Progress Report

Project Mittens

Kate Brown, Dan Hayes, James Parkington

We're in the middle of a massive surge in popularity surrounding the game of chess. What once was considered a niche hobby relegated to clubs of intellectuals has now becomes a prominent thread in the social fabric. This sudden rise in interest, no doubt magnified by the number of us at home, can be attributed to several factors, including:

- the increase in options for accessible chess-playing platforms
- the emergence of popular chess streamers garnering millions of views alongside the country's top gamers
- the portrayal of chess in mainstream television, particularly *The Queen's Gambit*

With this increase in attention comes a significant increase in players of all backgrounds. The added attention to the game has been amplified by the rapid development of advanced chess bots, evolving the game into a playing ground for machine intelligence, and thereby inviting an entirely new form of competition for eager new users.

This fever pitch for bot play broke the sound barrier with the emergence and sudden decommissioning of **Mittens** in January of 2022. Mittens was, at the time, the most competitive bot available for players to challenge on **chess.com.** Like every player on **chess.com** (*and the other platforms in the space*), each bot is given a rating that informs you of its general strength as a player. These ratings are generally based on Elo, a scoring algorithm devised by Arpad Elo in the 1960s. Players who are adept at finding winning combinations tend to have higher Elo values approaching and sometimes surpassing a value of **2,000**.

However, **chess.com** upon releasing Mittens bestowed it with an Elo of **1**! About **100** times worse than the poorest chess players on its platform. Naturally, this sparked curiosity and attention, followed by thousands of Twitch streams and YouTube videos of grandmasters playing Mittens and attempting to guess its Elo, even pitting Mittens against other well-known bots like Stockfish and Leela Chess Zero to get a stronger sense of its abilities.

This raised some major questions for the **3** of us chess fans of different backgrounds:

- How could someone develop a sense of a competitor's Elo without a value for reference?
- Can bots actually make human chess-players more competitive as a whole?
- What's the natural growth of a chess player improving via learning, rather than relying on bots? Can we tell when a player's Elo growth is influenced by working with chess engines?
- How many moves can a GM "know" at any given moment? How does that compare to the best engines?
- And most importantly, did Arpad Elo have any idea what was in store for the future when he devised this algorithm?

Hypotheses

Our goal is to substantiate these **4** premises with available math and research to conclude that **bots are responsible for a measurable growth in the average chess player's ability to play chess to such a degree that it is tempting for top players to not only study with them, but to cheat with them**.

1. Elo as a mathematical construct is **positioned to scale** with the introduction of a new "population" of players as bots, which can compute depths of move sequences too large for humans to comprehend.
2. Experienced players have an **intrinsic understanding of Elo** and its variations from platform to platform, and therefore can make educated guesses about a competitor's strength based on their opening choices, demonstration of known winning variations, and overall ability to generate creative patterns of move sequences.
3. The algorithms chess bots use to play at previously unreached levels of precision and accuracy have become **increasingly more efficient** than humans at processing move sequences over time, resulting in their accessibility for a wider audience.
4. Humans have shifted their perception of "good" chess and their ability to recreate it in large part to the increased accessibility of bots, positively impacting their skill levels, both naturally and artificially (*via cheating*).

Move Sequences

The major reason chess has remained a compelling game to study is because of its unassumingly high complexity. The number of possible chess games is a well-known and frequently cited statistic in the field of chess and computer science. It is often referred to as the **Shannon number** named after the mathematician Claude Shannon who first calculated a figure of 10^{120} in the 1950s[1]. He went on to claim that the number of possible move sequences on a chess board far exceeds the number of atoms in the observable universe.

Estimating the Number of Possible Games

For a rough estimate of the total number of possible chess games, we can use the branching factor formula, $p(d) = b^d$, to calculate the total number of possible games, where b is the branching factor, and d is the average length of a game in ply (or half-turns). You can understand d as the **depth** of known moves at any given point, while b can be thought of as the average number of legal moves available at each point in the game. When advanced players calculate their next moves, they typically review the most competitive move sequences available out to a depth of **8** to **10** ply to determine if a sequence may be winning.

In chess, the branching factor is around **35** and the average length of a game is around **80** ply[2]. We will later verify that figure for the branching factor by using known depths as base cases for a recurrence relation. For now, we can estimate the total number of possible games as follows:

$$\begin{aligned} p(d) &= 35^{80} \\ &= 3.353 \cdot 10^{123} \end{aligned}$$

In other words, this value is **3,353** followed by **120 zeroes**! This estimation assumes that all legal moves are equally likely at each point in the game, which is not entirely accurate since some moves are more strategically advantageous than others. However, it still provides a useful approximation of the number of possible chess games.

It's worth nothing that the theoretical endpoint to d has also been previously explored and iterated upon. Fabel, Bonsdorff, and Riihimaa propose that the maximum length of a chess game is **5,898** plies under the established **fifty-move rule** and the **threefold repetition rule**, both of which are designed to prevent endless games[3].

This theoretical maximum takes into account all possible move sequences while adhering to the rules of the game, including stalemate, checkmate, and draw by insufficient material. However, it's important to note that this maximum length is purely theoretical and doesn't account for the strategic choices that players make during a real game.

Comparing Chess with Other Games

Here's how chess compares in raw complexity to some other games that use pieces with limited mobility:

---

[1] Claude Shannon (1950). "Programming a Computer for Playing Chess". p.4. *Philosophical Magazine*. **41** (314). Archived from the original on 2020-05-23.

[2] Victor Allis (1994). *Searching for Solutions in Games and Artificial Intelligence*. p. 171. Ph.D. Thesis, University of Limburg, Maastricht, The Netherlands. ISBN 978-90-900748-8-7.

[3] Fabel, Bonsdorff, Riihimaa, (1974). Ajedrez y Matemáticas. p. 13 - 17. Editorial Martínez Roca.

- **Tic-tac-toe**: The branching factor varies depending on the position, but in the early game it is around **3**. Assuming an average game length of **9** moves, $p(d) \approx 3^9 = 19{,}683$.
- **Connect Four**: The branching factor is **7** in the early game. Assuming an average game length of **30** moves, $p(d) \approx 7^{30} = 5.5 \cdot 10^{24}$.
- **Checkers**: Checkers is another popular board game with limited piece mobility. The branching factor is around **5 - 10** in the early game, but decreases as pieces are removed and legal moves disappear. Assuming an average game length of **50** moves, $p(d) \approx 8^{50} = 8.9 \cdot 10^{37}$.
- **Scrabble**: Assuming a standard Scrabble game played with **100** tiles and an average game length of **25** moves, we can make an estimation based on the average number if playable tiles per turn, which is **7** for most of the game.

  Without considering the need to make legal words, $p(d) \approx \binom{100}{7} \cdot 7!^{25} = 5.82 \cdot 10^{75}$

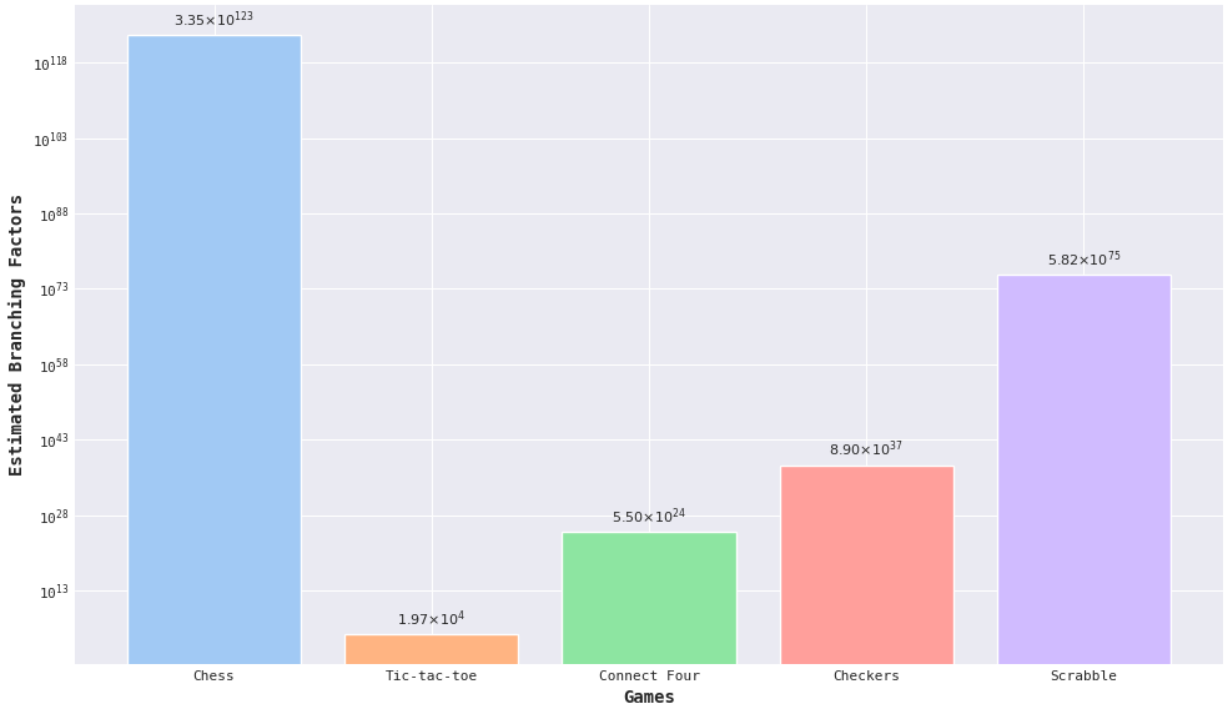

Figure 1: Estimated Branching Factors

Regarding the pursuit of a *winning condition* for each of these games, it must be noted that the branching factor doesn't necessarily correlate to algorithmic complexity.

For example, some games might have a relatively small branching factor but require more sophisticated heuristics to evaluate the strength of different moves, while others might have a larger branching factor but can be more easily pruned or optimized.

However, in general, games with higher game tree complexity and branching factors, like chess, are more computationally difficult than games with lower complexity, like checkers and tic-tac-toe, since evaluating the strength of different moves and find the optimal strategy requires consideration of more move sequences. This difficulty is precisely why chess has become a standard for evaluating the performance of artificial intelligence algorithms, particularly in the area of game-playing agents.

Analyzing Perft Results to Calculate Branching Factors

Earlier, we noted that both Shannon and Allis cite **35** as the average branching factor for a chess game of **80** ply. However, branching factor generally increases as the game progresses, since a moved piece inherently allows the other pieces behind it to also move more freely.

As such, we can back into a branching factor of $b(d)$ using the values of $p(d)$ to compute the ratio of the number of possible games at successive game lengths. Because the branching tree for chess increases one ply at a time, the relationship can be understood as:

$$b(d) = \frac{p(d)}{p(d-1)}$$

From here, we can use verified **Perft** (*performance test, move path enumeration*) results[4] to expand our previous analysis on the growth of the sequence derived from $b(d)$. Using the first **15** ply from analyses by Edwards and Banerjee, we receive the following branching factors:

$b(1) = p(1)/p(0)$	=	20	/	$p(0)$	=	20
$b(2) = p(2)/p(1)$	=	400	/	$p(1)$	=	20
$b(3) = p(3)/p(2)$	=	8,902	/	$p(2)$	=	22.255
$b(4) = p(4)/p(3)$	=	197,281	/	$p(3)$	=	22.161
$b(5) = p(5)/p(4)$	=	4,865,609	/	$p(4)$	=	24.663
$b(6) = p(6)/p(5)$	=	119,060,324	/	$p(5)$	=	24.470
$b(7) = p(7)/p(6)$	=	3,195,901,860	/	$p(6)$	=	26.843
$b(8) = p(8)/p(7)$	=	84,998,978,956	/	$p(7)$	=	26.596
$b(9) = p(9)/p(8)$	=	2,439,530,234,167	/	$p(8)$	=	28.701
$b(10) = p(10)/p(9)$	=	69,352,859,712,417	/	$p(9)$	=	28.429
$b(11) = p(11)/p(10)$	=	2,097,651,003,696,806	/	$p(10)$	=	30.246
$b(12) = p(12)/p(11)$	=	62,854,969,236,701,747	/	$p(11)$	=	29.964
$b(13) = p(13)/p(12)$	=	1,981,066,775,000,396,239	/	$p(12)$	=	31.518
$b(14) = p(14)/p(13)$	=	61,885,021,521,585,529,237	/	$p(13)$	=	31.238
$b(15) = p(15)/p(14)$	=	2,015,099,950,053,364,471,960	/	$p(14)$	=	32.562
$\cdots$						
$b(80) = p(80)/p(80-1)$	=	$3.353 \cdot 10^{123}$	/	$p(80-1)$	$\sim$	35

In other words, we know that after each player has moved a piece **4** times, there are **84,998,978,956** possible games that could have been played.

[4] Chess Programming Wiki. (last modified [29 August 2022]). "Perft Results." chessprogramming.org/Perft_Results.

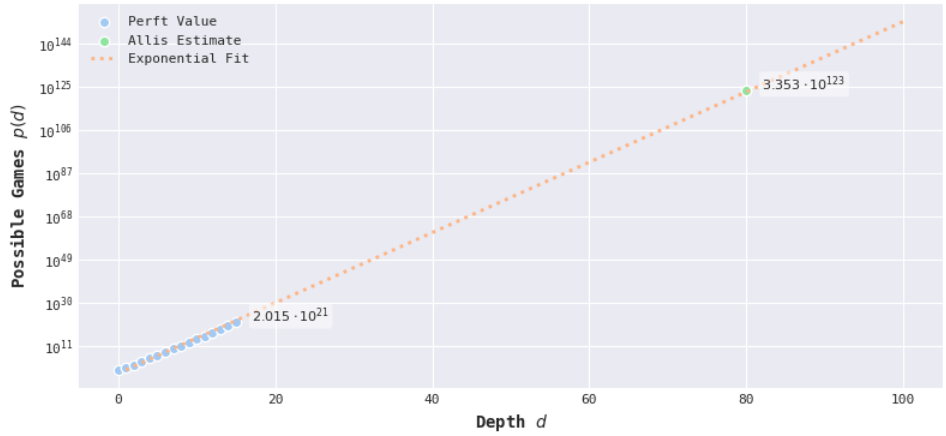

Figure 2 uses the following **exponential regression** to fit values of $P(d)$:

$$p(d) = e^{m \cdot d + b}$$

The fit line has the following slope (m) and y-intercept (b) values:

$$\begin{aligned} m &= 3.5733 \\ b &= -2.8118 \end{aligned}$$

Figure 2: $p(d)$ vs. d

To further quantify the growth of the number of possible games, we can analyze the sequence of branching factors using **big-O notation**, which represents the asymptotic upper bound of a function.

Given $b_{>1} \in \mathbb{R}^+$, and that the number of possible games grows exponentially with d, we have:

$$p(d) = \mathcal{O}(b^d)$$

This upper-bounded exponential growth is not unique to chess and can be found in other complex systems, such as the growth of decision trees in artificial intelligence, the development of protein folding patterns, and even the expansion of certain natural phenomena like the growth of crystals.

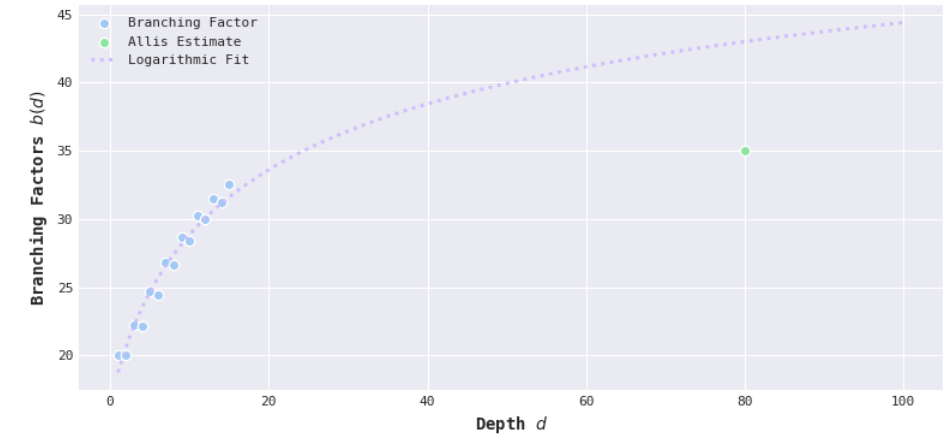

Figure 3 uses the following **logarithmic regression** to fit values of $b(d)$:

$$b(d) = W \log^X(d - Y) + Z$$

W, X, Y, & Z are constants:

- W: Scaling coefficient
- X: Curve-shaping exponent
- Y: Translation constant
- Z: Offset constant

$$\begin{aligned} W &= 21484.6793 \\ X &= 0.0014 \\ Y &= -6.4722 \\ Z &= -21487.1988 \end{aligned}$$

Figure 3: $b(d)$ vs. d

This exponential growth has implications for exceptionally strong players and algorithms alike, as they must efficiently traverse and evaluate vast, quickly growing game trees. Understanding the time complexity of chess not only allows us to appreciate its intricacy but also helps inform the development of more advanced heuristics and artificial intelligence techniques to tackle such a complex system.

Measuring the Growth of Complexity

Note that in spite of the ever-increasing number of move sequences per ply, the rate of increase for $b(d)$ seems to slows down as d increases, especially as it approaches the Allis estimate at $b(80)$. To express this mathematically and to analyze the growth rate of $b(d)$ over increases in d, we can define a separate function, $g(d)$:

$$g(d) = \frac{\log(b(d))}{\log(d)}$$

A higher value for this logarithmic ratio indicates a more rapid growth rate, while a lower value suggests slower growth.

By using the values from the $b(d)$ series, we can calculate $g(d)$ for the same ply depths to better understand and compare the growth behavior of the tree structure over time. Note that $g(1)$ is undefined, because $\log(1) = 0$ for any base, and we cannot divide by zero:

$$
\begin{aligned}
g(1) &= \log(b(1))/\log(1) &&= && 2.996 && / && 0 && = && \varnothing \\
g(2) &= \log(b(2))/\log(2) &&= && 2.996 && / && 0.693 && \approx && 4.322 \\
g(3) &= \log(b(3))/\log(3) &&\approx && 3.103 && / && 1.099 && \approx && 2.824 \\
g(4) &= \log(b(4))/\log(4) &&\approx && 3.098 && / && 1.386 && \approx && 2.235 \\
g(5) &= \log(b(5))/\log(5) &&\approx && 3.205 && / && 1.609 && \approx && 1.992 \\
g(6) &= \log(b(6))/\log(6) &&\approx && 3.197 && / && 1.792 && \approx && 1.785 \\
g(7) &= \log(b(7))/\log(7) &&\approx && 3.290 && / && 1.946 && \approx && 1.691 \\
g(8) &= \log(b(8))/\log(8) &&\approx && 3.281 && / && 2.079 && \approx && 1.578 \\
g(9) &= \log(b(9))/\log(9) &&\approx && 3.357 && / && 2.197 && \approx && 1.528 \\
g(10) &= \log(b(10))/\log(10) &&\approx && 3.347 && / && 2.302 && \approx && 1.454 \\
g(11) &= \log(b(11))/\log(11) &&\approx && 3.409 && / && 2.397 && \approx && 1.422 \\
g(12) &= \log(b(12))/\log(12) &&\approx && 3.400 && / && 2.484 && \approx && 1.368 \\
g(13) &= \log(b(13))/\log(13) &&\approx && 3.451 && / && 2.564 && \approx && 1.345 \\
g(14) &= \log(b(14))/\log(14) &&\approx && 3.442 && / && 2.639 && \approx && 1.304 \\
g(15) &= \log(b(15))/\log(15) &&\approx && 3.483 && / && 2.709 && \approx && 1.286 \\
&\cdots \\
g(80) &= \log(b(80))/\log(80) &&\approx && 3.555 && / && 4.382 && \approx && 0.811
\end{aligned}
$$

Given this calculation, $g(d)$ appears to be asymptotically converging to zero over time. This suggests that the growth of $b(d)$ is **sub-exponential**, or at a slower rate than exponential growth. However, the rate of decrease in $g(d)$ is quite small, so it's possible that $b(d)$ is still growing close to exponentially.

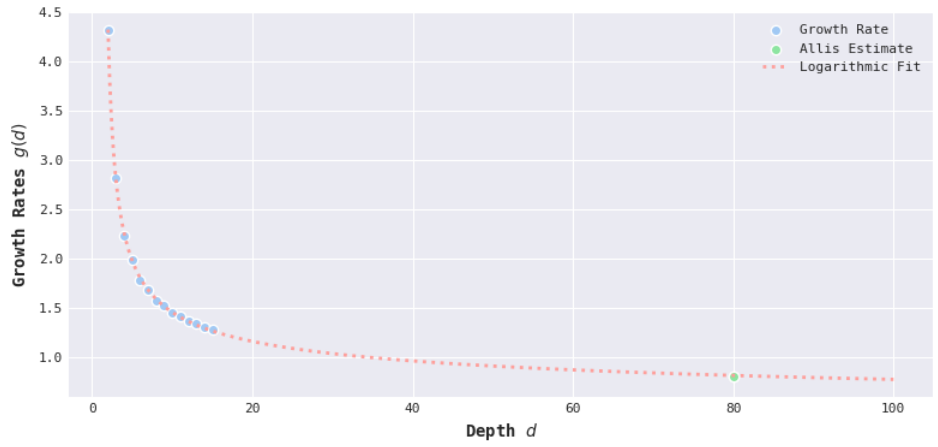

Figure 4 uses the following **logarithmic regression** to fit values of $g(d)$:

$$g(d) = \frac{W(d^X)}{d^Y + Z}$$

W, X, Y, & Z are constants:

- W: Numerator scaling coefficient
- X: Numerator exponent
- Y: Denominator exponent
- Z: Denominator offset constant

$$\begin{aligned} W &= 1.8516 \\ X &= 0.5382 \\ Y &= 0.7339 \\ Z &= -1.0415 \end{aligned}$$

Figure 4: $g(d)$ vs. d

Recall that the theoretical endpoint of chess, $d = 5898$, represents an extreme depth of analysis, which is far beyond the practical reach of even the most advanced chess engines or human players. That said, it is interesting that even at such an immense depth, $g(d) > 0$ using these constants, with an approximate value of **0.339**.

Chess can be considered a finite and deterministic game with perfect information. In principle, it can be solved through exhaustive search, similar to simpler games like tic-tac-toe or Connect Four. However, the unsolved nature of chess stems from its enormous game tree, and the fact that finding an optimal strategy requires searching through this tree more efficiently than today's readily available algorithms and computational resources allow.

The presence of a non-zero growth rate even at the game's theoretical endpoint suggests there is still a degree of complexity and branching that remains to be explored. Understandably, that complexity continues to be a fertile ground for research and innovation in the fields of artificial intelligence, machine learning, and game theory.

Contextualizing Acumen for Move Sequences

To summarize, the data above suggests that the number of possible games available to each player slowly decreases as the game progresses. This makes intuitive sense, since the primary objective is to place the opponent's king in checkmate, which requires restricting the mobility of the king and limiting the available moves of the other pieces. As a game progresses, the number of available moves decreases, and the importance of each move increases, making it more difficult for players to avoid bad move sequences.

The average chess opening lasts between **10** to **12** moves, although this can vary widely depending on the style of play and the specific opening being used. Top chess players must therefore possess an extraordinary level of skill and rote memorization to continue improving their game. Furthermore, they must be able to recognize and evaluate complex and advanced move sequences as the game progresses, and make more precise and effective moves than weaker players. This requires an incredible acumen developed through years of practice and pattern recognition, and the ability to anticipate and respond to their opponent's moves.

Mathematically speaking, we can think of a chess player's improvement as a function of the number of move sequences they are able to recognize, evaluate, and execute in sequence. As they become more skilled, they are able to recognize and evaluate more move sequences, leading to better moves and a higher chance of winning.

This idea of skill level as a function of performance is at the heart of the Elo rating system, which assigns a numerical rating to each player based on their performance in past games, and uses that rating to predict the outcome of future games. As players improve and find more winning move sequences against higher-rated opponents, their rating increases, leading to tougher competition and a greater opportunity to improve further. Ultimately, the Elo system provides a means of quantify a player's knowledge of the strongest move sequences over time.

Quantifying Chess Acumen

Arpad Elo, a Hungarian American physics professor and avid chess player, developed the Elo rating system in the 1960s to replace the then-current system, the Harkness system. Harkness's approach had serious limitations, in that it employed fixed-point adjustment to a player for each tournament win. The motivation behind Elo's work was the need for a more accurate, nuanced method of rating chess players that better accounted for an individual's performance against opponents of varying skill levels.

As such, Elo's rating system was built on relative inference based on other players' scores and assumed that the performance of players followed a normal distribution curve.[5] Elo's new methodology arrived as the chess world was undergoing significant changes in public perception. The rise of grandmasters like Bobby Fischer and the intensification of the rivalry between the United States and the Soviet Union in the realm of chess placed a greater emphasis on the need for a reliable rating system.

In the decades since its introduction, the Elo rating system has become an integral part of the chess world, serving as the primary means of measuring a player's skill and progress in the game. It has been widely adopted by national and international chess federations, such as FIDE and USCF, and has even been co-opted for use in other rating-based contexts, like League of Legends, Overwatch, and predicting election outcomes. Even Tinder uses an adaptation of Elo to rank its users' desirability.

Elo as a versatile construct has profoundly shaped the way players and spectators perceive skill, fair play, achievement, and competition. Despite the emergence of new technologies, increased access to chess knowledge, and the introduction of new populations of players, including artificial intelligence and bots, the core principles of Elo's work continue to provide a reliable framework for quantifying chess acumen.

Breaking Down the Equations

Regarding the implementation of the Elo system, its equations account for the variability in player performance by aligning it with a normal distribution curve, combining statistical probability theory and competition dynamics to more accurately describe the relationships between players.

Elo is underpinned by the following equation for a player's expected result (E_P) of a game:

---

[5] [5] Elo, A. (1978). The Rating of Chessplayers, Past and Present. p. 3-4.

$$X = \frac{R_O - R_P}{400}$$

$$E_P = \frac{1}{1 + 10^X}$$

R_P and R_O denote the current rating of the player and their opponent, respectively. The simplicity and efficiency of this equation allows the rating system to adapt seamlessly to different skill levels while ensuring accurate predictions of a game's outcome.

The choice of **400** as the scaling factor is somewhat arbitrary, but it was selected by Arpad Elo to maintain compatibility with the previous Harkness system. In essence, the value of **400** ensures that a difference of **200** points in rating indicates that the higher-rated player is expected to win approximately **75%** of their games against the lower-rated player. The scaling factor can be adjusted to change the sensitivity of the rating system, but **400** has proven to be a practical and widely accepted choice.

Once the expected results are calculated, a **Continuous Measurement** formula is applied to update each player's rating after a game. This formula is expressed as:

$$\Delta R_P = K \cdot (S - E_P)$$

Here, ΔR_P corresponds to the change in ratings, S indicates the actual result of the game *(either **1** for a win, **0** for a loss, or **0.5** for a draw)*, and K is a constant that normalizes the magnitude of rating changes after each game.

For higher-rated players or masters, K is typically set to a lower value to ensure that their ratings are less volatile and not significantly affected by a single win, loss, or draw. On the other hand, for lower-rated or weaker players, K is usually set higher, allowing their ratings to change more rapidly as they gain experience and improve their skills.

While this equation provides a means of updating player ratings based on individual games, Elo also designed a player's **Performance Rating** (PR) to offer a more comprehensive perspective on a player's skill level over multiple games. The Performance Rating is calculated as a weighted average of a player's opponents' ratings, adjusted by a scaling factor:

$$PR = \sum_{j=1}^{n} (R_{O_j} + 400 \cdot S_j) \cdot n^{-1}$$

This equation calculates the sum of the adjusted ratings of each opponent a player has faced in a series of games (j), where $j \in \{1 \ \ldots \ n\}$, with n being the total number of games played. S_j denotes the actual result of the game, and the constant **400** is again used to scale the contribution of each game with the same factor as the other formulas.

It is essential that we acknowledge the improvements and adaptations made to the chess-specific implementation of the Elo system since its inception. Elo initially noted that as the player population evolves over time, the rating system must adapt its scaling and normalization to maintain accuracy and relevance.

Notably, the emergence of computer chess engines has become one of the most influential additions to the chess landscape, essentially introducing a new population of players with unforeseen impact. As we'll explore later in this paper, AI-powered opponents are exceptionally skilled at exploring and exploiting optimal move sequences, effectively redefining our understanding of good chess at all levels.

Another significant improvement to chess Elo is the adjustment of K for different player levels, in which various federations have chosen their own values for normalization. The USCF utilizes a three-level K-factor, with different values for players below **2100** (**32**), between **2100-2400** (**24**), and above **2400** (**16**). FIDE also employs a similar approach, with K-factors of **40**, **20**, and **10** for corresponding rating bands. In both cases, modifying K allows for more accurate and dynamic ratings, ensuring that the system remains sensitive to players' skill levels and improvements over time.

Past & Present Elo Values

In order to gain insights into the evolution of Elo ratings over time, we conducted an in-depth analysis of the a series of FIDE rating files from the years 1971 to 2023. These files contain the Standard FIDE ratings of all registered players for January of each year.

Our primary goals were to examine the changes in Elo ratings over time and to explore the possible reasons behind these changes, with a focus on the last decade. To investigate these questions, we plotted the average Elo ratings of the top 1%, 10%, and 25% players for each year. The dataset reveals intriguing trends and insights into the development of chess skills and the Elo rating system over the past several decades.

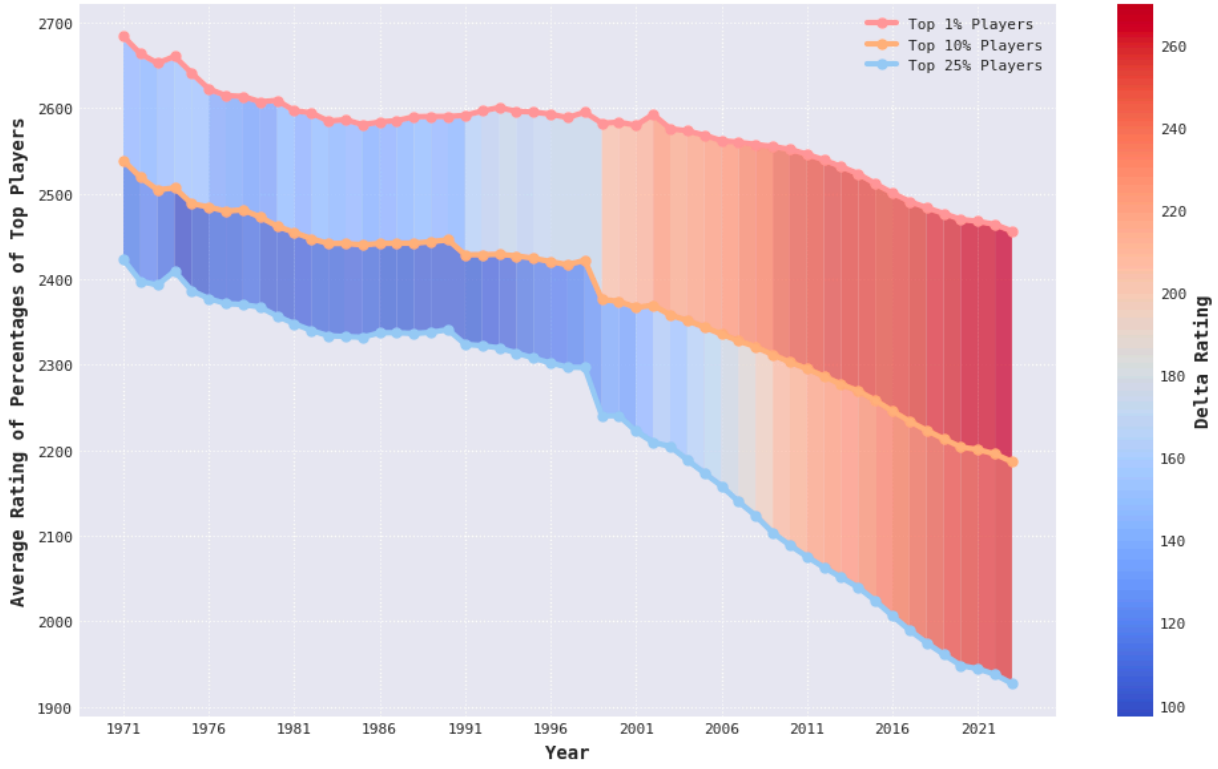

Figure 5: Average FIDE Rating by Year for Various Percentages of Top Registrars

For instance, in 1971, the average Elo rating for the top 1% was **2684**, but by 2023, the average Elo rating for the top 1% had decreased to **2456**, with the top 10% and top 25% undergoing similar declines. However, this

decrease in Elo ratings over time might not necessarily indicate that players are getting worse. The observed decrease in ratings could be attributed to factors such as the expansion of the player pool and rating distribution becoming more spread out. Given the advancements in chess resources, training methods, and technology, it is plausible that a **2500**-rated player today could be stronger than a player with the same rating from 50 years ago.

The increase in FIDE player acceptance can change the analysis due to the deflationary effect on ratings, diversification of rating distribution, and the influx of players with varying skill levels. Consequently, drawing direct comparisons between players from different periods is challenging, though the relationships of the players within each yearly pool yield even more interesting conclusions.

That said, we do know that top-level players have leveraged chess programs to study and memorize moves, surpassing their predecessors in terms of knowledge and Elo ratings, predominantly because many of them have created content using those tools and programs Putting this into perspective, the peak FIDE rating for the current strongest chess player, Magnus Carlsen, is **2860**, whereas in 1960, the highest-rated player, Mikhail Tal, had a rating of **2705**, a 155 point difference.

To illustrate the implications of that difference, consider the Elo rating equation applied to Magnus (E_A) and Mikhail (E_B):

$$E_A = \frac{1}{1 + 10^{\frac{2705 - 2860}{400}}} = \frac{1}{1 + 10^{-0.3875}} = \frac{1}{1.409732} \approx 0.709$$

$$E_B = \frac{1}{1 + 10^{\frac{2860 - 2705}{400}}} = \frac{1}{1 + 10^{0.3875}} = \frac{1}{3.4406} \approx 0.291$$

The Elo calculation suggests that Magnus has roughly a 71% chance of winning, which is more than 2 out of 3 games. However, considering the profound impact of bots on the highest level of play in chess, it is likely that Magnus would win 19 out of 20 games, or 95%. The highest-level chess players today use computers to perfect their opening, middle game plans, and endgames, memorizing the play of 3000+ rated computers, which gives them a significant advantage against players from pre-computer chess eras.

Throughout the years, we observed a widening gap between the top 1% and the top 10% or 25% of players. This widening gap suggests that the top players are increasingly distancing themselves from the rest of the player pool, possibly due to their improved creativity and execution of new move sequences. We speculate that with their already exceptional understanding of chess, these players are likely to benefit the most from the move creativity that top chess engines are renowned for. By thoroughly studying and incorporating the unique insights provided by these engines into their own gameplay, top players can further elevate their skills and develop novel strategies that give them an edge over their competitors. This could be a crucial factor in the increasing gap between the top players and the rest of the rating pool.

Moreover, the availability and accessibility of high-quality chess resources, databases, and training methods have grown exponentially in recent years, particularly for elite players. This access to cutting-edge tools and resources, coupled with the influence of advanced chess engines, allows top players to deepen their understanding of the game and push the boundaries of chess theory. That growing gap between the top 1%

and other players may create a sense of urgency among less-skilled populations to catch up. This urgency could result in more bot usage, as these players strive to improve their skills and keep up with the increasingly competitive chess world.

Projecting a Test Case's Ratings

To illustrate what a journey in Elo may look like, we can examine the case of Player A. This player creates a new chess account and plays 30 games on it, then compares the new account's rating to the rating of an account they have been playing on for years. The results from the 30 initial 5-minute games on the new account are presented in the figure below. These results, when analyzed in conjunction with the existing account's rating, offer insight into Player A's ability to understand and execute complex move sequences, thereby reflecting their chess intuition as measured by the Elo rating system.

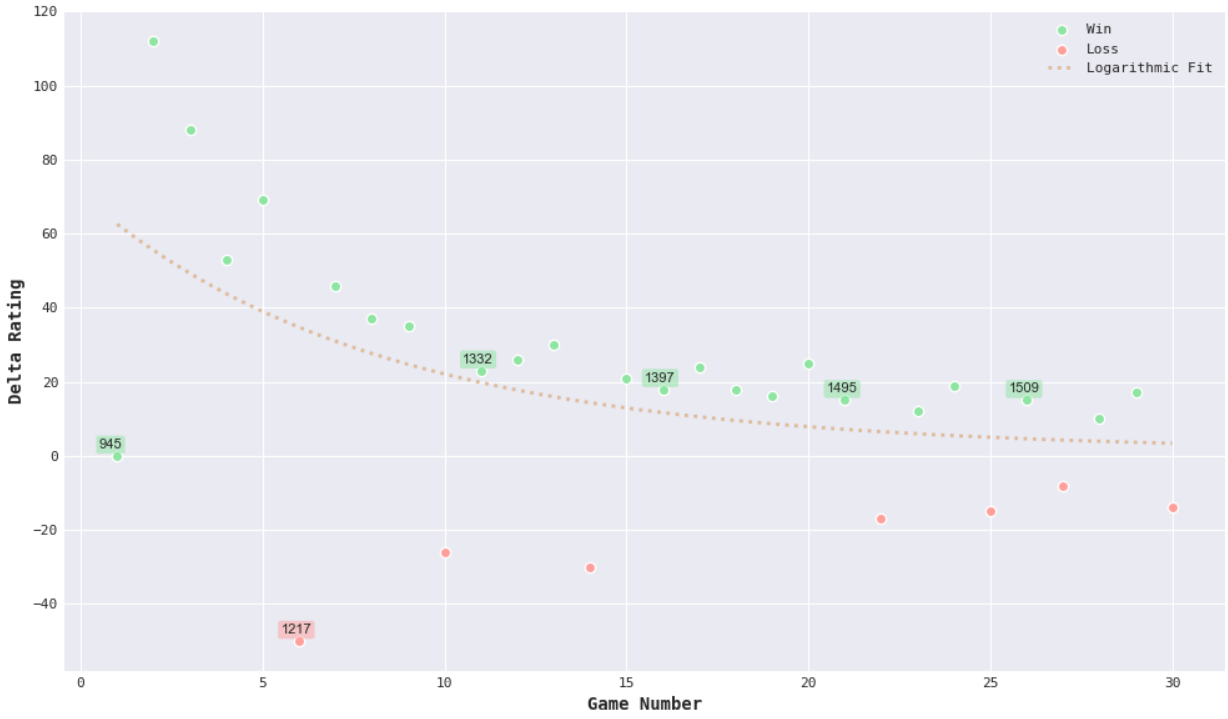

Figure 6: Player A's First 30 Games

Player A's current rapid rating on *chess.com* is **1812**. Although it is unclear whether Player A performed at an **1800** level during this assessment, they managed to reach a rating of **1514** after thirty games, receiving 23 wins and 7 losses.

Using the logarithmic regression derived from Player A's first 30 games, we can estimate their potential rating after 50, 100, 200, and 500 games. It is important to note that these projections are based on the current trend and may not accurately represent future performance. The custom logarithmic curve above generally follows the equation:

$$f(x) = W \cdot \log(x - Y)^X + Z$$

$$\begin{aligned} W &= 2.525 \cdot 10^{96} \\ X &= -130.570 \\ Y &= -200.177 \\ Z &= 1.05g \end{aligned}$$

As Player A's rating increases, it can be expected that their ability to process and select from various move sequences will improve. However, quantifying the exact number of move sequences they can handle is challenging. When compared to an established chess engine like LCZero, a human player's calculation capability is significantly lower. While a chess engine can evaluate millions of positions per second, human players rely on pattern recognition, intuition, and strategic understanding to make decisions.

Nonetheless, let's estimate Player A's rating after the specified number of games:

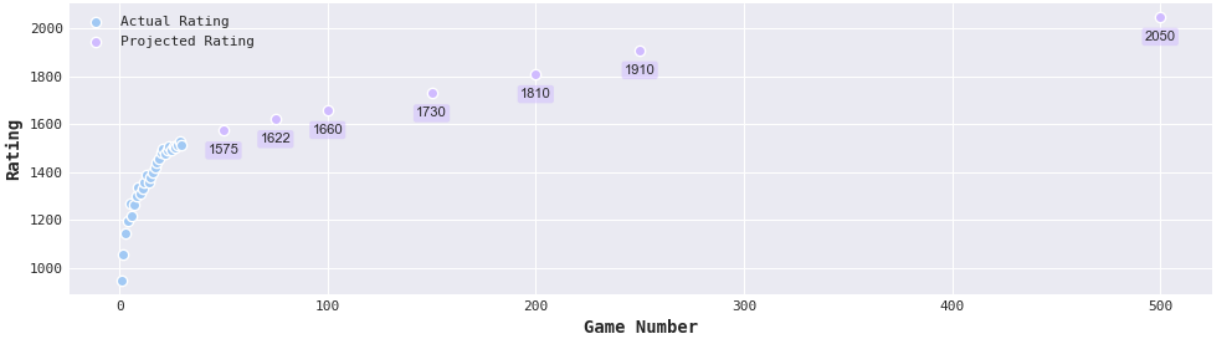

Figure 7: Projecting Player A's Rating at 500 Games

These projected ratings are purely based on the logarithmic regression and do not account for factors such as improvement in strategic understanding or changes in learning rate. As Player A's rating increases, their ability to calculate and select from move sequences will likely improve, but it will still be far from the capabilities of advanced chess engines.

The Bot Bump

Building on our previous observation of the widening gap between the top 1% and the rest of the player pool, we now turn our attention to the role of advanced chess engines in shaping modern chess strategies. The influence of artificial intelligence on the game and how it interacts with human decision-making, particularly for elite players, becomes a crucial factor in understanding the changing dynamics of chess competition.

As players reach higher ratings, they may become more comfortable breaking away from standard chess conventions, a notion supported by the work of mathematical physicist Roger Penrose[6]. Chess engines, with their immense computational power, excel at calculating vast numbers of move sequences and determining optimal moves in most situations. However, they are still bound by the limits of their algorithms and may struggle with positions that require a fundamentally different kind of reasoning, such as the infamous Penrose position. In contrast, humans possess creativity and intuition, allowing them to discern unique solutions in seemingly hopeless situations, as demonstrated by their ability to solve Penrose's position.

This difference in approach suggests that the interplay between human creativity and chess engine precision has the potential to push the boundaries of chess knowledge. Humans can leverage their creativity and intuition to challenge the assumptions made by chess engines and explore novel ideas that may not be immediately apparent in computational analysis.

Grandmaster Vladimir Kramnik's quote, "You don't even play your own preparation; you play your computer's preparation," highlights the significant role of chess engines in the preparation and training of elite players.

---

[6] ChessBase. (last modified [27 March 2017]). "A chess problem holds the key to human consciousness." en.chessbase.com/post/a-chess-problem-holds-the-key-to-human-consciousness.

Our data analysis reveals a striking correlation between the rising top engine Elo ratings and the increasing gap between the top 1% and 10% of players. The curves align closely, suggesting that elite players are leveraging the power of advanced chess engines to further hone their skills, leading to a widening rift between them and the rest of the player pool.

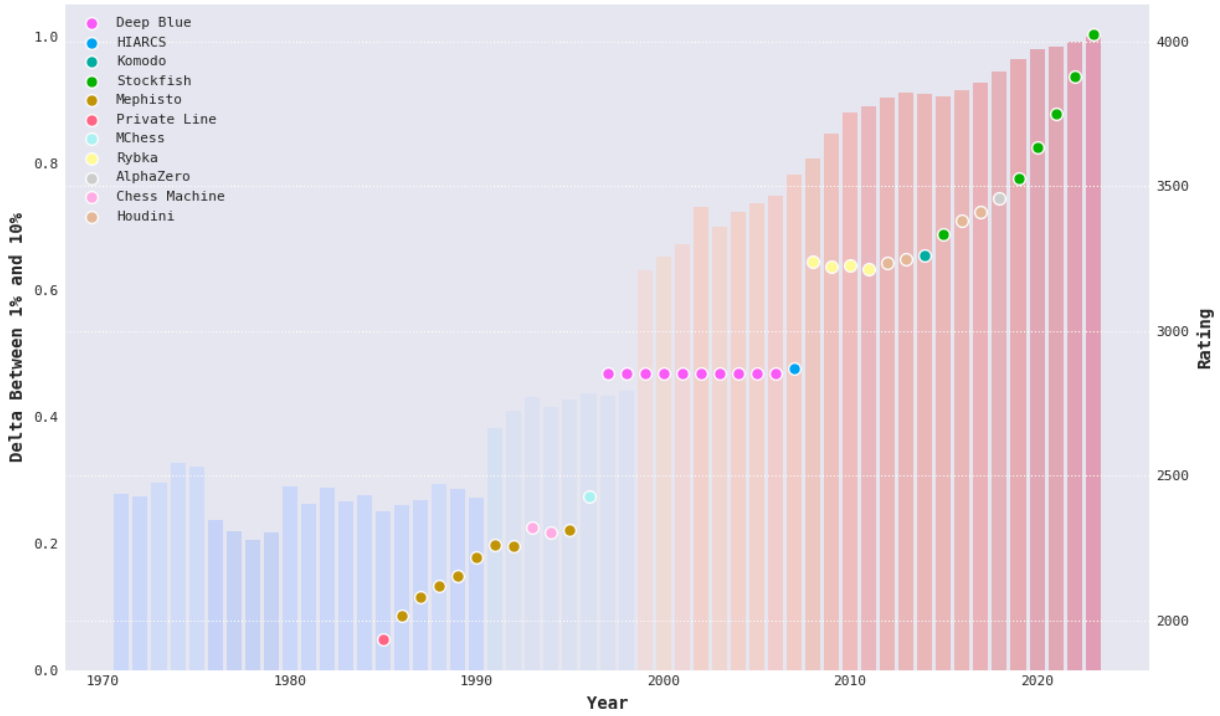

Figure 8: Bot Elo vs. Delta Rating of Top 1% Players

This chart, generated using data on leading engine Elo ratings since the 1980s, illustrates the parallel between engine ratings and the growing divide between these two groups of players. We re-use the same filled delta from the previous chart as a background bar chart to highlight the similarities across the two metrics. This correlation emphasizes the idea that top players are harnessing cutting-edge advancements in chess engines to improve their gameplay, contributing to the growing divide between the top 1% and the remaining players.

Bots and Their Algorithms

The rapid development and emergence of advanced chess bots, in tandem with the game's soaring popularity, has profoundly transformed how players interact with the game and substantially influenced the average chess player's capacity to enhance their skills. The complexity of chess lies in the vast number of possible move sequences, with an immense game tree that challenges even the most advanced artificial intelligence algorithms.

In this section, we will delve deep into the open-source chess bot **Leela Chess Zero** (Lc0), which has played a significant role in advancing chess AI due to its transparent methodologies and open-source nature. As Lc0 is an open-source project with a readily accessible GitHub repository and a wealth of academic research behind it, we can explore its construction and methodologies in detail, providing valuable insights into modern chess AI.

As we explore Lc0 and other chess bots, we will address some of the questions raised earlier in this paper, including how Mittens was designed to be "better" than other prominent bots and whether bots can actually make human chess players more competitive as a whole. Moreover, we will discuss the implications of these bots' increasing accessibility to a wider audience, as well as the temptation for top players to cheat by using them.

Introducing Leela

Lc0 was heavily inspired by its closed-source DeepMind predecessor, AlphaZero, which utilizes a combination of deep neural networks and reinforcement learning, and a novel tree search algorithm to achieve groundbreaking performance. The creators of Lc0 adopted a similar combination of methodologies, but with a distributed, community-driven effort to contribute to its training and development.

Lc0's commit history began in January 2018, shortly after AlphaZero's groundbreaking achievements were made public. Since then, multiple versions have been released, with each iteration demonstrating significant improvements over its predecessor. Unlike traditional chess engines, which often focus on material advantage and precise calculations, Lc0 exhibits an intuitive and creative approach to the game, drawing comparisons to human Grandmasters. This has led to a shift in how chess players and enthusiasts perceive the role of AI in chess, with an increased emphasis on understanding, learning from, and emulating Lc0's unique playing style.

Note that the inclusion of "Zero" in the name signifies that no human knowledge has been added to Lc0's understanding of chess[7], apart from the rules for piece movement and victory conditions.

Leela employs a unique combination of neural networking and Monte Carlo Tree Search (MCTS) with a PUCT algorithm (*explained below*) to evaluate positions and explore move sequences. This combination is precisely what differentiated it upon release from other prominent bots, which rely on finely-crafted, human-built systems for generating value and policy.

Decision-Making Via Neural Network

Lc0 is trained through a combination of supervised learning and reinforcement learning, which serve as the foundation for the bot's decision-making process. Its neural network takes a game state s_t as input and outputs two values[8]:

- $\mathbf{p}$: a **policy vector** representing the probability distribution over possible moves
- v: a scalar **value estimation** of the position between $[-1, 1]$

The network consists of several layers of convolutional and fully connected layers, with residual connections and batch normalization. The final layers of the network are split into two separate "heads," one for the policy and one for the value. The policy head outputs the move probabilities, while the value head estimates the value of the position.

We can understand the computation for a given s_t as:

$$f_\theta(s_t) \mapsto (\mathbf{p},\ v)$$

[7] LCZero. (accessed 27 March 2023). "Why Zero?" LCZero Dev Wiki. lczero.org/dev/wiki/why-zero/

[8] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354-359. p. 355.

Here, θ represents the network's parameters, which include the weights and biases of each layer. Once trained, the neural network serves as an evaluation function within the MCTS, guiding the search towards promising moves and estimating position values. By integrating the neural network into the MCTS framework, Lc0 can efficiently explore the game tree and make well-informed decisions in complex positions, ultimately leading to a strong and adaptive playing style.

Aligning Predictions with MCTS Policies

Once the neural network is integrated into the MCTS framework, it is crucial to ensure the policy and value heads provide meaningful guidance for the search process. To achieve this, the network is trained to minimize a loss function, which encourages accurate value estimation and a policy that aligns with the target probabilities obtained from the MCTS.

The loss function used for training the neural network is as follows:

$$L(\theta) = (z - v)^2 - \pi^T \log \mathbf{p} + c\|\theta\|^2$$

- $(z - v)^2$: the mean squared error between the predicted value head and the actual game outcome (z) for a given position
- $\pi^T \log \mathbf{p}$: the cross-entropy loss between the predicted probability distribution of legal moves and the search probabilities obtained from the MCTS (π^T), which are derived from the visit counts of the tree search
- $\|\theta\|^2$: an L2 regularization component that helps to prevent overfitting by penalizing large weights in the neural network
- c: a constant that controls the strength of the L2 regularization

This usage of a loss function in training the neural network is essential for Lc0's performance, as it facilitates the alignment of the network's predictions with the MCTS-derived search policies. The combination of mean squared error, cross-entropy loss, and L2 regularization has been widely adopted in the field of deep reinforcement learning. It has proven effective in various applications like robotic manipulation and locomotion, Deep Q-Networks, and even some frameworks for autonomous vehicles.

Balancing Exploration and Exploitation

Lc0 employs a variation of MCTS known as the **Predictor + Upper Confidence Bound Tree Search (PUCT)** algorithm[9], striking a balance between exploration and exploitation. While PUCT shares similarities with traditional MCTS, it diverges in its integration of a neural network to inform the search. This neural network supplants the need for rollouts, which are typically employed in standard MCTS models to sample and evaluate games until a terminal state is reached.

Here's a mathematical breakdown of the PUCT algorithm:

$$\mathbf{p} = [p(a_1, s_t),\ p(a_2, s_t),\ \ldots,\ p(a_n, s_t)]$$

$$\forall a \in \mathbf{p} : a_t = \underset{a}{\operatorname{argmax}} \left(Q(a, s_t) + c \cdot p(a, s_t) \cdot \frac{\sqrt{\sum_b \cdot N(b, s_t)}}{1 + N(a, s_t)} \right)$$

[9] LCZero. (accessed 27 March 2023). "Technical Explanation of Leela Chess Zero." LCZero Dev Wiki. lczero.org/dev/wiki/technical-explanation-of-leela-chess-zero/

$Q(a, s_t)$ is the **exploitation term**, an estimated mean value of taking some action a (*a chess move*) at s_t (*a given board orientation*).

By contrast, $c \cdot p(a, s_t) \cdot \frac{\sqrt{\sum_b \cdot N(b, s_t)}}{1 + N(a, s_t)}$ is the **exploration term**:

- c: a constant that controls the exploration-exploitation trade-off
- $p(a, s_t)$: the prior probability of taking a at s_t, as part of the vector produced by the neural network
- $N(a, s_t)$: the number of visits that a makes to s_t during the tree search
- $\sum_b \cdot N(b, s_t)$: the total number of visits to s_t summed over all actions b

The arguments of the maxima (argmax) are used to find the action that maximizes the balance between the exploitation of high-scoring moves and the exploration of less visited moves. Striking that balance ensures that the search considers a diverse range of move sequences. Because the neural network provides a powerful prior for the search, allowing it to focus on promising moves, argmax ensures that the search does not become too narrow, helping it avoid getting stuck in optimization spaces where a solution is "better" than all its neighboring solutions, but not necessarily the best possible solution.

When considering PUCT's **time complexity**, the algorithms performance is defined by the game's branching factor(s) and the maximum depth of the tree, which we've previously evaluated to be $\mathcal{O}(b^d)$. However, it is important to note that the actual number of iterations and the depth of the tree can be limited by available computational resources or desired search depth, as mentioned earlier.

On the other hand, while the complexity of evaluating a neural network depends on its architecture, it is generally linear with respect to the number of neurons and connections present in the model.

Once the PUCT algorithm has been executed and the argmax operation is complete, the resulting a_t would contain the move that has the highest combined score from the exploitation and exploration terms, which would be the next move played by the AI in-game.

Fine-Tuning the Algorithm's Search Strategy

During MCTS, Lc0 uses temperature scaling to control the level of exploration. Temperature scaling adjusts the move probabilities by raising them to the power of an inverse temperature parameter, τ, and then normalizing the probabilities[10]. Note that higher values of τ increase the level of exploration, while lower values make the search more focused on the highest-probability moves:

$$\alpha = \frac{1}{\tau}$$

$$p_a^* = \frac{N(a, s_t)^\alpha}{\sum_b N(b, s_t)^\alpha}$$

- $N(a, s_t)$: previously defined in the PUCT algorithm, and used in temperature scaling to adjust the probabilities based on the exploration of the search

[10] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of Go without human knowledge. Nature, 550(7676), 354-359. p. 358.

- $\sum_b N(b, s_t)^\alpha$: normalizes the adjusted probabilities by summing the raised probabilities to 1

The result of this formula, p_a^*, represents the adjusted probability of action a after applying temperature scaling. These probabilities are used during the selection phase of the search when deciding which child nodes to traverse. A temperature of $\tau = 1$ is used during the first 30 moves of each game in self-play to encourage exploration, while for the remainder of the game, the temperature is set to approach zero effectively choosing the move with the highest visit count.

These adjusted move probabilities will influence which nodes are chosen for traversal and thus have an impact on the overall search. By modulating the temperature parameter, the balance between exploration and exploitation can be fine-tuned, which allows Lc0 to adapt its playing style and decision-making based on the specific position and game context.

Bellman Equation

A key aspect of solving for the optimal $Q(a, s_t)$ values involves the Bellman equation, which can be used to express the relationships between the current game state and all future game states. The Bellman equation is based on the **principle of optimality**, which states that for an inductive strategy to be optimal, it must make the best possible decisions at every step, considering the new situations arising from previous actions.

By contrast, the more commonly-used **greedy inductive policy** is a strategy that chooses the action with the highest immediate reward at each time step, without considering the long-term consequences of these actions. While a greedy inductive policy can sometimes result in good decisions, it often fails to consider the overall impact of its choices, leading to suboptimal performance.

Backward induction is a method used to solve the Bellman equation by identifying the most optimal action at the last node in the decision tree (*in this case, checkmate or a draw*) and working backwards until the best action for every possible situation is determined.

We can understand how the PUCT derives value in terms of the Bellman constraint for Q-values[11]:

$$Q(a, s_t) = u(a, s_t) + \sum p(s_t^* : a, s_t) \cdot \max_a Q(a, s_t^*)$$

- $p(s_t^* : a, s_t)$: the probability of moving to a new state given the current state and action
- s_t^*: the board state after a is taken and the opponent has responded
- $u(a, s_t)$: a move's instantaneous utility, which can be replaced with the value head output, $v(s_t)$

Recall that $v(s_t) \in [-1,1]$ is the output of the value network, which estimates the probability of winning from the current position.

In the Lc0 context, we can take the maximum value of $Q(a, s_t)$ over the current action to represent the total value function $V(s_t)$ after incorporating the value network output into the policy network's probability predictions:

---

[11] Maharaj, S. et al. (2022). Chess AI: Competing Paradigms for Machine Intelligence. p. 3-4.

$$V(s) = \max_a \left(v(s) + \sum p(s_t^* : a, s_t) \cdot \max_a Q(s^*, a) \right)$$

Centi-Pawn Conversion

After applying the temperature-scaled policy and value results from the neural network back to the a_t under consideration, Lc0 takes the $Q(a, s_t)$ in the range of $[-1, 1]$ and converts it into a traditional centipawn (P_c) evaluation.

While these formulas are not readily available, given that the result of the Bellman equation is a win probability, the centi-pawn values generated by Lc0 may resemble the approach for converting win advantage to pawn advantage by Fischer and Kennan in 2007[12], which they devised for approximating Elo values:

$$P_c = 4 \log_{10} \frac{V(s)}{1 - V(s)}$$

Leela's ability to efficiently explore move sequences and evaluate positions stems from the synergy between the neural network and the MCTS. By leveraging the neural network's capacity for learning and generalization, it can quickly identify promising moves and focus its search efforts on the most relevant parts of the game tree. That said, the ongoing exploration of alternative search algorithms within the intersection of AI and Game Theory highlights the potential for future advancements, with PUCT remaining the dominant choice for the time being.

Ply, Skill, and the Sheer Number of Calculations

In this paper, we have explored the relationship between the number of possible move sequences in chess and a player's skill level in multiple sections. Our analysis suggests that as a game progresses, the number of strong the importance of each move increases, making it more difficult for players to avoid losing move sequences.

To further illustrate this point, we analyzed a random sample of **100,000** Lichess games from January 2023, and found that the average ply was around **20** at 600, **30** at 1300, **40** at 2500, and surpassing **50** at 2800. These ply values represent a precipitous increase in move sequences, which underscores just how much more complex chess becomes at higher skill levels.

Using our previously developed formula, $p(d) = e^{m \cdot d + b}$, we estimate the following quantities of move sequences the player must traverse:

- At **20**, there are approximately $1.6 \cdot 10^9$ possible move sequences.
- At **30**, this number jumps to $5.5 \cdot 10^{16}$.
- At **40**, the number of move sequences becomes even more daunting at $1.7 \cdot 10^{28}$.
- And finally, at a ply value of **50**, there are an estimated $1.7 \cdot 10^{42}$ possible move sequences.

The jump from a ply of **40** to **50** is particularly staggering, underscoring just how much more complex the game becomes at the highest levels of play.

---

[12] Chess Programming Wiki. (accessed March 27, 2023). "Pawn Advantage, Win Percentage, and Elo." https://www.chessprogramming.org/index.php?title=Pawn_Advantage,_Win_Percentage,_and_Elo

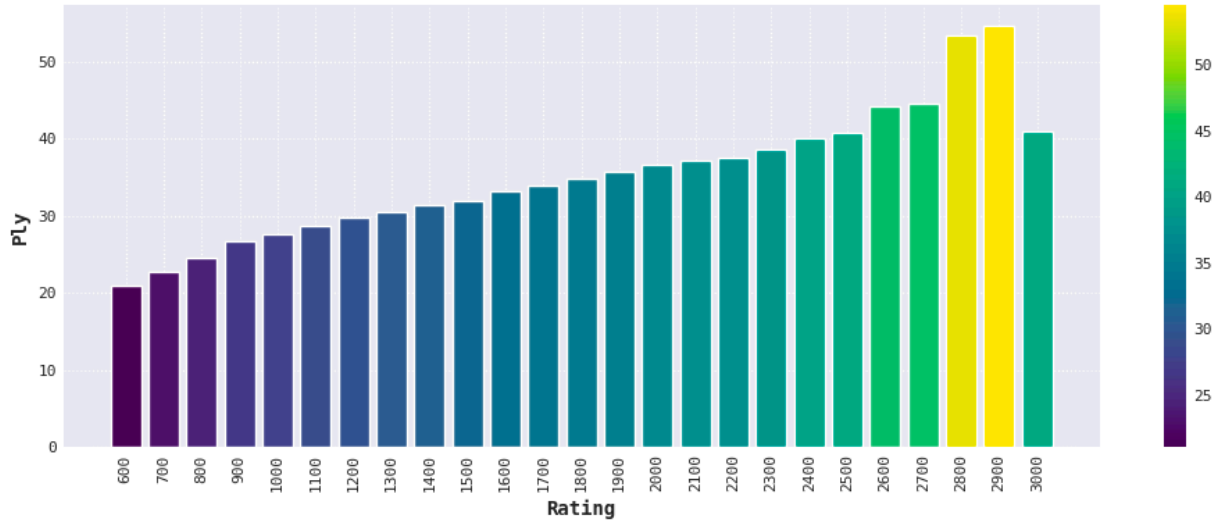

Figure 9: Ply vs. Elo Rating for 100,000 Lichess Games in Jan. 2023

When we consider that Stockfish's Elo rating at the end of 2022 was nearly 3900, we can begin to appreciate the enormous leap in skill level required to even understand chess and the sequences traversed at that level. That precipitous jump in complexity can only hope to be understood by players with a similar gift for eschewing chess convention in the name of brilliance, which further underscores the growing divide we seen between different percentiles of ranked players.

While artificial neural network (ANN) engines such as Lc0 have been praised for their more "intuitive" calculation, they are still far more concrete than human chess players in their ideas. Machine learning is very different from human learning, and the resulting chess play is often quite different as well. However, engines do have the potential to play more human-like than traditional chess engines, such as Stockfish, which are programmed using hundreds of heuristics governed by trial and error.

In conclusion, the relationship between ply and skill level in chess is a complex one that is influenced by many factors, including experience, knowledge, and pattern recognition. As players improve, they are able to recognize and evaluate more move sequences, leading to better moves and a higher chance of winning. The Elo rating system provides a means of quantifying a player's knowledge of the strongest move sequences over time, and can help to predict the outcome of future games. Overall, our analysis suggests that the complexity of chess grows exponentially with skill level, and underscores the incredible acumen required to play the game at the highest levels.