Building a chess engine

F20301_4 J 38_Batsaikhan Tuguldur

1. Introduction

Chess is a very complex and unique board game which the number of positions on the board is almost infinite. After each player makes a single move there are 400 possible positions. There are 5,362 distinct chess positions or 8,902 total positions after three moves (White's second move). The oldest recorded game was found in a 10th century manuscript. Humans have been playing chess and developing strategies for almost a thousand year until the chess computer, Deep Blue won a match against the world champion for the first time in 1996. Deep Blue's victory was considered a milestone in the history of artificial intelligence. Ever since then various chess engines were created with different evaluations and strategies and now none of the grandmasters can win against them.

Chess always intrigued me. Infinite number of positions are available on the board, and it feels very satisfying to reach a certain position I predicted, almost same feeling when you write a program without a bug. I used to watch a lot of videos on how complex and interesting chess engines are, and how differently they are programmed to play, but it never occurred to me to create a chess engine until I realized I am actually capable of doing it.

I tried two methods to write my chess engine. Even though first one was a failure, it was worth a try, and the second one is a success. I used python for both of my engine because of it has a chess library which provides all of the move generation and validation (board representation, illegal moves, castling, undoing a move and other common formats).

2. Attempt Number One: Neural Network

At first, I wanted to create a chess engine that plays like me, and it was possible because I had all the data I needed--1000 games played in an online chess website, lichess.org. Originally my plan was to use neural network and train the model with my own games, but it wasn't as simple as I thought. The most important part for creating a chess engine is how it evaluates a position and generates the next move, and it is where I failed. So how does an AI tell a good position from a bad position?

2.1. Board evaluation for neural network

Given a position, engines return a score of how good the position is from player 1's (player who chose white, player who moves first) perspective. This score is called an evaluation, typically shortened to eval, and the eval function is the soul of a chess engine. Let's say I put evaluations on each piece, pawn – 1 point, knight, bishop – 3 points, rook – 5, and so on. Thus, when the eval function only is determined by material points, eval function will say -1, if the black has 1 more pawn than me. But strong chess engines do not evaluate chess positions so easily, traditionally algorithms have measuring concepts such as material imbalance, piece mobility, king safety, etc. Whereas neural network-based chess engines work in a different way. They evaluate the position like human, based on the games they were trained on.

2.2. My neural network-based chess engine

I thought I could make an engine that plays like me by training the neural network with my game data. I had only 1000 games, and it wasn't nearly enough for any model to be trained. Chess games are put into computer by PGN file (Portable Game Notation (PGN) is a standard plain text format for recording chess games (both the moves and related data), which can be read by humans and is also supported by most chess software) [1]. Even though I managed to find my game data in PGN file, I couldn't implement it into the simple neural network I built.

After failing to use the PGN file I got from Lichess.org's[6] database, I gave up on creating a chess engine that plays like me, because even if I train the simple model I had with my games, it won't enough for it to play like me, and I will have to build more complex neural network. Considering the time and effort, I decided to give up on the idea, and moved on to create a traditional engine that works by algorithms, and I tried to put the ideas and strategies I use in my games to make it play more like me.

https://colab.research.google.com/drive/1XTHqxJ58zhh7EpCQWCHEp1R96FbyqPfQ

3. Attempt Number Two: Minimax Algorithm Implemented-Chess Engine [5]

Minimax algorithm is a decision rule used in artificial intelligence, decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst-case scenario. At each step, it assumes that player A is trying to maximize its chances of winning, and in the next turn player B is trying to minimize the chance of the player A winning. [4]

Minimax tree [2]

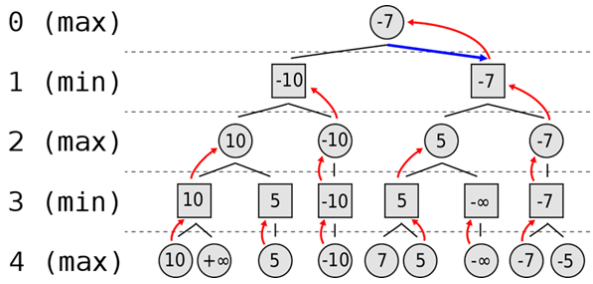

Figure 1

Therefore, in chess when using minimax algorithm, white will always try to maximize the evaluation score (Player A's score, the white's score), whereas the black will try to minimize it.

3.1. Board evaluation

In this chess engine board evaluation is based on two factors. First one is materialistically, meaning evaluating the chess pieces, and the second one is positionally, putting value in squares in which the pieces are on.

3.2. Materialistic score

Some pieces are stronger than others, that is why each piece needs to be assigned different values. I was taught pawn is 1 point, knight and bishop are 3 points, but in most cases, bishops are stronger than knights, and rook is 5 points, and the queen is 9 points. But in the engine the difference has to be made clear so the knight and bishop differently knight and bishop were given values respectively 320,330, and pawn = 100, rook = 500, queen = 900.

3.3. Positional score

Next depending on where it is standing some pieces are stronger than others. For example, it centralized knight should be more worth than rook sitting in the back rank. Basic principle to evaluate chess pieces based on. their positions is how they are centralized and how much square they are covering. Centralized pieces cover more grounds and hits more squares.

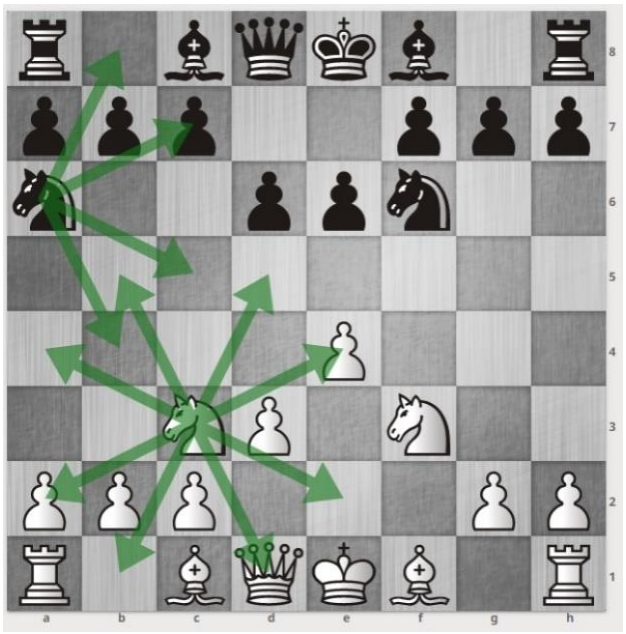

Figure 2

Black knight should have lower score because it is covering only 4 squares, whereas white knight is covering 8 squares, and has more squares available to move.

The following variables represent the positional score of the pieces: pawntable, knightstable, bishopstable, rookstable, queenstable, kingstable. To represent the board array of length of 64 is used for each piece, and each value of the arrays represents the positional scores of the respective pieces.

Following shows the positional scores for each piece and why they have different positional scores on the board.

pawntable =

```
[
    0, 0, 0, 0, 0, 0, 0, 0,
    5, 10, 10, -20, -20, 10, 10, 5,
    5, -5, -10, 0, 0, -10, -5, 5,
    0, 0, 0, 20, 20, 0, 0, 0,
    5, 5, 10, 25, 25, 10, 5, 5,
    10, 10, 20, 30, 30, 20, 10, 10,
    50, 50, 50, 50, 50, 50, 50, 50,
    0, 0, 0, 0, 0, 0, 0, 0]
```

Closer to being promoted, higher value the pawns get. And positions that has centralized pawns are stronger than the ones sitting in the back or sideways.

```
knightstable = [
    -50, -40, -30, -30, -30, -30, -40, -50,
    -40, -20, 0, 5, 5, 0, -20, -40,
    -30, 5, 10, 15, 15, 10, 5, -30,
    -30, 0, 15, 20, 20, 15, 0, -30,
    -30, 5, 15, 20, 20, 15, 5, -30,
    -30, 0, 10, 15, 15, 10, 0, -30,
    -40, -20, 0, 0, 0, 0, -20, -40,
    -50, -40, -30, -30, -30, -30, -40, -50]
```

More the knight centralized; stronger it is.

```
bishopstable = [
    -20, -10, -10, -10, -10, -10, -10, -20,
    -10, 5, 0, 0, 0, 0, 5, -10,
    -10, 10, 10, 10, 10, 10, 10, -10,
    -10, 0, 10, 10, 10, 10, 0, -10,
    -10, 5, 5, 10, 10, 5, 5, -10,
    -10, 0, 5, 10, 10, 5, 0, -10,
    -10, 0, 0, 0, 0, 0, 0, -10,
    -20, -10, -10, -10, -10, -10, -10, -20]
```

More the bishop centralized; stronger it is.

```
rookstable = [
    0, 0, 0, 5, 5, 0, 0, 0,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    -5, 0, 0, 0, 0, 0, 0, -5,
    5, 10, 10, 10, 10, 10, 10, 5,
    0, 0, 0, 0, 0, 0, 0, 0]
```

I like to put my rooks on the back rank because I tend to get checkmated on the back rank.

```
queenstable = [
     -20, -10, -10, -5, -5, -10, -10, -20,
    -10, 0, 0, 0, 0, 0, 0, -10,
    -10, 5, 5, 5, 5, 5, 0, -10,
    0, 0, 5, 5, 5, 5, 0, -5,
    -5, 0, 5, 5, 5, 5, 0, -5,
    -10, 0, 5, 5, 5, 5, 0, -10,
    -10, 0, 0, 0, 0, 0, 0, -10,
    -20, -10, -10, -5, -5, -10, -10, -20]
```

Queens should not stand in corners; they should be at the center and should be the backbone of attack and defense.

```
kingstable = [
    20, 30, 10, 0, 0, 10, 30, 20,
    20, 20, 0, 0, 0, 0, 20, 20,
    -10, -20, -20, -20, -20, -20, -20, -10,
    -20, -30, -30, -40, -40, -30, -30, -20,
    -30, -40, -40, -50, -50, -40, -40, -30,
    -30, -40, -40, -50, -50, -40, -40, -30,
    -30, -40, -40, -50, -50, -40, -40, -30,
    -30, -40, -40, -50, -50, -40, -40, -30]
```

Kings should not be traveling to the middle of the board; it is best to hide in the corners.

3.4. Evaluation Function

Finally, to calculate the scores following equations are used:

- `Equation 1: material = 100 * (wp - bp) + 340 * (wn - bn) + 350 * (wb - bb) + 450 * (wr - br) + 900 * (wq - bq)`

I change the score of the pieces from the common settings by how much I value them to make it play like myself. I decreased the score of rook by 50, increased the score of bishop and knight by 20.

Common setting is:

- `Equation 1: material = 100 * (wp - bp) + 320 * (wn - bn) + 330 * (wb - bb) + 500 * (wr - br) + 900 * (wq - bq)`

- `Equation 2: pawnsq = sum([pawntable[i] for i in board.pieces(chess.PAWN, chess.WHITE)])`
`pawnsq = pawnsq + sum([-pawntable[chess.square_mirror(i)]`
`for i in board.pieces(chess.PAWN, chess.BLACK)]) //goes for all the other pieces as well, bishopsq,knightsq,queensq,rooksq.`

- `Equation 3: eval = material + pawnsq + knightsq + bishopsq + rooksq + queensq + kingsqif board.turn:`
`    return eval`
`else:`
`    return -eval`

Equation 1 is for calculating material score by the number of differences between white and black pieces and their respective scores.

Equation 2 is for individual pieces scores by the square where the pieces are on.

The last one is for the final evaluation function which will return the summation of the material scores and the individual scores for white and negated when it is black. (Good position for White is Bad position for Black).

3.5. Finding the best move

In every move, the evaluation function will calculate the scores and return a value or a score, and the minimax algorithm will find the best score for the white by maximizing the score, whereas it will find the best move for black by minimizing the score.[5]

3.6. Alpha beta pruning

As the minimax algorithm searches for the best score by going through lots of calculation. Alpha beta pruning is a technique that eliminates unnecessary branches.

In simple terms, it is used to stop calculating positions that have been reached before. It saves a lot of execution time and space for memory.

```
def alphabeta(alpha, beta,
depthleft):
bestscore = -9999
if (depthleft == 0):
return quiesce(alpha, beta)
for move in board.legal_moves:
board.push(move)
score = -alphabeta(-beta, -alpha,
depthleft - 1)
board.pop()
if (score >= beta):
return score
if (score > bestscore):
bestscore = score
if (score > alpha):
alpha = score
return bestscore
```

3.7. Result

Game between chess engine and me

Following is the position after each side played four moves. /Black to move/

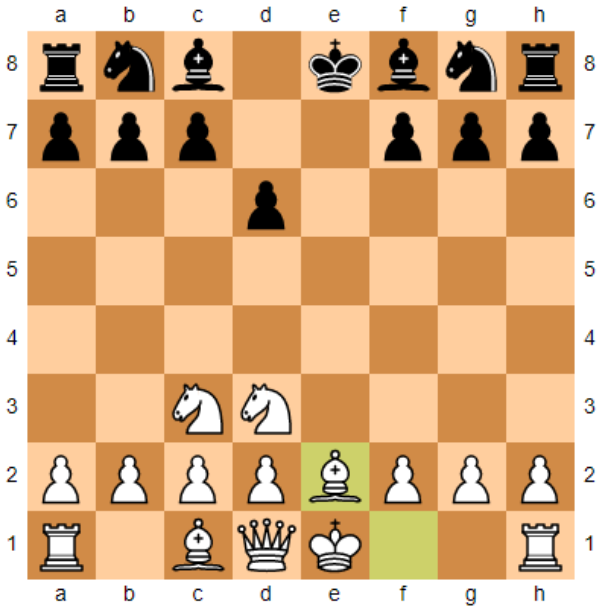

Figure 3

Evaluation for Black now is -990

After losing queen evaluation lowers by almost as much as the queen (900). The reason why it is so much lower is 2 knights and bishops are centralized and also the king is ready to be castled.

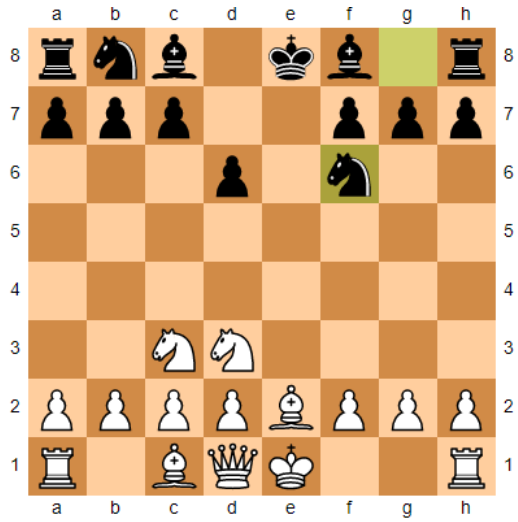

Figure 4

Evaluation for White is 940. Even though the number of pieces didn't change evaluation decreased by 50 because, knight's centralized its position.

Thus, it can be concluded that this chess engine works with algorithms that considers materials and positions.

5. Conclusion

Building neural network-based chess engine was more complex than I thought, and my biggest reason for failure is an insufficient knowledge of neural network. I thought I understood how neural network works after training titanic model, handwriting recognizer and cat and dog recognizer neural network. For those neural

networks, I didn't even bother to consider the portion of training samples, because all of them were given with considerable amount. And even if I did have enough samples another problem, I would run into was fixing the trained model to find only legal moves or rule out the illegal ones, so it was almost impossible for me to make one with the current knowledge I have of neural network. Therefore, I am quite satisfied with the decision to make a simpler chess engine.

At first, I was little disappointed in myself for not being able to develop neural network-based chess engine. But as I went on studying algorithms, it became more interesting. I never thought I would use the problems I solve in data structures and algorithm. But with minimax and alpha beta pruning algorithm, dynamic programming and data structures helped me to quite easily imagine how the algorithms work for decision making and furthermore, chess engine.

Even though my chess engine is considerably weak and doesn't play exactly like me I succeeded at putting some of my strategies and ideas into the engine such as valuing centralized positions and giving higher value for bishop and knight over rook in certain positions.

I was always intrigued by how the chess engine works, and I have seen many videos on programming chess engines but developing one by myself was completely different than I thought. It was much more complex and difficult yet very interesting. In the beginning I read about top 20 chess engines, and I thought "Oh I could do something like this", but in the end I couldn't even develop a simple neural network-based chess engine. But because of my failure, I understood the difference between having an idea of doing and practically doing it.

Code:

Code for the first failed attempt:
https://colab.research.google.com/drive/1XTHqxJ58zhh7EpCQWCHEp1R96FbyqPfQ?usp=sharing

Code for the chess engine:
https://colab.research.google.com/drive/1Ce3h_xcOuLva8almC8jHhKp62ttLyyWd?usp=sharing

Reference:

1. https://en.wikipedia.org/wiki/Portable_Game_Notation 2022/1/24
2. https://en.wikipedia.org/wiki/Minimax 2022/1/24
3. https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning 2022/1/24
4. https://www.youtube.com/watch?v=-ivz8yJ4l4E 2022/1/24

5. https://medium.com/dscvitpune/lets-create-a-chess-ai-8542a12afef 2022/1/24
6. https://lichess.org/