Using Evolutionary Programming to Create Neural Networks that are Capable of Playing Tic-Tac-Toe

David B. Fogel
ORINCON Corporation
9363 Towne Centre Dr.
San Diego, CA 92121
fogel@sunshine.ucsd.edu

Abstract - All intelligent systems are evolutionary. Simulating evolution provides a method for generating machine intelligence. To date, there have been three main efforts in simulating evolution: genetic algorithms, evolution strategies, and evolutionary programming. The current research focuses on the use of evolutionary programming for adapting the design and weights of a multi-layer feed forward perceptron in the context of machine learning. Specifically, it is desired to evolve the structure and weights of a single-hidden layer perceptron such that it can achieve a high level of play in the game tic-tac-toe without the use of heuristics or credit assignment algorithms. Conclusions from the experiments are offered regarding the relative importance of specific mutation operations, the necessity for credit assignment procedures and the efficiency and effectiveness of evolutionary search.

I. Introduction

Intelligence is that property which allows a system to adapt its behavior to meet desired goals in a range of environments. There are three naturally-occurring organizational forms of intelligence: phylogenetic (arising within the line of descent), ontogenetic (arising within the individual) and sociogenetic (arising within the group) [1,2]. Phylogenetic learning is the most ancient form of intelligence and gave rise to ontogenetic and sociogenetic forms. All intelligent systems are evolutionary. Each possesses a reservoir of learned behavior and a unit of mutability [2]. It is natural to simulate evolutionary processes in order to create machine intelligence.

The biological foundation for such simulations is the neo-Darwinian paradigm [3,4]. This argument asserts that the history of life can be accounted for by processes acting on and within populations and species. These processes are reproduction, mutation, competition and selection. Reproduction is an obvious property of all extant species. Mutations in any positively entropic system must occur. Competition is a natural consequence of existing populations expanding to fill a finite resource space. Selection is the result of competition for the finite available resources. Under such conditions, evolution becomes an inevitable process.

Mayr [3] succinctly summarizes some of the more important characteristics of the neo-Darwinian paradigm. These include:

1. The individual is the primary target of selection,
2. Genetic variation is largely a chance phenomenon and stochastic processes play a significant role in evolution,
3. Genotypic variation is largely a product of recombination and "only ultimately of mutation,"
4. "Gradual" evolution may incorporate phenotypic discontinuities,
5. Not all phenotypic changes are necessarily the consequences of *ad hoc* natural selection,
6. Evolution means changes in adaptation and diversity and not merely a change in gene frequencies, and
7. Selection is probabilistic, not deterministic.

Simulations of evolution, whether they are implemented as methods of machine learning or simply to solve a specific problem, should rely on these foundations.

II. Previous Efforts in Simulated Evolution

Simulated evolution has a long history. Speculation on the evolutionary nature of intelligence goes back at least to Cannon [5] and Turing [6], with the first simulations being performed by Fraser [7,8], Friedberg [9], Friedberg et al. [10], Bremermann [11], Fogel and colleagues [12,13], Schwefel and colleagues [14,15], and Holland and colleagues [16,17]. More complete descriptions of efforts in the field can be found in [18-21,27]. There are three widely researched paradigms in simulated evolution: genetic algorithms, evolution strategies and evolutionary programming. These approaches can be compared and contrasted.

The genetic algorithm and evolutionary programming have some obvious similarities. Both algorithms operate

on a population of candidate solutions, both algorithms subject these solutions to modifications, and both algorithms employ a selection criterion to determine which solutions to maintain for future generations. As proposed in [17], the genetic algorithm differs from evolutionary programming in the following regards:

1. Genetic algorithms use a coding (e.g., a bit string) of the parameters to be evolved, not the parameters themselves [22],

2. The number of offspring to be created from each parent is exponentially related to the parent's fitness relative to all other members of the current population [17, pp. 87-88], and

3. Parents create offspring through the use of genetic operators such as one-point crossover and inversion [17, pp. 89-120].

In contradistinction, in evolutionary programming [18]:

1. The representation for a problem follows in a top-down fashion from the problem. Rather than try to fit a single coding structure to every problem, each problem is regarded as being unique,

2. The number of offspring per parent is generally unimportant and successful simulations need not create more than a single offspring per parent, and

3. Offspring are created through various mutation operations that follow naturally from the chosen problem representation. No emphasis is put on specific genetic operations such as crossover and inversion. Selection is then made a probabilistic function of fitness.

Many of Holland's [17] original proposals have undergone significant revision since first defined. Much of the current research in genetic algorithms has forgone the use of bit strings (e.g. [23]). Experiments in [24] have indicated the inferiority of the one-point crossover and two-point crossover to uniform crossover, although these conclusions have received criticism in [25]. Other research [26] has indicated a greater role for mutation in evolutionary search than was admitted in [17], and illustrated cases where crossover can be detrimental to search.

The evolution strategies paradigm [14] is very similar to that of evolutionary programming. Both methods operate on a population of solutions, subject those solutions to changes through random mutation and compete existing solutions with respect to an objective function. In real-valued parameter optimization, both methods typically apply Gaussian perturbations to all components [27,28] and extensions to self-adapting variances of these perturbations have been made [27,29]. There is no requirement for specific mutation operations that follow the form of genetic transfer in biota, although Schwefel [27] has experimented with various forms of recombination. Evolutionary programming and evolution strategies both emphasize phenotypic changes [18,30].

III. Method & Materials

The current experiments focus on the use of evolutionary programming. Consider the problem of evolving multi-layered feed forward perceptrons capable of playing tic-tac-toe. The game is well-known but will be described in detail for completeness. There are two players and a three by three grid. Initially, the grid is empty. Each player moves in turn by placing a marker in an open square. By convention, the first player's marker is "X" and the second player's marker is "O." The first player moves first. The object of the game is to place three markers in a row. This results in a win for that player and loss for the opponent. Failing a win, a draw may be earned by preventing the opponent from placing three markers in a row. It can be shown by enumerating the game tree that at least a draw can be forced by the second player.

Attention will be devoted to evolving a strategy for the first player (an equivalent procedure could be used for the second player). A suitable coding structure must be selected. It must receive a board pattern as input and yield a corresponding move as output. The coding structure utilized in these experiments was a multi-layered feedforward perceptron (see Figure 1). Each hidden or output node performs a sum of the weighted input strengths, subtracts off an adaptable bias term and passes the result through a sigmoid filter $1/(1+\exp(-x))$. Only a single hidden layer was incorporated. This architecture was selected because: (1) variations can be shown to be universal function approximators [31], (2) it was believed to be adequate for the task, (3) the response to any stimulus could be evaluated rapidly, and (4) the extension to multiple hidden layers is obvious.

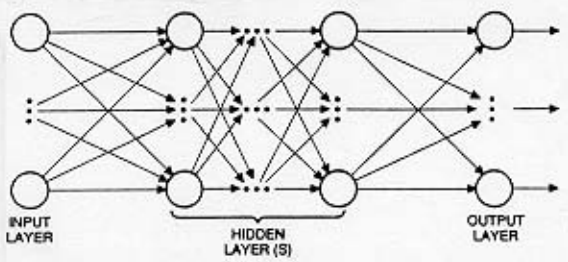

Figure 1. A Multi-Layer Feed Forward Perceptron

There were nine input and output units. Each corresponded to a square in the grid. An "X" was denoted by the value 1.0, an "O" was denoted by the value -1.0, and an open space was denoted by the value 0.0. A move was determined by presenting the current board pattern to the network and examining the relative strengths of the nine

output nodes. A marker was placed in the empty square with the maximum output strength. This procedure guaranteed legal moves. The output from nodes associated with squares in which a marker had already been placed was ignored. No selection pressure was applied to drive the output from such nodes to zero.

The initial population consisted of 50 parent networks. The number of nodes in the hidden layer was chosen at random in accordance with a uniform distribution over the integers [1,...,10]. The initial weighted connection strengths and bias terms were randomly distributed according to a uniform distribution ranging over [-0.5,0.5]. A single offspring was copied from each parent and modified by two modes of mutation:

1. All weight and bias terms were perturbed by adding a Gaussian random variable with zero mean and a standard deviation of 0.05, and

2. With a probability of 0.5, the number of nodes in the hidden layer was allowed to vary. If a change was indicated, there was an equal likelihood that a node would be added or deleted, subject to the constraints on the maximum and minimum number of nodes (10 and one, respectively). Nodes to be added were initialized with all weights and the bias term being set equal to 0.0.

A rule-based procedure that played nearly perfect tic-tac-toe was implemented in order to evaluate each contending network. The execution time with this format was linear with the population size and provided the opportunity for multiple trials and statistical results. The evolving networks were allowed to move first in all games. The first move was examined by the rule-base with the eight possible second moves being stored in an array. The rule-base proceeded as follows:

1. From the array of all possible moves, select a move that has not yet been played.

2. For subsequent moves:

a) with a 10 percent chance, move randomly, else

b) if a win is available, place a marker in the winning square, else

c) if a block is available, place a marker in the blocking square, else

d) if two open squares are in line with an "O", randomly place a marker in either of the two squares, else

e) randomly move in any open square.

3. Continue with (2) until the game is completed.

4. Continue with (1) until games with all eight possible second moves have been played.

The 10 percent chance for moving randomly was incorporated to maintain a variety of play in an analogous manner to a persistence of excitation condition. This feature and the restriction that the rule-base only looks one move ahead makes the rule-base nearly perfect, but beatable.

Each network was evaluated over four sets of these eight games. The payoff function varied in several sets of experiments. Due to space limitations, only the experiments with the payoff function {+1, -10, 0} will be described here, where the entries are the payoffs for winning, losing and playing to a draw, respectively. Other results are described in [18]. The maximum possible score over any four sets of games was 32. But a perfect score in any generation did not necessarily indicate a perfect algorithm because of the random variation in play generated by step 2a, above. After competition against the rule-base was completed for all networks in the population, a second competition was held in which each network was compared to 10 other randomly chosen networks. If the score of the chosen network was greater than or equal to its competitor, it received a win. Those networks with the greatest number of wins were retained to be parents of the next generation. Twenty trials were conducted. Evolution was halted after 800 generations in each trial.

IV. Experimental Results

The mean learning rate over all 20 trials when using {+1, -10, 0} is indicated in Figure 2. There is an initially rapid increase in performance as strategies that lose are quickly purged from the evolving population. After this first-order condition is satisfied, optimization continues to sort out strategies with the greatest potential for winning rather than drawing. The approximate 95 percent confidence limits around the mean are close to the average performance across all 20 trials. Figure 3 indicates the tree of possible games when competing the best evolved network from the first trial against the rule-based player, omitting random moves from step 2a. The tree is typical of the results across all trials. This selected network possessed 10 hidden nodes. It does not force a win in any branch of the tree when the rule-base makes no errors. But it also never loses.

V. Conclusions

These results and those offered in [18] indicate a capability for general problem solving. No information regarding the object of the game was offered to the evolutionary program. No hints regarding appropriate moves were given, nor were there any attempts to assign values to various board patterns. The final outcome (win, lose, draw) was the only available information regarding the quality of play. Heuristics regarding the environment were limited to:

1. There were nine inputs,

2. There were nine outputs, and

3. Markers may only be placed in empty squares.

Evolutionary programming was able to adjust the architecture and connections of the single hidden layer

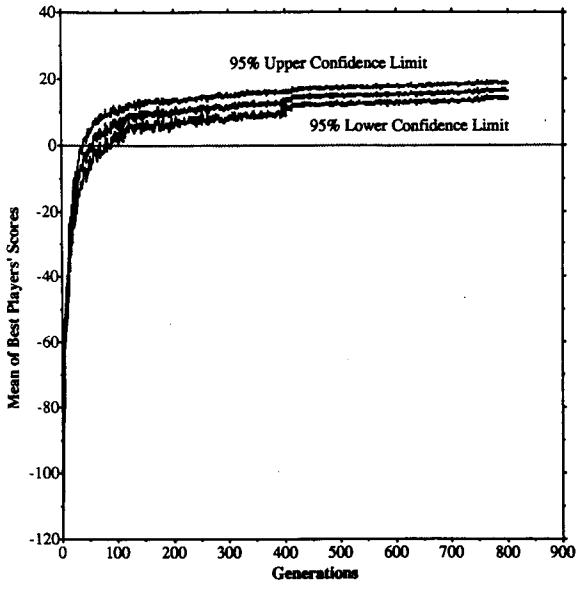

Figure 2. The Mean and Upper/Lower Confidence Limits of the Best Players' Scores Averaged Over All 20 Trials Using the Payoff Function {+1, -10, 0}.

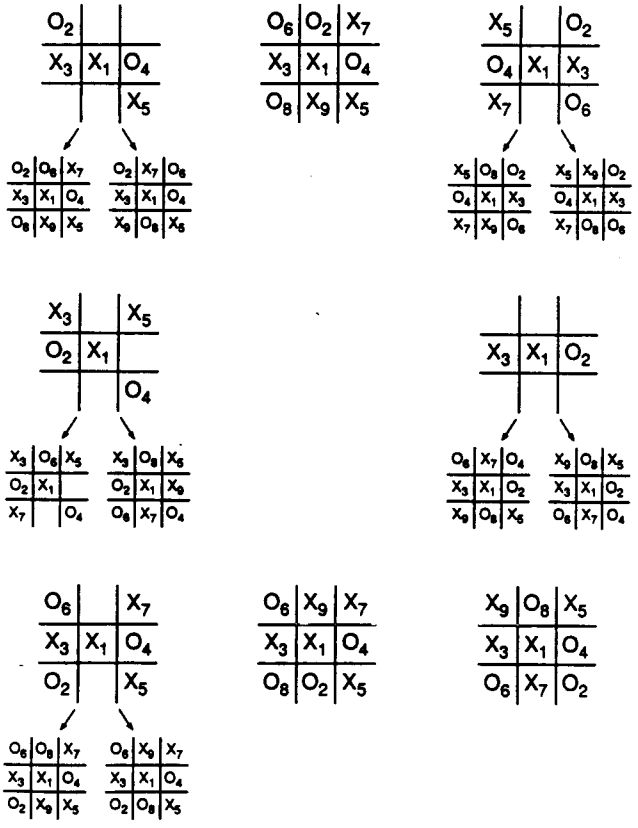

Figure 3. The Tree of All Possible Games When the Best Network from Trial #1 is Played Against the Rule-Based Opponent. The Network Plays as "X", the Rule-Base Plays as "O". The Subscripts Indicate the Order of Play.

perceptrons in order to adapt behavior in light of the given goal. While single hidden layer networks were sufficient for tic-tac-toe, other problems may be more easily addressed by more complex stimulus-response structures. But the evolutionary procedure is robust and no fundamental alteration of the basic algorithm is required to address problems of arbitrary complexity.

The results also illustrate how connectionist and rule-based systems can be supplemented with evolutionary learning. Testing the effectiveness of the evolving networks provided an absolute measure of performance. Evolution may surpass the performance of the available rule-base and then continue to generate even more effective strategies.

If it were desired to create a truly perfect tic-tac-toe algorithm using neural coding structures, a population of evolved first players could be put in competition against a population of evolved second players, with evolutionary programming essentially generating a minimax search over both populations. Alternatively, a single coding structure could be required to act both as first and second player. Or, following [32,33], the coding structure could be an explicit rule base of conditions and actions. Evolutionary programming does not restrict the form of the stimulus-response coding; the researcher is free to select a structure that appears most suitable for the task.

As recently as [34], it was still claimed that mutation does not generally advance a search for optimal solutions and that such searches require carefully structured rules of recombination. These claims are not correct. As indicated in the above experiments, abstractions of specific genetic mechanisms such as crossover, inversion and other recombinatory methods are not required for successful evolution. This follows similar evidence presented in other evolutionary studies ([20,28,35] and others).

In expressing his belief that intelligent, conscious systems could not arise strictly through evolution, Penrose [36] stated, "Any selection process of [the type illustrated by evolution] could act only on the *output* of the algorithms and not directly on the ideas underlying the actions of the algorithms. This is not simply extremely inefficient; I believe that it would be totally unworkable." The evidence reported here indicates that not only is such a method workable, it is also efficient.

ACKNOWLEDGMENTS

The author would like to thank W. Atmar, T. Bäck, L. J. Fogel, B. Scurlock, A. V. Sebald, and P. K. Simpson for their helpful comments and suggestions regarding this research.

REFERENCES

[1 N. Wiener, *Cybernetics*, Part II, MIT Press, Cambridge, MA, 1961.

[2] J.W. Atmar, "Speculation on the Evolution of Intelligence and Its Possible Realization in Machine Form," Doctoral Dissertation, New Mexico State University, 1976.

[3] E. Mayr, *Toward a New Philosophy of Biology: Observations of an Evolutionist*, Belknap Press, Cambridge, MA, 1988.

[4] A. Hoffman, *Arguments on Evolution: A Paleontologist's Perspective*, Oxford Univ. Press, New York, 1989.

[5] W. D. Cannon, *The Wisdom of the Body*, Norton & Co., New York, 1932.

[6] A.M. Turing, "Computing machinery and intelligence," *Mind*, Vol. 59, pp. 433-460, 1950.

[7] A.S. Fraser, "Simulation of genetic systems by automatic digital computers. I. Introduction," *Australian J. of Biol. Sci.*, Vol. 10, pp. 484-491, 1957.

[8] A.S. Fraser, "Simulation of genetic systems by automatic digital computers. II. Effects of linkage on rates of advance under selection," *Australian J. of Biol. Sci.*, Vol. 10, pp. 492-499, 1957.

[9] R.M. Friedberg, "A learning machine: Part I," *IBM J. of Res. & Dev.* Vol. 2, pp. 2-13, 1958.

[10] R.M. Friedberg, B. Dunham, and J. H. North, "A learning machine: Part II," *IBM J. of Res. & Dev.*, Vol. 3, pp. 282-287, 1959.

[11] H.J. Bremermann, "The evolution of intelligence. The nervous systems as a model of its environment," Technical Report No. 1, Contract No. 477(17), Dept. of Mathematics, Univ. of Washington, Seattle, July, 1958.

[12] L.J. Fogel, "Autonomous automata," *Industrial Research*, Vol. 4, pp. 14-19, 1962.

[13] L.J. Fogel, A.J. Owens and M.J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley & Sons, New York, 1966.

[14] H.-P. Schwefel, "Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik," Diploma Thesis, Tech. Univ. of Berlin, 1965.

[15] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann-Holzboog Verlag, Stuttgart, 1973.

[16] J.H. Holland, "Adaptive plans optimal for payoff-only environments," *Proc. of 2nd Hawaii Int. Conf. on System Sciences*, pp. 917-920, 1969.

[17] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, MI, 1975.

[18] D.B. Fogel, "Evolving Artificial Intelligence," Doctoral Dissertation, Univ. Cal. San Diego, 1992.

[19] D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[20] D.B. Fogel and W. Atmar (eds.), *Proc. of the First Annual Conference on Evolutionary Programming*, Evolutionary Programming Society, La Jolla, CA, 1992.

[21] H.-P. Schwefel and R. Männer, *Proc. of the First Conf. on Parallel Problem Solving from Nature*, Springer, Berlin, 1991.

[22] D.E. Goldberg and C.H. Kuo, "Genetic algorithms in pipeline optimization," *J. Comp. Civ. Eng.*, Vol. 1:2, pp. 128-141, 1987.

[23] L. Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[24] G. Syswerda, "Uniform crossover in genetic algorithms," *Proc. of the Third Int. Conf. on Genetic Algorithms*, J.D. Schaffer (ed.), George Mason Univ., pp. 2-9, 1991.

[25] R. Das and D. Whitley, "The only challenging problems are deceptive: Global search by solving order-1 hyperplanes," *Proc. of the Fourth Int. Conf. on Gen. Algs.*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, pp. 166-173, 1991.

[26] J.D. Schaffer and L.J. Eshelman, "On crossover as an evolutionarily viable strategy," *Proc. of the Fourth Int. Conf. on Gen. Algs.*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, pp. 61-68, 1991.

[27] H.-P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley, Chichester, 1981.

[28] D.B. Fogel and J.W. Atmar, "Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems," *Biol. Cyb.*, Vol. 63, pp. 111-114, 1990.

[29] D.B. Fogel, L.J. Fogel, W. Atmar and G.B. Fogel, "Hierarchic methods of evolutionary programming," *Proc. of the First Ann. Conf. on Evol. Prog.*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 175-182, 1992.

[30] T. Bäck and F. Hoffmeister, "Extended selection mechanisms in genetic algorithms," *Proc. of the Fourth Intern. Conf. on Gen. Algs.*, R.K. Belew and L.B. Booker (eds.), Morgan Kaufmann, pp. 92-99, 1991.

[31] A.R. Barron, "Statistical properties of artificial neural networks," *Proc. of the 28th Conf. on Decision and Control*, Tampa, FL, pp. 280-285, 1989.

[32] S.H. Rubin, "Case-based learning: A new paradigm for automated knowledge acquisition," *ISA Transactions*, Vol. 31:2, pp. 181-209, 1992.

[33] D.B. Fogel, "An evolutionary approach to representation design," *Proc. of the First Ann. Conf. on Evol. Prog.*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 163-168, 1992.

[34] J.H. Holland, "Genetic algorithms," *Scientific American*, July, 1992.

[35] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," *Proc. of the Fourth Intern. Conf. on Gen. Algs.*, R. K. Belew and L. B. Booker (eds.), Morgan Kaufmann, pp. 2-9, 1991.

[36] R. Penrose, *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*, Penguin Books, New York, 1989.