Predicting Chess Game Results using Neural Networks

Jonathan Ben Dov (ID. 311525448), Ilai Genish (ID. 207480773)

Submitted as final project report for Deep Learning, IDC, 2020

1 Introduction

Chess has been an area of vast research in the area of Deep Learning and artificial intelligence overall. The forefront of this research today is done by Google's deepmind group, but other leading researches in the past years include the paper by Oshri and Khandwala, and David, Netanyahu and Wolf [1]. These researches dealt mainly with prediction and automation of moves in a match and using neural networks to calculate the most efficient move.

We both are avid chess enthusiasts, and one of us played in a team in his childhood, thus researching this issue came naturally for us given our passion for it and its connection to Deep Learning.

We tried to deal with a problem which has not been researched vastly - given the beginning moves of the players, and the batch of first few number of moves, as well as other metadata related to the specific game and its participants, try to use neural networks to predict the outcome - white wins, black wins, or a draw between the two. The problem of classification of the result of the match has been researched, but not in the context of relation to the first n moves and their effect. The dataset we found is fairly new (2017) and belonged to the lichess app, a popular mobile app used for competitions and matches (which we use ourselves), and is composed of over 20,000 games. This dataset required extensive data engineering on the dataset (as will be described).

We tried to tackle this problem with several architectures - firstly, using a fairly simple architecture used on similar problems, then added LSTM layers and managed to model the problem to the best suited one, achieving accuracy of 65 percent.

2 Solution

2.1 General approach

2.2 Related Works

Pulido[2] tried to tackle this problem with the use of 20 movements, and using only machine learning algorithms - Random Forest and SVM, without neural networks. He reached 57 percent accuracy on test set.

Fan, Kuang and Lin (Stanford)[3] reviewed this problem given datasets from the international World Chess Federation, which dealt with player ratings during a period of several months as a time series. This gave an accuracy result of 55 percent.

Furthermore, when performing reduction to this problem to other result prediction problems, viewing result prediction given the game data in a certain time frame, Purucker[4], who gathered data on NFL matches from the first eight rounds of the competition using five features, consisting of yards gained, rushing yards gained, turnover margin, time of possession, and betting line odds managed to predict the result with a neural network model. This gave him 61 percent accuracy compared with 72 percent accuracy of the domain experts.

In comparison, Tax and Joustra[5] tried to compare various machine learning models to predict Dutch Football results using various match features as well as betting odds, and managed to get a classification accuracy of 54 percent with a neural network architecture. In comparison, Wiseman[6] predicted PGA golf scores based on the initial results of the beginning of the match with regression using a neural network (different task but similar in terms of data on the beginning of the competition), receiving a result close to the actual one in 67 percent of instances.

Finally, Challa, Moskovitch[7], who used a 3 layer neural network and an LSTM with data collected on Premier League football matches, on all participants of each team. This data was manipulated and engineered to show the individual player performances, and predict using the aforementioned architecture the result. This achieved 51 percent accuracy on test sets.

3 Solution

3.1 General approach

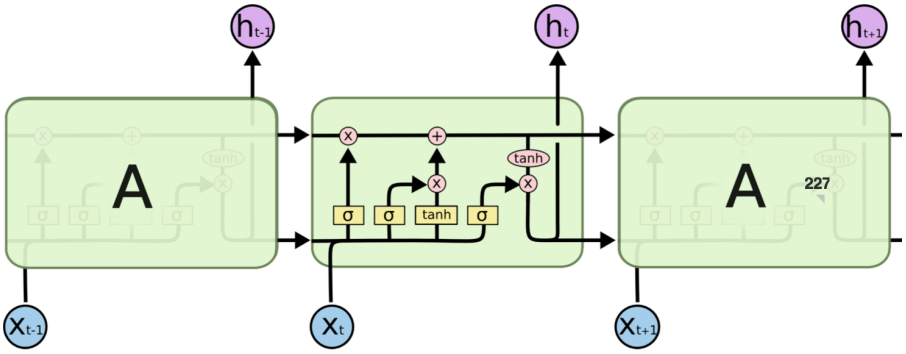

Figure 1: LSTM structure and effect of previous layers

The problem we were facing is a classification problem, and the dataset we received consisted of various features which will be explained further on, our approach was as follows - we will manipulate the data to suit our classification problem, and try different architectures to match this.

We knew that this problem is fairly similar to a time series-problem, as every move of the first twenty moves affects the next move. Given this fact, we knew that adding an RNN such as an LSTM layer could make sense, but decided to still test different architectures, and maybe reach a conclusion that an LSTM layer is useless and costly in our case, just as Challa, Moskovitch reached this conclusion, or see that our hypothesis is correct and what type of network will suit in the optimal way.

Furthermore, we wanted to feed it with a certain amount of changed number of moves to test and show that as it learns more moves, the network will know in a better manner how to predict the winner.

3.2 Data Engineering

Our dataset consisted of 20,000 games played in the lichess app, which had the following features: whether it was rated or not, the ranking of each player, the full list of the moves played in the game, the number of moves, the beginning time of the game, the end time, the increment rule(this is a timing frame in chess) the opening moves name (their nickname), and finally the result - white wins, black wins, or a draw.

We manipulated the data as follows: we firstly encoded the label to numerical values (so the model could run on it). Next, we encoded the open move feature, which was by the total amount of moves that appeared in all matches

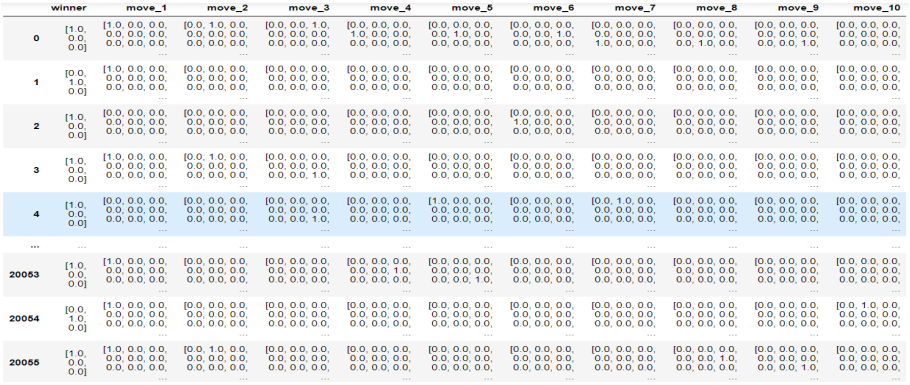

Figure 2: Final dataset - Every instance has 10 columns, each containing one hot encoded vector by the move played

Finally, we added 10 features, which will be the 10 first moves(and later on changed to 20,30,40 and 50 by our experimentation). As the moves where strings portraying the position moved from and to, we needed to one hot encode them and then return the corresponding value in the right move column. In order to make up for games which had lower than the amount of moves, we padded them to complete to the minimum required with 'null' values.

3.3 Design

Since we had some past experience using Keras, we decided to use it to implement our neural networks. Firstly, we decided to measure two general architectures, and inside each architecture measure its robustness(i.e. adding more layers) and measuring the change given change in various hyperparameters. This is the main outline of our experimentation:

- Model Architecture: **CNN** vs. **LSTM**
- Number of moves(10,20,30,40 and 50)
- Epoch number until convergence / prevention of overfitting
- Addition of regularization layers
- Batch sizes

We eventually narrowed it down to two main networks, in which both indicated that increasing the complexity of the network (adding more layers) did not yield any improvements to performance metrics.

We tried to use another LSTM layer to the second model, but this did not effect our results. It took us 5 epochs to train the first model, and 5 as well to train the second. Note that we could have trained the network for more epochs

and thus achieved a training accuracy as high as 95 percent, but we noticed that this was clearly a sign of overfitting and thus reached the optimal value when decreasing the epoch number and achieving lower training accuracy.

In terms of batch size, we experimented with batch sizes of 1, 16 and 64, and reached the optimal result with a batch size of 16.

4 Experimental results

We experimented on several architectures, but decided to portray the following two:

The first network we built was a fairly simple neural network, which consisted of convolution layers with relu activations, as well as linear layers as shown in the figure. We used categorical cross entropy as our loss result, which was best suited for this problem, and the adam optimizer as well as a softmax(as we are not dealing with binary classification).

The second network (and successful one) consisted of a different model with an LSTM layer as stated. The model consisted of the LSTM layer, relu activations and a dropout layer.

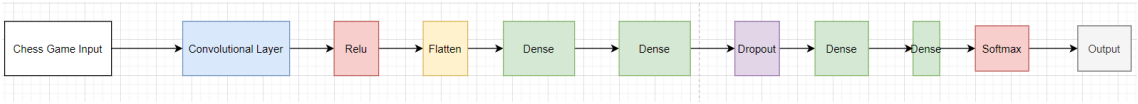

Figure 3: Model 1 - CNN

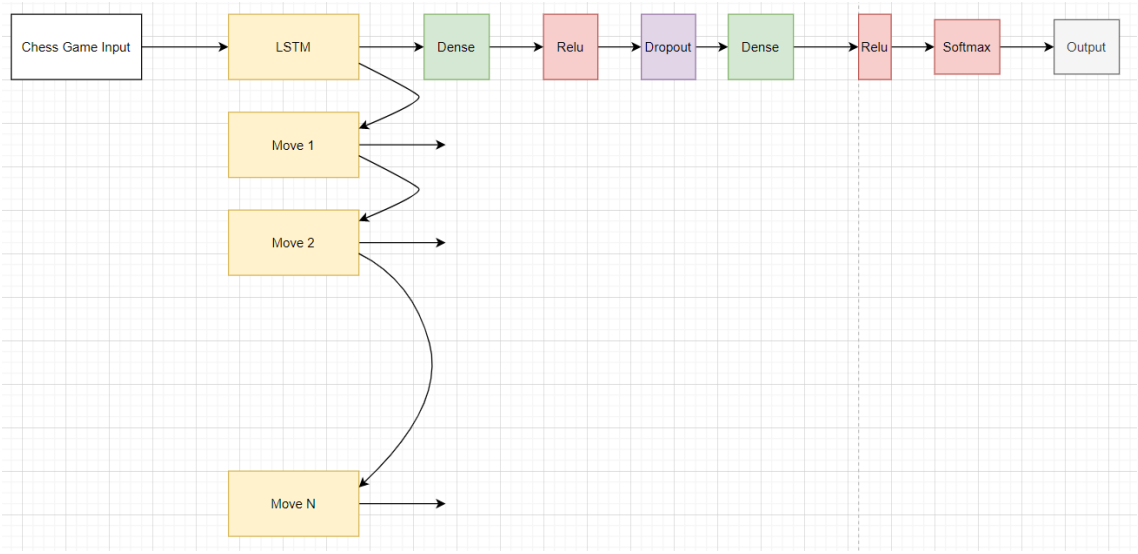

Figure 4: Model 2 - LSTM

The best result achieved for the first model was after learning the 50 first moves, and this achieved 79 percent accuracy on training, 60 validation and 59 on training.
For model 2, we reached the best result yet again after running the network for a dataset of the 50 first moves. This gave us 69 percent accuracy on training, 59 on test and 60 on validation.

Finally, when testing the amount of moves, we saw a direct increase in accuracy given the network learned more moves. This means that the more moves from the beginning of the match the network learned, the better it learned and predicted the outcome of the game.

LSTM

Moves	Training	Validation	Test
10	55	52	54
20	62	56	56
30	62	60	59
40	69	63	62
50	65	64	59

CNN

Moves	Training	Validation	Test
10	55	52	53
20	62	56	56
30	62	56	55
40	75	62	64
50	77	63	65

Figure 5: Accuracy Compared to number of Moves

5 Discussion

Given the complexity of the game of chess, after only 5 moves we have 4,865,609 number of game possibilities (by Shannons number[8]), we expected not to have a complete prediction, but to try to give a perspective as to how Deep Learning can try to help progress to solve this problem. Because of this high unpredictability this research could help improve decision making for participants upon making the beginning moves and choosing their beginning strategy, which

is practiced and analyzed widely by chess players. Thus, given this complexity, reaching the results we did is a fair progress to work upon. Furthermore, we believe that since our dataset was medium sized (20,000 instances and less after we filtered out) an addition of more data could have had some effect on our data.

Given the increase in accuracy when the network learned more moves, more research in this subject could decrease the level of unknowns to which chess players deal with when making a play.
Furthermore, given that many top researches nowadays implement robust networks to deal with various problems, we found the opposite - increasing the amount of layers did not yield any significant results, and our experimentation was mainly around the various hyperparameters of the network.

Furthermore, we believe further research could measure the change in even more moves and their effect on the accuracy (i.e. choosing even more moves than we chose) which would take more computational power, but this of course could only be possible with more significant computational power than we had available in this research. In addition, one could try to find the feature importance of various moves (i.e. how a certain move made a significant change in the prediction), and thus try to find importance's of certain moves which might have seemed minor in significance, as our prediction is currently more of a 'black box'.

6 Code

The following link contains a GitHub repository containing both the dataset and the notebook with the models and experiments:

https://github.com/jonathanbd135/Chess-Winner-Prediction-Using-Neural-Neworks.git

References

[1] David, Netanyahu and Wolf, DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess.

[2] Hector Apolo Rosales Pulido - Predicting the Outcome of a Chess Game by Statistical and Machine Learning techniques, Universitat Polit'ecnica de Catalunya, Barcelonatech

[3] Chess Game Result Prediction System,Zheyuan Fan, Yuming Kuang, Xiaolin Lin, Stanford University

[4] M.C.Purucker, Neural network quarterbacking

[5] Tax and Joustra, Predicting The Dutch Football Competition Using Public Data A Machine Learning Approach

[6] O. Wiseman, Using Machine Learning to Predict the Winning Score of Professional Golf Events on the PGA Tour (National College of Ireland), 2016.

[7] Challa, Moskovitch - A Sure Bet - on predicting outcomes of football matches, Stanford University

[8] Claud Shannon , XXII. Programming a Computer for Playing Chess 1.