A 'Brief' History of Game AI Up To AlphaGo, Part 2

About how we got from the crummy Chess programs of the late 50s all the way to DeepBlue | April 18, 2016

This is the second part of 'A Brief History of Game AI Up to AlphaGo'. Part 1 is here and part 3 is here. In this part, we shall cover just about four decades of progress, from the first victories of computers against people at Checkers and Chess all the way up to DeepBlue's victory against humanity's then-best living Chess player.

Computers Start To Win

1958

By the late 1950s, the industrious engineers at IBM were far from the only ones working on AI — excitement for the new field filled research groups in universities from the US to the Soviet Union. One such group was made up of Allen Newell and Herbert Simon (both attendants of the Dartmouth Conference) from Carnegie Mellon University, and Cliff Shaw from RAND Corporation. They collaborated on Chess AI from 1955 to 1958, culminating in "Chess Playing Programs and the Problem of Complexity"[1] which both summarized existing Chess AI research and contributed new ideas that they tested with the NSS (Newell, Shaw, and Simon) Chess program.

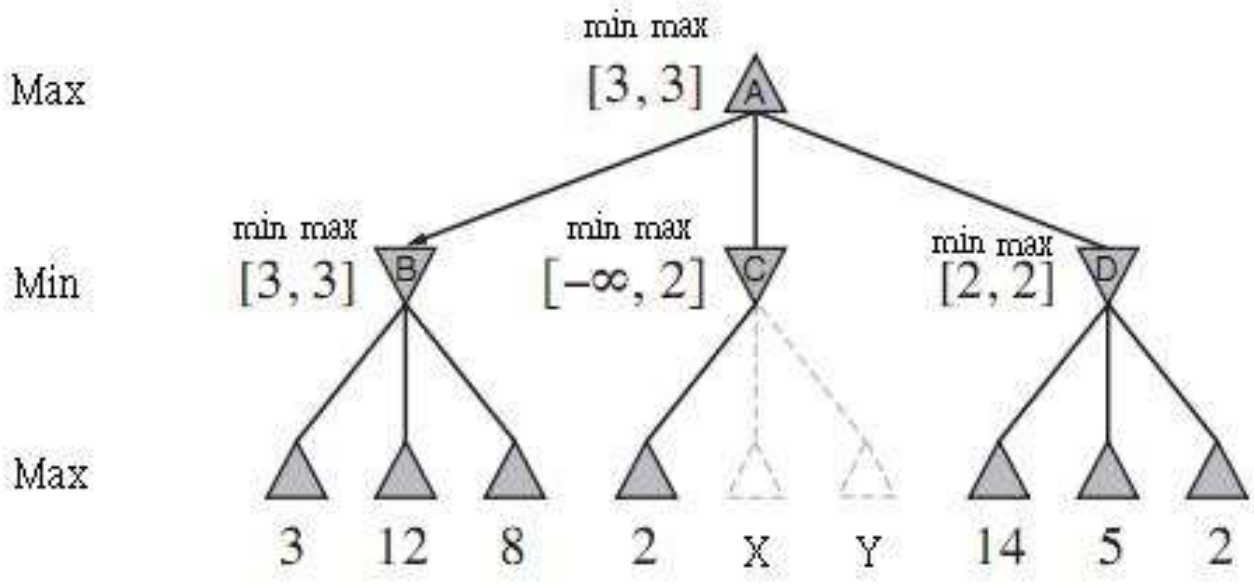
TABLE 1 Comparison of Current Chess Programs

	Turing	Kister, Stein, Ulam, Walden, Wells (Los Alamos)	Bernstein, Roberts, Arbuckle, Belsky (Bernstein)	Newell, Shaw, Simon (NSS)
Vital statistics				
Date	1951	1956	1957	1958
Board	8 × 8	6 × 6	8 × 8	8 × 8
Computer	Hand simulation	MANIAC-I 11,000 ops./sec	IBM 704 42,000 ops./sec	RAND JOHNNIAC 20,000 ops./sec
Chess program				
Alternatives	All moves	All moves	7 plausible moves Sequence of move generators	Variable Sequence of move generators
Depth of analysis	Until dead (exchanges only)	All moves 2 moves deep	7 plausible moves 2 moves deep	Until dead Each goal generates moves
Static evaluation	Numerical Many factors	Numerical Material, mobility	Numerical Material, mobility Area control King defense	Nonnumerical Vector of values Acceptance by goals
Integration of values	Minimax	Minimax (modified)	Minimax	Minimax
Final choice	Material dominates Otherwise, best value	Best value	Best value	1. First acceptable 2. Double function
Programming				
Language		Machine code	Machine code	IPL-IV, interpretive
Data scheme		Single board No records	Single board Centralized tables Recompute	Single board Decentralized List structure Recompute
Time	Minutes	12 min/move	8 min/move	1–10 hr/move (est.)
Space		600 words	7000 words	Now 6000 words, est. 16,000
Results				
Experience	1 game	3 games (no longer exists)	2 games	0 games Some hand simulation
Description	Loses to weak player Aimless Subtleties of evaluation lost	Beats weak player Equivalent to human with 20 games experience	Passable amateur Blind spots Positional	Good in spots (opening) No aggressive goals yet

45

A summary of 50s work on Chess AI from the NSS group. **(Source)**

Just as Shannon noted that master players use intuition to think selectively about moves, Newell, Shaw and Simon considered heuristics to be an important aspect of human Chess-playing. Like Bernstein's program, the NSS algorithm used a type of simple "intelligence" to choose which moves to explore. The group's most significant addition to Minimax was an approximation of something that became an essential part of future Chess playing programs: **alpha-beta pruning**. This is a small modification to Minimax that makes the algorithm avoid simulating moves that are clearly bad ('pruning' branches of the tree that need not be simulated), thus saving those precious computing resources for more promising moves. The efficiency gains can be huge, and alpha-beta pruning became as standard for future Chess programs as Minimax itself.



A tiny example of alpha-beta pruning. You are currently at position A and have three move options: B, C and D. You want to maximize your end-game score. For any move you make, the opponent will choose a move so as to minimize your score. The worst score you might get with option B is 3, so as

soon as you see your opponent has a response in option C that only nets you a score of 2, you can cease to explore option C because option B is already definitely better. **(Source)**

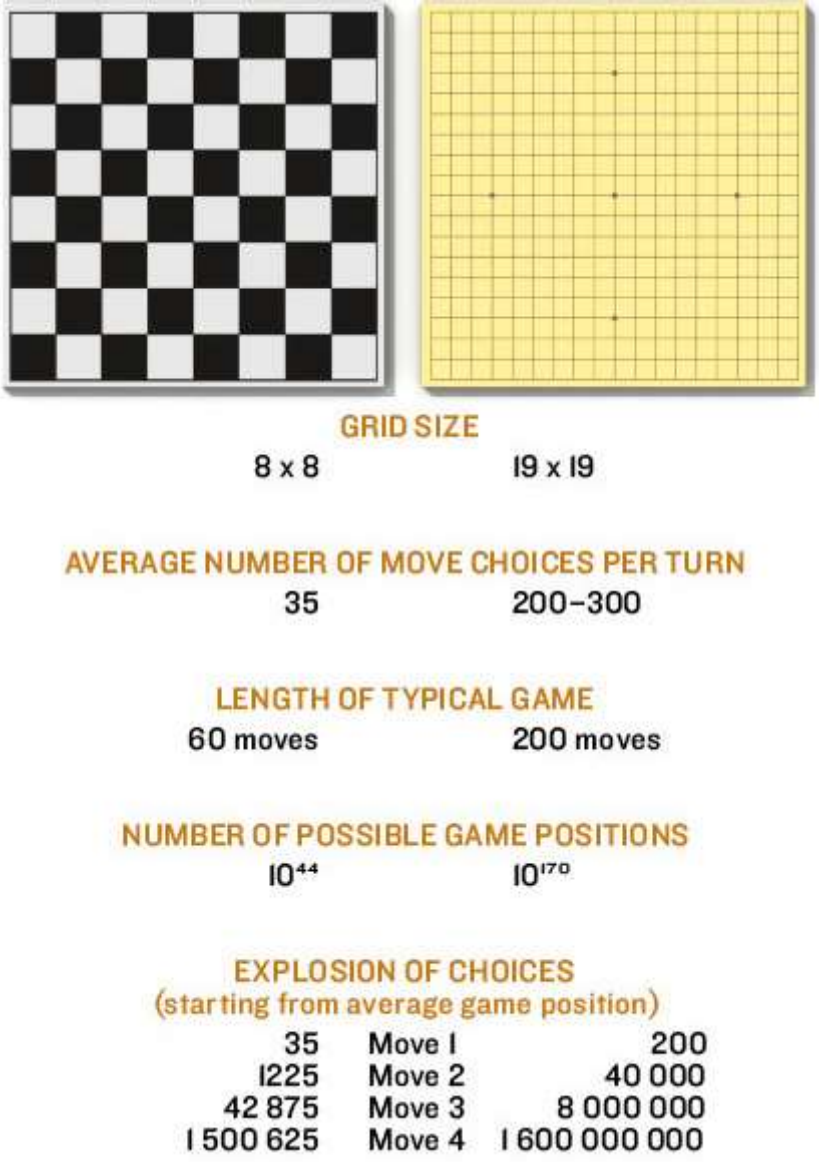
In emphasizing the need for such heuristics, the NSS group also argued that implementing them would be much easier with higher-level "interpreted" programming languages — again, this is in 1958! Back then programmers worked in the binary language of the computer, so another notable aspect of the NSS group's work is their use of a symbolic compiled programming language to implement a more complex program. As with Bernstein's program, the limitations of the hardware and of the code resulted in a rather shoddy Chess player. Still, it has been said to be the first chess program to beat (an almost humorously inexperienced) human player:

> *"In 1958, a chess program (NSS) beat a human player for the first time. The human player was a secretary who was taught how to play chess one hour before her game with the computer. The computer program was played on an IBM 704. The computer displayed a level of chess-playing expertise greater than an adult human could gain from one hour of chess instruction."*

1962

Meanwhile, Arthur Samuel's Checkers program played Checkers well already, and continued to get better. In 1962, Samuel and IBM had enough faith in the program to publicly pit it against a good human player. As described in a wonderful retrospective about the event, they strangely chose the human opponent to be Robert Nealy, who considered himself a master but was not ranked highly as a tournament player. Partially because of this, and partially because the program was good at Checkers, Nealy lost. Though it would soon be clear Samuel's program was no match for the best human players at the game — it was easily beaten by two of them at the 1966 world championship — the public and media reaction to its win in 1962 was not unlike the current media frenzy around AlphaGo:

> *"Wait! Hold the presses! A computer defeated a master checkers player! This was a major news story. Computers could solve the game of checkers. Mankind's intellectual superiority was being challenged by electronic monsters. To the technology-illiterate public of 1962, this was a major event. It was a precursor to machines doing other intelligent things better than man. How long could it possibly be before computers would be smarter than man? After all, computers have only been around for a few years, and already rapid progress was being made in the fledgling computer field of artificial intelligence. Paranoia."* Source



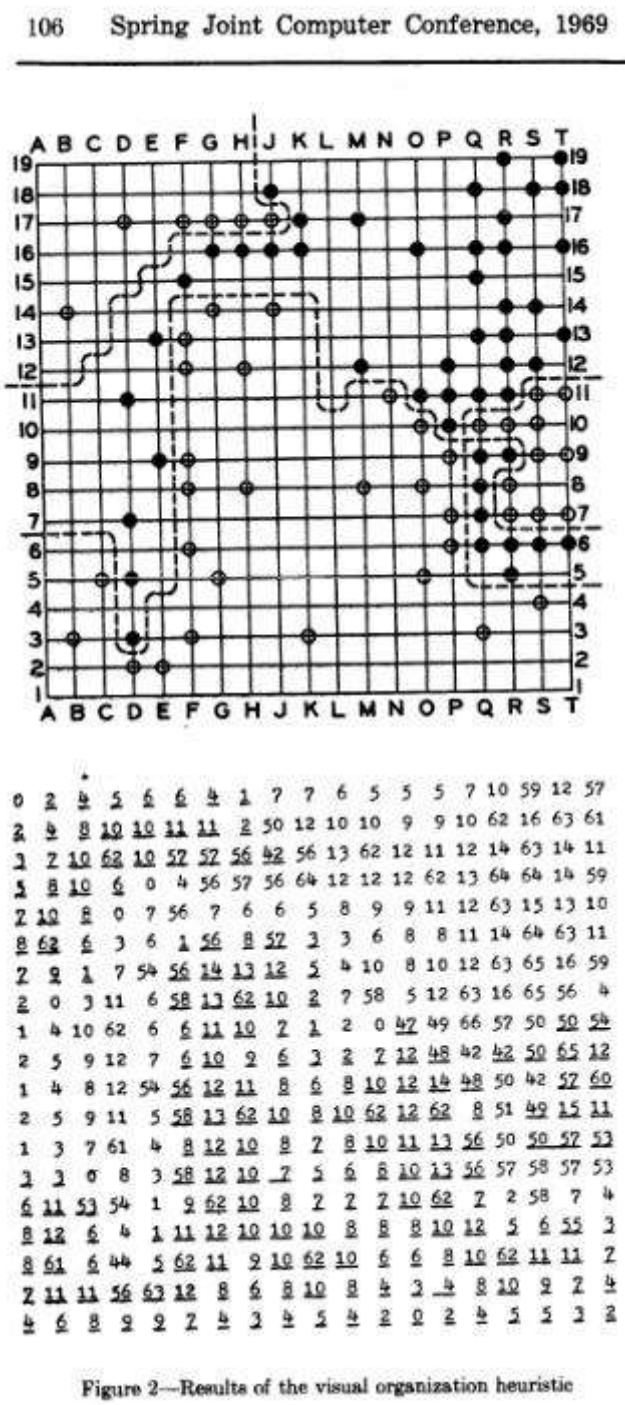
A comparison of Chess vs Go. Go has a much larger branching factor and a set of rules for which it is much harder to write an evaluation function (and here's a handy link to Wikipedia for those). **(Source)**

AlphaGo's victory is of course in a different league - Lee Sedol is unquestionably among the best players in the world and our computer-acclimated culture is less shocked by such a feat — but it is interesting to note the similarities between the two highly publicized events. Despite the fact Samuel's program was nowhere near as good as the best humans, its win gave the lasting impression Checkers was a 'simple' game that computers had already conquered and that Chess was the real challenge, much as Go was seen after Deep Blue's success with Chess. Speaking of which, 1962 was the year the first computer Go program was attempted with "Simulation of a Learning Machine For Playing Go"[2] by H. Remus (also at IBM!), though the resulting program was incomplete and never played a full game of Go. It would be half a decade more until a true Go program akin to Bernstein's Chess program or Samuel's Checkers program would play human players.

Meanwhile, yet more research teams in the Soviet Union and in the US were working on implementing Chess AIs. Notably, a group of students at MIT led by AI legend John McCarthy developed a Chess-playing program based on Minimax with alpha-beta pruning, and in 1966 faced it off against a program developed at the Moscow Institute of Theoretical and Experimental Physics (ITEP) by telegram. The Kotok-McCarthy program lost 3-1, and was in general very weak due to being limited to searching very few positions (fewer than Bernstein's

program, even). But, another student named Richard Greenblatt saw the program and, being a skilled chess player, was inspired to write his own - the Mac Hack. This program searched through many more positions and had other refinements, to the point that it could beat a ranked human player in a tournament in 1967 and win or draw multiple times more in succeeding tournaments. But it was still nowhere near as good as the best players.

Aside: more on Richard Greenblatt's Chess Program »



106 Spring Joint Computer Conference, 1969

Figure 2—Results of the visual organization heuristic

A figure showing Zobrist's visual representation from the paper.

Then, in 1968 a Go playing program reached the milestone that was conquered for Chess a whole decade earlier: beating a wholly inexperienced amateur. The program did not rely on tree search, but was rather based on emulating the way a human player “sees” an internal representation of a game position in Go so as to recognize patterns that matter for choosing the correct move. Interestingly, much of the power of AlphaGo is based on creating powerful

internal representations of the board with Machine Learning techniques commonly applied to visual tasks, so the intuition here was in a way quite right. This feat was achieved by Alfred Zobrist, as described in "A novel of visual organization for the game of GO"[3]:

> *"Given that a player "sees" a fairly stable and uniform internal representation, it follows that familiar and meaningful configurations may be recognized in terms of it. The result of visual organization is to classify a tremendous number of possible board situations into a much smaller number of recognizable or familiar board situations. Thus a player can respond to a board position he has never encountered, because it has been mapped into a familiar internal representation. This report will describe a simulation model for visual organization. … The program now has a record of two wins and two losses against human opponents. The opponents can best be described as intelligent adults who know how to play GO, have played from two to twenty games but have not studied the game. The program appears to have reached the bottom rung of the ladder of human GO players. "*

Because tricky ideas like this were necessary in order to cope with the huge branching factor and hard-to-codify heuristics of the game, progress for Go playing programs was much slower than for Chess or Checkers. It would be another decade until Bruce Wilcox developed a stronger program, again without reliance on traditional game AI techniques but with some limited tree search (as covered in this great Wired story). The approach there was to subdivide the bigger board into smaller regions that were easier to reason about, which would continue to be a hallmark of Go AIs. But even then, it was nowhere near even decent human play.

The same could not be said of Chess programs. Throughout the 70s, Chess AI progressed mostly by refining previously successful approaches. For instance, in the early 70s the Chess AI group at ITEP refined their program into a better version they named Kaissa, which went on to become the first computer Chess champion of the world in 1974 after squaring off against US programs. The program significantly benefited from faster computers and an efficient implementation that included alpha-beta pruning and some other tricks, for the first time showing the strength of the Shannon 'type A' AI strategy that relied more on fast search than smart heuristics or position evaluation.

But also by this point, it was becoming typical to use extra 'type B' ideas such as quiescence (basically searching further only after moves involving captures or checks, to not mistake trades for piece captures). It turned out to be very beneficial to selectively search further in certain tree paths than in others, so as to not miss critical turns in the game. As we'll see, these techniques ultimately proved sufficient to write Chess AIs that can beat all of humanity — though it would take a while longer to get there…

Humans Stop Winning… except for Go

1989

The first computer program to completely dominate humans at a complex game was not developed until about 3 decades after Samuel's Checkers program won that one game against Robert Nealy, and it was the Checkers program CHINOOK. The program was developed by a

team at the University of Alberta led by Jonathan Schaeffer, starting in 1989. By 1994 the best Checkers player on the planet only managed to play CHINOOK to a draw [4].

Chinook being put to the test against the world champion in Checkers. **(Source)**

By the 90s computers got orders of magnitude faster, and computer memory orders of magnitude larger, compared even to the computers of the 70s. This both enabled and enhanced the several techniques that powered CHINOOK: (A) a database of opening moves from games played by grandmasters, (B) alpha-beta tree search with an evaluation function based on a linear combination of many handcrafted features, and (C) an end-game database for all positions with fewer than eight pieces. And that's it! That's the recipe to a world-class Checkers playing program.

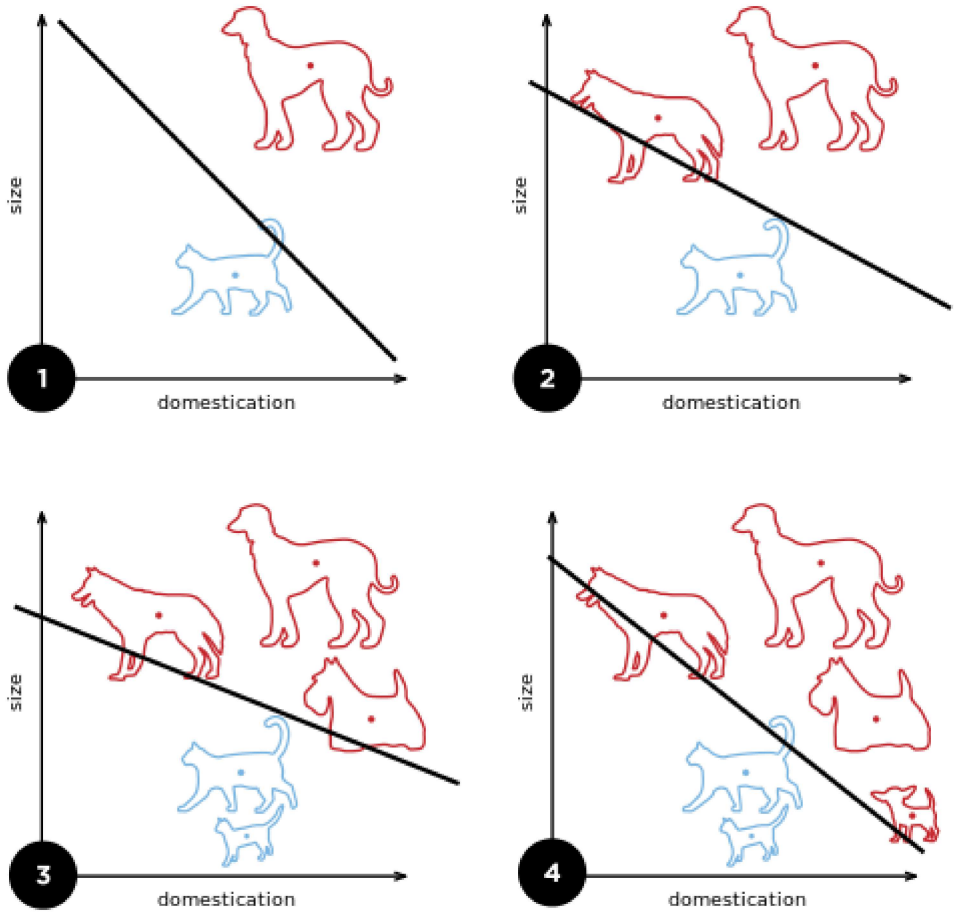
A similar recipe also powered a world-class Chess program developed around that time - Deep Thought. Developed by a team headed by Feng-hsiung Hsu, it incorporated all these ideas and had two notable extra strengths: custom hardware and smart selective extensions. According to a retrospective about its success, it was the fastest Chess program up to that point in terms of how many positions it could consider per second. This was achieved by performing move simulation and evaluation with custom circuit boards, which worked in tandem with software running on a powerful computer. In addition to being fast, Deep Thought was also smart: it had *singular extensions*, a nice type of selective extension of search past the default depth at promising positions. This allowed search depth to be extended considerably: "The result is that on the average, an N ply search penetrates along the principal variation to a depth of 1.5N and reaches a depth of 3N about once in a game"[56].

So, Deep Thought was successful precisely because it was a combination of 'type A' brute force AI (searching all positions up to a certain depth) and 'type B' selective search (searching past that depth in certain cases). By 1988, Deep Thought became the computer Chess champion of the world and, more impressively, beat Chess grandmaster Bent Larsen.

The Deep Thought team showing off their custom hardware when they won the Fredkin Intermediate Prize: "In 1988 Deep Thought and Grandmaster Tony Miles shared first place in the Software Toolworks Open in Los Angeles. Deep Thought had a 2745 performance rating, and moved its U.S. Chess Federation (USCF) rating up to 2551, and qualified for the $10,000 Fredkin Intermediate Prize as the first computer to achieve a USCF performance rating of 2500 over a set of 25 contiguous games in human tournaments." **(Source)**

Another interesting aspect of Deep Thought is that its evaluation function was automatically tuned using a database of games between master chess players, rather than having all the function's parameters hardcoded by its programmers. In this respect it harkened all the way back to Arthur Samuel's Checkers program, which also had the ability to 'learn' by tuning its evaluation function from experience. Though Chess programs improved over the decades due to increased computer speeds and ideas such as alpha-beta pruning and selective extensions, almost all programs still had no learning component and ultimately derived all their intelligence fully from their human creators. Deep Thought was a notable break from this trend.

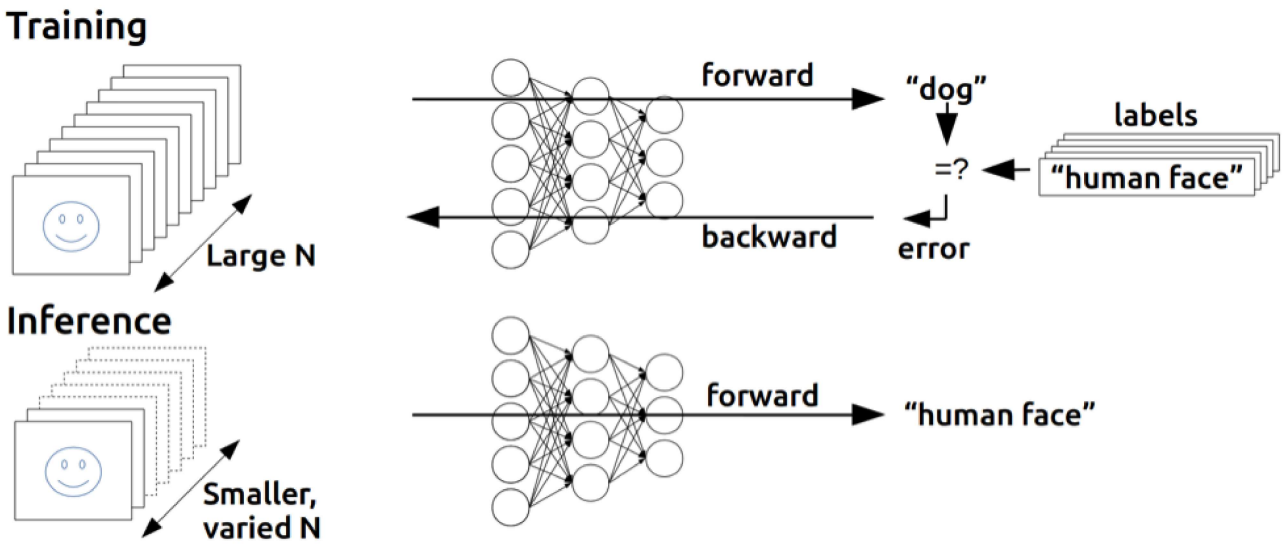
Still, the structure of Deep Thought's evaluation function encoded a lot of human intuition and knowledge about the game of chess, as was the norm. This is problematic, because the tough part in playing a game (how to evaluate positions and select moves) is essentially still solved by the programmer and not the program itself. It should be possible to write AI programs that could just learn this stuff by themselves, right? Right, and this was soon done for the first time using an algorithm that was later essential to AlphaGo: **neural networks**. Neural networks are a technique for **supervised machine learning**, which is just a category of algorithms that can learn to produce a desired output for some type of input by viewing many training examples of known input/output pairs of the same type. (For an in-depth explanation feel free to look at my little neural net writeup). Following a major 1986 paper describing how larger neural nets can be trained for tougher problems, they were all the rage in the late 80s and were being applied to many sorts of problems. One such application is the backgammon AI dubbed Neurogammon.



Visualization of supervised learning. The inputs are size and domestication, and the output is a classification of 'dog' or 'cat'. The dots already on the graph are the **training set** for learning, and the lines are the learned functions for getting an output for inputs not in the training set. **(Source)**, By Elizabeth Goodspeed - Own work, CC BY-SA 4.0

Like Go, Backgammon has a huge branching factor and the traditional tree-search-with-handcrafted-evalution-function approach does not work well. A large branching factor makes it impossible to search many moves ahead, and it is very difficult to write a great evaluation function to compensate. Gerald Tesauro, a researcher at the University of Illinois and later IBM (surprise!), and renowned Machine Learning researcher Terrence Sejnowski explored an approach based on learning a good evaluation function (a goal that had been abandoned since Arthur Samuel's work). As explained in their 1989 paper "A parallel network that learns to play backgammon", they trained a neural net to accept as input a backgammon game position and a potential move, and to output a score measuring the quality of that move [7]. This approach removes the need for engineers to attempt to encode human intuition when writing the program, which is ideal. However, to make the approach work well some human intuition was still encoded in the system in the form of **features** — derived aspects about the game position, for example piece counts in Chess — also used as input in addition to the raw game position.

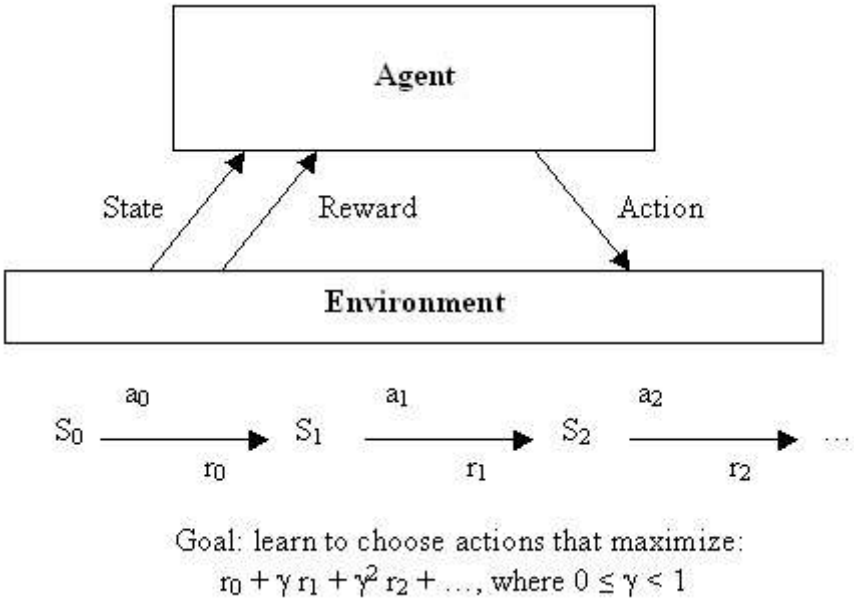
Aside: why use features? »



Supervised learning with neural nets. Basically, neural nets are made up of a bunch of units that each just output a weighted sum of their input, and the correct weights for a given application are learned from training data. Neurogammon worked exactly like this, except that the inputs were backgammon game positions, as well as derived features of the game positions, and the outputs were scores for the game position. (Source)

1992

With further improvements, the program was dubbed Neurogammon 1.0 and went on to win against all other programs at the 1989 First Computer Olympiad [8]. However, it was still not as strong as the best human players, a feat that would soon go to another neural net based program by Gerald Tesauro: TD-Gammon. First unveiled to the world in 1992, TD-Gammon was a hugely successful application of **reinforcement learning**. Unlike supervised learning, which approximates some function with particular types of input and outputs, reinforcement learning deals with finding optimal choices in different situations. More specifically, we think in terms of states (situations), in which an agent (the program) can take actions that change the agent's state in a known way (choices). Every transition between states results in a numeric 'reward', and figuring out the right action to take in a given state in order to get the highest reward in the long term, is what reinforcement learning is broadly about. Whereas supervised learning learns to approximate a function via examples of inputs and outputs, reinforcement learning generally learns from 'experience' of receiving rewards after trying actions in different states.

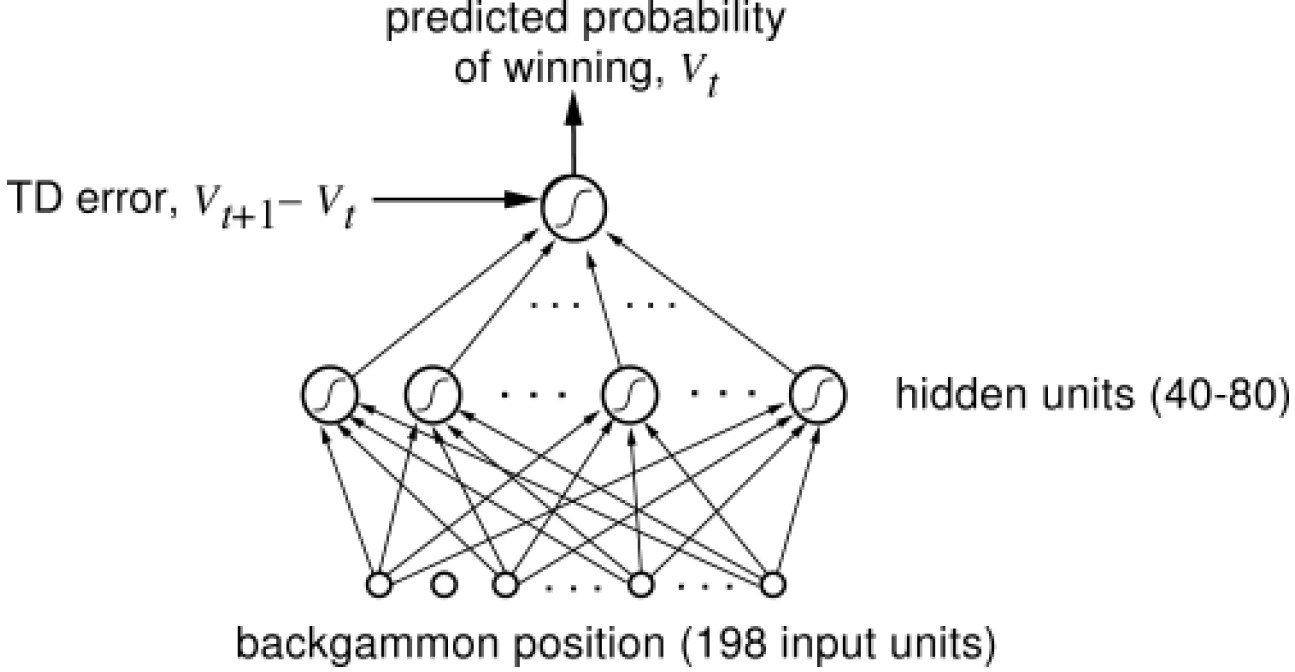


A visualization of the general idea of reinforcement learning. Rather than learning to compute a correct output given some input, as in supervised learning, the goal is to learn to choose a correct action in any state in order to obtain the maximum reward in the long term. **(Source)**

So, TD-Gammon learned by just playing games of backgammon against prior versions of itself, observing which player won, and using that experience to tune a neural net to produce a probability of winning from any given position. This is fundamentally different from Neurogammon, which required compiling a dataset of hundreds of moves with human-assigned scores and was thus much more cumbersome than just letting the program play games against itself for a few hours. Note this is very similar to what Arthur Samuel was trying to do all the way back in 1957 with his Checkers program that learned from self-play. In fact, the type of reinforcement learning TD-Gammon is based on, Temporal Difference Learning, was developed in 1986 by Richard Sutton as a formalization of the learning in Samuel's work[9].

1994

Besides learning through self-play, there was nothing fancy in the approach - instead of using tuned tree search the program just exhaustively looked at all positions two steps ahead and used the move that led to the largest probability of winning. With just the raw board positions as input — essentially no human intuition engineered into it — TD-Gammon achieved a level of play comparable to Neurogammon. And with the addition of Neurogammon's features, it became comparable to the best human players in the world[10].



The TD-Gammon neural net that learned to play expert-level Backgammon. The input later included features in addition to the raw board positions. **(Source)**

TD-Gammon is to this day a milestone in the history of AI. But, when researchers naturally tried to use the same approach for other games, the results were not quite as impressive. Sebastian Thrun's NeuroChess[11] (1995) was only comparable to commercial Chess programs on a low difficulty setting, and Markus Enzenberger's NeuroGo[12] (1996) likewise did not match the skill of existing (poor) Go AIs. In the case of NeuroChess, the discrepancy was surmised to be in large part due to the large amount of time it took to compute the evaluation function ("Computing a large neural network function takes two orders of magnitude longer than evaluating an optimized linear evaluation function (like that of GNU-Chess)"), making NeuroChess unable to explore nearly as many moves ahead as the commercial Chess program. The benefit of a better evaluation function just did not win out over a simpler one that allowed for many more positions to be explored during search.

1997

Which brings us back to Deep Thought. After the success of that program, some of the same team were hired by IBM and set out to create Deep Thought II, which was later renamed Deep Blue (Deep Thought x Big Blue = Deep Blue). By and large, Deep Blue was conceptually the same as Deep Thought but much, much beefier in terms of computing power — it was a custom- built supercomputer! Still, when it played Kasparov in 1996 Deep Blue lost with a score of 2-4. The team then spent a year making Deep Blue yet more powerful and tuning its evaluation function, and it was this version that historically beat Kasparov with a score of 3.5-2.5 on May 11th of 1997[13].

The supercomputer that powered Deep Blue **(Source)**

Kasparov vs Deep Blue **(Source)**

Deep Blue beat G. Kasparov in 1997

A short documentary about Kasparov vs Deep Blue.

The team credited many things with getting Deep Blue to the point that it could score this victory[13]:

> *"There were a number of factors that contributed to this success, including:*
> *1. a single-chip chess search engine,*
> *2. a massively parallel system with multiple levels of parallelism,*
> *3. a strong emphasis on search extensions,*
> *4. a complex evaluation function, and*
> *5. effective use of a Grandmaster game database"*

So, it would be wrong to claim DeepBlue won purely through "brute-force", since it included decades of ideas about how to tackle the AI problem of Chess. But brute-force surely was hugely important - DeepBlue was run with thirty processors inside a supercomputer working jointly with 480 single-chip chess search engines (16 per processor). When playing Kasparov it observed 126 million positions per second on average, and typically searched to a depth of between 6 and 12 plies and to a maximum of forty plies. All this allowed it to barely win, arguably due to uncharacteristic blunders on Kasparov's part. But, all that hardly matters; since then computers have continued to become exponentially faster, and today humanity's best Chess players are likely no match for programs you can run on your smartphone.

So, Checkers, Chess, and Backgammon had all been mastered by AI programs by the late 90s - what about Go? Even the best computer programs were poor matches for amateurs with some experience. The techniques we've seen so far — supervised learning, reinforcement learning, and well-tuned tree search — were all attempted and found insufficient to make a Go program that could challenge serious human players. To see why these approaches failed, and how their defects were addressed over the span of two decades culminating in the creation of AlphaGo, go on ahead to the final part of this history.

Acknowledgements

Big thanks to Abi See and Pavel Komarov for helping to edit this.

References

1. Allen Newell, Cliff Shaw, Herbert Simon (1958). Chess Playing Programs and the Problem of Complexity. IBM Journal of Research and Development, Vol. 4, No. 2, pp. 320-335. Reprinted (1963) in Computers and Thought (eds. Edward Feigenbaum and Julian Feldman), pp. 39-70. McGraw-Hill, New York, N.Y. pdf ↩

2. Remus, H. (1962, January). Simulation of a learning machine for playing Go. In COMMUNICATIONS OF THE ACM (Vol. 5, No. 6, pp. 320-320). 1515 BROADWAY, NEW YORK, NY 10036: ASSOC COMPUTING MACHINERY. ↩

3. Zobrist, A. L. (1969, May). A model of visual organization for the game of Go. In Proceedings of the May 14-16, 1969, spring joint computer conference (pp. 103-112). ACM. ↩

4. Schaeffer, J., Lake, R., Lu, P., & Bryant, M. (1996). CHINOOK, The World Man-Machine Checkers Champion. AI Magazine, 17(1), 21. ↩

5. Berliner, H. J. (1989). Deep Thought Wins Fredkin Intermediate Prize. AI Magazine, 10(2), 89. ↩

6. Anantharaman, T., Campbell, M. S., & Hsu, F. H. (1990). Singular extensions: Adding selectivity to brute-force searching. Artificial Intelligence, 43(1), 99-109. ↩

7. Tesauro, G., & Sejnowski, T. J. (1989). A parallel network that learns to play backgammon. Artificial Intelligence, 39(3), 357-390. ↩

8. Tesauro, G. (1989). Neurogammon wins computer olympiad. Neural Computation, 1(3), 321-323. ↩

9. Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine learning, 3(1), 9-44. ↩

10. Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural computation, 6(2), 215-219. ↩

11. Thrun, S. (1995). Learning to play the game of chess. Advances in neural information processing systems, 7. ↩

12. Enzenberger, Markus. "The integration of a priori knowledge into a Go playing neural network." (1996). ↩

13. Campbell, M., Hoane, A. J., & Hsu, F. H. (2002). Deep blue. Artificial intelligence, 134(1), 57-83. ↩ ↩[2]

Previous Post | Next Post