Anaconda Defeats Hoyle 6-0: A Case Study Competing an Evolved Checkers Program against Commercially Available Software

Kumar Chellapilla
David B. Fogel
Natural Selection, Inc.
3333 N. Torrey Pines Ct., Suite 200
La Jolla, CA 92037
kumar@natural-selection.com
dfogel@natural-selection.com

Abstract - We have been exploring the potential for a coevolutionary process to learn how to play checkers without relying on the usual inclusion of human expertise in the form of features that are believed to be important to playing well. In particular, we have focused on the use of a population of neural networks, where each network serves as an evaluation function to describe the quality of the current board position. After only a little more than 800 generations, the evolutionary process has generated a neural network that can play checkers at the expert level as designated by the U.S. Chess Federation rating system. The current effort reports on a competition between the best-evolved neural network, named "Anaconda," and commercially available software. In a series of six games, Anaconda scored a perfect six wins.

1 Introduction

Checkers is played traditionally on an 8×8 board with squares of alternating colors of red and black (see Fig. 1). There are two players, denoted as "red" and "white." Each side has 12 pieces (checkers) which begin in the 12 alternating squares of same color that are closest to that player's side, with the right-most square on the closest row to the player being left open. The red player moves first and then play alternates between sides. Checkers are allowed to move forward diagonally one square at a time, or, when next to an opposing checker and there is a space available directly behind that opposing checker, by jumping diagonally over an opposing checker. In the latter case, the opposing checker is removed from play. If a jump would in turn place the jumping checker in position for another jump, that jump must also be played, and so forth, until no further jumps are available for that piece. Whenever a jump is available, it must be played in preference to a move that does not jump; however, when multiple jump moves are available, the player has the choice of which jump to conduct, even when one jump offers the removal of more opponent's pieces (e.g., a double jump versus a single jump). When a checker advances to the last row of the board it becomes a king, and can thereafter move diagonally forward or backward. The game ends when a player has no more available moves, which most often occurs by having their last piece removed from the board but can also occur when all existing pieces are trapped, resulting in a loss for the player with no remaining moves and a win for the opponent (the object of the game). The game can also end when one side offers a draw and the other accepts.

Our current effort explores the potential for using neural networks to extract information about how to play expert-level checkers without being offered human expertise about the game in terms of features that would be believed to be important, end game databases, opening moves from grand masters, or similar information. Using a coevolutionary procedure, a population of neural networks has evolved a board evaluation function that can facilitate play at the expert level. We have tested the best-evolved neural network against commercially available software and report the results here. Section 2 provides background on prior efforts in computer checkers. Section 3 details the method we employed. Section 4 describes our results, and Section 5 offers a brief discussion of the implications of the results.

2 Background on Computer Programs for Checkers

There have been many attempts to design programs to play checkers since the late 1950s. These are documented in Schaeffer (1996) and for the sake of space only two will be described here. The current computer world champion checkers program is *Chinook*, designed by Schaeffer et al. at the University of Alberta. The program uses a linear handcrafted evaluation function that considers several features of the game board including: 1) piece count, 2) kings count, 3) trapped kings, 4) turn, 5) runaway checkers (unimpeded path to king);

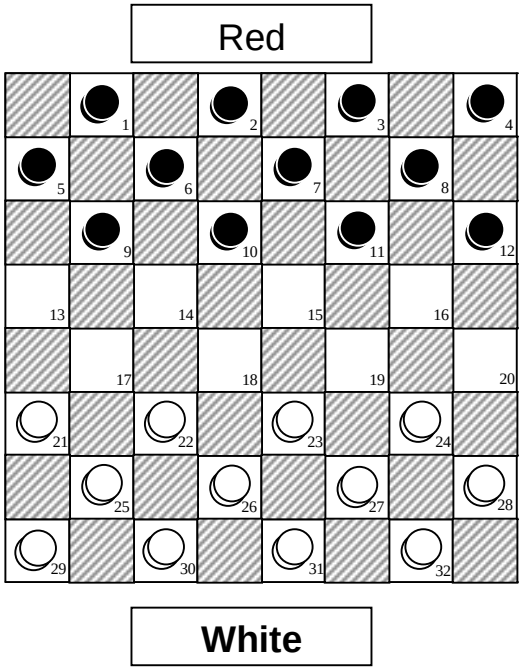

Figure 1. The opening board in a checkers game. Red moves first.

and other minor factors (Schaeffer, 1996, pp. 63-65). In addition, the program has: 1) access to a library of opening moves from games played by grand masters, 2) the complete endgame database for all boards with eight or fewer pieces, and 3) a "fudge factor" that was chosen to favor boards with more pieces over those with fewer pieces. This last facet was included to present more complicated positions to human opponents in the hopes of eliciting a mistake. No machine learning methods have been employed successfully in the development of Chinook. All of its "knowledge" has been programmed by humans. Chinook played to a draw after six games in a 40-game match against the former human world champion Marion Tinsley, the best player to have lived. (From 1950 until his death in 1995, Tinsley won every tournament in which he played and lost only three games.) Tinsley retired the match after these six games for health reasons and died shortly thereafter. Schaeffer (1996, p. 447) offered that Chinook was rated at 2814, with the best human players rated at 2632 and 2625.

In contrast to Chinook, the most well-known effort in designing an algorithm to play checkers is owed to Samuel (1959), which was one of the first apparently successful experiments in machine learning. The method relied in part on the use of a polynomial evaluation function comprising a subset of weighted features chosen from a larger list of possibilities. The polynomial was used to evaluate alternative board positions some number of moves into the future using a minimax strategy. The technique relied on an innovative self-learning procedure whereby one player competed against another. The loser was replaced with a deterministic variant of the winner by altering the weights on the features that were used, or in some cases replacing features that had very low weight with other features. Samuel's program, which also included rote learning of games played by masters, was played against and defeated R.W. Nealey in 1962. IBM Research News described Nealey as "a former Connecticut checkers champion, and one of the nation's foremost players."

The success of Samuel's program was overstated, and continues to be overstated. The 1962 match against Nealey was, in retrospect, pocked with errors on both sides, as has been demonstrated using Chinook to analyze the game (Schaeffer, 1996, pp. 93-97). Moreover, Nealey was not a "former Connecticut champion" as advertised at the time of the match, although he did earn this title later. Nealey defeated Samuel's program in a rematch the next year, and Samuel played four games with his program against both the world champion and challenger in 1966, losing all eight games. As Schaeffer (1996, p. 97) wrote: "The promise of the 1962 Nealey game was an illusion."

In contrast to relying on look-up tables, perfect end game databases, grand master openings, or even features about checker positions that are believed to be important, we considered the possibility of having neural networks learn to play checkers without such knowledge. This would appear to be a necessary precursor to any effort to generate machine intelligence that is capable of solving new problems in new ways. One measure of success is the level of play that can be attained against humans without preprogramming in the requisite knowledge to play well. Another measure concerns how well an evolved neural network can fair against other computer programs. The latter measure is assessed in part here.

3 Method for Evolving Neural Networks for Checkers from Scratch

Each board was represented by a vector of length 32, with each component corresponding to an available position on the board. Components in the vector could take on elements from $\{-K, -1, 0, +1, +K\}$, where K was the evolvable value assigned for a king, 1 was the value for a regular checker, and 0 represented an empty square. The sign of the value indicated whether or not the piece in question belonged to the player (positive) or the opponent (negative). A player's move was determined by evaluating the presumed quality of potential future positions. This evaluation function was structured as a feedforward neural network with an input layer, three hidden layers, and an output node. The second and third hidden layers and the output layer had a fully connected structure, while the connections in the first hidden layer were specially designed to possibly capture spatial information from the board. The nonlinearity function at each hidden and out-

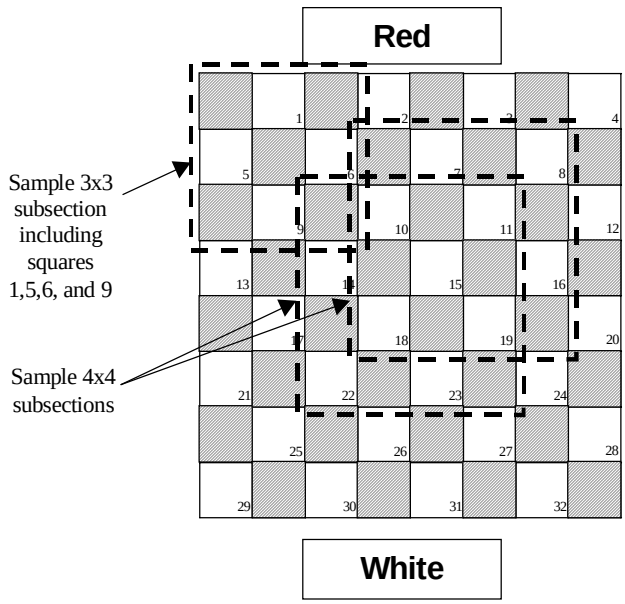

Figure 2. The first hidden layer of the neural networks assigned one node to each possible 3×3, 4×4, 5×5, 6×6, 7×7, and 8×8 subset of the entire board. In the last case, this corresponded to the entire board. In this manner, the neural network was able to invent features based on the spatial characteristic of checkers on the board. Subsequent processing in the second and third hidden layer then operated on the features that were evolved in the first layer.

put node was the hyperbolic tangent (tanh, bounded by ±1) with a variable bias term, although other sigmoidal functions could undoubtedly have been chosen. In addition, the sum of all entries in the input vector was supplied directly to the output node.

The neural architecture used a series of 91 preprocessing nodes that covered $n \times n$ square overlapping subsections of the board. These $n \times n$ subsections were chosen to provide spatial adjacency or proximity information such as whether two squares were neighbors, or were close to each other, or were far apart. All 36 3×3 square subsections of the board were provided as input to the first 36 hidden nodes in the first hidden layer. The following 25 4×4 square subsections were assigned to the next 25 hidden nodes in that layer, and so forth. Fig. 2 shows a sample 3×3 square subsection that contains the state of positions 1, 5, 6, and 9. Two sample 4×4 subsections are also shown. All possible square subsections of size 3 to 8 (the entire board) were given as inputs to the 91 nodes of the first hidden layer. This enables the neural network to generate features from these subsets of the entire board that could then be processed in subsequent hidden layers. Fig. 3 shows the general structure of the "spatial" neural network. At each generation, a player was defined by their associated neural network in which all of the connection weights (and biases) and king value were evolvable.

Note that no effort was made to include specific features that would require human expertise. The only feature of the board that was offered, indirectly, was the piece differential. This resulted from connecting the inputs directly to the output, which essentially counts the number of pieces for each player and indicates the difference. When kings are present, however, this is not always the case because each neural network evolves its own value of K.

When a board was presented to a neural network for evaluation, its scalar output was interpreted as the worth of that board from the position 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 input board. Similarly, the closer the output was to –1.0, the worse the board. All positions that were wins for the player were assigned the value 1.0, and likewise losses were assigned –1.0.

The coevolutionary procedure for neural learning was as follows. A population of 15 neural network strategies, P_i, $i = 1, \ldots, 15$, defined by the weights and biases for each neural network and its associated value of K, was created at random. Weights and biases were sampled uniformly at random over [–0.2, 0.2], with the value of K set initially to 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. To be consistent with the range of initialization, the self-adaptive parameters for weights and biases were set initially to 0.05.

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

$$\sigma'_i(j) = \sigma_i(j) \exp(\tau N_j(0,1)), \qquad j = 1, \ldots, N_w$$
$$w'_i(j) = w_i(j) + \sigma'_i(j) N_j(0,1), \qquad j = 1, \ldots, N_w$$

where N_w is the number of weights and biases in the neural network (here, 5046), $\tau = (2(N_w)^{0.5})^{-0.5} = 0.0839$, and $N_j(0,1)$ is a standard normal random variable sampled anew for every j. The offspring king value was determined by:

$$K_i = K_i + \delta$$

where δ was chosen uniformly at random from {–0.1, 0, 0.1}. For convenience, K' was bounded between [1.0, 3.0].

All parents and offspring competed for survival by playing games and receiving points for their resulting play. Each player in turn played one game against each of five randomly selected opponents from the population (with replacement). In each of these games, the player always played red, whereas the randomly selected opponent always played white. The players earned +1, –2, or 0 points for winning, losing, or playing to a draw, respectively. A draw was declared after 100 moves on each side. In total, there were 150 games per

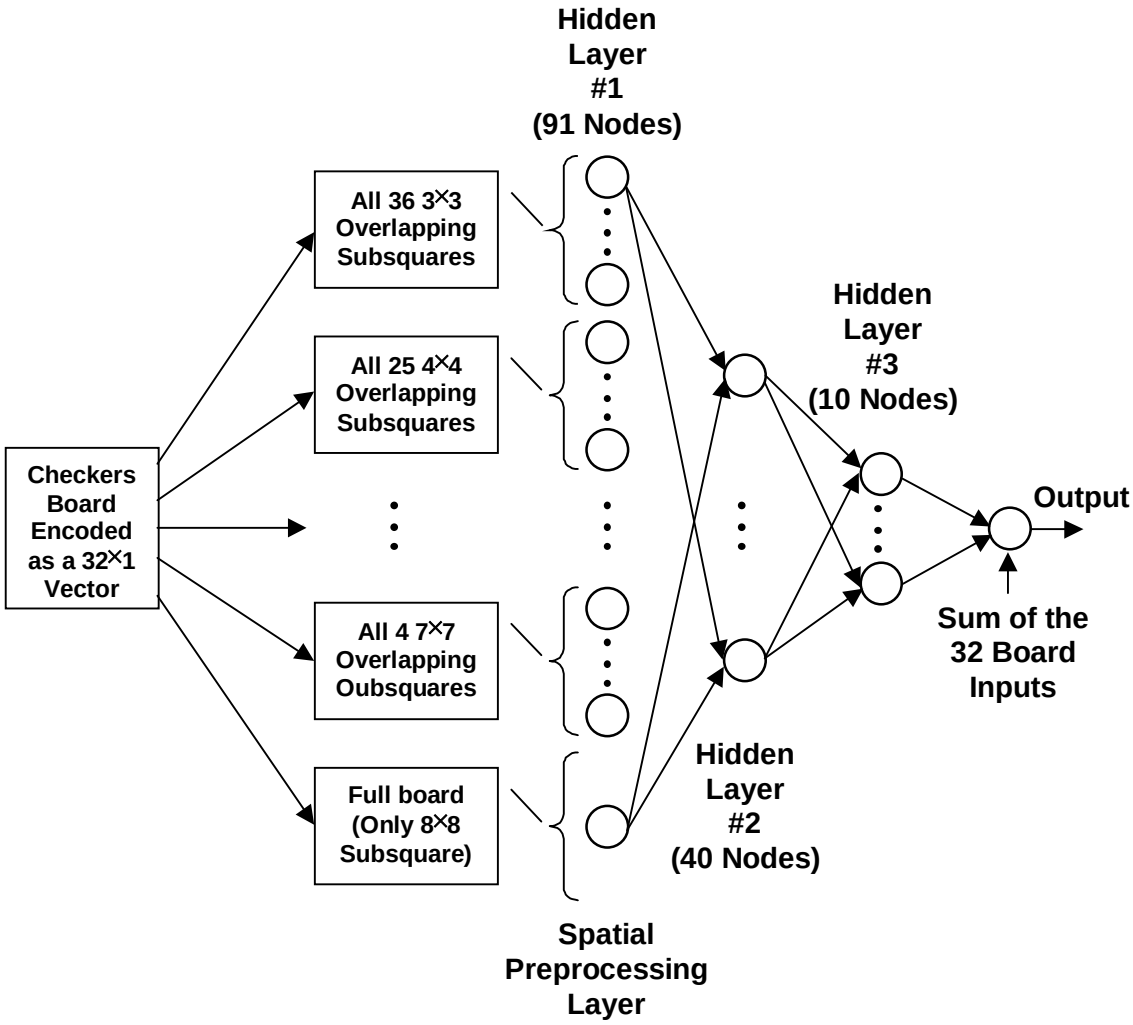

Figure 3. The complete "spatial" neural network used to represent strategies. The network served as the board evaluation function. Given any board pattern as a vector of 32 inputs, the first hidden layer (spatial preprocessing) assigned a node to each possible square subset of the board. The outputs from these nodes were then passed through two additional hidden layers of 40 and 10 nodes, respectively. 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.

generation, with each strategy participating in an average of 10 games. After all games were complete the 15 strategies with the highest point totals were retained as parents for the next generation and the process was iterated.

Games were played using a minimax search at a depth of four ply. The minimax move for each player was determined by selecting the available move that afforded the opponent the opportunity to do the least damage as determined by the network's evaluation. Four ply was chosen to allow for reasonable training time on the 400 MHz Pentium II (30 generations = 7 days). When forced moves were encountered, the ply was increased and the evaluation of a position was postponed until the board was in a "quiescent" state.

The evolutionary process was iterated for 250 generations with the resulting best-evolved neural network being tested against human competitors over the Internet (www.zone.com) (Chellapilla and Fogel, 1999). That neural network was found to perform at the level of "Class A," one step below "expert" (which is in turn one step below "master" and then "grand master"), with a rating of 1930.0. Based on observing the play of the program, we named it "Anaconda" because it often appeared to win its games by restricting the mobility of its opponents. Recall that nowhere in the input to the neural networks was any concept of mobility. To the extent that the evolutionary process generated neural networks that used this feature, it had to first *invent the concept*. After 840 generations, the best-evolved neural network was again played against people at the Internet site and its performance has improved to the expert level (Chellapilla and Fogel, 2000). Of interest here is to test the performance of the best-evolved

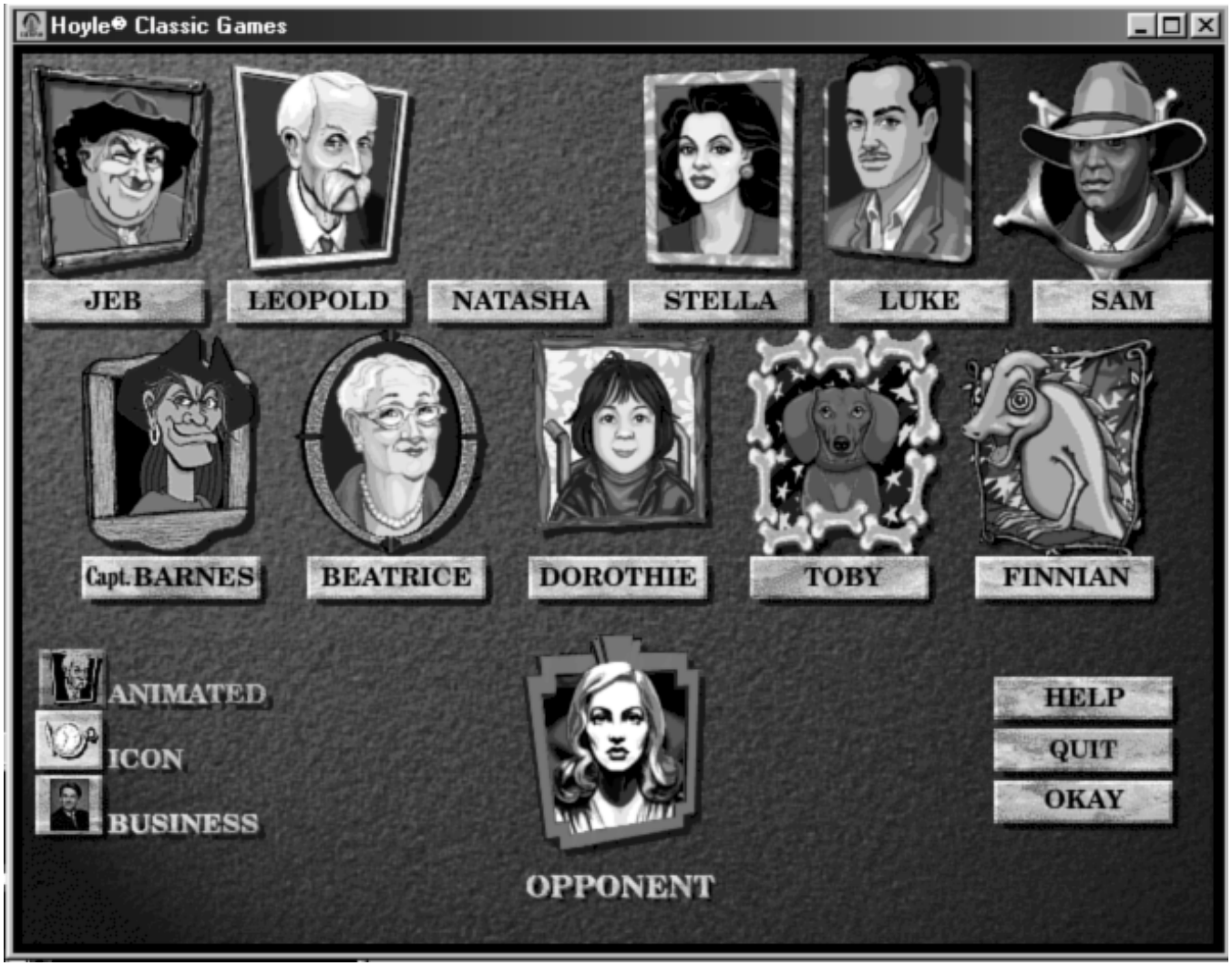

Figure 4. The various characters that are offered for competition in *Hoyle's Classic Games* for checkers. Games were played against Beatrice, Natasha, and Leopold, each of whom is described as having "expert" level play.

neural network at generation 840 against commercially available checkers-playing software.

4 Experiment and Results

The authors were provided a copy of *Hoyle's Classic Games* by Sierra Online for competing in an online checkers tournament. (The results of that tournament will be described elsewhere). The Classic Games CD offers a variety of different opponents with either "average" or "expert" skill at checkers (Fig. 4). A series of six games were played in which Anaconda played one game as red and one game as white against three characters who were designated as experts (Beatrice, Natasha, and Leopold). Anaconda relied mainly on a search ply of six moves. From previous experience with Anaconda at a search ply of eight moves, it is doubtful that a ply of six would provide Anaconda with the ability to play at the "expert" level by U.S. Chess Federation standards. Nevertheless, Anaconda achieved a perfect six wins and no losses or draws in the six games played. A typical game is offered in the Appendix.

5 Discussion

Whereas a great effort has been made to generate commercial chess programs that can play at the grand master level, there has been little comparable effort to do the same for checkers. The performance of the "expert" checkers programs in *Hoyle's Classic Games* does not appear to correlate well with the level of play that would be expected for a player rated above 2000 on the U.S. Chess Federation scale (this is the same scale used for checkers on www.zone.com, with 2000

being the threshold for designating expert-level play). The results of the competition suggest that a program like Anaconda could possess some commercial market value, particularly because its level of play should be tunable based on the number of moves that it looks ahead. Experiments to determine the effects of variable ply look-ahead on the level of attained performance remain for future work.

The experiment performed here, as well as the previous results documented in Chellapilla and Fogel (1999, 2000), demonstrate an ability for an evolutionary algorithm to start with essentially no preprogrammed information in the game of checkers (except for the possibility of using piece differential) and learn, over successive generations, how to play at a level that is challenging not only to humans, but to some of the programs they have created.

Early speculation on the potential success of this sort of effort was entirely negative. A. Newell, in Minsky (1961) offered: "It is extremely doubtful whether there is enough information in 'win, lose, or draw' when referred to the whole play of the game to permit any learning at all over available time scales." Minksy (1961) noted being in "complete agreement" with Newell. The challenge taken up in the protocol described here goes further by not offering feedback except after a series of games, where a neural network does not get to know which game was a victory and which was a defeat. The authors hope the results provide impetus for increased attention on the robust and useful application of coevolution to difficult problems where human expertise may be insufficient or altogether lacking.

Acknowledgments

The authors thank Paul Darwen and the reviewers for the comments and criticisms.

References

K. Chellapilla and D. B. Fogel, "Evolution, Neural Networks, Games, and Intelligence," *Proc. IEEE*, Vol. 87, No. 9, pp. 1471-1498, 1999.

K. Chellapilla and D. B. Fogel, "Evolving an Expert Checkers Playing Program without Using Human Expertise," *IEEE Trans. Pattern Analysis and Machine Intelligence*, in review.

M.L. Minsky, "Steps toward artificial intelligence," *Proc. IRE*, Vol. 49, No. 1, pp. 8-30, 1961.

A.L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, Vol. 3, No. 3, pp. 210-219, 1959.

J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers*, Springer, Berlin, 1996.

Appendix: Moves for Game 1 — Anaconda vs. "Beatrice" from *Hoyle's Classic Games.*

Beatrice moves first as Red. Anaconda moves second as White. Anaconda takes an early lead in moves 20-22, gives up pieces between moves 28-30, and comes back for a win in move 58 (a double jump). After Red plays 22-25 on move 60, Anaconda sees a way to force a win: game over.

1.R:11-15	1.W:22-18
2.R:15-22(f)	2.W:26-17
3.R:9-14	3.W:24-20
4.R:8-11	4.W:27-24
5.R:11-15	5.W:32-27
6.R:15-18	6.W:30-26
7.R:4-8	7.W:25-22
8.R:18-25(f)	8.W:29-22(f)
9.R:8-11	9.W:17-13
10.R:11-15	10.W:24-19
11.R:15-24(f)	11.W:28-19(f)
12.R:7-11	12.W:27-24
13.R:11-15	13.W:31-27
14.R:5-9	14.W:20-16
15.R:1-5	15.W:16-11
16.R:3-7	16.W:11-8
17.R:7-11	17.W:8-4
18.R:11-16	18.W:4-8
19.R:16-20	19.W:8-11
20.R:15-18	20.W:22-15(f)
21.R:2-7	21.W:11-2(f)
22.R:12-16	22.W:19-12(f)
23.R:10-19-28(f)	23.W:12-8
24.R:28-32	24.W:8-4
25.R:32-28	25.W:23-19
26.R:14-18	26.W:4-8
27.R:28-24	27.W:27-23
28.R:18-27	28.W:2-7
29.R:24-15(f)	29.W:7-10
30.R:15-19	30.W:10-1(f)
31.R:9-14	31.W:1-6
32.R:27-32	32.W:6-10
33.R:14-18	33.W:8-12
34.R:32-27	34.W:12-8
35.R:19-16	35.W:10-14
36.R:27-31	36.W:14-23(f)
37.R:31-22(f)	37.W:8-3
38.R:16-11	38.W:21-17
39.R:11-15	39.W:23-18
40.R:15-11	40.W:18-25(f)
41.R:11-15	41.W:25-22
42.R:15-11	42.W:22-18
43.R:20-24	43.W:18-23
44.R:24-28	44.W:23-18
45.R:28-32	45.W:18-23
46.R:11-16	46.W:3-7

47.R:16-12	47.W:7-11
48.R:32-28	48.W:17-14
49.R:28-24	49.W:14-10
50.R:24-28	50.W:10-6
51.R:28-24	51.W:11-15
52.R:5-9	52.W:6-2
53.R:9-14	53.W:2-6
54.R:14-17	54.W:13-9
55.R:12-8	55.W:6-10
56.R:8-4	56.W:9-6
57.R:4-8	57.W:15-11
58.R:8-15(f)	58.W:10-19-28(f)
59.R:17-22	59.W:6-1
60.R:22-25	