AI PROJECT REPORT

**Done by:**

RA1911029010093

RA1911029010095

RA19110290100102

**Chess Game**

**Introduction:**

Our motivation for this project is that the members of the group are chess players and felt it would be interesting to explore the idea of creating a chess AI based , a concept that we covered in this class. We have seen the emergence of current chess AI’s and wanted to create our own from scratch that played more human-like moves rather than the perfect move that other chess AI’s aim to play. The importance of creating a strong chess AI that will play moves similar to that of a human is that it would be vastly beneficial for people trying to learn chess, practice, and improve their skills. Current chess engines such as AlphaZero, Stockfish, and Leela Zero are far stronger than any human. These chess AI’s also play very counter-intuitive moves and openings such as the Reti Opening and the Queen’s Indian Defense. All chess engines rely on what is called an evaluation function, an algorithm to determine the strength of a given position. Almost all of these AI’s use a very complicated linear combination of characteristics of a position to evaluate it’s strength. The nature of these hard-coded evaluation functions makes the AI that is using them play counter intuitively. This makes them ineffective for training intermediate players. There have also 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada. been recent attempts to create a more intuitive AI with the introduction of the Maia Chess AI which is designed to specifically predict “human moves.” Our goal is to create a similar AI using deep learning. Essentially, instead of hard-coding this evaluation function, we want to use deep learning to evaluate the strength of a position. Our hope is that training our neural network with data about games played by humans will make the moves that it suggests more intuitive. Creating this evaluation function will be a matter of training our neural network. Then, we will use our trained network to predict the best possible move given a position.

**Problem Definition:**

The portion of this project that involves machine learning is the creation of the evaluation function. We built this function using an artificial neural network. The goal of our evaluation function is to take two chess positions as inputs and return which of these two positions is better for white. For example, given one position where white is better and another positions where black is better, the evaluation function should return [1,0] or [0, 1], depending on which position is better for white. This evaluation function is at the core of every chess AI. We aim to build this evaluation function from scratch by training it with pairs of positions and the outcomes of their respective games. We juxtapose two positions and pass them in to the neural network and train it on the outcome of the games from which the position is extracted from. Our hope is that we can train the neural network to learn how to compare the positions. We will then use the trained neural net to evaluate the strength of positions that would be the result of a legal move given a current board state. Since this is a deep-learning model, assumptions are based on prior beliefs about the type of hypothesis the model should be learning. We assume that a randomly chosen position from a game that white wins is a stronger position for white.

**Data Collection:**

Through extracting data with a python script, we actually have a unique data set that only exists with us. We used files in the pgn format that are used to represent chess games. Parsing these games with the python chess library, we extracted 200,000 random positions from over 20,000 grandmaster games from <https://www.pgnmentor.com/files.html> and mapped them to the corresponding game outcome. Then, we organized these board positions into pairs in which one position was eventually won by white while the other position was eventually won by black. Our expected outputs will be 10 or 01 depending on which game white won. To feed the data into our neural network, we encoded each board position into a binary bit string of length 774, storing information about the position, castling rights, and whose turn it is to play. Pairs of these bit strings are mapped to the outcome of the games and used as input/output pairs. Since we created a new data set by combining the PGN data with the python chess library, our data is novel. The data is already cleaned since we mapped random positions to their game outcomes. Because we are using pairs of positions as our input, we actually substantially increase the size of our data set. Thus, each epoch in our supervised training will be largely unique as we will shuffle the data set and pair the positions differently. This gives us an edge when training as we can generate a much larger training set with a limited amount of data. The games from which we extracted these positions come from a database of grand master games played throughout history. typically, a deep learning chess AI is trained on games played by high-level computers. However, given the goal of our project, which is to design an AI that plays more intuitively, our training set is based entirely upon games played by humans. We designed several methods that work together to generate the data. We have a method that parses each game and pushes a random amount of moves from the game onto a python board object from the python chess library. By doing this, we extract 10 unique and random positions from each game in our PNG format games. we store this large set of 200,000 positions in a numpy array containing all positions. We then label each of these positions with the corresponding results in a separate numpy array. We save these positions to our disk so that we can load them from a different python executable that trains our neural network with our data set. We picked this as our data set because the data is taken from real human games, so the machine will be trained on human moves, but since professional Grandmaster games are used, it will still play extremely well.

**Method:**

The primary concern of chess-ai is the decision-making part of the application. All functionality outside the scope of the AI are implemented using external libraries:

Chessboard GUI: Using the chessboard.js API

Game Mechanics: Using the chess.js API

The AI uses the [minimax algorithm](https://en.wikipedia.org/wiki/Minimax), which is optimised by [alpha-beta pruning](https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning).

The evaluation function uses [piece square tables](https://www.chessprogramming.org/Piece-Square_Tables) adapted from Sunfish.py, and eliminates the need for nested loops by updating the sum based on each move instead of re-computing the sum of individual pieces at each leaf node.

A global sum is used to keep track of black's evaluation score after each move, which is used to display the 'advantage' bar.

**Limitation:**

There are many ways in which you may try to make it stronger. First you could change from a board representation to a mutable array and add a fast way to enumerate pieces. Then you could implement dedicated capture generation, check detection and check evasions. You could also move everything to bitboards, implement parts of the code in C or experiment with parallel search!

**Results:**

At the time of Touchpoint 2, we had completed supervised training with an experimental loss function and learning rate. When we finalized our initial unsupervised training, we achieved good results. Our mean squared error for each of our layers was about .003. The weights of this network are stored along with our self-generated data set. We had a script set up for the supervised training that was fully functional, but too slow for our personal computers. Because of this, we must rent a GPU instance and try to run our program there. However, we are pleased with the amount of training data we have and the initial results of our greedy layer-wise learning for our base layers. After we are able to train our supervised learning model, we will have a fully functional evaluation function to center our chess AI around. We initially trained our supervised model with very low epoch numbers. However, as expected our model did not perform up to standards. We expected this to change when we have access to more computing power. At this point, our accuracy was around 70 percent. This does not exceed the baseline we defined as members of our group that have experience playing chess can more accurately assess the strength of positions. However, we are pleased to achieve this level of accuracy with the limited computing power available to us. There is definitely room for improvement which we detail in the next section.

**Results 2:**

After our first training session, we made considerable changes to our model. We doubled the data of the original training by adding more games and running our data extraction script again. We also designed a custom learning rate that exhibited a feature of exponential decay. Our epochs for unsupervised training were increased by a factor of 2 and the number of epoch for our supervised training was increased from 200 to 1000. We ran our new script with a much more powerful computer. With these changes, we were able to reach an accuracy of 77 percent, a 7 percent increase from our first training session. While this increase is non-trivial, it is not quite sufficient to play chess at a high level. We implemented an alpha-beta search algorithm that utilizes our trained neural network to output a move given a position. We also wrote a script that simulates a chess game that we can play against the AI. We played several games against the AI and determined it is not strong enough to beat the members of our team. However, we tested our AI’s skills against players online on the website chess.com. The results of this experiment put our AI at about an ELO of 600. We believe there is significant room for improvement by changing several factors. First, we must increase the accuracy of our neural network by adding more training data and increasing the diversity of the games that it is trained on. We thought about including lower level games into our training data. Another possible improvement is to make modifications to the architecture of our unsupervised training structure. After running the unsupervised training, our model fixed the weights and biases associated with the deep belief network during the supervised training. If we left these weights open for modification, we might be able to exhibit better results. The quality of our data can also be improved. For example, we can filter out positions that happen adjacent to a capture, as this results in a transient advantage for one side. The biggest improvement in the game-playing performance can most likely be made in the actual search algorithm. We haven’t included any of the rules of chess, including checkmate, into our algorithm. It is strictly running on the preferences of the neural net output.