PAUL GRÜNKE

Chess, Artificial Intelligence, and Epistemic Opacity

In 2017 AlphaZero, a neural network-based chess engine shook the chess world by convincingly beating Stockfish, the highest-rated chess engine. In this paper, I describe the technical differences between the two chess engines and based on that, I discuss the impact of the modeling choices on the respective epistemic opacities. I argue that the success of AlphaZero's approach with neural networks and reinforcement learning is counterbalanced by an increase in the epistemic opacity of the resulting model.

Keywords: *Opacity, Machine Learning, Modeling, Chess*

Author Information

Paul Grünke, Karlsruhe Institute of Technology, https://orcid.org/0000-0002-3576-1921

How to cite this article

Grünke, Paul. „Chess, Artificial Intelligence, and Epistemic Opacity",
Információs Társadalom XIX, 4. no (2019): 7–28.
https://dx.doi.org/10.22503/inftars.XIX.2019.4.1

Chess, Artificial Intelligence, and Epistemic Opacity

Paul Grünke

Abstract
In 2017 AlphaZero, a neural network-based chess engine shook the chess world by convincingly beating Stockfish, the highest-rated chess engine. In this paper, I describe the technical differences between the two chess engines and based on that, I discuss the impact of the modeling choices on the respective epistemic opacities. I argue that the success of AlphaZero's approach with neural networks and reinforcement learning is counterbalanced by an increase in the epistemic opacity of the resulting model.

1 Introduction

Games have always been a welcome area for AI developers to test and develop their newest techniques. Chess is the most famous game in this context and the one that has been studied the most by computer scientists. The fascination for a machine playing chess dates back to the late 18th century when a chess automaton was exhibited throughout Europe. Many of the great minds of early computer science such as Charles Babbage, Alan Turing and John von Neumann devised their own approaches towards a program that would be able to play chess. This culminated in the victory of IBM's "Deep Blue" computer versus the (at the time) reigning world champion, Garry Kasparov, in 1997, which was a major event for the artificial intelligence community. It also started the domination of computer engines in the game of chess; which has today developed so far that a match between a computer and a human player would not be interesting anymore[1] and all top players as well as many amateur players are relying very heavily on computers in their preparation and training (Chessentials 2019). Today's chess engines are very sophisticated programs that include specifically designed search algorithms and evaluation functions, incorporate opening books and endgame databases.

In 2017, 20 years after the success of Deep Blue, a new kind of chess engine has been introduced. AlphaZero is a chess engine based on a neural network created by British AI company DeepMind. Its only inputs were the rules of chess and within a few hours of training via playing against itself it became the strongest chess engine in the world. In this paper, I describe this new approach towards chess engines, discuss its differences from other approaches and investigate the hypothesis that the success of this approach with neural networks and reinforcement learning is counterbalanced by an increase in epistemic opacity of the resulting

[1]Hikaru Nakamura, at the time rated the sixth best player in the world, played a match against a strong chess engine with getting additional material or moves in 2016. He drew three games and lost one (Chabris 2016).

model. An increase in epistemic opacity usually leads to a decrease in our ability to understand and control the resulting model as well as limit our options of learning from it.

In the following sections, I first sketch the development of chess engines (Section 2), and then describe and compare AlphaZero and the strongest traditional chess engine (Section 3). In Section 4, I introduce the concept of epistemic opacity, compare the most advanced classical chess engine with AlphaZero with respect to their epistemic opacities, and discuss the different kinds of opacities that are involved and then conclude (Section 5).

2 History of Chess Engines and AI

In 1770, Wolfgang von Kempelen presented to the Habsburg Archduchess Maria Theresa what would be known as The Turk. It was a machine, which consisted of a wooden man sitting at a cabinet with a chessboard on top of it. The machine was able to move the chess pieces on the board seemingly autonomously. This in itself would not have been very impressive, if it were not for the fact that The Turk was a very good chess player defeating most of its challengers, including famous personalities of the time such as Napoleon Bonaparte, Benjamin Franklin and Charles Babbage. Theories about how the machine works developed quickly, sparked by the same disbelief that led the British author Philip Thicknesse to write: "That an automaton can be made to move the Chessmen properly, as a pugnacious player, in consequence of the preceding move of a stranger, who undertakes to play against it, is utterly impossible" (Thicknesse 1784). As it turned out, he was right for the time being since a small-statured player hidden in the wooden cabinet operated The Turk. Even though the Turk was a hoax it played an important role since it fascinated people with the idea of an intelligent machine and raised questions about the possibilities of thinking machines which are still relevant today (Morton 2015, Standage 2003).

Thicknesse's statement was proven wrong for the first time in the 1950s. Already in the late 1940s both Alan Turing and Claude Shannon (Shannon 1950) created algorithms that were able to play chess. Since no computers with the ability to compute the algorithm were available yet, Turing tried the program in 1951 by calculating everything manually. Also in 1951 Dietrich Prinz, a colleague of Alan Turing, implemented a program which was able to solve mate-in-2 problems. Finally, in 1957 Alex Bernstein, an IBM engineer, implemented a program on a computer for the first time, which was able to play a game of chess (Chessentials 2019).

The first chess engines were not very strong and could be beaten easily by strong amateur players. During the next decades, the quality of the chess engines increased continuously due to developments both in hardware and software and this culminated in the most important event in computer chess history: the IBM computer Deep Blue beating the reigning world champion Garry Kasparov in 1997 in a match of six games. After having won matches against predecessors of the program, Kasparov lost this match 3.5-2.5. Once again, the human-vs-machine setup in chess sparked fascination as well as fear of the developments of artificial intelligence. This time legitimately, as a machine had become strong enough to beat the arguably best chess player of all time (Krauthammer 1997). In the years after the Deep Blue match, a few more matches were played between world-class chess players and chess engines with mixed results, but these matches

stopped after 2006, when Vladimir Kramnik, Kasparov's successor as world champion, also lost a six game match. Chess engines now mainly play against each other in Chess Computer Championships. The winner of many of the most recent Computer Chess Tournaments and one of the highest-rated Chess Engines is an open-source program called Stockfish. The general programming approach of Stockfish is similar to the one of Deep Blue, using "sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades" (Silver et al. 2018).

In 2017, the AI company DeepMind presented a new kind of chess engine: AlphaZero. Its approach is radically different from the former chess engines; the only input given to AlphaZero were the rules of chess. Using reinforcement learning on a specifically tailored neural network, AlphaZero was then trained by playing against itself for nine hours. The resulting program played a match against Stockfish over 1000 games that it won convincingly (winning 155 games and losing 6). The result in itself is already surprising. Even more surprising to the chess community was the style of the new engine, which is radically different from previous ones. AlphaZero appears to play in a risky attacking style but nearly never runs the risk of losing. Besides that, AlphaZero introduced a number of new motifs and strategies in all stages of the game, which have already been adopted by elite chess players. Therefore, there are more than enough reasons for chess players to delve into the depths of AlphaZero, analyse its games and try to understand the reasons for its success. For philosophers of science, this is an opportunity to gain a better understanding of the success of neural networks and machine learning techniques and to discuss the possible limits inherent to this method.

3 Comparing AlphaZero and Stockfish – a new kind of chess engine

In this section, I compare the techniques used for programming AlphaZero and Stockfish respectively and highlight relevant differences. This will lay the foundation to assess their respective epistemic opacities in the next section.

Chess engines are generally based on two core components. They have a way to evaluate positions and they have a search algorithm that determines which moves are available and in which order they should be considered. For both of these components Stockfish and AlphaZero use significantly different approaches.

3.1 Stockfish

Stockfish represents chess positions by using a vector that has chess-specific features as elements. These are handcrafted and include elements, which represent for example the pieces each player still has left, the safety of the kings or the pawn structure. The programmers chose those features in cooperation with strong chess players. Each of these features has been included because it has proven relevant in human experience of playing chess and successful in test runs with the engine. Each of these features has a specific weight assigned to it and the evaluation of the position results from a sum of all the features multiplied with their weights. This evaluation is then output in pawn-equivalents.

+2.00 for example means that the player with the white pieces has an advantage that is evaluated equivalently to having two extra pawns (Silver et al. 2018).

In order to figure out what the best move in a specific position is, Stockfish spans a tree of possible developments and evaluates the resulting positions using the evaluation function. This evaluation is only applied to "quiet" positions, i.e. positions without unresolved captures or checks. Therefore, once the desired depth of calculation is reached, there is a quiescence search[2] implemented to resolve all possible tactical elements of the position before the evaluation function is applied to the resulting positions.

Since the amount of positions that have to be evaluated grows roughly by factor 40 for each new level of depth, many heuristics and algorithmic strategies are necessary to cut unnecessary searches and evaluations. The main tool used for this by Stockfish is the alpha-beta algorithm. It is a variant of the minimax algorithm that is common for the evaluation of all kinds of two player zero-sum games. Since it is assumed that both players will act optimally, one cannot simply use the highest value of all the evaluations as the result. Instead, one has to go through all the options to check whichof the possible moves of one player leaves the worst options for the other player, since the other player will always choose the best of these options. The alpha-beta search is an optimization of this algorithm that allows eliminating the need to search through all of the different paths. Assume that we already have found a move that guarantees player 1 an equal position, i.e. there is no move from player 2 in response to this move, which leads to any advantage for player 2. From now on, when we consider alternative moves for player 1 and any of the evaluations show an advantage for player 2, we can skip the rest of the search for this particular move, since it is inferior to the move that we already found before. Clearly, this approach works best if we consider the best moves as early as possible. If we would consider the moves in ascending order of strength, we would have to go through all of the possible positions. If instead we always manage to consider the strongest move for each player first, the alpha-beta algorithm would enable us to reduce the nodes that have to be evaluated from x to something close to the square root of x. Therefore, move-ordering is another very important part of the algorithm. Once the move list is generated, it gets ordered using a number of heuristics, most importantly trying the best move from the previous search first, if the search has been deepened (Silver et al. 2018, Samuel 1959).

Apart from these very sophisticated algorithms, Stockfish uses an opening book to choose moves in the first phase of the game as well an endgame tablebase that includes the best moves for all positions with six or less pieces left on the board.

The above-described techniques are used by most of today's strong chess engines as well as by earlier versions such as Deep Blue.

3.2 AlphaZero

AlphaZero uses none of the techniques described above.

At the core of AlphaZero is a deep neural network that has been trained via reinforcement learning[3]. There is no domain-specific knowledge or data used as input; the

[2] Quiescence searches are part of traditional chess engines for a long time already. The challenge for the programmer is to extend the search until a position is reached that is suitable to be evaluated by the evaluation function but use minimal resources to do that. For different techniques, see for example (Beal 1990).

training phase has been done exclusively via self-play. The same approach has been used for the games of Shogi and Go successfully[4].

Chess moves are represented in a two-step process in AlphaZero. The first step is to pick up a piece, i.e., identifying the square from which a piece is moved. The second step is deciding which move this piece should make. The input into the neural network therefore contains the 64 squares and all the move possibilities each piece would have on each of these squares. Each square is treated in the same way; this leads to the coding of many illegal moves in specific situations, which are then masked out by setting their probability to zero, reducing the move possibilities to those that are available according to the rules in the actual position. For each square, there are 56 possible queen moves (these moves also code rook, bishop, king and most pawn moves), one to seven squares in all eight directions in which the queen can move. Additionally there are 8 knight moves and 9 possibilities for pawns to promote in a piece different from a queen, either by moving to the last rank or by capturing a piece on the last rank in one of the two diagonal directions. This adds up to an input of 8x8x73 equalling 4672 move possibilities in all positions. Additional inputs are needed to represent information about special rules: the castling rights, the number of repetitions of the present position and the move count without progress with respect to the 50 moves rule (Silver et al. 2018).

The output of the neural network is a vector with two numbers for each move. The first one signifies the percentage of search time, which was attributed to this move and its consequences. The second one represents the estimated win probability that the neural network assigns to this move. This means that the neural network has the function of the evaluation function in a classical chess engine. It is trained to assign a probability to each of the possible moves, which represents the win probability when making this specific move. The metric of evaluation is one of the most obvious differences to classical chess engines. This evaluation via probabilities can be interpreted as integrating two aspects, which are not well-represented in Stockfish's evaluation function: The general complexity of the position and the potential risks that result from choosing a specific move. Essentially, this way of representing pays tribute to the fact that the computer cannot calculate all possibly relevant lines and the resulting uncertainty. This will be larger in complex positions with more moves, which cannot be ruled out quickly[5].

[3] I cannot provide a description of reinforcement learning or machine learning in general in this paper. Goodfellow et al. (2016) provide a good introduction.

[4] In 2018 Leela Chess Zero, an open-source program with the same approach has been released. By now it has reached a level comparable to Stockfish.

[5] Two behaviours that can be observed in AlphaZero's play can be directly related to this modelling choice. 1) In positions with a large advantage, AlphaZero might simplify the position even on the cost of some of the advantage. While Stockfish would always search for the option that promises the largest advantage, there is no incentive for AlphaZero to win quickly or to achieve unnecessarily large material advantages. This might sometimes lead to moves that seem counter-intuitive for humans such as unnecessary underpromotions. 2) In positions, which AlphaZero evaluates as negative, it might try to complicate the position, making moves that still have a higher degree of uncertainty instead of a well-analysed move, which is evaluated with a very low success probability (Sadler & Regan 2019).

The training of the neural network has been done entirely by self-play over a timespan of nine hours. At the beginning of the training phase, all the parameters in the neural network were initialized with random values, leading to seemingly random moves by AlphaZero. Most of these games ended in draws because of the 50-move rule[6], but some of them were decisive and based on these games, the parameters in the neural network were adjusted. Repeating this process for nine hours and 44 million games, the millions of parameters in the neural network were adjusted repeatedly. Via the tuning of these parameters, AlphaZero can represent patterns and strategies. After each decisive game, the parameters in the network are updated to evaluate each of the positions of the game as better or worse depending on the outcome of the game (Sadler & Regan 2019).

We can conclude that there are two major differences in the architectures of AlphaZero and Stockfish. The evaluation of positions is approached with different metrics. Stockfish evaluates positions in pawn equivalents; AlphaZero assigns win probabilities to positions. The most important difference lies in how the evaluations are reached. Stockfish uses handcrafted domain-specific knowledge in its evaluation function; AlphaZero reaches the evaluation of the position through its neural network, which has been trained by self-play and without the input of any domain-specific knowledge except for the rules of the game.

4 Epistemic opacity

> *"Whilst we cannot understand exactly how AlphaZero is thinking, we can explore the ways in which AlphaZero generates its innovative and active plans, and how it conducts its ferocious attacks through analysing its games."* (Sadler & Regan 2019, 74)

It is certainly not appropriate to associate the calculations that lead AlphaZero to make a decision with thinking. If interpreted a bit more metaphorical, this quote gives a first answer to one of the main questions of this paper: What can we learn about how Alpha-Zero works and how is it different to what we can learn about Stockfish or similar other chess engines?

Since AlphaZero managed to beat Stockfish, it would be very interesting and potentially beneficial for our understanding of the game of chess, to learn on what AlphaZero bases its decisions. It seems to be a reasonable assumption that it must have something encoded about the game of chess that goes beyond the handcrafted domain-specific features, which have been encoded in Stockfish.

The thesis that I want to defend here is that the introduction of a deep neural network and machine learning led to a different kind of epistemic opacity in AlphaZero than the one in Stockfish. This new kind of epistemic opacity prevents us from confidently identifying on what AlphaZero bases its decisions.

[6] A game of chess ends in a draw, if for 50 moves in a row, no pawn is moved and no piece is captured.

4.1 Defining epistemic opacity

At first, it seems counter-intuitive that either Stockfish or AlphaZero can be epistemically opaque in any way. Both are based on algorithms, which are determined processes. For each of the calculations that are done in any of the chess engines, it is always clearly defined which rule or calculation has to be applied next. In principle, all of these calculations could be done by pen and paper or printed out.

Nevertheless, in the philosophical debate epistemic opacity is discussed very prominently. Especially in the context of software that is based on machine learning techniques as in the case of AlphaZero, their potential black box-nature is a widely discussed topic. The introduction of such methods into the scientific practice raises questions about the aims of science and necessity of explanations or understanding.

Let us take the most commonly used definition of epistemic opacity by Paul Humphrey as a starting point to understand what is typically meant by epistemic opacity and why it is useful discussing it in the context of computer models and machine learning.

> *"[A] process is epistemically opaque relative to a cognitive agent X at time t just in case X does not know at t all of the epistemically relevant elements of the process. A process is essentially epistemically opaque to X if and only if it is impossible, given the nature of X, for X to know all of the epistemically relevant elements of the process."* (Humphreys 2011, 139)

One of the potential sources of opacity is the sheer amount of calculations that are done by Stockfish or AlphaZero during a game of chess, which lead to the evaluations of positions and the decision for a specific move. Stockfish calculates up to 60 million positions per second, so it is quite clear that no human cognitive agent will ever go through all of the calculations that are made during one entire game for example. Even though AlphaZero calculates less positions per second (Silver et al. 2018), the amount of calculations still exceeds everything a human agent would be able to go through in a reasonable time. Regarding AlphaZero, one can additionally argue that the steps during the training phase are epistemically relevant elements of the process, since they determine the final values of the parameters in the neural network and thereby have a significant impact on the process. This adds a large number of calculations to the epistemically relevant elements[7]. It is therefore clear that the basic kind of epistemic opacity that Humphreys defines in the first sentence of his definition is present in both Stockfish and AlphaZero.

Are these opacities essential in the sense of Humphreys' definition? To argue for this, one has to show that it would be impossible for the cognitive agent to go through all of these calculations because of their nature. Let us take a human as the cognitive agent

[7]There is a debate about what should count as epistemically relevant. Durán (2018) for example points to the fact that a limited amount of information about an algorithm might be enough for the justification of results. In this paper, I follow the more widespread approach to count all parts of the process as epistemically relevant to it. The reduction of epistemically relevant elements requires background information. There is not enough of it available yet in this case, therefore using the wide approach seems to be more appropriate.

X. He is limited through his life span and if we assume a discrete amount of time that is needed for each of the calculations, we can determine a maximum number of calculations that one human could possibly do and thereby determine a threshold for this opacity to become essential. This approach is however not very informative about the nature of the source of opacity. It seems somewhat arbitrary that there should be some number of calculations x, which are necessary for a process, for which the opacity is not essential; if at the same time, adding just one more calculation would make the opacity essential.

As has been argued in Boge & Grünke (2020), there might be another useful differentiation between types of opacities, namely "contingent epistemic opacity" and "fundamental epistemic opacity". Both epistemic opacity and essential epistemic opacity, as defined by Humphreys, are contingent on the nature of the cognitive agent. Fundamental epistemic opacity on the other hand refers to opacities which are grounded in the nature of the process rather than the nature of the agent:

> *"[A process is fundamentally epistemically opaque, if] given any agent with any nature, at no time will the agent be able to obtain all relevant pieces of information about the process." (*Boge & Grünke 2020, 9)

A thought experiment shows that the opacity, which is caused by the amount of calculations, is contingent. Consider the Laplacean demon, understood as an entity with unlimited cognitive resources, which would be able to perform as many calculations as necessary without using up any time (de Laplace 1814). For it, the algorithmic transparency of the processes would be enough to render them epistemically transparent (Boge & Grünke 2020).

With respect to this source of opacity, AlphaZero has introduced no different kind of opacity.

4.2 Model-opacity as fundamental opacity

The definition by Humphreys shapes the concept of epistemic opacity as something that is connected to processes. This makes it natural to look for sources of opacities, which may arise in the process of coming to the results via a string of calculations or the training phase of the neural network.

Sadler & Regan (2019) focus on the question "how AlphaZero is thinking" (74), but in order to learn the new things, which AlphaZero might have discovered about chess that make it more successful than traditional chess engines, we need to understand *what* AlphaZero is thinking (thinking must be understood metaphorically again, of course). All of AlphaZero's decisions are based on the neural network that has been trained. By tuning the parameters of the neural network, AlphaZero represents features of the positions, similarly to the way features of positions are represented in the handcrafted evaluation function of Stockfish. In the case of AlphaZero however, it is not clear which features are actually represented in the neural network, since they are developed during the training phase and not preselected through the programming. It is possible that AlphaZero learns features, which have been chosen for Stockfish, such as safety of the king or pawn structure. It could also be the case that AlphaZero learns features, which are a lot more complex and far re-

moved from the human way of describing concepts in chess. This uncertainty about what features are modelled in the neural network may constitute a different kind of opacity. It is not process-based but rather concerns the connection between the neural network and the real-world phenomenon, so it seems useful to call it *model-opacity*[8].

One natural approach of trying to overcome part of this opacity is a classical experimental one[9]. You come up with a hypothesis about the behaviour of the system and collect empirical evidence for or against it. The main source for empirical evidence in the case of AlphaZero are the games that have been published. Sadler & Regan (2019) analyse the games and try to identify features in AlphaZero's play. They show examples of well-known strategies, which have been adopted by AlphaZero, as well as positions in which AlphaZero does not follow any of the classical plans developed by humans and programmed into Stockfish[10]. However, the complexity of chess makes it impossible to isolate any specific feature in almost all cases. Each position can be described by many different features and their interactions. At the same time, these features cannot be deduced directly from the rules of chess but are human descriptions of patterns that have been recognized. The number of ways to describe features of a chess position is infinite and many might be functionally nearly equivalent but differ in some very specific ways. Therefore, in addition to the problem that we cannot isolate specific features, it is impossible to be sure that we have conceived all of the potential hypothesises. Even unlimited resources would not solve this problem. There is no algorithm that could solve this problem and therefore there is no algorithmic transparency, which would render this opacity contingent. Model-opacity therefore seems to be fundamental (cf. Boge & Grünke 2020 for a similar case).

4.3 Discussion of the opacities

The contingent opacity caused by the amount of calculation steps is similar for both Stockfish and AlphaZero. Even though there are differences in the concrete amount of calculations and AlphaZero has the training phase of the neural network, which influences the results and does not have a complementary part in the Stockfish architecture, the nature of these opacities is the same. All of these calculations are clearly defined and determined processes that can be solved if enough resources are available.

As described above, this is not the case for model-opacity. However, model-opacity does not exist for Stockfish. It does not exist because the modelling for Stockfish was intentional and goal-orientated. The modellers can be asked about the reasons and intentions for the implementation of specific features. This does not mean that Stockfish never makes moves which come unexpected to the modellers. No one who modelled parts of Stock-

[8] Model-opacity is discussed in Boge & Grünke (2020) for an example from high-energy physics. Sullivan (2019) uses the same term for the "complexity and black box nature of a model" (1). She introduces the term "link uncertainty" to describe the situation, that there is not enough empirical or scientific evidence to prove the connection between the model and a real-world phenomenon.

[9] Another approach is to find positions in which engines obviously misevaluate and try to derive information about the system from that (similar to adversarial networks in image classification). There are a number of constructions of positions with enormous material advantage for one player in which a human can quickly see that there is no way to achieve any progress but the computer nevertheless evaluates the position as very advantageous.

[10] The minority attack in the Carlsbad structure is used as a prominent example

fish has a complete comprehension of the interaction of all parts of the program and can calculate all the consequences of these interactions and therefore no one can make a confident prediction. Stockfish outperforms all human players in the game of chess after all. This is parallel to AlphaZero but can be explained with contingent opacity. What is possible when Stockfish makes an unexpected move is to analyse the reasons how the decision to make this specific move came about. By reviewing which evaluations led to the move and which features were evaluated in what way, a reconstruction of the decision process is possible. This is exactly what seems to have happened in the match between Deep Blue and Garry Kasparov. During the first game of the match, a bug in the code led to an unexpected move by Deep Blue. After the game, the programmers of Deep Blue could track down the reason why this move was chosen and fixed the bug (Anderson 2017). If this would happen with AlphaZero, the programmers would not know how to change the code of AlphaZero. There is no way to predict what the consequence of changing a single weight in the neural network would be for example. There is no way of confidently knowing about the way specific features are represented in AlphaZero's neural network and consequently they cannot be manually altered with a specific goal in mind.

This opacity of AlphaZero is of a different kind than the contingent ones and it does not only prevent humans from intentionally altering the program in an advantageous way but also prevents them from isolating and understanding the way in which AlphaZero represents chess, and which features of the game it chose in its neural network.

5 Conclusion

In this paper, I described the development of artificial intelligence in chess with its present-day culmination: the introduction of the neural network-based AlphaZero in 2017. I highlighted its differences from Stockfish, one of the highest rated traditional chess engines. The two most significant differences are the modelling process and the metric of evaluation, most notably the creation of the neural network of AlphaZero without any domain-specific knowledge about chess except for its rules.

The different modelling processes lead to different epistemic opacities. Both Stockfish and AlphaZero are epistemically opaque in a contingent way due to the very large number of calculations necessary during the training phase of AlphaZero as well as during the play phase for both Stockfish and AlphaZero. AlphaZero additionally also has model-opacity, which seems to be a fundamental kind of opacity. This opacity originates from the method that is used for creating the neural network: machine learning or, more specifically, reinforcement learning. Contrary to the modelling process of Stockfish, the modelling is not done intentionally by a human programmer, but through reinforcement learning – a statistical approach. This seems to make it impossible to connect the "reasoning" of AlphaZero to human reasoning in chess.

This fundamental opacity is not problematic in chess. Human chess players can still follow AlphaZero's games, use it as an inspiration and develop ideas and concepts from it, which they can integrate in their own games. For applications in domains other than games however, it is potentially ethically very problematic if human agents base their decisions on the output of a neural network with fundamental opacity.

Acknowledgements

The research for this paper was partially conducted as part of the research unit The Epistemology of the Large Hadron Collider, funded by the German Research Foundation (DFG; grant FOR 2063). I want to thank the participants of the Budapest Workshop on Philosophy of Technology 2019 for a very helpful discussion on the topics of this paper. I would also like to thank an anonymous reviewer for helpful suggestions on improvements of the original manuscript.

References

Anderson, Mark Robert. "Twenty years on from Deep Blue vs Kasparov: how a chess match started the big data revolution." Accessed May 12, 2020. http://theconversation.com/twenty-years-on-from-deep-blue-vs-kasparov-how-a-chess-match-started-the-big-data-revolution-76882

Beal, D Beal, Don F. "A generalised quiescence search algorithm." Artificial Intelligence 43, no. 1 (1990): 85-98.

Boge, Florian, and Paul Grünke. "Computer Simulations, Machine Learning and the Lapla cean Demon: Opacity in the Case of High Energy Physics", forthcoming in Resch, Kaminski, and Gehring (Eds.), *The Science and Art of Simulation II*, Springer (expected 2020).

Chabris, Christopher. "The Surprising Return of Odds Chess." Accessed May 12, 2020. https://www.wsj.com/articles/the-surprising-return-of-odds-chess-1461339115

Chessentials. "History Of Chess Computer Engines." Accessed May 12, 2020. https://chessentials. com/history-of-chess-computer-engines/

de Laplace, P. S. *A Philosophical Essay on Probabilities*. London: Chapman & Hall, Ltd. Translated from the 6th french edition by Frederick Wilson Truscott and Frederick Lincoln Emory, 1902 [1814].

Durán, J. M. 2018. *Computer Simulations in Science and Engineering: Concepts—Practices— Perspectives*. Cham: Springer Nature.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Humphreys, P. "Computational science and its effects". In Carrier, M. and Nordmann, A., editors, *Science in the Context of Application*, 131–142. Dordrecht, Heidelberg: Springer, 2011.

Morton, Ella. "The Mechanical Chess Player That Unsettled the World." Accessed May 12, 2020. https://slate.com/human-interest/2015/08/the-turk-a-chess-playing-robot-was-a-hoax-that-started-an-early-conversation-about-ai.html

Sadler, Matthew, and Natasha Regan. 2019. *Game Changer*. New in Chess.

Samuel, Arthur L. "Some studies in machine learning using the game of checkers." *IBM Journal of research and development* 3, no. 3 (1959): 210-229.

Schwartz, Oskar. "Untold History of AI: When Charles Babbage Played Chess With the Original Mechanical Turk." Accessed May 12, 2020. https://spectrum.ieee.org/tech-talk/tech-history/dawn-of-electronics/untold-history-of-ai-charles-babbage-and-the-turk

Shannon, Claude E. "XXII. Programming a computer for playing chess." *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41, no. 314 (1950): 256-275.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362, no. 6419 (2018): 1140-1144.

Standage, Tom. 2003. *The Turk: The Life and Times of the Famous Eighteenth-Century Chess-Playing Machine*. New York: Berkley Trade.

Sullivan, Emily. "Machine Learning and Understanding". forthcoming in: *British Journal for the Philosophy of Science* (2019). Available at: http://philsci-archive.pitt.edu/16276/

Thicknesse, Philip. 1784. *The Speaking Figure, and the Automaton Chess-player, exposed and detected*. London: Stockdale. 8vo. pp. 20.