LEARNING CHESS AND NIM WITH TRANSFORMERS

Michael DeLeo and Erhan Guven

Whiting School of Engineering, Johns Hopkins University, Baltimore, USA

ABSTRACT

Representing a board game's state space, actions, and transition model by text-based notation enables a wide variety of NLP applications suited to the strengths of language models. These few shot language models can help gain insight into a variety of interesting problems such as learning the rules of a game, detecting player behavior patterns, player attribution, and ultimately learning the game in an unsupervised manner. In this study, we firstly applied the BERT model to the simple combinatorial Nim game to analyze BERT's performance in the varying presence of noise. We analyzed the model's performance versus three agents, namely Nim Guru, a Random player, and a Q-learner. Secondly, we applied the BERT model to the game of chess through a large set of high ELO stockfish games with exhaustive encyclopedia openings. Finally, we have shown that model practically learns the rules of the Nim and chess, and have shown that it can competently play against its opponent and in some interesting conditions win.

KEYWORDS

Natural Language Processing, Chess, BERT, Sequence Learning

1. INTRODUCTION

Chess is one of the oldest board games and also one of the most researched computational problems in artificial intelligence. The number of combinational positions is around 10^43 with a branching factor of 35 and this makes the problem ultimately very challenging for even today's computational resources [1]. Current state of the art solutions to exploring chess are typically done by generating valid board positions and evaluating their advantage to win the game. Like an optimization problem, generating possible and promising positions is analogous to a feasible optimization surface and is built by a tree data structure representing each position reached from a previous position. Evaluating a position with a heuristic involves some knowledge of how a piece moves, their values, the position of the king, piece positions, pawn structures, the existence of a combination of moves that can lead to a forced mate and numerous other calculations to find the winning move or a combination of moves.

Stockfish, one of the best chess engines today, uses the minimax search with alpha-beta pruning to efficiently search through chess positions in order to identify the strongest branches of moves [2, 3]. By avoiding variations that will never be reached in optimal play, Stockfish is able to consider higher depths than an ordinary brute force state space search. This is essential to the engine's success as the game of chess has a large branching factor of 35 [1]. In other chess solutions this computationally search is avoided where it is possible to infer the outcome of the game, such as in a deterministic mating attack discovered by the search tree. IBM Deep Blue chess computer [4] used dedicated processors conducting tree searches faster on hardware to beat the world champion Garry Kasparov who is considered as one of the best human chess players ever. Alpha Zero [5] uses plain randomly played games without any chess knowledge but learns the moves from the game. A general-purpose reinforcement learning algorithm, and a general-purpose tree search algorithm are used to conduct combination of moves. The deep learning

DOI: 10.5121/ijnlc.2022.11501

engine learns the game as much as the hand-crafted knowledge injected in the Stockfish evaluation engine.

In this study, we evaluated and analyzed a different approach to learning chess, specifically using BERT transformer to learn the language of chess by observing the moves between players. We believe there is great potential in building a chess text corpus because of the unique advancements in few shot transformers where large networks and large amounts of data yield interesting results [14]. Language models extracted by the state-of-the-art models such as GPT-3 [6] are considered as few-shot learners. Among many other statistical information, a language model can involve rare patterns represented by only a few counts in a histogram which can be extracted by BERT transformers.

Our study evaluates the BERT language model starting from the initial board state s_0, and tests its understanding of a game's state space as the game tree increases in depth. In our study, we evaluate the BERT on two games, namely Chess and Nim. First, we analyzed the application of a language model to the simpler game of Nim due to its smaller state space, branching factor and action space allowing a complete analysis. We conducted experiments of Nim games with a series of agents such as a Guru player which plays through a mathematical approach, a random player blindly making valid Nim moves, as well as a Q-Learner model trained from Guru and random exploration. We save the results of each game, to be learned by the model. We then analyzed the model between a random, a Guru, and a Q-learner player with a controlled number of random games. And finally, we applied the model to the chess game by grandmaster games covering all possible openings from the chess opening encyclopedia [7].

A literature survey shows only a few very recent papers [8, 9] have applied NLP methods to chess but none of them used board/move text based on a grammar pattern to encode the game. As a novelty, the method in this paper encodes the game positions and moves in a specific text pattern based on Forsyth-Edwards Notation which is possibly easier to be learned than a full game in Portable Game Notation format [10]. Starting from the opening position, PGN conveys a position virtually between the moves without explicitly encoding. In a board game each position and move pair can be thought of a sentence passed to the other party. Thus, these sentences are learned by the language model, and they are somewhat order independent. The following sections will describe the NLP method and analyze its performance towards learning board games that use text representation of each position and move.

1.1. Chess State of the Art

Stockfish is under the GPL license, open source, and still one of the best chess programs making it a suitable candidate to teach a natural language model. There are two main generations of Stockfish which are the classical and NNUE variations. The latter is stronger of the two, and for the purposes of this research is what will be focused on. In this version, Stockfish relies on a neural network for its heuristic. NNUE stands for Efficiently Updateable Neural-Network [11]. This network is a lightweight fully connected neural network that gets marginally updated depending on the state space of the board, which is an optimization technique to improve its performance.

Alpha Zero is a deep reinforcement learning algorithm with a Monte Carlo Tree Search (MCTS) algorithm as its tree search algorithm [8]. MCTS is a probabilistic algorithm that runs Monte Carlo simulations of the current state space to find different scenarios [3]. An example of the MCTS being used for a game of tic tac toe is shown in Figure 1. Notice the tree branches off for various game choices. This is a critical component of the Alpha Zero model in that it allows it to project/simulate potential future moves and consequences. MCTS was chosen by the deep mind team as opposed to using an Alpha Beta search tree algorithm because it was more lightweight.

Alpha Zero is famous for beating Stockfish in chess with 155 wins out 1000 games. Stockfish won 6 games [8]. There is some debate however as to if more hardware would have helped

Stockfish. Nonetheless, the main advantage of Alpha Zero to Stockfish is that it is a deep learning model which can play itself millions of times over to discover how to best play chess. One of the impracticalities of it is that it is not open source however, and proprietary to DeepMind.

1.2. Chess Text Notation

In this study the chess notation is based on coordinate algebraic notation. This notation is based on the chess axes where a position is defined as s {a, b, ..., h} is the x axis, and {1, 2, ..., 8} is the y axis and represents two coordinates $\{(x!, y!,), (x", y")\}$. The first coordinate set represents the initial position, and the second set represents the position the piece moves to [10]. This notation is chosen to represent move states rather than other notations is because of its uniformity and how many tokens it would take to represent a full game position.

The Forsyth-Edwards Notation (FEN) is a notation that described a particular board state in one line of text with only ASCII characters [10]. A FEN sequence can completely describe any chess game state while considering any special moves. We use this notation to describe our board state in our experiments. An example of the FEN sequence is shown in Figure 1, this record represents the initial chess state.

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

Figure 1. Example FEN Sequence

1. Piece placement: each piece is represented by a letter r, n, b, q, etc. and the case indicates the player where uppercase is white, and lowercase is black. The rank of each section of piece is described by Standard Algebraic Notation (SAN) [10], and this describes the positions of those pieces.

2. Active color: represented by a “w” meaning white’s turn is next, and “b” meaning black’s turn is next.

3. Castling Availability: There are five tokens to represent castling availability, “-“ no one can castle, “K” white can castle king side, “Q” white can castle queen side, “k” black can castle king side, and “q” black can castle queen side.

4. En Passant.

5. Half Move Clock: Starts at zero and represents the number of moves since the last capture or pawn advance.

6. Full Move Clock: Starts at 1, increments after black’s move [10].

FEN is particularly useful because it provides a complete stateless representation of the game state in a single character sequence. This is possible because chess is a game where there are no unknowns, and everything represented visually is everything there is to the game space. These are the reasons for why we chose the game of chess and chose these notations for our experimentations.

1.3. Nim Game

Nim is a game of strategy between two players in which players remove items from three piles on each turn. Every turn the player must remove at least one item from exactly one of the piles. There are two different versions of the game goal: the player who clears the last pile wins or the player who has to take the last piece loses the game.

1.3.1. Game Space

A Nim experiment consists of three piles, with some quantity for each pile. The notation is described below where n is the quantity, and x_i is the ith pile [15].

$$S = \{(x_1, x_2, x_3) \mid x_i \in N_0, x_i \leq n, i = \{1,2,3\}\}\ for\ some\ n \in N$$

The action space is defined as a vector A with two components such that (α, β) ϵ A. Where α is the pile x_i to take from, and β is the amount taken. A visual representation of what the state space looks like when initialized for three piles, and ten items each is shown in Figure 2.

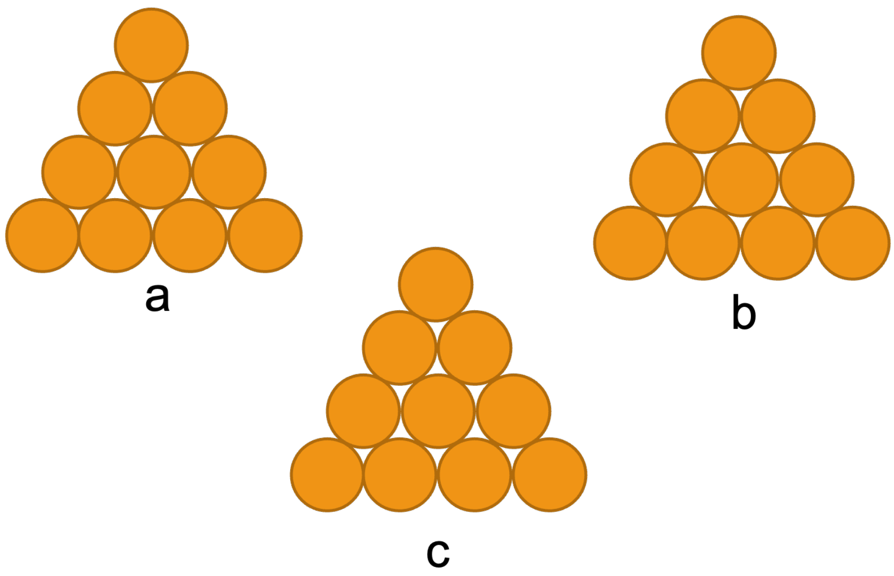

Figure 2 – NIM Example State Space

1.3.2. Guru

The primary agent in Nim is the GURU player, whom has a theoretical best strategy for a state s if such state s ϵ P or s ϵ N where P and N are defined as [15]:

$$P \cap S, N \cap S, P \cup N = S$$

With Conditions:

All terminal positions are P-positions.
From all N-positions, it is possible to move to a P-position.
From all P-positions, every move is to a N-position.

The optimal strategy of Nim is to make moves where the resulting state is in the P set. The authors of the Guru algorithm came up with the nim-sum operator ⊕:{0,1}^n x {0,1}^n→{0,1}^n, which is defined as [15]

$$[(x_1, x_2, \dots, x_n), (y_1, y_2, \dots, y_n)] \longmapsto (z_1, z_2, \dots, z_n)\ where\ z_i = \begin{cases} 1\ if\ x_i \neq y_i \\ 0\ if\ x_i = y_i \end{cases} = x_i + y_i (mod\ 2)$$

The Nim-Sum operator is essentially the XOR operator, and some state $s \in S, s = \{x_0, x_1, x_2\}$ can be represented in a binary format such that $x_i \in \{0,1\}^n, i = 1,2,3$ [15].

We need one more theory to show that the nim-sum operator is the optimal solution [15].

$$(x_1, x_2, x_3) \in P\ \leftrightarrow x_1 \oplus x_2 \oplus x_3$$

Algorithm 1 Optimal agent for an impartial game

Input: A percept, namely the current state $s \in \mathcal{S}$. A partition of $\mathcal{S}$ into $\mathcal{P}$-positions and $\mathcal{N}$-positions as defined above.

if $s \in \mathcal{N}$ **then**
  return $x \in Actions(s) \cap \mathcal{P}$ {Choose an action such that the resulting state is a $\mathcal{P}$-position.}
else
  return $x \in Actions(s) \subset \mathcal{N}$ {All of the actions belong to the $\mathcal{N}$-positions, choose any.}
end if

Figure 3 – Algorithm 1 Guru [15]

1.3.3. Q-Learner

The Q-Learner agent's goal is to achieve as good a policy as possible. This is done by obtaining the utility function from the Bellman equation [15]. The Q-Learner learns by relying on an exploratory agent (random) to properly explore the state space while the QTable is updated depending on the rewards and penalties of the game.

Algorithm 2 Q-learning agent

Input: A percept, namely the current state $s' \in \mathcal{S}$ and a reward r'.
Persistent:

- Q, a table of action values indexed by state and action, initially zero
- N_{sa}, a table of frequencies for state-action pairs, initially zero
- s, a, r, the previous state, action and reward, initially zero

if $Terminal(s)$ **then**
  $Q[s, None] \leftarrow r'$
end if
if $s \neq Null$ **then**
  $N_{sa} \leftarrow N_{sa} + 1$
  $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
end if
$s, a, r \leftarrow s', \arg\max_{a'} f(Q[s', a'], N_{sa}[s', a']), r$
return a

Figure 4 – Algorithm 2 Q-Learner [15]

1.4. BERT Model

The BERT model (Bidirectional Encodings for Representations of Transformers) is a language model which is designed to pretrain on bidirectional representations on unlabeled text by jointly conditioning on context from both the right and left sides [12]

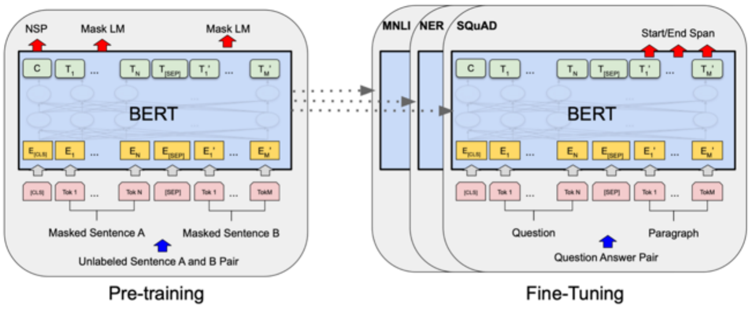

Figure 5. BERT Architecture [12]

The Bidirectional Encoder Representations from Transformers (BERT) model is a supervised model, that achieved state of the art on Q&A tasks before GPT. It's a lightweight, deep learning model that is trained to learn bidirectional representations of context in unlabeled text. The general architecture can be seen in Figure 2, and it should be noted that it is like GPT-1 in terms of its architecture and size [13].

2. Methodology

The objective of our study is to train a transformer model on text sequence datasets in such a way that it can learn to accurately play and understand the games. We apply the BERT model to both Nim and Chess. In this section we will lay our procedure for procuring the data for these experiments as well as our methodology for training the transformer.

2.1. Nim Data Collection

Our Nim experiment consisted of using three agents: a random player, a guru player, and a Q-learner. Each experiment is initialized to three piles, and ten items per pile (i.e. [10,10,10]). Two-hundred games are played for every combination of each agent so that every player's behaviors for every playstyle are explored. Additionally, there are ten tiers of randomness in play where the first tournament of players will have no randomness and the level of injected randomness increases for each subsequent tournament of players. The randomness threshold increases by 10% until we reach a 100% random tournament. **Figure 6** provides a visual of how we intended to collect the data. This process would be executing an exhaustive tournament with a given randomness threshold.

Figure 6 – NIM Data Collection

We can define the chance to play a random move as:

$$Rand < P_{rt}$$

Where:

P_{rt}: Our randomness threshold in the range of 0 to 1, with 0.10 increments

Such that when a random number is greater than the threshold, we play a random move for the player whose turn it is instead of allowing the player to make a move. Additionally, this move still gets recorded with the agent's identifier instead of the random agent's identifier. Following this process of injecting randomness is a suitable way for simulating noise in our Nim data collection.

When we save games such as in **Figure 6**, we need to store our state space, transition model, and action. We choose to represent the game in a stateless form where a given sequence has no directly stated connection to other states except for its own child state implicitly. By keeping our data stateless we can store it in a way where we can have a data store of unordered sequences. **Figure 7** illustrates this format where the three piles are the state space with a slash delimiter, followed by a player identifier which serves to inform a transition model for whose turn is it for this state space. Finally, after the player identifier the action is stored which represents the action the agent took for the sequence's state space.

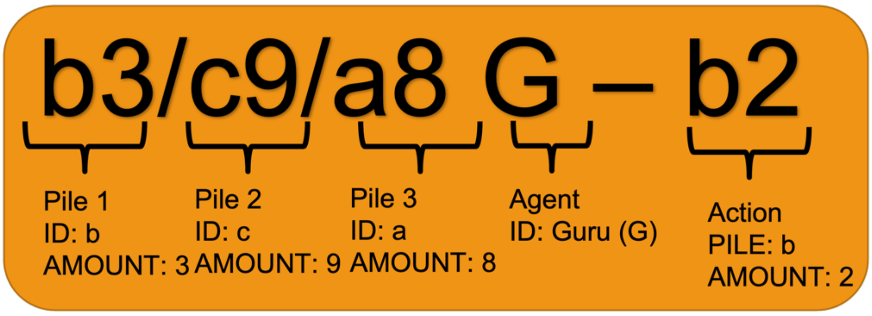

Figure 7 – NIM Sequence

When we approached the game of Nim we had to decide what grammar to give a Nim text corpus. The general vocabulary of our data is that there are three state identifier tokens (a, b, c), amounts of 0-10, and an agent tag (G, R, Q, X, W). The grammar of a sequence is that states should be separated by slashes, and that a move should have a dash before it. This helps the model to differentiate the purpose of the different words in the sequence.

In our experiments we set up two different experiments with the player identifier position. The first was to just input the player identifier, the second was to input some form of value for costs/rewards. The latter was indicated by a X/W indicating if the next move resulted in a win or not.

2.2. Chess Data Collection

The chess experiment uses Stockfish 14, and python 3.9. The Stockfish engine is configured to use NNUE, with one thread, default depth, and one for the value of MultiPV. The max depth that can be set is 20, however that slows the experiment down too much and so a value of 1 is chosen for the sake of getting a large dataset. Additionally, Stock- fish is set to an ELO rating of 3900. To gather a large amount of data, one million games of chess are played. A timeout for moves is set for 200 to discourage runaway stalemate games. The data collection activity takes about 4-6 days.

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1 [MOVESEP] f2f4
```

Figure 8. Example Chess Sequence

1. The basic routine of the program is to initialize a fresh game with the Stockfish engine, and the stated configurations

2. Select the best move from Stockfish and submit for each player until the game is over

3. Record the moves (FEN and algebraic coordinate) as they are selected and store

4. At six moves end the game, delimit each set of moves with the next line tokens and perform post-processing

An example of the data returned by a game of chess is shown in Figure 3 where each line consists of a FEN position, the player, and the next move chosen. This example is the opening chess board, followed by a separator token and a move for F2 to F4.

2.3. Pretraining BERT

For each experiment, once the data is generated, the BERT Word Piece Tokenizer is trained on the entire set of data such that it can get a full scope of the sequences. Since information is encoded into words and letters being capitalized, the tokenizer must accommodate for this. Therefore, the vocabulary includes capitalization.

We utilize the datasets hugging face library to load all our datasets and deliminate by the end of line token. Those datasets are tokenized and collated in parallel then split into training and testing sets with a 20% split. For the training procedure, we use the hugging face trainer with a 15% MLM (masked language modelling, refer to [12]) probability.

To provide inference with the model, a hugging face pipeline is used where the state sequence is provided and a [MASK] token is placed at the end where the move token would be. For example, a sequence for Nim would be a10/b10/c10 G – [MASK], where the pipeline would fill in the MASK token for the move.

2.4. Initial Analysis of BERT Model on Nim

As a few shot learner and as an unsupervised learner [6, 14] BERT language model can extract patterns that are expressed only a few times and in midst of very high noise. The following experiment used the language model trained using the games between Guru and random players. A number of games are played between the Guru which is a rule based player, and a random player generating random but valid moves. Since the Nim game outcome heavily depends on the first player move (like Tic Tac Toe), an equal number of games are played by swapping the first player. Each game start from a non- zero number of pieces in three piles, so that a Guru player can lose a game against a random player since it might be given a losing position in the first place. The number of possible positions or the feature space size is 113 (equals 1331) for three piles and 10 pieces to start the game. Theoretically one needs at least this many positions to fill up the feature space for a Guru to make a move so the game data would have at least one sample of every board position and winning (or the "right") move by Guru. Note that for the Q-learner, such a learning approach takes close to 300k games (against a random player) to be able to be on par with a Guru player [15].

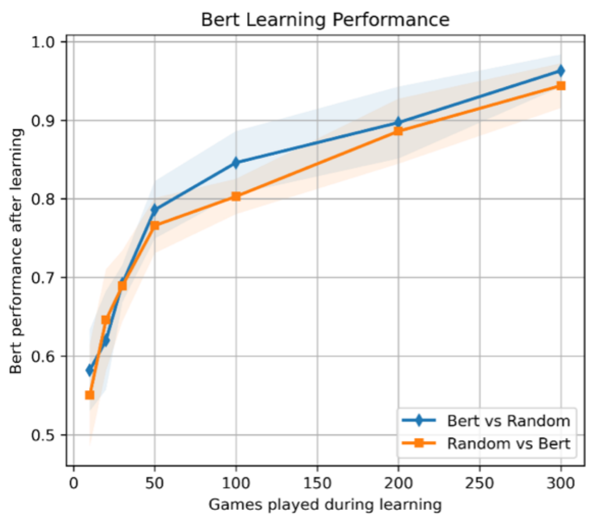

Figure 9. BERT Learning Nim Game from Random Player

The experiment trains a transformer by a certain number of games defined as a match, played between the Guru and the random player. Every training starts from reset and the trained model is used by the BERT player to make a move. An average game against Guru by the random player takes empirically ~6.5 many moves. Thus, 10- game match of Guru-random and an additional 10-game match of random-Guru would provide around 130 unique moves possibly. This space covers only 10% of the feature space (game board) presenting an almost impossible learning problem. Against all odds, as shown by the experimentation, the BERT player learns every move that Guru makes and plays accordingly when faced a random player. The range of number of games (match size) is changed from 10 to 300 where the latter makes the BERT player an excellent challenger for the random player. This is the direct result of the few-shot learning method presented by the transformers.

3. Experimental Results

Following the methodology stated of performing collections of data into a standard dataset format, data was collected for both Nim and chess experiments. Some of the characteristics for these datasets are listed in Table 1 - Dataset Metrics.

Table 1. Dataset Metrics

Methodology Metric	**Nim**	**Chess**
Number of Games	30,000	30,000
Total Unique Game States	7973	2575
Total Unique Moves	30	892
Dataset Length	423,480	2657
Average Sequence Length	15.09	74.28
Dataset Size (MB)	8.1	167.9

Notably with our method of data collection and data format, choosing longer sequences to represent a system will cause the dataset's memory size to grow by an order of M where M is the current dataset length. This caused issues when we initially tried to generate a chess dataset that contained one million games and created an 8 GB text corpus. This is also one of the reasons we added the Nim experiment, to test our hypothesis on a smaller scoped dataset.

The hardware used for the experimentation is an RTX-A6000, an i9 processor, and 128 GB of RAM.

3.1. Results Nim

Following the data collection, two BERT word piece tokenizers were trained on each variant of the Nim datasets: X/W and Player ID. The vocabularies for each tokenizer were relatively small and are shown in **Figure 10** and **Figure 11**respectively. The vocabulary size maxed out around 60 tokens for each tokenizer because the game of Nim is not that complicated in sequence form.

```
{'a3': 43, 'a1': 33, '8': 15, 'b8': 59, 'b9': 61, 'b1': 35, '4': 11, '##9': 23, 'c8': 58, 'G': 17, 'c4': 46, '##5': 31, '##6': 28, 'b3': 44, '##2': 32, 'c5': 48,
'c3': 42, '9': 16, '2': 9, 'c1': 34, 'a': 20, '6': 13, 'a10': 65, 'c0': 38, 'a8': 57, 'R': 19, '7': 14, 'b4': 45, 'c6': 51, 'a7': 56, 'c7': 54, '##3': 27, '[MAS
K]': 4, 'c9': 60, '[PAD]': 0, 'a5': 50, '##1': 24, '[CLS]': 2, '-': 5, 'b5': 49, 'b10': 64, '##7': 26, 'a9': 62, '1': 8, '0': 7, 'c': 22, 'a0': 36, '3': 10, 'a6
': 53, '5': 12, 'Q': 18, 'a4': 47, 'c10': 63, 'b0': 37, '##0': 25, 'a2': 41, '/': 6, '##4': 29, '[UNK]': 1, 'b6': 52, 'c2': 40, 'b2': 39, 'b7': 55, '##8': 30, 'b
': 21, '[SEP]': 3}
```

Figure 10. BERT Tokenizer Tokens for Nim with Player IDs' G, Q and R.

```
{'a0': 35, '/': 6, '9': 16, 'b1': 34, 'c1': 33, 'a6': 52, '7': 14, 'c9': 59, '##1': 24, '[UNK]': 1, '[CLS]': 2, 'c3': 42, 'b9': 61, 'a9': 60, 'c8': 56, '[MASK]':
4, 'b': 20, '##7': 30, 'c2': 39, 'a2': 40, 'b3': 41, '##8': 22, '1': 8, 'c0': 37, 'a1': 32, 'c4': 44, '-': 5, '##5': 26, 'b8': 57, '2': 9, 'W': 17, '5': 12, 'a10
': 63, 'c7': 54, 'b0': 36, 'c10': 64, '6': 13, '##6': 27, '##0': 25, '[PAD]': 0, 'c': 21, '##4': 31, '0': 7, '4': 11, 'b5': 48, '3': 10, 'a4': 46, 'a3': 43, '[SE
P]': 3, 'b2': 38, 'X': 18, '##2': 23, 'a': 19, 'b6': 50, 'a7': 55, '8': 15, 'b10': 62, 'c6': 51, '##3': 29, 'a8': 58, 'b4': 45, '##9': 28, 'a5': 49, 'b7': 53, 'c
5': 47}
```

Figure 11. BERT Tokenizer Tokens for Nim with Win States

We can verify the tokenizers captured the game state tokens and move tokens by inspecting their vocabulary. Since for Nim, the game state is being represented by a letter a, b, or c and a quantity we can see that those tokens do exist in the vocabularies.

Recall that two datasets were generated with partitions to designate artificial noise, the first had a special indicator for which agent made this move and the other had an indicator to as if this move won the game. We trained a fresh BERT model on each partition of each dataset and put each model into a roster where each agent played every single other agent, the results are below. The evaluation was performed with 1000 games of every permutation of every agent for each level of randomness for a total of 5000 games. The total wins for each partition were collected and that is what is shown in **Figure 12**, **Figure 13**, **Figure 14**, and **Figure 15**. For each level of randomness and for each graph, it took 20 minutes to train the BERT model.

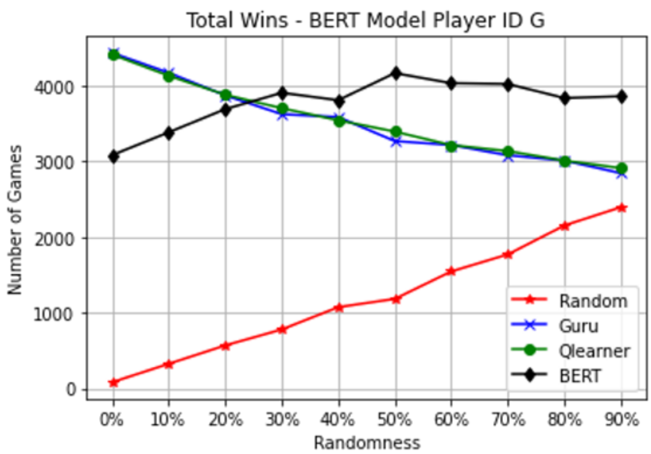

Figure 12. Nim Player ID G. The BERT model inferenced and played with the Guru (G) ID token being specified.

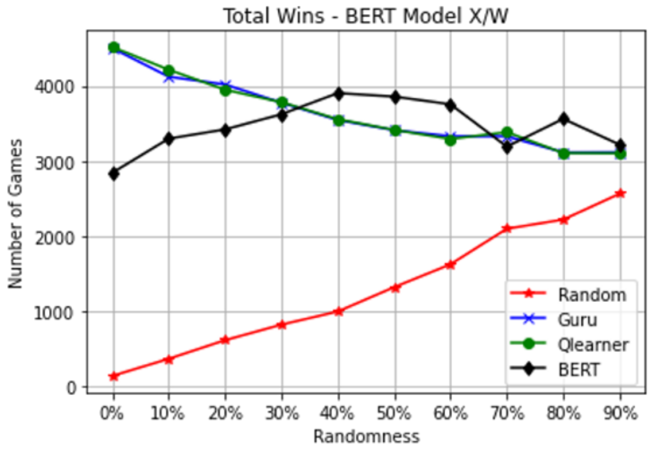

Figure 13. Nim X/W Win State

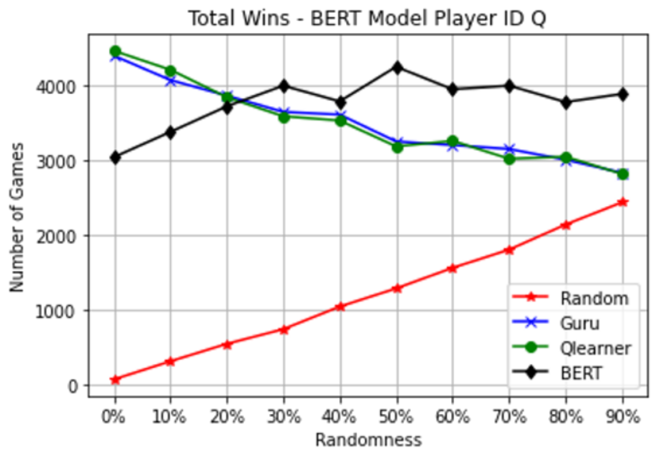

Figure 14. Nim Player ID Q. The BERT model inferenced and played with the Q learner (Q) ID token being specified.

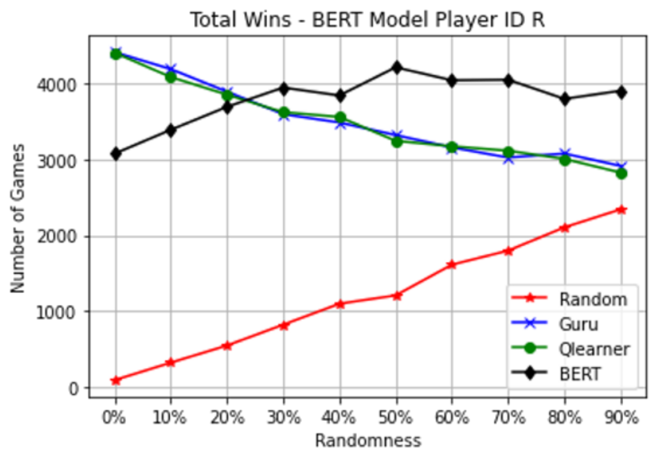

Figure 15. Nim Player ID R. The BERT model inferenced and played with the random agent (R) ID token being specified.

The BERT model consistently beats the other agents as the level of randomness increases in the dataset. This supports our original hypothesis because it shows evidence that the BERT model can identify the strongest signal (Guru and Q-learner) despite the random noise. This is especially evident in the 90% index of the results. Despite learning from a dataset where the players were only making one out of every ten of their moves, the model performed better than them. This is shown in **Figure 12**, **Figure 14**, and **Figure 15** where at randomness threshold of 30% the BERT model outperforms all the agents. This also held true up till 100% random. The model did not perform well however with a win/loss indicator system (graphed in Figure 8). In fact, the model somewhat follows the same performance trend as the agents.

The process of playing as one player or the other is defined within the state space of the text sequence. This is because the text sequence for a Nim sequence is generalized as the game space followed by an indicator token, and the corresponding move. For example, the sequence a10/b10/c10 G – [MASK] indicates that this should be a Guru agent move and a sequence such as a10/b0/c0 W – [MASK] indicates this is a winning move. These types of indicators are encoded into the dataset, and the transformer model learned these patterns.

The role of these indicators in the performance of the model is interesting. As it shows that one could perform additionally postprocessing on the dataset to add more attributes from which the model could learn from.

3.2. Results Chess

Only one BERT word piece tokenizer was trained on the Chess dataset. Its total vocabulary size is 16,000 tokens so it is not possible to show here like the Nim vocabularies.

The chess dataset was created with the first three moves (for each max level chess engine) per game. This is important to keep in mind as the BERT model seemed to perform well given that it had a very small subset of all possible chess moves.

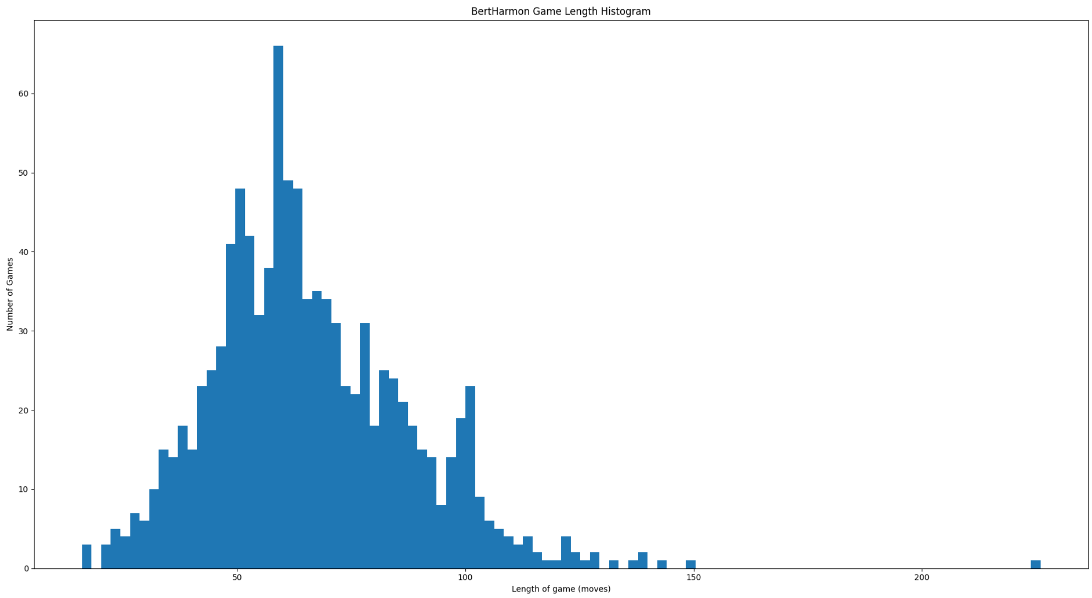

Figure 16. BERT Chess Game Length Distribution Versus Grandmaster Chess Engine

The BERT Chess model was not proficient enough to win chess games, so instead we show how well it did at playing chess against a grandmaster level chess player (Stockfish at max ELO). Additionally, it took around 3 days to train. Since the choice of a move given a board space is an open-ended answer, the model could technically answer with any text that it had in its vocabulary. We consider this as a feature of the model that it was given the option of answering in an incorrect format. As a result, we benchmarked its accuracy in terms of giving valid chess moves (shown in **Figure 17**). Given that it was only aware of the opening states of chess, it is impressive that at 35 moves into a game (35 moves for each player) it has an accuracy of 75%. When the model got a move wrong, we substituted a stockfish move in its place, and kept the game going.

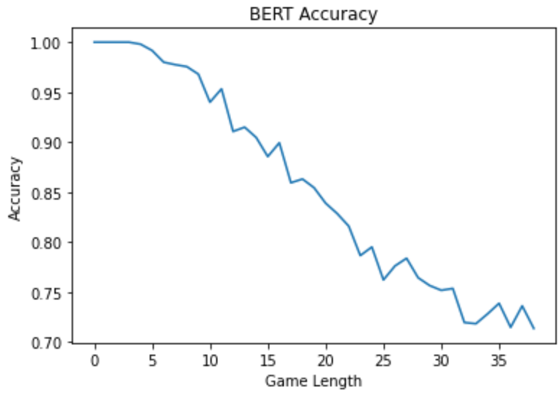

Figure 17. BERT Accuracy for Choosing Valid Chess Moves

In addition to measuring accuracy, we also measured the game length endurance of the model to understand how long it could play against a grandmaster stockfish engine till it lost. The results are graphed in Figure 11. Surprisingly BERT could survive for on average 32 moves (65 moves total for the game, ~32 per player), and at most we saw a game lasting for more than 200 moves.

4. CONCLUSIONS

In conclusion, we have shown that the BERT model is capable of learning both the games of Nim and Chess. We built text corpuses by representing game states and moves in a text sequence format. We have shown that the BERT transformer model is able to learn games in the context of very little information, in the presence of large quantities of noise, and in the presence of a large amount of data. The BERT model has been shown to learn the behaviors and patterns of primary game agents Guru, Q-learner, and stockfish such that the model can emulate their actions.

The results of our research should encourage BERT and other transformer models to be used as few-shot learners in situations where data is expensive to gather, difficult to clean, and in very high dimensional learning environments.

Transformer language models can represent input sequences efficiently through various autoencoding steps such as BERT, ALBERT, RoBERTa, ELECTRA, etc. Exploring the performance of these language models can help improving chess language models in this study. Second, these models can be used to cluster chess opening positions in order to compare and contrast to the ECO chess openings taxonomy. Future work will explore the clustering of chess positions to build taxonomy of openings, middle game positions and end game positions. This approach is analogous to text summarization where BERT approaches are known to be successfully applied. Third, future work will investigate player attribution in chess by analyzing various master games in chess databases as certain player styles are known to exist, such as Karpov likes closed and slow games, as Kasparov and Tal like open and sharp games.

Additionally, by representing a game space in our text sequence format, there are several interesting use cases with BERT such as authorship attribution, author playstyle and game space deduction. Given a dataset of games played by grandmasters, one could train this model and assess the probability that a given move has been made by Kasparov or another grandmaster by solving for the author/agent token instead of a move token. Additionally, if we solve for the move token then one could identify how a specific grandmaster would play given the board state. These two use cases allow for someone to prepare against a specific opponent. Given there are three spaces of the sequence, the last portion to solve for is the game space, and interestingly one could solve for the game space to suggest what is the most likely game space to precede this move for this player. All these use cases generalize for practical real-world problems that can be conceptualized into a state-independent text corpus such as predicting consumer behavior.

ACKNOWLEDGEMENTS

Special thanks to Booz Allen Hamilton, Johns Hopkins University, as well as our friends and family for their support.

REFERENCES

[1] Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P. and Szafron, D. (1991). “Reviving the Game of Checkers. Heuristic Programming in Artificial Intelligence 2: the second computer olympiad (eds. D.N.L. Levy and D.F. Beal), pp. 119-136. Ellis Horwood Ltd., Chichester, England.

[2] Maharaj, Shiva, Nick Polson, and Alex Turk. "Chess AI: competing paradigms for machine intelligence." Entropy 24.4 (2022): 550.

[3] Magnuson, Max. "Monte carlo tree search and its applications." Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal 2.2 (2015): 4.

[4] Hsu, Feng-hsiung. "IBM's deep blue chess grandmaster chips." IEEE micro 19.2 (1999): 70-81.

[5] McGrath, Thomas, et al. "Acquisition of chess knowledge in alphazero." arXiv preprint arXiv:2111.09259 (2021).

[6] Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.

[7] Matanović, A., M. Molorović, and A. Božić. "Classification of chess openings." (1971).

[8] Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science 362.6419 (2018): 1140-1144.

[9] Stöckl, Andreas. "Watching a Language Model Learning Chess." Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021). 2021.

[10] Edwards, Steven J. "Portable game notation specification and implementation guide." Retrieved April 4 (1994): 2011.

[11] Nasu, Yu. "Efficiently updatable neural-network-based evaluation functions for computer shogi." The 28th World Computer Shogi Championship Appeal Document (2018).

[12] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[13] Radford, Alec, et al. "Improving Language Understanding by Generative Pre-Training." (2018)

[14] Radford, Alec, et al. "Language models are unsupervised multitask learners." OpenAI (2019)

[15] E. JÄRLEBERG, “Reinforcement Learning Combinatorial Game Nim (2011).pdf,” KTH Royal Institute of Technology, 2011.

Authors

Michael DeLeo is an engineer and researcher. He currently works at Booz Allen Hamilton as a Machine Learning Engineer. He graduated from Penn State with a BS in Computer Engineering (minors in Math and Computer Science). He is also currently studying for his masters in Artificial Intelligence at Johns Hopkins University where he is performing research on NLP.

Erhan Guven is a faculty member at JHU WSE. He also works at JHU Applied Physics Lab as a data scientist and researcher. He received the M.Sc. and Ph.D. degrees from George Washington University. His research includes Machine Learning applications in speech, text, and disease data. He is also active in cybersecurity research, graph analytics, and optimization.