# Evolution, Neural Networks, Games, and Intelligence

Kumar Chellapilla
University of California at San Diego
Dept. Electrical and Computer Engineering
La Jolla, CA  92093
kchellap@ece.ucsd.edu

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

## Abstract

Intelligence pertains to the ability to make appropriate decisions in light of specific goals and to adapt behavior to meet those goals in a range of environments. Mathematical games provide a framework for studying intelligent behavior in models of real-world settings or restricted domains. The behavior of alternative strategies in these games is defined by each individual's stimulus-response mapping. Limiting these behaviors to linear functions of the environmental conditions renders the results to be little more than a façade: Effective decision making in any complex environment almost always requires nonlinear stimulus-response mappings. The obstacle then comes in choosing the appropriate representation and learning algorithm. Neural networks and evolutionary algorithms provide useful means for addressing these issues. This paper describes efforts to hybridize neural and evolutionary computation to learn appropriate strategies in zero- and nonzero-sum games, including the iterated prisoner's dilemma, tic-tac-toe, and checkers. With respect to checkers, the evolutionary algorithm was able to discover a neural network that can be used to play at a near-expert level without injecting expert knowledge about how to play the game. The implications of evolutionary learning with respect to machine intelligence are also discussed. It is argued that evolution provides the framework for explaining naturally occurring intelligent entities and can be used to design machines that are also capable of intelligent behavior.

**Keywords**: evolutionary computation, neural networks, artificial intelligence, computational intelligence, prisoner's dilemma, tic-tac-toe, checkers

## I. Introduction

One of the fundamental mathematical constructs is *the game*. Formally, a game is characterized by sets of rules that govern the "behavior" of the individual players. This behavior is described in terms of stimulus-response: The player allocates resources in response to a particular situation. Each opportunity for a response is alternatively called a *move* in the game. Payoffs accrue to each player in light of their allocation of available resources, where each possible allocation is a *play* in the game. As such, the payoffs determine whether the game is competitive (payoffs for players vary inversely), cooperative (payoffs for players vary directly), or neutral (payoffs for players are uncorrelated). This last category is often used to describe the moves of a single player acting against *nature*, where nature is not considered to have any intrinsic purpose and indeed the meaning of a *payoff* to nature is not well defined. Thus the concept of gaming is very flexible and can be used to treat optimization problems in economics [1], evolution [2], social dilemmas [3], and the more conventional case where two or more players engage in competition (e.g., chess, bridge, or even tank warfare).

Whereas the rules dictate what allocations of resources are available to the players, *strategies* determine how those plays in the game will be undertaken. The value of a strategy depends in part on the type of game being played (competitive, cooperative, neutral), the strategy of the other player(s), and perhaps exogenous factors such as environmental noise or observer error (i.e., misinterpreting other players' plays).

The fundamentals of game theory were first laid out in [1]. There are many subtleties to the concept of a game and for sake of clarity and presentation we will focus here on the rather simplified case where two players engage in a series of moves and face a finite range of available plays at each move. In addition, all of the available plays will be known to both players. These are considerable simplifications as compared to real-world settings, but the essential aspects of the game and its utility as a representation of the fundamental nature of purpose-driven decision makers that interact in an environment should be clear.

In zero-sum games, where the payoff that accrues to one player is taken away from the other (i.e., the game is necessarily competitive), a fundamental strategy is to minimize the maximum damage that the opponent can do on any move. In other words, the player chooses a play such that they guarantee a certain minimum expected payoff. When adopted by both players, this minimax strategy [1] corresponds to the *value* of a play, or the value of the game. This is an essential measure for if the value of a game is positive for one player in a zero-sum game then they will always be able to assure a minimum expected payoff that is greater than their opponent's corresponding negative value.

But what if the strategies adopted are not minimax? Certainly not all players seek to minimize the maximum damage that an opponent can do. Some players are risk takers, rather than being risk adverse. Consider the case where you play a game against an opponent where you each have two plays: *A* or *B*. If you both play *A* then you will receive $1. If you play *A* and they play *B*, you will receive $2. Alternatively, if you play *B* and they play *A* you will receive $100, but if you both play *B* you will receive nothing. By adopting the minimax strategy, you will choose to play *A* knowing that you will never do worse than receive $1. But you will also never have the chance to earn $100, and the difference between the worst you can do with *B* and the worst you can do with *A* is only

$1. The conservative minimax strategy seems to almost work against common sense in this case.

The situation can be shown to be much worse than this with very little effort. Suppose that the value of a play is not dependent on any single play, but rather on a series of plays in a game. For instance, the final outcome of a game of chess might be a move that checkmates the opponent, but the value of the final checkmate is really a function of all of the plays of the game up to that point. In this case, there are no explicit payoffs for the intermediate moves and this poses a credit assignment problem. The temptation is to attribute part of the value of the final move to each of the preceding moves, but to do so implicitly treats the intervening steps from beginning to end in a linear sum-of-the-parts fashion. Real-world games are rarely decomposable into these sorts of building blocks and it is a fundamental mistake to treat nonlinear problems as if they were linear. In the absence of intermediate values for alternative plays, the minimax strategy can only be treated by *estimating* a value for each play, something that must be calculated online as the game is played. This poses a significant challenge.

And what if the game is not zero-sum, or not competitive? It makes little sense to speak of minimizing the maximum expected damage from another player that is friendly, even actively seeking to bolster your payoff.

The speed of modern computers has opened a new opportunity in simulating games and searching for optimal strategies. Moreover, the advent of evolutionary computation now allows us to use the computer as a tool to discover strategies where heuristics may be unavailable. A limitation of the approach is that these strategies must be represented as data structures within the evolutionary algorithm. As such, the constraints that are imposed on these structures can affect the results that are observed. For example, if strategies are constrained to linear functions of previous plays this might impose a severe limitation on the resulting behaviors. Thus the utility of implementing neural networks as models for generating strategies in complex games becomes apparent: As nonlinear universal functions, they offer flexible models for abstracting the behavior of any measurable strategy. The combination of evolutionary computation and neural networks appears well suited for discovering optimal strategies in games where classic game theory is incapable of providing answers.

This paper surveys some recent efforts to evolve neural networks in two different game settings: 1) the iterated prisoner's dilemma, a nonzero-sum game of imperfect information, and 2) more standard two-player zero-sum games of perfect information such as tic-tac-toe and checkers. The evidence presented indicates that there is a profitable synergy in utilizing neural networks to represent complex behaviors and evolution to optimize those behaviors, particularly in the case where no extrinsic evaluation function is available to assess the quality of performance. The paper concludes with a discussion regarding the implications of evolutionary search to learning and machine intelligence.

## II. EVOLVING NEURAL STRATEGIES IN THE ITERATED PRISONER'S DILEMMA

The conditions that foster the evolution of cooperative behaviors among individuals of a species are rarely well understood. In intellectually advanced social animals, cooperation between individuals, when it exists, is often ephemeral and quickly reverts to selfishness, with little or no clear indication of the specific circumstances that prompt the change. Simulation games such as the prisoner's dilemma have, for some time, been used to gain insight into the precise conditions that promote the evolution of either decisively cooperative or selfish behavior in a community of individuals. Even simple games often generate very complex and dynamic optimization surfaces. The computational problem is to reliably determine any ultimately stable strategy (or strategies) for a specific game situation.

The prisoner's dilemma is an easily defined nonzero-sum, noncooperative game. The term nonzero-sum indicates that whatever benefits accrue to one player do not necessarily imply similar penalties imposed on the other player. The term noncooperative indicates that no preplay communication is permitted between the players. The prisoner's dilemma is classified as a "mixed-motive" game in which each player chooses between alternatives that are assumed to serve various motives [4].

The typical prisoner's dilemma involves two players each having two alternative actions: cooperate (C) or defect (D). Cooperation implies increasing the total gain of both players; defecting implies increasing one's own reward at the expense of the other player. The optimal policy for a player depends on the policy of the opponent [5, p. 717]. Against a player who always defects, defection is the only rational play. But it is also the only rational play against a player who always cooperates for such a player is a fool. Only when there is some mutual trust between the players does cooperation become a reasonable move in the game.

The general form of the game is represented in Table I (after [6]). The game is conducted on a trial-by-trial basis (a series of moves). Each player must choose to cooperate or defect on each trial. The payoff matrix that defines the game is subject to the following constraints:

$$2\gamma_1 > \gamma_2 + \gamma_3$$
$$\gamma_3 > \gamma_1 > \gamma_4 > \gamma_2$$

The first constraint ensures that the payoff to a series of mutual cooperations is greater than a sequence of alternating plays of cooperate-defect against defect-cooperate (which would represent a more sophisticated form of cooperation [7]). The second constraint ensures that defection is a dominant action, and also that the payoffs accruing to mutual cooperators are greater than those accruing to mutual defectors.

In game-theoretic terms, the one-shot prisoner's dilemma (where each player only gets to make one move: cooperate or defect) has a single dominant strategy (Nash equilibrium) (D,D), which is Pareto dominated by (C,C). Joint defection results in a payoff, $\gamma_4$, to each player that is smaller than the payoff, $\gamma_1$, that could be gained through mutual cooperation. Moreover, defection appears to be the rational play regardless of the opponent's decision because the payoff for a defection will either be $\gamma_3$ or $\gamma_4$ (given that the opponent cooperates or defects, respectively), whereas the payoff for cooperating will be $\gamma_1$ or $\gamma_2$. Since $\gamma_3 > \gamma_1$ and $\gamma_4 > \gamma_2$, there is little motivation to cooperate on a single play.

Defection is also rational if the game is iterated over a series of plays under conditions in which both players' decisions are not affected by previous plays. The game degenerates into a series of independent single trials. But if the players' strategies can depend on the results of previous interactions then “always defect” is not a dominant strategy. Consider a player who will cooperate for as long as his opponent, but should his opponent defect, will himself defect forever. If the game is played for a sufficient number of iterations, it would be foolish to defect against such a player, as least in the early stages of the game. Thus cooperation can emerge as a viable strategy [8].

The iterated prisoner's dilemma (IPD) has itself emerged as a standard game for studying the conditions that lead to cooperative behavior in mixed-motive games. This is due in large measure to the seminal work of Robert Axelrod. In 1979, Axelrod organized a prisoner's dilemma tournament and solicited strategies from game theorists who had published in the field [9]. The 14 entries were competed along with a 15th entry: On each move, cooperate or defect with equal probability. Each strategy was played against all others over a sequence of 200 moves. The specific payoff function used is shown in Table II. The winner of the tournament, submitted by Anatol Rapoport, was "Tit-for-Tat":

1. Cooperate on the first move.
2. Otherwise, mimic whatever the other player did on the previous move

Subsequent analysis in [3], [5, pp. 721-723] and others indicated that this Tit-for-Tat strategy is robust because it never defects first and is never taken advantage of for more than one iteration at a time. Boyd and Lauberbaum [10] showed that Tit-for-Tat is not an evolutionarily stable strategy (in the sense of [2]). Nevertheless, in a second tournament, reported in [11], Axelrod collected 62 entries and again the winner was Tit-for-Tat.

Axelrod [3] noted that 8 of the 62 entries in the second tournament can be used to reasonably account for how well a given strategy did with the entire set. Axelrod [12] used these eight strategies as opponents for a simulated evolving population of policies by considering the set of strategies that are deterministic and use outcomes of the three previous moves to determine a current move. Because there were four possible outcomes for each move, there were $4^3$ or 64 possible sets of three possible moves. The coding for a policy was therefore determined by a string of 64 bits, where each bit corresponded with a possible instance of the preceding three interactions, and six additional bits that defined the player's move for the initial combinations of under three iterations. Thus there were $2^{70}$ (about $10^{21}$) possible strategies.

The simulation was conducted as a series of steps:

1. Randomly select an initial population of 20 strategies.
2. Execute each strategy against the eight representatives and record a weighted average payoff.
3. Determine the number of offspring from each parent strategy in proportion to their effectiveness.
4. Generate offspring by recombining two parents' strategies, and, with a small probability, effect a mutation by randomly changing components of the strategy.
5. Continue to iterate this process.

Recombination and mutation probabilities averaged one crossover and one-half a mutation per generation. Each game consisted of 151 moves (the average of the previous tournaments). A run consisted of 50 generations. Forty trials were conducted. From a random start, the technique created populations whose median performance was just as successful as Tit-for-Tat. In fact, the behavior of many of the strategies actually resembled Tit-for-Tat [12].

Another experiment in [12] required the evolving policies to play against each other, rather than the eight representatives. This was a much more complex environment: The opponents that each individual faced were concurrently evolving. As more effective strategies propagated throughout the population, each individual had to keep pace or face elimination through selection (this protocol of coevolution was offered as early as [13-15], see [16]). Ten trials were conducted with this format. Typically, the population evolved away from cooperation initially, but then tended toward reciprocating whatever cooperation could be found. The average score of the population increased over time as "an evolved ability to discriminate between those who will reciprocate cooperation and those who won't" was attained [12].

Several similar studies followed [12] in which alternative representations for policies were employed. One interesting representation involves the use of finite state automata (e.g., finite state machines (FSMs)) [17, 18]. Figure 1 shows a Mealy machine that implements a strategy for the IPD from [19]. FSMs can represent very complex Markov models (i.e., combining transitions of zero-order, first-order, second-order, and so forth) and were used in some of the earliest efforts in evolutionary computation [20, 21]. A typical protocol for coevolving FSMs in the IPD is as follows:

1. Initialize a population of FSMs at random. For each state, up to a prescribed maximum number of states, for each input symbol (which represents the moves of both players in the last round of play) generate a next move of C or D and a state transition.
2. Conduct IPD games to 151 moves with all pairs of FSMs. Record the mean payoff earned by each FSM across all rounds in every game.
3. Apply selection to eliminate a percentage of FSMs with the lowest mean payoffs.
4. Apply variation operators to the surviving FSMs to generate offspring for the next generation. These variation operators include: i) alter an output symbol, ii) alter a next-state transition, iii) alter the start state, iv) add a state, randomly connected, v)

delete a state and randomly reassign all transitions that went to that state, and vi) alter the initial move.
5. Proceed to step 2 and iterate until the available time has elapsed.

Somewhat surprisingly, the typical behavior of the mean payoff of the surviving FSMs has been observed to be essentially identical to that obtained in [12] using strategies represented by lookup tables. Figure 2 shows a common trajectory for populations ranging from 50 to 1000 FSMs taken from [19]. The dynamics that induce an initial decline in mean payoff (resulting from more defections) followed by a rise (resulting from the emergence of mutual cooperation) appear to be fundamental.

Despite the similarity of the results of [12] and [19], there remains a significant gap in realism between real-world prisoner's dilemmas and the idealized model offered so far. Primary among the discrepancies is the potential for real individuals to choose intermediate levels of cooperating or defecting. This severely restricts the range of possible behaviors that can be represented and does not allow intermediate activity designed to engender cooperation without significant risk or intermediate behavior designed to quietly or surreptitiously take advantage of a partner [22, 23]. Hence, once behaviors evolve that cannot be fully taken advantage of (those that punish defection), such strategies enjoy the full and mutual benefits of harmonious cooperation. Certainly, many other facets must also be considered, including 1) the potential for observer error in ascertaining what the other player did on the last move (i.e., they may have cooperated but it was mistaken for defecting), 2) tagging and remembering encounters with prior opponents, and 3) the possibility of opting out of the game altogether (see [24, 25]). The emphasis here will be on the possibility for using neural networks to represent strategies in the IPD and thereby generate a continuous range of behaviors.

Harrald and Fogel [26] replaced the FSMs with multilayer feedforward perceptrons (MLPs). Specifically, each player's strategy was represented by a MLP that possessed six input nodes, a prescribed number of hidden nodes, and a single output node. The first three inputs corresponded to the previous three moves of the opponent, while the second three corresponded to the previous three moves of the network itself (Figure 3). The length of memory recall was chosen to provide a comparison to Axelrod [12]. The behavior on any move was described by the continuous range $[-1, 1]$, where $-1$ represented complete defection and 1 represented complete cooperation. All nodes in the MLP used sigmoidal filters that were scaled to yield output between $-1$ and 1. The output of the network was taken as its move in the current iteration.

For comparison to prior work, the payoff matrix of Axelrod [12] was approximated by a planar equation of both players' moves. Specifically, the payoff to player *A* against player *B* was given by:

$$f(\alpha,\beta) = -0.75\alpha + 1.75\beta + 2.25$$

Where $\alpha$ and $\beta$ are the moves of the players *A* and *B*, respectively. This function is shown in Figure 4. The basic tenor of the one-shot prisoner's dilemma is thereby maintained: Full defection is the dominant move, joint payoffs are maximized by mutual full cooperation.

An evolutionary algorithm was implemented as follows:

1. A population of a given number of MLPs was initialized at random. All of the weights and biases of each network were initialized uniformly over [–0.5, 0.5].
2. A single offspring MLP was created from each parent by adding a standard Gaussian random variable to every weight and bias term.
3. All networks played against each other in a round-robin competition (each met every other one time). Encounters lasted 151 moves and the fitness of each network was assigned according to the average payoff per move.
4. All networks were ranked according to fitness, and the top half were selected to become parents of the next generation.
5. If the preset maximum number of generations, in this case 500, was met, the procedure was halted; otherwise it proceeded to step 2.

Two sets of experiments were conducted with various population sizes. In the first, each MLP possessed only two hidden nodes (denoted as 6-2-1, for the six input nodes, two hidden nodes, and one output node). This architecture was selected because it is a minimum amount of complexity that requires a hidden layer. In the second, the number of hidden nodes was increased by an order of magnitude to 20. Twenty trials were conducted in each setting with population sizes of 10, 20, 30, 40, and 50 parents.

Table III provides the results in five behavioral categories. The assessment of apparent trends or instability was admittedly subjective, and in some cases the correct decision was not obvious (e.g., Figure 5a). But in general the results showed:

1. There was no tendency for cooperative behavior to emerge when using a 6-2-1 MLP regardless of the population size.
2. Above some minimum population size, cooperation was likely when using a 6-20-1 MLP.
3. Any cooperative behavior that did arise did not tend toward complete cooperation.
4. Complete and generally unrecoverable defection was the likely result with the 6-2-1 MLPs, but could occur even when using 6-20-1 MLPs.

Figure 5 presents some of the typical observed behavior patterns.

The results suggest that the evolution of mutual cooperation in light of the chosen payoff function and continuous behaviors requires a minimum complexity (in terms of behavioral freedom) in the policies of the players. A single hidden layer MLP is capable of performing universal function approximation if given sufficient nodes in the hidden layer. Thus the structures used to represent player policies in the current study could be tailored to be essentially equivalent to the codings in which each move was a deterministic function of the previous three moves of the game [12]. While previous studies observed stable mutual cooperation [12], cooperative behavior was never observed with the 6-2-1 MLPs, but was fairly persistent with the 6-20-1 MLPs. But the *level* of cooperation that was generated when using the 6-20-1 neural networks was neither complete nor steady. Rather, the mean payoff to all parents tended to peak below a value of 3.0 and decline, while complete mutual cooperation would have yielded an average payoff of 3.25.

The tendency for cooperation to evolve with sufficient complexity should be viewed with caution for at least three reasons. First, very few trials with 6-20-1 MLPs exhibited an increase in payoff as a function of the number of generations. The more usual result was a steady decline in mean payoff, away from increased cooperation. Second, cooperative behavior was not always steady. Figure 6 indicates the results for trial 10 with 20 parents using 6-20-1 neural networks executed over 1500 generations. The behavior appeared cooperative until just after the 1200th generation, at which point it declined rapidly to a state of complete defection. A recovery from complete defection was rare, regardless of the population size or complexity of the networks. It remains to be seen if further behavioral complexity (i.e., a greater number of hidden nodes) would result in more stable cooperation. Finally, the specific level of complexity in the MLP that must be attained before cooperative behavior emerges is not known, and there is no a priori reason to believe that there is a smooth relationship between the propensity to generate cooperation and strategic complexity.

The striking result of these experiments is that the emergent behavior of the complex adaptive system in question relies heavily on the representation for that behavior and the dynamics associated with that representation. The fortuitous early choice of working with models that allow only the extremes of complete cooperation or defection happened to lead to models of social interaction which imbued "the evolution of cooperation" [3]. We can only speculate about what interpretation would have been offered if this earlier choice had instead included the option of a continuum of behaviors. Would Axelrod's seminal book [3] have been titled "The Evolution of Unstable Cooperation that Often Leads to Total Catastrophe"?

In this case, the combination of evolutionary computation and neural networks offers a distinctly useful approach to modeling complex adaptive systems. There are no training algorithms for such systems because the desired behavior must emerge from the model, rather than be programmed into it. Thus evolution provides a basis for allowing that emergence. It is perhaps difficult to envision another reasonable alternative. Further, the behaviors of the individuals that are modeled must be sufficiently flexible to provide confidence that they provide sufficient fidelity. The inclusion of a continuous range of behaviors, as can be accomplished using neural networks, provides an advantage not found in simpler classic models of the IPD and one that could be carried over to versions that involve an arbitrary number of players [51].

## III. Evolving Neural Strategies in Tic-Tac-Toe

In contrast to the nonzero-sum IPD, attention can be given to evolving strategies in zero-sum games of perfect information. One simple example of such a game is *tic-tac-toe* (also known as *naughts and crosses*). The game is well known but will be described in detail for completeness. There are two players and a three-by-three grid (Figure 7). 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 a 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.

The game is sufficiently complex to demonstrate the potential for evolving neural networks as strategies, with particular attention given to devising suitable tactics without utilizing expert knowledge. That is, rather than rely on the common artificial intelligence approach of programming rules of behavior or weighted features of the problem that are deemed important by a human expert, neural networks can be evolved simply on the basis of the information contained in their “win, lose, and draw” record against a competent player.

Fogel [27, p. 232] devoted attention to evolving a strategy for the first player (an equivalent procedure could be used for the second player). A suitable coding structure was required. It had to receive a board pattern as input and yield a corresponding move as output. The coding structure utilized in these experiments was a MLP (Figure 8). Each hidden or output node performed a sum of the weighted input strengths, subtracted off an adaptable bias term, and passed the result through a sigmoid nonlinearity, $(1 + e^{-x})^{-1}$. Only a single hidden layer was incorporated. This architecture was selected because:

1. Variations of MLPs are universal function approximators.
2. The response to any stimulus could be evaluated rapidly.
3. The extension to multiple hidden layers is obvious.

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.
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 to evaluate each contending network (see below). 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) Move randomly in any open square.
3. Continue with step 2 until the game is completed.
4. Continue with step 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 [28, 29, pp. 362-363]. 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 three sets of experiments over {+1, –1, 0}, {+1, –10, 0}, and {+10, –1, 0}, where the entries are the payoff for winning, losing, and playing to a draw, respectively. The maximum possible score over any four sets of games was 32 under the first two payoff functions and 320 under the latter payoff function. 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 with 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 successive generations. Thirty trials were conducted with each payoff function. Evolution was halted after 800 generations in each trial.

The learning rate when using {+1, −1, 0} was generally consistent across all trials (Figure 9). The 95 percent confidence limits around the mean were close to the average performance. The final best network in this trial possessed nine hidden nodes. The highest score achieved in trial 2 was 31. Figure 10 indicates the tree of possible games if the best-evolved neural network were played against the rule-based algorithm, omitting the possibility for random moves in step 2a. Under such conditions, the network would force a win in four of the eight possible branches of the tree and has the possibility of losing in two of the branches.

The learning rate when using {+1, −10, 0} was considerably different (Figure 11) from that indicated in Figure 9. Selection in light of the increased penalty for losing resulted in an increased initial rate of improvement. Strategies that lose were quickly purged from the evolving population. After this first-order condition was satisfied, optimization continued to sort out strategies with the greatest potential for winning rather than drawing. Again, the 95 percent confidence limits around the mean were close to the average performance across all 30 trials. Figure 12 depicts the tree of possible games if the best network from trial 1 were played against the rule-based player, omitting random moves from step 2a. It would not force a win in any branch of the tree, but it would also never lose. The payoff function was clearly reflected in the final observed behavior of the evolved network.

The learning rate when using {+10, −1, 0} appeared similar to that obtained when using {+1, −1, 0} (Figure 13). The 95 percent confidence limits were wider than was observed in the previous two experiments. The variability of the score was greater because a win received 10 more points than a draw, rather than only a single point more. Strategies with a propensity to lose were purged early in the evolution. By the 800th generation, most surviving strategies lost infrequently and varied mostly by their ability to win or draw. The tree of possible games against the rule-based player is shown in Figure 14. The best-evolved network in trial 1 could not achieve a perfect score except when the rule-based procedure made errors in play through random perturbation (step 2a).

These experiments indicate the ability for evolution to adapt and optimize neural networks that represent strategies for playing tic-tac-toe. But more importantly, no a priori information regarding the object of the game was offered to the evolutionary algorithm. 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 information available regarding the quality of play. Further, this information was only provided after 32 games had been played; the contribution to the overall score from any single game was not discernible. Heuristics regarding the environment were limited to the following:

1. There were nine inputs.

2. There were nine outputs.
3. Markers could only be placed in empty squares.

Essentially then, the procedure was only given an appropriate set of sensors and a means for generating all possible behaviors in each setting, and was restricted to act within the "physics" of the environment (i.e., the rules of the game). Nowhere was the evolutionary algorithm explicitly told in any way that it was playing tic-tac-toe. The evidence suggests the potential for learning about arbitrary games and representing the strategies for allocating resources in neural networks that can represent the nonlinear dynamics underlying those decisions.

## IV. COEVOLVING NEURAL STRATEGIES IN CHECKERS

One limitation of the preceding effort to evolve neural networks in tic-tac-toe was the use of a heuristic program that acted as the opponent for each network. Although no explicit knowledge was programmed into the competing networks, a requirement for an existing knowledgeable opponent is nearly as limiting. The ultimate desire is to have a population of neural networks learn to play a game simply by playing against themselves, much like the strategies in [12, 19] learned to play the IPD through coevolution. Before describing the method employed here to accomplish this with the framework of the game of checkers, we must first describe the rules of the game.

Checkers is traditionally played on an eight-by-eight board with squares of alternating colors (e.g., red and black, see Fig. 15). There are two players, denoted as “red” and “white” (or “black” and “white,” but here for consistency with a commonly available website on the internet that allows for competitive play between players who log in, the notation will remain with red and white). Each side has 12 pieces (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 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 vs. a single jump). When a checker advances to the last row of the board it becomes a king, and can thereafter move diagonally in any direction (i.e., 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 it can also occur when all existing pieces are trapped, resulting in a loss for that 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.[1]

## *A. Method*

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 feed forward 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 output node was the hyperbolic tangent (tanh, bounded by $\pm 1$) 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.

Previous efforts in [30] used neural networks with two hidden layers, comprising 40 and 10 units respectively, to process the raw inputs from the board. Thus the $8 \times 8$ checkers board was interpreted simply as a $1 \times 32$ vector, and the neural network was forced to learn all of the spatial characteristics of the board. So as not to handicap the learning procedure in this manner, the neural network used here implemented a series of 91 preprocessing nodes that covered $n \times n$ square overlapping subsections of the board. These $n$ x $n$ 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 possible $3 \times 3$ square subsections of the board were provided as input to the first 36 hidden nodes in the first hidden layer. The following 25 $4 \times 4$ square subsections were assigned to the next 25 hidden nodes in that layer, and so forth. Figure 16 shows a sample $3 \times 3$ square subsection that contains the states of positions 1, 5, 6, and 9. Two sample $4 \times 4$ 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 enabled the neural network to generate features from these subsets of the entire board that could then be processed in subsequent hidden layers (of 40 and 10 hidden units, following [30]). Figure 17 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.

It is important to note immediately that (with one exception) no attempt was made to offer useful features as inputs to a player's neural network. The common approach to designing superior game-playing programs is to use a human expert to delineate a series of board patterns or general features that are weighted in importance, favorably or unfavorably. In addition, entire opening sequences from games played by grand masters and look-up tables of end game positions can also be stored in memory and retrieved when appropriate. Here, these sorts of "cheats" were eschewed: The experimental question at hand concerned the level of play that could be attained simply by using evolution to extract linear and nonlinear features regarding the game of checkers and to optimize the interpretation of those features within the neural network.[2] The only feature that could be claimed to have been offered is a function of the piece differential between a player and its opponent, owing to the sum of the inputs being supplied directly to the output node. The output essentially sums all the inputs which in turn offers the piece advantage or disadvantage. But this is not true in general, for when kings are present on the board, the value $K$ or $-K$ is used in the summation, and as described below, this value was evolvable rather than prescribed by the programmers a priori. Thus the evolutionary algorithm had the potential to override the piece differential and invent a new feature in its place. Absolutely no other explicit or implicit features of the board beyond the location of each piece were implemented.

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 (e.g., no remaining opposing pieces) were assigned the value of exactly 1.0 and likewise all positions that were losses were assigned $-1.0$.

To begin the evolutionary algorithm, a population of 15 strategies (neural networks), $P_i$, $i = 1, ..., 15$, defined by the weights and biases for each neural network and the strategy's associated value of $K$, was created at random. Weights and biases were generated by sampling from a uniform distribution over $[-0.2,0.2]$, with the value of $K$ set initially to 2.0. Each strategy had an associated self-adaptive parameter vector $\sigma_i$, $i = 1, ..., 15$, 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 an offspring strategy by varying all of the associated weights and biases, and possibly the value of $K$ as well. Specifically, for each parent $P_i$, $i = 1, ..., 15$ an offspring $P¢_i$, $i = 1, ..., 15$, was created by:

$$\sigma¢_i(j) = \sigma_i(j)\exp(\ \tau N_j\ (0,1)\ ),\ j = 1, ..., N_w$$
$$w¢_i(j) = w_i(j) + \sigma¢_i(j)N_j(0,1),\ j = 1, ..., N_w$$

where $N_w$ is the number of weights and biases in the neural network (here this is 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:

$$K¢_i = K_i + d$$

where $d$ was chosen uniformly at random from {–0.1, 0, 0.1}. For convenience, the value of $K'_i$ was constrained to lie in [1.0, 3.0] by resetting to the limit exceeded when applicable.

All parents and their offspring competed for survival by playing games of checkers 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 five games, the player always played red, whereas the randomly selected opponent always played white. In each game, the player scored –2, 0, or +1 points depending on whether it lost, drew, or won the game, respectively (a draw was declared after 100 moves for each side). Similarly, each of the opponents also scored –2, 0, or +1 points depending on the outcome. These values were somewhat arbitrary, but reflected a generally reasonable protocol of having a loss be twice as costly as a win was beneficial. 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 fail-soft alpha-beta search [31] of the associated game tree for each board position looking a selected number of moves into the future. The minimax move for a given ply was determined by selecting the available move that affords the opponent the opportunity to do the least damage as determined by the evaluation function on the resulting position. The depth of the search, $d$, was set at four to allow for reasonable execution times (30 generations on a 400 MHz Pentium II required about seven days, although no serious attempt was made to optimize the run-time performance of the algorithm). In addition, when forced moves were involved, the search depth was extended (let $f$ be the number of forced moves) because in these situations the player has no real decision to make. The ply depth was extended by steps of two, up to the smallest even number that was greater than or equal to the number of forced moves $f$ that occurred along that branch. If the extended ply search produced more forced moves then the ply was once again increased in a similar fashion. Furthermore, if the final board position was left in an "active" state, where the player has a forced jump, the depth was once again incremented by two ply. Maintaining an even depth along each branch of the search tree ensured that the boards were evaluated after the opponent had an opportunity to respond to the player's move. 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 or not that ply corresponded to the opponent’s move or the player’s move. For more on the mechanics of alpha-beta search, see [31].

This evolutionary process, starting from completely randomly generated neural network strategies, was iterated for 230 generations (approximately eight weeks of evolution). The best neural network (from generation 230) was then used to play against human opponents on an internet gaming site (www.zone.com). Each player logging on to this site is initially given a rating, $R_0$, of 1600 and a player’s rating changes according to the following formula (which follows the rating system of the United States Chess Federation (USCF)):

$$R_{New} = R_{Old} + C\left(Outcome - W\right) \tag{1}$$

where

$$W = \frac{1}{1 + 10^{\left(\frac{R_{Opp} - R_{Old}}{400}\right)}}$$

$$Outcome \in \left\{1 \text{ if Win, } 0.5 \text{ if Draw, } 0 \text{ if Loss}\right\}$$

$R_{Opp}$ is the rating of the opponent, and $C = 32$ for ratings less than 2100.[4]

Over the course of one week, 100 games were played against opponents on this website. Games were played until (1) a win was achieved by either side, (2) the human opponent resigned, or (3) a draw was offered by the opponent and (i) the piece differential of the game did not favor the neural network by more than one piece and (ii) there was no way for the neural network to achieve a win that was obvious to the authors, in which case the draw was accepted. A fourth condition occurred when the human opponent abandoned the game without resigning (by closing their graphical-user interface) thereby breaking off play without formally accepting defeat. When an opponent abandoned a game in competition with the neural network, a win was counted if the neural network had an obvious winning position (one where a win could be forced easily in the opinion of the authors) or if the neural network was ahead by two or more pieces; otherwise, the game was not recorded. There was a fifth condition which occurred only once wherein the human opponent exceeded the four minute per move limit (imposed on all rated games on the website) and as a result forfeited the game. In this special case, the human opponent was already significantly behind by two pieces and the neural network had a strong position. In no cases were the opponents told that they were playing a computer program, and no opponent ever commented that they believed their opponent was a computer algorithm.

Opponents were chosen based primarily on their availability to play (i.e., they were not actively playing someone else at the time) and to ensure that the neural network competed against players with a wide variety of skill levels. In addition, there was an attempt to balance the number of games played as red or white. In all, 49 games were played as red. All moves were based on a ply depth of $d = 6$ and infrequently 8, depending on the perceived time required to return a move (less than 30 seconds was desired). The vast majority of moves were based on $d = 6$.

## *B. Results*

Figure 18 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 evolved neural network dominated players rated 1800 and lower, and had a majority of wins vs. losses against opponents rated between 1800 and 1900. Figure 19 shows the sequential rating of the neural network and the rating of the opponents played over all 100 games. Table IV provides a listing of the class intervals and designations of different ratings accepted by the USCF.

Given that the 100 independent games played to evaluate the player could have been played in any order (since no learning was performed by the neural network during the series of games played on the website), an estimate of the network's true rating can be obtained by sampling from the population of all possible orderings of opponents. (Note that the total number of orderings is $100! \approx 9.33 \times 10^{157}$, which is too large to enumerate.) The network's rating was calculated over 5000 random orderings drawn uniformly from the set of all possible orderings using Eq. (1). Figure 20 shows the histogram of the ratings that resulted from each permutation. The corresponding mean rating was 1929.0, with a standard deviation of 32.75. The minimum and maximum ratings obtained were 1799.48 and 2059.47. Figure 21 shows the rating trajectory averaged over the 5000 permutations as a function of the number of games played. The mean rating starts at 1600 (the standard starting rating at the website) and steadily climbs above 1850 by game 40**.** As the number of games reaches 100, the mean rating curve begins to saturate and reaches 1929.0 which places it subjectively as an above-average Class A player.

The neural network’s best result was recorded in a game where it defeated a player who was rated 2210 (master level). At the time, this opponent was ranked 29th on the website out of more than 40,000 registered players. The sequence of moves is shown in the Appendix. Certain moves are annotated, but note that these annotations are not offered by an expert checkers player (instead being offered here by the authors). Undoubtedly, a more advanced player might have different comments to make at different stages in the game. Selected positions are also shown in accompanying figures. Also shown is the sequence of moves of the evolved network in a win over an expert rated 2024, who was ranked 174th on the website.

This is the first time that a checkers-playing program that did not incorporate preprogrammed expert knowledge was able to defeat a player at the master level. Prior efforts in [30] defeated players at the expert level and played to a draw against a master.

## *C. Discussion*

The results indicate the ability for an evolutionary algorithm to start with essentially no preprogrammed information in the game of checkers (except the piece differential) and learn, over successive generations, how to play at a level that is just below what would qualify as “expert” by a standard accepted rating system. The most likely reason that the best-evolved neural network was not able to achieve a higher rating was the limited ply depth of $d = 6$ or 8. This handicap is particularly evident in two phases of the game:

1. In the end game, where it is common to find pieces separated by several open squares, a search at $d = 6$ may not allow pieces to effectively “see” that there are other pieces within eventual striking distance.
2. In the middle of the game, even if the neural network could block the opponent from moving a piece forward for a king, it would often choose not to make this block. The

ply depth was sufficient to see the opponent getting the king, but insufficient to see the damage that would come as a result several moves after that.

The first of these two flaws resulted in some games that could have been won if a larger ply were available, and it is well known that some end game sequences in checkers can require a very high ply (e.g., 20-60 [32]). But the second flaw was more devastating because this “Achilles heel” allowed many weaker yet aggressive players (rated 1600-1800) the opportunity to get an early king. All of the losses to players rated below 1800 can be traced to this flaw. Whereas weaker players tend to have the objective of getting a king as quickly as possible, better players tend to play for position and set up combination moves that will capture pieces. Thus this pitfall was rarely exploited by superior opponents.

The current world champion checkers program is *Chinook* [32-34]. The program uses alpha-beta search with a preprogrammed evaluation function based on features of the board, opening moves from games between grand masters, and a complete endgame database for all positions with eight or fewer pieces remaining on the board (440 billion possibilities). It is this level of knowledge that is required to play at the world champion level (2814 rating). Everything that Chinook “knows” represents human expertise that was programmed into it by hand. The algorithm is deterministic, no different than a calculator in its process, computing each next move with exacting immutable precision.

It must be quickly admitted that the best-evolved spatial neural network cannot compete with Chinook; however, recall that this was not the intention of the experimental design. The hypothesis in question was whether or not there was sufficient information simply in the final results from a series of games for evolution to design strategies that would defeat humans. Not only was this answered affirmatively but also the level of play was sufficient to defeat several expert-level players and one master-level player.

It is of interest to assess the significance of this result in comparison with other methods of machine learning applied to checkers. Undoubtedly the most widely known such effort is due to Arthur Samuel from 40 years ago [35]. Samuel’s method relied on a polynomial evaluation function that comprised a sum of weighted features of a checkers board. All of the features were chosen by Samuel rather than designed *de novo* by his program (as performed here). The coefficients of the polynomial were determined using a simple updating procedure that was based on self-play. For each game, one side would use the polynomial that won the prior game and the other would use a variant of this polynomial. The variation was made deterministically. The winner would continually replace the loser, and occasionally some of the features with low weight would be replaced by others. By 1956, Samuel’s program had learned to play well enough to defeat novice players [34, p. 93].

There is, however, a misconception regarding the quality of play that was eventually obtained by Samuel’s program. When played against humans of championship caliber, the program performed poorly. It did have one early win in 1962 against Robert Nealey, an eventual Connecticut state champion. This win received considerable publicity but in fact the game itself can be shown to have been poorly played on both sides (Schaeffer [34] analyzed the moves of the game using Chinook and identified many mistakes in play for both Nealey and Samuel's program). It would be more accurate to assert that Nealey lost the game than to say that Samuel's program earned the win. In a rematch the following year, Nealey defeated Samuel's program in a six-game match. Later, in 1966, Samuel played his program against the contenders for the world championship (Walter Hellman and Derek Oldbury). Both Hellman and Oldbury defeated the program four straight times each [34]. These negative results are rarely acknowledged.

After watching Samuel's program lose to another program developed with the support of Duke University in 1977, the American Checkers Federation Games Editor Richard Fortman was quoted as stating "There are several thousand just average Class B players who could beat either computer without difficulty" [34, p. 97]. The true level of competency for Samuel's program remains unclear, but there is no doubt that the common viewpoint that Samuel "solved" the game of checkers, or that his results have yet to be surpassed [36, p. 19], are in error.

Nevertheless, there is also no doubt that Samuel's use of self-play was innovative. Even after the highly visible publication of [35], there were only four other (known) efforts in evolutionary computation over the next decade to coevolve strategies in settings where no extrinsic evaluation function was used to judge the quality of behavior [13-15, 37]. Samuel might have easily envisioned embarking on the course that has been laid out in this paper, extending the number of competitors beyond only two, and using random variation to change coefficients in the polynomial evaluation function. It might have even been obvious as something to consider.

The immediate limitation facing Samuel, and this is an important point to reckon when considering the recent resurgence of interest in evolutionary computation, would have been the available computing power. The IBM 7094 machine that he used to play Nealey could perform six million multiplications per minute. By comparison, the Pentium II 400 MHz computer we used can exceed this by three orders of magnitude. We required about 1440 hours (60 days) of CPU time to evolve the spatial neural network that was tested here. For Samuel, this would have translated into about 165 years! But this number could be lowered to about 20 years by recognizing the rate at which computer speeds increased over this timeframe. Had Samuel started in 1959 with the approach offered here, he might have had a tangible result in 1979. Even then, it would have been difficult to assess the level of play of the neural network because there were no internet gaming sites where players of various skill levels could be challenged. The neural network's ability would have had to be assessed in regulation match play.

It is enticing to speculate on the effect that a nearly expert-rated checkers-playing neural network would have had on the artificial intelligence community and the end of the 1970s, particularly one that was evolved without any extrinsic knowledge incorporated in features to evaluate the board (with the exception of piece differential). Regardless, it is clear that the computer technology for applying evolutionary algorithms to significant problems in machine learning has only recently caught up with the concept. All of the pioneers who worked on evolutionary computation from the early 1950s to 1970s [16] were 20-40 years ahead of their time.

## V. Intelligence in Decision Making

Neural networks provide a versatile representation for complex behaviors in games because they are universal function approximators. They do not present the only possible choice for such functions, but their modularity, ease of design, and the relatively simple processing that takes place at each node make them a suitable selection. The experiments illustrated here in the iterated prisoner's dilemma, tic-tac-toe, and checkers show that neural networks can generate useful strategies in both zero- and nonzero-sum games. More importantly, the combination of evolutionary computation and neural networks provides a means for designing solutions to games of strategy where there are no existing experts and no examples that could serve as a database for training. The ability to discover novel, high-performance solutions to complex problems through coevolution mimics the strategy that we observe in nature where there is no extrinsic evaluation function and individuals can only compete with other extant individuals, not with some presumed “right answer.”

The evolution of strategies in games relates directly to the concept of intelligence in decision making. The crux of intelligent behavior is the ability to predict future circumstances and take appropriate actions in light of desired goals [21]. Central to this ability is a means for translating prior observations into future expectations. Neural networks provide one possibility for accomplishing this transduction; evolution provides a means for optimizing the fidelity of the predictions that can be generated and the corresponding actions that must be taken by varying potentially both the weights and structure of the network. For example, in each game of checkers, the neural network evaluation function served as a measure of the worth of each next possible board pattern and was also used to anticipate the opponent's response to actions taken. The combination of evolution and neural networks enabled a capability to both measure the worth of alternative decisions and optimize those decisions over successive generations.

The recognition of evolution as an intelligent learning process has been offered many times ([38, 39] and others). In order to facilitate computational intelligence, it may be useful to recognize that all learning reflects a process of adaptation. The most important aspect of such learning processes is the “development of implicit or explicit techniques to accurately estimate the probabilities of future events” [40] (also see [41]). Ornstein [40] suggested that as predicting future events is the “forte of science,” it is reasonable to examine the scientific method for useful cues in the search for effective learning techniques. Ref. [21, p. 112] went further and developed a specific correspondence between natural evolution and the scientific method (later echoed in [42]). In nature, individual organisms serve as hypotheses concerning the logical properties of their environment. Their behavior is an inductive inference concerning some as yet unknown aspects of that environment. The validity of each hypothesis is demonstrated by its survival. Over successive generations, organisms generally become better predictors of their surroundings. Those that fail to predict adequately are “surprised,” and surprise often has lethal consequences.

In this paper, attention has been given to representing hypotheses in the form of neural networks, but evolution can be applied to any data structure. Prior efforts have included the evolution of finite state machines [21], autoregressive moving-average (ARMA) models [43], and multiple interacting programs represented in the form of symbolic expressions [44] to predict a wide range of environments. The process of evolution provides the means for discovering, rejecting, and refining models of the world (even in the simplified cases of the “worlds” examined in this paper). It is a fundamental basis for learning in general [41, 45, 46].

Genesereth and Nilsson [47] offered “Artificial intelligence is the study of intelligent behavior. Its ultimate goal is a theory of intelligence that accounts for the behavior of naturally occurring intelligent entities and that guides the creation of artificial entities capable of intelligent behavior.” Yet research in artificial intelligence has typically passed over investigations of the primal causative factors of intelligence to more rapidly obtain the immediate consequences of intelligence [41]. Efficient theorem proving, pattern recognition, and tree searching are symptoms of intelligent behavior. But systems that can accomplish these feats are not, simply by consequence, intelligent. Certainly, these systems have been applied successfully to specific problems, but they do not generally advance our understanding of intelligence. They solve problems, but do not solve the problem of how to solve problems.

Perhaps this is so because there is no generally accepted definition of the intelligence that the field seeks to create. A definition of intelligence would appear to be prerequisite to research in a field termed *artificial intelligence*, but such definitions have only rarely been provided (the same is true for *computational intelligence*). And when they have been offered, they have often been of little operational value.

For example, Minsky [48, p. 71] suggested “Intelligence ... means ... the ability to solve hard problems.” But how hard does a problem have to be? Who is to decide which problems are hard? All problems are hard until you know how to solve them, at which point they become easy. Such a definition is immediately problematic. Schaeffer [34, p. 57] described artificial intelligence as the field of “making computer programs capable of doing intelligent things,” which begs the question of what in fact are *intelligent things*? Just two pages later, Schaeffer [34, p. 59] defined artificial intelligence as “AI creates the illusion of intelligence.” Unfortunately, this definition requires an observer and necessarily provokes the question of just who is being fooled? Does a deterministic world champion checkers program like Chinook give the “illusion of intelligence”? And what if it does? No amount of trickery, however clever, is going to offer an advance on the problem of how to solve problems. *Faking* intelligence can never be the basis of a scientific theory that accounts for the behavior of naturally intelligent organisms (cf. [38]).

Intelligence involves decision making. Decisions occur when there is the selection of one from a number of alternative ways of allocating the available resources. For any system to be intelligent, it must consistently select appropriate courses of action under a variety of conditions in light of a given purpose. The goals (purpose) of biological organisms derive from their competition for the available resources. Selection eliminates those variants that do not acquire sufficient resources. Thus, while evolution as a process is purposeless, the primary purpose of all living systems is survival. Those variants that do not exhibit sufficiently suitable behavior are stochastically culled. The genetically programmed and learned behaviors of the survivors (and thus the goal of survival) are reinforced in successive generations through intense competition.

This basic idea has been suggested many times ([49, p. 3], [41], and others). But the definition of intelligence need not be restricted to biological organisms. Intelligence can be a property of *any* purpose-driven decision maker. It must apply equally well to humans, colonies of ants, robots, social groups, and so forth. Thus, more generally, intelligence may be defined as *the capability of a system to adapt its behavior to meet its goals in a range of environments*, [20, 21, p. 2]. For a species, survival is a necessary goal in any given environment; for a machine, both purpose and environments may be specified by the machine's designer.

Evolution then is the process that accounts for intelligent behavior in "naturally occurring entities," and this process of population-based search with random variation and selection can be simulated and used for the creation of intelligent machines. A population is required because single point-to-point searches are often insufficiently robust to overcome local pathologies. Selection is required because without a fitness criterion (implicit or extrinsic) and a procedure for eliminating poor solutions, the search would degenerate into a purely random walk. And randomness is required because deterministic searches cannot learn to meet any unforeseen change in the environment. Hofstadter [50, p. 115] offered, "to a program that exploits randomness, *all pathways are open*, even if most have very low probabilities; conversely, to a program whose choices are always made by consulting a fixed deterministic strategy, many pathways are *a priori* completely closed off. This means that many creative ideas will simply never get discovered...." Randomness is a fundamental aspect of intelligence. Indeed, the life process itself provides the most common form of intelligence [41]. The experiments presented here will have served the authors' purpose if they encourage others to pursue the limits of evolution's ability to discover new solutions in new ways.

## Footnotes

1. The game can also end in other ways: 1) by resignation, 2) a draw may be declared when no advancement in position is made in 40 moves by a player who holds an advantage, subject to the discretion of an external third party, and if in match play, 3) a player can be forced to resign if they run out of time, which is usually limited to 60 minutes for the first 30 moves, with an additional 60 minutes for the next 30 moves, and so forth.

2. It might be argued that providing nodes to process spatial features is a similar “cheat.” Note, however, that the spatial characteristics of a checkers board are immediately obvious, even to a person who has never played a game of checkers. It is the invention and interpretation of alternative spatial features that requires expertise, and no such information was preprogrammed here.

## References

[1] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton, NJ: Princeton University Press, 1944.

[2] J. Maynard Smith, *Evolution and the Theory of Games*. Cambridge, U.K.: Cambridge University Press, 1982.

[3] R. Axelrod, *The Evolution of Cooperation*. New York: Basic Books, 1984.

[4] A. Rapoport, “Optimal Policies for the Prisoner's Dilemma,” University of North Carolina Tech Report No. 50, 1966.

[5] D. Hofstadter, *Metamagical Themas: Questing for the Essence of Mind and Pattern*. New York: Basic Books, 1985.

[6] A. Scodel, J. S. Minas, P. Ratoosh, and M. Lipetz, “Some Descriptive Aspects of Two-Person Non-Zero Sum Games,” *Journal of Conflict Resolution*, vol. 3, pp. 114-119, 1959.

[7] P. J. Angeline, “An Alternate Interpretation of the Iterated Prisoner's Dilemma and the Evolution of Non-Mutual Cooperation,” presented at Artificial Life IV, Cambridge, MA, Edited by R. Brooks and P. Maes, MIT Press, 1994, pp. 353-358.

[8] D. Kreps, P. Milgrom, J. Roberts, and J. Wilson, “Rational Cooperation in the Finitely Repeated Prisoner's Dilemma,” *J. Econ. Theory*, vol. 27, pp. 326-355, 1982.

[9] R. Axelrod, “Effective Choice in the Iterated Prisoner's Dilemma,” *Journal of Conflict Resolution*, vol. 24, pp. 3-25, 1980.

[10] R. Boyd and J. P. Lorberbaum, “No Pure Strategy is Evolutionarily Stable in the Repeated Prisoner's Dilemma,” *Nature*, vol. 327, pp. 58-59, 1987.

[11] R. Axelrod, “More Effective Choice in the Prisoner's Dilemma,” *Journal of Conflict Resolution*, vol. 24, pp. 379-403, 1980.

[12] R. Axelrod, “The Evolution of Strategies in the Iterated Prisoner's Dilemma,” in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. London: Pitman, 1987, pp. 32-41.

[13] N. A. Barricelli, “Numerical Testing of Evolution Theories. Part II: Preliminary Tests of Performance, Symbiogenesis and Terrestrial Life,” *Acta Biotheoretica*, vol. 16, pp. 99-126, 1963.

[14] J. Reed, R. Toombs, and N. A. Barricelli, “Simulation of Biological Evolution and Machine Learning,” *Journal of Theoretical Biology*, vol. 17, pp. 319-342, 1967.

[15] L. J. Fogel and G. H. Burgin, “Competitive Goal-Seeking through Evolutionary Programming,” Air Force Cambridge Research Laboratories Contract AF 19(628)-5927, 1969.

[16] D. B. Fogel (ed.), “Evolutionary Computation: The Fossil Record,” Piscataway, NJ: IEEE Press, 1998.

[17] G. H. Mealy, “A Method of Synthesizing Sequential Circuits,” *Bell Systems Tech. Journal*, vol. 34, pp. 1054-1079, 1955.

[18] E. F. Moore, “Gedanken-Experimetns on Sequential Machines: Automata Studies,” *Annals of Mathematical Studies*, vol. 34, pp. 129-153, 1957.

[19] D. B. Fogel, “Evolving Behaviors in the Iterated Prisoner's Dilemma,” *Evolutionary Computation*, vol. 1, pp. 77-97, 1993.

[20] L. J. Fogel, “On the Organization of Intellect,” Doctoral dissertation: UCLA, 1964.

[21] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. NY: John Wiley, 1966.

[22] T. To, “More Realism in the Prisoner's Dilemma,” *J. Conflict Resolution*, vol. 32, pp. 402-408, 1988.

[23] R. Marks, “Breeding Hybrid Strategies: Optimal Behavior for Oligopolists,” presented at Proc. Third International Conference on Genetic Algorithms, San Mateo, CA, Edited by J. D. Schaffer, Morgan Kaufmann, 1989, pp. .

[24] D. B. Fogel, “On the Relationship between the Duration of an Encounter and the Evolution of Cooperation in the Iterated Prisoner's Dilemma,” *Evolutionary Computation*, vol. 3, pp. 349-363, 1995.

[25] E. A. Stanley, D. Ashlock, and L. Tesfatsion, “Iterated Prisoner's Dilemma with Choice and Refusal of Partners,” presented at Artificial Life III, Reading, MA, Edited by C. G. Langton, Addison-Wesley, 1994, pp. 131-175.

[26] P. G. Harrald and D. B. Fogel, “Evolving Continuous Behaviors in the Iterated Prisoner's Dilemma,” *BioSystems*, vol. 37, pp. 135-145, 1996.

[27] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.

[28] B. D. O. Anderson, R. R. Bitmead, C. R. Johnson, P. V. Kokotovic, R. L. Kosut, I. M. Y. Mareels, L. Praly, and B. D. Riedle, *Stability of Adaptive Systems: Passivity and Averaging Analysis.* Cambridge, MA: MIT Press, 1986.

[29] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[30] K. Chellapilla and D. B. Fogel, “Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge,” *IEEE Trans. Neural Networks*, in review, 1999.

[31] H. Kaindl, “Tree Searching Algorithms,” in *Computers, Chess, and Cognition*, T. A. Marsland and J. Schaeffer, Eds. New York: Springer, 1990, pp. 133-168.

[32] J. Schaeffer, R. Lake, P. Lu, and M. Bryant, “Chinook: The World Man-Machine Checkers Champion,” *AI Magazine*, vol. 17, pp. 21-29, 1996.

[33] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron, “A World Championship Caliber Checkers Program,” *Artificial Intelligence*, vol. 53, pp. 273-290, 1992.

[34] J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers*. New York: Springer, 1996.

[35] A. L. Samuel, “Some Studies in Machine Learning Using the Game of Checkers,” *IBM Journal of Research and Development*, vol. 3, pp. 210-219, 1959.

[36] J. H. Holland, *Emergence: From Chaos to Order*. Reading, MA: Addison-Wesley, 1998.

[37] M. Conrad and H. H. Pattee, “Evolution Experiments with an Artificial Ecosystem,” *J. Theoretical Biology*, vol. 28, pp. 393-409, 1970.

[38] A. M. Turing, “Computing Machinery and Intelligence,” *Mind*, vol. 59, pp. 433-460, 1950.

[39] L. J. Fogel, “Autonomous Automata,” *Industrial Research*, vol. 4, pp. 14-19, 1962.

[40] L. Ornstein, “Computer Learning and the Scientific Method: A Proposed Solution to the Information Theoretical Problem of Meaning,” *J. of the Mt. Sinai Hospital*, vol. 32, pp. 437-494, 1965.

[41] J. W. Atmar, “Speculation on the Evolution of Intelligence and Its Possible Realization in Machine Form,” Doctoral dissertation, Las Cruces, NM: New Mexico State University, 1976.

[42] M. Gell-Mann, *The Quark and the Jaguar*. New York: W.H. Freeman, 1994.

[43] D. B. Fogel, *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*. Needham, MA: Ginn Press, 1991.

[44] P. J. Angeline, “Evolving Predictors for Chaotic Time Series,” presented at Applications and Science of Computational Intelligence, Edited by S. K. Rogers, D. B. Fogel, J. C. Bezdek, and B. Bosacchi, SPIE, 1998, pp. 170-180.

[45] N. Weiner, *Cybernetics*, vol. Part 2. Cambridge, MA: MIT Press, 1961.

[46] L. J. Fogel, *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. NY: John Wiley, 1999.

[47] M. R. Genesereth and N. J. Nilsson, *Logical Foundations of Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann, 1987.

[48] M. L. Minsky, *The Society of Mind*. New York: Simon and Schuster, 1985.

[49] E. B. Carne, *Artificial Intelligence Techniques*. Washington D.C.: Spartan Books, 1965.

[50] D. R. Hofstadter, *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. New York: Basic Books, 1995.

[51] R. E. Marks, “Breeding Hybrid Strategies: Optimal Behavior for Oligopolists,” *Proc. 3rd Intern. Conf. Genetic Algorithms*, J. D. Schaffer (ed.), Morgan Kaufmann, San Mateo, CA, pp. 198-207.