Name: Frank Gabel
Course: Applied Computer Science
Student number: 3537204
Date: July 10, 2019

Artificial Intelligence for Games: Seminar

Some studies in machine learning using the game of checkers

Arthur L. Samuel (1959)

Frank Gabel

Contents

1 Introduction: History and Rules

This report aims at introducing the reader to the first proceedings of checkers AI, pioneered in Arthur Samuel's seminal paper "Some studies in machine learning using the game of checkers" from 1959 and its successor which was published in 1967 [1, 2].

1.1 History

Checkers is one of the oldest known games played by humanity, with the earliest form of the game found at an archaeological site in ancient Ur, Mesopotomia (nowadays Iraq), dating back to approximately 3000 B.C. [3]. In more recent centuries, different variants were then played in different parts of Eurasia including ancient France, Egypt as well as Britain. Even in Homer's Odyssey, reference is made to games being played in the palace of Ulysses in Ithaca, Greece; also Plato makes frequent mention of the games in his writings.
The modern version of the game was developed in the 12^{th} century when French Philip Mouskat came up with the idea of playing checkers on a chess board. With a new board configuration and new rules set, the game eventually made its way to England and the Americas and finally became known as "checkers". Many years later, checkers additionally became the first game in which a machine beat a human, laying the foundations to what is now known as "machine learning". Interestingly, checkers is even believed to be much older than chess [4]. In the following, the rules of modern checkers will be recapitulated[1].

1.2 Rules

Checkers is a two-player non-cooperative game where opponents sit on opposite sides of the gameboard. One player has the dark pieces; the other has the light pieces. Players alternate turns and may not move an opponent's piece. A move consists of moving a piece diagonally to an adjacent unoccupied square - if the adjacent square contains an opponent's piece and the square immediately beyond it is vacant, the piece may be captured (and removed from the game) by jumping over it.

Only the dark squares of the board are used. A piece may move only diagonally into an unoccupied square. When presented, capturing is mandatory. A player loses the game when either being out of legal moves or out of pieces.

[1] This version is kept short deliberately - for a longer version, the reader is referred to the official rulebook of the *World Checkers Draughts Federation*[5].

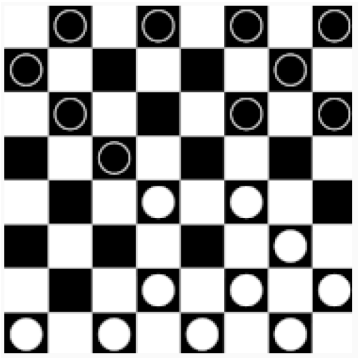

Figure 1: A typical mid-game checkers position.

1.3 The suitability of checkers for early AI

Before turning to Samuel's work marking the cradle of game-playing AIs, it makes sense to talk about how checkers and AI interplay - this is relevant because back in 1950, computing power was sparse, making overly complex games even harder to beat. In that context, checkers is a comparatively easy game:
The number of possible games that can be played (also known as the *game tree size*, the number of leaf nodes in the game tree) is 10^{31} (Chess: 10^{123}, Go: 10^{350}) and the average branching factor (the *branching factor* is the number of legal moves for a given position) is only 2.8 (Chess: 35, Go: 280).
This shows that Checkers possesses a relative simplicity, making it a good candidate for early proof-of-concept AIs.

2 Some studies in machine learning I (1959)

The main contribution of the 1959 paper by Samuel [1] were two methods that used utilized *learning*, creating one of the first competent AI programs: **rote learning** and **learning by generalization**. Before turning to these methods in particular, some prerequisites are conveyed.

2.1 Samuel's general approach to solving checkers

2.1.1 The mini-max and alpha-beta pruning algorithms

The underlying approach to Samuel's checkers program was a Minimax algorithm[2], a recursive algorithm often used in decision-making and game theory. This algorithm generates the entire game search space for a given position (portrayable as a game tree) and returns the move associated with the highest reward irrespective of the opponent's moves.
Minimax can be optimized using a pruning technique called *alpha-beta*, where "impossible" (in the sense that neither player is interested in playing a move of this branch) branches of trees are not considered.

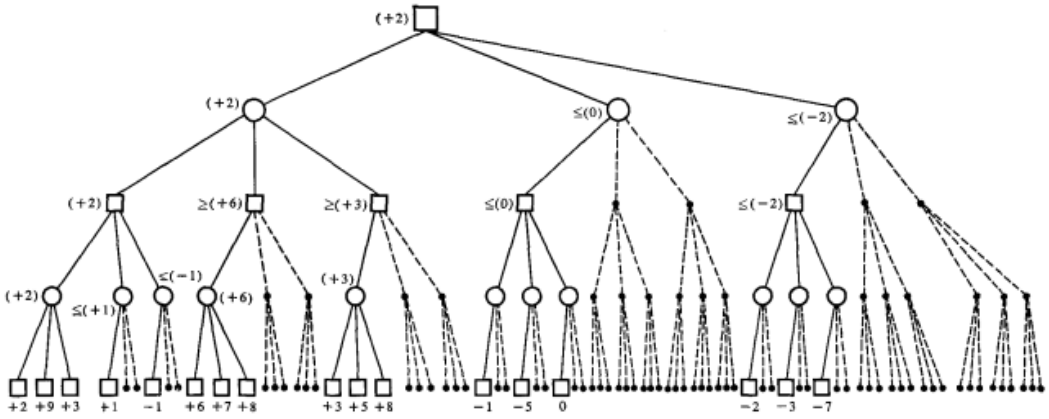

Figure 2: A (look-ahead) move tree in which alpha-beta pruning is fully effective if the tree is explored from left to right. Board positions for a look-ahead move by the first player are shown by squares, while board positions for the second player are shown by circles. The branches shown by dashed lines can be left unexplored without influencing the final move choice at all.

[2]To keep the report somewhat brief, no details of this well-known algorithm are given here. Interested readers can find an excellent introduction in section 5 of [6].

Table 1: A selected list of board parameters (possibly) contributing to the evaluation function. Samuel proposed a total of X terms. The term “passive piece” is very important here - meaning a piece is on the board, but can’t move. An exhaustive list can be found in the appendix of the original paper [1].

feature	description of the corresponding board position
Ratio	Relative piece/kings advantage
Advancement	The parameter is credited with 1 for each passive man in the 5th and 6th rows (counting in passive’s direction) and debited with 1 for each passive man in the 3rd and 4th rows.
Pole	The parameter is credited with 1 for each passive man that is completely surrounded by empty squares.
Apex	The parameter is debited with 1 if there are no kings on the board, if either square 7 or 26 is occupied by an active man, and if neither of these squares is occupied by a passive man.
Center control	The parameter is credited with 1 for each of the following squares: 11, 12, 15, 16, 20, 21, 24 and 25 which is occupied by a passive man.
Back	The parameter is credited with 1 if there are no active kings on the board and if the two bridge squares (1 and 3, or 30 and 32) in the back row are occupied by passive pieces.
...	...

2.1.2 Heuristics

To this day, evaluating non-trivial positions in games like checkers, chess and Go is computationally imfeasible due to the depth and width of the game tree. In order to evaluate these positions and form a decision on which move to chose next, *heuristics* are used. Hereby, each encountered board position gets evaluated according to certain numerical features such as the ones described in table 1 which are then combined to a linear “evaluation function”[3]. Heuristics face an accuracy-effort trade-off where their simplified decision process leads to reduced accuracy [7] (in other words, heuristics face the problem of non-objectivity in their decisions), but they were the only way of evaluating board positions and thus trying to automate decision processes.

In fact, the use of heuristics apes human behaviour in such board games - good players mentally jump from one line of play to another without seeming to complete any one line of reasoning. In doing so, each of these terminating board positions is evaluated according to what the player thinks about the goodness of these particular positions. If a position is deemed disadvantageous, moves leading to this position will be avoided.

[3]While the original paper uses the term “scoring polynomial”, the term evaluation function has been established in literature since and will therefore be used within this report.

2.2 Rote Learning

Rote learning simply consisted of saving a description of each board position encountered during play together with its backed-up heuristic value and the associated next best move (as determined by the minimax procedure). If a position that had already been encountered were to occur again as a terminal position of a search tree, the depth of the search could be amplified as this position's stored value had been cached earlier. One initial problem was that the program was not encouraged to move along the most direct path to a win. Samuel gave it a "a sense of direction" by decreasing a position's value a small amount each time it was backed up a level during a Minimax run. Samuel found this discounting-like technique essential to successful learning. Rote learning produced slow but continuous improvement that was most effective for opening and endgame play. His program became a "better-than-average novice" after learning from many games against itself, a variety of strong human checkers players and from book games in a supervised learning mode[4].

2.3 Generalization learning

While the method of *rote learning* certainly improved the performance of the checkers program, it did not allow for a dynamic improvement during actual competitive play. Samuel's second, and most important contribution was *generalization learning*. Here, the algorithm plays against itself. Black thereby uses the best currently available evaluation function and holds it constant through the game. White, however, starts with the same function and tries to improve it during the game. If White wins, Black is given the improved function and the cycle begins again. If White loses too often (say three times in a row), it is considered to be on the wrong track and given a drastically and arbitrarily changed evaluation function.

[4] Although being conceptually related to useful learning methods such as kNN, this method has received conceptual headwind as it barely entails the *transfer* part which some authors argue is an essential part of "learning" [8].

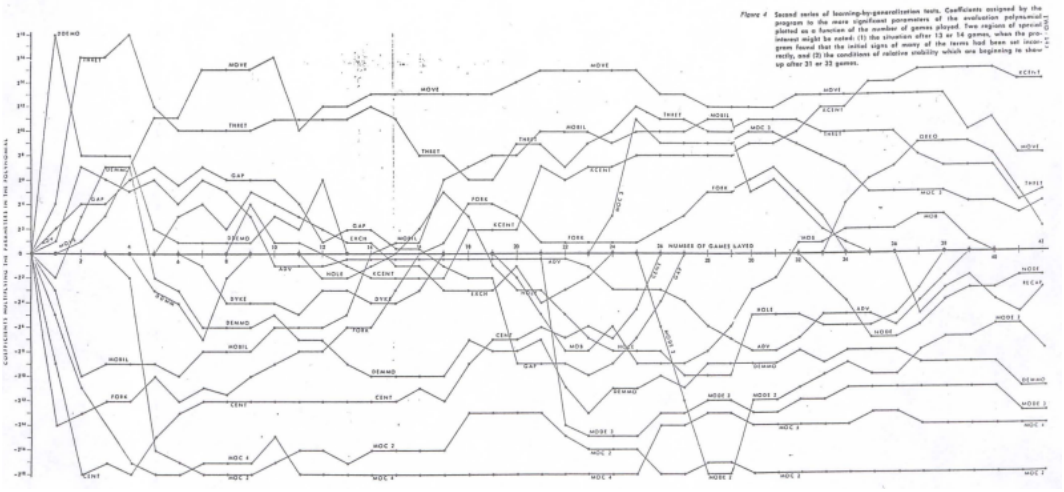

Figure 3: A series of tests using learning-by-generalization. Coefficients of the terms of the evaluation function in use are plotted against the number of games played. Looking closely, one can find terms that vanish and ones that appear while playing.

This way, the weights of an evaluation function are changed towards more optimal setups. Generalization learning also involves a mechanism for replacing terms with little contribution to evaluation because of coefficients being close to zero. A term rejected this way is placed on the bottom of a "reserve list" and may thus expect another "chance" on average 176 moves later.

2.4 Conclusion

Samuel's first paper on his checkers program marks a historical landmark in machine learning. It was actually the first time a computer playing a game won against a human ever[5] and it was one of the first times that the idea of "machine learning" was mentioned, nicely subsumed by a quote from the original paper: "Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort."
Arthur Samuel did not finish working on checkers AI after having written this paper. His proceedings will be the topic of the following chapter.

[5]This human was Arthur Samuel himself, who did, despite intellectual capabilities, consider himself only an average checkers player.

3 Some studies in machine learning II (1967)

3.1 Introduction

Eight years after his first paper on the topic, Arthur Samuel reports on the substantial improvements made to his checkers program since then [2][6]. He points out that the new iteration still entails a MiniMax search algorithm with Alpha-Beta search at its core. He identifies the following weak points in his 1959 algorithm:

- the absence of an effective procedure to generate new parameters for the evaluation procedure
- the incorrectness of the assumption of linearity which underlies the use of a linear polynomial
- the general slowness of the learning procedure
- the inadequacies of the heuristic procedures used to prune and terminate the tree search
- the absence of any strategy considerations for altering the machine mode of play in the light of the tactical situations as they develop during play

He argues that no progress has been made with respect to the first of these defects - however, some progress has been made in overcoming the other four limitations.In the following, the techniques used will be described.

3.2 Extensions to Samuel's earlier algorithm

3.2.1 Generation of new parameters for the evaluation function

The evaluation procedure of the checkers algorithm for minimax is dependent on heuristics to assess the "goodness" of a future position or move - Samuel describes that there are still problems with this approach that he was not able to solve in an adequate manner.

3.2.2 Non-linear evaluation function

Samuel had long been aware that parameters of his original evaluation function do not interact. To replace the simple linear combination of weighted values, he used the tech-

[6] In order to understand this chapter, knowledge of the techniques used in "Some studies in machine learning I" (as outlined in the previous chapter) is assumed.

nique of multilevel *signature tables* (as seen in Figure 4). Hereby, the final evaluation value is found through tree search where root nodes (on the left side of Figure 4) were manifestations of handcrafted feature groups[7] and the leaf node showed the sum of edges corresponding to the features that were "active" at a particular board position. By grouping these features in a contentually meaningful way, interaction effects between these features could be modelled (although in a rather arbitrary way). Learning these signature tables entailed going through book games and increasing two counters for signature table manifestations depending on whether the current evaluation played a move played in the book or not. Finally, evaluation scores were updated based on correlation coefficients between what should have been played and what was actually played. Using this system on 250,000 book games, Samuel found that the evaluation function rated book games as first or second preferences 64 % of the time (and first, second, third and fourth 90 % of the time)[8].

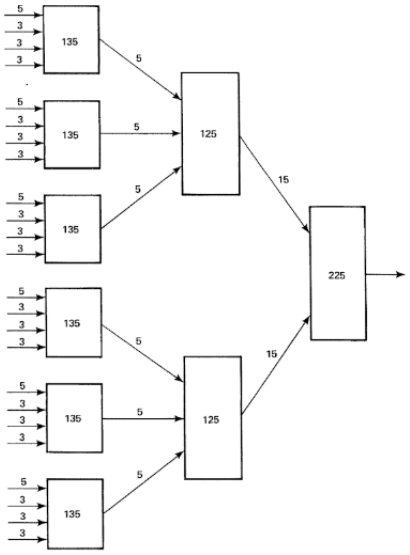

Figure 4: A 3-layer signature table scheme. Numbers on lines indicate numbers of permissible variable values. Values in boxes indicate table sizes. Features according to board positions were "fed in" on the left. The final evaluation score could then be seen on the right.

[7]Due to insufficient computational power, single features could only take three or five distinct values. Feature groups were then characterized as, e.g. O-++-

[8]There is no mention in the paper if these values were calculated on a validation set.

3.2.3 The general slowness of the learning procedure

A large section of the paper is dedicated to improvements (that is, optimizations) of the Alpha-Beta search algorithm. This entailed thresholding possible directions in the search tree to not explore branches with tiny, yet computationally costly improvements as well as more aggressive forward pruning. Also, he implemented ways of first exploring the most promising paths of the search tree by using so-called "plausibility analyses" - i.e. crudely scanning the search tree and sorting the branches by their evaluated goodness.

3.2.4 The absence of any longer-term playing strategy

The chief defect of the program in the recent past, according to several checker masters, seems to have been its failure to maintain any fixed strategy during play. In essence, every move of a game is seen as a completely new problem. Samuel mentions (without having conducted experiments) that signature tables could be used to insert "strategic thinking" into his program by grouping parameters into "signature types" in a manner related to long term goals in the game. Through adjustment of higher levels interactions, adjustment to strategies could then be made.

3.3 Conclusion

Summing up, the improvements made to Samuel's checkers program in the years after his first paper were mostly two-fold: for one, Samuel achieved significant speed-ups by optimizing the tree-search algorithm and performance improvements by learning non-linear evaluation functions using signature tables. In his own conclusion, he states the following "While the goal outlined in the original paper, that of getting the program to generate its own parameters, remains as far in the future as it seemed to be in 1959, we can conclude that techniques are now in hand for dealing with many of the tree pruning and parameter interaction problems which were certainly much less well understood at the time of the earlier paper. Perhaps with these newer tools we may be able to apply machine learning techniques to many problems of economic importance without waiting for the long-sought ultimate solution." [2]
I will use that last statement to bridge to the last section which is an historical interpolation between how machine learning works today and how it worked back then, trying to attribute certain developments to the early work of pioneers.

4 Beyond checkers: the broader context - How did Arthur Samuel contribute to the evolution of machine learning?

Arthur Samuel paved the way for subsequent work on machine learning[9]. His vehicle for this work was the game of checkers which conveniently entailed hundreds of book games that Samuel's algorithm could learn from.
We already talked about the particular constituents and contributions evolving of Arthur Samuel's checker program - essentially working alone, he invented several techniques such as rote learning and generalization learning, using underlying techniques as mutable evaluation functions and signature tables. As these contributions are more than 60 years old, this final chapter now deals with what has evolved from these ideas. The concept of *rote learning* built the basis of what is now known as *memoization*[10], an optimization technique of storing the results of expensive function calls and returning the cached result when the same inputs occur again.
The rationale behind generalization learning is what constitutes today's machine learning - improving mappings without explicit programming, just by using sampled data.
It's interesting to think about whether today's machine learning would work differently without the work of pioneers like Claude Shannon and Arthur Samuel - while this question will remain unanswered, Samuel's (and Claude Shannon's) approaches to learning board games are still in heavy use today - the once (that is, before reinforcement learning started to dominate) strongest chess engine in the world, Stockfish, is using a variant of Alpha-Beta-Search[11] (sped-up by extensive pruning using human chess knowledge).

Samuel did not only influence machine learning. As [9] points out, Samuel's checker work greatly influenced the instruction set of early IBM computers as one of the earliest examples of nonnumeric computation. The logical instructions of these computers were put in at his instigation and were quickly adopted by all computer designers because they are useful for most nonnumeric computation [10].
Arthur Samuel kept working on checkers until the mid 1970s [11], at which point his program achieved sufficient skill to challenge advanced players.

[9] Not least to be seen by the number of citations (2805 as of July 2019), even beating Claude Shannon's "Programming a computer for playing chess" which stands at 1375 at the time of this writing.
[10] Not to be confused with memorization, the process of storing something in memory for recall at a later point.
[11] Actually, readers well-versed in C++ can check the code as it's open source: https://github.com/kobolabs/stockfish/blob/master/search.cpp (where the actual Alpha-Beta search algorithm is implemented in the function **id_loop(Position& pos)** that takes a board position as input).

References

1. A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. *IBM Journal of Research and Development* 3:3, 1959, pp. 210–229.
2. A. Samuel. “Some studies in machine learning using the game of checkers. II—Recent progress”. *Annual Review in Automatic Programming* 6, 1969, pp. 1–36. DOI: 10.1016/0066-4138(69)90004-4. URL: https://doi.org/10.1016/0066-4138(69)90004-4.
3. *The Checkered History of Checkers*. https://www.checkershistory.com/. (Accessed on 06/20/2019).
4. *www.checkerplay.com/strategy/which-came-first-chess-or-checkers/*. http://www.checkerplay.com/strategy/which-came-first-chess-or-checkers/. (Accessed on 06/20/2019).
5. W. C. D. Federation. *Microsoft Word - WCDF Rules of Checkers 2012.doc*. http://www.wcdf.net/rules/rules_of_checkers_english.pdf. (Accessed on 06/20/2019).
6. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, NJ, 1995.
7. G. Gigerenzer. “Why Heuristics Work”. *Perspectives on Psychological Science* 3:1, 2008. PMID: 26158666, pp. 20–29. DOI: 10.1111/j.1745-6916.2008.00058.x. eprint: https://doi.org/10.1111/j.1745-6916.2008.00058.x. URL: https://doi.org/10.1111/j.1745-6916.2008.00058.x.
8. R. E. Mayer. “Rote Versus Meaningful Learning”. *Theory Into Practice* 41:4, 2002, pp. 226–232. DOI: 10.1207/s15430421tip4104_4. eprint: https://doi.org/10.1207/s15430421tip4104_4. URL: https://doi.org/10.1207/s15430421tip4104_4.
9. J. McCarthy and E. A. Feigenbaum. “In Memoriam: Arthur Samuel - Pioneer in Machine Learning.” *AI Magazine* 11:3, 1990, pp. 10–11. URL: http://dblp.uni-trier.de/db/journals/aim/aim11.html#McCarthyF90.
10. J. Lee and J. Lee. *International Biographical Dictionary of Computer Pioneers*. Fitzroy Dearborn, 1995. ISBN: 9781884964473. URL: https://books.google.de/books?id=ocx4Jc12mkgC.
11. J. Schaeffer. *One Jump Ahead - Computer Perfection at Checkers*. 2. Aufl. Springer Science Business Media, Berlin Heidelberg, 2008. ISBN: 978-0-387-76576-1.