Evaluating chess positions with machine learning

Sunwoo Choi
Christopher Dunn,
Erik McKelvey,
Matthew Proulx,

June 18, 2022

Abstract

Three different machine learning models were implemented to evaluate the odds of the different outcomes from any board position in chess. The same data containing thousands of chess games played by artificial intelligence chess players was used to train each model. Different values for the hyperparameters of each model were tested and compared, and the models were tuned to the parameters with the lowest error. Each of the models performed differently, but the convolutional neural network had the lowest mean squared error.

1 Introduction

1.1 Chess

Chess is a two-player board game played on a grid of 64 tiles. Each player has 16 pieces: 8 pawns, 2 rooks, 2 knights, 2 bishops a king and a queen. Players take turns moving one piece, and each piece moves in different ways. Pieces can be captured by moving a piece on to their tile, and the game is won when the opponents king can be captured. Modern chess was first played in the 16th century. Since then it has exploded in popularity and has become one of the most popular games in human history.

1.2 Data

A dataset containing 2370 chess games from the Free Internet Chess Server database was used to create a pandas dataframe. The dataframe contains a sample for every board position in the database, and 748 boolean features per sample. Most of the features represent whether a certain piece exists in a certain location, and the others represent the castling abilities, en passant locations, and a label with the result of the game. For each sample, the result is 1 if the player who's turn it is wins, 0 if they lose, and 0.5 if there is a draw. To ensure

that the pawns always move in a consistent direction, for every sample where it is blacks turn the board is mirrored over the horizontal center line.

2 Random Forest

A random forest is a set of decision trees each built with different data, resulting in a array of trees to predict by majority vote. The aim of a random forest is to reduce overfitting and reliance on a single feature. It does so by building each decision tree on different features, so the forest will have a more generalized estimation of the data.

2.1 Tuning

The performance of each forest is calculated using MSE (Mean Squared Error).

2.1.1 Forest type

Both classification and regression could work for the previously described data. Given the final goal was to build a program to evaluate the best next move given the board state, a classifier would have to be modified to be useful. The modified classifier built n decision tree classifiers, and the predictions would return the mean prediction from each tree, a pseudo classification which could show move preference. A classic sklearn random forest regressor was also built. To test them, both forests were were build with 5 trees and 75% features for each tree. The custom classifier won, with a 0.0420 MSE, slightly beating the classic regressor at 0.0467 MSE.

2.1.2 Hyperparameters

The two hyperparameters which have a large effect on the forest's predictions are the number of trees in the forest and the fraction of features available to each tree. The evaluation criteria was also tested with several different options, but no major changes were observed. Several random forests were built to test the number of trees parameter, with a constant 75% features. Several more forests were build to test the fraction of features, with a constant 50 trees. See the results from these trees in figure 1. It's clear that initially, adding more trees results in better results, but after 50 trees the improvement is so minimal that 450 additional trees will only reduce the MSE by 0.0008, which is not worth the time to compute the additional trees. From the tuning, the optimal random forest seems to have 50 trees and 75% of features, with mean_squared_error as the evaluation criteria.

2.2 Performance

Through several rounds of tuning, the optimal random forest for this problem is a custom classifier with 50 trees and 75% of features for each tree. The model

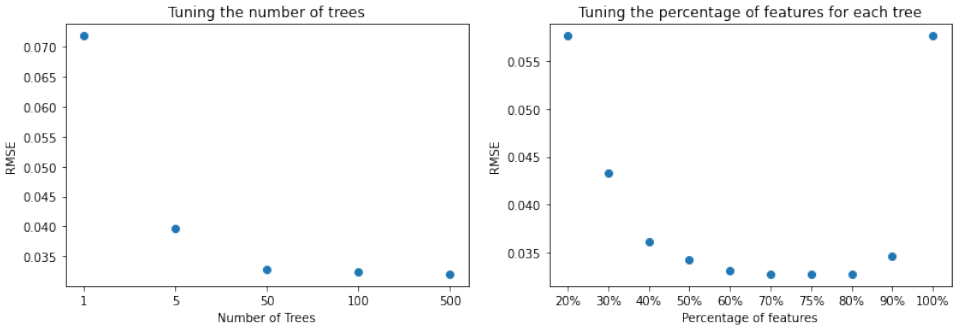

Figure 1: Tuning the random forest

built with these hyperparameters has a 0.0328 mean squared error.

3 Sequential Neural Network

An SNN (sequential neural network) is a series of algorithms that are used to recognize patterns and relationships in datasets. They can be applied in image recognition, business decisions based on previous results and chess move predictions. An SNN is usually made up of a set of layers each of which has a separate function.

3.1 Description

A five layer SNN was created with a layer structure as follows: Dense, Dropout, Dense, Dense, Dense (Output). A dense layer takes input from the previous layer, applies a function that will detect correlation between the different elements and applies a linear combination of all nodes in the layer to calculate the values passed to the next layer. The final Dense layer converts its input to have three dimensions so that the output can be compared to the label. Dropout layers are used to prevent overfitting by setting a fraction, know as the droprate, of nodes to 0.

3.2 Tuning

3.2.1 Dense

The SNN was tested with both the sigmoid activation function as well as the ReLU activation function and it was observed that there was a negligible difference between the two. After testing a network implemented with each function and a constant number of nodes the sigmoid activation function was selected. The SNN was then tested using varying number of nodes from 10 to 100 in increments of 10. It was found that the accuracy peaks at around 30 nodes.

3.2.2 Dropout

The dropout layer was tested with droprates ranging from 0 to 45% and it was found that they were most accurate at 25% as shown in figure 2. It can also be seen in figure 2 that the lowest loss was seen at a droprate of 5%. Since a droprate of 5% also has a high accuracy, this was the rate selected for the SNN.

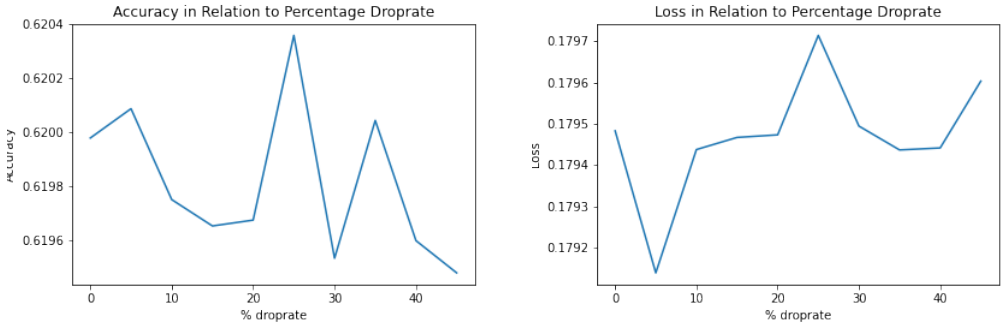

Figure 2: Accuracy and Loss in Relation to Droprate

3.3 Performance

Through running the test with different hyperparameters, we found that a 5% droprate, 5 layers with 30 nodes each gave us our maximum accuracy of 62%. Once the SNN has been fitted to the data, we can give it individual board states and it will tell us the win/draw/loss rate for the current player of that board state.

4 Convolutional Neural Network

A convolutional neural network (CNN) is a deep learning algorithm that takes grid data such as images, videos or chess boards. The CNN decides importance to various aspects to the input data and distinguish the difference between data. For chess, CNN takes the board and separates it into several parts to evaluate the board state.

4.0.1 CNN Model

A CNN was created with three convolution layers, one hidden dense layer and one output layer. The three convolution layers have the same parameters. The layers have 32 of output filters, the cardinality of the output space, in convolution and 6×6 kernel size. For all convolution layers, a droprate of 10% and the ReLU activation function are applied. Since the dense layer only accepts one dimensional inputs, the last convolution layer is flattened. The dense layer uses

ReLU activation function and has an output space cardinality of 64. The last output layer uses sigmoid activation function and has a one dimensional output space. Since the regression model cannot calculate the accuracy, the model is only dealing with Mean Squared Error.

4.0.2 Input/Output Data Structure

Since CNN takes multidimensional inputs, the input data structure is different than the previous models. The CNN takes 8×8 array with seven channels as input. Six of the channels represent the location of the six types of pieces on the 8×8 chess board. The current player's pieces are represented by a value of 1, the other player's pieces are represented by a value of -1 and the empty board position is represented as 0. The first channel shows the piece movement from the previous board. Since the CNN model assumes that it is the white player's turn, the previous movement in the first channel made by the black player is represented as -1.

The output is a single value in the range 0 to 1. The higher value stands for more advantageous board state to the current player and vice versa.

4.1 Tuning

In CNN, filters and batch size are the main parameters to decide the behaviour of CNN. We tried 3 different filters and batch sizes. Tested filters are 8, 16 and 32. Tested batch size are $2 \times 2, 4 \times 4$ and 6×6. Figure 3 shows that 32 of filters with (6, 6) batch size has the lowest mean squared error.

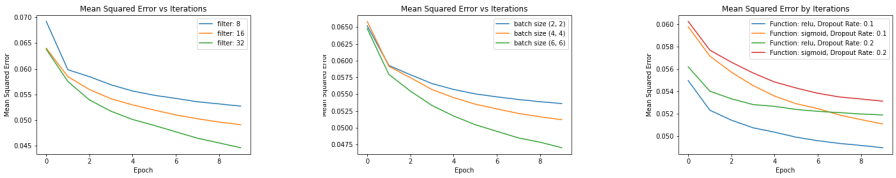

Figure 3: Loss in Relation to Tested Parameters

The model is trained with two different dropout rate and activation functions. The tested dropout rate are 10% and 20% each tested with with the ReLU and sigmoid functions. Figure 3 shows the mean squared error of models with different dropout rates and activation functions. The ReLU activation function with a dropout rate of 10% has the lowest mean squared error.

4.2 Performance

The CNN model is trained with parameters from tuning part. The model is updated using back propagation, after 32 samples are processed. The total number of complete passes through training data set is 10. To evaluate the model, we divide the data set into training set and test set. The data set is

randomly shuffled and 80 percent of data set is used for training the model and 20 percent of data set is used for testing the model. The final the MSE is 0.0296 for training data and 0.0302 for validation data.

5 Results

The goal for this project, was to create a machine learning model to accurately estimate chess positions. To this end, 2370 games of chess were evaluated and parsed out into 306208 different samples which would train our models. Three different models were developed and tuned, a random forest, a sequential neural network, and a convolutional neural network. Each model has different data and prediction formats due to the requirements from each model. As a result, it is difficult to directly compare the results of each model, but it seems like the random forest and convolutional neural network have a better accuracy than the sequential neural network.

5.1 Future work

Using machine learning to develop game playing algorithms isn't unheard of, and this work could be used as a baseline for that. Since the models can predict the likelihood of a win/draw/loss for certain board states, they could be used to create an AI that could play chess. Unfortunately, from the testing, selecting only the best move using these models will almost always lead to a draw. The problem is more complicated than simply choosing the best board, since there are things to consider like whether taking a piece is more valuable than not, or whether to castle at a certain point in the game. So in the future, these models could be used along with many other features to create a working chess AI that is capable of actually completing a game.

References

[1] Ludens, and Seberg. "Fics Games Database." FICS Games Database. Accessed August 10, 2021. https://www.ficsgames.org/download.html.
[2] Saha, Sumit. "A Comprehensive Guide to Convolutional Neural Networks - the eli5 Way." Medium. Towards Data Science, December 17, 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.