Chess Winner

Evyatar Or (302433966) Arie Kfir Hayne (ID. 205601172)

Submitted as a final project report for Deep Learning Course,
IDC, March 2021

1 Introduction

In this project we attempt to predict the winner of a chess game by the moves players make at the beginning of the game. note, our model does not try to predict the winner by the state of the board, but gets as input the list of moves the players make. We believe the list of moves and the responses that players make to each other hold more information about the game and players then the state of the board itself since it might give insight about the nature of the players.

eventually we got an accuracy of 64% which we consider to be good for reasons explained in the discussion section.

1.1 Related Works

There are a lot of works online related to chess, but we couldn't find any that are related to what we are trying to do. As baseline we used an architecture that aims to solve sentiment analysis and built our model with adjustments on it.

Initially instead of trying to predict the winner we considered trying to predict the player with the higher ranking, but eventually made the conclusion that trying to predict the winner will be more interesting and feasible, speaking only from personal amateur experience with chess, watching a game unfold we sometimes get the gut feeling of who is 'winning' but not necessarily who has more skill.

2 Chess introduction

2.1 chess format

Before we started working on the dataset we researched data structures that represent chess game as strings. We found that there are two common ways, PGN and FEN. FEN strings represent the board state and hold no memory on the previews states (a markov chain of states), PGN's are a simple representation of the moves by a list of short strings.

2.2 Explaining PGN notation

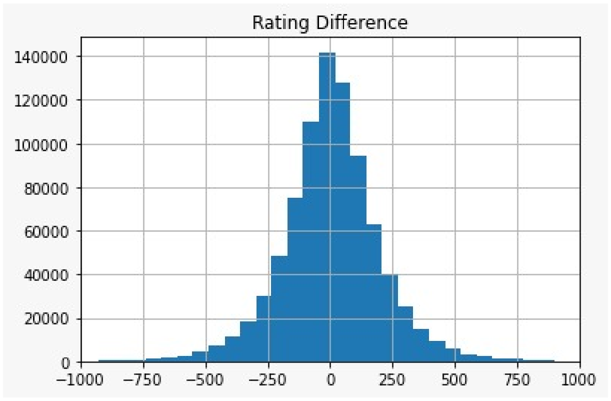
A move in PGN notation is represented with a few chars for example Bf4 means 'Bishop moves to f4', other set pieces are represented with different letters Rock(R),King(K),Queen(Q),Bishop(B),Knight(N), a pawn is represented with no char before the coordinates, for example c5 means 'pawn moves to c5'. Some additional information is given with other letters like 'x', for example Bxe7 means 'the bishop killed what was in e7'. More symbols give more information about the move done and others are reserved for special moves like castling (long and short).

A slice of a game represented in PGN notation for 14 first moves of a game: 1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6

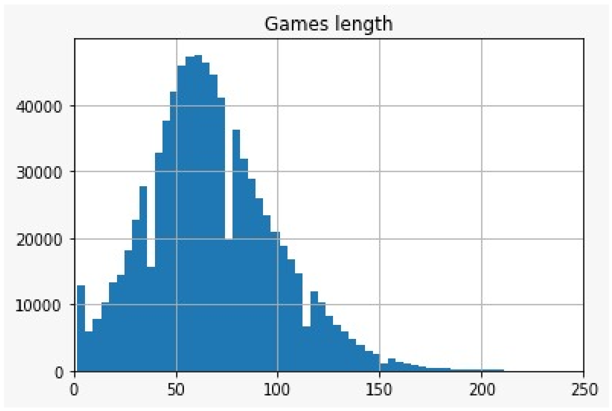
2.3 Data Processing and initial analysis

1. Our data was collected from https://www.kaggle.com/ironicninja/online-chess-games.

2. This dataset holds move-list, winner, and ranking for 843K chess games played by ranked players of different skill levels, as seen in the fig below.

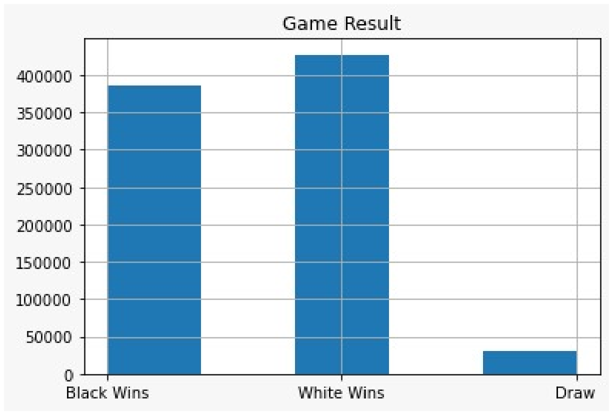


3. Examining the general length of games we found the graph below.



4. We removed games that have less or equal then 50 moves since we are going to try to predict the result of the games using the first 50 moves of the game.

5. While watching the dataset we see that there are not enough games that end in a draw. so we decided to remove these games in order to simplify the mission with more coherent data.

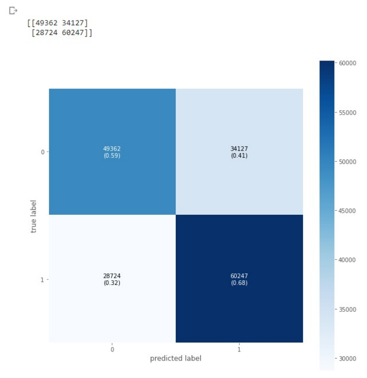


3 Solution

3.1 General approach

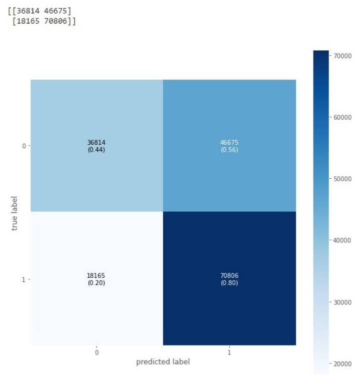
Our general approach was trying to run different models with different model configuration in order to incrementally find ones that yield better results. As said we adopted an architecture that aims to solve a sentiment-analysis problem and tried to experiment from there. We run our network testing its performance using LSTM vs GRU and found a slight advantage with GRU cells, additionally we experimented with different values of dropout and eventually found the value 0.5. In addition we tweaked with the layer sizes trying to find an optimal structure that would yield the best results and eventually found 64. Generally speaking, of course we could not try all the possible combinations to solve this problem, our method was to tweak a certain element of the network (hyper-parameter), find an optimal value for it and then move on and try to find an optimal value to the next hyper-parameter. This method of-course suffers from several blind spots in our search space for an optimal architecture.

Below is an example for our DropOut research stage, on that stage we still experimented with LSTM layer and found that a dropout of 0.5 gave us the best result, we kept this value in later stages even after tweaking other major configurations in the network (as said, not an optimal method, but we had a time limit, and this process can be endless), note that matrices are not even symmetrical in this stage of the research.



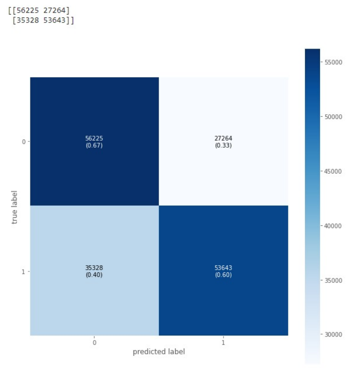
```
[[49362 34127]
 [28724 60247]]
```

Figure 1: drop out 0.3



```
[[36814 46675]
 [18165 70806]]
```

Figure 2: drop out 0.7



```
[[56225 27264]
 [35328 53643]]
```

Figure 3: drop out 0.5

3.2 Design

We used an open source project that originally implements sentiment analysis Models in TensorFlow(there is a link in Code References section). We tokenized the move set by calculating to max possible moves possible under the PGN notation which approximates to about 3000. Our NN is comprised of a main layer of GRU cells with a dropout of 0.5 (which experimentally gave the best results). As we said in the previous section we run two different model each of them we run we different configuration.

3.3 Configurations

our model took the following configurations

model variables config

1. dropOut - we run the NN with different value of dropOut. 0.3,0.7,0.5
2. LayerSize - we tried to run the NN with different size from the set of 32,64,128
3. learning rate - 0.01 / 0.001 / 0.0001
4. optimizer - Adam / RMSprop / Adamax

We run it for 6 epochs each epoch running for about 5 minutes, about half an hour per configuration.

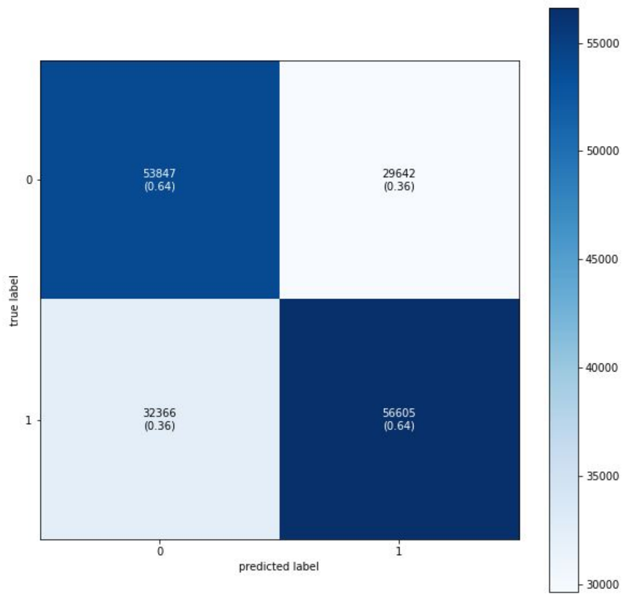
4 Experimental results

While settling on the architecture that maximize the accuracy we got to a score of 0.64 on our test set but while working on different model configurations we noticed something worrying,

While our training accuracy kept increasing as the training stage progressed from epoch to epoch our validation accuracy kept fluctuation up and down not improving along with the training accuracy. This is a hint to the possibility of and over-fitting limit that we reached at some point in the training stage.

Eventually on our final model both the training accuracy and the validation accuracy kept increasing along side each other from epoch to epoch. For our confusion matrix we got some encouraging results:

```
[[53847 29642]
 [32366 56605]]
```



As we should expect the confusion matrix is symmetrical since there should be no major preference between successfully predicting ’white wins’ versus ’black wins’. We can expect a minor preference to ’white wins’ just because there are more white wins then black wins, so the network will be safer to ’bet’ on white wins more often.

5 Discussion

An important issue that we need to note is that even given a very good human observer an expert experienced chess player, even this skilled human might not be accurate in predicting the winner by the initial set of moves, so our model is probably limited by a 'glass ceiling' of what is theoretically possible. Eventually many chess games are won and lost following a critical mistake and those are made by humans, under conditions that are hard to predict. Seeing that we got an accuracy higher then 50% is a win for us under these conditions.

While working on the project we did experiment with trying to create a predictions from the first 40-60 moves. We found an increment in accuracy the more moves we add, this is a good sign that indicated that our model is indeed using the information in the moves list. Since we could see that the more moves the model gets the more accurate the it becomes.

We can think of several ways to try to improve the model, like farther exploration of combinations for dropOut, optimizer, additional layers and layers sizes, along with adding more games data and longer training.

6 Code References

- Colab notebook -

  `https://colab.research.google.com/drive/1keViuo-lWhOsTCIYSwF6xouoaikWLpXI?usp=sharing`

- Database - `https://www.kaggle.com/ironicninja/online-chess-games`

- sentiment-analysis baseline - `https://www.kaggle.com/imranzaman5202/twitter-sentiment-analysis-using-nn`