KnightSky: Learning

—

Aubhro

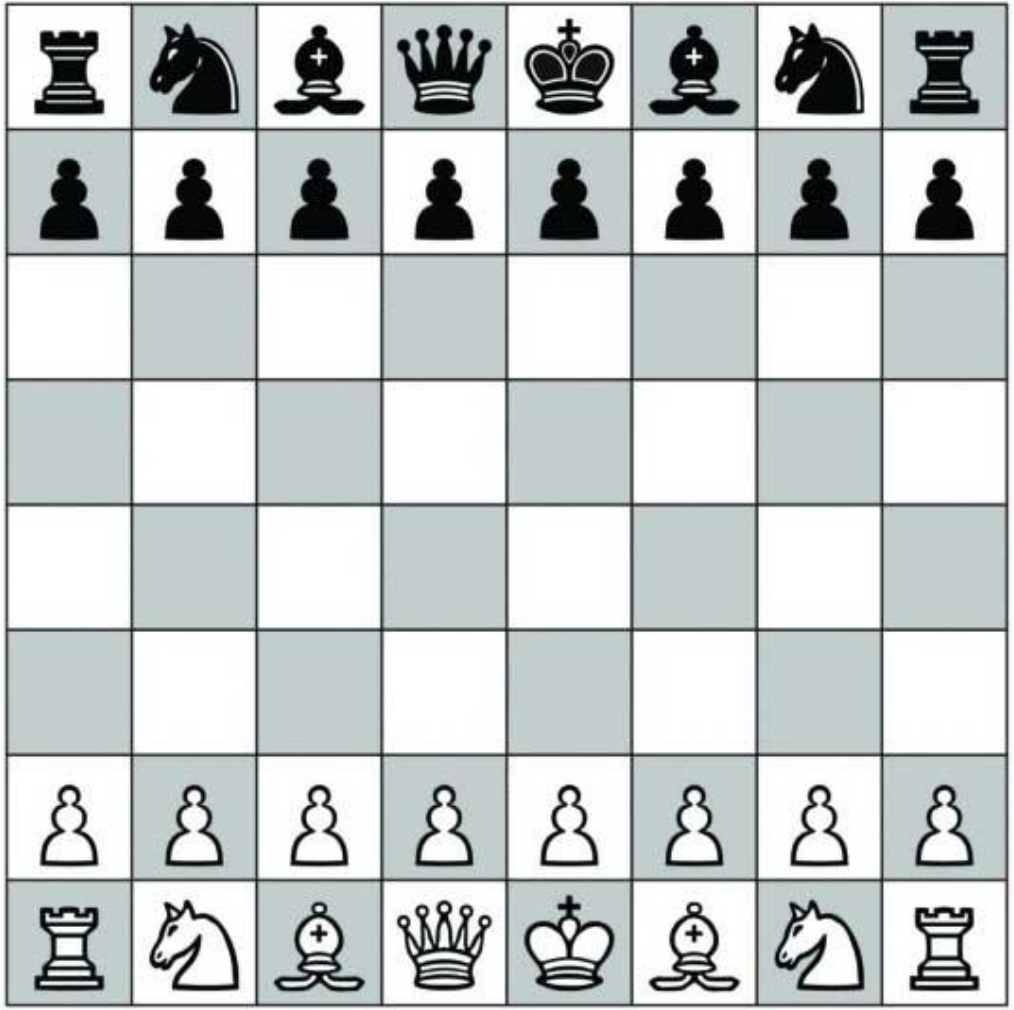
Chess Engine

—

Sengupta

About Me

- Senior at Lynbrook High School. VP of CS Club. I spend most of my time goofing off from boring things like homework, and instead spend that time coding.
- Github: https://github.com/LordDarkula
- Twitter: https://twitter.com/LordDarkula



How Chess Engines Work

- A chess engine accepts the position of the board as an input and outputs the move it thinks is the best

$$f(\text{Board}) = \text{Move}$$

Evaluation Functions

- Used to evaluate positions on a chessboard
- Usually sum of all your advantages minus sum of all your opponent's advantages
- At its simplest form, it uses the number of pieces times the worth of each piece to evaluate the position
  - Pawn - 1
  - Knight - 3
  - Bishop - 3.5
  - Rook - 5
  - Queen - 9

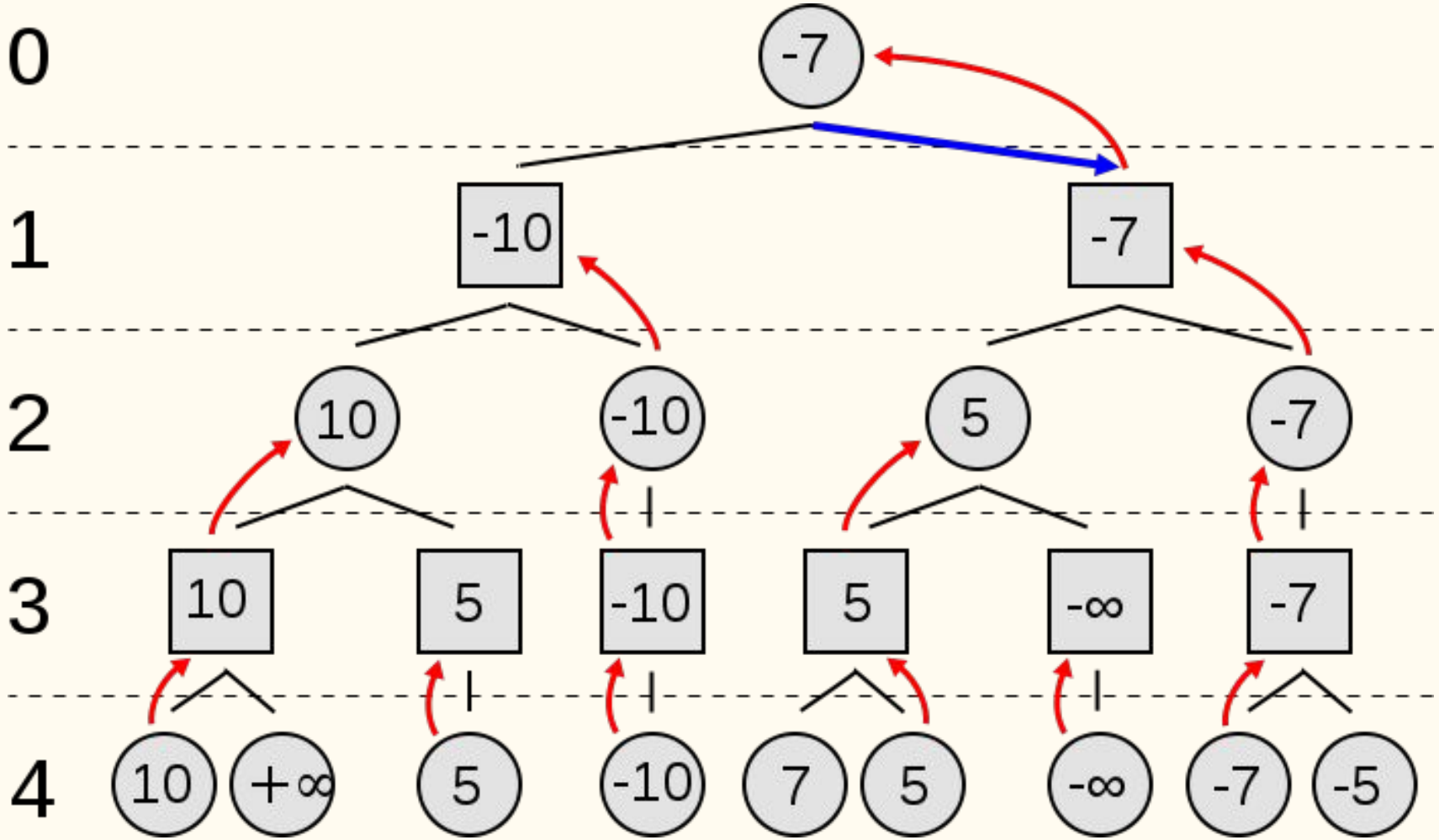
$$f(Board) = \sum \text{myPiece} - \sum \text{opponentPiece}$$

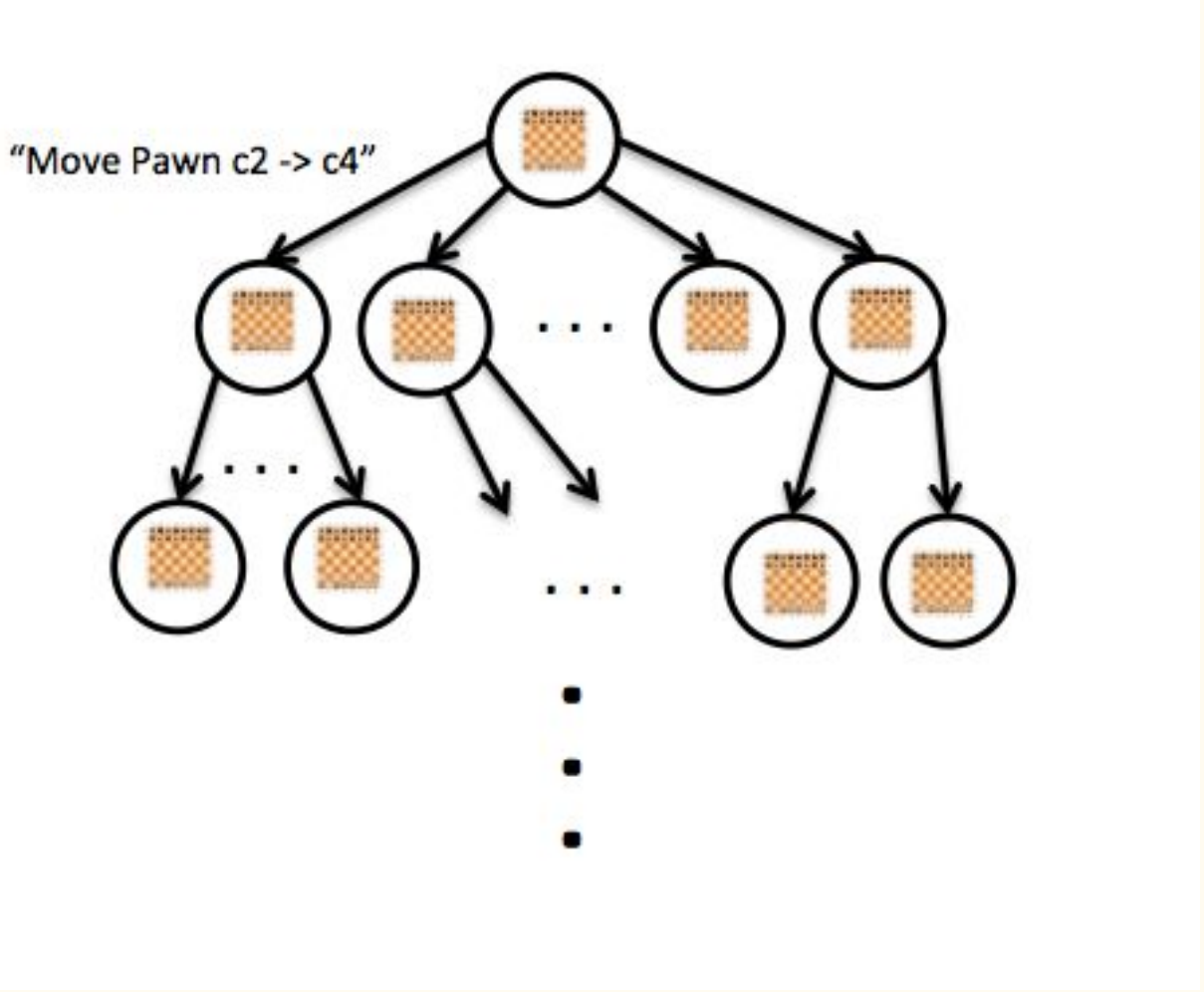
Minimax

- We do not care about gaining an advantage over the short term, but instead care about gaining an advantage over the long term
- f(Board) = max([f(all possible moves)])
- However, since it is nearly impossible to calculate every possible chess position, as there are around 10^43, we stop at an arbitrary point

Minimax Code Example

```
def depthSearch(self, position, depth, color):
        if depth == 1:
            return self.best_move(position, color)

        moves = position.all_possible_moves(color)
        for move in moves:
            test = position.copy()
            test.update(move)

            best_reply = self.depthSearch(test, depth=depth - 1, color=color.opponent())

            if my_move is None or my_move[1] < -best_reply[1]:
                my_move = move, -best_reply[1]

        return my_move
```


0
1
2
3
4
-7
-10
-7
10
-10
5
-7
10
5
-10
5
-∞
-7
10
+∞
5
-10
7
5
-∞
-7
-5


"Move Pawn c2 -> c4"

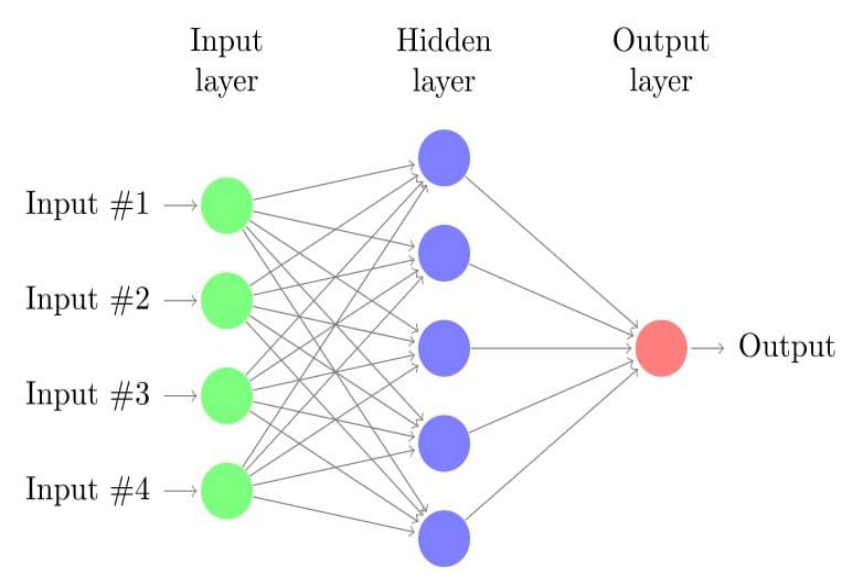
Evaluation Functions: The Problem

- Since chess is a complicated game, evaluation functions must also be complex
- Evaluation functions consist of hundreds of handcrafted lines of code to handle many different cases
- Because the code is usually crafted by human grandmasters, the engine is limited by grandmasters

Features and Labels

- Build a convolutional neural network to classify a chess position as good for white and good for black
- Each board will be fed in as a 8 x 8 x 12, as there are 12 types of pieces (6 white and 6 black)
- f([768]) = [2]
- We will use a one hot vector for the output ([0, 1] - white, [1, 0] - black)

The Science of a Neural Network



- Modelled after a human brain
- Consists of inputs, outputs, and neurons that fire when a certain threshold is reached
- Thresholds are modelled with activation functions
- Convolutional networks are a type of neural network that excel in image recognition

Training Data

- All chess games were downloaded from the FICS game database in PGN format
- Moves are read and converted to Move class
- Since games are by expert players, it is assumed that all moves made are optimal
- The position after each move is made is recorded as being good for that player

Specifications

- 3 Convolutional layers
- 1 Dropout layer
- 1 fully connected layer

```
x = tf.placeholder(tf.float32, shape=[None, 8 * 8 * 12], name='x_placeholder')
y_ = tf.placeholder(tf.float32, shape=[None, 2], name='y_placeholder')
keep_prob = tf.placeholder(tf.float32)
```

```
x_image = tf.reshape(x, [-1, image_size, image_size, 1])

W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

model = conv_layer(x_image, W_conv1, b_conv1, name='conv1')

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

model = conv_layer(model, W_conv2, b_conv2, name='conv2')

W_conv3 = weight_variable([5, 5, 64, 128])
b_conv3 = bias_variable([128])

model = conv_layer(model, W_conv3, b_conv3, name='conv3')

model = tf.nn.dropout(model, keep_prob)

W_fc2 = weight_variable([1024, 2])
b_fc2 = bias_variable([2])

y_conv = tf.matmul(model, W_fc2) + b_fc2
```

Future Improvements

- Allow for better and easier testing
- Make engine faster
- Use more training data
- Build feature to allow engine to play itself and train