Evolving an Expert Checkers Playing Program without Using Human Expertise

Kumar Chellapilla
David B. Fogel*
Natural Selection, Inc.
3333 N. Torrey Pines Ct., Suite 200
La Jolla, CA 92037
dfogel@natural-selection.com
(858) 455-6449
(858) 455-1560 fax
* corresponding author

Abstract

An evolutionary algorithm has taught itself how to play the game of checkers without using features that would normally require human expertise. Using only the raw positions of pieces on the board and the piece differential, the evolutionary program optimized artificial neural networks to evaluate alternative positions in the game. Over the course of several hundred generations, the program taught itself to play at a level that is competitive with human experts (one level below human masters). This was verified by playing the best-evolved neural network against 165 human players on an Internet gaming zone. The neural network's performance earned a rating that is better than 99.61 percent of all registered players at the website. Control experiments between the best-evolved neural network and a program that relies on material advantage indicate the superiority of the neural network both at equal levels of look-ahead and CPU time. The results suggest that the principles of Darwinian evolution may be usefully applied to solving problems that have not yet been solved by human expertise.

Introduction

A fundamental approach to artificial intelligence has been to capture human expertise in the form of programmed rules of behavior that, when executed on a computer, emulate the behavior of the human expert. Simulating the symptoms of intelligent behavior rather than the mechanisms that underlie that behavior was a natural outgrowth of the pressure to design useful software applications. This approach, however, presents a significant limitation: Artificial intelligence programs are restricted mainly to those problems that

are, in large part, already solved. The challenge of creating machines that can learn from their own experience without significant human intervention has remained unsolved.

Efforts in machine learning have often been aimed at reinforcement learning, where a series of actions leads to success or failure and some form of credit or blame for the final result can be apportioned back to each action (see [1, 2] and many others). In a game of strategy, such as checkers, the final result of win, lose, or draw is associated with numerous decisions made from the first to the final move. The *credit assignment problem* then has been to find rules for rewarding "correct" actions and penalizing "incorrect" actions. It must be acknowledged, however, that describing any particular action as correct or incorrect may be vacuous because the end result is typically a nonlinear function of the interaction of the moves played by both players. The final result is more than simply the sum of the attributed worth of each move in a series. Nevertheless, this approach has been adopted largely based on a long-standing belief that there is insufficient information in "win, lose, or draw" when referred to the entire game to "provide any feedback for learning at all over available time scales" [3]. Reinforcement learning seeks to take the amorphous "win, lose, or draw" and decompose it into hopefully meaningful smaller "chunks" that can indicate whether or not specific actions should be favored or disfavored.

Experiments described here indicate that, in contrast, a machine learning algorithm based on the principles of Darwinian evolution can generate an expert-level checkers playing program without relying on any specific credit assignment algorithms (cf. [9], which evolved a backgammon player, although not at the expert level; also see [10] and [11] for efforts to evolve neural networks to play Go and Othello, respectively). Furthermore, this level of play is attained without using features about the game that would require human expertise. Only the position of the pieces on the board, the spatial characteristics of the checkerboard, and the piece differential are made available as explicit data to be processed. The evolutionary algorithm optimizes a population of artificial neural networks (i.e., networked layers of nonlinear processing elements) that are used to evaluate the worth of alternative potential positions. The procedure starts from a

collection of completely random networks that then compete against each other for survival and the right to generate offspring networks through a process of random variation. Survival is determined by the quality of play in a series of checkers games played against opponents from the same population. Over successive generations of variation and selection, the surviving neural networks extract information from the game and improve their performance. The best-evolved neural network has been played against 165 human opponents over the Internet and has earned an expert rating at a well-known gaming web site (www.zone.com). The details on the evolutionary approach, the evaluation of the best neural network's rating, and the results of control experiments are offered below.

Background and Method

Checkers (also known as "draughts") is played on an eight-by-eight board with squares of alternating colors. There are two players who take sides denoted by "red" or "white" (or "black" and "white"). Each side has 12 pieces (also called checkers) that begin in the 12 alternating squares of the same color that are closest to that player's side, with the right-most square on the closest row to the player remaining open (Figure 1). Checkers move forward diagonally one square at a time or, when possible, may jump over an opposing checker into an empty square. Jump moves are forced, although if more than one possible jump is available, the player may choose which jump to execute. When a checker advances to the last row of the board it becomes a king and may move forward or backward diagonally. The game ends when one side cannot make a legal move, which is most commonly brought about by removing that player's final piece. The player who cannot move loses and the other player wins. Alternatively, a draw may be declared upon mutual agreement of the players, or in tournament play at the discretion of a third party under certain circumstances. Tournament play also involves a time restriction (60 minutes for the first 30 moves) which if violated results in a loss.

For the present experiments, each checkerboard was represented by a vector of length 32, with each component corresponding to an available position on the board. Components in

the vector were elements from {$-K$, -1, 0, $+1$, $+K$}, where 0 corresponded to an empty square, 1 was the value of a regular checker, and K was the number assigned for a king. A common heuristic has been to set this value at 1.5 times the worth of a checker, but such expertise was eschewed in these experiments. Instead, the value of K was evolved by the algorithm. The sign of the value indicated whether or not the piece belonged to the player (positive) or the opponent (negative).

A player's move was determined by evaluating the presumed quality of the resulting future positions. The evaluation function was structured as an artificial neural network comprising an input layer, three internal processing (hidden) layers, and an output node (Figure 2). The first internal processing layer was designed to indicate the spatial characteristics of the checkerboard without indicating explicitly how such knowledge might be applied. The remaining internal and output neurons operated based on the dot product of the evolvable weighted connections between nodes and the activation strength that was emitted by each preceding node. Each node used a hyperbolic tangent (tanh, bounded by ± 1) with an evolvable bias term. In addition, the sum of all entries in the input vector was supplied directly to the output node, constituting a measure of the relative piece differential.

When a board was presented to a neural network for evaluation, the scalar output of the network was interpreted as the worth of the board from the perspective of the player whose pieces were denoted by positive values. The closer the output was to 1.0, the better the evaluation of the corresponding board. Similarly, the closer the output was to –1.0, the worse the board. All positions that were wins for the player (e.g., no remaining opposing pieces) were assigned a value of exactly 1.0, and likewise all losing positions were assigned a value of –1.0.

The evolutionary algorithm began with a randomly created population of 15 artificial neural networks (also described as *strategies*), P_i, $i = 1, \ldots, 15$, defined by the weights and biases for each network and the associated value of K. Weights and biases were sampled uniformly over [–0.2, 0.2], simply to provide a small range of initial variability,

with the value of K set initially at 2.0. Each strategy had an associated self-adaptive parameter vector σ_i, $i = 1, \ldots, 15$, where each component corresponded to a weight or bias and served to control the step size of the search for new mutated parameters of the neural network. The self-adaptive parameters were initialized at 0.05 for consistency with the range of initial weight and bias terms.

Each "parent" generated an offspring strategy by varying all of the associated weights and biases, and possibly the K value as well. Specifically, for each parent P_i, $i = 1, \ldots, 15$, an offspring P'_i was created by:

$$\sigma'_i(j) = \sigma_i(j) \exp(\tau N_j(0,1)), \; j = 1, \ldots, N_w$$

$$w'_i(j) = w_i(j) + \sigma'_i(j) N_j(0,1), \; j = 1, \ldots, N_w$$

where N_w is the total number of weights and bias terms in the neural network (here, 5046), $\tau = 1/\text{sqrt}(2\ \text{sqrt}(N_w)) = 0.0839$, and $N_j(0,1)$ is a standard Gaussian random variable resampled for every j. The offspring king value K' was obtained by:

$$K'_i = K_i + \delta$$

where δ was chosen uniformly at random from $\{-0.1, 0, 0.1\}$. For convenience, K'_i was constrained to the range [1.0, 3.0].

All parents and offspring competed for survival by playing games of checkers and receiving points for the results of their play. Each neural network in turn played one game against five randomly selected opponents from the population (with replacement). In each of these five games, the player always played red and the selected opponent played white. Each player scored –2, 0, or +1 points for a loss, draw, or win, respectively. These values were chosen as being reasonable and no optimality is claimed. A draw was declared if a game lasted for 100 moves. In total, there were 150 games per generation, with each strategy participating in an average of 10 games. After all games were complete, the 15 strategies that received the greatest total points were retained as parents for the next generation and the process was iterated.

Each game was played using a minimax alpha-beta search [4] of the associated game tree that results from looking ahead over a selected number of moves. The ply depth of the

search, *d*, was set at four (i.e., two moves for each side) to allow for reasonable execution times (30 generations required about 7 days on a 400-MHz Pentium II). The ply was extended in units of two any time a forced jump move was encountered because in these situations no real decision is available. In addition, the ply was also extended if the final look-ahead left the board in a state where there was a forced jump. The best move to make was chosen by iteratively minimizing or maximizing over the leaves of the game tree at each ply according to whether it was the opponent's or the player's move.

This process of having the evolving neural networks play against each other was iterated for 840 generations (6 months), whereupon the best neural network was used to play against human opponents on the web site http://www.zone.com. The game is played using a Java-applet with an associated interface for allowing players to communicate via instant messages (Figure 3). Players at this site are rated according to a standard system (following the tradition of the United States Chess Federation) where the initial rating is $R_0 = 1600$ and the player's score is adjusted based on the outcome of a match and the rating of the opponent:

$$R_{new} = R_{old} + C(Outcome - W)$$

where $W = 1/(1+10^{((R_{opp} - R_{old})/400)})$, $Outcome \in \{1 \text{ if Win}, 0.5 \text{ if Draw}, 0 \text{ if Loss}\}$, R_{opp} is the opponent's rating, and $C = 32$ for ratings less than 2100, $C = 24$ for ratings between 2100 and 2399, and $C = 16$ for ratings at or above 2400. Standard designations for the level of play are shown in Table 1.

Results

Over the course of two months, we played 165 games against human opponents at the web site using the best-evolved network to determine our moves. No opponent was told that they were playing against a computer program, and only two of the opponents ever mentioned a suspicion that we were in fact relying on a program. (More correctly, the program was relying on us, as we did not use the program to boost our own ratings, but

rather merely served as the conduits between the neural network and its opponents.) Games were played mainly using a ply of $d = 8$ to ensure reasonably fast play (i.e., under one minute per move, typically). Occasionally, lower or higher ply were necessary or possible. Opponents were chosen based on availability to play and to ensure that the best-evolved neural network competed against a diverse array of skill levels and with roughly equal numbers of games as red and white (84 and 81, respectively). Figure 4 shows a histogram of the number of games played against players of various ratings along with the win-draw-loss record attained in each category.

The neural network's final rating depends on the order in which the games are played; however, because no learning was undertaken during the 165 games these could be recorded in any sequence. An estimate of the network's rating can be generated by sampling from the population of permutations of possible orderings of opponents. The network's rating was calculated over 5000 random permutations and resulted in a mean of 2045.85 with a standard error of 0.48. (A head-to-head match against Chinook using a five-ply search provided an independent verification of this rating. These results appear in [16].) Figure 5 shows the mean rating trajectory and indicates that it has saturated. This level of play earns the network an expert rating and placed it better than 99.61% of all rated players at the web site. The best result from the 165 games was obtained when the network defeated a player rated 2173, just 27 points away from the master level, who was ranked 98th out of over 80,000 registered players. The annotated moves of the game are offered in the appendix.

Two control experiments were conducted to determine whether or not the neural network was providing any advantage beyond the piece differential, and whether or not the neural network was worth the extra required CPU time as compared to a program that relies solely on the piece differential. There are seven possible first moves in checkers. A series of 14 games was played for each control experiment: the series comprised all possible first moves with the neural network playing both as red and then as white. In the first experiment, both the neural network and a piece-differential player (using the common heuristic $K = 1.5$) were given an eight-ply look ahead combined with the minimax rule. In

the second experiment, the two programs were given two minutes to search ahead as far as they could on a Celeron 466 MHz computer. The neural network comprises over 5000 weights and therefore requires more computation to assess the best move at a prescribed ply as compared with a simple piece-differential player. Games were played until (1) either player won the game, or (2) the two players toggled positions indefinitely. In the latter case, a shareware program called Blitz98 was used to play out the game and assess a winner. Blitz98 is a strong checkers program and uses an endgame database to compute optimal moves.

In the first experiment, which gave each player an equal look ahead, of the 14 games, 2 were played out to completion, with the best-evolved neural network winning both of these. Of the remaining 12 games, the neural network held a material advantage in 10 games and was behind in the other 2. Using Blitz98 to play out the 12 incomplete games, the neural network was awarded the victory in 8 of them, the piece-differential player earned the win in 1 game, and the remaining 3 games were played out as draws. By all measures, the best-evolved neural network was significantly better than the piece-count player. The results are statistically significant using a nonparametric sign test ($P < 0.02$). Using the standard rating formula, the results suggest that the best-evolved neural network at eight ply is about 400 points better than the piece-differential player at the same eight ply (based on 10 wins out of 11 decisions). A program that relies only on the piece count and an eight-ply search will defeat a lot of people, but it’s not an expert. The best-evolved neural network is.

In the second experiment, based on two minutes of search time for each player, with the same termination criteria as before, the best-evolved neural network again won two games outright and lost none. Of the remaining 12 games, the neural network was ahead on material in 6 games, behind in 1, and even in 5. Based on using Blitz98 to complete the 12 games, the final tally was 8 wins for the neural network, 2 losses, and 4 draws. The piece-differential player was typically searching two ply more than the neural network, and occasionally as many as six ply more. The neural network could typically complete an eight-ply search in two minutes, but in some cases could only complete six ply. In

contrast, the piece-differential player could normally complete a 10-ply search in the allowed two minutes, but sometimes could complete a 14-ply search. Despite having a common horizon of two or more ply beyond the neural network, the piece-differential player was beaten decisively (an estimated 240-point difference in rating based on 8 wins in 10 decisions, with a *P*-value of 0.1094 using a nonparametric sign test).

This demonstrates that the extra computation involved in using the best-evolved neural network is indeed valuable. The compelling rationale for explaining how the piece-differential player can lose to the neural network when the piece-differential player looks farther ahead in the game is that the evolutionary program has captured useful positional information in the neural network that overcomes the deficit. We cannot yet identify what features might be contained in the neural network, but there is little doubt that they are indeed real, and are more valuable than the extra look into the future provides.

The results also indicate that the best-evolved neural network is able to defeat a piece-differential player that incorporates random variation in evaluating moves. Despite perhaps being counterintuitive, the simple combination of material evaluation and small random variations in evaluation can sometimes be a useful heuristic [12]. While monitoring the evolution, we noted that the best neural networks in the initial population, which relied on the piece differential and small random weights, were replaced early on by superior offspring. Moreover, there were many occasions when the best neural network at one generation was replaced by a neural network that demonstrated improved performance.

We did not observe any evidence of a so-called “rock-paper-scissors” effect, where the performance of the surviving neural networks regresses over successive generations in light of competition against a limited set of opponents [9]. Furthermore, our observation holds across three other independent trials that evolved neural networks to play checkers: Additional generations of evolution have led consistently to higher performance ratings [13-15]. Developing a theory to predict when the rock-paper-scissors effect may be expected remains an open area of research.

Discussion

To appreciate the level of play that has been achieved, it may be useful to consider the following thought experiment. Suppose you are asked to play a game on an eight-by-eight board of squares with alternating colors. There are 12 pieces on each side arranged in a specific manner to begin play. You are told the rules of how the pieces move (i.e., diagonally, forced jumps, kings) and that the piece differential is available as a feature. You are not, however, told whether or not this differential is favorable or unfavorable (there is a version of checkers termed “suicide checkers” where the object is to “lose” as fast as possible) or if it is even valuable information. Most importantly you are not told the object of the game. You simply make moves and at some point an external observer declares the game over. They do not, however, provide feedback on whether or not you won, lost, or drew. The only data you receive comes after a minimum of five such games, and is offered in the form of an overall point score. Thus you cannot know with certainly which games contributed to the overall result, or to what degree. Your challenge is to induce the appropriate moves in each game based only on this coarse level of feedback.

It is of interest to place the result of evolving an expert-rated player in the context of other computer programs for playing checkers. In 1959, Samuel [5] used a deterministic learning procedure where boards were evaluated based on a weighted polynomial comprising features that were believed to be important for generating good moves. One polynomial was played against another that was a slight perturbation of the first, with the winner becoming the progenitor for the next iteration. Over several iterations, the level of performance improved. In 1962, Samuel’s program defeated Robert Nealey, who was described as a Connecticut state champion (he actually earned this title later). However, not only did Nealey defeat Samuel’s program in a rematch, the program also lost eight straight games to the world champion and challenger in 1968, and lost to a program from

Duke University in 1977. In the opinion of the American Checkers Federation Games editor, Samuel's program could not compete at the Class B level [6].

The current world-champion checkers program is called "Chinook" [7], rated at 2814. The closest human competitors are rated in the high 2600s [6]. Chinook relies on features that were chosen using human expertise and weighted by hand tuning. It also includes a look-up table of transcribed games from previous grand masters and a complete endgame database for all cases with up to eight pieces on the board (440 billion possible states). Once the endgame is reached, Chinook never makes a mistake because the final outcomes that can be achieved from these states have been enumerated. Chinook does not use self-learning techniques to improve its play, relying instead on opening books, perfect information in the endgame, and on high-speed computation to look ahead as many ply as possible. (Chinook also has other routines to determine when to use opening books, how to extend a search, and other facets that affect the quality of play.)

The evolved neural networks certainly cannot compete with Chinook, or with players at the lesser-ranked master level. Yet the evolutionary program exhibits a flexibility that cannot be achieved with Chinook or other similar approaches (e.g., Deep Blue in chess) that rely on all of their "intelligence" being preprogrammed. The evolutionary algorithm is capable of adapting its play to meet more demanding challenges from superior opponents. It can invent new and unorthodox tactics. Indeed, it was common for human opponents to offer comments that the network's moves were "strange" yet the level of play was often described as "very tough" or with other complimentary terms (Figure 3).

This evolutionary approach to machine learning offers the potential for exploring any possible pathway presented by a complex problem. It does not complete an exhaustive search, yet it can quickly identify and converge on useful solutions. This may provide an effective means for going beyond the structured conventional engineering approach that is common to many forms of human design. The process is inherently parallel and could be accelerated greatly by utilizing a distributed network of small and inexpensive

processors. Furthermore, no explicit design criteria are required. At a minimum, all that is necessary is to be able to compare two solutions and indicate which is better. The process of Darwinian evolution can then optimize the solution for the given problem. In this manner, evolutionary machine learning can meet and eventually exceed the capabilities of human expertise.

References

1. M. L. Minsky, “Steps toward artificial intelligence,” *Proc. IRE*, vol. 49, 1961, pp. 8-30.
2. G. Tesauro, “Practical issues in temporal difference learning,” *Machine Learning*, vol. 8, 1992, pp. 257-277.
3. A. Newell quoted in [1], p. 20.
4. H. Kaindl, “Tree searching algorithms,” in *Computers, Chess, and Cognition*, T.A. Marsland, T. and J. Schaeffer, eds. New York: Springer, 1990, pp. 133-168.
5. A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM J. Res. Dev*., vol. 3, pp. 210-219, 1959.
6. J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers*, New York: Springer, 1996, pp. 97, 447.
7. J. Schaeffer, R. Lake, P. Lu. and M. Bryant, “Chinook: The world man-machine checkers champion,” *AI Magazine*, vol. 17, 1996, 21-29.
8. J.B. Pollack and A.D. Blair, “Co-evolution in the successful learning of backgammon strategy,” *Machine Learning*, Vol. 32:3, pp. 225-240, 1998.
9. N. Richards, D. Moriarty, and R. Miikkulainen, “Evolving neural networks to play go,” *Applied Intelligence*, Vol. 8, pp. 85-96, 1998.
10. D.E. Moriarty and R. Miikkulainen, “Discovering complex Othello strategies through evolutionary neural networks,” *Connection Science*, Vol. 7, pp. 195-209, 1995.
11. D.F. Beal and M.C. Smith, “Random evaluations in chess,” *ICCA Journal*, Vol. 17:1, pp. 3-9, 1994.
12. K. Chellapilla K and D.B. Fogel (1999) "Co-Evolving Checkers Playing Programs using only Win, Lose, or Draw," *AeroSense99, Symposium on Applications and Science of Computational Intelligence II*, Vol. 3722, K. Priddy, S. Keller, D.B. Fogel, and J.C. Bezdek (eds.), SPIE, Bellingham, WA, pp. 303-312, 1999.

13. K. Chellapilla K and D.B. Fogel, "Evolving Neural Networks to Play Checkers without Expert Knowledge," *IEEE Trans. Neural Networks*, Vol. 10:6, pp. 1382-1391, 1999.
14. K. Chellapilla and D.B. Fogel. “Evolution, Neural Networks, Games, and Intelligence,” *Proc. IEEE*, Vol. 87:9, Sept., pp. 1471-1496, 1999.
15. D.B. Fogel, “Using Evolutionary Programming to Construct Neural Networks that are Capable of Playing Tic-Tac-Toe,” Proceedings of the 1995 IEEE International Conference on Neural Networks, San Francisco, CA, pp. 875-880, 1993.
16. D.B. Fogel and K. Chellapilla, “Verifying Anaconda’s Expert Rating by Competing Against Chinook: Experiments in Co-Evolving a Neural Checkers Player,” *Neurocomputing*, in press, 2001.

Appendix

This appendix contains the complete sequence of moves from a selected game where the best-evolved neural network from generation 840 defeated a human rated 2173 (expert level). The notation for each move is given in the form *a*-*b*, where *a* is the position of the checker that will move and *b* is the destination. Forced moves (mandatory jumps or occasions where only one move is available) are indicated by (f). Accompanying the sequence of moves is a figure showing the end state of the game.

Game Moves:

Game Against Human Rated 2173 (Expert)
Human Plays Red, Neural Network Plays White
(f) denotes a forced move
Comments on moves are offered in brackets

Human	Computer	Comment
1.R:11-15	1.W:23-19	
2.R:9-14	2.W:27-23	
3.R:8-11	3.W:22-18	[Computer initiates a double swap]
4.R:15-22(f)	4.W:25-18-9	
5.R:5-14	5.W:26-22	
6.R:11-15	6.W:32-27	
7.R:4-8	7.W:22-17	
8.R:8-11	8.W:17-13	
9.R:11-16	9.W:24-20	[Computer initiates a double swap, Figure 6]
10.R:15-24(f)	10.W:20-11	
11.R:7-16(f)	11.W:27-20-11	[Computer double jumps back during the double swap and gains a checker during the swap]
12.R:12-16	12.W:28-24	
13.R:16-20	13.W:31-27	
14.R:10-15	14.W:30-26	
15.R:6-10	15.W:13-9	
16.R:1-5	16.W:29-25	
17.R:3-7	17.W:11-8	
18.R:14-17	18.W:21-14(f)	[Human initiates a double swap]
19.R:10-17(f)	19.W:8-3	[Computer gets a King, Figure 7]
20.R:5-14(f)	20.W:3-10-19(f)	[The two successive swaps result in the Human falling behind by a King and a piece - Human resigns, Figure 8]

Figure Legends:

Figure 1. The opening board in a checkers game. Red moves first.

Figure 2. The artificial neural network used to represent checkers playing strategies. The network served as the board evaluation function. Given any board pattern as a vector of 32 inputs, the first hidden layer (spatial preprocessing) assigned a node to each possible square subset of the board. Each available square in a given subset was assigned a weighted connection and the dot product of the weights and elements in the available squares was filtered through a hyperbolic tangent function. The outputs from these nodes were then passed through two additional hidden layers of 40 and 10 nodes, respectively. No optimality is claimed for this particular design. (The authors felt it was a reasonable choice based on prior efforts to evolve neural networks to play tic-tac-toe [15].) The final output node was scaled between [−1,1] and included the sum of the input vector as an additional input. All of the weights and bias terms of the network were evolved, as well as the value used to describe kings.

Figure 3. A sample from a game played on the web site. The upper-left panel shows the present state of the game between Decepticon12345 (an expert-rated opponent) and David1101 (the neural network). David1101 has just double jumped over the opponent's king and a checker and has gained a king in the process (upper-right square). The remarks below the board show Decepticon12345 commenting on David1101's play with the acronym "gm," which stands for "good move."

Figure 4. The performance of the best-evolved neural network after 840 generations, played over 165 games against human opponents on the Internet checkers site. The histogram indicates the rating of the opponent and the associated performance against opponents with that rating. Ratings are binned in intervals of 100 units (i.e., 1650 corresponds to opponents who were rated between 1600 and 1699). The numbers above each bar indicate the number of wins, draws, and losses, respectively. Note that the evolved network generally dominated opponents who were rated less than 2000, and played competitively with those rated between 2000 and 2099.

Figure 5. The mean sequential rating of the best-evolved neural network over 5000 random permutations of the 165 games played against human opponents. The mean

rating starts at 1600 (the standard starting rating at the web site) and steadily climbs to a rating of above 2000 by game 60. As the number of games reaches 165, the mean rating curve saturates at a value of 2045.85, earning the neural network an expert rating.

Figure 6. The game reaches this position on move number 8 as the computer (White) initiates a move sequence to go up one piece (10.R:15-24 (f), 10.W:20-11; 11.R:7-16(f), 11.W:27-20-11). The human never recovered from this deficit.

Figure 7. The game reaches this position on move number 19 as the computer (White) earns a king by moving 8-3, leaving the human (Red) in a forced situation to play 5-14(f). The computer's reply 3-10-19(f) putting the computer up by a piece (see Figure 8). At this point the human resigned.

Figure 8. The final position of the game on move number 20.

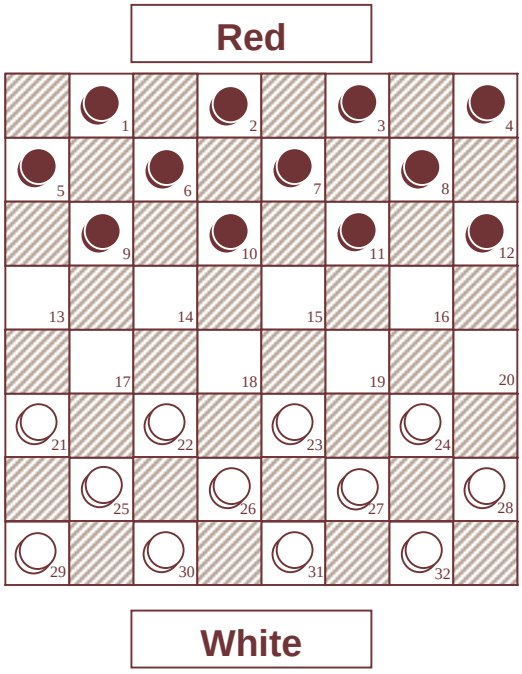
Red
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
White

Figure 1.

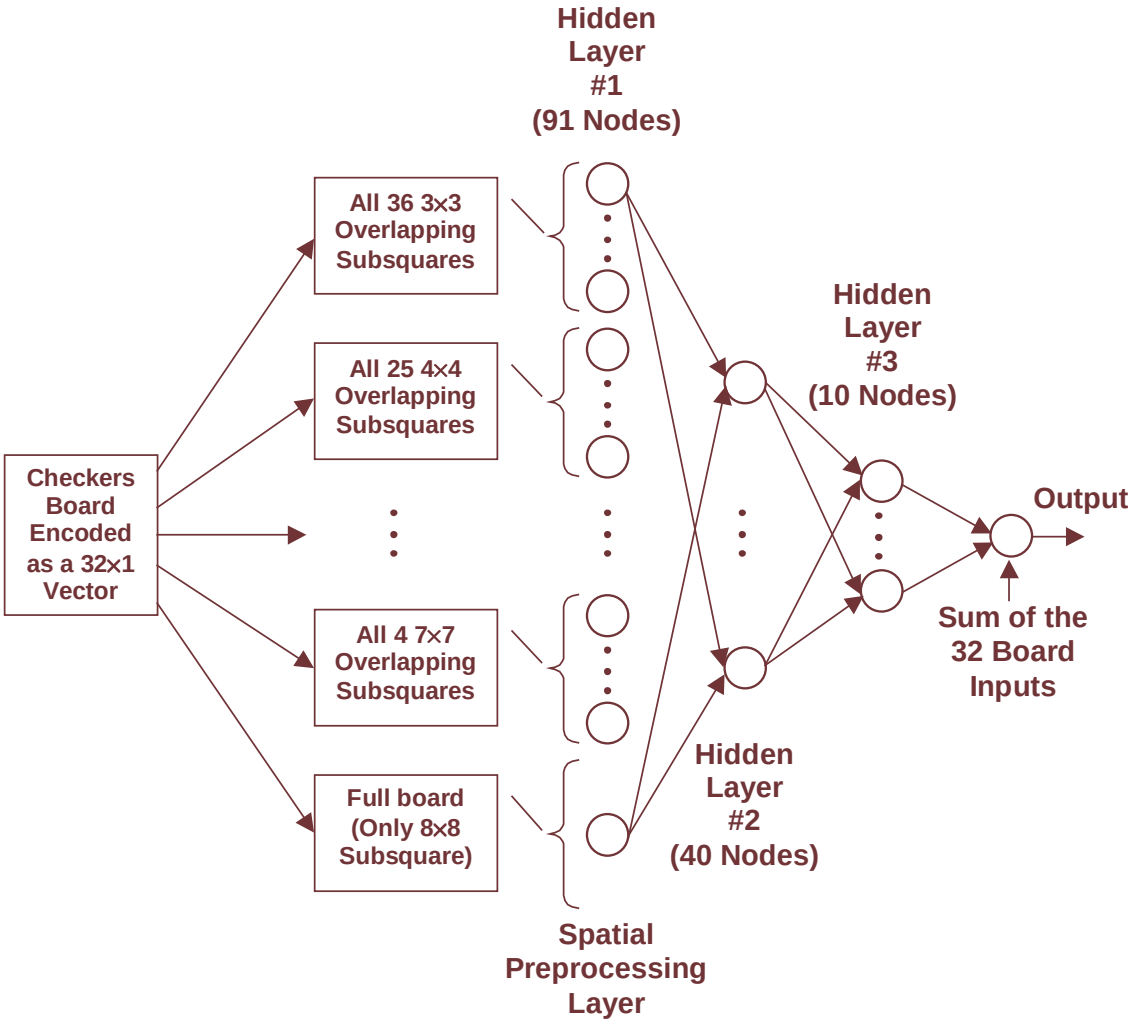
Hidden
Layer
#1
(91 Nodes)
All 36 3×3
Overlapping
Subsquares
All 25 4×4
Overlapping
Subsquares
All 4 7×7
Overlapping
Subsquares
Full board
(Only 8×8
Subsquare)
Checkers
Board
Encoded
as a 32×1
Vector
Hidden
Layer
#3
(10 Nodes)
Output
Sum of the
32 Board
Inputs
Hidden
Layer
#2
(40 Nodes)
Spatial
Preprocessing
Layer

Figure 2.


Rated Room 1:Table 10
red
Decepticon1234!
Resign
Draw
white
David1101
Options
Decepticon12345> hi
David1101> hi
Decepticon12345> gm
David1101> thanks

Figure 3.

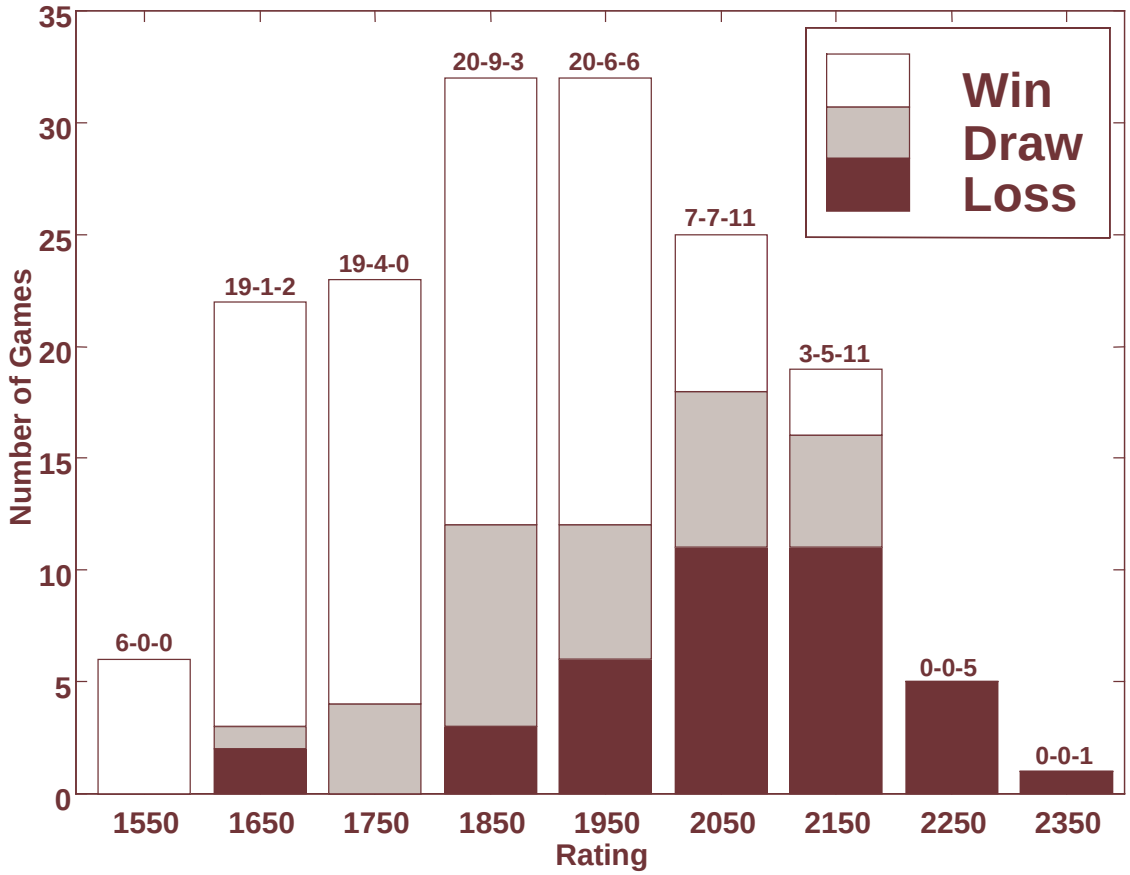
35
30
25
20
15
10
5
0
Number of Games
6-0-0
19-1-2
19-4-0
20-9-3
20-6-6
7-7-11
3-5-11
0-0-5
0-0-1
1550
1650
1750
1850
1950
2050
2150
2250
2350
Rating
Win
Draw
Loss

Figure 4.

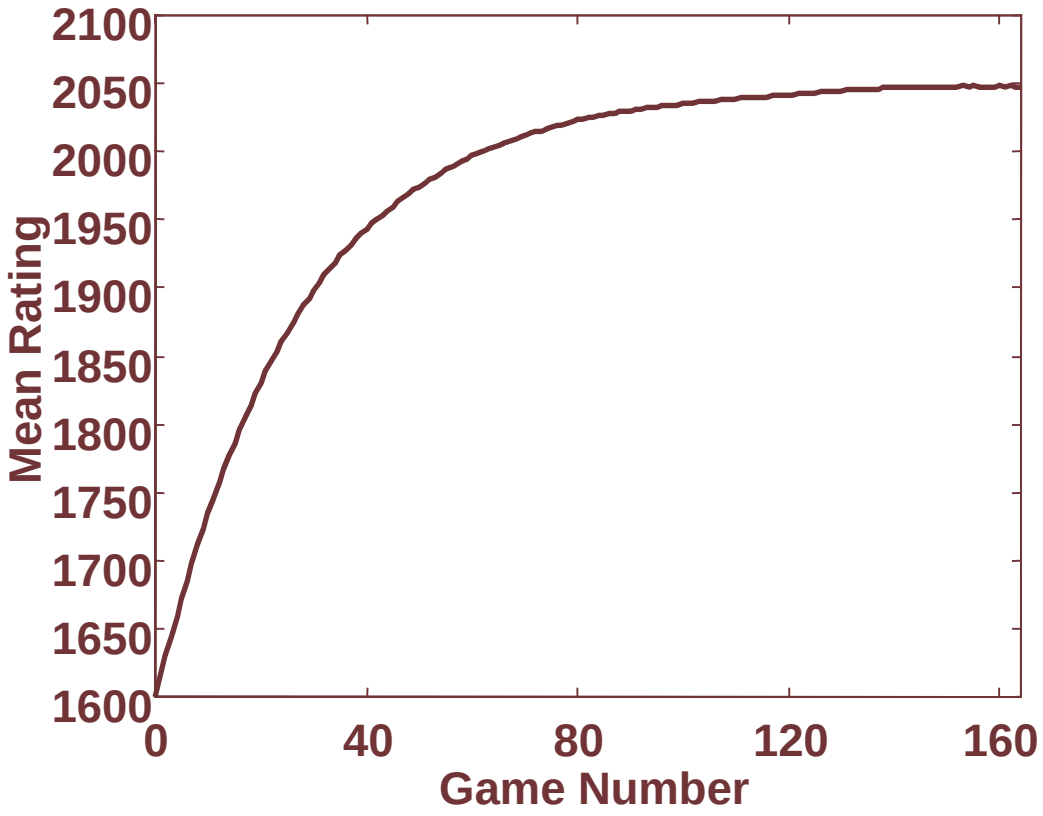
2100
2050
2000
1950
1900
1850
1800
1750
1700
1650
1600
Mean Rating
0
40
80
120
160
Game Number

Figure 5.

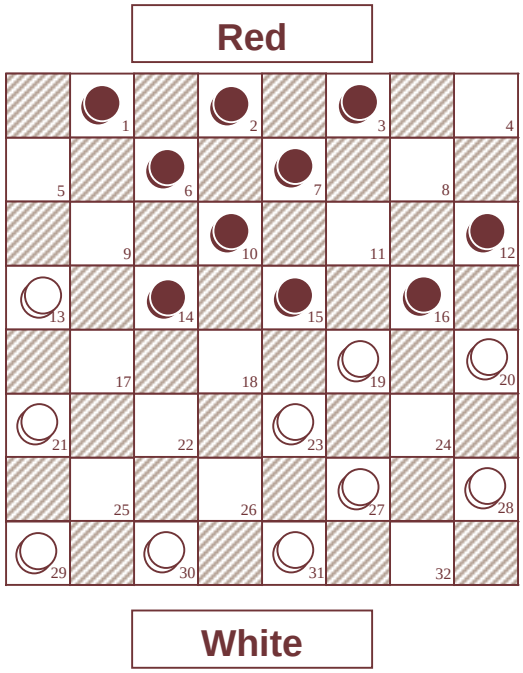
Red
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
White

Figure 6.

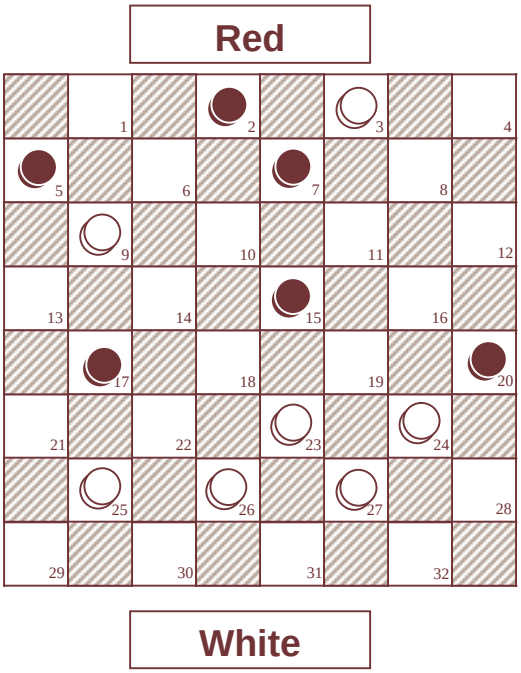
Red
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
White

Figure 7.

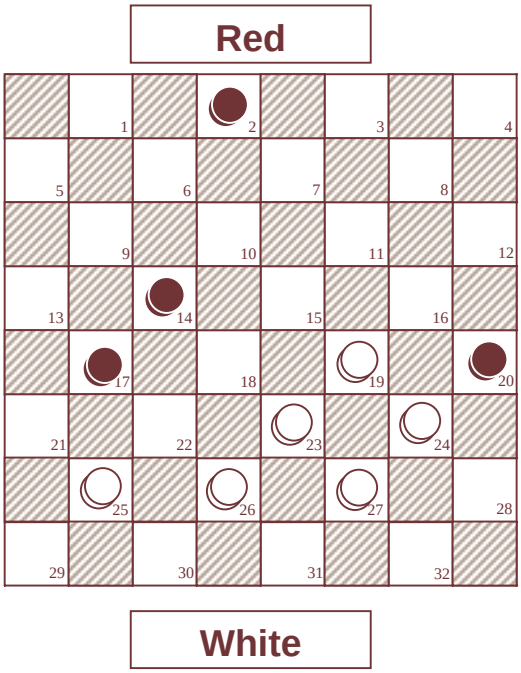
Red
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
White

Figure 8.

Table 1. The relevant categories of player indicated by the corresponding range of rating score.

Class	Rating
Senior Master	2400+
Master	2200-2399
Expert	2000-2199
Class A	1800-1999
Class B	1600-1799
Class C	1400-1599
Class D	1200-1399
Class E	1000-1199
Class F	800-999
Class G	600-799
Class H	400-599
Class I	200-399
Class J	below 200