Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art

J. David Schaffer
Philips Labs
Briarcliff Manor
NY 10510

Darrell Whitley
Colorado State Univ.
Fort Collins
CO 80523

Larry J. Eshelman
Philips Labs
Briarcliff Manor
NY 10510

Abstract

Various schemes for combining genetic algorithms and neural networks have been proposed and tested in recent years, but the literature is scattered among a variety of journals, proceedings and technical reports. Activity in this area is clearly increasing. We provide an overview of this body of literature drawing out common themes and providing, where possible, the emerging wisdom about what seems to work and what does not. Combinations have been both supportive (they are used sequentially) and collaborative (they are used simultaneously). Supportive combinations typically involve using one of these methods to prepare data for consumption by the other—for example, using a genetic algorithm to select features for use by neural network classifiers. Supportive combinations have already achieved some success on real world tasks. Collaborative combinations typically involve using the genetic algorithm to determine the neural network weights or the network topology or both. Using a genetic algorithm as a replacement for back propagation does not seem to be competitive with the best gradient methods (e.g., quickprop). Where gradient or error information is not available, however, genetic algorithms may be a promising learning method. Many methods have been proposed for evolving network topologies, but comparison studies are very rare. There seems to be a trend away from direct encodings of the network connections in the chromosomes toward more compact encodings that then expand into the complete network topology in a process analogous to ontogeny. A few studies, often in the *artificial life* domain, have focused on the natural interplay between evolution and learning. Work combining genetic algorithms and neural networks appears to be in an early and very creative phase that should begin to shift towards a more disciplined search for effective methods.

0-8186-2787-5/92 $3.00 © 1992 IEEE

1 Introduction

Genetic Algorithms (GAs) and Neural Networks (NNs) represent two technologies that are currently enjoying a burgeoning of interest among computer scientists and engineers. The interest in these computational schemes seems to have been stirred both by recent advances in our understanding of their dynamic behavior and the realization that they can be harnessed to solve some very challenging problems. Although both were inspired by information processing schemes used by nature, the time constants involved are very different: nervous systems typically adapt over very short time periods while evolution does so over numerous generations. Recently, researchers have begun to consider whether a combination of the two might provide a synergy with more problem solving power that either of them alone. The preliminary work in this direction is the subject of this survey. Where there is a sufficient body of work with similar focus, we have tried to pull out essential similarities and differences; we have not described each work in detail. However, much of the work represents unique foci with many novel details, and here we have tried to give enough detail that the reader can judge the implications.

We have assumed that the reader has some familiarity with both genetic algorithms and neural networks, but have included pointers to seminal works and good introductions for the reader who is new to one or both of these fields.

The interest in neural networks stems from the capabilities of these machines to perform complex computational tasks such as pattern recognition, associative recall and learning. The idea that an assemblage of relatively simple computational elements working in parallel can give rise to emergent behavior (Forrest 1990) that is robust in the face of failure of some of the elements holds the promise of a new and useful class of computing machines. The discovery of training procedures that are capable of leading such a machine towards a desired behavior has provided programmability. All the elements for the emergence of a new and powerful technology appear to be present. The roots of the neural network field are often traced from the landmark paper by McCulloch and Pitts (1943), through the perceptron work in the 1950s and 1960s (Rosenblatt 1958; Minsky and Papert 1969), and finally to the recent reawakening of interest sparked by the new models of Hopfield (1982), the PDP research group of Rumelhart and McClelland (1986), and Hinton (1989). For a good introduction to the major neural network models, see Lippmann (1987).

Yet attempts to harness these capabilities, while yielding some notable successes, have also encountered some difficulties. For instance, back propagation of error signals (Hinton 1989), a popular training procedure, is known to implement a gradient descent which has the drawbacks of being slow for large problems and being susceptible to becoming stuck in local minima. Furthermore, it is a supervised learning procedure that requires the trainer to provide a correct response for every input used during training. Besides being rather unbiological, this requirement is also sometimes impractical. Back propagation's speed and robustness are sensitive to several of its control parameters and the best parameters to use seem to vary from problem to problem. When the network topologies involve recurrent connections (feedback) back propagation methods cannot be applied with the same confidence. Furthermore, there is yet no known approach for specifying an appropriate topology for a new problem. Several researchers have begun to seek robust methods for overcoming these

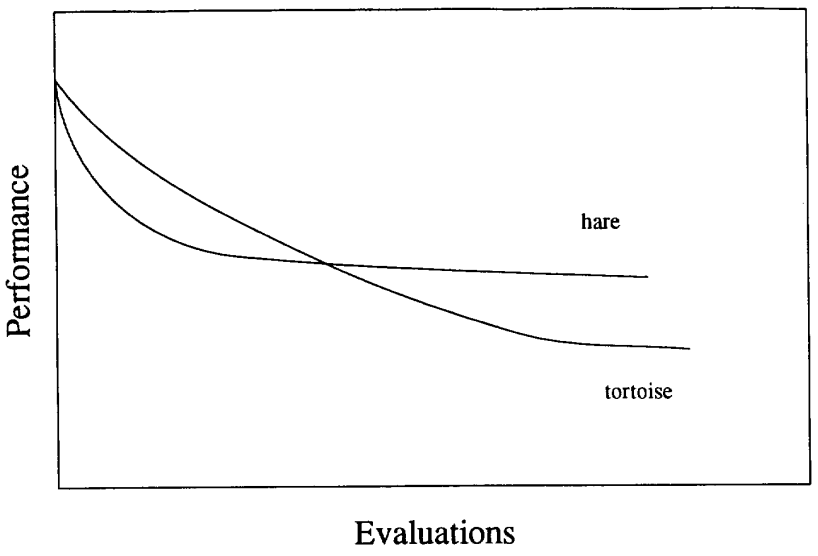


Figure 1: Tortoise and hare behavior

kinds of problems. One such method may be genetic algorithms.

Using a simulation of evolution in a digital computer to solve daunting search problems has been tried since the 1960s (Bremermann 1962; Fogel Owens and Walsh 1966; Reed Toombs and Barricelli 1967; Bagley 1967; Rechenberg 1984). In the mid 1970s, Holland (1975) introduced and analyzed the expected behavior of a particular genetic algorithm that employed a population of trial solutions, fitness-proportional reproduction and the production of offspring by crossover (recombination), inversion and mutation. Theoretical analyses suggest that genetic algorithms can quickly locate high performance regions in extremely large and complex search spaces. Furthermore, some natural insensitivity to noisy feedback is expected because of the distributed and repeated sampling in the population. These expectations are based on the insight that genetic algorithms simultaneously balance a preservation of building blocks (constituent parts of the objects being tested) that have consistently been found in the better than average parents while they generate and test new building blocks in the offspring (Goldberg 1989; Holland 1992).

While experimental work has demonstrated that genetic algorithms are effective search techniques, it has also begun to illustrate some of their limitations. Genetic algorithms are known to be sensitive to control parameters (e.g., population size, rates of recombination and mutation, methods of selection). A user can, by altering the genetic operators, produce different algorithms that one might compare to the tortoise and the hare. (See Figure 1.) A hare rapidly exploits the best building blocks in the early populations, but by doing so, looses genetic diversity and eventually stagnates. A tortoise, on the other hand, makes less rapid progress early on, but by maintaining more genetic diversity, may eventually surpass the hare by locating even better combinations of building blocks. We hasten to point out that the best form of genetic algorithm to employ for a given problem depends on the complexity

of the search space and the goals of the search process. If the computation time needed for the tortoise to surge ahead is beyond your resource limits, a hare is a better choice (Schaffer and Eshelman 1991).

Another problem of particular relevance when applying genetic algorithms to neural networks is one we call *competing conventions*. This problem has been noted by several researchers and good discussions of its appearance in the context of combinations of genetic algorithms and neural networks (COGANNs) are provided by Radcliffe (1990; 1991) and Whitley, Starkweather and Bogart (1990). A genetic algorithm operates on strings often called chromosomes or genotypes. Before evaluating a given genotype, it is first mapped into a solution to the task at hand, called a phenotype. If this mapping is many-to-one, then different genotypes map into the same, or equivalent phenotypes even though their genotypes are quite different. To illustrate, consider a genotype that is a concatenation of the link weights in the simple neural network shown in Figure 2. The genotypes (Figure 2a) map into the phenotypes shown below them (Figure 2b). Note that the only difference in the phenotypes is the switching of the two hidden nodes. Since permuting the hidden nodes of a feed forward network will not alter its function, the two phenotypes in Figure 2 will perform the same function and exhibit the same fitness. However, the arbitrariness of the representation (the convention) is unknown to the genetic algorithm which must operate on the very different genotypes. Crossover between these two genotypes in Figure 2a is unlikely to produce an offspring with a successful recombination of the parents' building blocks; it may, for example, produce an offspring with two hidden neurodes[1] that compute nearly the same function of the inputs and therefore the offspring will lack the ability to compute what the other hidden neurode computed in the parents. Competing conventions in a problem representation means that an error surface which is unimodal to an algorithm such as back-propagation can become multimodal to a genetic algorithm, where each peak corresponds to a different convention. Crossovers among parents utilizing the same convention are likely to be successful while crossovers between dissimilar conventions are unlikely to be effective as suggested in Figure 2c. The number of competing conventions grows exponentially with respect to the number of hidden neurodes, since each permutation representing a different ordering of hidden neurodes and represents a different convention. The competing conventions problems may render crossover essentially impotent as a search operator, at least until the population has converged on a single convention (a process biologists call competitive exclusion). However, this process will almost invariably be accompanied by unwanted convergence among loci where more search in needed (premature convergence). In spite of the growing concern with this problem, Hancock (1992) presents results that suggest it may be less of a problem than many suppose. These results may be attributable to Hancock's use of small populations and high selection pressure leading to early convergence to a single convention.

Like any search algorithm, the strength of a genetic algorithm comes from its sampling bias; the only unbiased algorithms are exhaustive enumeration and random search. However, any algorithm that employs a bias to guide its future samples can be mislead in a search space with the right structure. There is always an Achilles heal. Genetic algorithms are

[1] We use the term neurode in this paper rather than any of the other commonly used terms: neuron (it properly refers to living nerve cells), unit, or node (too vague).

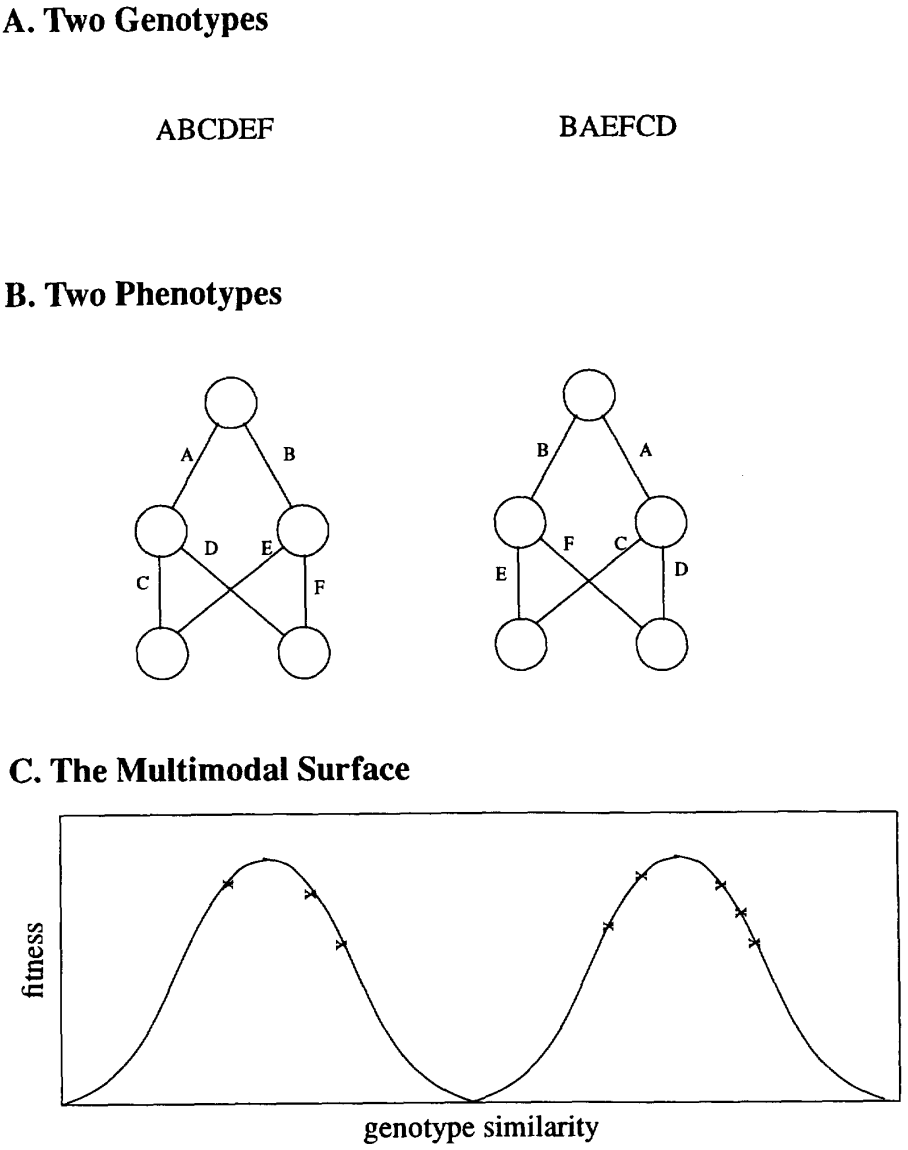


Figure 2: An illustration of competing conventions

broadly applicable because their bias is relatively general and "weak" compared to many other search algorithms. It is nonetheless important to characterize the kinds of problems that can potentially mislead genetic search. The class of GA-deceptive problems has begun to be identified (Goldberg 1987; Liepins and Vose 1991; Das and Whitley 1991; Forrest and Mitchell 1991). At this time, however, it is unknown to what extent deceptiveness tends to be present in different representations.

For this review, we have chosen an engineering focus. By this we mean that approaches to combinations of genetic algorithms and neural networks will be examined from the point of view of building artifacts with useful behaviors. Hence, certain related literature will not be surveyed. This includes the biological literature on population genetics and the literature on neurophysiology including selectionist models of neural development and learning (e.g., Changeux 1980; Conrad Kampfner and Kirby 1988; Edelman 1987). ¿From an engineering perspective, we are unconcerned that the models used for neural networks may be gross simplifications of the complexities of real neurons or that the symbolic strings processed by genetic algorithms are only analogically related to the complexities of real chromosomes. There is also a body of work that combines genetic algorithms with other computational formalisms (e.g., classifier systems (Booker Goldberg and Holland 1990)) which we will not cover, even though it can often be shown that these formalisms are, in specific cases, computationally equivalent to neural networks (Davis 1989a; Davis 1989b). Finally, there is a growing body of work on artificial life which sometimes combines genetic algorithms and neural networks. To the extent that this work focuses on modeling aspects of living systems it is outside the scope of this review. However, to the extent that we feel a useful method has been employed that may contribute to the building of engineering artifacts, we will mention it here. The choices have necessarily been somewhat idiosyncratic. For other reviews of work combining genetic algorithms and neural networks, often with slightly different focus, the reader may consult Mühlenbein and Kindermann (1989), Belew, McInerney and Schraudolph (1990), Radcliffe (1990), Rudnick (1992), Weiss (1990), or Yao (1992).

That evolution and learning could work synergistically together was the main point of a short but influential paper by Hinton and Nowlan (1987). Although Darwinian theory does not allow the inheritance of acquired characteristics (Lamarkian evolution), learning (acquired behaviors) can still influence the course of evolution. This notion is referred to as the Baldwin effect (Hinton and Nowlan 1987; Belew McInerney and Schraudolph 1990) and provides some insight into the nature of the synergy expected from combining genetic algorithms and neural networks. An ability to learn eases the burden on evolution. Evolution only has to get close to the goal: learning can fine tune the behavior. Conversely, evolution can ease the burden of learning by initializing new generations with behaviors well tuned to those aspects of the environment that remain fixed.

Another way of comparing these paradigms is to see to what extent evolution and learning might be able to achieve the same adaptation. In their preliminary study, Mühlenbein and Kindermann (1989) used an adaptive heuristic critic to do the learning in networks that played an iterated prisoner's dilemma and compared their results with those of Axelrod who used a genetic algorithm (Axelrod 1987) to evolve strategies. Their results show their "phenotype learning" (neural network) quickly adapting to specific opponents, whereas Axelrod's "genotype learning" seemed to more broadly explore the space of strategies. They assert that both types of learning seem to be equally powerful, but that a proper division of labor between evolution and learning can be more powerful yet.

For this review, we have divided work combining genetic algorithms and neural networks into two broad categories. In *supportive combinations* one of these technologies (GAs or NNs) usually plays the role of the primary problem solver while the other plays a supporting role, either setting up the task for solution or analyzing the solution once it have been achieved.

They may also be used independently on the same task. In *collaborative combinations* the genetic algorithm and neural network function together to solve the problem. Among collaborative approaches, the two main groupings are those that use the genetic algorithms as the learning method for a neural network and those that use a genetic algorithm to define the topology for a neural network (which in turn often employs a local learning method such as back propagation).

2 Supportive Combinations

2.1 Using Neural Networks to assist Genetic Algorithms

Most researchers have found it more natural (and nature-like) to use genetic algorithms to assist neural nets than vice versa. An important exception is the XROUT system developed by Kadaba, Nygard and Juell (Kadaba and Nygard 1990; Kadaba et al. 1991) to solve the Vehicle Routing Problem (VRP). In the VRP, there is a known set of stop points with demands for services, and a fixed fleet of vehicles for servicing the stops. The problem is to assign the stops to vehicles and to specify the orders in which the vehicles visit the stops so as to minimize the distance traveled. Kadaba et al. developed a hybrid system in which the genetic algorithm played two roles: (1) finding a set of good parameters that a heuristic procedure uses to produce high-performance assignment stop points; (2) finding a good set of selection-heuristics for finding the minimum tour. The parameters encoded in the chromosome for finding the stop points are an initial set of seed assignments that heuristics try to improve upon. The parameters encoded in the chromosome for finding the minimum tour determine the selection-insertion heuristics (e.g., nearest neighbor, cheapest insertion, etc.) for each stop point. They also used this approach for traveling salesman problems (Nygard and Kadaba 1990).

Kadaba et al. used neural nets to produce an initial population for the two genetic algorithm searches. The underlying concept is that there appears to be some natural groupings among the problems and that certain sets of heuristics make better starting points for some groups than for others. The neural network's job is to learn this grouping and suggest starting points for the genetic algorithm. In both cases half the population is initialized randomly and half from parameters produced by the neural net. Both neural nets are pattern associators, matching the descriptors of the incoming problem with good parameters. The neural networks are trained using back propagation.

They compared the performance of XROUT with four alternative, well known routing algorithms on 25 problems. XROUT outperformed the other algorithms for all the problems, as measured in terms of distance traveled.

2.2 Using Genetic Algorithms to assist Neural Networks

We have divided the ways genetic algorithms have been used to assist neural networks into three categories, according to what stage they are used in the process: (1) Using a genetic algorithm to select features or to transform the feature space used by a neural net classifier;

(2) using a genetic algorithm to select the learning rule or the parameters that control learning in a neural net; and (3) using a genetic algorithm to analyze a neural net.

2.2.1 Data preparation

Often the key to getting good results with a pattern classifier lies as much with how the data is presented as with the classifier used. The genetic algorithm has been used to prepare the data in two ways: (1) transforming the feature space and (2) selecting a subset of relevant features.

The first method, transforming the feature space, has been mainly applied to nearest-neighbor type algorithms, although it is applicable to certain types of neural net classifiers as well. Kelly and Davis (1991) used a genetic algorithm to find rotations of a data set and scaling factors for each attribute that improve the performance of a k nearest neighbors (KNN) classification algorithm. By letting a genetic algorithm choose the rotation and scaling parameters, the data is aligned in a manner such that intraclass differences are diminished and interclass differences are magnified. They suggest that similar techniques might work with Kanerva associative memory networks.

A second approach is to restrict the feature set. Often a restricted feature set will improve the performance of a neural network classifier as well as reduce computation requirements. Several researches have used genetic algorithms to select the features. This is one problem where the genetic algorithm bit string representation is very natural—each locus indicating the presence or absence of the feature. The main drawback is the computation time required to train each neural network classifier using the features specified by the chromosome.

Chang and Lippmann (1991) used a genetic algorithm to reduce the feature set for a KNN classifier for a complex speech recognition task. They were able to reduce the features by a factor of five (from 153 to 33 features). They also used a genetic algorithm to create new features — i.e., polynomial functions of the original features — for a KNN classifier. This is somewhat akin to transforming the feature space.

Brill, Brown, and Martin (1992) also used a genetic algorithm to select the feature set for a KNN classifier. However, their ultimate goal was a reduced feature set for a counterpropagation network. They found that even though they used a KNN to evaluate the feature sets in their genetic algorithm search, the feature sets selected also worked well for counterpropagation networks. The reason for using the KNN classifier for the genetic algorithm search is, of course, that evaluation (training) is much faster with a KNN network. On the other hand, a counterpropagation net is faster as a final classifier.

Guo and Uhrig (1992) have used a genetic algorithm to select features for a lateral feedback network, a modified version of back propagation. The problem was to train a set of neural networks to monitor accident scenarios in a nuclear power plant. They found that the performance of the genetic algorithm was strongly dependent upon the fitness function. In particular, they found that not only training error must be taken into account, but also the number of inputs selected (the fewer the better). Furthermore, they found it important to scale the fitness function over time, so that selection pressure is weak at the beginning of the search and increases over time. Brill et al. also used a fitness function that took into

account the number of inputs selected.

Preliminary work to identify feature sets for encoding Chinese characters is presented by Hsu and Wu (1992). The goal of this work is to develop a neural network that predicts the next character based on the last several, allowing the user to simply select a character from a menu rather than having to key enter it.

Instead of using a genetic algorithm to select features for a neural net, Wilson (1990) used a genetic algorithm to evolve feature predicates for a neural net that was used to solve the six-bit Boolean multiplexer problem. Each predicate computed the logical conjunct of some set of the features, and was the input to a perceptron. The chromosome encoded 16 predicates. For the 6 multiplexor input lines, each predicate was encoded using '11' to mean the predicate was connected, '00' to mean that the predicate was connected but the value was inverted, and '10' and '01' to mean that input of the multiplexer was ignored by the predicate. This is equivalent to selecting weights for a completely connected first layer from {-1,0,+1}. For each chromosome a perceptron was trained on random input strings, using the average score over the entire training period of 100 epochs. The network achieved excellent performance, but more trials were required to solve the problem than were needed for other methods.

2.2.2 Evolving network parameters and learning rules

Belew, McInerney and Schraudolph (1990), before using a genetic algorithm to find good initial weights for a back propagation network, used a genetic algorithm to find good values for the learning rate and momentum parameters. They found that the genetic algorithm consistently converged on learning rates much higher than conventional wisdom calls for (i.e., 2.5 as opposed to 0.1). They conjectured that this was due to their restricting the number of learning epochs to a low number relative to the difficulty of the problem (200 epochs for a six-bit symmetry problem).

Harp, Samad and Guha (1989) also found that the genetic algorithm selected higher than recommended learning rates, but that it was problem dependent. Using the genetic algorithm to determine the topology for a back propagation net, they also let it set the back propagation learning rate (which could vary among neurode groups). In addition, they included a parameter in the chromosome that determined the exponential decay for the learning rate parameter as a function of the training epoch, thus allowing the learning rate to change during training. On a digit recognition task the genetic algorithm favored a learning rate of 0.4 with 0.0 decay. On the XOR problem, on the other hand, good individuals had very high initial learning rates (6.4 and 12.8) but also higher decay rates (0.3 and 0.4). In both cases, networks were rewarded for speed of learning (reaching criterion in a minimum number of epochs).

Schaffer Caruana and Eshelman (1990), when using the genetic algorithm to determine the topology for a back propagation net, also used the genetic algorithm to set the back propagation learning parameters. In addition to the learning rate and momentum, they included in the chromosome a parameter that controlled the range of the initial weights (e.g., for a 0.5 value, all initial weights would be chosen randomly between -0.5 and +0.5).

For a 4-bit coding problem with a maximum of 2000 training epochs, the genetic algorithm favored relatively low learning rates (0.25 and 0.5) although these were among the highest allowed in their coding. It also favored low initial weight ranges (0.125 and 0.25). Eshelman has found (in an unpublished study) when using a genetic algorithm to set these three back propagation parameters for a 4-bit parity problem that the genetic algorithm consistently favored large initial weight ranges (e.g., 4.0) and moderate learning rates (e.g., 0.6) when given a limited number of training epochs (e.g., 100). This was especially apparent on nets of more than one hidden layer. Although a net with large initial random weights often did very poorly, it on occasion did extremely well.

Instead of simply having the genetic algorithm discover good back propagation learning parameters, another possibility, explored by Chalmers (1990), is to have the genetic algorithm discover the learning algorithm. In other words, the chromosome encodes the procedure for changing the connection strengths of the network over time based on past performance. Chalmers constrained the learning rule to be a function of information local to the connections—i.e., activation of the input neurode, activation of the output neurode, the training signal on the output neurode, and the current connection strength between two neurodes. The learning rule was expressed as the linear function of the four dependent variables and their six pairwise products. The chromosome encoded the coefficients for the ten variables and a scaling factor. The learning rules were evaluated by testing them on a diverse set of linear separable mapping tasks, ranging from two to seven inputs, but always with a single output neurode, and no hidden layers. The genetic algorithm found a number of learning rules that worked well, variants of the delta-rule being among the best. Chalmers also found that if only a few tasks were used to train a network, the rules evolved by the genetic algorithm were not as general as rules evolved with many (e.g., 20) tasks, and thus failed to do as well on new tasks. Chalmers concluded that he did not expect this approach to have much promise for supervised learning for feedforward networks, where we already have good learning rules, but speculated that it might be usefully extended to other types of networks.

2.2.3 Using genetic algorithms to explain and analyze neural networks

Instead of using the genetic algorithm to build better neural networks, a few researchers have used genetic algorithms to help explain or analyze neural networks. As Eberhart and Dobbins (1991) point out, one of the barriers to acceptance of neural networks is the "lack of explanation facilities similar to those available in most expert systems." In order to explore the 'decision surface' of a neural network, they used the genetic algorithm to discover what they call 'codebook vectors', i.e., input patterns that "result in maximum or nearly maximum activation values for a given output neurode." The input vectors are represented in the chromosome by a set of real values between 0.0 and 1.0. The genetic algorithm was used to discover three different types of codebook vectors: (1) maximum activation vectors—the output node is activated; (2) minimum activation vectors—the output node is off; and (3) decisions vectors—the output node is at the decision threshold. They used multiple runs of the genetic algorithm with different random seeds to find a set of vectors of each type. They illustrate this approach for an XOR network and for an appendicitis diagnostic network.

Both networks were trained using back propagation. They suggest that a large set of these codebook vectors could serve as the basis for a neural network explanation facility. Further extension of this work is presented by Eberhart (1992). A similar approach using ART-1 networks is described by Caudell (1992) who also suggests taking "undecided" cases to an oracle for proper classification and then adding these new cases to the training set. This provides one method of augmenting meager training sets in domains were training sets are difficult or expensive to acquire.

Suzuki and Kakazu (1991) also used a genetic algorithm to help analyze a neural network. In their case they wanted to analyze the basins of attraction of a correlational associate memory model. At the basin the recall process displays threshold and monotonous transition phenomena. They try to model these phenomena by determining a 'characteristic measurement', which they assume can be modeled as a polynomial function of degree 2. They use a genetic algorithm to discover the optimal coefficients for the characteristic measurement.

2.3 Using genetic algorithms and neural networks independently on the same task

Schizas, Pittichis and Middleton (1992) compare feed forward neural networks, the Kohonen self-organizing feature map neural network, a genetic algorithm with a classifier system, and a K-means algorithm on a task calling for the classification of clinical electromyographs (EMGs). After commenting on the relative performance of each method[2] on their task, they suggest a diagnostic system that uses all these classifiers (except the K-means which did poorly on this non-linearly separable problem) and reports to the user the votes of each method on unknown cases. The hope is that idiosyncrasies of any one system may be compensated for by the group.

3 Collaborative Combinations

3.1 Genetic Algorithms for Neural Network learning

3.1.1 Background and Motivation

The idea that neural networks can be trained using genetic algorithms has occurred to many researchers. Typically, this involves optimizing the weights in a neural network with a predefined topology. One straightforward way of doing this is to encode each weight in the network as a binary substring; the entire binary encoding would be a string composed of these concatenated substrings. Each member of the genetic population would therefore specify a complete set of weights for the neural network. Fitness can be calculated by summing the squared errors over a set of training data.

[2]Schizas et al. deal with a multiclass or multiconcept task. The reader may also be interested in the comparisons among learning methods for single concepts by Wnek et al. (1990).

There are several basic arguments that suggest it may be advantageous to apply genetic algorithms to the neural network weight optimization problem. For example, it is argued that genetic algorithms have the potential to produce a global search of weight space and thereby to avoid local minima. Also, it may be advantageous to apply genetic algorithms to problems where gradient information is difficult or costly to obtain. This implies that genetic algorithms can potentially be applied to reinforcement learning problems with sparse feedback, for training fully recurrent neural networks, or for training networks with nondifferentiable transfer neurodes.

The disadvantage that seems obvious with genetic algorithms is one of time scale. Since natural evolution is slow, wouldn't learning via an evolutionary paradigm also be slow? This also gives rise to a related scaling problem: can results achieved on small problems be extended to larger problems?

Researchers have applied genetic algorithms to several different types of problems and have made many different kinds of modifications to the genetic algorithm in order to improve performance. It is useful to review some of the typical applications and algorithm modifications in order to evaluate what has been accomplished as well as to establish an agenda for future research.

3.1.2 Foundation work in training neural networks

Bergman and Kerszberg (1987) used a genetic algorithm employing only selection and mutation to decide if prespecified links should be excitatory or inhibitory. They observed that their task, employing feedback links and requiring the learning of successive inputs that differed only by a scale factor, could not be learned unless they restricted the search space somewhat (e.g., biased the weights towards inhibitory links, biased the links toward feed forward). Perhaps the first published work using a genetic algorithm with crossover to learn the weights in a feed forward neural network was by Whitley (1989). These experiments involve the use of genetic algorithms using crossover to optimize small feed-forward neural networks for some of the basic test problems in neural network learning: exclusive-or, encoders and adders. In general, the genetic algorithm handled these small problems, but was somewhat slow and did not always converge to an acceptable solution.

Montana and Davis (1989) report the first successful application of a genetic algorithm to a relatively large neural network problem. The problem involved classification of passive sonar data from arrays of underwater acoustic receivers. The network they trained was composed of approximately 500 connections. However, the genetic algorithm they used differed in several functional ways from traditional genetic algorithms as described by Goldberg (1989). One difference was that the Montana and Davis algorithm was a type which Syswerda (1989) has referred to as a "steady-state" genetic algorithm. This change was not completely novel, since this is essentially the same as the GENITOR algorithm used by Whitley (Whitley and Kauth 1988; Whitley 1989a). Instead of having generational replacement, offspring are produced one or two at a time and then compete for inclusion in the population. In the GENITOR algorithm, each string is evaluated and the population is sorted by ranking strings in terms of their evaluations. A random selection function with a linear bias towards the higher ranked strings is used to stochastically choose two parents

for recombination. The parents are recombined so as to produce a single offspring (i.e., two offspring are generated and one is randomly discarded). After the offspring is evaluated it replaces the lowest ranked string in the population and is inserted into its appropriate rank location (Whitley and Kauth 1988). In general, one or two offspring can be created and replacement can be probabilistic such that lower ranking strings are typically replaced.

The other major differences in the algorithm used by Montana and Davis are as follows. First, the encoding is real-valued instead of binary. Each parameter (weight) is represented by a single real value and so that recombination can only occur between weights. Second, a small population is used (e.g., 50 individuals). Third, the mutation rates were higher than those used by most traditional genetic algorithms.

The algorithm of Montana and Davis (1989) also provided an option that improved offspring using back-propagation. Thus, the algorithm could be run in a hybrid mode, combining both genetic search and hill-climbing by gradient search. However, Montana and Davis report that the genetic algorithm without hill-climbing produced results as good or better than the hybrid version of the algorithm. They also report that the genetic algorithm produced results superior to back propagation for this application.

Whitley et al. (1990) modified the GENITOR algorithm to reflect the algorithm changes used by Montana and Davis. Using a real-valued encoding, a population of 50 and mechanisms to promote higher mutation rates, they also found that this modified genetic algorithm produced results competitive with back propagation. These results were obtained on both a neural network that adds two two-bit numbers and a relatively large signal pulse detection problem. The signal pulse detection data was processed using a network of approximately 500 connections and training times were somewhat faster than back propagation.

Porto and Fogel (1990) also used an evolutionary scheme with real coded parameters and no crossover on simulations of autonomous underwater vehicle navigation tasks. They report comparable, if somewhat slower learning than observed with back propagation.

The work of Whitley, Dominic and Das (1991) suggests that the genetic algorithms which use real valued encodings, small populations and relatively high mutation rates may produce a search dominated by stochastic genetic hill-climbing. This interpretation was arrived at by shrinking the population from 50 down to 1 string, while at the same time reducing the use of crossover and increasing the use of mutation. Generally the algorithm converged faster as the population size was shrunk, although the probably of actually reaching a viable solution dropped by about 10%. The strong role of genetic hill-climbing in the algorithm which was studied may also explain how this and similar algorithms are able to achieve relatively good results in spite of the *competing conventions* problems. The use of small populations, the strong reliance on mutation and the (generally) monotonic nature of the search produced by steady state genetic algorithms allows the search to avoid exploring too many *competing conventions* and to focus search around those partial solutions which become highly ranked in the population. We also refer to this type of algorithm as a "genetic hill-climber" to distinguish it from more conventional genetic algorithms. The claim, often seen in the literature, that genetic algorithms result in a global search therefore needs to be seriously considered on a case by case basis. If a genetic based algorithm is using hill-climbing, then one cannot automatically assume that the algorithm is also producing a global search.

3.1.3 Training neural networks without gradient information

While a few applications of genetic hill-climbers to neural network weight optimization have produced results roughly competitive with standard back propagation (Montana and Davis 1989; Whitley Starkweather and Bogart 1990) there are supervised training paradigms, such as cascade correlation using quickprop (Fahlman and Lebiere 1990), which are significantly faster than back propagation. There are domains, however, where genetic algorithms can make a unique contribution to neural network learning. In particular, because genetic algorithms do not require or use derivative information, the most appropriate applications are problems where error gradient information is unavailable or costly to obtain.

Most neural network applications for supervised learning use feed-forward networks with sigmoid transfer functions or radial basis functions. These choices make gradient information relatively easy to obtain. However, if one wishes to use other, more complex kinds of transfer neurodes, such as *product neurodes* or if one wished to train fully recurrent networks, then computing gradient information becomes far more costly. For these kinds of networks, genetic algorithms may represent a viable alternative. Similarly, training networks in situations where only sparse reinforcement is available is a difficult learning problem because one does not know in advance what the "correct" output of the network should be at each time step; therefore, the derivative information needed to drive gradient descent methods is not directly available. This means that genetically training networks for neurocontrol applications is also an appropriate and reasonable application.

3.1.4 Genetic approachs to reinforcement learning and neurocontrol

For reinforcement learning applications the set of target outputs that correspond to some set of inputs used to train the net are not known a priori. Rather, the evaluation of the network is performance based. Most existing algorithms attempt to convert the reinforcement learning to a supervised learning problem by indirectly or heuristically generating a target output for each input. Some approaches compute an inverse of a system model. The system model maps inputs (current state and control actions) to outputs (the subsequent state). Given a target state, the inverse of the system model can be used to generate actions, which can then be used as a output target (the appropriate action) for a separate controller. This general description is applicable to methods such as "back propagation through time." "Adaptive critic" methods use a separate evaluation net that learns to predict or evaluate performance at each time step. The prediction can then be used to heuristically generate output targets for an "action" net which controls system behavior. Note, however, that both of these methods compute target outputs and the resulting gradients either indirectly or heuristically. (The problem of building the system model has also been approached using a combination of a genetic algorithm and a neural network by Kargupta and Smith (1991)).

Genetic algorithms can be directly applied to reinforcement learning problems because genetic algorithms do not use gradient information. Genetic algorithms only require a relative measure of performance for each set of weight vectors that is evaluated. Whitley et al. (1992) have successfully applied genetically training neural nets to controlling an inverted pendulum (Anderson 1989), a chaotic nonlinear bioreactor control problem (Ungar

1991) and a "plant controller" problem that was included in the 1988 American Control Conference Showcase (Anderson and Miller 1991). These results suggest that genetic reinforcement learning is competitive with the adaptive-critic AHC architecture as used by Anderson (1987) for reinforcement learning. The algorithms have been compared according to several measures. First, the consistency, speed and reliability of learning itself can be evaluated. A second measure of performance is related to generalization, or how well it handles inputs that are not part of the training data. In both cases, the genetic algorithm displays good performance.

It is interesting to note that the network structures used by Littman and Ackley (1991; Ackley and Littman 1992) to study the interaction between learning and evolution are related to network structures used in control applications. Like the AHC algorithm, Ackley and Littman's evolutionary reinforcement learning model has both an evaluation net and an action net. But whereas the AHC algorithm trains the evaluation net using temporal difference methods, evolutionary reinforcement learning uses a genetic algorithm to train the evaluation net. Both algorithms use the output of the evaluation net to train the action net using a form of *reinforcement back propagation.* While no one has done so, this evolutionary reinforcement learning model could also be applied to more traditional kinds of neurocontrol applications.

In a similar vain, but using a slightly different approach, Nolfi, Elman and Parisi (1990) have combined the evolution of a critic network with the adaptation of an action network using the critic's output. Each agent that must locate food deposits in a simulated world, has a feedforward network consisting of four input neurodes, seven hidden neurodes and four output neurodes, with full connections between layers. Two of the input neurodes are sensory neurodes and two of the output neurodes are action neurodes. The other two input neurodes receive the output from the previous time step, and the other two output neurodes are interpreted as predictors of the next sensory input. The chromosome encodes the initial weights of the network. Back propagation is used to change the weights from the two prediction neurodes and the weights from the hidden neurodes to the input neurodes. In other words, only the weights from the hidden neurodes to the two action output neurodes are not changed. The interesting thing about this network is that, unlike the Ackley and Littman model, there is no mechanism for reinforcement. Instead of reinforcing 'good' behavior, learning tries to model the world. Nolfi et al. found that the agents with learning performed better than simpler agents without the two additional output neurodes—the sensory prediction neurodes—and learning.

Perhaps the most ambitious proposal for combining evolution and learning without access to an outside teacher has included cultural aspects: parents pass on their acquired knowledge to the next generation. In an article providing a detailed analysis of the Hinton and Nowlan model, Belew (1989a) suggests that the model might profitably be extended to include cultural learning. (Alternatively, one can view Belew's cultural model as introducing a Lamarkian element into evolution.) Belew (1989b) suggested two ways for injecting culture into the model. The first alternative is to allow successful parents to confer their advantage to their offspring—not genetically, but by biasing their chance of guessing correctly the correct links in the network. The second alternative is to have successful individuals broadcast this new information not to their offspring but to randomly selected individuals in the next

generation. One effect of this change to the model is that the population finds and converges more quickly upon the fit network. Also the cultural model has an advantage if the fitness function is nonstationary. The broadcast model, however, is not as effective as the lineage model unless the new information is broadcast to a large number of individuals.

3.1.5 Fully recurrent neural networks for neurocontrol

Several other researchers have done work that relates in one way or another to neurocontrol (de Garis 1989; 1990a; 1990b; Torreele 1991; Ichikawa and Sawa 1992; Beer and Gallagher 1992). Wieland (1990; 1991) has used genetic algorithms to train a fully recurrent net for the pole balancing problem as well as to more complicated versions of the problem. Since a fully recurrent network is being trained, only pole position and cart position are supplied as inputs; the fully recurrent net can learn to compute its own velocity information. Among some of the more difficult versions of the pole balancing problems studied by Wieland is a problem where two-poles are attached to the cart and simultaneously balanced and another where a jointed pole (i.e., one pole attached to the top of another) is balanced. In these experiments the value of the output neurode was used to determine the magnitude of the force applied to the cart.

In Wieland's experiments a binary encoding of weights is used. The networks have 6 to 10 neurodes, with 8 weights fanning in to each neurode. Each weight was coded as a byte (8 bits) and bytes were used to represent the values -1, $-\frac{253}{256}$ $-\frac{251}{256}$ ... $\frac{253}{256}$, 1. The size of the binary encodings used by Wieland (approximately 400 bits to 700 bits) is somewhat larger than the networks others have trained with binary encodings. Wieland reports that using a population of 511 six-node networks the genetic algorithm consistently solved the single pole problem in at most 12 generations (approximately 6000 recombinations). The jointed pole problem required 20 to 30 generations using a population of 512 ten-node recurrent networks. The two-pole problem was the most difficult, requiring over 150 generations using a population of 2084 strings representing ten-node recurrent networks.

The binary encoded genetic algorithms used by Wieland as well as the mapping of functionality to specific nodes are different from the algorithms used by Montana and Davis (1989) and by Whitley et al. (1990). The implementation is very close to the standard genetic algorithm (Holland 1975; Goldberg 1989). There are also subtle differences in the way Wieland coded the neural network optimization problems. The weight increments which were decoded from the binary substrings are much smaller than those used by previous researchers, which has the effect of increasing the probability that weight summation values will tend to be concentrated in the active range of the the sigmoid transition function. Small changes such as this may account for Wieland positive results using binary encodings, but these kinds of questions merit further attention.

Torreele (1991) has also worked with fully recurrent networks. Torreele studied recurrent nets with linear threshold neurodes for a temporal multiplexer problem and for a blinker problem. For this latter problem, there are no inputs and the network must learn to output a signal for n time steps and then to output no signal (i.e., go inactive) for n time steps. Torreele reports successful learning on these small problems (4 neurodes in each network).

3.1.6 Genetic learning and constructive learning

Genetic learning algorithms based on the cascade correlation algorithm are a new addition to the literature on training neural networks using genetic algorithms. The current volume contains papers by Potter (1992) and by Karunanithi et al. (1992).

The Cascade correlation learning architecture dynamically builds a suitable cascade structure by training and installing one hidden neurode at a time until the problem is successfully learned. Two important features of the Cascade Correlation Learning Architecture are: (1) it trains only one layer of weights at any time and (2) it trains and adds one hidden neurode to the neural network at a time. These features are important considerations when using genetic algorithms because they can be exploited not only to improve scalability but also to eliminate the *competing conventions* problem among hidden neurodes. If only one hidden neurode is being trained by the genetic algorithm, then there are not multiple solutions presenting competing conventions, although it is still possible for the same basic solution to be scaled differently.

For purposes of the current discussion there are two major issues that future researchers who wish to use genetic algorithms to train neural networks should consider. First, if researchers are doing supervised learning with feed-forward network, how do their methods compare with back propagation algorithms, cascade correlation and the methods reported by Potter (1992) and Karunanithi et al. (1992). Their results on hard problems appear to offer faster training times than has previous been possible using genetic algorithms to train neural networks. Yet, the results are still inferior to using conventional cascade correlation. Second, although there has been little work in this direction to date, comparisons should include not only learning speed, but also generalization. Finally, are there domains in which genetic learning combined with this type of constructive approach has advantages over the Cascade Correlation Learning Architecture using quickprop? Training networks with nondifferential neurodes, such as product neurodes, might represent such an application.

3.2 Genetic Algorithms to Specify Neural Network Topology

Perhaps the most intuitively obvious way to combine a genetic algorithm with a neural network it to try to imitate nature by having the genotype code the architecture or topology (how many neurodes to use and how to connect them) of the network and then use another local learning method to fine tune the weights. Judging by the number of published experiments of this type, this intuition has occurred to many.

Any attempt to implement this approach must devise a way to handle each of these tasks:

A. A genotype representation must be devised and an attendant mapping from genotype to phenotype (analogous to ontogeny) must be provided.

B. There must be a protocol for exposing the phenotype to the task environment.

C. There must be a learning method (usually local) to fine tune the network function. (Local learning is sometimes omitted by researchers who give the genetic algorithm the entire burden of learning by including the connection strengths in the genotype.)

D. There must be a fitness measure.

E. There must be a method for generating new genotypes from old ones (a GA).

Rather than providing descriptions of individual research, we will survey this area by treating each of these factors in turn pointing out similarities and differences.

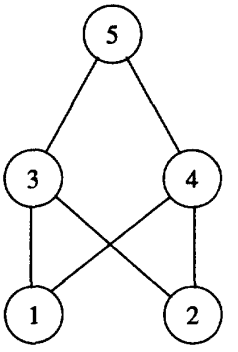
3.2.1 A. Representation

One key feature of a network architecture specification is the directness of the chromosome representation. At one extreme, the most direct representations explicitly encode each connection in the genotype. For instance, one can simply concatenate the rows of a binary connection matrix as illustrated in Figure 3. A connection matrix is an $N \times N$ array in which element n_{ij} is zero if there is no link between neurodes i and j and one if there is a link. N is the total number of neurodes in the network. Any connectivity pattern among the N neurodes can thus be expressed as a bit string of length N^2. The obvious drawback to this scheme is the explosive increase in the genotype length as networks grow. Nevertheless, this approach was used by Miller and Todd in preliminary studies with Hegde (1989) that showed that task-specific architectures could be evolved for small tasks. Miller and Todd have also explored more complex issues such as the evolution of associative learning (Todd and Miller 1990) and sensitization and habituation (Todd and Miller 1991). More recently, Rudnick (1992) used a direct encoding, but found that generalization performance improved if he restricted the links represented to only one layer in a otherwise fixed architecture.

Direct encodings that included both links and weights (limited to ± 1) were used by Bornholdt and Graudenz (1991) to learn Boolean functions, by Collins and Jefferson (1991) for an "ant farm" task, by Martí (1992a) for finding a winner-take-all network, by KrishnaKumar (1992) for control and modeling of dynamic systems, and by Lindgren et al. (1992) to learn regular languages. Koza and Rice (1991) also use a direct encoding for both links and weights, but theirs is in terms of Lisp S-expressions. They can thus include connectivity and real valued weights (i.e. not limited to bit strings), but their representation requires a special crossover operator that preserves the syntax of proper S-expressions. Elias (1992) used bit strings to code addresses of neurode connections for a linked list representation. The addresses coded not only the neurode but also the location on its dendritic branch which simultaneously served to specify the weight of the connection (synapses close to the "cell body" have greater weight than those more distant). Also coding connections as addresses, Das and Whitley (1992) focus on sparse distributed memories which are equivalent to neural networks. Another approach that uses a direct representation was used by Whitley, Starkweather and Bogart (1990) where they began with an initial architecture with all possible feed-forward connections. This architecture was then trained to learn a task. After learning, the genetic algorithm was used to find which links could be eliminated in order to achieve a specific learning objective — in this case, faster learning. After finding initial reductions,

A conection matrix

```
0 0 1 1 0
0 0 1 1 0
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
```



The chromosome for the whole matrix

0011000110000010000100000

The chromosome for the feed forward portion only

0110110011

Figure 3: A direct coding of a connection matrix

the genotype could be redefined to cover the reduced networks and the genetic algorithm could be restarted. The new genotype contained a zero or one for each link in the reduced network, not every possible link. The evaluation of each emaciated network began with the previously learned weight, thus saving considerable training time. Part of the motivation for this particular study was to find smaller nets in order to reduce the number of connections needed for a hardware implementations.

One aspect of a genotype representation that may be exploited if done right (or cause problems if done wrong) is the adjacency relationship among the genes. A pair of coadapted genes are less likely to be disrupted by traditional crossover if they are close together. An approach to optimizing the ordering of the genes along the chromosome was proposed by Martí (1992b) in which an outer genetic algorithm whose genotype controls the genotype of an inner genetic algorithm which searches for the network topology.

One approach to a less direct encoding is to code in the genotype only the values of a few parameters which specify the topology within a limited set of architectures. This method was used by Caudell (Caudell and Dolan 1989; Caudell 1990) in which the parameters used were peculiar to the class of optical hardware interconnection being implemented. The conclusion of this preliminary work was that the parameters used contained too little specificity of the connectivity and hence the genetic algorithm could not locate useful building blocks. The resulting network performance was not competitive. A more general parametric encoding, for strictly feed forward architectures, was tested by Schaffer, Caruana and Eshelman (1990) in which there was a parameter coding the number of neurodes in each hidden layer. The maximum number of hidden layers was fixed as was the maximum number of neurodes in

each layer. Complete connectivity between adjacent layers was assumed. In addition, each hidden layer could be eliminated with a single bit and several back propagation parameters were also included in the genotype. They observed the consistent evolution of an effective architecture for a toy problem. Hancock and Smith (1991) used a unique parameterization for a face recognitions task. Their input layer consisted of five types of neurodes sensitive to five types of input signal, all arranged in a Cartesian grid (5 x 11 x 11 = 605 input neurodes). The genotype parameters controlled the number and connectivity (receptive field location and input neurode type(s)) of the neurodes in the hidden layer. All neurodes in the hidden layer were connected to the output layer. A more elaborate parameterization has been implemented by Harp et al. (Harp Samad and Guha 1989; 1990; Harp and Samad 1991a; 1991b) which allows for feedback connections. They have applied their method to a range of problems including digit recognition and sine wave approximation.

Even less direct encodings call for more elaborate procedures to implement the mapping to the phenotype. Context free grammars were tested by Kitano (1990a; 1992). The number of levels of intermediate symbols was preset, limiting the size of the resulting connection matrix to 2^l where l is the number of levels of intermediate symbols. It is Kitano's intuition that the class of architectures expressible by these methods may embody a structural regularity that will be exploitable for large problems. Good results were observed on simple XOR and 4-x-4 and 8-x-8 encoder/decoder problems. Note that the resulting connection matrices could call for feedback links which had to be explicitly removed if the problem called for a strictly feed forward architecture. A somewhat similar method has been used by Gruau (1992) where the genotype codes links and weights (again limited to ± 1 for learning Boolean functions). This method is also similar to that of Koza and Rice (1991) since the chromosome string encodes a syntactic tree structure and special crossover and mutation operators must be employed that preserve this syntax. Gruau shows how a genetic algorithm using this scheme can learn to scale neural networks for regular Boolean functions with compact genotypes by evolving topologies for 50-bit parity and 40-bit symmetry problems. A novel feature of this work is that the input and output layers are also encoded in the genotype requiring the evaluation method to deal with architectures that may have too few or too many neurodes in these layers. A third indirect method has been proposed by Mjolsness et al. (Mjolsness and Sharp 1986; Mjolsness Sharp and Alpert 1989; Sharp Reinitz and Alpert 1991) based on a model of neural development. To date this method has not been empirically tested for evolving a neural network topology for a specified task.

Lindgren et al. (1992) have begun efforts to quantify the difficulty of their learning tasks with the intention of assessing the impact on scalability. Hence, they began their work deliberately with simple tasks and direct encodings. The indirect representations gain their compactness and hence their improved scalability to larger problems by limiting the space of topologies to those with certain regularities. An important, but still open question is how much regularity can be exploited without eliminating topologies that may be needed for some problems. Since the human genotype contains far fewer genes than the human brain contains neurons (let alone synapses), nature has obviously found such a compromise.

Besides directness, another feature of the genotype-to-phenotype mapping process is its determinism. Most research so far has employed a strictly deterministic process, but Dress (1987; 1990), Bergman and Kerszberg (1987) and Harp et al. have all used some stochasticity

in their processes. While a large degree of indeterminism can obviously complicate search by introducing noise into the evaluation, a small amount may contribute to a more compact representation and reduce the search burden on the genetic algorithm. The genetic algorithm is known to be tolerant of a modicum of noise.

One other feature of the genotype representation is the explicit coding of the spatial relationships among the neurodes. Most researchers leave this unspecified, but some have included it, usually for vision tasks where a grid pattern imposes specific neighborhood relations among the neurodes and this pattern can influence the connectivity, e.g., Harp et al., Carugo (1991).

3.2.2 B. Network performance protocol

Since most of the work to date has involved small tasks with few binary inputs, most protocols have exposed the networks to all possible input patterns from a single task during training. A number of schemes have been tested against more than one task (Harp et al.; Bornholdt and Graudenz 1991; Caudell and Dolan 1989; Kitano 1990b; Whitley Starkweather and Bogart 1990). Some have restricted training to samples drawn from the possible inputs, e.g., for sine waves (Harp et al. 1989), for detecting successive inputs differing only in magnitude (Bergman and Kerszberg 1987), for feature vectors (Dolan and Dyer 1987) and for coding tasks with noise (Schaffer Caruana and Eshelman 1990; Rudnick 1992). Some have used essentially sequential tasks such as pattern invariance (Bergman and Kerszberg 1987), robot walking (de Garis 1989; 1990b), and animats in a simulated environment (Dress and Knisley 1987). While most tests have been on small problems such as XOR and coder/decoder nets, a few systems have been tested on problems of practical utility such as digit recognition (Harp et al. 1989; Schiffmann Joost and Werner 1991), robot walking (de Garis 1990b), face recognition (Hancock and Smith 1991), and control of dynamic systems (KrishnaKumar 1992; Weiland 1991; Whitley Dominic Das and Anderson 1992).

3.2.3 C. Network learning and neurode type

The most common practice has been to evolve topologies for feed forward networks of logistic neurodes and to use back propagation for local learning. A few have forsaken local learning and had the genetic algorithm learn the weights and the topology together (Collins and Jefferson 1991; Koza and Rice 1991; Bornholdt and Graudenz 1991; Gruau 1992). Some have deviated from strictly feed forward designs by allowing within layer links (Miller Todd and Hegde 1989; Whitley Starkweather and Bogart 1990) feedback links (Bergman and Kerszberg 1987; Lehar and Weaver 1987; Collins and Jefferson 1991; Bornholdt and Graudenz 1991; de Garis 1990a), and competitive learning or Hebbian learning (Dolan and Dyer 1987; Todd and Miller 1990). Non-logistic-function neurodes have also been tried: Collins and Jefferson (1991), Koza and Rice (1991), and Bornholdt and Graudenz (1991) used threshold neurodes, while Bergmann and Kerszberg (1987) used linear neurodes, Lehar and Weaver (1987) used Grossberg field neurodes, Dress and Knisley (1987) used their own biologically motivated neural model, and Elias (1992) used a dendritic branching scheme that is not only more like the biological neurons, but is also implementable in silicon with desirable properties.

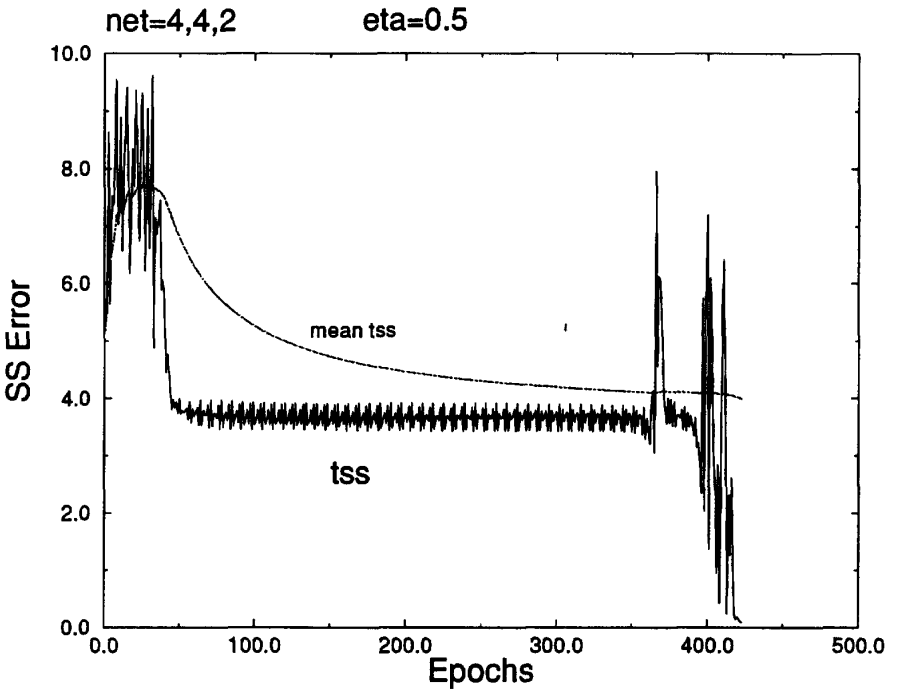


Figure 4: An illustration of non-monotonic learning with backprop

3.2.4 D. Network performance measures

Here again there is a predominant practice: genotype fitness is equated with least mean square (LMS) error of the output neurodes on the training set. Where speed of learning is considered important, it is usually encouraged by setting a limit on the number of epochs (sweeps through the training set) when performance is measured. Some have proposed using the integral of the squared error over epochs. Both of these methods can be problematical given the non-monotonic nature of back propagation learning curves. Figure 4 shows a learning curve which is not uncommon. Note that besides the ups and downs of the error, there is a sudden and drastic drop just at the end. (The run terminated at that point because a performance threshold had been achieved.) Repeated runs exhibited similar patterns, but the epoch at which the final successful learning occurred varied considerably. Such behavior obviously makes assessing performance at a preselected epoch problematical. Furthermore, the integral of the error may make it hard to distinguish between a network that learned successfully just before halting and one that did not. This is not to suggest that LMS error cannot be used, only that care should be exercised. If a genetic algorithm is manipulating many network parameters, and it is impossible to monitor all networks that are evaluated, then untoward effects like those in Figure 4 may well be operating unbeknown to the researcher.

When Whitley, Starkweather and Bogart (1990) sought to eliminate links from an already-trained network, they recognized that an explicit tradeoff between training time and network complexity would be needed. They allowed the simpler networks, which presumably would have to suffer greater loss of accuracy than their more complex competitors, more epochs to adjust their weights. In the general software tool developed by Harp, Samad and Guha (1989), the user was allowed to specify the performance measure by providing relative weights on features such as accuracy, complexity and speed of learning.

If one wishes to demonstrate the feasibility of evolving a topology that can learn a task, then LMS error on the training set may be adequate, but Schiffmann, Joost and Werner (1991) suggested using the performance of the trained networks on a separate test set as a measure of generalizability of the topology, a method that was demonstrated by Schaffer, Caruana and Eshelman (1990) and Rudnick (1992).

3.2.5 E. The genetic algorithm

A number of researchers have used a procedure where a single parent genotype is randomly mutated to produce a single child. If the child performs worse than the parent then it is discarded, otherwise the parent is discarded and the child becomes the parent for the next offspring. We hasten to point out that while this process is evolution-like, it is not a genetic algorithm; it is simple random hillclimbing.

With a population of parents, even without crossover, one has an algorithm of the class that has been studied for some time under the names *evolution strategy* (Europe) (Rechenberg 1984) and *evolutionary programming* (USA) (Fogel and Atmar 1990). This class of algorithm exhibits a simulated-annealing-like behavior[3] that can be quite successful and avoids the difficulties of competing conventions mentioned earlier. This problem has been anticipated by Bergman and Kerszberg (1987) and has been noted by Montana and Davis (1989) and Whitley et al. (1990).

Nevertheless, the majority of researchers have used some form of genetic algorithm including crossover. We have noted the special crossover operators required by some representations (Koza and Rice 1991; Gruau 1992). In addition, Mühlenbein and Kindermann (1989) have used a distributed genetic algorithm where genotypes can only mate among their near neighbors and have reported better performance than when allowing panmictic mating (random pairing among the whole population). Gruau (1992) has emulated this form of mating on a serial machine. Another dimension worth considering is the selection scheme. Rudnick (1992) found the most dramatic improvement over his baseline genetic algorithm occurred when he tested Eshelman's population-elitist selection (Eshelman 1991).

Studies rigorously comparing different approaches are as yet very rare. However, this situation will change as research moves beyond demonstrations of feasibility to more concentrated attempts to find effective computational paradigms. To aid this process we encourage workers to report all significant details of their genetic algorithm implementations. The performance of genetic algorithms is known to be quite sensitive to details such as population

[3]The simulated-annealing-like behavior is that offspring that are worse than their parents can survive, and the probability of this occurring is higher for initial populations than it is for latter, more fit populations.

sizes, methods of selection for reproduction and replacement, the vigor of crossover, rates and kinds of mutation, etc. This obviously also applies to the implementation details of the neural networks. Such reporting is essential to replication, a procedure that will become more widely practiced as more impressive performances are achieved.

3.3 Other Collaborative combinations

Finally, we mention three collaborative combinations that do not fit the two previous categories. They represent methods for combining the adaptive strategies of both genetic algorithms and neural networks into a single adaptive system.

Ackley (1987a; 1987b) developed a system called SIGH in which two layers of neurodes are taken to represent (1) a bit string representation of a solution to an optimization problem and (2) a population of individuals who try to influence the solution. Each bit in the solution is set by an election process among the population who in turn have their positions (activation levels) adjusted in reaction to the election results. These processes are stochastic and use Boltzmann probability distributions influenced by temperature parameters. In addition, during each cycle the current solution is evaluated against the optimization task and its performance used to adjust the link weights between the two layers of neurodes. He reported good performance for this method on a highly multipeaked problem and reasonable performance across a range of other problem types.

A recent paper by Gonzalez-Seco (1992) has proposed a neural network implementation of the genetic algorithm. The idea is that genetic learning may be a useful capability for a neural network implemented brain. The tasks of wiring this subsystem to the rest of the "brain" such as sensory inputs and memory systems is left for future work, but neural subsystems for computing reproduction and crossover are presented. Somewhat similar is the use of a neural network as the crossover operator within a genetic algorithm. This idea is proposed by Shonkwiler and Miller (1992) for solving optimization problems with nonlinear constraints.

4 Conclusions

The use of genetic algorithms to assist neural networks or vice versa may be the most promising area of combination for quick return on the investment of effort. Preparing data by one method for digestion by the other has been used successfully for several real world problems already.

In the case of weight optimization, genetic algorithms are often compared against gradient methods for supervised learning problems that involve training feedforward neural networks. In most studies genetic algorithms have proven to be inferior to the best gradient based methods; in the few cases where a genetic algorithm has been shown to be competitive with back propagation, the genetic algorithm is still slower than more advanced training methods, such as cascade correlation. Several researchers have tried using the genetic algorithm in a more limited way. The genetic algorithm has been used to find good initial weights for

networks before applying back propagation. Genetic algorithms have also been used to try to optimize the learning rate and momentum term. Overall, the results have been mixed.

Nevertheless, there are learning applications where genetic algorithms can make a very real contribution: specifically these are applications where gradient methods are not directly applicable. Future work should particularly attempt to exploit these kind of applications. This could mean training networks that use more complex kinds of neurode functions where gradient information is more difficult to obtain. And certainly there needs to be more work investigating the use of genetic algorithms to train fully recurrent networks. Similarly, neurocontrol applications and applications that involve only sparse reinforcement as feedback represent domains where genetic algorithms might be usefully employed.

Perhaps the greatest potential for long range impact lies in the use of genetic algorithms to evolve neural network topologies. However, this is also one of the more difficult areas of research where we still have a great deal to learn. To be successful it must be possible to develop indirect representations for neural network architectures that will scale well to networks of significant size; at the same time, scaling up should occur in a way that allows exploitation of useful structures and computational properties that have already been developed with respect to smaller neural network problems. One possibility is that using more sophisticated neural models will allow for the same complex computations to be performed by simpler networks (e.g., Beer 1990).

The main argument against this paradigm is that it will require excessive computing time. Whether the problem with computation time can be solved remains to be seen. Some obvious ways in which the computational time can be reduced include: using specialized neural network hardware, employing the best local learning algorithms available, using improved genetic algorithms and parallel implementations of the genetic algorithm, and by finding the best division of labor between the local and evolutionary learning paradigms to make the best possible use of training time. An interesting idea in this latter domain was proposed by Hillis (1990) when he used two interdependent populations, both evolving concurrently. One population evolved problem solvers whose fitness reflected how well they performed on the test cases they faced. The other population evolved test cases whose fitness reflected how poorly the problem solvers did on them. The dynamics of coevolving populations is relatively new to genetic algorithm research and one that may hold much promise for engineering application.

Acknowledgements

Dr. Whitley's work in this area is supported by NSF grant IRI-9010546.

References

[1] Ackley, D.H. (1987a). *A connectionist machine for genetic hillclimbing.* Boston, MA: Kluwer Academic Publishers.

[2] Ackley, D.H. (1987b). An empirical study of bit vector function optimization. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing* (pp. 170-204). San Mateo, CA: Morgan Kaufmann.

[3] Ackley, D.H. & Littman, M.L. (1992). Interactions between learning and evolution. In C. Langton, C. Taylor, J.D. Farmer and S. Rasmussen (Eds.), *Artificial Life II* (pp. 487-509). Reading, MA: Addison Wesley.

[4] Anderson, C. W. (1987). *Strategy Learning with Multilayer Connectionist Representations*, Technical report TR87-509.3, Waltham, MA: GTE Labs.

[5] Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine, 9,* 31-37.

[6] Anderson, C. W., & Miller, W. T.(1990). A challenging set of control problems. In T. Miller, R. Sutton and P. Werbos (Eds.), *Neural Networks for Control* (pp. 475-510). Cambridge, MA: MIT Press.

[7] Austin S. (1991, October). Genetic neurosynthesis. *Proceedings of AIAA Aerospace VIII.* Baltimore, MD.

[8] Axelrod, R. The evolution of strategies in the iterated prisoner's dilemma. In L. Davis (Ed.), *Genetic algorithms and simulated annealing* (pp. 32-41). San Mateo, CA: Morgan Kaufmann.

[9] Bagley, J.D. (1967). The behavior of adaptive systems which employ genetic and correlation algorithms. Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

[10] Ball N. (1990). Adaptive signal processing via genetic algorithms and self-organizing neural networks. *Proceedings IEEE Workshop on Genetic Algorithms, Simulated Annealing and Neural Networks.* University of Glasgow, Scotland.

[11] Beer, R.D. (1990). *Intelligence as adaptive behavior.* New York: Academic Press.

[12] Beer, R.D. & Gallagher, J.C. (1992, Summer) Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior.* (pp. 91-122).

[13] Belew, R.K. (1989a). When both individuals and populations search: Adding simple learning to the genetic algorithm. In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 34-41). San Mateo, CA: Morgan Kaufmann.

[14] Belew, R.K. (1989b, September). *Evolution, learning and culture: Computational metaphors for adaptive algorithms.* CSE Technical report CS89-156, La Jolla, CA: University of California at San Deigo.

[15] Belew, R.K. & McInerney, J. (1989, May). *Using the genetic algorithm to wire feed-forward networks.* Technical abstract, Computer Science & Engineering Dept., La Jolla, CA: University of California at San Diego.

[16] Belew, R.K., McInerney, J. & Schraudolph N.N. (1990, June). *Evolving networks: Using genetic algorithms with connectionist learning.* CSE technical report CS90-174, La Jolla, CA: University of California at Dan Diego.

[17] Bergman, A. & Kerszberg, M. (1987). Breeding intelligent automata. In M. Caudell and C. Butler (Eds.), *IEEE international conference on neural networks* (pp. II-63 - II-70). San Diego, CA: IEEE.

[18] Booker, L.B., Goldberg, D.E. & Holland, J.H. (1990). Classifier systems and genetic algorithms. In J.G. Carbonell (Ed.), *Machine learning paradigms and methods* (pp. 235-282). Cambridge, MA: MIT Press.

[19] Bornholdt, S. & Graudenz, D. (1991). *General asymmetric neural networks and structure design by genetic algorithms.* DESSY 91-046 Hamburg, Germany: Deutsches Electronen-Synchrotron.

[20] Bremermann, H.J. (1962). Optimization through evolution and recombination self-organizing systems. Yovits, Jacobi & Goldstein (Eds.), *Self-organizing systems* (pp. 93-106). Washington, D.C.: Spartan Books.

[21] Brill, F.Z., Brown, D.E. & Martin, W.N. (1992). Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks, 3* (2), 324-328.

[22] Carugo, M.H. (1991). *Optimization of parameters of a neural network, applied to document recognition, using genetic algorithms.* Nat. Lab. technical note No. 049/91, Eindhoven, The Netherlands: N.V. Philips.

[23] Caudell, T.P & Dolan, C.P. (1989). Parametric connectivity: Training of constrained networks using genetic algorithms. In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 370-374). San Mateo, CA: Morgan Kaufmann.

[24] Caudell, T.P. (1990). Parametric connectivity: Feasibility of learning in constrained weight space. *IEEE international joint conference on neural networks* (pp. I-667 - I-675). Hillsdale, NJ: Lawrence Erlbaum.

[25] Caudell, T.P. (1992). Genetic algorithms as a tool for the analysis of adaptive resonance theory neural network sets. (in this volume).

[26] Caudill, M. (1991, March). Evolutionary neural networks. *AI Expert*, pp. 28-33.

[27] Chalmers, D.J. (1990). The evolution of learning: An experiment in genetic connectionism. In D.S. Touretsky, J.L. Elman, T.J Sejnowski & G.E. Hinton (Eds.) *Proceedings of the 1990 connectionist models summer school* (pp. 81-90). San Mateo, CA: Morgan Kaufmann.

[28] Chang, E.J. & Lippmann, R.P. (1991). Using genetic algorithms to improve pattern classification performance. In R.P. Lippmann, J.E. Moody & D.S. Touretsky (Eds.), *Advances in neural information processing 3* (pp. 797-803). San Mateo, CA: Morgan Kaufmann.

[29] Changeux, J.-P. (1980). Genetic determinism and epigenesis of the neuronal network: Is there a biological compromise between Chomsky and Piaget? In M. Piatelli-Palmarini (Ed.), *Language and learning - The debate between Jean Piaget and Noam Chomsky* (pp. 184-202). New York: Routledge & Kegan Paul.

[30] Collins, R. & Jefferson, D. (1991). An artificial neural network representation for artificial organisms. In H.-P. Schwefel & R. Männer (Eds.), *Parallel problem solving from nature* (pp. 269-263). Berlin: Springer-Verlag.

[31] Conrad, M., Kampfner, R.R. & Kirby, K.G. (1988). Neuronal dynamics and evolutionary learning. In M. Kochen & H.M. Hastings (Eds.) *Advances in cognitive science* (pp. 169-189). Boulder, CO: Westview Press

[32] Das, R. & Whitley, D. (1991). The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 166-173). San Mateo, CA: Morgan Kaufmann.

[33] Das, R. & Whitley, D. (1992). Genetic sparse distributed memory. (in this volume).

[34] Davis, L. (1989a). Mapping Neural Networks into Classifier Systems In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 375-378). San Mateo, CA: Morgan Kaufmann.

[35] Davis, L. (1989b). Mapping Neural Networks into Classifier Systems In D.S. Touretsky (Ed.), *Advances in neural information processing 1* (pp. 49-56). San Mateo, CA: Morgan Kaufmann.

[36] Dolan, C.P. & Dyer, M.G. (1987). Toward the evolution of symbols. In J.J. Grefenstette (Ed.), *Second international conference on genetic algorithms* (pp. 123-131). Hillsdale, NJ: Lawrence Erlbaum.

[37] Dolan, C.P. & Dyer, M.G. (1987). *Symbolic schemata in connectionist memories: Role binding and the evolution of structure.* Technical report UCLA-AI-87-11, Los Angeles, CA: UCLA AI Laboratory.

[38] Dorsey, R.E., Johnson, J.D. & Mayer, W. (1992). A genetic algorithm for the training of feedforward neural networks. In A. Whinston & J.D. Johnson (Eds.), *Advances in artificial intelligence in economics, finance and management* JAI Press.

[39] Dress, W.B. (1987). Darwinian optimization of synthetic neural systems. In M. Caudill & C. Butler (Eds.), *IEEE international conference on neural networks* (pp. III-769 - III-776). San Diego, CA: IEEE.

[40] Dress, W.B. & Knisley, J.R. (1987). A Darwinian approach to artificial neural aystems. *IEEE Conference on Systems, Man, and Cybernetics* (pp. 572-577). New York: IEEE.

[41] Dress, W.B. (1990). *Electronic life and synthetic intelligent systems.* Technical report, Oak Ridge, TN: Instrumentation and Controls Division, Oak Ridge National Laboratory.

[42] Eberhart, R.C. & Dobbins, R.W. (1991). Designing neural network explanation facilities using genetic algorithms. *IEEE international joint conference on neural networks* (pp. 1758-1763). Singapore: IEEE.

[43] Eberhart, R.C. (1992). The role of genetic algorithms in neural network query-based learning and explanation facilities. (in this volume).

[44] Edelman, G.M. (1987). *Neural Darwinism.* New York: Basic Books.

[45] Elias, J.G. (1992). Genetic generation of connection patterns for a dynamic artificial neural network. (in this volume).

[46] Eshelman, L.J. (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination In G. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 265-283). San Mateo, CA: Morgan Kaufmann.

[47] Fahlman, S. & Lebiere, C. (1990). *The cascade-correlation learning architecture.* CMU-CS-90-100, Pittsburgh: PA: Carnegie-Mellon University.

[48] Fenanzo, A.J. (1986, July). Darwinian evolution as a paradigm for AI research. *SIGART Newsletter*, (97), 22-23.

[49] Fogel, L.J., Owens, A.J. & Walsh, M.J. (1966). *Artificial intelligence through simulated evolution.* New York: John Wiley & Sons.

[50] Fogel, D.B. & Atmar, J.W. (1990). Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics* 63, 111-114.

[51] Fontanari, J.F. & Meir, R. (1991, November). Evolving a learning algorithm for the binary perceptron. *Network*, *2* (4), 353-359.

[52] Forrest, S. (Ed.), (1990). *Emergent computation.* Amsterdam: North Holland.

[53] Forrest, S. & Mitchell, M. (1991). The performance of genetic algorithms on Walsh polynomials: Some anomalies and their explanation. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 182-189). San Mateo, CA: Morgan Kaufmann.

[54] de Garis, H. (1989). *WALKER, A genetically programmed, time dependent, neural net which teaches a pair of Sticks to walk.* Technical report Fairfax, VA: Center for AI, George Mason University.

[55] de Garis, H. (1990a). *Genetic programming: Building nanobrains with genetically programmed neural network modules.* In *IEEE international joint conference on neural networks* (pp. III-511 - III-516). San Diego, CA: IEEE.

[56] de Garis, H. (1990b). Genetic programming: Evolution of a time dependent neural network module which teaches a pair of stick legs to walk. IN L.C. Aiello (Ed.), *Proceedings of the 9th European Conference on Artificial Intelligence* (pp. 204-206). San Mateo, CA: Morgan Kaufmann.

[57] de Garis, H. (1990c). BRAIN building with GenNets. *Proceedings of international neural networks conference, ICNN-90* (pp. 1036-1039). Paris: Kluwer Academic Publishers.

[58] Goldberg, D.E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In L. Davis (Ed.), *Genetic algorithms and simulated annealing* (pp. 74-88). San Mateo, CA: Morgan Kaufmann.

[59] Goldberg, D.E. (1989). Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison Wesley.

[60] Gonzalez-Seco, J. (1992). A genetic algorithm as the learning procedure for neural networks. *IEEE international joint conference on neural networks* (pp. I-835 - I840). Baltimore, MD: IEEE.

[61] Guo, Z. & Uhrig, R.E. (1992). Use of genetic algorithms to select inputs for neural networks. (in this volume).

[62] Gruau, F. (1992). Genetic synthesis of Boolean neural networks with a cell rewriting development process. (in this volume).

[63] Hancock, P. (1990). GANNET: Design of a neural network for face recognition by genetic algorithm. *Proceedings of IEEE Workshop on Genetic Algorithms, Simulated Annealing and Neural Networks.* Glasgow: University of Glasgow.

[64] Hancock, P.J.B. & Smith, L.S. (1991). GANNET: Genetic design of a neural net for face recognition. In H.-P. Schwefel & R. Männer (Eds.), *Parallel problem solving from nature* (pp. 292-296). Berlin: Springer-Verlag.

[65] Hancock, P.J.B. (1992). genetic algorithms and permutation problems: a comparison of recombination operators for neural structure specification. (in this volume).

[66] Harp, S.A., Samad, T. & Guha, A. (1989). Towards the genetic synthesis of neural networks. In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 360-369). San Mateo, CA: Morgan Kaufmann.

[67] Harp, S.A., Samad, T. & Guha, A. (1990). Designing application-specific neural networks using the genetic algorithm. In D.S. Touretsky (Ed.), *Advances in neural information processing 2* (pp. 447-454). San Mateo, CA: Morgan Kaufmann.

[68] Harp, S.A. & Samad, T. (1991a). Genetic optimization of self-organizing feature maps. *IEEE international joint conference on neural networks* (pp. I-341 - I-346). Seattle, WA: IEEE.

[69] Harp, S.A. & Samad, T. (1991b). The genetic synthesis of neural network architecture. In L. Davis (Ed.), *Handbook of genetic algorithms* (pp. 202-221). New York: Van Nostrand Reinhold.

[70] Heistermann, J. (1990). Learning in neural nets by genetic algorithms. In R. Eckmiller, G. Hartmann & G. Hauske (Eds.), *Parallel processing in neural systems and computers* (pp. 165-168). Amsterdam: Elsevier Science Publishers.

[71] Heistermann, J. (1991). The application of a genetic approach as an algorithm for neural networks. In H.-P. Schwefel & R. Männer (Eds.), *Parallel problem solving from nature* (pp. 297-301). Berlin: Springer-Verlag.

[72] Hillis, W.D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedures. In S. Forrest (Ed.), *Emergent computation* (pp. 228-234). Amsterdam: North Holland.

[73] Hinton, G.E. (1989). Connectionist learning procedures *Artificial Intelligence Journal 40*, 185-234.

[74] Hinton, G.E. & Nowlan, S.J. (1987, June). How learning can guide evolution. *Complex Systems*, pp. 495-502.

[75] Hoffgen, K.-U., Sieman, H.P. & Ultsch, A. (1991). Genetic improvements of feedforward nets for approximating functions. In H.-P. Schwefel & R. Männer (Eds.), *Parallel problem solving from nature* (pp. 302-306). Berlin: Springer-Verlag.

[76] Holland, J.H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

[77] Holland, J.H. (1992, July). Genetic algorithms. *Scientific American*, pp. 66-72.

[78] Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79, 2554-2558.

[79] Hsu, L.S. & Wu, Z.B. (1992). Input pattern encoding through generalized adaptive search. (in this volume).

[80] Ichikawa, Y. & Sawa, T. (1992, March). Neural network application for direct feedback controllers. *IEEE Transactions on Neural Networks*, pp. 224-231.

[81] Janson , D. & Frenzel, J. (in press). Training product unit neural networks with genetic algorithms. *IEEE Expert.*

[82] Kadaba, N. & Nygard, K.E. (1990). Improving the performance of genetic algorithms in automated discovery of parameters. In B.W. Porter & R.J. Mooney (Eds.), *Proceedings of the seventh international conference on machine learning* (pp. 140-148). San Mateo, CA: Morgan Kaufmann.

[83] Kadaba, N., Nygard, K.E. & Juell, P.L. (1991, vol 2, iss 1). Integration of adaptive machine learning and knowledge-based systems for routing and scheduling applications. *Expert Systems with Applications*, pp. 15-27.

[84] Kargupta, H. & Smith, R.E. (1991). System identification with evolving polynomial networks. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 370-376). San Mateo, CA: Morgan Kaufmann.

[85] Karunanithi, N., Das, R. & Whitley, D. (1992). Genetic cascade learning for neural networks. (in this volume).

[86] Keesing, R. & Stork, D.G. (1991). Evolution and learning in neural networks: The number and distribution of learning trials affect the rate of evolution. In R.P. Lippmann, J.E. Moody & D.S. Touretsky (Eds.), *Advances in neural information processing 3* (pp. 804-810). San Mateo, CA: Morgan Kaufmann.

[87] Kelly, J.D. & Davis, L. (1991). Hybridizing the genetic algorithm and the K nearest neighbors classification algorithm. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 377-383). San Mateo, CA: Morgan Kaufmann.

[88] Kitano, H. (1990a). Designing neural network using genetic algorithm with graph generation system. *Complex Systems* 4, 461-476.

[89] Kitano, H. (1990b). Empirical studies on the speed of convergence of neural network training using genetic algorithms. *Proceedings of the national conference on artificial intelligence* (pp. 789-795). Cambridge, MA: MIT Press.

[90] Kitano, H. (1992, March). *Neurogenetic learning: An integrated method of designing and training neural networks using genetic algorithms.* CMU-CMT-92-134 Pittsburg, PA: Carnegie-Mellon University.

[91] Koza, J.R. & Rice, J.P. (1991). Genetic generation of both the weights and architecture for a neural network. *IEEE international joint conference on neural networks* (pp. II-397 - II-404). Seattle, WA: IEEE.

[92] KrishnaKumar, K. (1992). Immunized neurocontrol: Concepts and initial results. (in this volume).

[93] Lehar, S. & Weaver, J. (1987). A developmental approach to neural network design. In M. Caudill & C. Butler (Eds.), *IEEE international conference on neural networks* (pp. II-97 - II-104). San Diego, CA: IEEE.

[94] Liepins, G.E. & Vose, M.D. (1991). Deceptiveness and genetic algorithm dynamics. In G. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 36-50). San Mateo, CA: Morgan Kaufmann.

[95] Lindgren, K., Nilsson, A., Nordahl, M. & Rade, I. (1992). Regular Language Inference using evolving neural networks. (in this volume).

[96] Lippmann, R.P. (1987, April). An introduction to computing with neural nets. *IEEE ASSP Magazine*, pp. 4-22.

[97] Littman, M.L. & Ackley, D.H. (1991). Adaptation in constant utility non-stationary environments. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 136-142). San Mateo, CA: Morgan Kaufmann.

[98] Maričić, B. & Nikolov, Z. (1990). GENNET - system for computer aided neural network design using genetic algorithms. *IEEE international joint conference on neural networks* (pp. I-102- I-105). Hillsdale, NJ: Lawrence Erlbaum.

[99] Martí, L. (1992a). Genetically generated neural networks II: Searching for an optimal representation. *IEEE international joint conference on neural networks* (pp. II-221 - II-226). Baltimore, MD: IEEE.

[100] Martí, L. (1992b). Genetically generated neural networks I: Representational effects *IEEE international joint conference on neural networks* (pp. IV-537 - IV-542). Baltimore, MD: IEEE.

[101] McCulloch, W.S. & Pitts, W. (1943). A logical calculus of ideas imminent in nervous activity *Bulletin Mathematical Biophysics*, 5, 115-133

[102] Miller, G.F., Todd, P.M. & Hegde, S.U. (1989). Designing neural networks using genetic algorithms. In J.D. Schaffer (Ed.), *Third international conference on genetic algorithms* (pp. 379-384). San Mateo, CA: Morgan Kaufmann.

[103] Minsky, M. & Papert, S. (1969). *Perceptrons: An introduction to computational geometry.* Cambridge, MA: MIT Press.

[104] Mjolsness, E. & Sharp, D.H. (1986). *A preliminary analysis of recursively generated networks.* Research Report YALEU/DCS/RR-476, Hew Haven, CT: Yale University.

[105] Mjolsness, E., Sharp, D.H. & Alpert, B.K. (1989). Scaling, machine learning, and genetic neural nets. *Advances in Applied Mathematics, 10*, 137-163.

[106] Moed, M.C. & Saridis, G.N. (1990, September-October). A boltzman machine for the optimization of intelligent machines. *IEEE Transactions on Systems Man and Cybernetics 29* (5), 1094-1102.

[107] Montana, D. (1992 in press). A weighted probabilistic neural network *Advances in neural information processing 4* San Mateo, CA: Morgan Kaufmann.

[108] Montana, D.J. & Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *Proceedings of eleventh international joint conference on artificial intelligence* (pp. 762-767). San Mateo, CA: Morgan Kaufmann.

[109] Montana, D.J. (1991). Automated parameter tuning for interpretation of synthetic images. In L. Davis (Ed.), *Handbook of genetic algorithms* (pp. 282-311). New York: Van Nostrand Reinhold.

[110] Mühlenbein, H. & Kindermann, J. (1989). The dynamics of evolution and learning – Towards genetic neural networks. In R. Pfeifer, Z. Schreter, F. Fogelman-Soulie & L. Steels (Eds.), *Connectionism in perspective* (pp. 173-197). Amsterdam: Elsevier Science Publishers B.V. (North-Holland).

[111] Nolfi, S., Elman, J.L. & Parisi, D. (1990, July). *Learning and evolution in neural networks.* CRL Technical Report 9019, La Jolla, CA: University of California at San Diego.

[112] Nygard, K.E. & Kadaba, N. (1990). Modular neural networks and distributed adaptive search for traveling salesman algorithms. *Proceedings of the SPIE, 1294*, 442-451.

[113] Porto, V.W. & Fogel, D.B. (1990, June). Neural network techniques for navigation of AUVs. *Proceedings of the IEEE Symposium on Autonomous Underwater Vehicle Technology* (pp. 137-141). Washington, DC: IEEE.

[114] Potter, M.A. (1992). A genetic cascade-correlation learning algorithm. (in this volume).

[115] Radcliffe, N.J. (1990). *Genetic neural networks on MIMD computers.* Unpublished doctoral dissertation, University of Edinburgh, Edinburgh, Scotland.

[116] Radcliffe, N.J. (1991). *Genetic set recombination and its application to neural network topology optimization.* Technical report EPCC-TR-91-21, University of Edinburgh, Edinburgh, Scotland.

[117] Rechenberg, I. (1984). The evolution strategy. A mathematical model of Darwinian evolution. In E. Frehland (Ed.), *Synergetics – From microscopic to macroscopic order* (pp. 122-132). Berlin: Springer-Verlag.

[118] Reed, J., Toombs, R. & Barricelli, N.A. (1967). Simulation of biological evolution and machine learning. *Journal of Theoretical Biology*, 17, 319-342.

[119] Reeves, C. & Steele, N. (1991, November). Genetic algorithms and the design of artificial neural networks. *Microarch, 6*(1), 15-20.

[120] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 386-407.

[121] Rudnick, W.M. (1992). Genetic algorithms and fitness variance with an application to the automatic design of artificial neural networks. Unpublished doctoral dissertation, Oregon Graduate Institute of Science and Technology, Portland.

[122] Rumelhart, D.E., McClelland, J.L. & the PDP Research Group. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition volume 1: Foundations.* Cambridge, MA; Bradford Books/MIT Press.

[123] Schaffer, J.D., Caruana, R.A. & Eshelman, L.J. (1990). Using genetic search to exploit the emergent behavior of neural networks. In S. Forrest (Ed.), *Emergent computation* (pp. 244-248). Amsterdam: North Holland.

[124] Schaffer, J.D. & Eshelman, L.J. (1991). On crossover as an evolutionarily viable strategy In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 61-68). San Mateo, CA: Morgan Kaufmann.

[125] Schaffer, J.D., Eshelman, L.J. & Offutt, D. (1991). Spurious correlations and premature convergence in genetic algorithms. In G. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 102-112). San Mateo, CA: Morgan Kaufmann.

[126] Schiffmann, W. & Mecklenburg, K. (1990). Genetic generation of backpropagation trained neural networks. In R. Eckmiller, G. Hartmann & G. Hauske (Eds.), *Parallel processing in neural systems and computers* (pp. 205-208). Amsterdam: Elsevier Science Publishers.

[127] Schiffmann, W., Joost, M. & Werner, R. (1991). Performance evaluation of evolutionarily created neural network topologies. In H.-P. Schwefel & R. Männer (Eds.), *Parallel problem solving from nature* (pp. 274-283). Berlin: Springer-Verlag.

[128] Schizas, C.N., Pattichis, C.S. & Middleton, L.T. (1992). Neural networks, genetic algorithms, and k-means algorithm: In search of data classification (in this volume).

[129] Scholz, M. (1991). A learning strategy for neural networks based on a modified evolutionary strategy. In H.-P. Schwefel & R. Männer (Eds.), *Parallel problem solving from nature* (pp. 314-318). Berlin: Springer-Verlag.

[130] Schwefel, H.-P. (1981). *Numerical optimization of computer models.* Chichester, England: Wiley.

[131] Sebald, A.V. & Fogel, D.B. (1991). Using evolutionary neural networks for arterial waveform discrimination. *IEEE international joint conference on neural networks* (pp. II-A-955) Seattle, WA: IEEE.

[132] Sharp, D.H., Reinitz, J. & Mjolsness, E. (1991, January). *Genetic algorithms for genetic neural nets.* Research report YALEU/DCS/TR-845, New Haven, CT: Yale University.

[133] Shonkwiler, R. & Miller, K.R. (1992). Genetic algorithm/neural network synergy for nonlinearly constrained optimization problems. (in this volume).

[134] Stacey, D.A. & Kremer, S. (1991). The Guelph Darwin project: The evolution of neural networks by genetic algorithms. *IEEE international joint conference on neural networks* (pp. II-A-957) Seattle, WA: IEEE.

[135] Syswerda, G. Uniform crossover in genetic algorithms. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 2-9). San Mateo, CA: Morgan Kaufmann.

[136] Suzuki, K. & Kakazu, Y. (1991). An approach to the analysis of the basins of the associative memory model using genetic algorithms In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 539-546). San Mateo, CA: Morgan Kaufmann.

[137] Todd, P.M. & Miller, G.F. (1990). Exploring adaptive agency II: Simulating the evolution of associative learning. In S. Wilson & J.-A. Meyer(Eds.), *Proceedings of the international conference on simulation of adaptive behavior: ¿From animals to animats* (pp. 306-315). Cambridge, MA: MIT Press.

[138] Todd, P.M. & Miller, G.F. (1991). Exploring adaptive agency III: Simulating the evolution of habituation and sensitization. In H.-P. Schwefel & R. Männer (Eds.), *Parallel problem solving from nature* (pp. 307-313). Berlin: Springer-Verlag.

[139] Torreele, J. (1991). Temporal processing with recurrent networks: An evolutionary approach. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 555-561). San Mateo, CA: Morgan Kaufmann.

[140] Ungar, L.H. (1990). A bioreactor benchmark for adaptive network-based process control. In T. Miller, R. Sutton and P. Werbos (Eds.), *Neural Networks for Control* (pp. 387-402). Cambridge, MA: MIT Press.

[141] Weiss, G. (1990, May). *Combining neural and evolutionary learning: Aspects and approaches.* Report FKI-132-90, Munich: Technische Universitat Munchen.

[142] Whitley, D. (1989a). Applying genetic algorithms to neural network learning. *Proceedings of the Seventh Conference of the Society of Artificial Intelligence and Simulation of Behavior* (pp. 137-144). Sussex, England: Pitman Publishing.

[143] Whitley, D. (1989b). Genetic algorithm applications: Neural nets, traveling salesmen and schedules. In J. Alexander (Ed.), *Proceedings of the 1989 Rocky Mountain Conference on Artificial Intelligence* (pp. 207-216). Denver, CO: Rocky Mountain Conference on AI.

[144] Whitley, D. & Kauth, J. (1988). GENITOR: A different genetic algorithm. Technical Report CS-88-101, Fort Collins, CO: Colorado State University.

[145] Whitley, D., Starkweather, T. & Bogart, C. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14, 347-361.

[146] Whitley, D., Dominic, S. & Das, R. (1991). Genetic reinforcement learning with multilayer neural networks. In R.K. Belew & L.B. Booker (Eds.), *Fourth international conference on genetic algorithms* (pp. 562-569). San Mateo, CA: Morgan Kaufmann.

[147] Whitley, D., Dominic, S., Das, R. & Anderson, C. (1992, in press). Genetic reinforcement learning for neurocontrol problems. *Machine Learning.*

[148] Wieland, A.P. (1990). Evolving controls for unstable systems. In D.S. Touretsky, J.L. Elman, T.J Sejnowski & G.E. Hinton (Eds.) *Proceedings of the 1990 connectionist models summer school* (pp. 91-102). San Mateo, CA: Morgan Kaufmann.

[149] Wieland, A.P. (1991). Evolving neural network controllers for unstable systems. *IEEE international joint conference on neural networks* (pp. II-667 - II-673). Seattle, WA: IEEE.

[150] Wilson, S.W. (1990). Perceptron redux: Emergence of structure. In S. Forrest (Ed.), *Emergent Computation* (pp. 249-256). Amsterdam: North Holland.

[151] Wnek, J., Sarma, J., Wahab, A.A. & Michalski, R.S. (1990). Comparing learning paradigms via diagramatic visualization: A case study in single concept learning using symbolic, neural net and genetic algorithm methods. In W.R. Zbigniew, M. Zemankova & M.L. Emrich (Eds.), *Methodologies for intelligent systems, 5* (pp. 428-437). New York: Elsevier Science Publishing.

[152] Yao, X. (1992). *A review of evolutionary artificial neural networks.* Technical report. Victoria, Australia: Commonwealth Scientific and Industrial Research Organization.