## 

## CMPSC 448

## 

## Chess Bot Using CNN & RNN

## Andrei Mudrov

## Yousef alRabiah

## 

In the process of selecting our final project, we considered developing a chess bot capable of competing with humans. For this, we needed to find a dataset that addressed this problem, and then train our model using this dataset. Given the current board layout, the chess bot would suggest a move to counter the opponent. The program lacks an interface, so the game is simulated on Chess.com, with the bot's output being manually inputted. The chess bot operates using a combination of Convolutional Neural Networks and Recurrent Neural Networks.

Our project drew inspiration from AlphaZero, a chess engine developed by DeepMind in 2017. Traditionally, chess engines relied on heuristic tree searches and complex mathematics. However, AlphaZero revolutionized this approach by defeating other engines using a neural network.

In researching chess engines, we discovered a website with a collection of chess games dating back to 2005 ([computerchess.org](https://www.computerchess.org.uk/ccrl/4040/games.html)). We downloaded 1,719,507 fully complete chess games. These games were in Portable Game Notation (PGN), which we had to parse for our model to understand. After parsing, we obtained chess moves in Forsyth-Edwards Notation (FEN), which describes positions in a chess game. To create a chess bot, Python offers standard libraries like Chess, Gym, and Gym-chess. These libraries assist in validating chess moves, parsing, and tracking the game state.

Our processed data took the form of a 3D array of size 8 by 8 by 58, where the first and second indexes represented the square on the board and had 58 features. The first 6 channels represent the existence of one of the 6 chess pieces - pawn, knight, bishop, rook, queen, and king. The next 5 channels represent the availability of castling for each king, and whose turn it was, the 12th and 13th channels represent the material count for each side. The channels 14 to 56 represent the legal moves for a piece on this square, and the last channel holds the value of the piece square tables for the existing piece on this square. Piece square tables represent the importance of a piece standing on a certain square. We extracted all of these features from FEN board notations. The expected output was a 1D array of size 4762, where each value represents how good the move is. This size was selected because theoretically, there are 4762 possible moves in universal chess interface (UCI) notation. We used the OpenAI library gym to encode moves from UCI into a number from 0 to 4671, the same way AlphaZero did.

For training the CNN, we used two convolutional layers, one max pool layer, and multiple dense layers. Using these layers together creates the convolutional neural network that we use to train our chess bot. Afterwards, we used the scikit-learn library to fit the model. As for the RNN, we used a ConvLSTM2D layer, a dense layer, and a Flatten layer. Both models share similar code. Finally, we used unsupervised evolutionary training to make the bot compete against itself, with 20 players playing 3 games each. Then we selected the top 5 players for our model. New 20 players were generated from those five by keeping the top 5 players, creating 10 with gene exchange (our genes being the weights of the neurons), and the last 5 players were randomly mutated top 5 players. The initial 20 players were generated by randomly mutating the model that went through supervised learning. We selected the top 5 players by the sums of their wins minus the number of their losses.

We faced many challenges in creating our project. The first challenge was the dataset. The dataset we found consisted of 1.7 million games between computers. With our available setup, it would take more than 100,000 hours to go through all the games and transform them into data for training. The second challenge was the training time. Even though we used 100 games and 7 generations of evolutionary learning, it took multiple hours to complete, yielding almost no results. The models after the supervised learning were able to make random legal moves and, after the evolutionary learning, were able to draw the Martin bot, the lowest-ranked bot on chess.com. The chess bot was able to generate chess moves; however, upon inspection, we realized that it was aiming for a draw instead of a win. It would repeat moves until the game was over, thus minimizing its loss against other similar bots.

In conclusion, we believe that our training method and data selection were sufficiently successful, but our inability to properly train and optimize our chess engine prevents us from advancing further.