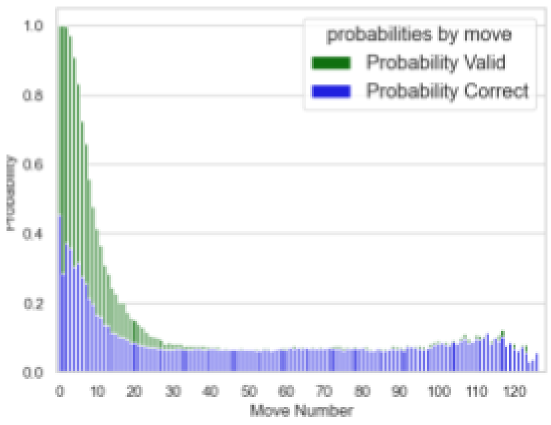


Figure 7: graph showing the N-gram probabilities by move.

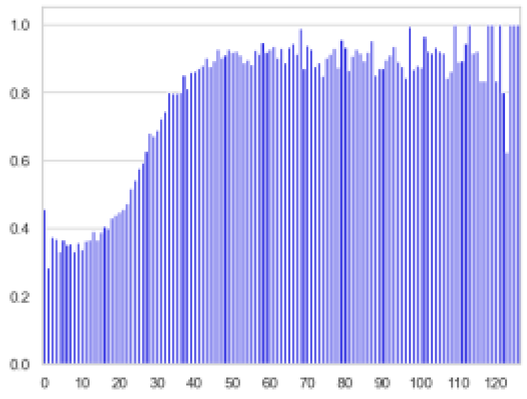


Figure 8: probability correct if already valid distribution by move

I thought this was a good baseline as while the embedding model takes advantage of looking at all moves the N-gram only looks at 5 moves at a time. This will help to distinguish how much info could be learned from just the last few moves vs looking at the whole game. The results seem to show that there is some understanding of chess being learned. There is also a high correlation of a valid move also being a correct prediction.

5 Results

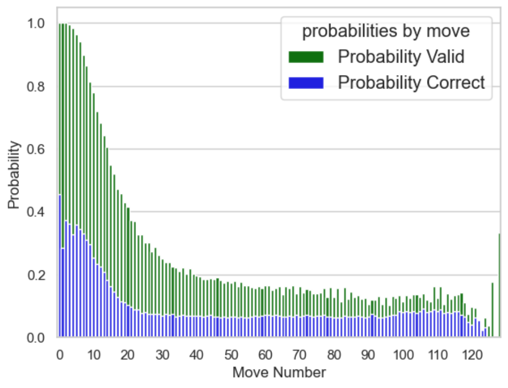


Figure 9: graph showing the Embedings probabilities by move.

When using the index models with a window of 5 and a vector of size 32 for the embeddings the results were an improvement over the baseline. The probability that a move is valid is 25.8% a ten percent improvement over the baseline. The probability that the move is predicted correctly went up 2% to roughly 9.8%. The chance that the move is correct given it predicted a valid move drops from 79% to 43% but this isn't a result of a drop in accuracy rather it seems to be that valid moves went up more than correct moves.

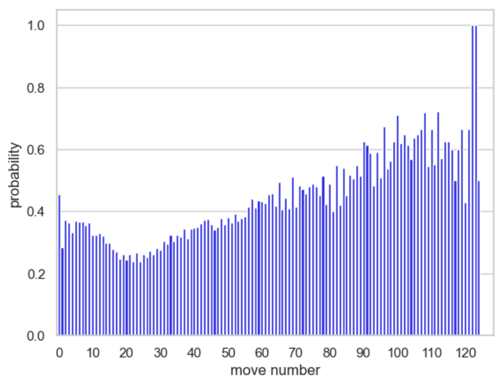


Figure 10: graph showing the Embedings probability correct if already valid distribution by move.

The curve in Figure 9 shows a very quick decline between moves 10 and 30 in the number of valid predictions for moves. This is similar to what happens between 10 and 20 in the N-gram probability chart in Figure 7. This may be related to the quick growth in the diversity of moves in Figure 5.

	Probabilities
Probability Valid	0.25847380269764053
Probability Correct	0.09774219577125763
Probability Correct if Valid Percentage > 0	0.43325046116069066

Table 4: probabilities of embedding model

6 Conclusion

In conclusion there does seem to be some amount of chess game knowledge in both the N-gram and embedding models however the knowledge drops off very fast as the game goes on. For the embedding model, the valid move predictions seem to stay stagnant near 20% for about 80 moves which seems promising to explore further. The N-gram also shows the same pattern where the valid predictions stay stagnant at 10% for a long time after the initial drop.

Overall, it would be very interesting to see how adding more chess game examples helps. If the baseline 20% starts rising with more games then it might be possible to solve the problems with using embedding models to play chess by just putting more examples in the dataset. While it may never be possible to get to 100% accuracy there is a lot of chess data available in online databases.

7 Future Work

In the future I would like to experiment with how big the embedding vectors need to be to get similar results, it may be possible to use smaller embedding vectors. I would also like to see how the results differ as the number of games seen grows; this will allow me to see if scaling up is a viable option. I would also like to experiment with finding a way to keep track of ordering with embeddings.

References

Andreas Stöckl. 2021. Watching a Language Model Learning Chess. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 1369–1379, Held Online. INCOMA Ltd.

Nogueira, R., Jiang, Z., & Lin, J. (2021). Investigating the Limitations of Transformers with Simple Arithmetic Tasks. *ArXiv*. /abs/2102.13019