

Comparison of AI algorithms in the game of Chess

CS7IS2 Project (2019-2020)

Abishek Vaithylingam, Akash Verma, Pooja Ganesh Teje, Suhrid Chatterjee

vaithyla@tcd.ie, vermaal@tcd.ie, tejep@tcd.ie, chattesu@tcd.ie

Abstract. This paper compares search algorithms in AI like Depth-First Search (DFS), Mini-Max algorithm and Feed-Forward Neural Networks based on various metrics like performance, speed and accuracy, number of steps, etc. in the game of chess. Each algorithm starts from scratch by analysing the board while the Feed-Forward Neural Network implementation is trained from scratch, starting with knowledge about the different weights, positions of the chess pieces and knowledge about the rules of chess using combinations of unsupervised pre training and supervised training. The paper analyses these algorithms in detail, draws focus to their similarities, fundamental differences and working principles.

1 Introduction

Chess is a strategic game of intellect between two players that requires high levels of sophisticated logic and reasoning. Believed to be based on the Indian game, Chaturanga [1], it has always been assumed to be a game that cannot be bested by computers and was often listed with activities that can only be performed by humans like painting, writing and poetry. While there has been little success in building AI for creative activities like writing poetry, there has been significant progress and success in developing AI to play single player and multi-player games like chess. Building an accurate chess engine using AI proves to be a difficult and complex problem as it is difficult to formulate rules to determine which branches are to be pruned and which branches are to be explored further. There has been significant research attempting to make chess game engines more selective while trying not to overlook important continuations. This logical reasoning can be done by humans as we have “intuition” and the ability to confine to complex rules while making exceptions for the rules when needed. Since the complex rules are understood by humans by playing, observing other players and analysing these moves, we have not been able to successfully convert our knowledge and intuition into well-defined rules for AI that covers all situations.

This paper presents a comparison of three different search algorithms in AI that have been used for solving the game of chess. The algorithms used are Depth-First Search (DFS) algorithm, MiniMax algorithm and Feed-Forward Neural Networks. Each of these algorithms use different interfaces to interact with users and to play games. The paper also evaluates these algorithms based on their accuracy, success rate, failure rate and other parameters like training time, execution time, loss, etc.

2 Related Work

Due to its sophisticated, yet unambiguous nature, chess is a comprehensively researched topic in Artificial Intelligence. Numerous research papers have been documented covering all aspects of computer chess since the early 1950s with numerous chess engines that have been developed by professionals, researchers and amateurs. Given a chess position, the objective is to find a move, out of all permissible moves for that position, that maximizes our chance of winning the game.

Stickel et. al. [2] presents a paper that focuses on using consecutively bounded Depth-First Search for automated deduction of moves in the game of chess. The methodology adopted in the paper involves repeatedly performing exhaustive Depth-First Search with the depth bounds increasing from 1 to n . The result of this approach is analogous to the Breadth-First Search approach. Instead of maintaining the results of the search problem at the $(n-1)^{th}$ level, the earlier results of the search are computed again. Ontanon et. al. [4] presents a paper that studies the implementation of the MiniMax algorithm at real time, called the RTMM (Real Time MiniMax) algorithm. The RTMM algorithm is a real-time adaptation of the regular MiniMax algorithm. The paper also discusses its application with respect to Real Time Strategy (RTS) games and provides an empirical evaluation and comparison of the algorithms in multiple real-time game scenarios. Koenig et. al. [5] provides a different approach to solving the classic search problem with a model to perform real time heuristic search with the MiniMax algorithm by introducing the MiniMax Learning Real-Time A* (Min-Max LRTA*) method. The paper gives an outline of the LRTA* method, a heuristic search method which is employed in deterministic domains at real time. The study is also expanded to analyse domains with non-deterministic nature and applied to navigation deduction tasks. Dendek et. al. [6] proposed a novel approach to introduce “learning” in the domain of two player chess games with a Neural Network based classification approach in a hybrid AI search model. Their study introduced a well-defined structure of metric space in the space of deductions for the chess program to utilize the referenced learning model. Si et. al. [7] proposed a new architecture for building computer chess programs using three different chess game engines. The paper adopted the use of three-layered feed-forward neural networks in the chess engines to help the model learn the sophisticated rules of chess. The paper also presents a detailed analysis of the different metrics achieved using the feed-forward neural network architecture to solve the game of chess like performance, accuracy, loss, training time, execution time, success and failure rates.

Computer programs designed to simulate two player chess games have significantly improved over the last few decades. Researchers have studied the transition of computer chess programs from the first program that was incapable of even being a challenge for novice players to the present-day advanced AI based chess programs that outperform even the strongest chess champions of the world. Despite current day chess programs being advanced, a deficit these algorithms still face is the inadequate ability of the programs to “learn” to make efficient or better moves.

3 Problem Definition and Algorithms

The problem statement is as follows- “Given a chess position, the objective is to find a move, out of all legal moves for that position, that maximizes our chance of winning the game”. To solve this problem, the study focuses on the use of three different search algorithms in Artificial Intelligence. The algorithms that have been used and

compared are the Depth-First Search (DFS) algorithm, Mini-Max algorithm and Feed-Forward Neural Networks.






Symbol					
Piece	Pawn	Knight	Bishop	Rook	Queen
Value	1	3	3	5	9

Table 1: Symbols of chess pieces and their values

3.1 Feed-forward Neural Networks

This study makes use of a fully connected feed-forward neural network consisting of 32 hidden layers with 8 nodes each to perform the task of classification.

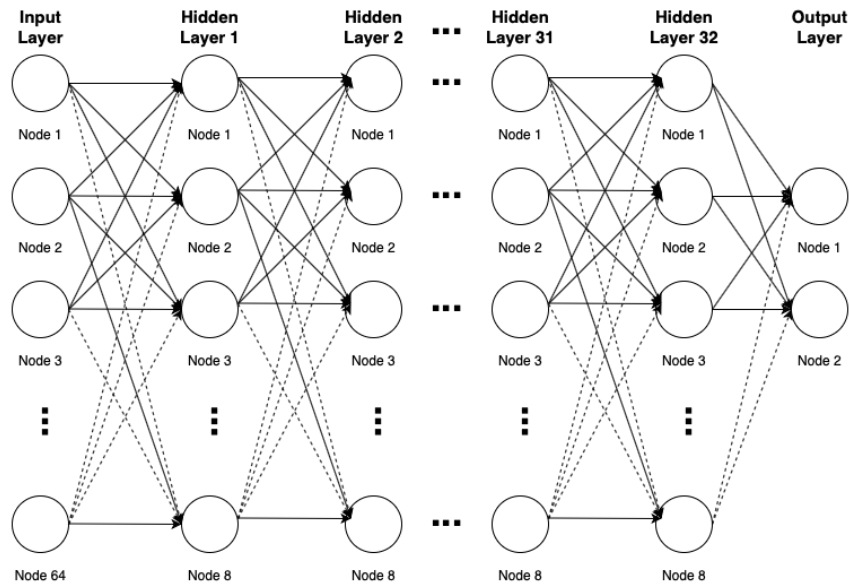


Figure 1: Architecture of the Fully Connected Feed-Forward Neural Network

The input layer of the feed-forward neural network consists of 64 input nodes which are split into 3 groups, each of which equates to the position of that unique piece on the board and the output layer consists of 2 output nodes, each of the nodes corresponding to the chances of a player winning. A ReLU activation function has been used for the hidden layer and a Softmax activation function has been used for the output layer so that the sum of the fully connected Feed-Forward Neural Network outputs would be equal to one. The loss function used for this model was a categorical cross entropy function. The selection and tuning of these hyperparameters is of utmost

importance as they determine the ability of the model to manage complexities, the time taken to train and test the model.

3.2 MiniMax Algorithm

The MiniMax algorithm is a commonly used specialized AI search algorithm that generates the most efficient order of moves for an entity in a zero-sum two player chess game, where one entity's gains come at the cost of the other entity's losses. Every level in the game tree generated by the Min-Max algorithm represents a choice of moves by one of the two entities participating in the game. The pseudo code for the MiniMax algorithm is given below-

```
int maxi( int depth ) {
    if ( depth == 0 ) return evaluate();
    int max = -oo;
    for ( all moves ) {
        score = mini( depth - 1 );
        if( score > max )
            max = score;
    }
    return max;
}
int mini( int depth ) {
    if ( depth == 0 ) return -evaluate();
    int min = +oo;
    for ( all moves ) {
        score = maxi( depth - 1 );
        if( score < min )
            min = score;
    }
    return min;
}
```

The max player (player to make a move) reviews the evaluation after performing all the possible moves and the move with the most optimal evaluation is the chosen move in a one-ply (only one player can move) search scenario. However, the game becomes very complex in a scenario where two players are involved in the game (two-ply search scenario) as the opponent is driven to make the best possible move too. The resultant score of every move is the score of the worst moves made by the opponent. The number of planning tasks that are solvable are limited by the MiniMax algorithm as they are excessively pessimistic in nature since it is only possible to solve tasks with a finite upper limit of start state goal distance.

3.3 Depth-First Search Algorithm

Depth-first search (DFS) is a commonly adopted algorithm in game simulations for performing the task of tree traversals and searching. The algorithm traverses the whole distance for each branch before backtracking, starting from the root node. Typically, the players have to choose one out of several choices of moves where each move consequently generates a new set of possible moves, ultimately producing an elaborate tree-graph of possible moves.

The current state is represented by the root node, the states of the game are represented by the nodes, moves/decisions are represented by the edges in the tree and the final state of the game is represented by the leaf nodes. The algorithm uses backtracking as a technique of correcting itself when it encounters a wrong decision, i.e. a move that does not lead to a win. In such a situation, the algorithm goes back to the previous node to try a different path that will ultimately result in a win.

The algorithm uses simple estimation of the states of the chess pieces that is based on the summation of chess pieces held by one chess player over the other, with the assumption that both entities playing the game are making the most optimal moves. The pseudo code to find the most optimal move for a specific state using the Depth-First Search (DFS) algorithm is given below-

```
int MAX_DEPTH = 3;
int tryMove( state, humanOrPC, depth ) {
    if ( depth >= MAX_DEPTH ) return null, calculateScore( state );
    var maxScoreCanHave = humanOrPC ? max_int : -max_int;
    var rightMove = null;
    for (move : getAllPossibleMoves() ) {
        var bestOfTheOther = tryMove( makeMove( state, move ), humanOrPC ^ 1,
        depth+1 );
        if ( ( bestOfTheOther < maxScoreCanHave && humanOrPC ) ||
        (bestOfTheOther > maxScoreCanHave && !humanOrPC ) ) {
            maxScoreCanHave = bestOfHuman;
            rightMove = move;
        }
    }
    return rightMove, maxScoreCanHave;
}
```

4 Experimental Results

This section provides details of the evaluation of the two player computer chess games using a Feed-Forward Neural Network, the DFS algorithm and the MiniMax algorithm. The evaluation between these engines seemed obvious based on the problem statement chosen, which is to simulate a competition of the algorithms against each other, analyse and evaluate the outcomes.

4.1 Methodology

The performance of each algorithm was evaluated using a round-robin system of having the three algorithms compete against each other. The Lichess platform [9] was used to simulate these games as it provides insights that helped us analyse the games. The three algorithms we chose played one series of games against each other, with each series having three games. In every game, it was distinguished as to which algorithm was making smarter decisions and how far ahead it was able to formulate its moves. We simulated every move made by each chess engine using Lichess and noticed that there were observable patterns discovered throughout the course of these games. For example, one of the algorithms seemed to focus on takeover of the opponent's pieces, irrespective of the impact of the move. Other seemingly better algorithms inclined towards making better decisions that ultimately serve the purpose of attacking the opponent's king and thus, winning the game.

4.2 Results

Based on the study conducted, it is clearly visible that the Feed-Forward Neural Network performs significantly better than both the DFS and the MiniMax algorithms. The competition between the DFS and MiniMax algorithm was challenging but given the same depth, the DFS algorithm outperformed the Minimax algorithm.

Tournament	White Player	Black Player	White Wins	Black Wins	Tie
DFS Vs MiniMax	MiniMax	DFS	0	2	1
MiniMax Vs Feed-Forward Neural Network	Minimax	Feed-Forward Neural Network	0	3	0
DFS Vs Feed-Forward Neural Network	Feed-Forward Neural Network	DFS	3	0	0

Table 2: Experimental Outcomes

The Lichess platform [9] was used to allow multiple algorithm implementations to play against one another. However, the most important reason we chose the Lichess platform was to use its analysis tools that help us evaluate the overall game using graphs. The results of the analysis are as follows-

1. DFS(Black) VS MiniMax (White)

DFS succeeded in winning two games and the third was a tie. The DFS algorithm made smarter moves than the MiniMax algorithm. While the MiniMax algorithm largely relied on the pawn and queen pieces, the DFS algorithm used the knight pieces with combinations of either the queen or bishop pieces and hardly used the pawns. Both the algorithms seemed to think of future moves to kill the opponent pieces while caring less for their own pieces (a sacrificial tendency, of sorts) unless it was the queen. Surprisingly, the DFS did not seem to even care about the king. Ultimately, DFS had more accurate and better results than the MiniMax algorithm in these games.

Based on the study conducted, the following insights could be drawn from our graphical analysis of every game played-

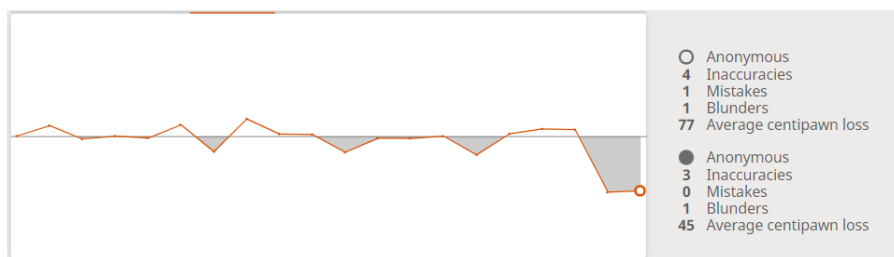


Figure 2: Analysis of Game 1.1

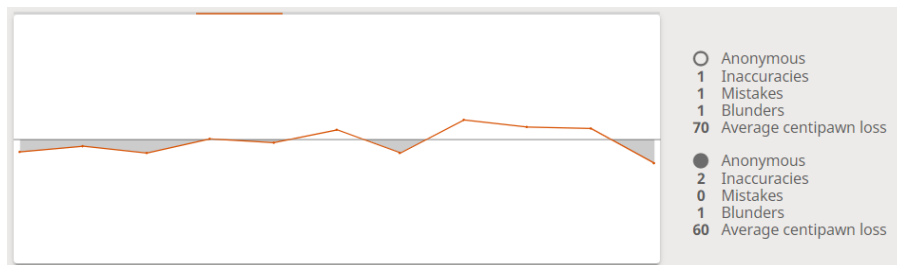


Figure 3: Analysis of Game 1.2

2. Minimax (White) VS Feed-Forward Neural Network (Black)

The Feed-Forward Neural Network won every game that was conducted. It dominated the games with smarter moves throughout the short duration of each game. Minimax concentrates on futile elimination of pieces over attacking the king but also fails to identify fairly obvious imminent threats to its own king.

Based on the study conducted, the following insights could be drawn from our graphical analysis of every game played-

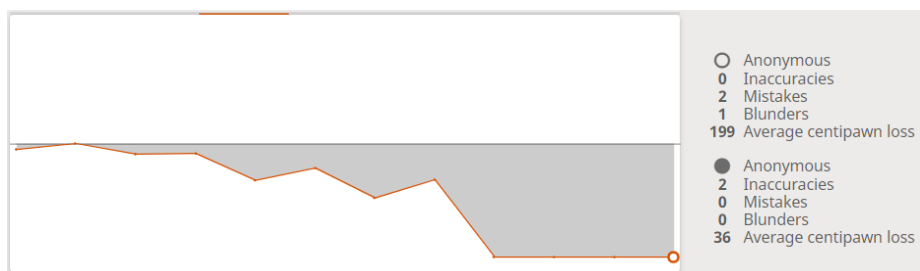


Figure 5: Analysis of Game 2.1

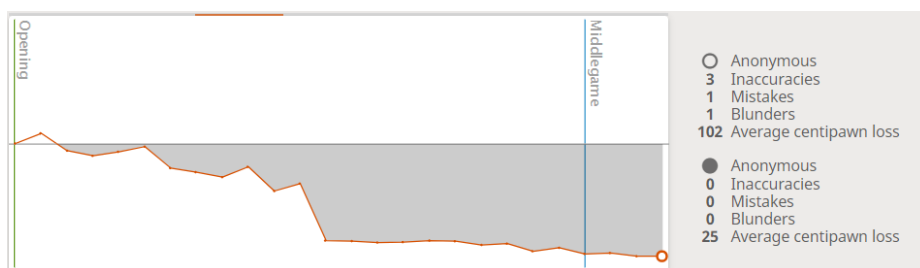


Figure 6: Analysis of Game 2.2

3. Feed-Forward Neural Network (White) VS Depth-First Search (Black)

The Feed-Forward Neural Network made smarter moves than the DFS algorithm due to the training process. It regularly attacked the opponent with 'check' moves to the king one way or another. The DFS algorithm used a more defensive approach. It was also observed that towards the end of each game, the DFS (Black) could not realize that its king piece had reached a 'checkmate' scenario and continued to play the game.

Based on the study conducted, the following insights could be drawn from our graphical analysis of every game played-

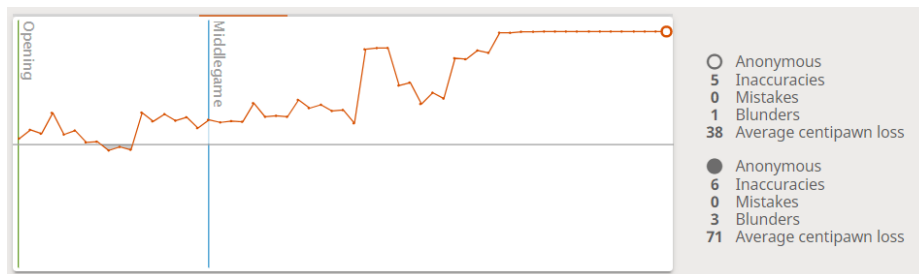


Figure 8: Analysis of Game 3.1

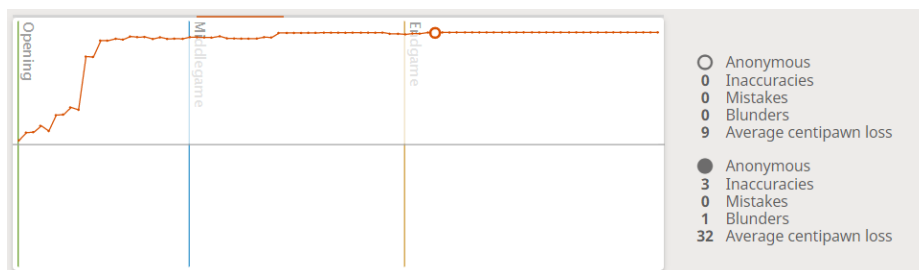


Figure 9: Analysis of Game 3.2

4.3 Discussion

From the above experiments, we evaluated the three algorithms based on their performance, speed, accuracy, number of steps and the smartness of the algorithm's decisions and moves. This can be verified using the total value of the remaining chess pieces from the table below. The study also analyses the nature of moves made, i.e. what pieces were generally attacked and sacrificed by the algorithm.

From the study conducted, the algorithms are evaluated based on the value of the remaining chess pieces in the table below-

Tournament	Game 1	Game 2	Game 3
DFS Vs MiniMax			
DFS	39	33	35
MiniMax	39	27	35
MiniMax Vs Feed-Forward Neural Network			
MiniMax	33	31	39
Feed-Forward Neural Network	36	34	39
DFS Vs Feed-Forward Neural Network			
DFS	17	5	22
Feed-Forward Neural Network	30	19	19

Table 3: Total values of the remaining chess pieces

Based on the observations in the above table and graphs, we can see that the Feed-Forward Neural Network implementation has outperformed the other two algorithms in terms of protecting its own pieces based on their importance i.e. value of the chess piece. It is also interesting to note a distinction between the DFS and MiniMax algorithm. Both the algorithms have almost equal value of the chess pieces when they play against each other but when they played against a more powerful opponent, i.e. the Feed-Forward Neural Network, it was observed that the DFS algorithm played for a longer duration of time while the MiniMax algorithm used a more defensive strategy of saving its own chess pieces instead of taking an offensive approach to winning the game. The DFS algorithm made optimal moves and used few unproductive paths. The memory requirement is minimal for the DFS algorithm as it only traverses nodes in the memory. Since the search states could occur repeatedly, it is also possible that the DFS algorithm might not reach the goal state at all in some instances. The MiniMax algorithm reaches its goal state in all the games. Since the algorithm visits every board state twice (first to record child nodes of state (if any) and second to calculate the heuristic values of nodes), it takes more time for execution, although it could be sped up by pruning. The Feed-Forward Neural Network performed very well with adequate training, This implementation leverages the use of CPU resources to perform multiple tasks at the same time, making it efficient, but also hardware dependent. It is also not possible to determine the training time for the Feed-Forward Neural Network, making it difficult to determine when the model has completed training.

5 Conclusions

In conclusion to the paper, our analysis shows that the Feed-Forward Neural Network performed better than both the DFS and MiniMax algorithms due to vast availability of training data that surpasses the 3-depth knowledge of the other two algorithms. Both, the DFS and MiniMax algorithms took more processing time and resources when the depth was increased to a value greater than 3, even leading the game to freeze for a few moments. However, the gains achieved due to increasing the depth was negligible when the two algorithms played against the Feed-Forward Neural Network implementation. The game between the DFS and MiniMax algorithms might be seen as even since they both use the DFS method for finding the best outcome while applying different rules to win. Minimax tried to minimize its loss whereas DFS tried to use the sum of the value of pieces to find a better move. The tie breaker that led to DFS having an advantage was the implementation of bonus positions in the board implementation that led the DFS algorithm to win the game. We can try to improve these results by increasing the depths and computing resources to the threshold at which the DFS and MiniMax algorithms come close to the Feed-Forward Neural network implementation.

References

1. "[History of Chess](#)". [Accessed 9 April 2020].
2. Stickel, Mark & Tyson, Mabry. (1985). An Analysis of Consecutively Bounded Depth-First Search with Applications in Automated Deduction. IJCAI. 1073-1075.
3. Nilsson, N. I. Principles of Artificial Intelligence. Tioga Publishing Co., Palo Alto, California, 1980.
4. Ontanon, Santiago. (2012). Experiments with Game Tree Search in Real-Time Strategy Games.
5. S. Koenig, "Minimax real-time heuristic search", Artificial Intelligence, vol. 129, no. 1-2, pp. 165-197, 2001. [Accessed 9 April 2020].
6. C. Dendek and J. Mandziuk, "A Neural Network Classifier of Chess Moves," 2008 Eighth International Conference on Hybrid Intelligent Systems, Barcelona, 2008, pp. 338-343.
7. Jie Si and Rilun Tang, "Trained neural networks play chess endgames," IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339), Washington, DC, USA, 1999, pp. 3730-3733 vol.6.
8. "[Minimax](#)". [Accessed 9 April 2020].
9. "[Lichess](#)". [Accessed 9 April 2020].