CS230

Chess Agent Prediction using Neural Networks

John Dalloul
Stanford University
jdalloul@stanford.edu

Mark Bechthold
Stanford University
markpb2@stanford.edu

Abstract

This project aimed to use neural networks to predict the agents of a chess game, i.e. whether the player with white pieces is a human or computer and whether the player with black pieces is a human or computer. Several different neural network architectures were assessed, including naive logistic regression, dense networks with varying numbers of hidden layers, and convolutional networks with 3D convolutions. The most successful architecture was comprised of two 3D convolutions prior to six fully-connected hidden layers and a four neuron output layer, using ReLU activations following each layer; this architecture achieved an accuracy of 79.3% on the test set (4,004 games; 1,001 from each class of human/computer as white and human/computer as black), significantly higher than the majority classifier accuracy of $\sim$25%. Further experimentation with hyper-parameter tuning for this architecture and more training data are promising avenues to increase performance on this classification task.

1 Introduction

Humans have played chess for centuries and the dominant styles of chess have evolved over the years just as the theory of the opening, middle game, and end game has changed. In recent years, chess engines have been developed that consistently outperform the best human players in the world, and other engines have been developed to play against average human players on online applications. Recently, with the release of the Netflix show *The Queen's Gambit* and most people staying home during this past summer, the popularity of chess has increased dramatically, and as more and more people play chess online, the use of chess engines has become more prevalent.

This project aims to use neural networks to determine whether the players in a game of chess are human or AI based on their moves throughout the game to see if human and AI styles of play are distinguishable. This could reveal important aspects of chess that are not immediately obvious based on existing theory. In addition, it could be possible to flag certain players in online games to be investigated for cheating if their moves tend be more AI than human. Specifically, to achieve our aims, the input to our algorithm is a set of consecutive positions form a chess game. We then use our neural network to output a predicted classification of the two players as either human or AI.

2 Related work

There are many AI-related projects in chess from the creation of chess engines to play against humans[1] to the analysis of existing games using neural networks[2]. Existing AI projects in chess have dealt with computers attempting to predict human moves[3]; some engines have also been built on neural networks that learned various patterns to predict moves[4]. In the game of Go, there

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

exists a statistical analysis to identify human vs AI players[5]. Some research has discussed the challenges associated with identifying cheating in chess games[6]. However, there does not seem to be an existing AI vs. human neural network classifier for chess. We will try to leverage existing data sets of chess games and will look at existing chess engine neural networks to inform our classifier architecture.

3 Dataset and Features

The initial input dataset[7] was comprised of 8,141 games of chess broken up into 3 categories: human vs human ($\sim$ 3,400 games), human vs AI ($\sim$ 1100 games), and AI vs AI ($\sim$ 3,600 games) with all human players having elo ratings between 1800 and 1999 since these players represent a subset of very strong players that are not at a grandmaster or international master level. In the original dataset each game had headers providing contextual information such as elo rating, whether or not white or black was an AI, date and time, etc. After this header, there were the sequence of moves in standard chess notation as shown in (Figure 1). Using the python-chess library we cleaned the data, filtering out games that had fewer than 40 total moves (number of moves by white + number of moves by black) as games with fewer moves seemed more challenging to classify since the search space at the beginning of a chess game is much smaller; because of this, AI and human play is less likely to diverge. We then constructed sparse arrays of dimension $(8 \times 8 \times 6)$ for each board configuration in a given game where the 8×8 dimensions represented the chess board dimensions and the depth of 6 represented the 6 types of chess pieces where layer 0 represented pawns, layer 1 represented rooks, layer 2 represented knights, layer 3 represented bishops, layer 4 represented queens, and layer 5 represented the king. We used a 1 to indicate that a white piece was present on a given square, a 0 if no piece was present, and a -1 if a black piece was present (Figure 2). For example, a '1' at index (5, 5, 3) would represent a white bishop at position (f3) on a chess board. A single game was then represented by an $(8 \times 8 \times 6 \times 40)$ matrix where there are 40 total moves in a game. Each game was labeled with either 00, 01, 10, or 11 - 00 for human v human, 01 for human v AI, 10 for AI v human, and 11 for AI v AI where the first position shows the nature of the player with the white pieces and the second position shows the nature of the player with the black pieces. For training our neural network, we split the dataset into a training set comprised of 80% of the inputs from each of the game categories. Our validation set is comprised of the remaining 20% of games from each of the human vs human, AI vs human, and AI vs AI categories. For early iterations we started with a validation set and training set. After confirming that our network was working correctly, we expanded the dataset to consist of 8,000 examples from each class in the training set, 999 examples from each class in the validation set and 1001 (due to an off by 1 error) from each class for the test set.

```
[Event "FICS rated standard game"]
[Site "FICS freechess.org"]
[FICSGamesDBGameNo "471871422"]
[White "kubbybulin"]
[Black "Enyerbrok"]
[WhiteElo "1856"]
[BlackElo "1894"]
[WhiteRD "28.7"]
[BlackRD "45.7"]
[TimeControl "900+0"]
[Date "2020.05.03"]
[Time "23:14:00"]
[WhiteClock "0:15:00.000"]
[BlackClock "0:15:00.000"]
[ECO "B10"]
[PlyCount "125"]
[Result "1/2-1/2"]

1. e4 c6 2. Nf3 d5 3. exd5 cxd5 4. d4 Bg4 5. Be2 Nf6 6. Nc3 a6 7. Be3 e6 8. Qd2 Bb4 9. a3 Ba5 10. b4
Bc7 11. h3 Bxf3 12. Bxf3 Nbd7 13. O-O O-O 14. Na4 Qe7 15. Rac1 Nb6 16. Nxb6 Bxb6 17. c4 dxc4 18. Rxc4
Rad8 19. Rd1 Nd5 20. Rdc1 Nxe3 21. fxe3 h6 22. Qe2 e5 23. Qd3 Rfe8 24. Qe4 exd4 25. Qxe7 Rxe7 26.
exd4 Bxd4+ 27. Kh1 b5 28. Rc6 Bb2 29. Rb1 Be5 30. Rxa6 Rde8 31. Ra8 Bb8 32. g4 Kh7 33. Bc6 Re1+ 34.
Rxe1 Rxe1+ 35. Kg2 Be5 36. Bxb5 Ra1 37. Bd3+ g6 38. Ra7 Bg7 39. a4 Ra2+ 40. Kf3 Ra3 41. Ke4 f5+ 42.
gxf5 gxf5+ 43. Ke3 Kg6 44. b5 Rb3 45. a5 Rb4 46. b6 Bd4+ 47. Ke2 Bxb6 48. axb6 Rxb6 49. Ra5 Rb2+ 50.
Kf3 Rb3 51. Ke3 Kg5 52. Rxf5+ Kh4 53. Rf3 Kg5 54. Rf5+ Kh4 55. Rf3 Ra3 56. Rf4+ Kg3 57. Rg4+ Kxh3 58.
Rg6 h5 59. Kd2 h4 60. Bf5+ Kh2 61. Rh6 h3 62. Rxh3+ Rxh3 63. Bxh3 {Neither player has mating
material} 1/2-1/2
```

Figure 1: An image of an example game from the initial uncleaned dataset.

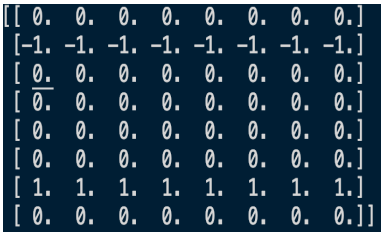

Figure 2: An image of the starting board configuration looking at the pawn slice of the $(8 \times 8 \times 6)$ matrix representing a board state.

4 Methods

We used Pytorch to implement a basic neural network to run on our dataset. Our inputs were the games, and our output were the class labels. We elected to start with a set of fully connected hidden layers rather than performing a 3D convolution. Despite the fact that this problem and our input data structure seemed to lend itself nicely to convolution we wanted to establish a baseline with a more rudimentary network for comparison first. To implement the fully connected layers, we permuted our original matrices from the dataset into matrices of dimension $(6 \times 40 \times 8 \times 8)$ which we then flattened, creating 15,360 features, and used the cross-entropy loss function with L_2 regularization. Thus our training data was of the form $(m \times 6 \times 40 \times 8 \times 8)$ where m was the number of training examples. We used ReLU activation functions between hidden layers and included biases for each neuron, performing a Softmax of the final layer to classify the inputs. We used Adam as our optimization algorithm, keeping the default beta parameters. Then, we tried a variety of hyper-parameters to determine the optimal number of hidden layers and the number of neurons in these hidden layers. Flattening the input matrices seemed like the best approach for implementing a fully connected network since a traditional fully connected network, where the features were board position-move pairs, would likely be far too sparse to learn anything meaningful given the size of our initial dataset.

Given the locality of pieces in certain regions of a chess board, the natural matrix configuration to represent chess boards, and the sequence nature of consecutive chess moves, we felt that a convolutional neural network could potentially offer a richer understanding of this problem. To this end, after implementing a fully connected version of the network, we then created a convolutional network, using Pytorch's convolutional functionality. We used a series of 3D filters fully connected in the number of channels where the first dimension of the filter corresponded to the number of moves assessed at a given time, the second dimension corresponded to the size of the height of the board being assessed with the filter, and the third dimension corresponded to the size of the width of the board being assessed with the filter. For the filter sizes we used two filters of size $(5 \times 3 \times 3)$ to assess (3×3) regions of the board five moves at a time, as we felt that sequences of several moves would be more illuminating in classifying humans vs AI. The region of (3×3) was selected since many of the interactions between chess pieces can be isolated to (3×3) regions. For example, pawns and knights can only affect pieces within a (3×3) region. We also implemented a stride of dimension $(2 \times 1 \times 1)$ so as to not have too much overlap between 5 move sequences while still capturing most of the relevant information of groupings of moves. For each convolutional layer, we increased the numbers of filters, starting with the initial 6 input channels and increasing to 64 and then 128 filters so that we could learn more complex features.

5 Experiments/Results/Discussion

The following metrics are given for algorithms run on the larger dataset (8k training, 1k validation, and 1k testing data for each of the 4 classes); from this dataset, the majority classifier accuracy would be expected to be 25%. For a baseline metric, we implemented a basic logistic regression algorithm, flattening each input (initially of dimension $(m \times 6 \times 40 \times 8 \times 8)$) and having 4 neurons in an output layer with no hidden layers, using the built-in Softmax activation in the Cross Entropy Loss function. Using log-scaled random hyper-parameter search across the learning rate and weight decay parameters for the Adam optimizer (learning rate: 10^{-3} to 10^{-5}, weight decay: 10^{-3} to 10^{-6}), the highest logistic regression validation accuracy was found to be 73.6% (Figure 3) with learning rate

of $8.9 * 10^{-4}$ and weight decay of $6.4 * 10^{-4}$. A mini-batch size of 64 examples was used for the logistic regression experiment and all other experiments described in this section.

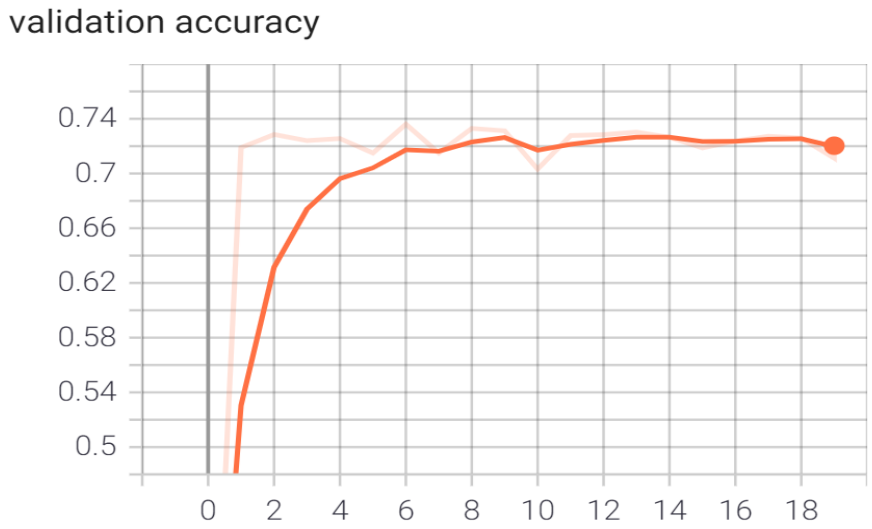

Figure 3: Validation accuracy for logistic regression across 20 epochs.

Next, we assessed multiple depths of fully-connected neural networks; we chose dense neural networks with one hidden layer (referred to as Dense1), three hidden layers (referred to as Dense3), and six hidden layers (referred to as Dense6). Dense1's hidden layer contained 512 neurons, Dense3's hidden layers contained 512, 512, and 64 neurons, in order, and Dense6's hidden layers contained 2048, 2048, 512, 512, 128, and 128 neurons, in order. Each dense neural network used a ReLU activation for the hidden layers and had an output layer of 4 neurons (with Softmax activation via the Cross Entropy Loss function). Although log-scale random hyper-parameter search was done for Dense1, the search was unable to be completed for the remainder of the experiments due to time constraints. Instead, the remaining experiments utilize a learning rate of $5 * 10^{-4}$ and a weight decay of $1 * 10^{-4}$, which were determined to be the best values by a systematic search at the beginning of the project. All three dense networks performed similarly, with Dense1, Dense3, and Dense6 achieving peak validation set accuracies of 76.8%, 76.8%, and 76.9%, respectively (Figure 4).

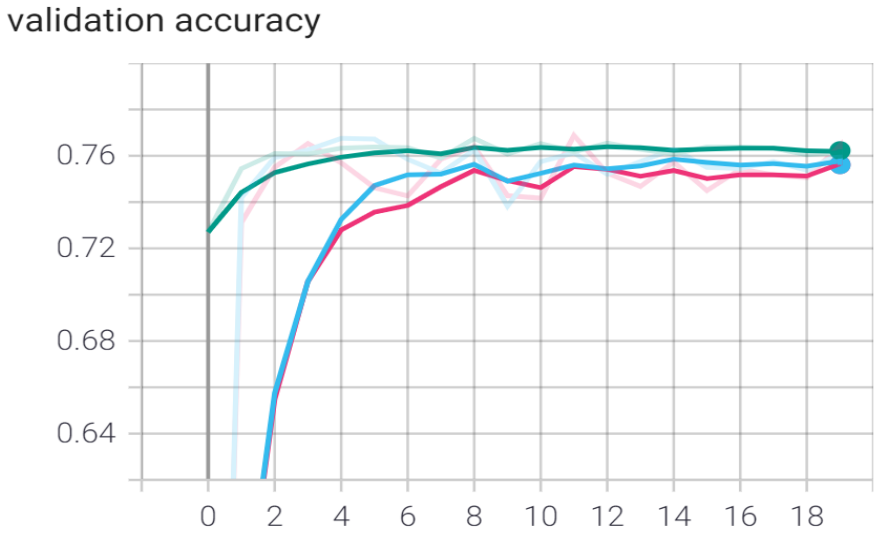

Figure 4: Validation accuracy for fully connected networks across 20 epochs. The green curve corresponds to Dense1; the blue curve corresponds to Dense3; and the pink curve corresponds to

Dense6.

To try to capture the regionality of chess moves and sequences of moves, experiments were conducted in adding convolutional layers prior to the hidden layers of Dense1, Dense3, and Dense6; these will be referred to as Conv1, Conv3, and Conv6, respectively. Two 3D convolutional layers were added: the first layer took as input the initial $(m \times 6 \times 40 \times 8 \times 8)$ dimensional data matrix and used 64 filters, each of size $(5 \times 3 \times 3)$ and with stride $(2 \times 1 \times 1)$, producing an output volume of dimensionality $(m \times 64 \times 18 \times 6 \times 6)$. The second 3D convolutional layer used 128 filters, each of size $(3 \times 3 \times 3)$ and with stride $(2 \times 1 \times 1)$, producing an output volume of dimensionality $(m \times 128 \times 8 \times 4 \times 4)$. This volume was flattened and then input into a fully-connected component of the appropriate depth. The addition of the 3D convolutional layers achieved slightly increased, nearly identical model performance, with Conv1, Conv3, and Conv6 achieving peak validation set accuracies of 77.5%, 77.8%, and 78.4%, respectively (Figure 5).

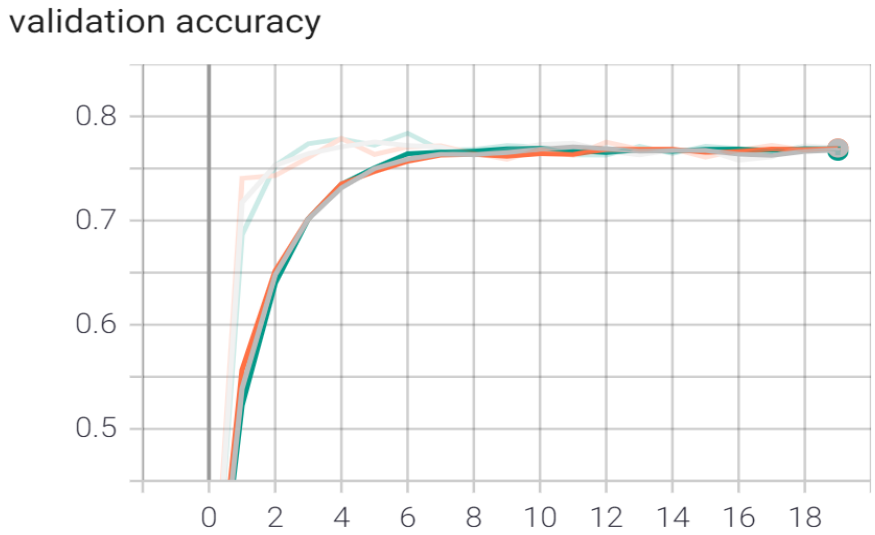

Figure 5: Validation accuracy for the convolutional networks across 20 epochs. The green curve corresponds to Conv1; the gray curve corresponds to Conv3; and the orange curve corresponds to Conv6.

Since the Conv6 architecture achieved the highest validation set accuracy during experimentation, it was chosen to be further investigated. Adding dropout layers following the activations of each fully-connected hidden layer (with keep probability of 0.8) did decrease variance between training and validation set accuracy, but significantly decreased the absolute value of both the training set accuracy and validation set accuracy after 20 epochs of training. Thus, Conv6 without dropout was assessed on the test set of 4,004 chess games (1,001 for each class). The Conv6 architecture achieved an overall accuracy of 79.3% on the test dataset, a pleasantly surprising increase over its peak validation set accuracy of 78.4%. The normalized confusion matrix of the classification by the Conv6 architecture is included below (Figure 6).

Predicted Label

True Label

	HH	HC	CH	CC
HH	0.19495126	0.01949513	0.01899525	0.01674581
HC	0.04048988	0.19345164	0.00474881	0.01149713
CH	0.03324169	0.00424894	0.1992002	0.01349663
CC	0.02274431	0.01024744	0.01074731	0.20644839

Figure 6: Confusion matrix where H stands for human and C stands for computer; the first letter represents the player with white pieces, i.e. HC indicates a game where the player with white pieces was a human and the player with black pieces was a computer. The classification accuracy is 77.9%

for HH, 77.3% for HC, 79.6% for CH, and 82.5% for CC.

Based on these results there seems to be promise in our convolutional neural network approach, given the substantial improvements on the majority classifier as well as the improvements from logistic regression to fully connected and from fully connected to convolutional. However, these improvements are relatively small and different architectures and hyper-parameters could be assessed to potentially increase these gains. Furthermore, our network exhibited significant variance between training set accuracy and validation set accuracy; for all of the experiments apart from the logistic regression, our networks achieved nearly 100% training accuracy while achieving <80% validation accuracy. Despite attempting various regularization techniques such as increasing L_2 regularization, implementing dropout, and increasing the amount of training data, the variance did not decrease significantly. Although increasing data improved our validation accuracy somewhat across experiments, the variance remained relatively high. Increasing L_2 regularization via the weight decay parameter of the Adam optimizer had little to no impact on variance, and even decreased validation set accuracy in some experiments with values $>10^{-4}$.

6 Conclusion/Future Work

All algorithms and model architectures assessed performed significantly better than the majority classifier accuracy ($\sim$70 - 80% for the experiments versus $\sim$25% for majority classifier accuracy for the balanced, larger dataset). Adding hidden layers provided a $\sim$4.4% increase in performance over naive logistic regression ($\sim$76.8% with hidden layers versus 73.6% without hidden layers); adding convolutional layers prior to fully-connected layers provided a $\sim$1.4% increase in performace ($\sim$77.9% with convolutional layers versus $\sim$76.8% without convolutional layers). The highest performing architecture included two 3D convolutional layers prior to 6 fully-connected hidden layers, ending with a 4-neuron output layer (see above for details of convolutional and fully-connected layers), achieving a peak validation set accuracy of 78.4% and a test set accuracy of 79.3%.

With more time and computing resources, there are several hyper-parameters that we would further optimize; these include further tuning of the learning rate and weight decay, the number of neurons in the various hidden layers, the size, stride, and number of channels for the convolutional layers, and the number of convolutional and fully-connected layers. Furthermore, with more computational power more training data can be used to tune model weights. We were able to collect a dataset of millions of games across all classes, but due to limited computational power and the need for quick iteration we were only able to use a small fraction of this data. Using a greater portion of the dataset could boost classifier accuracy and decrease variance between training and validation set accuracies.

7 Contributions

John focused a large portion of his work on tuning hyperparameters and working with tensor board to analyze results while Mark worked more closely with processing the data for the network and working on some of theory based on the challenges we confronted with our results. We implemented the actual network together, working through a Pytorch tutorial on creating a neural network.

References

[1] Silver, David, et al. "A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play." *Science*, vol. 362, no. 6419, 2018, pp. 1140–1144., doi:10.1126/science.aar6404.

[2] Sabatelli, Matthia, et al. "Learning to Evaluate Chess Positions with Deep Neural Networks and Limited Lookahead." *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods*, 2018, doi:10.5220/0006535502760283.

[3] McIlroy-Young, Reid, et al. "Aligning Superhuman AI with Human Behavior." *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, doi:10.1145/3394486.3403219.

[4] Oshri, Barak. "Predicting Moves in Chess using Convolutional Neural Networks." (2015).

[5] Zyga, Lisa. "Distinguishing between Humans and Computers in the Game of Go." *Phys.org*, Phys.org, 6 Nov. 2017, phys.org/news/2017-11-distinguishing-humans-game.html.

[6] Barnes, David J., and Julio Hernandez-Castro. "On the Limits of Engine Analysis for Cheating Detection in Chess." *Computers & Security*, vol. 48, 2015, pp. 58–73., doi:10.1016/j.cose.2014.10.002.

[7] Ludens@freechess.org. "FICS Games Database." *Search*, www.ficsgames.org/.

[8] Matteson, Andrew. "Analyzing Chess Positions with Python." *Medium*, Analytics Vidhya, 8 Feb. 2021, medium.com/analytics-vidhya/analyzing-chess-positions-with-python-26d73b7c892.