University of St Andrews

Game Playing: Learning and Self-Play

Student ID 190029087

December 18, 2020

1 Abstract

Alike humans, game playing in artificial intelligence (AI) helps exploit various aspects of state spaces guided by the rules of the game. Such variations in turn offer greater chances of realizing the effectiveness as well as the limitations of algorithms in terms of the game's score and the convergence criteria. This essay sheds light upon the historical account of game playing in AI while scrutinizing the way recent advancements in AI have shaped game playing. Using the specimen of few deterministic and stochastic games, I study the progress that machine learning and deep learning has owed to their development. Further, considering the dynamics of individual games and the evidence from literature, I try to answer if common training strategies such as self-play work equally good for all games or if they demand different analysis.

2 Introduction

Ever since its inception in 1930s, AI has focused on learning tasks that are considered arduous and time-consuming for humans. Like us, AI agents leverage learning to explore and exploit their abstract environments through filtering out of irrelevant information. That said, the abstract nature of game-playing has been among the most sought applications of learning for humans as well as human-like intelligence. Though a greater part of history of AI has prioritized to make game-playing engaging and fun for human players, there are certain type of games that have been of special attraction to the AI community for fulfilling the above goal in some scenarios while outsmarting humans in other. In particular, strategic games with discrete set of actions and well-separated boundary between states (*e.g.,* chess, poker, etc.) have been much advantageous to the applications of AI rather than physical outdoor games such as ice hockey with huge set of possible actions and imprecise rules. The first family of strategic games that garnered the attention of game AI belong to the **deterministic** category in the sense that each discrete action allowed by their rules result in a unique output.

The foundation of strategic game AI can be traced back to 1949 when Claude Shannon proposed the **minimax** strategy for two-player games which involved selecting a move that returned the highest possible score at the end of the game provided the both players played optimally. Formally, if we represent the states of a game for players *{MAX, MIN}* using a tree where the first level represents the move of MAX and each successive level draws on the alternate turns of players, then the minimax value of each node is the *utility* of being in the corresponding state given that the game proceeds optimally thereafter. Once the minimax values of all nodes are known to the agent, it can then simulate the game past any point by merely following the best-case path of moves. While this is a reasonably safe strategy to follow, the minimax search has a major flaw in that it involves finding all possible paths to the end of the game. For games such as chess with branching factor of 35, this demands huge memory costs due to the *combinatorial explosion* of game states after few moves. Inspired by master players who selectively consider only promising moves out of all possible moves, Shannon adapted an idea of plausible move generators to minimax - the typical of which was **alpha-beta pruning** that removed the branches of game tree having no influence on the final decision.

Shortly after, Alan Turing in 1951 introduced his first-ever theoretical chess game, *viz.* TurboChamp, which worked by assigning point values to each possible game state, and selecting the move with the highest average possible point value. The origin of game AI can be traced back to the same year when Arthur Samuel first applied

minimax with *machine learning* for playing checkers [12]. The program introduced **rote-learning** - a form of memorization to store the minimax values of previously evaluated states, that allowed it to beat an amateur player after 8-10 hours of machine-playing time. This marked the dawn of mathematical model building for game AI as Samuel's strategy was soon expanded to the game of chess when IBM developed a fully functional chess playing game in 1957.[1] The applications of machine learning stretched rapidly to other deterministic games - the most popular being the game of Go. The players in Go take turns to place white or black stones on board with the goal of either capturing opponent's stones or strategically surrounding empty spaces to make territorial points - a behaviour that was considered hard for computers to mimic.

Whilst *deterministic* games assume well-defined action-to-output mappings, they fail to account for unpredictable circumstances of real world that can put an agent into previously unseen events. The notion of **stochastic** games in AI reflect such events by including a random element in their implementation [14]. Thus for a stochastic game comprising state space M, an action set S^i for each player $i \in I$, there exists a transition probability P from MXS given by P(A|m,s) which depicts the agent's probability of moving to the next state A given its current state $m \in M$ and the current action profile $s \in S$ (for deterministic games, P(A|m,s) = 1). *Backgammon* is one such renowned game which employs the roll of two dices to determine a player's next move. To reflect this uncertainty, its game-tree of legal moves incorporate chance nodes that depict all possible moves that the player might make on a state - *i.e.*, given two dices, each player has 21 distinct rolls with the six doubles *1-1, 2-2, ..., 6-6* having a probability of 1/36 each while the other 15 distinct rolls having a probability of 1/18 each (see figure 1). The rest of the essay thus talks about the progress of game AI using the examples of Go and Chess as deterministic games, and Backgammon as their stochastic counterpart.

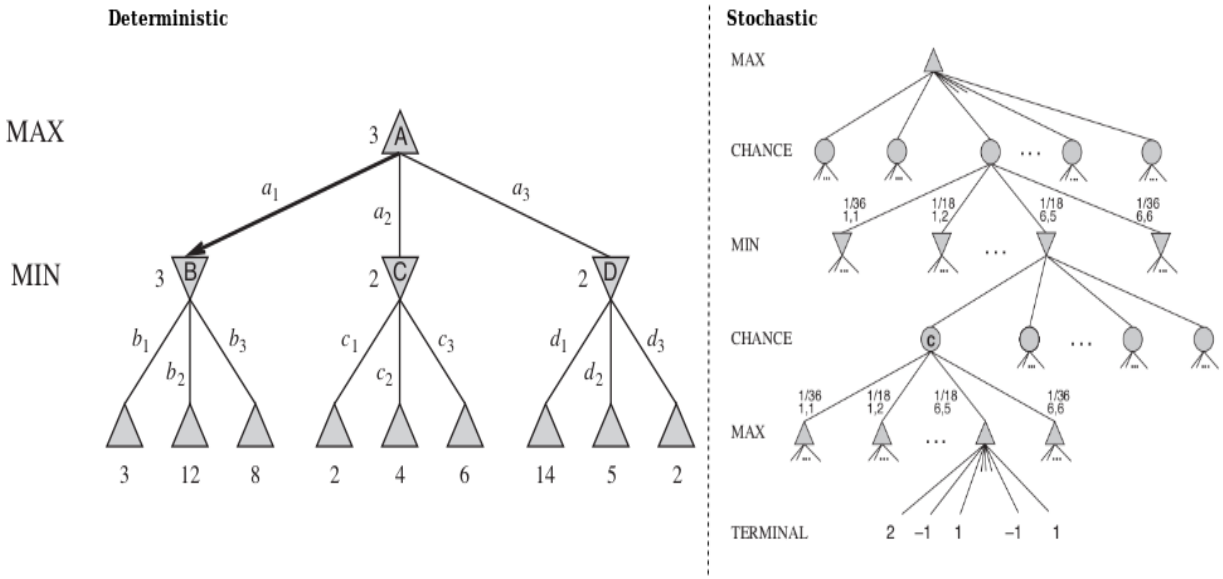

Figure 1. An adaption from (Russell and Norvig, 2016) showing schematic game trees for two-player deterministic (Chess) and stochastic (backgammon) games.

2.1 Self-play games

While game-playing requires the knowledge of the environment, games are special case of AI applications in that the agent has a complete information of the world and does not require any external output for decision making. However, games involving human players impose following limitations:

1. Regardless of the complexity of algorithm being used, the game must be responsive in real-time (most games only allow 16-33 milliseconds to acquire CPU resources for processing the next step)[2].

2. Despite having acquired most sophisticated skills, it is desirable for a game-playing agent to make its moves appear realistic enough to feel like competing a human-like opponent. This fulfills the purpose of games to be entertaining as mentioned in section 2.

[1] Source: https://archive.ph/20120721202324/http://www.chessville.com/BillWall/EarlyComputerChessPrograms.htm
[2] https://www.gamedev.net/articles/programming/artificial-intelligence/the-total-beginners-guide-to-game-ai-r4942/

The restrictions are what fueled the momentum of *self-playing games* in the field of game AI. Relaxing the constraint of human supervision, self-play benefits from the fact that the agent is no longer limited by human feedback and can instead progress by pure optimization of the game's reward. This can be backed by the fact that unlike the dynamics of our world enforcing most of the strategic constraints we face in our universe, fixed universes such as games can be observed to allow much more efficient exploration if such strategic constraints are to be induced upon an agent by the behavior of another agent that too abides by the rules of the universe.

3 Machine learning in game playing

On account of machine learning (ML) algorithms implicitly possessing the capacity to automate model building by learning from experience, the very first of these were the **opening book learning** strategies [7]. Based loosely on the *rote-learning* technique [13], they mimic the fact that human players learn to play certain positions by heart instead of analyzing them every time. The key idea is to compute the goodness of a move from readily available resources to update the program's self-learned book of moves. The branching factor for chess however, makes open book learning efficient only in opening or endgame play. For mid-game positions when most pieces are still alive, chances of finding similar positions in another game are considerably low. (Slate, 1987) presented a solution to this by **learning from mistakes** in which a program maintained global transposition tables with sets of permanently stored positions that get updated as the program played more games. The program would then selectively search those points in depth near which it made a mistake in the previous games and on resolving, added the correct move to its table [6]. The concept was later used in the iconic match played by IBM's DEEP BLUE against the then world-chess champion, wherein the algorithm employed techniques for extending its book by constructing all positions up to the 30^{th} move from a database of 700,000 games (see section 3.3).

While the above techniques hold good for deterministic games, stochastic games benefit minimally from deep searches and pre-computed transposition tables. Taking this into account, the idea of **simulation search** employs a search technique evaluating positions by playing a multitude of games against itself with varying parameters. This brought a breakthrough for the game of backgammon when in 1991, the former World Champion Bill Robertie played a 31-game match against a TD-gammon game trained using *simulation* and stated that he did well to an average of 0.20 to 0.25 points per game thus making it the strongest backgammon-playing program [10]. The following subsections further delve into key breakthroughs in game-playing AI based on ML algorithms.

3.1 Learning Search Control

A relatively less explored ML technique involves automatically tuning the multitude of parameters (*e.g.*, search depth, move ordering heuristics, etc.) that can increase the efficiency of game-playing programs. (Donninger, 1996) first leveraged this to his chess program NIMZO by assigning a higher priority to automatic optimization of search parameters than to tuning the parameters of NIMZO's evaluation function. The efficiency of alpha-beta search in chess can also be improved by identifying the important regions of a position and then ordering the moves per their influence upon these regions [5]. (Dahl, 1999) further applied similar shape-evaluating networks to the game of Go in order to avoid weak moves.

3.2 Learning of Evaluation Functions

Learning of evaluation function involves assigning relative importance to the pieces of knowledge that reflect the worth of the current board state (*e.g.*, the number, kind and position of black and white pieces on the board for chess). The most uninvolved way for this is **supervised learning** approach which provides the program with example positions for which the exact value of evaluation function is known so that the program can adjust the weights of its parameters to minimize the error of the evaluation function on such positions. However, for the game of chess, the quantitative evaluation of such positions still posit a huge problem - *e.g.*, the degree of advantage that a black bishop possess is ambiguous. An approach to alleviate this involves **comparison training** in which the program is typically fed a set of moves along with an input informing about their relative order. This strategy was employed by the NEUROGAMMON program [19] which became the first game-playing program based entirely on machine learning to win a tournament at the First Computer Olympiad. However, the use of expert games in comparison training has fundamental flaw in that the program eventually imitates human playing style making it less effective in exploring new moves.

The above shortcomings gave birth to the idea of **reinforcement learning** which allows an agent to learn from its own experience by receiving feedback for adjusting its evaluation function. Thus, the learning process can rate the success of its own actions by generally assigning *discounted reward* to actions taken earlier in the game reflecting the assumption that later positions in the game contribute more to the final outcome. However,

reinforcement learning on its own is too slow at evaluating the values of positions that large number of games are required to figure it out.

(Tesauro, 1992) addressed the problem by combining it with **temporal-difference learning** wherein each move of a game was labelled with a position several moves deeper in the game rather than the final outcome of the game. This made the early steps of a game independent of the mistakes made later. When applied to TD-GAMMON alongside the addition of number of self-play games and search depth, it soon reached the first world-championship strength [21].[3] On the other hand, the game of chess would take another six years when the KNIGHTCAP chess program [1] depicted the capabilities of an expert player using the TD-learning strategies.

3.3 Deep learning-driven game playing

As with all other applications of AI, the rise of deep learning introduced major implications in game-playing. The first expert-level implementation to the game of chess can be identified as IBM's DEEP BLUE that defeated the then world chess champion with 3.5-2.5 in a six-game match (see figure 2). The hype of deep learning further took over expectations after 2016 when DeepMind's AlphaGo [15] with no prior domain knowledge and trained using self-play defeated the human European Go champion by 5 games to 0. The program's strength lied in using distinct neural networks to evaluate board positions and to select moves while performing look-ahead search. The event ignited the era of deep reinforcement learning algorithms to game AI by soon being adapted to other games. The eureka moment in chess came when DeepMind's AlphaZero [16] depicted superhuman level performance without using look-up tables. Trained using self-play, the extension of AlphaGo's algorithm defeated StockFish 8 in 155 and drew 839 matches in a 1000-game tournament in 2017.[4] (Silver et al., 2018) further showed the reinforcement learning algorithm of AlphaZero could be generalized to defeat world champion programs in the game of Shogi and Go.

Figure 2. Figure simulating DEEP BLUE's opening white move and the Caro-Kann defense response by Garry Kasparov in the last game of the six-game 1997 chess tournament. DEEP BLUE's answer with a knight sacrifice remained the deciding point leading to its victory within 19 moves. Image source: https://tinyurl.com/rgdo89l .

While backgammon had deep learning applied way back with NEUROGAMMON[19] and TD-GAMMON[20] programs, expert deep learning systems started regaining notice around 2013 when (Molin et al., 2013) showed that the use of bayesian approach to train a neural network outperformed TD-GAMMON. Further, the hype of deep reinforcement learning also tapped backgammon with the emergence of AlphaGo. Consequently, (Finnman and Winberg, 2016) depicted that the use of deep learning agent resulted in superior accuracy than that of Q-tables.

[3] TD-GAMMON was subsequently the first computer game depicting expert-level performance.

[4] Source: https://www.chess.com/news/view/updated-alphazero-crushes-stockfish-in-new-1-000-game-match

4 Self-play: importance and limitations

The importance of self-play in *chess* can be realized from the victory of DeepMind's AlphaZero over Stockfish in 2018. While Stockfish utilized up to 512 CPU threads with look-up capacity of 70,000,000 positions per second, its strategies were continuously modified by open-source developers. AlphaZero, on the other hand, was trained through four hours of self-play and could only evaluate 80,000 positions per second. However, the training motivated it to pick up moves that brought a greater reward towards the end of the game even if not immediately. Self-play has nonetheless been used in the state-of-art programs for several games including Go (AlphaGo Zero) and DOTA 2 (the Open AI Five program[5])- a game hosting much larger and continuous state space than Go and Chess.

For *backgammon*, where an agent can hardly rely on the quality of moves, self-play gives rise to cycles of strategies that restrict the agent's learning ability. Let us suppose that an agent learns a strategy B to beat strategy A and becomes the champion; then a new agent learns a strategy C to beat B in a new game which is again beaten by another agent using A. Such cycles (see figure 3) effectively lead to relative expertise where the agent believed in winning the current match rather than absolute expertise which would be characterized by the agent optimizing its universal knowledge of game. Although this flaw clearly fails self-learning agents in other tasks, the dynamics of backgammon make cycles have relatively minor consequences. In particular, the instability of the game with respect to predictions of winning until all contact is broken and one side has a clear advantage is the greatest factor ensuring the success of agents even with relative expertise.

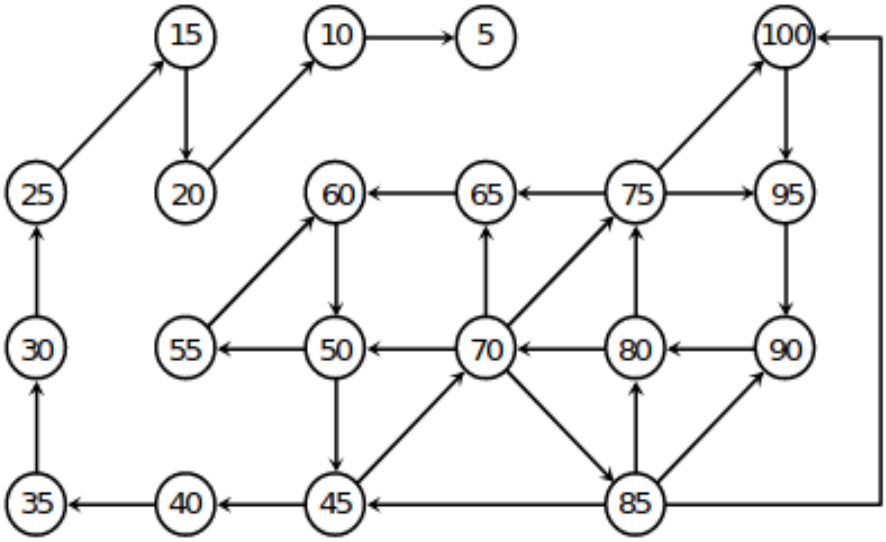

Figure 3. Figure adapted from (Pollack et al., 1997) depicting the partial graph of food-chain in self-play for each 5000th player: arrows denote the dominance relationship after playing 1000 games against each other.

If we view self-play as a meta-game between a teacher and a student with the teacher's objective being correction of student's mistakes while the student trying to appease the teacher by avoiding mistakes, then the meta-game invites the possibility of mediocre stable states where the agent learns to exploit the winning strategies of the game. In particular, if the game permits draws then the agent might learn to draw itself, solve its meta-game equilibrium and stop learning further while if draws are forbidden, the agent might learn to play slightly sub-optimal moves that allow it to throw new games to itself and thus totally avoid the possibility of correction by the teacher. A sample of the above flaw is evident from the case of OpenAI's 1v1 bot[6] which was initially superior to human players but was quickly defeated by pro-gamers as they developed several counter meta-games by identifying its playing style[7]. The defect can be addressed if the agent gets to encounter all different meta-games during the train time so that a single optimal policy is developed to counter them all. Thus, self-play agents today typically learn by playing games against a combination of themselves and their previous versions. But this also limits their only opponent to be their own policy and since their previous policies are bound to be weaker than or on par with the current, the

[5]Source: https://openai.com/five/
[6]https://openai.com/blog/dota-2/
[7]One such counter-strategy was to claim the first tower as mentioned in this subreddit: https://tinyurl.com/ycx55qmp.

self-play strategies of most games do not incorporate enough variance to counter all complex strategies that human players can form against it. A partial solution to this could be training ensembles of agent in parallel and making an agent play against all of them[8] or rather incorporate a fast moving model of meta-games that can allow rapid domain randomization between entirely different human-developed meta-games so that the agent gets to observe an unseen play style much frequently during training.[9]

5 Conclusion

The use of AI has opened doors of tremendous possibilities in game-playing while making it an ideal depiction what the integration of thousands of years of human knowledge and the potential of computers can bring together. Using the examples of chess and backgammon amongst other strategic games, we discuss that while the new advances in machine learning algorithms have cut-off the need for extensive domain knowledge, the use of self-play techniques have further catalyzed the growth of intelligent game agents by giving rise to programs that can acquire super-human skills in a matter of hours. Although the history of game AI focuses largely at improving the experience of human players, the recent breakthroughs in deep reinforcement learning have owed them the potential to generalize and the intellect to beat human experts on a range of games. Based on the dynamics of game universes, we talk about the strength and limitations of self-play and deep learning agents in learning absolute expertise. As Francois Chollet once said - "the most important problem for AI today is abstraction and reasoning", the increasingly demanding game industry indeed presents one of the most exciting challenges for modern AI as they compete to outsmart each other while adhering to human-like reasoning and abstraction. Given the importance of games in development of practical skills and exercise of wisdom among humans, it would be thrilling to observe how the advancements in AI can boost up our grasp of pre-existing games and if this helps them to find a way as integral part of our cultures.

References

Jonathan Baxter, Andrew Tridgell, and Lex Weaver. Knightcap: a chess program that learns by combining td (lambda) with game-tree search. *arXiv preprint cs/9901002*, 1999.

Fredrik A Dahl. Honte, a go-playing program using neural nets. *Machines that learn to play games*, pages 205–223, 1999.

Chrilly Donninger. Che: A graphical language for expressing chess knowledge. *ICGA Journal*, 19(4):234–241, 1996.

Peter Finnman and Max Winberg. Deep reinforcement learning compared with q-table learning applied to backgammon, 2016.

Kieran RC Greer, Piyush C Ojha, and David A Bell. A pattern-oriented approach to move ordering: The chessmaps heuristic. *ICGA Journal*, 22(1):13–21, 1999.

Jia-miene Hsu. A strategic game program that learns from mistakes. *Master's Thesis, Northwestern University, Evanston, Illinois*, 1985.

Robert M Hyatt. Book learning-a methodology to tune an opening book automatically. *ICGA Journal*, 22(1):3–12, 1999.

David Dal Molin, Viking Flyhammar, and Saman Bidgol. Using machine learning to teach a computer to play backgammon. 2013.

Jordan B Pollack, Alan D Blair, and Mark Land. Coevolution of a backgammon player. In *Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems*, pages 92–98. Cambridge, MA: The MIT Press, 1997.

B ROBERTIE. Learning from the machine: Robertie vs. td-gammon, 1993.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[8] Source: https://openai.com/blog/competitive-self-play/#overfitting

[9] Idea from: https://himanshusahni.github.io/2018/09/18/open_ai_five_and_the_limits_of_self_play.html

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.

Arthur L Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. In *Computer Games I*, pages 366–400. Springer, 1988.

Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

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

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.

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

David J Slate. A chess program that uses its transposition table to learn from experience. *ICGA Journal*, 10(2): 59–71, 1987.

Gerald Tesauro. Neurogammon: A neural network backgammon learning program. *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad, Chichester, England*, 1989.

Gerald Tesauro. Temporal difference learning of backgammon strategy. In *Machine Learning Proceedings 1992*, pages 451–457. Elsevier, 1992.

Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.