CHESSGPT: Predicting Chess Moves Using NLP Techniques.

TEAM# 20:

|  |  |  |
| --- | --- | --- |
| *Chetan Reddy Valluru*  *CV410*  *cv410@scarletmail.rutgers.edu* | *LayaKishore Reddy Desireddy*  *LD786*  *ld786@scarletmail.rutgers.edu* | *Bhavana Vemula*  *BV234*  *bv234@scarletmail.rutgers.edu* |

#### 1. Introduction

##### 1.1 The Challenge of Chess Move Prediction

In the realm of artificial intelligence, the game of chess has long served as a benchmark for evaluating the strategic reasoning capabilities of AI systems. This project, titled "ChessGPT," focuses on the intricate task of predicting future moves in a chess game using natural language processing (NLP) techniques and language models. Our primary objective was to develop an AI agent capable of analyzing a sequence of chess moves and generating informed recommendations for the next best move, ultimately aiming to win the game.

##### 1.2 Treating Chess Moves as a Language

The core concept behind our approach involves treating chess moves as a language. We leveraged a causal language model, specifically a pre-trained GPT-2, to learn the patterns and strategies inherent in chess games. This innovative approach views the prediction of the next chess move as a language modeling task, where the model learns to generate a sequence of moves that aligns with the rules and strategies of chess. By considering previous moves as context, the model can make informed predictions about the most suitable next move.

##### 1.3 Baseline Model: The Foundation

Our exploration began with the standard GPT-2 architecture with an AutoTokenizer as our baseline model. This baseline served as a benchmark, allowing us to assess the performance improvements achieved through subsequent fine-tuning and architectural modifications.

##### 1.4 Exploring Architectures: A Journey of Refinement

We embarked on a journey of exploring various architectures to identify the most effective approach for chess move prediction:

1. **Base GPT-2:** The standard GPT-2 model served as the initial baseline.
2. **GPT-2 with LoRA:** GPT-2 fine-tuned using Low-Rank Adaptation (LoRA) for efficient training with reduced computational resources.
3. **GPT-2 with Reinforced Fine-Tuning:** GPT-2 with enhancements like positional embeddings, custom tokenization, and a custom loss function to penalize illegal moves While.
4. **Chess Transformers from Scratch:** A custom transformer model coded from scratch, built specifically for chess move prediction, Vocab contains just chess moves in SAN format.

##### 1.5 Summary of Results: The Best Approach

Among the architectures we explored, the custom-built Chess Transformers model (Model 4) emerged as the most successful. It consistently generated legal moves and completed entire games without errors. Other models showed varying degrees of success in generating legal moves, with the base GPT-2 model generating up to 15 legal moves before encountering difficulties, similarly Model 2 and Model 3 generated up to 20 and 30 moves respectively.

#### 2. Data Collection and Preparation

##### 2.1 Data Source: High-Quality Chess Games

The foundation of our AI models lies in the quality of data used for training. We utilized a dataset of 100,000 high-level chess games in PGN (Portable Game Notation) format. To ensure high-quality strategic data, we included only games where at least one player had an Elo rating above 2500, indicating a high level of chess expertise.

##### 2.2 Data Cleaning: Preparing for Training

Data cleaning played a crucial role in preparing the dataset for model training. We performed several cleaning steps:

1. **Extracting Moves:** We extracted only the core move notation by removing prefixes like "W1.", "B1.".
2. **Removing PGN Data Features:** We focused solely on move sequences, removing metadata like time, players, and results.
3. **Adding Special Tokens:** We added special tokens <bos> (beginning of sequence) and <eos> (end of sequence) to help the model identify the start and end of move sequences.

##### 2.3 Train-Test Split: Evaluating Performance

To evaluate the performance of our models and prevent overfitting, we split the dataset into training and validation sets(80:20 ratio).

##### 2.4 Data Characteristics: Sequences of Moves

The data consisted of sequences of chess moves in algebraic notation (e.g., e4, Nf3, Bb5). The distribution across classes (possible moves) was likely uneven, with some moves being significantly more frequent than others.

##### 2.5 Data Snippets: Before and After

**Before Processing:**

* [Event "F/S Return Match"]
* [Site "Belgrade, Serbia JUG"]
* [Date "1992.11.04"]
* [Round "29"]
* [White "Fischer, Robert J."]
* [Black "Spassky, Boris V."]
* [Result "1/2-1/2"]
* [WhiteElo "2785"]
* [BlackElo "2660"]
* [Time Control "40/7200:3600"]
* [Termination "adjudication"]
* e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7.
* Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. Nc3 Qc7 13. a3 Bb7 14. Bg5 h6 15. Bh4 Rfe8 16.
* Rc1 Bf8 17. cxb5 axb5 18. Nxb5 Qb8 19. Nc3 exd4 20.
* Nxd4 c5 21. Nf5 g6 22. Ne3 Nxe4.....

**After Processing:**

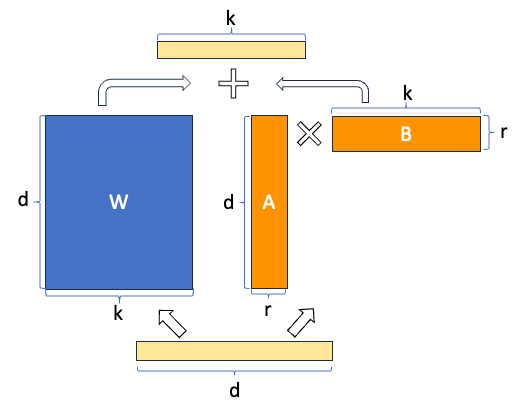
* <bos> e4 e5 Nf3 Nc6 Bb5 a6 Ba4 Nf6 O-O Be7 Re1 b5 Bb3 d6 c3 O-O h3 Nb8 d4 Nbd7 c4 c6 Nc3 Qc7 a3 Bb7 Bg5 h6 Bh4 Rfe8 Rc1 Bf8 cxb5 axb5 Nxb5 Qb8....<eos>

#### 3. Experiments: A Tale of Four Models

##### 3.1 Model 1:Base GPT-2: The Starting Point

* We started with the pre-trained GPT-2 model as our baseline.
* It was fine-tuned on the chess move prediction task, adjusting the model's weights to perform well on this specific data.

##### 3.2 Model 2: GPT-2 with LoRA: Efficiency in Focus



* To improve efficiency, we used Low-Rank Adaptation (LoRA) for fine-tuning.
* LoRA works by injecting trainable rank decomposition matrices into each layer of the Transformer architecture, reducing the number of trainable parameters in the model.
* This approach allows for efficient fine-tuning of large language models, especially when computational resources are limited.

##### 3.3 Model 3:Gpt2 with reinforced fine tuning(Custom Loss Function with illegal move penalization)

* We enhanced the GPT-2 model with positional embeddings, custom tokenization, and a custom loss function.
* Positional embeddings help the model understand the order of moves in a chess game, which is crucial for strategic decision-making.
* By adding <w> for White win and <b> for Black win tokens at the beginning of the move sequence for every chess game during training, we enable the model to generate winning moves when prompted. This approach ensures that specifying the desired winning player at the start of the generation process guides the model to produce winning moves, rather than just any moves or potentially losing moves.
* The custom loss function penalized illegal moves, encouraging the model to generate only valid moves, The idea here is while training we know model uses teacher forcing techniques when model predicts a wrong move but when the model predicts a wrong move that move has to be at least a legal move even if it is not correct move, So while training irrespective of a correct move or wrong move if the model predicts an illegal move we do loss = loss\*2, but when it predicts legal move loss = loss, this way model is guided towards legal move prediction.

##### 3.4 Model 4:Chess Transformers from Scratch:

A screenshot of a computer program

Description automatically generated

* **Custom Architecture**: This model was built from scratch (Similar to GPT2 model) specifically for chess move prediction, ensuring all tokens are relevant and optimized for chess moves.
* **Optimized Tokenization**: A custom tokenizer was used to efficiently encode and decode chess moves, further enhancing the model's efficiency and performance.
  + This custom tokenizer contains vocab size of around 10,000 tokens, where each token is a chess move in Standard Algebraic Notation.
* **Positional Encoding and Self-Attention**: The model incorporates positional encoding and self-attention mechanisms, enabling it to effectively capture the relationships between moves and their positions in the game.

**Performance**

* **Completes the game**: This model was able to complete entire games with legal moves, demonstrating its ability to learn and generate valid chess move sequences.
* **All tokens are relevant**: The custom architecture and tokenizer ensure that all tokens are relevant and contribute to the model's understanding of chess.

**Why Model 4 is Better**

* **Higher accuracy**: Model 4 exhibits higher accuracy in predicting legal moves compared to the other models.
* **Better handling of game-specific scenarios**: The custom architecture and training process enable Model 4 to handle game-specific scenarios more effectively.
* **Efficient training and inference**: The optimized model design leads to efficient training and inference processes.
* **Reduced irrelevant tokens**: The use of relevant tokens contributes to better utilization of computational resources.

**Conclusion**

Model 4 provides the best performance and efficiency for chess move prediction due to its custom architecture and optimized tokenization. Future work may involve further fine-tuning and optimization to enhance the model's capabilities and accuracy.

#### 4. Evaluation: Measuring Performance

##### 4.1 Basic Evaluation: Can the Model Play a Game?

* We first checked if the models could complete a game without making illegal moves.
* If a model couldn't complete a game, we counted the number of legal moves it generated before breaking.

##### 4.2 Playing Against Stockfish: The Real Test

* We evaluated the models by having them play 100 games against Stockfish, a strong chess engine.
* We tracked the number of wins, draws, and losses for each model.
* This evaluation provided insights into the model's ability to compete against a challenging opponent.

##### 4.3 Game Analysis: Understanding Move Quality

* We analyzed the games played against Stockfish to understand the quality of moves made by our models.
* We categorized moves as "best moves," "good moves," "mistakes," and "blunders" based on Stockfish's evaluation.
* We calculated the accuracy of the models based on the ratio of number of good and best moves to total number of moves.
* This analysis helped us identify areas where the model excelled and areas where it needed improvement.

### 5. Results: And the Winner Is...

Our exploration of different model architectures culminated in a head-to-head competition between our AI chess players and the formidable Stockfish engine. This rigorous evaluation process aimed to uncover the strengths and weaknesses of each model, ultimately revealing the champion.

#### 5.1 Basic Evaluation: Can it Play a Full Game?

The initial test for our models was demonstrating the ability to play a complete chess game without making illegal moves. This basic evaluation served as a filter, ensuring that only models capable of generating valid move sequences would proceed to the next stage.

* **Model 1 (Base GPT-2)** stumbled early on, managing to generate only up to **15 legal moves** before faltering.
* **Model 2 (GPT-2 with LoRA)** showed improvement, generating **20 legal moves** before encountering difficulties.
* **Model 3 (GPT-2 with Reinforced Fine-Tuning)** inched further, generating **27 legal moves** before hitting a wall.
* **Model 4 (Chess Transformers from Scratch)** emerged as the champion in this stage, successfully completing entire games while adhering to the rules of chess.

#### 5.2 100 Games Against Stockfish: The Real Test

The true test of our models' capabilities lay in their performance against Stockfish, a powerful chess engine renowned for its strategic prowess. We pitted each model against Stockfish in **100 games**, meticulously tracking the outcomes: wins, losses, and draws.

* **Model 4** once again demonstrated its superiority, securing a **10% win rate** against Stockfish. While a **19% loss rate** highlights areas for improvement, the impressive **69% draw rate** underscores its ability to hold its own against a formidable opponent. The remaining **2% of games had unknown outcomes(Couldn’t complete the game)**.

#### 5.3 Game Analysis: A Deeper Dive

To gain a more nuanced understanding of our models' performance, we delved into the individual games played against Stockfish. We analyzed the quality of moves made by each model, categorizing them as "best moves," "good moves," "mistakes," and "blunders" based on Stockfish's evaluations.

This analysis revealed interesting insights:

* **Total White Best Moves**: 207
* **Total White Good Moves**: 192
* **Total White Mistakes**: 84
* **Total White Blunders**: 308
* **Total Black Best Moves**: 257
* **Total Black Good Moves**: 202
* **Total Black Mistakes**: 64
* **Total Black Blunders**: 268

These numbers suggest that while Model 4 can generate good moves, it still has a tendency to make suboptimal choices, particularly when playing as white.

From these numbers, we can calculate move accuracies:

* **White Move Accuracy**: 50.34%
* **Black Move Accuracy**: 65%

This indicates that the model performs better when playing as black, potentially due to reacting to Stockfish's moves rather than dictating the pace of the game.

#### 5.4 Detailed Draw Breakdown

The **69 draws** against Stockfish were further analyzed to understand the reasons behind the drawn outcomes:

* **Stalemate**: 3
* **Insufficient Material**: 0
* **80-move Rule**: 62
* **Threefold Repetition**: 4

The high number of draws due to the 80-move rule suggests that Model 4 can play defensively and avoid losses, but may struggle to push for a win against a strong opponent.

#### 6. Conclusion: Lessons Learned

#### 6.1 Key Takeaways from Experiments and Results

* **Custom architectures outperform fine-tuned models**: Our custom-built Chess Transformer (Model 4) significantly outperformed pre-trained models like GPT-2 in terms of both generating legal moves and achieving a respectable performance against a chess engine.
* **Efficient training is crucial**: Techniques like LoRA (Low-Rank Adaptation) are essential for efficiently training large language models, especially when computational resources are limited.
* **Chess-specific knowledge enhances performance**: Incorporating chess-specific knowledge, such as positional embeddings and custom loss functions that penalize illegal moves, significantly improves the model's ability to generate plausible and legal chess moves.

#### 6.2 Lessons Learned

* **The power of custom architectures**: Designing a model tailored to a specific domain, like chess, can lead to significant performance gains compared to relying solely on pre-trained models.
* **Efficiency in training**: Techniques like LoRA are vital for managing computational resources when working with large language models.
* **Domain-specific knowledge is key**: Incorporating domain-specific knowledge into the model architecture and training process is essential for achieving optimal performance.

#### 7. Future Work: Further Enhancements

* Incorporating the actual board state into the model's input could enhance its understanding of the game's context.
* Exploring multi-modal learning by combining transformer models with convolutional neural networks (CNNs) to process the visual representation of the chessboard might further improve performance.
* More advanced evaluation metrics beyond the evaluation strategy we used, could provide deeper insights into the model's strengths and weaknesses.

### 8. User Interface: Bringing ChessGPT to Life

We utilized Streamlit, a Python library specifically designed for building interactive web applications, to develop a user-friendly UI for our ChessGPT system.

#### 8.1 Key Features and Design Choices

* **Interactive Chessboard**: The UI features an interactive chessboard that dynamically updates as the game progresses.
* **Move Input**: A simple text input field allows users to enter their moves in standard algebraic notation (e.g., e4, Nf3).
* **AI Response**: The UI displays the AI's response move clearly.
* **Game Controls**: Intuitive buttons enable users to make moves, reset the game, and view the move history.
* **Game Status**: The UI provides real-time feedback on the game status, including check, checkmate, and stalemate notifications.
* **Move History**: Users can expand a section to view the complete history of moves made in the game.

#### 8.2 Streamlit: A Powerful Tool for UI Development

Streamlit's simplicity and flexibility made it an ideal choice for developing our ChessGPT UI.

#### 8.3 User Experience Considerations

We prioritized user experience (UX) by focusing on clarity, ease of use, and visual appeal.

#### 8.4 Future Enhancements

* **Visualizing AI's Thought Process**: Displaying the top move candidates considered by the AI, along with their associated probabilities.
* **Difficulty Levels**: Implementing different difficulty levels by adjusting the AI's search depth or exploration parameters.
* **Personalized Feedback**: Providing users with personalized feedback on their moves, potentially by integrating with a chess engine for analysis.
* **Aesthetic Enhancements**: Incorporating more visually appealing design elements, such as custom themes and animations, could further enhance the overall user experience.

###### 9. Research Paper Summary

**"Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm" by Silver et al. (2017)**

This paper introduces AlphaZero, a general reinforcement learning algorithm that achieves superhuman performance in chess, shogi, and Go without any human knowledge beyond the basic game rules. AlphaZero utilizes a deep neural network that is trained through self-play, where the algorithm plays millions of games against itself, progressively improving its strategies. The paper's findings highlight the power of self-play and deep learning in mastering complex games, offering valuable insights for developing AI agents for chess and other strategic games.

#### 10. Final Thoughts

This project provided a deep dive into the application of natural language processing techniques to the game of chess. We explored various architectures, fine-tuning methods, and evaluation strategies to develop a model capable of predicting chess moves with reasonable accuracy. The custom-built Chess Transformers model emerged as the most successful approach, demonstrating the potential of tailoring AI models to specific domains. While there is still room for improvement, this project highlights the exciting possibilities at the intersection of AI and chess.