Journal of Machine Learning Research 01 (2021) 1-8 Submitted 12/10

Reinforcement Learning for Chess AI Agents

Connor Raymond Stewart CRSTEWART@UWATERLOO.CA
Cheriton School of Computer Science
University of Waterloo
200 University Ave W
Waterloo, ON N2L3G1, Canada

Abstract

The various research projects have attempted to develop a chess program that can autonomously play a game of chess at high levels with minimum human intervention. Systems have been developed using many different methods, including more traditional methods like Alpha-beta Pruning to Reinforcement Learning and Deep Learning. This article serves as a literature survey for the various methods used for chess AI agents, and it pays close attention to the topic of chess AI systems for reinforcement learning agents. The article describes the differences between chess AI optimization implementations and the different strengths and weaknesses of different AI approaches. Approaches to reinforcement learning systems and comparisons between reinforcement learning systems and other systems are made. Future research in machine learning, reinforcement learning, and deep learning are explored and elaborated. New directions for AI systems and state-of-the-art features associated with the surveyed systems are outlined in the report herein. The article herein is to inform and elaborate on the various developments in chess AI systems with a focus centred on reinforcement learning so that readers may build a more robust understanding of and more significant appreciation for the various developments and future directions which can be taken in the fields of cognitive artificial intelligence, reinforcement learning, and computational strategy systems.

Keywords: AlphaZero, Stockfish, Reinforcement Learning, Machine Learning, Artificial Intelligence, Alpha-beta Pruning, Optimization, Temporal Difference, Markov, Monte Carlo, Q-Learning, TDLead, Neural Networks, Deep Learning, Cognitive artificial Intelligence, Data Science.

1 Introduction

The problem being selected is a literature survey on reinforcement learning in chess AI. Topics including various AI models, their methodology, effectiveness, and what we can learn from them are discussed.

Through the study of chess AI, computer scientists can gain insight into logical and mathematical strategies in AI agents over long periods. Chess is a classic turn-based strategy game where players take consecutive turns moving black and white pieces on an asymmetrical board. When the black and white players in chess take turns, chess is an inherently balanced game where both players have equal odds of success. Learning more about chess strategy and determining ways to use reinforcement learning agents to optimize chess strategies can help AI developers learn new theoretical models for reinforcement learning approaches to real-world problems.

Chess has always been thought of as a game of intelligence; perhaps this is because there are many different possible combinations of moves and positions in chess. It has been calculated that there are as many as 10^{43} different board positions, and it has been estimated that the number of board positions is 10^{120}; however, only 10^{20} critical positions in chess need to be known and are positionally balanced (Shannon, 1950). Much insight can be gained by teaching a computer to play chess efficiently, as it proves there are ways reinforcement learning agents can optimize strategic decision-making in the face of vast permutations of choices.

Real-world applications for this type of AI can be seen everywhere. Domains such as traffic, management logistics, self-driving vehicles, and manufacturing could benefit from reinforcement learning agents, optimizing positional placement in turn-based systems. Chess can be used as a general scaffolding for discrete turn-based systems where an AI agent can use reinforcement learning methods to optimize strategy. Chess itself is a two-player game, so more direct applications of studying chess-based reinforcement learning can include implementations in turn-based strategy games.

Playing chess is an excellent simulation exercise for an AI developer. The process of learning chess with human players allows for the buildup of complex mental models with heuristics, openings, strategies, and theories. Letting an AI agent process chess-based information to learn these strategies is a vital step forward in AI development. By learning to play chess and optimize behaviour to maximize victories, a chess AI can learn a complex set of rules to win at a simple game like chess. The enormous number of moves and permutations of games in chess gives an AI agent much room to develop a neural network that allows for a nearly perfect chess strategy. Games like chess are an excellent example for reinforcement learning, as the rules are straightforward, they can be programmed into a learning model, and the model can train itself over time based on the ground rules upon which chess is built. The concept of letting a reinforcement agent learn by optimizing behaviour based on a predefined set of rules can be extended to the real world in terms of systems where rules are predefined, as would appear in inventory management systems. For example, a system could learn to optimize a finite number of constraints to create an optimal strategy to organize an inventory system. Of course, the process of learning itself is the most relevant reason to study reinforcement learning for chess AI since we – as AI developers – get to build mental models and heuristics of our own to create artificially intelligent systems in higher-scope applications.

2 Survey

Multiple techniques for reinforcement learning in AI systems will be discussed, along with their respective strengths and weaknesses. The properties of the techniques, along with desirable traits, will be summarized in detail and elaborated. Traditional techniques will be described for reference and contrasted with reinforcement learning methods. Examples of techniques themselves include AlphaZero, Stockfish, Upper Confidence Bound on Trees (UCT), VADER, Minimax, Temporal Difference (TD), TDLeaf, gradient descent, exponentiated gradient, DeepChess, CRAFTY, FALCON, KnightCap, NeuroChess, GNUChess, and SAL.

In terms of tournament evaluation, we see that the AlphaZero algorithm outperforms the Stockfish algorithm with a win-draw rate of 25:25 when playing as white and 3:47 when playing as black (Silver *et al.* 2018). However, tournament evaluation is not the only important factor when analyzing AI performance. Commonly, chess rankings are based on a system known as Elo, where players' relative skill levels are calculated and ranked against each other. Results of the DeepChess system show that the DeepChess system has 63.2 points higher Elo than the CRAFTY system (David *et al.* 2016). The DeepChess system also has a 10.4-point higher Elo than the FALCON system when DeepChess is trained for thirty minutes and has a 96.2-point higher Elo when trained for one hundred and twenty minutes (David *et al.* 2016). VADER vs VADER chess games allowed the neural network to progressively learn the game of chess from scratch (Chaturvedi & Pradhan, 2020). The VADER model was able to learn chess at a rate that let it gain several thousand years of chess knowledge and strategy within a few weeks, and VADER was able to develop new chess strategies to beat other AI models (Chaturvedi & Pradhan, 2020). VADER learned faster than a genetic algorithm and efficiently demonstrated a clear understanding of chess, even though it only started with the basic rules of chess built-in (Chaturvedi & Pradhan, 2020). Therefore, VADER shows that a reinforcement learning agent can learn to play a classic chess game and divide its strategies unsupervised. Within precisely 308 games, the KnightCap engine was a 1650-rated Elo player to a 2150-rated Elo player (Baxter & Weaver, 2000). Based on testing samples from Internet Chess Club (ICC) test trials, Levinson & Weber's model shows that similar models take months to achieve within a few days of training results (Levinson & Weber, 2001). The Levison & Weber system achieves this by creating an internal representation for the chess game, by assigning appropriate evaluations in the random of zero to one inclusive onboard positions, and by updating the internally represented weights of global optimization functions once all the positions in a game have been assigned (Levinson & Weber, 2001). The final point is achieved using internal interconnected nodes with associated weights in a 2-layer regression network to determine the loss between the estimated prediction and the actual result (Levinson & Weber, 2001). The KnightCap learning algorithm can be extended by using a more extensive board state database and adjusting optimizing coefficients for each learning and playing skill reached after only a few trained games (Block-Berlitz et al., 2008). One of the main disadvantages of AlphaGo – note at this point, AlphaGo could only play Go – was that it required a database of 30 million human moves to train the model to play games like people; they have the machine play against itself to optimize itself better (Scharre et al., 2018). Such a large dataset was overcome by the invention of AlphaGo Zero, a computer system that taught itself to play games entirely through self-play without requiring human examples (Scharre et al., 2018). The AlphaZero system – AlphaZero is a generalization of AlphaGo Zero for games other then go – achieved superhuman performance within four hours and learned by only knowing the fundamental rules of chess itself (Scharre et al., 2018).

The program SAL first tested temporal difference learning by Michael Gherrity (Block-Berlitz *et al.*, 2008). SAL allows for the realization of move generations for different games and determines the next move using search-tree-based algorithmic approaches to learn good versus wrong moves from previous games (Block-Berlitz *et al.*, 2008). The SAL system uses temporal difference learning to optimize network parameters by comparing evaluations to rooted nodes in the search tree (Block-Berlitz *et al.*, 2008). NeuroChess uses a neural network - as discussed above - to evaluate a TD method based on root nodes to modify coefficients, allowing NeuroChess to learn from itself (Block-Berlitz *et al.*, 2008). Both SAL and NeuroChess are tested against the prominent GNUChess program as a benchmark (Block-Berlitz *et al.*, 2008). Relating to previous topics above, the KnightCap chess program is extended by using a more extensive and more complete board state database and adjusted optimization coefficients for each position class (Block-Berlitz *et al.*, 2008). The optimization allows for a higher level of play than the original KnightCap algorithm, and the speed by which the model is trained is far faster than traditional learning methods (Block-Berlitz *et al.*, 2008).

DeepChess is four times slower than FALCON, likely because DeepChess is a non-linear evaluator, whereas FALCON is based on a genetic algorithm (David *et al.* 2016). Complexity and data structure are essential in chess AI since it determines the rate by which a machine learning model can learn, and therefore its extendibility to other applications. The VADER algorithm could quickly achieve faster learning rates and surpass human intelligence levels within a few weeks using reinforcement learning (Chaturvedi & Pradhan, 2020). Accuracy rates for the VADER method increase with the dropout rate, whereas supervised learning algorithms barely experience accuracy increases (Chaturvedi & Pradhan, 2020). KnightCap is roughly ten times slower than CRAFTY, which is more significant than DeepChess (Baxter & Weaver, 2000). KnightCap is also 6000 times slower than Deep Blue, another populator training method (Baxter & Weaver, 2000). A model proposed by Block et al. is based on an extension of the KnightCap algorithm, and it can achieve a 2000-point Elo level play in only 72 games (Block-Berlitz *et al.*, 2008).

Neural network-based reinforcement learning in computer chess is a notable strategy for optimization AI agents. The NeuroChess engine is trained by temporal-difference learning to predict a game outcome (Silver *et al.* 2018). The KnightCap engine evaluated position by a neural network that used an attack table (Silver *et al.* 2018). The Meep engine evaluated position by a linear evaluation function trained under a temporal-difference learning variant known as TreeStrap (Silver *et al.* 2018). The Giraffe evaluated positions by a neural network, including mobility/attack-and-defend maps, and is trained by a TD(leaf) self-play (Silver *et al.* 2018). Finally, DeepChess is a trained neural network that performs pair-wise evaluations of positions and is trained using supervised learning (Silver *et al.* 2018).

In contrast, trees can determine chess AI moves instead of neural networks. For example, the Upper Confidence Bound on Trees (UCT) starts with an empty game tree and runs several simulations (Goldwaser & Thielscher). Nodes on the upper confidence bound on Trees stores a count calculated by finding the argmax value, which is defined as a present formula (Goldwaser & Thielscher). The UCT method has been shown to outperform Q-Learning with Monte Carlo Search methods. However, extensions to the AlphaZero algorithm with GGP environments outperform the UCT benchmarks (Goldwaser & Thielscher). VADER is a reinforcement learning algorithm that can beat traditional AI methods using convolutional neural networks, which later becomes a self-learning AI (Chaturvedi & Pradhan, 2020). In the Minimax search strategy. The chess engine considers the adversary's best move and utilizes savage power strategies to create a decision tree consisting of player moves and adversaries' best moves (Chaturvedi & Pradhan, 2020). An Alpha Beta Technique is another classical chess AI system that attempts to decrease the number of nodes evaluated by a minimax algorithm since there are many chess move permutations (Chaturvedi & Pradhan, 2020). The Principal Variation Search (PVS), Quiescent Searching method, Null Move Pruning process, and Monte Carlo Search consolidate the tree and remove excess moves from the agent's consideration (Chaturvedi & Pradhan, 2020). These systems speed up processing time so that the AI agent can calculate a move in time.

Multiple search technique approaches are discussed and strengths and weaknesses are covered. For example, product propagation – a classic minimax method – can be used as a baseline comparison, and the candidate moves method implementation in Minimax search procedures provides optimal moves but is slow (Chole & Gadicha, 2020). Minimax systems tend to form a classic example of a chess AI which can be used as a simple benchmark. Testing shows that the Alpha-beta methods are faster than Minimax but cannot search an entire tree and are imperfect (Chole & Gadicha, 2020). Alpha-beta pruning is a deeper search which sacrifices some nodes in a decision tree, further limiting move accuracy for the sake of speed and depth (Chole & Gadicha, 2020). Principle variation search methods allow faster alpha-beta techniques but provide more cut-offs (Chole & Gadicha, 2020). Secondly, transpositions tables require iterated searches and can be unstable (Chole & Gadicha, 2020). Lastly, quiescence searches allow the evaluation to return stable values but give unstable predictions (Chole & Gadicha, 2020). Null move pruning reduces the cost of search but provides a failure in cut-off leads to waste computational efforts (Chole & Gadicha, 2020). Monte Carlo Tree searches provide robust approaches for complex AI problems at the expense of creating search dynamics that are not fully understood by the AI-agent processing the next move (Chole & Gadicha, 2020).

Interestingly, deep learning methods can solve problems with higher efficiency but require a vast amount of gameplay data which can be limited without large databases (Chole & Gadicha, 2020). Finally, AZ-style algorithms can train AI agents for chess with few parameters and accurate evaluations at the expense of costly computation expenses used for training (Chole & Gadicha, 2020). Implementing a neural network allows for a system to train from grandmaster level play to find optimal chess movement strategies (Chole & Gadicha, 2020). The TDLeaf method is a variation of the TD algorithm used in the KnightCap chess program, allowing temporal difference learning (TD) in conjunction with game-tree searches (Baxter & Weaver, 2000). An exponentiated gradient can outperform gradient descent in games with less than thirty training runs, as can be seen with the results of a 2-ply search with initially random weights (Levinson & Weber, 2001).

Temporal difference learning can be used to attempt to maximize the expected long-term future reward when training the chess algorithm (Baxter & Weaver, 2000). As an extension of a deep minimax search discussed above, TDLeaf learning can be incorporated into a chess program – through the KnightCap engine – to calculate positional features at the cost of speed (Baxter & Weaver, 2000). Gradient descent learning agents with 3-tuples universally outperform 2-tuples (Levinson & Weber). A good model with relevant features is necessary to accelerate learning and decrease the importance of search methods, as can be seen with games where 3-tuple agents are tested against random agents with a greater search depth (Levinson & Weber, 2001).

3 Analysis

Significant advances in machine learning and reinforcement learning have been made in the past couple of years, and it is expected that many more applications for reinforcement learning will be made in the future. Primarily, we can see that the Stockfish – which runs using alpha-beta pruning – is weaker than AlphaZero implementations which use reinforcement learning. AlphaZero – which can learn playing against itself – uses an even more generic implementation than AlphaGo, which learns based on previous games played by human players. The fact that AlphaZero can beat the best chess AI engines globally without outside training shows that there are many ways to devise chess strategies. AlphaZero can learn how to handle a chess game by itself using only its codebase and the base rules for chess itself and can optimize itself via a deep neural network to create a system to handle chess perfectly. We can infer from the above that chess algorithms are moving away from classical AI algorithmic implementations, which rely on the tree, data structures, and pruning, and are instead moving towards machine learning models based on mathematical optimization strategies which allow computers to build precise coefficients to learn to play chess. It was initially thought that chess would be too complex for a machine learning agent to learn to play a good game of chess and that there were potentially too many possible permutations of chess movements for an optimization agent to handle. However, reinforcement learning approaches have worked since chess has a predefined set of moves and rules and a predefined victory state. Since we already know the starting state for a chess game and understand what the completion state should look like (the side the agent is playing as is supposed to checkmate the opponent's king), a neural network can optimize its connectivity to approximate a set of behaviours to learn to play chess. The approach taken by AlphaZero is interesting since it does not rely on approximation like alpha-beta pruning – which is inexact due to the vast number of chess move permutations – but instead relies on reinforcement learning with can optimize towards a perfect game of chess down to an infinitesimally slim margin.

In AlphaGo, value functions and policies are incorporated into the model and are learned from initial samples from human expert data and later improved through self-play (Schmid *et al.*, 2021). A deep network approximates the value function, and a prior policy helps to guide the selection of actions during a tree search (Schmid *et al.*, 2021). AlphaGo Zero later removed the initial training from human data, and AlphaZero then reached state-of-the-art performance in chess using minimal domain knowledge (Schmid *et al.*, 2021).

Deep neural networks have seen wide applications in many AI applications today and serve as state-of-the-art image recognition and pattern recognition systems (Scharre *et al.*, 2018). Part of the appeal of a chess engine generated by neural networks like AlphaZero is that it can devise its strategies independently of human thought and judgement about the game of chess. Instead, then building on top of centuries of chess strategy, theory, and motifs, AlphaZero was able to create its strategies and motifs which augment – or in some cases, diverge from – centuries of classical chess thought. There is an open opportunity to develop future strategy engines using AlphaZero and other reinforcement learning agents. By extension to chess, any turn-based strategy game with definable *tiles* (the black and white tiles in chess are an example of a tile on a board) can be determined by AlphaZero. Games like Go, Checkers, or Elmo and real-life applications for inventory management systems in autonomous robotic vehicles could be explored. Autonomous robotic vehicles could drive around a room with floor spaces partitioned into dedicated tiles; an AlphaZero-like program could determine optimal strategies to coordinate efforts and competition between two opposing sets of vehicles.

Interestingly, human hand-crafted strategies are no longer essential; instead of using heuristics and motifs based on human-designed strategy, a higher-level abstract concept behind strategy and tactics, in general, has appeared. Human heuristics and motifs are not very useful if a reinforcement learning agent can build past several millennia's worth of human knowledge within a couple of hours, starting from a clean slate. Instead, determining the underlying process by which humans gain insight into strategy is more insightful than simply writing strategies; all it takes is a single algorithmic design, data structure, reinforcement learning agent, machine learning model, artificial intelligence paradigm, etcetera to ultimately surpass all human knowledge within a given domain (as we have seen with chess) within hours, from scratch. Therefore, learning more methods to apply the underlying patterns humans learn to solve problems (machine learning, data science, and reinforcement learning are all examples) becomes more important than human knowledge of the problems themselves. Learning about the process of learning opens a new angle that AI developers will need to explore in the future.

Other optimization methods are commonly used throughout the literature explored. Examples would include gradient descent methods, the exponentiated gradient, linear regression, and value function approximation. Optimization methods are used for calculating reward values associated with peace movements, loss values associated with piece transfers and positional realignments, and heuristic values for chess pieces throughout the game. Across different journals, publications, and implementations, exact optimization methods change even when using the same algorithm or method. For example, methods used with AlphaZero change with different publications, and chess engines can have varying implementations. Therefore, the exactness of different chess evaluations can change by author and publication. Furthermore, different CPU, GPU, API, OS, or TPU systems can change the precision of the calculations for the optimization methods and neural network systems. Thus, different authors with differing setups can obtain different results, meaning multiple publications by multiple authors across a range of different testing systems would be required to determine the effectiveness of one chess engine over another.

One key takeaway from the articles is that there are various chess engine implementations and design strategies. Algorithmic designs like alpha-beta pruning and reinforcement learning methods can build highly advanced chess engine systems. Optimization methods from data science, linear regression, probability, mathematical statistics, operations research, and game theory can all be used to determine parameters and constraints for the chess engine models. There are many different directions someone could take to optimize parameters, and many different mathematical systems have been proven at various points to optimize the parameters presented under the chess engine models.

Some open problems remain in chess despite recent advances in chess AI. Overall, chess is still not a solved game in the strictest sense. So far, there is no such thing as a perfect game in chess as far as anyone is aware, and no chess engine can succeed one hundred percent of the time if chess were to be solved, then all games would either end in a draw between white in black, a victory for white or – improbable – a victory for black. However, no chess engine has achieved perfect results in chess, meaning there is still room for improvement in reinforcement learning models in the future. There is a chance that a reinforcement learning model will approach a point of chess match optimization where it can consistently achieve a victory or draw state in a chess match against an opponent agent (including when playing against itself). If a chess engine appears to solve chess, it would still take time to prove that no new angles to chess are available that have not been explored. A mathematician would need to either prove mathematically that the AI agent has resolved all possible chess strategies, or a computer scientist would need to run a chess agent against itself to test all possible chess games worth considering. Theoretically, it has not been proven – to the best of my knowledge – that chess has a forced victory for white or a forced draw. Work in chess theory could be made to prove that chess is a solvable game that results in a forced victory or draw, or if chess runs until the fifty-move limit without piece capture or the stalemate condition is triggered.

4 Conclusion

Surveys of the literature reviewed have allowed insight into the plethora of different AI systems that can be created to play chess. Reinforcement learning systems have the advantage of learning from a clean slate knowing only the baseline rules of chess and can train against itself to find an optimal chess gameplay strategy. Hand-crafted systems like alpha-beta pruning can still be effective as there is a degree of transparency behind how the AI agent generated its move, unlike machine learning systems where alpha-beta pruning is not a black box. Transparency allows easy debugging and parameter optimization, whereas black box systems require in-depth theoretical knowledge of the system and its mathematical/computational properties. Multiple different AI systems and frameworks were reviewed, including bandit algorithms (like the upper confidence bound), reinforcement learning systems (like AlphaZero), search algorithm methods (like Stockfish's alpha-beta pruning), and temporal difference learning systems (like SAL). Different systems reveal that there are multiple angles to approach the topic of AI from in chess strategy and that many different angles could be used, combined, and optimized for future implementations.

Another interesting topic was the variety of different implementations for specific models. For example, there are many ways to implement reinforcement learning, TD-learning, or Q-learning, and the number of possible implementations is staggering. More so, there are an insane number of different GPU, TPU, CPU, Operating Systems, APIs, and programming languages someone could use to run the implementations.

Different system setups and model implementations can have vastly different results for AI agent implementations and dramatically alter runtime performance, training time, and game success rates. There are many different ways to rank engine performance for chess, with Elo being the primary system to rank player skill. Other training speeds, accuracy, error, and computational complexity methods differ vastly, and different implementations have different results. A standardized system or nomenclature for AI systems and implementations could be developed to simplify implementing and organizing these different approaches. A naming system and categorization scheme would be beneficial for building, designing, and implementing various chess AI schemes.

Reinforcement learning seems to be the best approach for chess gameplay when compared to the other methods surveyed. Systems like the AlphaZero implementation seem to function better than other systems and require a shorter training time. AlphaZero does not require prior human knowledge to learn to play chess, which opens up possibilities for future AI agent implementations and opens up philosophical and pragmatic questions and concerns about the extendibility of AI to our current world. It was seen that AI reinforcement learning systems could devise new strategies that humans did not learn or know of for the thousands of years chess has existed.

More research on extending reinforcement learning agents to other turn-based two-player environments could be done using the AlphaZero implementation. Applications in inventory management, robots, and autonomous vehicle transportation could be made. Reinforcement systems are lightweight and can produce new information (like chess matches) for other systems to analyze or process. Many different AI systems exist for games like chess, and a standardized framework for nomenclature and categorization could be developed for future reference. Finally, reinforcement learning agents could be applied to more challenging turn-based strategy games and used for video game AI opponents, which opens up the possibility of creating new video game genres since a self-learning AI brings new potential gameplay mechanics that do not already exist.

Future research can be done on chess AI to find a better AI model in the future, perhaps more advanced designs in deep learning can be explored as was discussed earlier in the survey, and perhaps deep learning methods can remove the AlphaZero model from its current prominence. A future angle that deep learning research could take is to combine itself with reinforcement learning approaches like AlphaZero to obtain testing databases. One of the primary advantages of deep learning is that it can solve problems with greater efficiency than other models, but the main disadvantage of deep learning is that it requires a vast amount of gameplay data that can be hard to obtain without access to large datasets. A program like AlphaZero could play games against itself and provide deep learning model data. The AlphaZero system would input the results of games into databases while training itself and provide the data to the deep learning system to allow the deep learning system to train itself using AlphaZero's games. The above would potentially mitigate one of the main drawbacks of deep learning since it would provide infinite datasets to learn from but potentially increase the training stage's time complexity even more since a second chess engine is being used.

References

Adrian Goldwaser and Michael Thielscher. Deep Reinforcement Learning for General Game Playing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1701-1708, 2020.

Ankur Chaturvedi and Rahul Pradhan. Deep Learning model for Chess Game Player. *International Journal of Advanced Science and Technology*, 29(03):5288-5298, 2020.

Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256–275, 1950.

David Silver, Thomas Hubert,Julian Schrittwieser,Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot,Laurent Sifre, Dharshan Kumaran, Thore Graepel,Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *SCIENCE*, 362(6419):1140-1144, 2018.

Eli Omid David, Nathan S. Netanyahu, and Lior Wolf. DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess. *International Conference on Artificial Neural Networks (ICANN)*, 9887:88-96, 2016.

Henk Mannen. *LEARNING TO PLAY CHESS USING REINFORCEMENT LEARNING WITH DATABASE GAMES*. Masters Thesis,UTRECHT UNIVERSITY, Utrecht, 2003.

Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Learning to Play Chess Using Temporal Differences. *Kluwer Academic Publishers*, 40(3):243-263, 2000.

Marco Block, Maro Bader, Ernesto Tapia, Marte Ramírez, Ketill Gunnarsson, Erik Cuevas, Daniel Zaldivar, and Raúl Rojas. Using Reinforcement Learning in Chess Engines. *Research in Computing Science*, 35:31-40, 2008.

Paul Scharre, Michael C. Horowitz, and Robert O. Work. What is Artificial Intelligence? *Center for a New American Security*, 2018.

Robert Levinson and Ryan Weber. Chess neighborhoods, Function Combination, and reinforcement learning. Technical Report 133-150, Computers and Games, Second International Conference, Hamamatsu, Japan, 2001.

Vikrant Chole and Vijay Gadicha. A Review towards human intuition based chess playing system using AI & ML. *Turkish Journal of Computer and Mathematics Education*, 11(2):792-797, 2020.

Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, Elnaz Davoodi, Alden Christianson, Michael Bowling. Player of Games. *DeepMind*, 2021.