CSC2541-f18 Course Project Report
Human-Like Chess Engine

Reid McIlroy-Young
University of Toronto
reid.mcilroy.young@mail.utoronto.ca

Karthik Raja Kalaiselvi Bhaskar
University of Toronto
karthikraja.kalaiselvibhaskar@mail.utoronto.ca

Abstract

We used supervised training to create a series of chess engines based on humans play at different levels of skill. We compared them to other engines and to human players and found that self-play trained engines would sometimes behave more human-like than the supervised ones, although we believe this may be due to improper hyperparameter selection. The three methods we used for comparing to humans present a novel set of tools for evaluating human-like behaviour in complex reinforcement learning systems and hope to develop them further.

1 Introduction

Since the 80s computers have been working towards perfecting the game of chess, with the most notable even being the 1997 match where Kasparov was defeated by an IBM machine, *Deep Blue*[9]. The goal of IBM was to create system that could win chess games and to this end Deep Blue took some very inhuman actions. Most notably, at a critical juncture *Deep Blue* did not take the obvious line and instead took a much more delicate and refined path that requires much more modern chess engines to discover. The move was thus, both inhuman and unexpected, it was later discovered, to have been randomly selected[16]. This kind of decision making is only possible with algorithmic systems.

Modern chess engines have far exceeded human ability [15] in the last two decades, but despite humans learning from them there are significant difference between human and computer chess strategies. This is most evident when looking at end games, if there are seven pieces in a legal configuration the optimal moves for each player are known, these solutions are called endgame tablebases and are available commercially. Six pieces tablebases are available for free and are much smaller 100 TB vs 90 GB compressed, so those are what most researchers use. Again when humans are compared to computers their moves are different. Humans will be less likely to give up pieces, will make generalization mistakes where they assume a pattern holds more than they should and will play to their open having more opportunity to make a mistake than the optimal move[1].

The differences in human and computer learning of chess suggest the mechanism of learning is different. We are attempting to train a computer system to behave more like a human though supervised training with a general deep reinforcement learning system, the Leela Zero Chess project [4], an open source implementation of the well known *AlphaZero* chess engine created by *DeepMind*. These Human AI (haibrid) engines will hopefully allow a better understanding of how deep reinforcement learning systems can be trained and how humans learn allowing for better chess teaching aids.

1st Deep Reinforcement Learning, Final Project Report (CSC2541-F18 2018), Toronto, Canada.

A major concern that arises when once the engines have been trained is that of evaluating them. We develop three methods for comparing their behaviour to humans. First comparing their win rates to a variety of existing chess engines whose success against humans is known. Second to compare the probability distributions produced by the engines on a single board state to empirically derived probabilities distributions across many humans encountering the same board. Finally to compare the engines ability to predict each move a human makes while playing a single game.

1.1 Previous Works

This is not a well explored area. There has been some work done on 'move tables' as an addition to forward pruning technique that can act as a Transposition tables for moves not for position to make it play much more like a human [6]. The results have not been promising with the resulting system not performing well and only having behaviour for the early game.

Very recently there has been some success training a convolutional neural network to play *Candy Crush* with a tunable win rate [7]. This allows for better prediction of human behaviours for a given level. Sadly, *Candy Crush* is a much simpler game than chess so the working system does not have to do policy selection and can succeed without long term memory. But the work that has been done suggests that emulating human game playing is within the computational bounds of a contemporary RL system.

2 Method

2.1 Data

For this project we need games played between humans with average skill levels, instead of the collections of advanced player's games more commonly used [2]. To this end we used a subset of the 432,335,939 games on Lichess [3] (on November 2018). Lichess maintains a player ELO rating system which allowed us to extract games played between evenly matched ELO players. We considered players evenly matched if they had the same ELO when rounded up to the nearest hundred. The counts for each bin are shown in figure 1. During training games were randomly sampled from a selected bin, thus giving an engine that has only been trained on one ELO range of players. We also set aside the 22,971,939 games from September of 2018 as a holdout set.

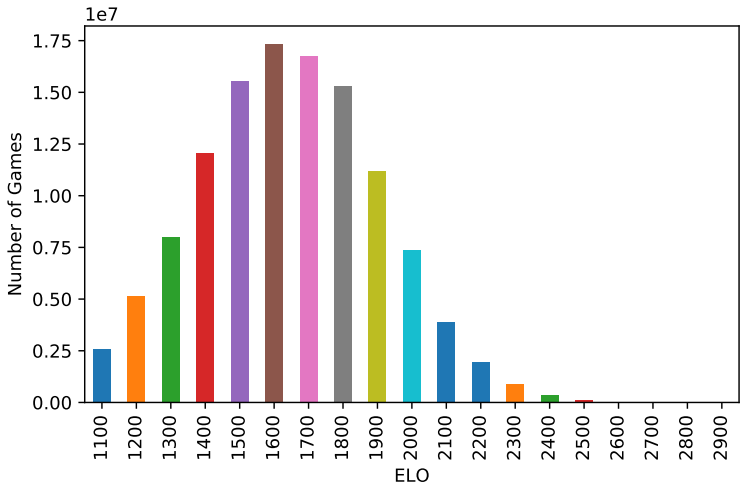

Figure 1: Distribution of games between similar ELO players

2.2 ELO

The main system used for evaluating chess players and evaluating chess engines ELO [5] is much less useful than one might expect. Since ELO relies on the win rate of a single individual against other ranked individuals within some time period and does not refer to any external factors [5] it is excellent at rating players within a well connected player community. But when attempting to compare chess players or engines that are not already ranked or capable of directly playing each other the ELO does not work, this is before considering that engine performance varies greatly depending on the computing resources provided. Thus when a group presents an ELO rating for their engine, e.g. [14], their ELO is only well defined within their testing system. Therefore is more useful to view ELO as a simple ranking and to ignore the values when comparing between groups.

We have two separate ELO rankings used throughout our paper. First, there is the ELO used by Lichess, these numbers are derived based on human players playing ranked events and are accurate for most individuals. Second the ELO ranking used by Leela chess to compare different generations of their engines, these are less accurate and prone to inflation, since every new engines must beat the previous ones.

We did attempt to create our own ELO ranking system for comparing our own engines but the overhead in getting well defined ELOs for the benchmarking is high and simply ranking a series of engines provides nearly the same information. That said, measuring our own ELO would be considered in future work.

2.3 AlphaZero Chess

AlphaZero chess engine created by *DeepMind* is the current best reinforcement learning architecture for chess (amongst other games)[14]. It relies on a zero human input training system of iterating self-play and backpropagation perfected originally for playing Go [13][15]. When playing the engine will employ a Monte-Carlo Tree Search [10] variant, Predictor + Upper Confidence Bound tree search (PUCT) [11]. This employs a policy-value network with the values being produced from a single forked deep neural network.

The policy network outputs a probability distribution across all legal moves for a single board state, giving for each the likelihood it is the correct move. How the best move is defined here though is just as the most likely to be chosen by an unthinking actor, i.e. by a player in the training data. The value network alternatively gives the likelihood of a board leading to a win, thus a single number for each board. The value head learns by comparing the board states given in training to the final results of the match. Thus although the policy network and value network have the similar outputs, the same residual layers and the same training data, their outputs can differ significantly.

When playing the engine does a series of PUCT rollouts where it assigns a Q (estimated value), N (number of times this node has been traversed) and U (Upper Confidence Bound, exploration reward initialized with policy network) to each of the possible following moves (nodes). Then selects the one that maximizes$U + Q$ and evaluates that node in the same manner. If the rollout has not reached a terminal state within some number of moves it halts and used the value network for the value of the nodes, which is then added to all the Q values up the tree (minus future discount factor). By doing thousands of rollouts per game move the engine can far exceed the raw outputs from examining a single board state.

2.4 Leela Chess

The *AlphaZero* chess engine created by *DeepMind* is closed source and does not have an version available to use [14]. There is though an free and open source implementation of the zero training and playing system, *Leela Chess Zero* [4]. As the name suggests *Leela Chess Zero* is trained via self play and besides tweaks to the hyper parameters and residual layers is a *clean room* reimplementation of the *AlphaZero* system. For our work we replaced the self play with human games, thus going back from unsupervised to a supervised domain.

We collected a few different generation of Leela weights, each with a different ELO (as measure against the other Leela engines). These Leelas along with Stockfish, a well regarded non-machine learning based chess engine [17], form our testing suite. The Leela engines are all referred to in this report as *LeelaEngine t3-* then a number, the number is their ELO. The Stockfish engines have

three numbers associated with them, the second marked with a *d* is the search depth and is the most significant factor in their performance.

2.5 Training

For training we selected five ELO bins of games (1100, 1400, 1500, 1600, 2000, 2100) and used the same procedure on each, see table 1 for the parameters used. The parameters selected was based on both suggestions in the Leela chess documentation and a preliminary training run with $464,408$ randomly selected games (a cleaning script was run on games from 2014 and 2015 until it crashed). The 64×6 residual layers we are using matches the first successful Leela weights, although deeper networks are noew being used. The supervised engines will be referred to as *HaibridEngine* some number *-64x6-140000* the number is the ELO bin the weights where trained with. The performance of the testing and final networks will be discussed later.

The training was done on an Microsoft Azure GPU compute virtual machine using two *NVIDIA Tesla K80* and each network took approximately 3 days to completely train. During training the prediction error rate and mean squared error of the policy head were computed against the holdout data. The prediction error rate for the holdout data would quickly converge to 30% after a few hours of training, while the MSE would also converge to near .33. Notably the training data error rate would be only slightly lower while the MSE would significantly decrease. These very low accuracy results are normal for Leela chess training and due to the complexity of chess playing along with the lack of a tree search means they are only useful for seeing gross errors in training.

ELO	1100	1400	1600	2000	2100
Number of games	$2,527,950$	$11,375,345$	$17,341,991$	$7,353,857$	$3,908,191$
Residual Blocks		6			
Filters		64			
Batch Size		$8,192$			
Total Steps		$8,192$			
Testing Holdout		10%			
Learning rates		0.02 start			
		0.02 after $100,000$ steps			
		0.002 after $130,000$ steps			

Table 1: Summary of training parameters

3 Results

Since our goal is to create more human like play, not optimal play, evaluating performance is difficult so we tried many approaches. Additionally, each of the results took at least one day to generate and some took much longer. This great time usage is partially due to the evaluations being run on CPU only system, although it has 40 cores running in parallel to do the work. *HaibridEngine* (Human AI) are the five we trained their ELOs are the first number. *LeelaEngine* are the Leelas, with the first number being their ELO. Both HaibridEngine and Leela engines were given the same constraints: 10 seconds per move or $10,000$ rollouts per move

StockfishEngine are the Stockfish with the second number being their search depth. We also have a random agent that selects a random legal move each turn [1].

We also created a Lichess bot (found at lichess.org/@/haibrid_bot) running with various generations of haibrid networks but were not able to reliably get more than a few human opponents a day to play so we were unable to use real human games in our analysis.

[1]The random agent can be interesting to watch play, here is game of it playing itself, it also appears that random behaviour causes the Stockfish analysis to struggle

3.1 Engine Win Rates

The first thing to consider with our newly trained networks is how well do they performer against other Leela weights and against Stockfish. Figure 2 shows the win rate of every engine against every other engine across 540 games for each combination. The engines have been sorted by class and the ranked by skill within their class. Also their has been a small amount of noise added (with the `--noise` option) to the neural engines as a source of entropy. This shows that the Leela chess engines' ELO ranking are a good ranking (although two very new $4000+$ were removed that did not obey the trend, likely due to noise in comparing high level engines). It is also interesting that after a depth of 5 Stockfish is nearly unbeatable, but this may be due to the computational efficiency it has over the neural engines as it is an order of magnitude or more faster at examining a position.

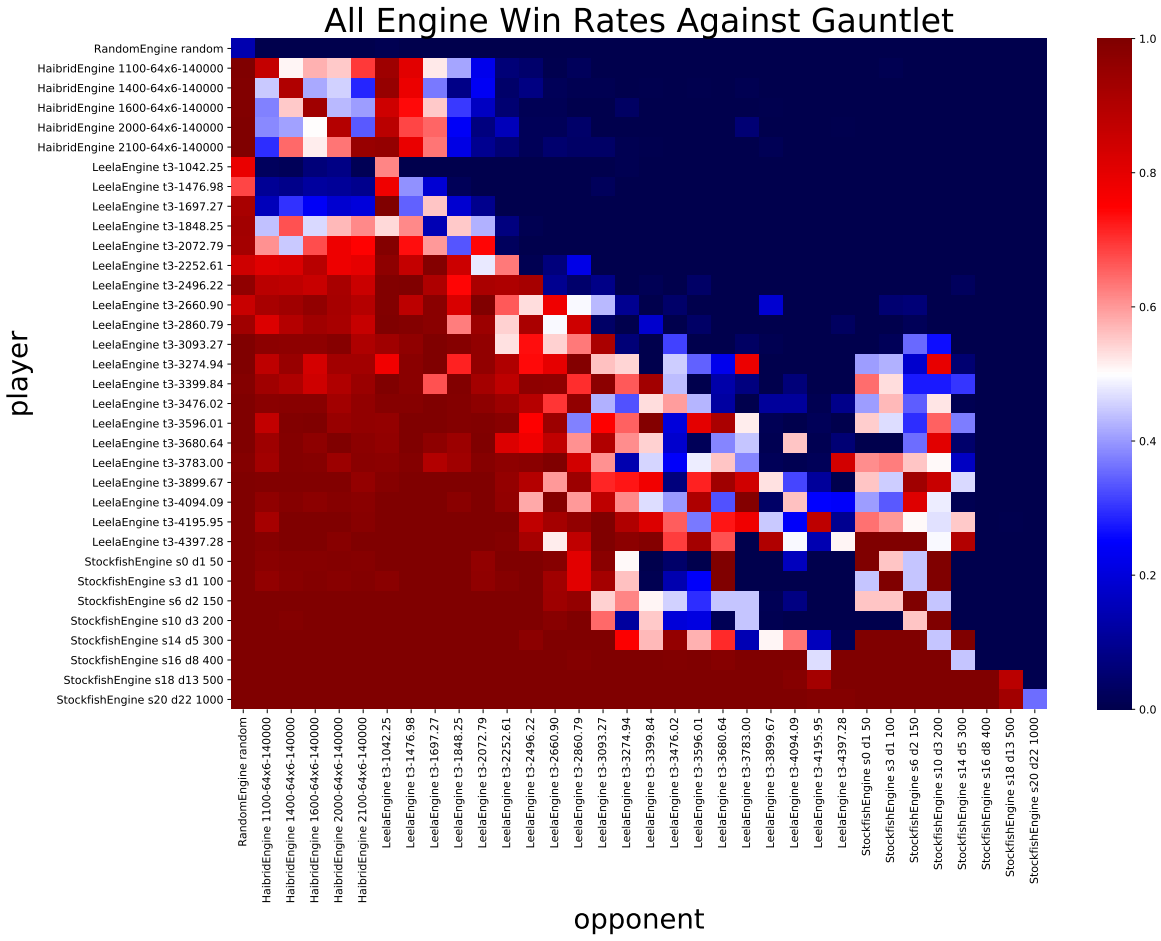

Figure 2: Win rates of each engine vs each other engine in 540 games

As we are primarily concerned with the haibrid engines those were run through a second gauntlet of 2355 per match up. Figure 3 shows the win rates for the haibrids, while Figure 4 shows the rate of games tying. These results are disappointing on two fronts. First the haibrids all performer worse than our earlier testing version. Second, they do not have a strong ranking, which suggests the input game's ELO is not significant.

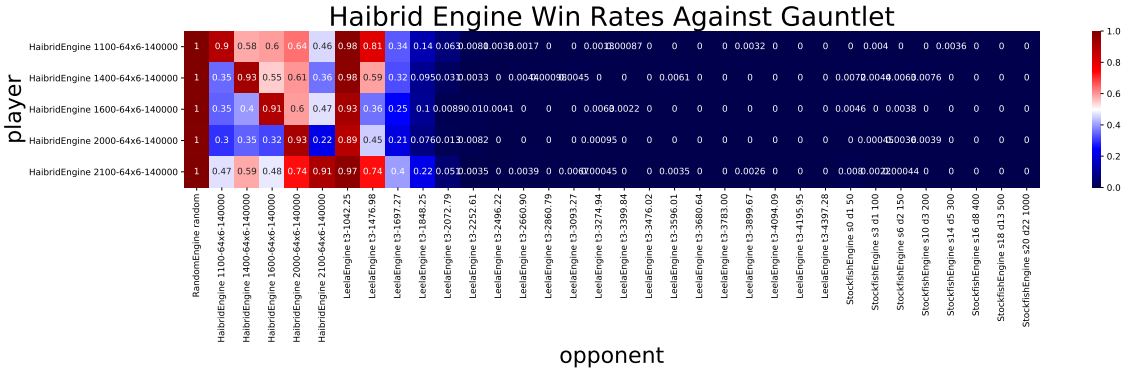

Figure 3: Win rates of each engine haibrid engine vs all other engine in 2355 games, each cell has the actual value written in it

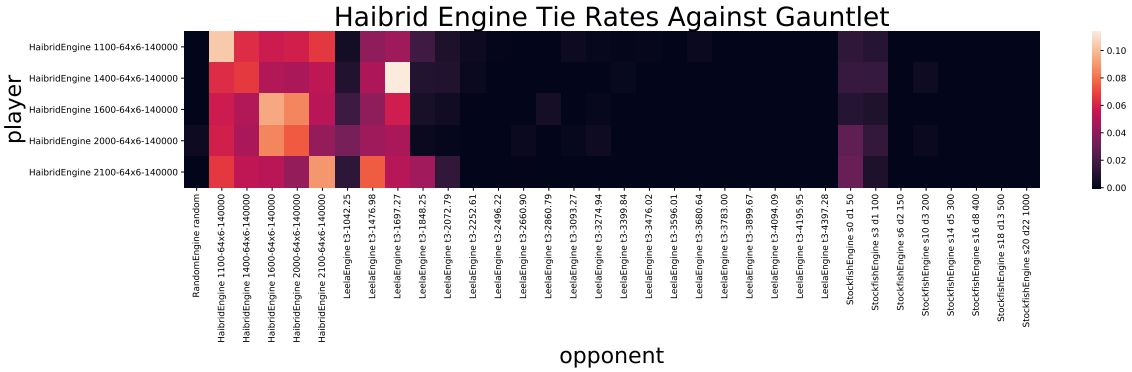

Figure 4: Tie rates of each engine haibrid engine vs all other engine in 2355 games

3.2 KL Divergence of Board States

Since we have a huge number of human games it is possible to find a good number of board states that many humans have traversed. We collected all board states encountered by 50 or more humans, giving $115,227$ examples. Then created a probability distribution over all move they made $P_{Human}(board)$, by simply normalizing the counts of their moves. Then for each of the haibrid engines and two Leelas we gave them each board state and took the outputs of the policy network $P_{Policy}(board)$ and computed the KL divergence for the policy and human probabilities for each of the boards in our sample. Table 2 has the summary statistics.

Again the results of this analysis are disappointing, the Leela engines perform significantly better than the haibrids. But the haibrids have a much higher variance and as a histogram of the KL divergences for an engine is created, Figure 5 shows two representative samples. It becomes clear that the haibrid engine is much higher variance, not necessarily that different of a distribution.

We did the same procedure for the outputs after 10,000 rollouts on a board state, i.e. the Q and N values, and arrived and similar results. Note that interpreting Q or N values as probabilities is not very well defined and the policy network is explicitly attempting to predict the selected move. Thus we believe that the policy outputs are the most significant and did not include the other results due to space constraints.

$D_{KL}\left(P_{Policy}\|\|P_{Human}\right)$	mean	median	variance
Haibrid			
1100	1.808995	1.607913	1.459266
1400	1.614821	1.380003	1.287912
1600	1.674275	1.450223	1.336098
2000	1.609266	1.372704	1.305895
2100	1.374900	1.160331	0.998643
Leela			
t3-1697.27	1.017941	0.958877	0.225320
t3-2072.79	0.991182	0.914810	0.247753

Table 2: Summary statistics of the KL divergence per board state between humans and the neural engines

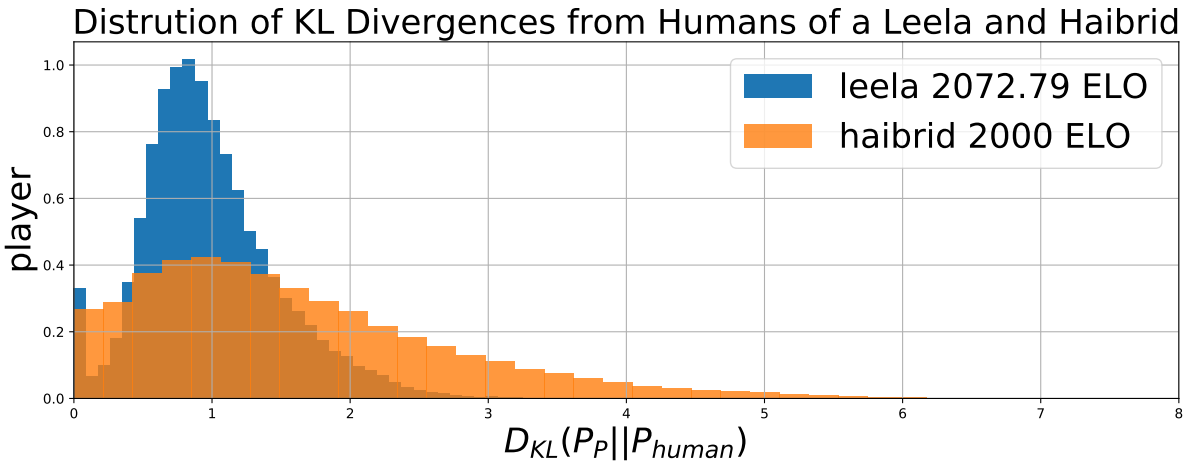

Figure 5: Tie rates of each engine haibrid engine vs all other engine in 2355 games

3.3 Game Trajectory Accuracy

The final way we compared the haibrids to humans was by sampling 500 games by players from every bin of 100 ELO between 1000 and 2100 and from games with more than 15 moves, i.e. 11 samples of 500 games each. For each engine we gave it the games from each sample and asked ti to play as black. The instead of making the moves it suggested we took the move the human made and recorded if it was the same as the engine. Then we count the hits and misses and record the hit rate. The summary statistics of each engines' hit rate are shown in Table 3 for all games. When looked at by the ELO rating individually the results are similar to the table so they are not included do to space constraints.

There are few interesting results results. First the Leela outperforms Stockfish even at low ELOs. This suggests that the neural architecture is better suited for matching humans. This shows us that matching human play is not the same as finding perfect play since Stockfish can easily beat most of those Leelas at the computational levels used here. Second the haibrids perform very similarly to low ELO Leelas, but we know from the KL divergence measurements their predictions are much higher variance. We are unsure how to interpret this and will be studying it closer in future work.

Engine	mean hit rate	median hit rate	95th percentile	variance
random	0.064036	0.056156	0.162400	0.002662
hiabrid_1100	0.269881	0.266667	0.416574	0.007385
hiabrid_1400	0.263925	0.261394	0.410767	0.007048
hiabrid_1600	0.269598	0.262924	0.414941	0.007080
hiabrid_2000	0.269605	0.266667	0.416705	0.007423
hiabrid_2100	0.281941	0.278889	0.436141	0.007631
leela_1042.25	0.146847	0.140634	0.266460	0.004929
leela_1476.98	0.200263	0.197392	0.333333	0.006972
leela_1697.27	0.219106	0.219087	0.354839	0.007099
leela_1848.25	0.264179	0.262273	0.395472	0.006945
leela_2072.79	0.288041	0.286161	0.419999	0.006723
leela_2252.61	0.323486	0.327381	0.461628	0.007332
leela_2496.22	0.347539	0.348184	0.487796	0.007568
leela_2660.90	0.358109	0.359186	0.500167	0.007688
leela_2860.79	0.372131	0.370257	0.511642	0.007513
leela_3093.27	0.394458	0.394015	0.545651	0.008590
leela_3274.94	0.414042	0.415838	0.560550	0.008514
leela_3399.84	0.428330	0.430237	0.578600	0.008881
leela_3476.02	0.444666	0.441353	0.595648	0.008843
leela_3596.01	0.438798	0.435948	0.587240	0.008755
leela_3680.64	0.442014	0.442981	0.600000	0.008927
leela_3783.00	0.441862	0.442677	0.595497	0.009136
leela_3899.67	0.435884	0.435417	0.588991	0.008976
leela_4094.09	0.433680	0.436508	0.587634	0.009239
leela_4195.95	0.440221	0.441520	0.594469	0.009018
leela_4397.28	0.433032	0.432749	0.591717	0.009206
stockfish_0s_50m_1d	0.365618	0.365728	0.512626	0.008092
stockfish_3s_100m_1d	0.365618	0.365728	0.512626	0.008092
stockfish_6s_150m_2d	0.351685	0.351512	0.500000	0.008403
stockfish_10s_200m_3d	0.339002	0.342928	0.482583	0.008314
stockfish_14s_300m_5d	0.338813	0.344130	0.490847	0.008821
stockfish_16s_400m_8d	0.371510	0.367896	0.531787	0.009032
stockfish_18s_500m_13d	0.381152	0.384841	0.537362	0.009774
stockfish_20s_1000m_22d	0.387298	0.391333	0.547866	0.010298

Table 3: Summary statistics of the KL divergence per board state between humans and the neural engines

4 Conclusion

Generally the haibrids performed quite poorly as chess engines and as predictors of human behaviour. This lead us to examine their training procedures and compare it to the AlphaZero [15] training parameters. Our batch size was twice theirs and four time higher than Leela's standard [4], also we had many more training examples than either would us in a single session. Too large batch size can leads to significant decreases in training performance [12] and we recently started training some haibrids with a lower batch size that are already exceeding any of the one examined here.

The methods for analysing human like play will explore will need further work to validate, but the trajectory analysis in particular seems like it could be a good tool as it is not strongly correlated with win rate. The KL divergences likely need some further tuning as if the humans only select two move out of a possible 10 for a board state the neural engines will consider all possible moves which can lead to the KL divergences discarding much of the information contained in the engine's probability distribution.

We plan to continue work in this area along a few different paths.By modifying the training loss calculations and values to further encourage human like behaviour over raw game wins. Or by changing the neural network's design and policy selection, using an GAN [8] would be the first step.

References

[1] Ashton Anderson, Jon Kleinberg, and Sendhil Mullainathan. Assessing human error against a benchmark of perfection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(4):45, 2017.

[2] Omid E David, Nathan S Netanyahu, and Lior Wolf. Deepchess: End-to-end deep neural network for automatic learning in chess. In *International Conference on Artificial Neural Networks*, pages 88–96. Springer, 2016.

[3] Thibault Duplessis. Lichess, 2018.

[4] Gary Linscott Gian-Carlo Pascutto. Leela chess zero, 2018.

[5] Mark E Glickman. The glicko system. *Boston University*, 1995.

[6] Kieran Greer. Dynamic move chains and move tables in computer chess. *CoRR*, abs/1503.04333, 2015.

[7] Stefan Freyr Gudmundsson, Philipp Eisen, Erik Poromaa, Alex Nodet, Sami Purmonen, Bartlomiej Kozakowski, Richard Meurling, and Lele Cao. Human-like playtesting with deep learning. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

[8] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[9] Garry Kasparov. The chess master and the computer. *The New York Review of Books*, 57(2):16–19, 2010.

[10] Rémi Munos et al. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends® in Machine Learning*, 7(1):1–129, 2014.

[11] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[12] Jared Kaplan Sam McCandlish and Dario Amodei. An empirical model of large-batch training. *openai preprint*, 20187.

[13] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[14] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[16] Nate Silver. *The signal and the noise: why so many predictions fail–but some don't*. Penguin, 2012.

[17] Joona Kiiski et al. Tord Romstad, Marco Costalba. Stockfish: A strong open source chess engine, 2018.