Introduction

Since its inception, the field of artificial intelligence (AI) has sought to measure the intelligence of machines by comparing their performance against that of humans. In Alan Turing's seminal paper [1], he suggested that a good way to evaluate machine intelligence is to have machines play games against human adversaries. Intuitively, the argument is that if greater intelligence causes one to be better at winning games, then for a machine to win a game against a human, it must have greater intelligence (at least at playing that game). This conclusion seems especially valid for two-player, zero-sum games, where one player wins if and only if the other loses. A great deal of AI research has thus focused on building AI systems that can play two-player, zero-sum games at a high skill level.

AI approaches to game-playing can largely be grouped into two categories: classical techniques (also known as Good Old-Fashioned AI, or GOFAI), and connectionism. Classical techniques tend to focus on explicitly representing abstract knowledge about the game in some sort of knowledge base, and then efficiently searching through that knowledge when it's time to select the next move. In connectionist approaches, the system's knowledge is represented *implicitly* within one or more neural networks, and these networks process the game state as input before outputting an evaluation of the available moves. Each approach has its advantages, and each has led to super-human performance at some games. In this paper, we will look at some major milestones where AI systems first beat humans at checkers, chess, and backgammon; and we will use these systems to discuss some of the strengths and weaknesses of each approach. Then we will look at recent work on playing Go that combines deep neural networks with classical AI methods to achieve super-human performance where neither approach had previously been able to on its own.

Chess & Checkers

The concept of a game-playing machine is at least as old as the late 18th century, when Wolfgang von Kempelen tried to convince people that his chess "automaton" was more than just a person hiding in a box [2]. However, it wasn't until nearly two hundred years later, with the birth of computer science and artificial intelligence, that substantial progress started to be made towards a more legitimate version. In 1950, Claude Shannon introduced several important strategies for distilling games into computation, with a focus on chess in particular [3]. He formally defined an *evaluation function*, which represents whether or not a given board position can be won, and he proposed several strategies for learning approximate versions of such functions. Shannon also explicitly described the optimal strategy for a two-player, zero-sum game: players select actions that alternate between maximizing and minimizing the evaluation function of the resulting position. Shannon pointed out that, in chess, it was infeasible for a computer to search for such an optimal strategy using brute-force, since that would require looking at approximately 10^{120} possible games [3]. Instead, he proposed using heuristics (such as the number and location of pieces) to guide the search towards "good" strategies.

Subsequent research into AI game-playing built on Shannon's foundations, and adapted these general principles to other games. Arthur Samuel chose to focus on the game of checkers instead of

chess, since it was comparatively much simpler (approximately 10^{31} possible games [4]), while at the same time still featuring many of the aspects of chess that made it such an interesting problem [5]. Samuel explicitly represented the problem as a search tree, where nodes corresponded to board positions, and the levels of the tree alternated between maximizing and minimizing the evaluation function, as in Shannon's formulation [3]. Samuel's program exploited this tree structure with two different types of learning, which he called "rote learning" and "learning by generalization" [5]. The former memorized board positions that the program had previously encountered, along with their corresponding evaluations and frequencies, in order to greatly increase the depth of the search. The latter involved a procedure for modifying the weights of the evaluation function, both to imitate expert games, and to improve performance when the program played against itself. A later version of Samuel's program [6] added a technique called "alpha-beta pruning," which further exploited the structure of the search tree by ignoring certain subtrees based on the assumption that the opponent would play optimally.

After Samuel's program achieved some modest success against amateur human players, research attention largely shifted from checkers to other games. In 1965, Richard Bellman proposed using dynamic programming to build endgame databases, where the search tree was built up *backward* from completed games [7]. In the late 1960s, the Greenblatt chess program [8] introduced the idea of using hash tables to avoid repeated searching of identical positions. In chess (and many other games), it is possible to reach a given board position via multiple distinct sequences of moves or *transpositions*. Normally, each copy of the position would need to be searched independently, but with hashing, the results of the first search can be saved in a *transposition table* for later use. Greenblatt also added a table of expert "book openings" to aid the program during the beginning of the game [8]. A few years after Greenblatt's work, Alfred Zobrist proposed an improved method for building transposition tables [9]. Zobrist hashing used XOR operations of piece locations to create a hashing scheme that could recognize all possible board positions while avoiding hash collisions between them. These improvements and others gradually improved the performance of chess- and checkers-playing programs. However, despite all of this progress on game-playing machines, humans retained their world-champion titles in both checkers and chess into the mid-1990s, at which point, they abruptly lost both.

Chinook & Deep Blue

The first computer program to become a world champion in chess or checkers was the Chinook checkers program [10]. In 1994, Chinook played six games (drawing each time) against the World Checkers Champion, Marion Tinsley, before Tinsley was forced to withdraw from the match due to health problems [10]. Chinook subsequently defended its title a year later, against number-two player Don Lafferty. Meanwhile, a team at IBM was preparing their chess engine, Deep Blue [11], for a match against the World Chess Champion, Gary Kasparov. By the end of 1997, Deep Blue had claimed that title for itself as well [12].

The Chinook system drew mainly on techniques employed by previous chess and checkers programs. It combined tree search with alpha-beta pruning and heuristics to guide the search towards more useful lines; it used a hand-tuned evaluation function that incorporated a variety of human-designed features; it contained an extensive endgame database for all positions involving eight pieces or fewer;

and it featured an opening book that was carefully constructed to improve Chinook's early-game performance against Tinsley [10]. Apart from incremental improvements to these methods, two main innovations set Chinook apart from its predecessors. First, Chinook ran on substantially improved hardware that enabled significantly faster and deeper parallel search [10]. Second, the team added pre-computed "tactical tables", which extended the search in certain key positions and helped correct an error where Chinook would sometimes give up two pieces in exchange for one [10]. According to the team, Chinook succeeded where other programs had failed, because it "searched better and deeper, evaluated positions better, had access to more and better-quality endgame databases, and had access to 12 times as much (and better-quality) opening knowledge" [10].

Likewise, there were several aspects of Deep Blue that enabled it to outperform previous chess engines. First, it ran on dedicated, special-purpose hardware, designed specifically for playing chess [12]. The Deep Blue hardware enabled a massively parallel, non-uniform search that could dynamically increase search depth for important positions [12]. Additionally, Deep Blue incorporated vast amounts of domain knowledge. Its non-uniform search was supported by an incredibly complex evaluation function consisting of approximately 8,000 human-designed features, where many of the weights were tuned by hand [12]. Deep Blue also used an opening book of 4,000 positions (manually created), a large endgame database for all positions involving five pieces or fewer, and an "extended book", compiled from 700,000 expert games, which was used as a heuristic to guide the search [12].

Chinook's and Deep Blue's victories represented important milestones for the field, and decisive victories for Classical AI. Their super-human performance was the result of decades of refinement and iteration on the same core idea: represent game knowledge explicitly, and then search through that knowledge as efficiently as possible to decide on the next move.

Legacy and Limitations

Building on their victory in 1995, Chinook's authors shifted their focus to *solving* the game of checkers, that is, achieving perfect play. In 2007, they formally proved that if both sides play optimally, the game is a draw [13]. Their result used Chinook's forward search with alpha-beta pruning, plus an improved backward search involving an even larger 10-piece endgame database, as well as a second type of forward search called proof-number search [4]. Proof-number search starts from the initial board position and assigns each subsequent position a *proof number*, a measure of how many more positions need to be considered in order to solve the entire tree. The algorithm then repeatedly checks the board position with the smallest proof number until the tree is solved [4]. By proving that checkers is a draw, the authors effectively secured Chinook's title as the permanent world champion — it literally cannot lose.

Meanwhile, in the world of chess, the techniques behind Deep Blue inspired an array of competing chess engines. All of the current top-rated engines are far more powerful than Deep Blue, and run on a small fraction of the hardware. The current best engine, Stockfish 9 [14], has an Elo rating of about 3450 [15]. By comparison, Magnus Carlsen, the top human player, has an Elo rating of 2843 [16]. This difference translates to approximately a 97% win probability for Stockfish (with nearly all of the remaining probability being a draw) [17].

While it's safe to say at this point that computers have far surpassed human ability in both chess and checkers, it's worth pointing out some limitations of Classical AI techniques. First, while the Chinook team was able to solve checkers, their approach is hard to scale to larger problems. Chess, with its much larger game tree, remains far from solved [13]. Second, and perhaps more importantly, the chess engines that are descended from Deep Blue rely on the same GOFAI approach: human-designed evaluation functions, endgame tables, and parallel, heuristic-driven, forward search with pruning. These methods, while incredibly effective for chess and checkers, must be completely redesigned for other games. The version of Deep Blue that defeated Kasparov was a supercomputer capable of considering 100-200 million positions per second [12], and yet, tragically, all it could do was play chess.

Backgammon

Unlike chess and checkers, which are completely deterministic games, backgammon involves an element of chance. Every turn, players roll dice to determine which moves they can make, and then they attempt to choose the best move available. This randomness makes backgammon a more difficult game to analyze. Despite backgammon only having about as many board positions as checkers, the dice make it so that the branching factor at each position is approximately 400 (versus about 35 in chess and 8 in checkers), which means deep forward search is computationally intractible [18, 19]. Moreover, backgammon cannot be solved in the same sense that checkers was solved: even the best backgammon strategies can only win in expectation. There's always some chance that the dice will favor the opponent. It may come as a surprise then, that backgammon was actually the *first* game in which a world champion was defeated by a computer program.

One of the earliest explorations into computer backgammon was Hans Berliner's BKG program [18], the same program that would go on to defeat the world champion, Luigi Villa, in 1980 [19]. Berliner quickly recognized that forward search in backgammon would be difficult, and so, in true GOFAI fashion, he focused his attention on developing a powerful evaluation function and extensive endgame tables. To achieve good performance, Berliner found that the evaluation function needed to be able to represent non-linear relationships between its various features [18, 19]. However, manually adjusting the parameters of such a complex evaluation function turned out to be difficult. To simplify the problem, Berliner partitioned the set of board positions in to several smaller "state-classes", and then linearly combined the features within each state-class [18]. This effectively added a crude form of overall non-linearity while still being feasible to tune by hand. However, in some board positions, the state-classes introduced strange edge-case behavior, since different possible moves could land in distinct state-classes [18]. To handle these situations, BKG approximated the expected value of each move by combining the evaluation functions for the different state-classes, and then it would select whichever move had the highest expected value [18].

While minor improvements to these techniques were enough for BKG to beat Villa 7-1 [19], the victory was less meaningful than it seemed. Although BKG technically won the match, this was in large part due to chance, rather than greater skill. According to Berliner, "The analysis confirm[ed] that Villa, as expected, [was] certainly the better player" [19]. Berliner went on to say, "A match to 7 points is not considered very conclusive in backgammon. A good intermediate player would

probably have a 1/3 chance of winning such a match against a world-class player" [19]. From a historical perspective, the win was significant, but from a statistical perspective, it was not.

TD-Gammon

Achieving a statistically significant victory over a world champion proved more difficult. While Berliner and others continued to apply classical techniques, a different line of research was achieving success with connectionist approaches. By 1992, Gerald Tesauro's TD-Gammon [20] was nearly matched in skill with top human players. The program had previously played a series of matches against human experts (including four world champions), where it performed at an advanced level for more than 80 games [20]. Then, in a 40-game match against former World Champion Bill Robertie, TD-Gammon lost by just a single point [20]. Although TD-Gammon did not technically win a championship, its skill in these games represented a significant improvement over BKG [20].

There were two fundamental breakthroughs that allowed Tesauro's program to compete with top human players. The first was the backpropagation learning rule [21], which allowed for efficient training of multi-layer neural networks. These larger neural networks could represent the complex, non-linear evaluation functions needed for backgammon. By enabling the evaluation function to be tuned *automatically*, rather than by hand, backpropagation effectively solved the parameter adjustment problem that Berliner had observed with BKG.

The second breakthrough was a reinforcement learning technique called "Temporal Difference" learning [22] (or TD). TD measures differences (or errors) in the predictions of a model (e.g. a neural network) from one timestep to the next, and then updates the parameters of the model to make the first prediction more like the second. This type of update is a natural fit for backgammon, where deep search is intractible, but where searching one or two steps ahead is perfectly reasonable. TD-Gammon used a variation of TD learning called TD(λ) [22], which is a generalization that takes multiple previous predictions into account and scales their contribution to the update based on recency. The λ parameter controls the relative importance of short-term TD-errors versus long-term TD-errors.

TD-Gammon combined these two methods with Monte-Carlo "rollouts" in order to train the network [20]. In each rollout, the program would play a single game to completion, keeping track of the moves it had made. After the game was over, the program would modify the network weights using TD(λ) and backpropagation. Subsequent games used different random seeds, so that over a large number of games, the network could learn to approximate the expected value of each move [20].

One major difference that separated TD-Gammon from previous game-playing programs was that its neural networks could be trained entirely from self-play. Given only the raw board position, the program could automatically discover features that were good enough to compete against previous programs [20]. And while the version that competed against humans did incorporate a few expert features that summarized important details about the board position, TD-Gammon did not use any opening books, endgame databases, or even expert games [20]. It was essentially self-taught.

Legacy and Limitations

Since the branching factor in backgammon is so large, TD-Gammon could only look two moves ahead at each board position. This meant that it couldn't quite match the experts on tactical play [20]. However, due to the strength of its learned evaluation function, the program's positional knowledge of the game was often superior to human masters [20]. After the matches in 1992, professional players noticed that in some positions, TD-Gammon would select a move that challenged the conventional human wisdom [20]. In several cases, detailed Monte-Carlo rollout analysis confirmed that TD-Gammon's move choice was in fact better than the previously accepted "best" human strategy, and professionals changed how they played as a result [20].

TD-Gammon was able to improve on decades of human study largely because it was self-taught. The only expert knowledge it received, apart from the rules of the game, was a set of board features—statistics, essentially—about things like the number and location of pieces [20]. The program then learned how best to combine these features (along with raw board positions) and determine the relative importance of each [20]. Because the program never saw opening books, endgame databases, or expert games, it was effectively free from human bias. It played strategies not because they were popular, but because they were actually the best in a statistical sense.

The success of TD-Gammon inspired a number of modern backgammon programs (such as GNU Backgammon [23]) that all essentially share the same basic architecture: neural network evaluation functions trained by Monte-Carlo rollouts and TD, combined with a shallow forward search to enhance tactical play. Despite substantial improvements in computing hardware, backgammon programs are still computationally limited to approximately 4-move forward search. As a result, this architecture is fairly specialized for backgammon—other board games generally require much deeper forward search. While the core techniques in TD-Gammon can certainly be applied elsewhere, other games demand additional strategies to achieve world-class skill.

Go

The game of Go is immensely complex. It is played on a 19×19 board, which means the branching factor rivals that of backgammon, and the game tree is twice as deep as chess. With roughly 10^{360} possible board positions [24], Go is in a class of its own. Expert play requires a careful mix of short-term tactics and long-term strategy, and an almost *qualitative* analysis of board positions [24]. When Chinook, TD-Gammon, and Deep Blue were all winning against world champions, Go programs could not even win against human beginners [25]. In 2014, the best Go programs, using all the techniques described above (and more), had still only reached the level of an intermediate amateur [24]. True mastery of the game was thought to be beyond the reach of computer programs for at least another decade [24].

Then, in 2015, Google DeepMind made history when their program AlphaGo [24] beat European champion Fan Hui 5-0. Six months later, AlphaGo won 4-1 against Lee Sedol (winner of 18 international titles), and the following year, it defeated the World Go Champion, Ke Jie, 3-0 [24]. Almost overnight, DeepMind's program had outsmarted the best human players at one of the world's most challenging games.

AlphaGo

In many ways, AlphaGo represented a marriage of classical and connectionist techniques. Rapid advances in deep neural networks allowed AlphaGo to learn rich representations of the board position and a powerful evaluation function. At the same time, efficient tree-search techniques, combined with Monte-Carlo rollouts, enabled the program to consistently select good moves, despite the massive size of Go's game tree.

AlphaGo's neural networks looked quite different from the ones in TD-Gammon. First, they were orders of magnitude larger [20, 24]. Recent advances in deep learning had allowed for efficient training of complex networks with significantly more layers than their predecessors [26]. These deeper networks were capable of learning powerful hierarchical representations of their inputs, and, with the advent of cheap, commercial GPUs, they could also learn those representations quickly. The second improvement since TD-Gammon was that AlphaGo's networks were "convolutional". Convolutional neural networks [26] make strong assumptions about how their weights should be spatially related, and these assumptions work well for grid-based games like Go. At each layer, the network learns to process the input in small 2-D patches, and then it applies the same transformation to each patch. Layers of convolutions allow the network to learn small, local features that apply anywhere on the board, as well as more abstract high-level features that describe the overall position. AlphaGo was comprised of multiple convolutional neural networks: a policy network was trained to predict the next move in a set of human expert positions, then trained via self-play to maximize win probability; and a value network was trained to *predict* the win probability of the improved policy network [24].

To cope with Go's large branching factor, AlphaGo needed to rely on Monte-Carlo evaluation similar to what was used in backgammon. However, while a shallow four-move forward search is fine for backgammon, it is not nearly deep enough for the intense tactical play and long-term strategy of Go. Other leading computer Go programs achieved deeper search using an algorithm called Monte-Carlo Tree Search (MCTS) [27]. MCTS is an efficient, heuristic-based, forward search algorithm that is in some ways similar to proof-number search. MCTS reduces computation by evaluating the most potentially useful positions in the game tree first, based on how likely each position is to lead to a win, plus a bonus term that prefers relatively unexplored positions. In AlphaGo, the selected board position is evaluated by running a Monte-Carlo rollout and combining the result with the value network's prediction; then the search tree is updated and the process repeats [24]. Since MCTS hinges on being able to execute these rollouts quickly, AlphaGo also had a much smaller, faster version of the policy network for selecting moves during the rollouts [24].

Looking Forward

At its core, AlphaGo relied on techniques as old as AI itself: heuristic-based forward search, plus a powerful evaluation function. Yet its mastery of Go also came from its ability to represent knowledge implicitly in state-of-the-art deep neural networks. The team at DeepMind had found a way for classical and connectionist techniques to complement one another, producing a system that was much more capable than the sum of its parts.

After AlphaGo's victories against Lee and Ke, DeepMind developed another version of the software, called AlphaGo Zero [28], which simplified the neural network architecture and learned entirely from self-play. Starting from just the rules of the game, AlphaGo Zero was able to defeat the most advanced version of AlphaGo, 100-0 [28]. Then, to show that their approach could generalize, DeepMind subsequently developed a third program, AlphaZero [29], which played Go, chess, and shogi (Japanese chess), all at a level that was better than any previous program.

AlphaGo and its successors represent the state of the art in computer game-playing, and their abilities far exceed those of human experts. Still, it's important to note that—like Deep Blue—AlphaGo is designed for a specific task: playing Go. Even AlphaZero is limited to playing whichever one of the three games it was trained for. Perhaps it's time for AI to go beyond games as a measure of machine intelligence. While there are still some games (e.g. StarCraft [30]) where humans play better than machines, games will always represent an overly simplified environment for testing AI: the rules and the goal are both well-defined and pre-determined. For AI systems to truly be intelligent in the way that humans are, we may need to take them out of their simplified game environments and see how they do in the real world.

References

[1] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236), 1950.

[2] Claude E Shannon. A chess-playing machine. *Scientific American*, 182(2):48–51, 1950.

[3] Claude E Shannon. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[4] Louis Victor Allis et al. *Searching for solutions in games and artificial intelligence*. Rijksuniversiteit Limburg, 1994.

[5] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[6] Arthur L Samuel. Some studies in machine learning using the game of checkers. II - Recent progress. *IBM Journal of research and development*, 11(6):601–617, 1967.

[7] Richard Bellman. On the application of dynamic programing to the determination of optimal play in chess and checkers. *Proceedings of the National Academy of Sciences*, 53(2):244–247, 1965.

[8] Richard D Greenblatt, Donald E Eastlake III, and Stephen D Crocker. The greenblatt chess program. In *Proceedings of the November 14-16, 1967, fall joint computer conference*, pages 801–810. ACM, 1967.

[9] Albert L Zobrist. A new hashing method with application for game playing. *ICCA journal*, 13(2):69–73, 1970.

[10] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. CHINOOK the world man-machine checkers champion. *AI Magazine*, 17(1):21, 1996.

[11] Feng-hsiung Hsu, Murray S Campbell, and A Joseph Hoane Jr. Deep blue system overview. In *Proceedings of the 9th international conference on Supercomputing*, pages 240–244. ACM, 1995.

[12] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep Blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[13] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *science*, 317(5844):1518–1522, 2007.

[14] Stockfish - Open Source Chess Engine. `https://stockfishchess.org/`.

[15] CCRL 40/40 Rating List. `http://www.computerchess.org.uk/ccrl/4040/rating_list_all.html`.

[16] Carlsen, Magnus - FIDE Chess Profile. `https://ratings.fide.com/card.phtml?event=1503014`.

[17] Elo Win Probability Calculator. `https://wismuth.com/elo/calculator.html#rating1=3450&name2=Carlsen%2C+Magnus`.

[18] Hans J Berliner. Experiences in Evaluation with BKG - A Program that Plays Backgammon. In *IJCAI*, volume 5, pages 428–433, 1977.

[19] Hans J Berliner. Backgammon computer program beats world champion. *Artificial Intelligence*, 14(2):205–220, 1980.

[20] Gerald Tesauro. TD-Gammon: A self-teaching backgammon program. In *Applications of Neural Networks*, pages 267–285. Springer, 1995.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[22] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[23] GNU Backgammon. `http://www.gnubg.org/`.

[24] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[25] Jay Burmeister. Computer Go. Technical Report CS-TR-339, Department of Computer Science, The University of Queensland, 1995.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[27] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

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

[29] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv e-prints*, December 2017.

[30] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: a new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.