Evolutionary Artificial Neural Networks[1][2]

Xin Yao

Department of Computer Science

University College, The University of New South Wales

Australian Defence Force Academy

Canberra, ACT, Australia 2600

[1]This article is a much extended version of *X. Yao, "Evolutionary artificial neural networks," International Journal of Neural Systems, Vol. 4, No. 3 (September 1993), pp.203–222.*

[2]Published in **Encyclopedia of Computer Science and Technology**, ed. A. Kent *et al.*, Vol. 33, pp.137–170, Marcel Dekker Inc., New York, NY 10016, 1995.

Abstract

Evolutionary artificial neural networks (EANNs) [1] result from combinations of artificial neural networks (ANNs) and evolutionary search procedures such as genetic algorithms (GAs). This article introduces the concept of EANNs, reviews the current state-of-the-art and indicates possible future research directions.

1 Introduction

EANNs refer to a special class of artificial neural networks (ANNs) in which evolution is another fundamental form of adaptation in addition to learning. The evolution in EANNs is often simulated by GAs or other evolutionary algorithms. It is used to perform various tasks, such as weight training, architecture design, learning the learning rules, input feature selection, reinforcement learning, initial weight selection, ANN analysis, etc. One distinct feature of EANNs is their adaptability to a dynamic environment. In other words, EANNs can adapt to an environment as well as changes in the environment. The two forms of adaptation, i.e., evolution and learning in EANNs make their adaptation to a dynamic environment much more efficient. In a broader sense, EANNs can be regarded as a general framework for adaptive systems, i.e., systems that can change their architectures and learning rules according to different environments without human intervention.

Since we are most interested in exploring possible benefits arising from the interactions between ANNs and evolutionary search procedures, instead of in ANNs and evolutionary search procedures themselves, we shall concentrate on the two most popular models of ANNs and evolutionary search procedures in our study, i.e., feed-forward ANNs [2, 3] and GAs [4, 5] without trying to cover all kinds of models. Most discussion in this article is applicable to other models however.

1.1 Artificial Neural Networks

1.1.1 Architectures

An ANN consists of a set of processing elements, also known as neurons or nodes, which are interconnected with each other. It can be described as a directed graph in which each node i performs a function, f_i, of the form

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right) \tag{1}$$

where y_i is the output of the node i, x_j is the jth input to the node, w_{ij} is the connection weight between the node and input x_j, θ_i is the threshold (or bias) of the node, and f_i is the node transfer function. Usually, the node transfer function is a nonlinear function such as a heaviside function, a sigmoid function, a Guassian function, etc.

ANNs can be divided into feed-forward ones and recurrent ones according to their connectivity. An ANN is a feed-forward one if there exists a numbering method which numbers all the node in the network in such a way that there is no connection from a node with a large number to a node with a smaller number. All the connections are from nodes with small numbers to nodes with larger numbers. An ANN is a recurrent one if such a numbering method does not exist.

In Eq.1, each term in the summation only involves one input x_j. High-order ANNs are those that contain high-order nodes, i.e., nodes in which more than one input are involved in some of the terms in the summation. For example, a second-order node can be described as

$$y_i = f_i\left(\sum_{j,k=1}^{n} w_{ijk} x_j x_k - \theta_i\right)$$

where all the symbols have similar definitions as those in Eq.1.

The architecture of an ANN is determined by its topological structure, i.e., connectivity and the transfer function of each node in the network.

1.1.2 Learning

Learning in ANNs belongs to the class of learning by examples. It is also called training in ANNs because the learning is achieved by adjusting the connection weights[1] in ANNs iteratively so that trained (or learned) ANNs can perform certain tasks. Learning in ANNs can roughly be divided into supervised, unsupervised and reinforcement learning. Supervised learning is based on direct comparison between the actual output of an ANN and the desired correct output, also known as the target

[1]Thresholds (biases) can be viewed as connection weights with fixed input -1.

output. It is often formulated as the minimization of an error function such as the total mean square error between the actual output and the desired output. A gradient descent-based optimization algorithm like backpropagation is then used to adjust connection weights in the ANN iteratively in order to minimize the error. Reinforcement learning is a special case of supervised learning where the exact desired output is unknown. It is based only on the information of whether the actual output is correct or not. Unsupervised learning is solely based on the correlations among input data. No feedback information from an ANN is available.

Learning in ANNs is achieved by adjusting connection weights iteratively. The essence of a learning algorithm is the learning rule, i.e., weight-updating rule which determines how connection weights are updated. Examples of popular learning rules include the delta rule, the Hebbian rule and the anti-Hebbian rule [2].

More detailed discussion of ANNs can be found in Hertz *et al.*'s book [2].

1.2 Evolutionary Search Procedures

Evolutionary search procedures are those search algorithms that are developed from ideas and principles of natural evolutionary systems. They are also called evolutionary algorithms in this article although some researchers distinguish between genetic algorithms and evolutionary algorithms. In this article, evolutionary algorithms include genetic algorithms [4, 5], evolutionary programming [6, 7] and evolution strategies [8, 9]. One important feature of all these algorithms is their population-based search strategy. Individuals in a population compete and exchange information with each other in order to perform certain tasks.

Two major applications of evolutionary algorithms are learning and optimization. For an evolutionary learning system, the learning task to be performed by it determines the environment in which the system evolves. A dynamic environment is one that changes over time.

Evolutionary algorithms are particularly good at dealing with large complex spaces

which contain many local optima. They are less likely to be trapped in a bad local minimum than traditional gradient-based search algorithms. They do not depend on gradient information so that they are quite suitable for problems where such information is unavailable. They can even deal with problems where no explicit and/or exact objective function is available. These features make them much more robust than other search algorithms. Bäck and Schwefel [10] gives an overview of evolutionary algorithms for parameter optimization.

1.2.1 Genetic Algorithms

GAs refer to a class of algorithms which can be described by Figure 1, where each algorithm is defined by its encoding method of individuals, its selection mechanism, and its genetic operators.

1. Generate the initial population $G(0)$ at random, and set $i = 0$;
2. REPEAT
   (a) Evaluate each individual in the population;
   (b) Select parents from $G(i)$ based on their fitness in $G(i)$;
   (c) Apply genetic operators to the parents and use the results to form $G(i+1)$;
   (d) $i = i + 1$;
3. UNTIL 'termination criterion' satisfied

Figure 1: Genetic algorithms.

1.3 Evolution in EANNs

Evolution is introduced into EANNs at roughly three levels; the evolution of connection weights, the evolution of architectures and the evolution of learning rules.

The evolution of connection weights introduces an adaptive approach to connection weight training, especially in the reinforcement learning and recurrent network learning paradigm where gradient-based training algorithms experience great difficulties. The evolution of architectures enables EANNs to adapt their architectures to different tasks, i.e., environments without human intervention, and thus provides an evolutionary approach to automatic EANN architecture design. The evolution of learning rules is a process of learning to learn in EANNs where the adaptation of learning rules to an environment is achieved through evolution. It can also be regarded as an adaptive process of automatic learning rule design.

It is obvious that EANNs are powerful adaptive systems. This article is concerned with such systems. Combinations of ANNs and evolutionary algorithms which do not result in such systems, e.g., ANN analysis using GAs [11, 12, 13], will not be covered by this article.

1.4 Organization of the Article

The rest of the article is organized as follows. Section 2 discusses the evolution of connection weights. The aim here is to find a near optimal set of connection weights for an EANN with a fixed architecture through evolution. GAs are most often used in this evolutionary weight training process. Various methods of encoding connection weights and their advantages and disadvantages are discussed in this section. Comparisons between the evolutionary approach and conventional training algorithms, like back-propagation, are also made. The comparisons suggest that fast gradient based training algorithms would be more efficient than evolutionary training algorithms if gradient information is cheaply available, or else evolutionary training algorithms would be a better choice. In general, no single algorithm is an overall winner for all kinds of networks.

Section 3 is devoted to the evolution of architectures, i.e., the evolution which leads to a near optimal architecture for the tasks at hand. It is known that the architecture

of an EANN determines the information processing capability of the EANN. The architecture design has become one of the most important tasks in EANN research and development. Two important issues in the evolution of architectures, i.e., how to encode architectures and which evolutionary search procedure should be used will be addressed in this section. As far as automatic architecture design is concerned, there also exist other non-evolutionary approaches like constructive and destructive learning algorithms [14, 15, 16, 17, 18, 19, 20, 21].

If imagining EANN's connection weights and architectures as their "hardware", it is easier to understand the importance of the evolution of EANN's "software" — learning rules. Section 4 addresses the evolution of learning rules in EANNs and examines the relationship between learning and evolution, e.g., how learning guides evolution and how learning itself evolves. It is demonstrated that EANN's learning ability can be developed and improved through evolution. Although research on this topic is still in its early stages, further studies will no doubt benefit research on both EANNs and machine learning as a whole.

Section 5 first describes a general framework of EANNs in terms of adaptive systems where interactions among three levels of evolution are considered. The framework provides a common basis for comparing different EANN models and algorithms if a broader view of evolutionary search procedures is taken. The section then gives a brief summary of the article. It concludes with a list of suggested readings on EANNs.

2 The Evolution of Connection Weights

Weight training in ANNs is usually formulated as minimization of an error function, such as the total mean square error between target outputs and actual outputs, by iteratively adjusting connection weights. Most training algorithms, like back-propagation (BP) and conjugate gradient algorithms [2, 3, 22, 23], are based on gradient descent search. There have been some successful applications of BP algorithms

in various areas [24, 25, 26]. However, drawbacks of the BP algorithm do exist due to its gradient descent nature [27, 28, 29]. It often gets trapped in a local minimum of the error function and is very inefficient in finding a global minimum if the error function is multimodal and nondifferentiable. A detailed review of the current state of the BP algorithm and other learning algorithms can be found in [2, 3].

One way to overcome gradient-descent-search-based training algorithms' shortcomings is to adopt EANNs, i.e., to formulate the training process as the evolution of connection weights in the environment determined by the architecture and the learning task. Global search procedures like GAs can then be used effectively in the evolution to find a near optimal set of connection weights for the EANN. The fitness of an EANN can be defined according to different needs. Two important factors which often appear in the fitness function are the error between target outputs and actual outputs and the complexity of the EANN. Unlike the case in gradient-descent-search-based training algorithms, the fitness function does not have to be differentiable or even continuous since GAs do not depend on gradient information in search. Because GAs are good at dealing with large, complex, nondifferentiable and multimodal spaces which are the typical space defined by an error function or fitness function, a lot of work has been done on the evolution of connection weights [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64].

The evolutionary approach to weight training in EANNs consists of two major stages. The first stage is to decide the genotype representation of connection weights, i.e., whether in the form of binary strings or not. The second one is the evolution itself simulated by a GA or other evolutionary algorithms, in which genetic operators like crossover and mutation have to be decided in conjunction with the representation scheme. Different representation schemes and genetic operators can lead to very different training performance. A typical cycle of the evolution of connection weights is shown in Figure 2.

1. Decode each individual (genotype) in the current generation into a set of connection weights and construct a corresponding EANN with the weights.

2. Evaluate each EANN by computing its total mean square error between actual outputs and target outputs. (Other error functions can also be used.) The fitness of an individual is determined by the error. The higher the error, the lower the fitness. The optimal mapping from the error to the fitness is usually problem dependent.

3. Reproduce a number of children for each individual in the current generation according to its fitness.

4. Apply genetic operators, such as crossover and mutation, to each child individual generated above and obtain the next generation.

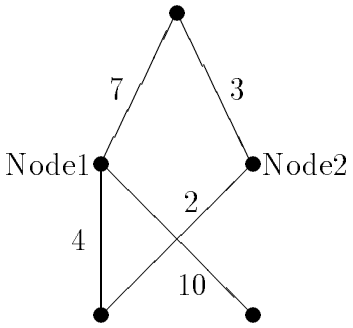
Figure 2: A typical cycle of the evolution of connection weights. (Reprinted with permission from Ref. 1.)

2.1 Binary Representation

Since the binary representation has been shown to be beneficial in GA's search [4, 5], one way to represent connection weights is to encode them in binary strings [29, 30, 32, 41, 42, 45, 56, 57]. In such a representation scheme, each connection weight is represented by a number of binary bits with certain length. An EANN is represented by concatenation of all the connection weights in the network.

A heuristic concerning the order of the concatenation is to put connection weights to the same hidden/output node together. Hidden nodes in EANNs are in essence feature extractors and detectors. Separating inputs to the same hidden node far apart in the binary representation would increase the difficulty of constructing useful feature detectors because they might be destroyed by recombination operators easily.

Figure 3 gives an example of the binary representation of an EANN whose architecture is pre-defined. Each connection weight in the EANN is represented by 4 bits, the whole EANN is represented by 24 bits where the weight `0000` indicates no connection between the two nodes.



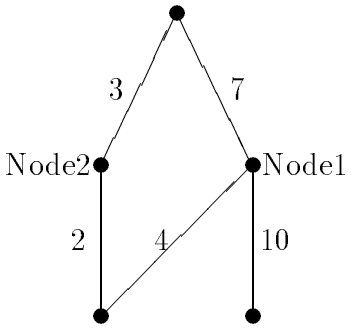
`0100 1010 0010 0000 0111 0011`

(b)

Figure 3: (a) An EANN with connection weights shown; (b) A binary representation of the weights, assuming that each weight is represented by 4 bits.

A problem which haunts EANNs is the permutation problem [36, 65]. It is caused by the many to one mapping from the representation to the actual EANN since two EANNs which order their hidden nodes differently have different representations but

are functionally equivalent. This problem makes recombination operators difficult to produce highly fit children. For example, EANNs shown by Figure 3(a) and Figure 4(a) are equivalent, but they have different genotype representations as shown by Figure 3(b) and Figure 4(b). In general, any permutation of the hidden nodes will produce functionally equivalent EANNs but with different genotype representations.



0010 0000 0100 1010 0011 0111

(b)

Figure 4: (a) An EANN which is equivalent to that given in Figure 3(a); (b) Its binary representation under the same representation scheme.

The advantages of the binary representation lie in its simplicity and generality. It is very easy to apply standard crossover and mutation to binary strings. There is little need to design complex and tailored genetic operators. The binary representation also facilitates digital hardware implementation of EANNs. The disadvantage of the binary representation is its poor scalability. EANNs with several thousand connections are not uncommon nowadays but GAs operating on several thousand bits are inefficient.

There are several encoding methods, such as uniform, Gray, exponential, etc., that can be used in the binary representation. They encode real values with different ranges and precisions given the same number of bits. The issue of precision is especially important. If too few bits are used to represent each connection weight, the training might fail because some combinations of real connection weight are unable (or very

hard) to be approximated by discrete values. On the other hand, if too many bits are used, binary strings representing large EANNs will become extremely long and the evolution very inefficient.

2.2 Real Number Representation

There have been some debates on the cardinality of the genotype alphabet. Some researchers argued that the minimal cardinality, i.e., the binary representation, might not be the best [52, 66]. Formal analysis of nonstandard representations and operators based on the concept of equivalent classes [67] has given representations other than k-ary strings a more solid theoretical foundation. Real numbers were proposed to represent connection weights directly, i.e., one real number per connection weight [31, 33, 34, 52]. An EANN is represented by a set of real numbers, instead of binary strings. For example, a real number representation of the EANN given by Figure 3(a) could be `(4,10,2,0,7,3)`. A recombination operator can exchange real numbers between two sets but cannot change numbers themselves. These real numbers can only be changed by random mutations. Usually, a Gaussian random number is added to a real number.

As connection weights are represented by real numbers, standard genetic operators dealing with binary strings can no longer be applied directly. In such circumstances, an important task is to carefully design a set of genetic operators which are tailored to deal with real numbers in order to improve the speed and accuracy of the evolutionary training. Montana and Davis [31] defined a large number of tailored genetic operators which incorporated many heuristics about training EANNs. The major aim was to retain useful feature detectors formed around hidden nodes during evolution. Their results showed that the evolutionary training approach was much faster than the BP training algorithm at least for the problems they considered. Bartlett and Downs [34] also demonstrated that the evolutionary approach was faster and had better scalability than the BP training method.

Although tailored genetic operators are often used in conjunction with the real number representation of connection weights. This does not mean that a set of complex genetic operators is indispensable. Simpler genetic operators can also be used. For example, Fogel *et al.* [33] adopted only one genetic operator (excluding selection) — Gaussian random mutation — in the evolution of connection weights. A possible improvement over this method is to use Cauchy random mutation [68].

It is worth pointing out that the permutation problem and the destructive effect of crossover on feature detectors in EANNs still exist when representing connection weights in real numbers, but the problem concerning the representation precision of weights no longer exists.

2.3 Comparison Between Evolutionary Training and Gradient-Based Training

As indicated at the beginning of Section 2, the evolutionary training approach is attractive because it can handle the global search problem better in a vast, complex, multimodal and nondifferentiable space. It does not depend on the gradient information of the error (or fitness) function, thus is particularly appealing when the gradient information is unavailable or very costly to get. For example, the evolutionary approach has been used in reinforcement learning [47, 64, 69, 70, 71], recurrent network learning [45, 64, 72], and higher order network learning [56, 57]. Moreover, the same evolutionary algorithm can be used to training many different networks regardless of whether they are feedforward networks, recurrent networks, or higher order networks. The general applicability of the evolutionary approach saves a lot of human efforts in developing different training algorithms for different types of network.

The evolutionary training approach also makes it easier to generate EANNs with some special characteristics. An often used method to decrease EANN's complexity and improve its generalization ability is to include a penalty term in the fitness function. Unlike the case in gradient-based training, we do not have to worry about

whether this term is differentiable or even continuous. Other requirements, such as weight sharing and weight decaying, can also be incorporated into the fitness function easily.

Evolutionary training is generally rather slow in comparison with fast variants of the BP algorithm [73] and conjugate gradient algorithms [23, 74] when the gradient information is cheaply available but it can deal with a wide range of networks where such gradient information is unavailable. Even if the gradient information is available, evolutionary training can still be faster in some cases [34, 38, 44]. Prados [38] described a GA-based training algorithm which is "significantly faster than methods that use the generalized delta rule (GDR)." For the three tests reported in his paper [38], the GA-based training algorithm "took a total of about 3 hours and 20 minutes, and the GDR took a total of about 23 hours and 40 minutes." Bartlett and Downs [34] also gave a modified GA which was "an order of magnitude" faster than the BP algorithm for the 7-bit parity problem. The modified GA seemed to have better scalability than the BP algorithm as well since it was "around twice" as slow as the BP algorithm for the XOR problem.

Interestingly, quite different results were reported by Kitano [75]. He found that the GA-BP method, the method that runs a GA first and then the standard BP, "is, at best, equally efficient to faster variants of back propagation in very small scale networks, but far less efficient in larger networks." The test problems he used include the XOR problem, various size encoder/decoder problems, and the two-spiral problem.

The discrepancy between two seemingly contradictory results can be attributed at least partly to different GAs and BP algorithms compared. That is, whether the comparison is between a standard GA and a fast BP algorithm, or between a fast GA and a standard BP algorithm. It is well known that both GAs and BP algorithms are sensitive to the parameters used in them. BP algorithms are also very sensitive to their initial conditions.

2.4 Hybrid Training

One major problem of GAs is their inefficiency in fine-tuned local search although they are good at global search. The efficiency of evolutionary training can be improved significantly by incorporating a local search procedure into the evolution, i.e., combining GA's global search ability with local search's fine-tuning ability. GAs can be used to first locate a good region in the space and then a local search procedure is used to find a near optimal solution in this region. The local search algorithm could be the BP algorithm [36, 75] or other random search algorithms [34, 76].

Belew *et al.* [36] used GAs to search for a near optimal set of initial connection weights and then used the BP algorithm to perform local search from these initial weights. Their results showed that the hybrid GA/BP approach was more efficient than either the GA or BP algorithm used alone. If taken into account the fact that the BP algorithm has to run several times in practice in order to find good connection weights due to BP's sensitivity to initial conditions, the hybrid training algorithm is quite competitive in comparison with gradient-based training algorithms. Similar work on the evolution of initial weights has also been done on competitive learning neural networks [77].

It is interesting to consider finding good initial weights as locating a good region in the space. Defining that the basin of attraction of a local minimum is composed of all the points, sets of weights in this case, which can converge to the local minimum through a local search algorithm, then a global minimum can easily be found by the local search algorithm if the GA can locate any point, i.e., any set of initial weights, in the basin of attraction of the global minimum. Figure 5 illustrates a simple case where there is only one connection weight in the EANN. If a GA can find a initial weight like w_{I2}, it would be easy for a greedy local search algorithm to arrive at the globally optimal weight w_B despite of that w_{I2} itself is not so good as w_{I1}.

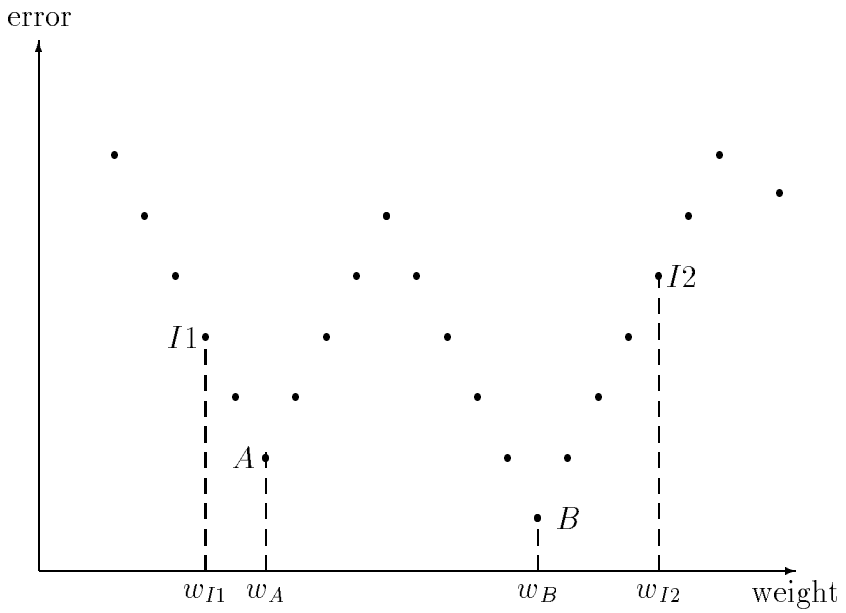


Figure 5: An illustration of using a GA to find good initial weights such that a greedy local search algorithm can find the globally optimal weights easily. w_{I2} in this figure is an optimal initial weight because it can lead to the global optimum w_B using a local search algorithm. A higher error value means a lower fitness value, i.e., a worse EANN.

3 The Evolution of Architectures

Section 2 assumes that the architecture of an EANN is pre-defined and fixed during the evolution of connection weights. This section concerns the design of EANN architectures. The architecture of an EANN includes its topological structure, i.e., connectivity, and the transfer function of each node in the EANN. As indicated at the beginning of this paper, architecture design is crucial in the successful application of EANNs because the architecture has significant impact on an EANN's information processing capabilities. Given a learning task, an EANN with only a few connections and linear nodes may not be able to perform the task at all due to its limited information processing capability while an EANN with a large number of connections and nonlinear nodes may be too powerful for the task to have good generalization ability.

Unfortunately, the architecture design is still very much a human expert's job. It depends heavily on the expert experience and a tedious trial-and-error process. There is no systematic way to design a near optimal architecture automatically for a given task. Research on constructive and destructive algorithms is an effort made towards the automatic design of architectures [14, 15, 16, 17, 18, 19, 20, 21]. Roughly speaking, a constructive algorithm starts with a minimal network (network with minimal number of hidden layers, nodes, and connections) and adds new layers, nodes and connections if necessary during training while a destructive algorithm does the opposite, i.e., starts with the maximal network and deletes unnecessary layers, nodes, and connections during training.

Design of the optimal architecture for an EANN can be formulated as a search problem in the architecture space where each point represents an architecture. Given some performance (optimality) criteria, e.g., fastest learning, lowest complexity, etc., about architectures, the performance level of all architectures forms a surface in the space. The optimal architecture design is equivalent to finding the highest point on this surface. There are several characteristics with such a surface as indicated by Miller *et al.* [78] which make GA-based evolutionary algorithms a better candidate for

searching the surface than those constructive and destructive algorithms mentioned above. These characteristics are [78]

- The surface is *infinitely large* since the number of possible nodes and connections is unbounded.
- The surface is *nondifferentiable* since changes in the number of nodes or connections are discrete and can have a discontinuous effect on EANN's performance.
- The surface is *complex* and *noisy* since the mapping from an architecture to its performance is indirect, strongly epistatic, and dependent on the evaluation method used.
- The surface is *deceptive* since similar architectures may have quite different performance.
- The surface is *multimodal* since different architectures may have similar performance.

Similar to the evolution of connection weights, two major stages involved in the evolution of architectures are the genotype representation scheme of architectures and the evolution itself. The key issue is to decide how much information about an architecture should be encoded into the genotype representation. At one extreme, all the detail, i.e., every connection and node of an architecture can be specified by the genotype representation, e.g., by some binary bits. This kind of representation schemes is called the *direct encoding scheme* or the *strong specification scheme*. At the other extreme, only the most important parameters of an architecture, such as the number of hidden layers and hidden nodes in each layer are encoded. Other detail about the architecture is left to the training process to decide. This kind of representation schemes is called the *indirect encoding scheme* or the *weak specification scheme*. After a representation scheme has been worked out, the evolution of architectures can progress according to the cycle shown in Figure 6.

1. Decode each individual in the current generation into an architecture. If the indirect encoding scheme is used, further detail of the architecture is specified by some developmental rules or a training process.

2. Train each EANN with the decoded architecture by a pre-defined learning rule (some parameters of the learning rule could be learned during training) starting from different sets of random initial connection weights and if any learning rule parameters.

3. Define the fitness of each individual (encoded architecture) according to the above training result and other performance criteria such as the complexity of the architecture.

4. Reproduce a number of children for each individual in the current generation based on its fitness.

5. Apply genetic operators to the children generated above and obtain the next generation.

Figure 6: A typical cycle of the evolution of architectures. (Reprinted with permission from Ref. 1.)

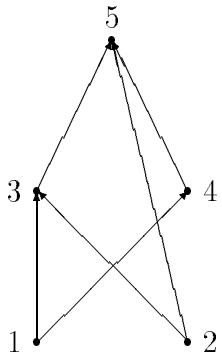
Because of advantages of the evolutionary design of architectures, a lot of research has been carried out in recent years [37, 46, 49, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]. However, almost all the research only deals with the topological structure of EANNs and little has been done on the evolution of node transfer functions let alone the evolution of both topological structures and node transfer functions. We will concentrate on the evolution of topological structures and analyze the genotype representation scheme of topological structures in Section 3.1 and 3.2. For convenience sake, the term architecture will be used interchangeably with the term topological structure in these two sections. Section 3.3 discusses the evolution of node transfer functions briefly.

3.1 The Direct Encoding Scheme

In the direct encoding scheme, each connection in an architecture is directly specified by its binary representation [82, 81, 78, 29, 98, 97, 93, 95]. For example, an $N \times N$ matrix $C = (c_{ij})_{N \times N}$ can represent an architecture with N nodes, where c_{ij} indicates presence or absence of the connection from node i to node j. We can use $c_{ij} = 1$ to indicate a connection and $c_{ij} = 0$ no connection. In fact, c_{ij} can even be connection weights from node i to node j so that both the topological structure and connection weights of an EANN are evolved at the same time [93, 99, 97, 98, 94, 46, 49, 41].

Each such matrix has a direct one-to-one mapping to the corresponding architecture. The binary string representing an architecture is just the concatenation of rows (or columns) of the matrix. Constraints on architectures being explored can easily be incorporated into such representation scheme by setting constraints on the matrix, e.g., a feedforward EANN will have non-zero entries only in the upper-right triangle of the matrix. Figure 7 and 8 give examples of the direct encoding scheme of architectures. It is obvious that such an encoding scheme can handle both feedforward and recurrent networks.

The direct encoding scheme is relatively simple and straightforward to implement.



(a)

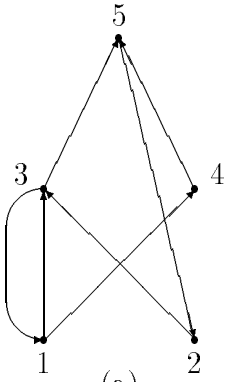
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)

0110 101 01 1

(c)

Figure 7: An example of the direct encoding of a feedforward EANN. (a), (b) and (c) show the architecture, its connectivity matrix and its binary string representation respectively. Because only feedforward architectures are under consideration, the binary string representation only needs to consider the upper-right triangle of the matrix.



(a)

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

(b)

00110 00100 10001 00001 01000

(c)

Figure 8: An example of the direct encoding of a recurrent EANN. (a), (b) and (c) show the architecture, its connectivity matrix and its binary string representation respectively.

It is suitable for the precise and deterministic handling of small architectures, i.e., architectures with a small number of nodes and connections. It may facilitate rapid generation and optimization of tightly pruned interesting designs that no one has hit upon so far [78]. It, however, does not scale well since large architectures require very large matrices to represent. One way to cut down the size of matrices is to use as much domain knowledge as possible to reduce the search space. For example, if there is only a complete connection between two neighboring layers and no other connections, a feedforward architecture can be encoded by the number of hidden layers and nodes in each hidden layer. The length of the genotype representation can be reduced greatly in this case [81, 82]. The evolution of architectures is an adaptive process where no prior knowledge about architectures is assumed but the availability of such knowledge can increase the efficiency of the evolution. It is very easy to incorporate such knowledge if available into the evolution.

Another flexibility provided by the evolution of architectures stems from the fitness definition. There is virtually no limitation such as being differentiable or continuous on how the fitness function should be defined at Step 3 in Figure 6. The training result pertaining to an architecture such as the error and the training time is often used in the fitness function. The complexity measurement such as the number of nodes and connections is also used in the fitness function. As a matter of fact, a lot of criteria based on the information theory or statistics [101, 102, 103] can readily be introduced into the fitness function without much difficulty. Improvement on EANN's generalization ability can be expected if these criteria are adopted. Schaffer *et al.* [81] presented an experiment which showed that an EANN designed by the evolutionary approach had better generalization ability than one trained by the BP algorithm using a human designed architecture.

One major problem of the evolutionary design of architectures is the permutation problem as illustrated by Figure 3 and 4 in Section 2.1. Because two functionally equivalent EANNs which order their hidden nodes differently have two different geno-

type representations, the probability of producing a highly fit child from them is very low. Some researchers hence abandoned crossover and adopted only mutation in the evolution of architectures [49] although it was shown that crossover was useful and important in increasing the efficiency of the evolution [104, 52, 65]. Hancock [65] suggested that the permutation problem might "not be as severe as had been supposed" with the population size and the selection mechanism he used because "The increased number of ways of solving the problem outweigh the difficulties of bringing building blocks together." It is worth indicating that most studies on the permutation problem concentrate on the genetic algorithm used, e.g., genetic operators, population sizes, selection mechanisms, etc. While it is necessary to investigate the algorithm, it is equally important to study the genotype representation scheme, since the performance surface defined at the beginning of Section 3 is decided by both the representation and genetic operators. More research is needed to further understand the impact of the permutation problem on the evolution of architectures.

3.2 The Indirect Encoding Scheme

In order to reduce the length of the genotype representation of architectures, the indirect encoding scheme has also been used by many researchers [80, 84, 87, 83, 88, 79, 105, 106, 96, 107] where only most important features of an architecture are encoded in the genotype representation. The detail about each individual connection are specified by developmental rules. The indirect encoding scheme is not only more compact but also biologically more plausible than the direct encoding scheme. According to the discoveries of neuroscience, it is impossible for genetic information encoded in chromosomes to specify independently the whole nervous system.

3.2.1 Parametric Representation

An architecture can be specified by many parameters such as the number of hidden layers, the number of hidden nodes in each layer, the number of connections between

two layers, etc. These parameters are often encoded in various forms in the genotype representation. Harp *et al.* [80, 84] used a "blueprint" to represent an architecture which consists of one or more segments representing an area (layer) and its efferent connectivity (projections). The first and last area are constrained to be the input and output area respectively. Each segment includes two parts of the information: (1) that about the area itself such as the number of nodes in the area and the spatial organization of the area; and (2) that about the efferent connectivity. It should be noted that only the connectivity pattern instead of each connection is specified here. The detailed node-to-node connection is specified by implicit developmental rules, e.g., the network instantiation software used by Harp *et al.* [80, 84]. Similar parametric representation methods with different sets of parameters have also been proposed by others [83, 87]. An interesting aspect of Harp *et al.*'s work is their combination of learning parameters with architectures in the genotype representation so that learning parameters can evolve along with architecture parameters. Thus a near optimal combination of learning parameters and architectures can be evolved.

Although the parametric representation method can reduce the length of binary strings specifying EANN's architectures, it still lacks the scalability needed by many real-world applications since the length grows quickly as the number of nodes in EANNs increases. Moreover, developmental rules play only a limited role here. They tend to be some fixed assumptions about architectures which are made according to our prior knowledge.

3.2.2 Developmental Rule Representation

A quite different indirect encoding method from the above is to encode developmental rules which are used to construct architectures in the genotype representation [79, 105, 107, 96]. The shift from the direct optimization of architectures to the optimization of developmental rules results in many advantages such as more compact representation, better scalability of the method and better modularity and generalization ability of the

resultant architecture. The destructive effect of crossover will also be lessened since the developmental rule representation is capable of preserving promising building blocks found so far [79].

A developmental rule is usually described by a recursive equation [105] or a generation rule similar to a production rule in a knowledge based system with a left-hand side and a right-hand side [79]. The connectivity pattern of the architecture in the form of a matrix is constructed from a basis, i.e., a single element matrix by repetitively applying suitable developmental rules to non-terminal elements in the current matrix until the matrix contains only terminal[2] elements which indicate the presence or absence of a connection that is until a connectivity pattern is fully specified.

Kitano [79] used a modified version of graph generation system [108] which includes a set of graph generation rules to construct connection matrices since each connection matrix corresponds to a directed graph. Each developmental rule he used, i.e., a graph generation rule consists of a left-hand side (LHS) which is a non-terminal element and a right-hand side (RHS) which is a 2×2 matrix with either terminal or non-terminal elements. A typical step of constructing a connection matrix is to find rules whose left-hand sides appear in the current matrix and replace all the appearance with respective right-hand sides. Each rule is represented by five *allele* positions corresponding to five elements in a rule in a genotype. The length of genotypes is variable with the first position fixed to an initial element, i.e., the basis. Each position in a genotype can take the value of an element in the range from "A" to "p". The 16 rules with "a" to "p" on the left-hand side and 2×2 matrices with only 1's and 0's on the right-hand side are pre-defined and do not participate in evolution. That is, not all developmental rules are obtained through evolution.

Given a set of developmental rules as shown in Figure 9, an architecture can

[2]In this paper, a terminal element is either 1 (existence of a connection) or 0 (non-existence of a connection) and a non-terminal element is a symbol other than 1 and 0. These definitions are slightly different from those used by Kitano [79].

be generated by applying the rules in three steps as described in Figure 10. The architecture generated is the same as that shown in Figure 8(a). The only difference is the numbering of nodes.

$$S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix} \quad C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix} \quad \cdots$$

$$a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \cdots$$

Figure 9: Examples of some developmental rules used to construct a connectivity matrix. S is the initial element (or state).

Consistently better results with various size encoder/decoder problems were reported by Kitano [79] in comparison with the direct encoding scheme. The developmental rule representation method however is not very good at fine-tuning detailed connections among single nodes because it concentrates on connections among groups of nodes. The addition of a fine-tuning process in which a dynamic node adjustment algorithm similar to constructive or destructive algorithms could be used after the evolution seems to be a good way to further improve the performance of the architecture.

Mjolsness *et al.* [105] described a similar rule encoding method where rules are represented by recursive equations which specify the growth of connection matrices. Coefficients of these recursive equations, represented by decomposition matrices, are encoded in genotypes and optimized by simulated annealing instead of GAs. Connection weights are optimized along with connectivity by simulated annealing since each entry of a connection matrix can have a real-valued weight, rather than just 1 or 0. One advantage of using simulated annealing instead of GAs in the evolution is

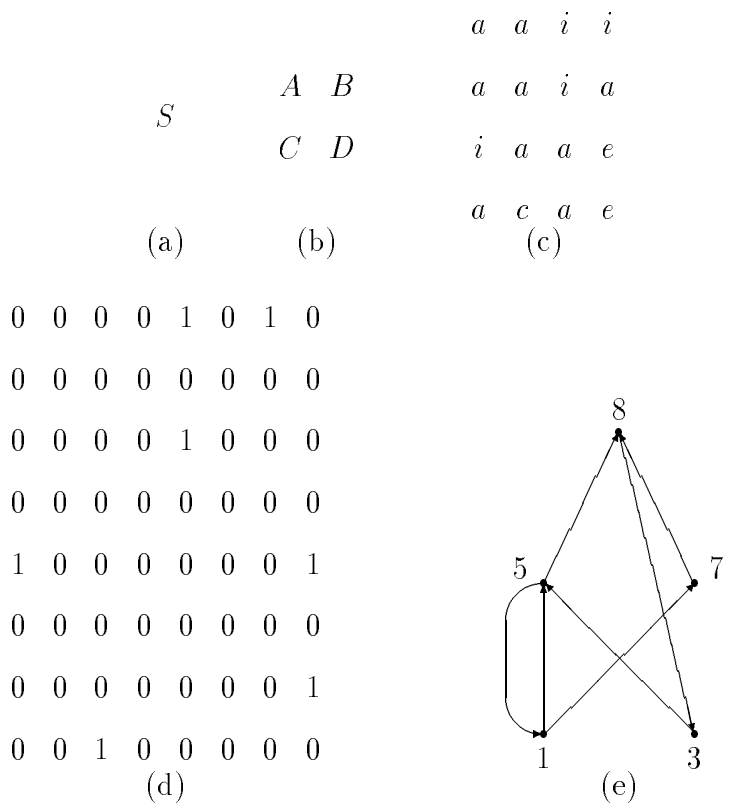


Figure 10: Development of an EANN architecture using the rules given in Figure 9. (a) The initial state; (b) Step 1; (c) Step 2; (d) Step 3 when all the entries in the matrix are terminal elements, i.e., either 1 or 0; (e) The architecture. The nodes in the architecture are numbered from 1 to 8. Isolated nodes are not shown.

the avoidance of the destructive effect of crossover. Wilson [109] also used simulated annealing in his work on EANN architecture design.

Recent research on the developmental rule representation [107, 96] has demonstrated its effectiveness and efficiency in the evolution of architectures. Gruau [107] reported a number of successful experiments with his "cellular encoding" method including experiments on the 50-input parity problem and the 40-input symmetry problem. Similar to Kitano's work [79], the cellular encoding method is also based on grammar rules but only EANNs which implement boolean functions are considered. The restriction that weights can only be $+1$ or -1 makes the evolution of both architectures and connection weights easier. It is unclear at present how well the method could perform when real-valued connection weights are used.

3.2.3 Fractal Representation

Merrill and Port [106] proposed another method for encoding architectures which is based on the use of fractal subsets of the plane. They argued that the fractal representation of architectures was biologically more plausible than the developmental rule representation. It however seems unlikely that their method could have better scalability than the developmental rule representation because they used three real-valued parameters, i.e., an edge code, an input coefficient and an output coefficient to specify *each* node in an architecture. Fast simulated annealing [110] was used in the evolution.

3.2.4 Other Representations

A very different approach to the evolution of architectures is recently proposed by Andersen [100]. His representation is unique in that each individual in a population represents a hidden node in an architecture instead of the architecture itself. An architecture is built layer by layer, i.e., hidden layers are added one by one if the current architecture cannot reduce the training error below certain threshold. Each hidden

layer is constructed automatically through an evolutionary process which employs the GA with sharing. The sharing should help the evolution to form different feature detectors in the population so that each hidden node could be used as a different feature detector.

One problem with Andersen's method is that there are usually several hidden nodes which have very similar functionality, i.e., which are basically the same feature detector in a population. Such redundancy has to be removed by an additional clean-up algorithm. Another restriction on Andersen's method is its layer by layer approach which can only deal with multilayer perceptrons.

3.3 The Evolution of Node Transfer Functions

All the discussion on the evolution of architectures so far only covers the topological structure of an architecture. The transfer function of each node in the architecture has been assumed as fixed and pre-defined by human experts although the transfer function has been shown to be an important part of an architecture and have significant impact on EANN's performance [111, 112, 113]. The transfer function is often assumed to be the same for all the nodes in an architecture or at least for all the nodes in the same layer.

Stork *et al.* [114] are to our best knowledge the first to apply GAs to the evolution of both topological structures and node transfer functions even though neural networks consisting of only 7 nodes were investigated. The transfer function was specified in the structural genes in their genotype representation. It was much more complex than the usual sigmoid function as they tried to model a biological neuron in the tailflip circuitry of the crayfish.

White and Ligomenides [99] adopted a simpler approach to the evolution of both topological structures and node transfer functions. For each individual (i.e., EANN) in the initial population, 80% nodes in the EANN used the sigmoid transfer function and 20% nodes used the Gaussian transfer function. The evolution was used to decide

the optimal mixture between these two transfer functions automatically. The sigmoid and Gaussian transfer function themselves were not evolvable. No parameters of the two function were evolved.

3.4 The Evolution of Both Architectures and Connection Weights

One problem of the evolution of architectures which have not been touched is the noisy evaluation of genotypes. The evaluation of a genotype is noisy because it is the actual architecture, i.e., the phenotype created from a genotype through developmental rules that is used to evaluate the genotype's fitness. In practice, a popular way to evaluate genotypes is to transform a genotype (an encoded architecture) into a phenotype (a fully specified architecture) first through developmental rules and then train the phenotype from different initial connection weights generated at random. The training result is used as part of the fitness function to evaluate the genotype. Due to the stochastic nature of random initial weights, the fitness of a genotype evaluated in this way is very noisy. The evaluation could be much noisier if the development rules are not deterministic either in an indirect encoding scheme.

One way to alleviate this problem is to employ the evolution of both architectures and connection weights at the same time [105, 93, 99, 97, 98, 94, 46, 49, 41, 100, 107] although the noise introduced by stochastic development rules is hard to removed. Such co-evolution is a natural choice if EANNs with only binary threshold nodes are considered [93, 107, 100]. In general, combination of two levels of evolution into one will increase the search space. Suppose the size of the architecture space is $|S_A|$ and the size of the connection weight space is $|S_W|$, then the total size of the two search spaces at two levels is $|S_W| + |S_A|$ while the total size of the combined one level search space is $|S_W| \times |S_A|$. The trade-off between efficiency and accuracy has to be made in practice.

3.5 The Evolution of Input Features

The evolution of input features is used to reduce and/or combine all possible inputs to an EANN into a compact and effective set. It can be thought as an evolutionary approach to dimension reduction. In most cases, the number of inputs and outputs of an EANN are decided by the problem at hand. For example, a hand-printed digit recognition network may naturally have 64×64 inputs and 10 outputs. Unfortunately, the number of inputs to an EANN is usually very large for real world problems. There may also be a lot of redundancy among different inputs. A large number of inputs to an EANN increase the size of the EANN and thus require more training data during training in order to achieve a reasonable generalization ability for the EANN. Preprocessing is often needed to reduce the number of inputs to an EANN. Various dimension reduction techniques including the principal component analysis have been used for this purpose.

The problem of reducing the number of inputs to an EANN can also be formulated as a search problem. Given a set of inputs, we want to find an optimal subset of it which has the fewest number of elements and the performance of the EANN using this subset is no worse than that of the EANN using the whole set. The evolutionary approach has been identified as a good way to search this vast space [115, 116, 117, 118, 119]. Very good results, i.e., better performance with fewer inputs have been reported from these studies. In the evolution of input features, each individual in the current population represents a subset of all the inputs. The evaluation of each individual is carried out by training an EANN with these inputs and using the result as part of the fitness function. Such evaluation is very noisy however due to the reason explained in Section 3.4.

Not only does the evolution of input features provide a way to discover important features from all the inputs adaptively, it can also be used to discover new training examples. Zhang and Veenker [120] described an active learning paradigm where a training algorithm based on GAs can select training examples for itself.

4 The Evolution of Learning Rules

A training algorithm may have different performance when applied to different architectures. The design of training algorithms, more fundamentally the learning rules used to adjust connection weights, depends on the type of architectures under investigation. Different variants of the Hebbian learning rule have been proposed to deal with different architectures. However, the design becomes very hard when there is little prior knowledge about EANN's architecture which is often the case in practice. It is desirable to develop an automatic and systematic way to adapt the learning rule to an architecture and the tasks at hand. Designing a learning rule by human experts often implies some assumptions which are not necessarily true about EANN's architectures and tasks to be performed by the EANN. For example, the widely accepted Hebbian learning rule has recently been shown to be outperformed by a new rule proposed by Artola *et al.* [121] in many cases [122]. The new rule can learn more patterns than the optimal Hebbian rule and can learn exceptions as well as regularities. It is however still hard to say that this rule is optimal for all other architectures. In fact, what is needed from an EANN is its ability to adjust its learning rule adaptively according to its architecture and the tasks to be performed. In other words, an EANN should learn its learning rule rather than have it designed by human experts. Since evolution is one of the most fundamental forms of adaptation, it is not surprising that the evolution of learning rules has been introduced into EANNs in order to learn their learning rules.

The relationship between evolution and learning is extremely complex. Various models have been proposed [123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136] but most of them deal with the issue of how learning can guide evolution [123, 124, 125, 126] and the relationship between the evolution of architectures and that of connection weights [127, 128, 129]. Research into the evolution of learning rules is still in its early stages [130, 131, 132, 133, 135, 136]. This research is important not only in providing an automatic way of optimizing learning rules and in modeling the

1. Decode each individual in the current generation into a learning rule.
2. Construct a set of EANNs with randomly generated architectures (or a pre-defined architecture in some cases) and initial connection weights and train them using the decoded learning rule.
3. Calculate the fitness of each individual (encoded learning rule) according to the above training result and other criteria if any.
4. Reproduce a number of children for each individual in the current generation according to its fitness.
5. Apply genetic operators to child individuals produced above and obtain the new generation.

Figure 11: A typical cycle of the evolution of learning rules. (Reprinted with permission from Ref. 1.)

relationship between learning and evolution but also in modeling the creative process since newly evolved learning rules can deal with a complex and dynamic environment. The research will help us better understand how creativity can emerge in artificial systems, like EANNs and how to model the creative process in biological systems. A typical cycle of the evolution of learning rules can be described by Figure 11.

Similar to the reason indicated in Section 3.4, the fitness evaluation of each individual, i.e., the encoded learning rule is noisy although some techniques could be used to alleviate this problem, e.g., a weighted average of the training results on EANNs with different initial connection weights could be used in the fitness function. If an architecture cannot be pre-defined, each individual in the evolution has to be evaluated by different architectures as well. Additional noise will be introduced into the evaluation in this case.

4.1 The Evolution of Algorithmic Parameters

The adaptive adjustment of BP algorithm's parameters such as the learning rate and momentum through evolution could be considered as the first attempt of the evolution of learning rules [80, 36]. Harp *et al.* [80] encoded BP algorithm's parameters in genotypes together with EANN's architecture. This evolutionary approach is different from the non-evolutionary one such as Jocobs's work [137] because the evolution of both the algorithmic parameters and architectures facilitates further exploration of interactions between the learning algorithm and architectures such that a near optimal combination of a BP algorithm with an architecture can be evolved. The cost of this benefit as mentioned in Section 3.4 is a larger search space and thus longer computation time.

Other researchers [36, 77] also used an evolutionary process to find parameters for the BP algorithm but EANN's architecture was pre-defined. The parameters evolved in this case tend to be optimized towards the architecture rather than general applicable ones. There are a number of BP algorithms with an adaptive learning rate and momentum where a non-evolutionary approach is used. Further comparison between the two approaches would be quite useful.

4.2 The Evolution of Learning Rules

The evolution of algorithmic parameters is certainly interesting but it hardly touches the fundamental part of a training algorithm, i.e., its learning rule or weight adjusting rule. Adapting a learning rule through evolution is expected to enhance EANN's adaptivity greatly in a dynamic environment.

Unlike the evolution of connection weights and architectures which only deal with static objects in an EANN, i.e., weights and architectures, the evolution of learning rules has to work on the dynamic behavior of an EANN. The biggest problem here is how to encode the dynamic behavior of a learning rule into static genotypes. Trying

to develop a universal representation scheme which can specify any kind of dynamic behaviors is clearly impractical let alone the prohibitive long computation time required to search such a learning rule space. Constraints have to be set on the type of dynamic behaviors, i.e., the basic form of learning rules being evolved in order to reduce the representation complexity and the search space.

Two basic assumptions which have often been made on learning rules are: (1) The weight-updating of a connection depends only on local information such as the activation of the input node, the activation of the output node, the current connection weight, etc.; and (2) The learning rule is the same for all connections in an EANN. A learning rule is assumed to be a linear function of these local variables and their products. That is, a learning rule can be described by the function

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1,i_2,\cdots,i_k=1}^{n} \left(\theta_{i_1 i_2 \cdots i_k} \prod_{j=1}^{k} x_{i_j}(t-1) \right) \tag{2}$$

where t is time, Δw is the weight change, $x_1, x_2, \cdots, x_n$ are local variables and θ's are real-valued coefficients which are decided by the evolution of learning rules. Due to a large number of terms in Eq.2 which make the evolution extremely slow and impractical, further constraints are often set based on either biological or other heuristic knowledge in order to reduce the number.

There are basically three issues in the evolution of learning rules; determination of a subset of terms described in Eq.2, representation of their coefficients as genotypes and the evolution simulated by an evolutionary algorithm. Chalmers [130] defined the form of learning rules as a linear combination of four local variables and their six pairwise products. No third or fourth order[3] terms were used. Ten coefficients and a scale parameter were encoded in a binary string via exponential encoding. The architecture used in the fitness evaluation was fixed because only single layer EANNs were considered and the number of inputs and outputs were fixed by the learning task at hand although the architecture could be generated at random and even evolved

[3]The order is defined as the number of variables in a product.

along with learning rules. After 1000 generations, starting from a population of randomly generated learning rules, the evolution discovered the well-known delta rule [138, 2] and some of its variants. These experiments although simple and preliminary, demonstrated the potentiality of the evolution of learning rules in discovering novel learning rules not merely known ones. However, constraints set on learning rules could prevent some learning rules from being evolved such as those which include third or fourth order terms.

Similar experiments on the evolution of learning rules were also carried out by others [133, 131, 132, 135, 136]. Fontanari and Meir [133] used Chalmers' approach to evolve learning rules for binary perceptrons. They also considered four local variables but only 7 terms were adopted in their weight-updating function which included one first order, three second order and three third order terms in Eq.2. Baxter [135] took one step further than just the evolution of learning rules. He tried to evolve complete EANNs in a single level of evolution. It is clear that the search space of possible EANNs would be enormous if strict constraints were not set on the connection weights, architectures and learning rules. In his experiments, only EANNs with binary threshold nodes were considered, so the weights could only be $+1$ or -1. The number of nodes in EANNs was fixed. The learning rule concerned with only two boolean variables. Although Baxter's experiments were rather simple, they confirmed that complex behaviors could be learned and EANN's learning ability could be improved through evolution.

Bengio *et al.*'s approach [131, 132] is slightly different from Chalmers' in the sense of that gradient descent algorithms and simulated annealing were also employed to simulate evolution. Four local variables and one zeroth order, three first order and three second order terms in Eq.2 were used in their learning rules.

Research related to the evolution of learning rules includes Parisi *et al.*'s work on econets although they did not evolve learning rules explicitly [126, 139]. They put their emphasis on the crucial role of the environment in which the evolution

happened while only using some simple neural networks in their study. The issue of environmental diversity is closely related to the noisy evaluation of genotypes as indicated in Section 3.4 and at the beginning of Section 4. There are two possible sources of noise. The first is the decoding process (morphogenesis) of genotypes. The second is introduced when a decoded learning rule is evaluated by using it to train EANNs. The environmental diversity is essential in obtaining a good approximation to the fitness of the decoded learning rule and thus in reducing the noise from the second source. If a general learning rule which is applicable to a wide range of architectures and learning tasks is needed, the environmental diversity has to be very high, i.e., many different architectures and learning tasks have to be used in the fitness evaluation of each learning rule.

5 Concluding Remarks

Although evolution has been introduced into EANNs at various levels, they can roughly be divided into three; the evolution of connection weights, architectures and learning rules. This section first describes a general framework for EANNs and then gives some conclusions.

5.1 A General Framework for EANNs

A general framework for EANNs is described by Figure 12 [140]. The evolution of connection weights proceeds at the lowest level on the fastest time scale in an environment determined by an architecture, a learning rule and learning tasks. There are however two alternatives to decide the level of the evolution of architectures and that of learning rules; either the evolution of architectures is at the highest level and that of learning rules at the lower one or vice versa. The lower the level of evolution is, the faster the time scale it is on.

From the point of view of engineering, the decision on the level of evolution de-

pends on what kind of prior knowledge is available. If there is more prior knowledge about EANN's architectures than that about their learning rules or a particular class of architectures is pursued, it is better to put the evolution of architectures at the highest level because such knowledge can be encoded in architecture's genotype representation to reduce the (architecture) search space and the lower level evolution of learning rules can be more biased towards this kind of architectures. On the other hand, the evolution of learning rules should be at the highest level if there is more prior knowledge about them available or there is a special interest in certain type of learning rules. Unfortunately, there is usually little prior knowledge available about both architectures and learning rules in practice except for some very vague statements [141]. In this case, it is more appropriate to put the evolution of architectures at the highest level since the optimality of a learning rule makes more sense when evaluated in an environment including the architecture to which the learning rule is applied.

Figure 12 can also be viewed as a general framework of adaptive systems if we do not restrict ourselves to GAs and three levels. Simulated annealing, gradient descent search and even exhaustive search can be considered as special cases of evolutionary algorithms. For example, the traditional BP network can be considered as a special case of our general framework with one-shot (only-one-candidate) search used in the evolution of architectures and learning rules and the BP algorithm used in the evolution of connection weights. In fact, the general framework provides a basis for comparing various specific EANN models according to search procedures they used at three different levels since it defines a three-dimensional space where 0 represents one-shot search and +1 represents exhaustive search along each axis. Each EANN model corresponds to a point in this space.

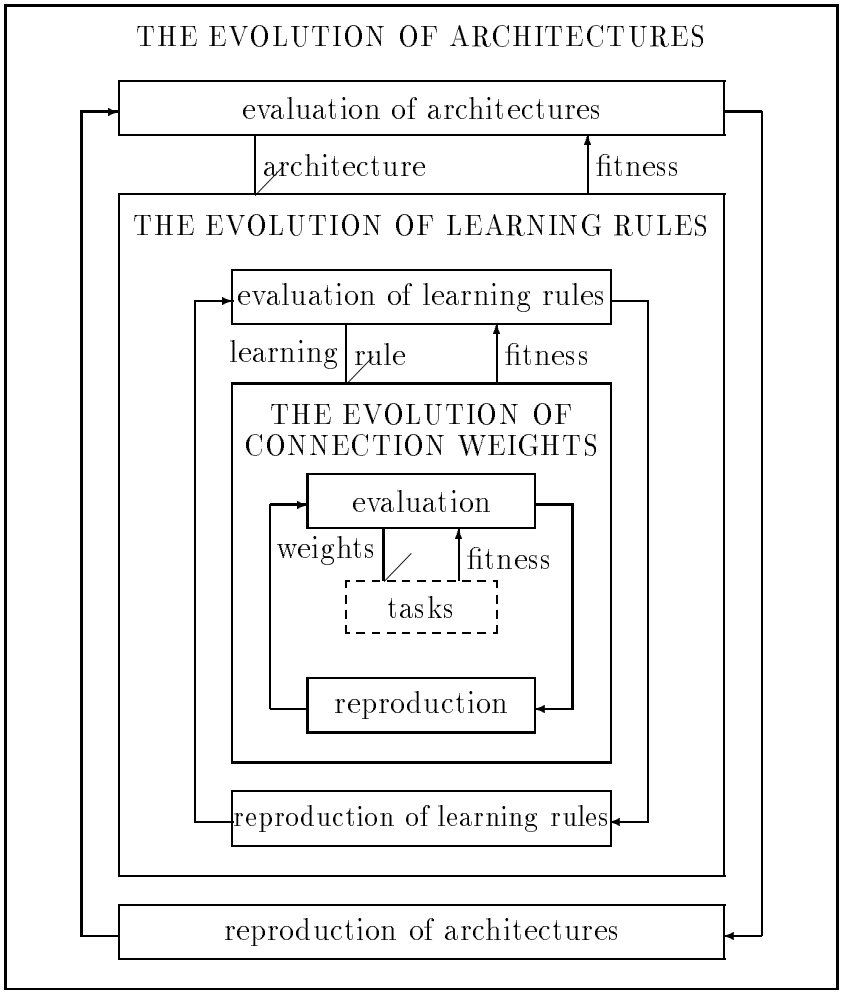


Figure 12: A general framework for EANNs. (Reprinted with permission from Ref. 1.)

5.2 Conclusions

Evolution can be introduced into EANNs at three levels. The evolution of connection weights provides an adaptive approach to connection weight training especially in reinforcement learning and recurrent network learning where gradient-based training algorithms experience great difficulties. Due to the simplicity and generality of the evolution and the fact that gradient-based training algorithms often have to run multiple times in order to avoid being trapped in a bad local optimum, the evolutionary approach is quite competitive in comparison with other approaches.

The evolution of architectures is used to generate a near optimal architecture automatically for the tasks at hand. It has several advantages over knowledge-based heuristic methods of architecture design because of the characteristics of the design problem given at the beginning of Section 3. The indirect encoding scheme for architectures has been shown to have good scalability. It can facilitate generation of EANNs with good generalization ability by incorporating various factors into the fitness function. The direct encoding scheme is good at fine tuning an architecture but has poor scalability.

The evolution of learning rules allows an EANN to adapt its learning rule to its environment so that an efficient learning rule can be obtained. In a sense, the evolution provides EANNs with the ability of learning to learn. It also helps to model the relationship between learning and evolution. Preliminary experiments have shown that efficient learning rules can be evolved from randomly generated rules. Current research on the evolution of learning rules normally assumes that learning rules can be specified by Eq.2. While constraints on learning rules are necessary to reduce the search space in the evolution, they might prevent some interesting learning rules from being discovered. The evolution of learning rules is closely related to the artificial life research where the evolution of complex behaviors is studied and there is no explicit representation of a learning rule.

Global search procedures such as GAs are usually computationally expensive to

run. It would be better not to use GAs at all levels of evolution in practice. It is however beneficial to introduce global search at some levels of evolution especially when there is little prior knowledge available at that level and the performance of the EANN is required to be high because the trial-and-error and other heuristic methods are very inefficient in such circumstances. There have already been some experiments which demonstrate the advantages of using hybrid global and local search at different levels [36] but the issue of an optimal combination of different search procedures needs further investigation.

With the increasing power of parallel computers, the simulation of large EANNs becomes feasible. Such simulations can not only discover possible new EANN architectures and learning rules but also offer a way to model the creative process as a result of EANN's adaptation to a dynamic environment.

5.3 Suggested Reading

1. X. Yao, "A review of evolutionary artificial neural networks," *International Journal of Intelligent Systems*, **8**:539–567, 1993.

2. D. Whitley and J. D. Schaffer (ed.), *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92),* IEEE Computer Society Press, Los Alamitos, CA, 1992.

3. D. Whitley, S. Dominic, R. Das and C. W. Anderson, "Genetic Reinforcement Learning for Neurocontrol Problems," *Machine Learning*, **13**:259–284, 1993.

References

[1] X. Yao.

Evolutionary artificial neural networks.

International Journal of Neural Systems, 4(3):203–222, 1993.

[2] J. Hertz, A. Krogh, and R. Palmer.

Introduction to the Theory of Neural Computation.

Addison-Wesley, Reading, MA, 1991.

[3] D. R. Hush and B. G. Horne.

Progress in supervised neural networks.

IEEE Signal Processing Magazine, 10(1):8–39, January 1993.

[4] J. H. Holland.

Adaptation in Natural and Artificial Systems (1st MIT Press Edn).

The MIT Press, Cambridge, MA, 1992.

[5] D. E. Goldberg.

Genetic Algorithms in Search, Optimization, and Machine Learning.

Addison-Wesley, Reading, MA, 1989.

[6] L. J. Fogel, A. J. Owens, and M. J. Walsh.

Artificial Intelligence Through Simulated Evolution.

John Wiley & Sons, New York, NY, 1966.

[7] D. B. Fogel.

System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling.

Ginn Press, Needham Heights, MA 02194, 1991.

[8] H.-P. Schwefel.

Numerical Optimization of Computer Models.

John Wiley & Sons, Chichester, 1981.

[9] T. Bäck, F. Hoffmeister, and H.-P. Schwefel.

A survey of evolution strategies.

In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Int'l Conf. on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, San Mateo, CA, 1991.

[10] T. Bäck and H.-P. Schwefel.

An overview of evolutionary algorithms for parameter optimization.

Evolutionary Computation, 1(1):1–23, 1993.

[11] R. C. Eberhart.

The role of genetic algorithms in neural network query-based learning and explanation facilities.

In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 169–183. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[12] T. P. Caudell.

Genetic algorithms as a tool for the analysis of adaptive resonance theory network training sets.

In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 184–200. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[13] R. C. Eberhart and R. W. Dobbins.

Designing neural network explanation facilities using genetic algorithms.

In *Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Singapore)*, volume 2, pages 1758–1763. IEEE Press, New York, NY, 1991.

[14] S. E. Fahlman and C. Lebiere.

The cascade-correlation learning architecture.

In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

[15] M. Frean.

The upstart algorithm: a method for constructing and training feedforward neural networks.

Neural Computation, 2:198–209, 1990.

[16] M. C. Mozer and P. Smolensky.

Skeletonization: a technique for trimming the fat from a network via relevance assessment.

Connection Science, 1:3–26, 1989.

[17] J. Sietsma and R. J. F. Dow.

Creating artificial neural networks that generalize.

Neural Networks, 4:67–79, 1991.

[18] Y. Hirose, K. Yamashita, and S. Hijiya.

Back-propagation algorithm which varies the number of hidden units.

Neural Networks, 4:61–66, 1991.

[19] Y. LeCun, J. S. Denker, and S. A. Solla.

Optimal brain damage.

In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann, San Mateo, CA, 1990.

[20] A. Roy, L. S. Kim, and S. Mukhopadhyay.

A polynomial time algorithm for the construction and training of a class of multilayer perceptrons.

Neural Networks, 6:535–545, 1993.

[21] J.-N. Hwang, S.-S. You, S.-R. Lay, and I-C. Jou.

What's wrong with a cascaded correlation learning network: a projection pursuit learning perspective.

Technical report, Department of Electrical Engineering, FT-10, University of Washington, Seattle, WA 98195, 1993.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams.

Learning internal representations by error propagation.

In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I*, pages 318–362.

MIT Press, Cambridge, MA, 1986.

[23] M. F. Møller.

A scaled conjugate gradient algorithm for fast supervised learning.

Neural Networks, 6:525–533, 1993.

[24] K. J. Lang, A. H. Waibel, and G. E. Hinton.

A time-delay neural network architecture for isolated word recognition.

Neural Networks, 3:33–43, 1990.

[25] S. S. Fels and G. E. Hinton.

Glove-talk: a neural network interface between a data-glove and a speech synthesizer.

IEEE Trans. on Neural Networks, 4:2–8, 1993.

[26] S. Knerr, L. Personnaz, and G. Dreyfus.

Handwritten digit recognition by neural networks with single-layer training.

IEEE Trans. on Neural Networks, 3:962–968, 1992.

[27] J. D. Schaffer, D. Whitley, and L. J. Eshelman.

Combinations of genetic algorithms and neural networks: a survey of the state of the art.

In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 1–37. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[28] R. S. Sutton.

Two problems with backpropagation and other steepest-descent learning procedures for networks.

In *Proc. of 8th Annual Conf. of the Cognitive Science Society*, pages 823–831. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[29] D. Whitley, T. Starkweather, and C. Bogart.

Genetic algorithms and neural networks: optimizing connections and connectivity.

Parallel Computing, 14:347–361, 1990.

[30] D. Whitley.

The GENITOR algorithm and selective pressure: why rank-based allocation of reproductive trials is best.

In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 116–121. Morgan Kaufmann, San Mateo, CA, 1989.

[31] D. Montana and L. Davis.

Training feedforward neural networks using genetic algorithms.

In *Proc. of Eleventh Int'l Joint Conf. on Artificial Intelligence*, pages 762–767. Morgan Kaufmann, San Mateo, CA, 1989.

[32] T. P. Caudell and C. P. Dolan.

Parametric connectivity: training of constrained networks using genetic algorithms.

In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 370–374. Morgan Kaufmann, San Mateo, CA, 1989.

[33] D. B. Fogel, L. J. Fogel, and V. W. Porto.

Evolving neural networks.

Biological Cybernetics, 63:487–493, 1990.

[34] P. Bartlett and T. Downs.

Training a neural network with a genetic algorithm.

Technical Report, Dept. of Elec. Eng., Univ. of Queensland, January 1990.

[35] J. Heistermann and H. Eckardt.

Parallel algorithms for learning in neural networks with evolution strategy.

In D. J. Evans, G. R. Joubert, and F. J. Peters, editors, *Proc. of Parallel Computing 89*, pages 275–280. Elsevier Science Publishers B.V., Amsterdam, 1989.

[36] R. K. Belew, J. McInerney, and N. N. Schraudolph.
Evolving networks: using genetic algorithm with connectionist learning.
Technical Report #CS90-174 (Revised), Computer Science & Engr. Dept. (C-014), Univ. of California at San Diego, La Jolla, CA 92093, USA, February 1991.

[37] N. J. Radcliffe.
Genetic Neural Networks on MIMD Computers (compressed edition).
PhD thesis, Dept. of Theoretical Phys., University of Edinburgh, Scotland, UK, 1990.

[38] D. L. Prados.
Training multilayered neural networks by replacing the least fit hidden neurons.
In *Proc. of IEEE SOUTHEASTCON '92*, volume 2, pages 634–637. IEEE Press, New York, NY, 1992.

[39] H. de Garis.
Steerable genNets: the genetic programming of steerable behaviors in genNets.
In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proc. of the First European Conference on Artificial Life*, pages 272–281. MIT Press, Cambridge, MA, USA, 1991.

[40] H. de Garis.
Using the genetic algorithm to train time dependent behaviors in neural networks.
In R. S. Michalski and G. Tecuci, editors, *Proc. of the First International Workshop on Multistrategy Learning (MSL-91)*, pages 273–280. Center for Artificial Intelligence, Fairfax, VA, USA, 1991.

[41] M. Srinivas and L. M. Patnaik.

Learning neural network weights using genetic algorithms — improving performance by search-space reduction.

In *Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Singapore)*, volume 3, pages 2331–2336. IEEE Press, New York, NY, 1991.

[42] H. de Garis.

GenNets: genetically programmed neural nets — using the genetic algorithm to train neural nets whose inputs and/or outputs vary in time.

In *Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Singapore)*, volume 2, pages 1391–1396. IEEE Press, New York, NY, 1991.

[43] A. Homaifar and S. Guan.

Training weights of neural networks by genetic algorithms and messy genetic algorithms.

In M. H. Hamza, editor, *Proc. of the Second IASTED International Symposium on Expert Systems and Neural Networks*, pages 74–77. Acta Press, Anaheim, CA, USA, 1990.

[44] D. L. Prados.

New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques.

Electronics Letters, 28(16):1560–1561, July 1992.

[45] A. P. Wieland.

Evolving neural network controllers for unstable systems.

In *Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Seattle)*, volume 2, pages 667–673. IEEE Press, New York, NY, 1991.

[46] J. R. Koza and J. P. Rice.

Genetic generation of both the weights and architecture for a neural network.

In *Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Seattle)*, volume 2, pages 397–404. IEEE Press, New York, NY, 1991.

[47] S. Dominic, R. Das, D. Whitley, and C. Anderson.

Genetic reinforcement learning for neural networks.

In *Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Seattle)*, volume 2, pages 71–76. IEEE Press, New York, NY, 1991.

[48] F. A. Dill and B. C. Deer.

An exploration of genetic algorithms for the selection of connection weights in dynamical neural networks.

In *Proc. of IEEE 1991 National Aerospace and Electronics Conference NAECON 1991*, volume 3, pages 1111–1115. IEEE Press, New York, NY, 1991.

[49] S. Bornholdt and D. Graudenz.

General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks*, 5:327–334, 1992.

[50] B. Maricic.

Genetically programmed neural network for solving pole-balancing problem.

In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks: Proc. of Int'l Conf. on Artificial Neural Networks — ICANN-91, Vol. 2*, pages 1273–1276. North-Holland, Amsterdam, 1991.

[51] J. J. Spofford and K. J. Hintz.

Evolving sequential machines in amorphous neural networks.

In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks: Proc. of Int'l Conf. on Artificial Neural Networks — ICANN-*

91, Vol. 1, pages 973–978. North-Holland, Amsterdam, 1991.

[52] F. Menczer and D. Parisi.

Evidence of hyperplanes in the genetic learning of neural networks.

Biological Cybernetics, 66:283–289, 1992.

[53] Y. Ichikawa and T. Sawa.

Neural network application for direct feedback controllers.

IEEE Transactions on Neural Networks, 3:224–231, 1992.

[54] A. W. O'Neil.

Genetic based training of two-layer, optoelectronic neural network.

Electronics Letters, 28(1):47–48, Jan. 1992.

[55] J. Heistermann.

A mixed genetic approach to the optimization of neural controllers.

In P. Dewilde and J. Vandewallele, editors, *Proc. of CompEuro 92*, pages 459–464. IEEE Computer Soc. Press, Los Alamitos, CA, 1992.

[56] D. J. Janson and J. F. Frenzel.

Application of genetic algorithms to the training of higher order neural networks.

Journal of Systems Engineering, 2:272–276, 1992.

[57] D. J. Janson and J. F. Frenzel.

Training product unit neural networks with genetic algorithms.

IEEE Expert, 8(5):26–33, 1993.

[58] A. V. Scherf and L. D. Voelz.

Training neural networks with genetic algorithms for target detection.

Proceedings of the SPIE (Conf. on Science of Artificial Neural Networks, Orlando, FL, USA), 1710, pt.1:734–741, 1992.

[59] E. DeRouin and J. Brown.

Alternative learning methods for training neural network classifiers.

Proceedings of the SPIE (Conf. on Science of Artificial Neural Networks, Orlando, FL, USA), 1710, pt.1:474–483, 1992.

[60] M. A. Lewis, A. H. Fagg, and A. Solidum.
Genetic programming approach to the construction of a neural network for control of a walking robot.
In *Proc. of 1992 IEEE International Conference on Robotics and Automation*, volume 3, pages 2618–2623. IEEE Computer Soc. Press, Los Alamitos, CA, 1992.

[61] H. B. Penfold, O. F. Diessel, and M. W. Bentink.
A genetic breeding algorithm which exhibits self-organizing in neural networks.
In M. H. Hamza, editor, *Proc. of the IASTED International Symposium Artificial Intelligence Application and Neural Networks - AINN '90*, pages 293–296. ACTA Press, Anaheim, CA, USA, 1990.

[62] M. L. Gargano, R. A. Marose, and L. von Kleeck.
An application of artificial neural networks and genetic algorithms to personnel selection in the financial industry.
In *Proc. of The First International Conference on Artificial Intelligence on Wall Street*, pages 257–262. IEEE Computer Soc. Press, Los Alamitos, CA, 1991.

[63] J. G. Elias.
Genetic generation of connection patterns for a dynamic artificial neural network.
In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 38–54. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[64] R. D. Beer and J. C. Gallagher.
Evolving dynamical neural networks for adaptive behavior.
Adaptive Behavior, 1:91–122, 1992.

[65] P. J. B. Hancock.

Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification.

In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 108–122. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[66] J. Antonisse.

A new interpretation of schema notation that overturns the binary encoding constraint.

In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 86–91. Morgan Kaufmann, San Mateo, CA, 1989.

[67] N. J. Radcliffe.

Equivalence class analysis of genetic algorithms.

Complex Systems, 5:183–205, 1991.

[68] X. Yao.

Simulated annealing with extended neighbourhood.

Int. J. of Computer Math., 40:169–189, 1991.

[69] D. Whitley, S. Dominic, and R. Das.

Genetic reinforcement learning with multilayer neural networks.

In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Int'l Conf. on Genetic Algorithms*, pages 562–570. Morgan Kaufmann, San Mateo, CA, 1991.

[70] D. H. Ackley and M. S. Littman.

Learning from natural selection in an artificial environment.

In *Proc. of Int'l Joint Conf. on Neural Networks, Vol. I*, pages 189–193, Washington, DC, 1990. Lawrence Erlbaum Associates, Hillsdale, NJ.

[71] D. Whitley, S. Dominic, R. Das, and C. W. Anderson.
Genetic reinforcement learning for neurocontrol problems.
Machine Learning, 13(2/3):259–284, 1993.

[72] J. Torreele.
Temporal processing with recurrent networks: an evolutionary approach.
In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Int'l Conf. on Genetic Algorithms*, pages 555–561. Morgan Kaufmann, San Mateo, CA, 1991.

[73] S. E. Fahlman.
Faster-learning variations on back-propagation: an empirical study.
In D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 38–51. Morgan Kaufmann, San Mateo, CA, 1988.

[74] E. M. Johansson, F. U. Dowla, and D. M. Goodman.
Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method.
Int'l J. of Neural Systems, 2(4):291–301, 1991.

[75] H. Kitano.
Empirical studies on the speed of convergence of neural network training using genetic algorithms.
In *Proc. of the Eighth Nat'l Conf. on AI (AAAI-90)*, pages 789–795. MIT Press, Cambridge, MA, 1990.

[76] X. Yao.
Optimization by genetic annealing.
In M. Jabri, editor, *Proc. of Second Australian Conf. on Neural Networks*, pages 94–97, Sydney, Australia, 1991.

[77] J. J. Merelo, M. Patón, A. Cañas, A. Prieto, and F. Morán.

Optimization of a competitive learning neural network by genetic algorithms.
In *Proc. of Int'l Workshop on Artificial Neural Networks (IWANN'93)*, pages 185–192. Springer-Verlag, 1993.
Lecture Notes in Computer Science, Vol. 686.

[78] G. F. Miller, P. M. Todd, and S. U. Hegde.
Designing neural networks using genetic algorithms.
In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 379–384. Morgan Kaufmann, San Mateo, CA, 1989.

[79] H. Kitano.
Designing neural networks using genetic algorithms with graph generation system.
Complex Systems, 4:461–476, 1990.

[80] S. A. Harp, T. Samad, and A. Guha.
Towards the genetic synthesis of neural networks.
In J. D. Schaffer, editor, *Proc. of the Third Int'l Conf. on Genetic Algorithms and Their Applications*, pages 360–369. Morgan Kaufmann, San Mateo, CA, 1989.

[81] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman.
Using genetic search to exploit the emergent behavior of neural networks.
Physica D, 42:244–248, 1990.

[82] S. W. Wilson.
Perceptron redux: emergence of structure.
Physica D, 42:249–256, 1990.

[83] N. Dodd, D. Macfarlane, and C. Marland.
Optimisation of artificial neural network structure using genetic techniques implemented on multiple transputers.

In P. Welch, D. Stiles, T. L. Kunii, and A. Bakkers, editors, *Proceedings of Transputing'91*, pages 687–700. IOS, Amsterdam, 1991.

[84] S. A. Harp, T. Samad, and A. Guha.

Designing application-specific neural networks using the genetic algorithm.

In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 447–454. Morgan Kaufmann, San Mateo, CA, 1990.

[85] W. B. Dress.

Darwinian optimization of synthetic neural systems.

In M. Caudill and C. Butler, editors, *Proc. of the 1st IEEE Int'l Conf. on Neural Networks, Vol. 3*, pages 769–775. IEEE, New York, NY, 1987.

[86] A. Bergman and M. Kerszberg.

Breeding intelligent automata.

In M. Caudill and C. Butler, editors, *Proc. of the 1st IEEE Int'l Conf. on Neural Networks, Vol. 3*, pages 63–69. IEEE, New York, NY, 1987.

[87] P. J. B. Hancock.

GANNET: desgin of a neural net for face recognition by genetic algorithm.

Technical Report CCCN-6, Center for Cognitive and Computational Neurosicence, Dept. of Computing Sci. and Psychology, Stirling University, Stirling FK9 4LA, UK, August 1990.

[88] C. P. Dolan and M. G. Dyer.

Towards the evolution of symbols.

In *Proc. of 2nd Int'l Conf. on Genetic Algorithms and Their Applications*, pages 123–131. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[89] B. Fullmer and R. Miikkulainen.

Using marker-based genetic encoding of neural networks to evolve finite-state behaviour.

In F. J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proc. of the First European Conference on Artificial Life*, pages 255–262. MIT Press, Cambridge, MA, USA, 1991.

[90] M. Zaus and R. Megnet.

Fusion-technology and the design of evolutionary machines for neural networks.

In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks: Proc. of Int'l Conf. on Artificial Neural Networks — ICANN-91, Vol. 2*, pages 1165–1168. North-Holland, Amsterdam, 1991.

[91] N. Dodd.

Optimisation of neural-network structure using genetic techniques.

In G. Rzevski and R. A. Adey, editors, *Proc. of the Conf. on Applications of Artificial Intelligence in Engineering VI*, pages 939–944. Elsevier Applied Science, London, UK, 1991.

[92] S. J. Marshall and R. F. Harrison.

Optimization and training of feedforward neural networks by genetic algorithms.

In *Proc. of the Second IEE International Conference on Artificial Neural Networks*, pages 39–43. IEE Press, London, UK, 1991.

[93] S. Oliker, M. Furst, and O. Maimon.

A distributed genetic algorithm for neural network design and training.

Complex Systems, 6:459–477, 1992.

[94] L. Martí.

Genetically generated neural networks I: representational effects.

In *Proc. of Int'l Joint Conf. on Neural Networks (IJCNN'92 Baltimore), Vol. IV*, pages 537–542. IEEE Press, Piscataway, NJ, 1992.

[95] W. Schiffmann, M. Joost, and R. Werner.

Synthesis and performance analysis of multilayer neural network architectures.

Technical Report 16/1992, University of Koblenz, Institute für Physics, Rheinau 3-4, D-5400 Koblenz, 1992.

[96] H.-M. Voigt, J. Born, and I. Santibáñez-Koref.

Evolutionary structuring of artificial neural networks.

Technical report, Technical University Berlin, Bionics and Evolution Techniques Lab, Ackerstraße 71-76 (ACK1), D-1000 Berlin 65, 1993.

[97] F. J. Marín and F. Sandoval.

Genetic synthesis of discrete-time recurrent neural network.

In *Proc. of Int'l Workshop on Artificial Neural Networks (IWANN'93)*, pages 179–184. Springer-Verlag, 1993.

Lecture Notes in Computer Science, Vol. 686.

[98] E. Alba, J. F. Aldana, and J. M. Troya.

Fully automatic ANN design: a genetic approach.

In *Proc. of Int'l Workshop on Artificial Neural Networks (IWANN'93)*, pages 399–404. Springer-Verlag, 1993.

Lecture Notes in Computer Science, Vol. 686.

[99] D. White and P. Ligomenides.

GANNet: a genetic algorithm for optimizing topology and weights in neural network design.

In *Proc. of Int'l Workshop on Artificial Neural Networks (IWANN'93)*, pages 322–327. Springer-Verlag, 1993.

Lecture Notes in Computer Science, Vol. 686.

[100] H. Andersen.

A constructive algorithm for a multilayer perceptron based on co-operative population concepts in genetic algorithms.

Master's thesis, University of Queensland, Department of Electrical and Computer Engineering, Brisbane, Qld 4072, Australia, September 1993.

[101] M. Bichsel and P. Seitz.

Minimum class entropy: a maximum information approach to layered networks.

Neural Networks, 2:133–141, 1989.

[102] D. B. Fogel.

An information criterion for optimal neural network selection.

IEEE Trans. on Neural Networks, 2:490–497, 1991.

[103] J. Utans and J. Moody.

Selecting neural network architectures via the prediction risk: application to corporate bond rating prediction.

In *Proc. of the First Int'l Conf. on AI Applications on Wall Street*, pages 35–41. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[104] W. M. Spears and V. Anand.

A study of crossover operators in genetic programming.

In Z. W. Ras and M. Zemankova, editors, *Proc. of the 6th International Symposium on Methodologies for Intelligent Systems (ISMIS '91)*, pages 409–418. Springer-Verlag, Berlin, Germany, 1991.

[105] E. Mjolsness, D. H. Sharp, and B. K. Alpert.

Scaling, machine learning, and genetic neural nets.

Advances in Applied Mathematics, 10:137–163, 1989.

[106] J. W. L. Merrill and R. F. Port.

Fractally configured neural networks.

Neural Networks, 4:53–60, 1991.

[107] F. Gruau.

Genetic synthesis of boolean neural networks with a cell rewriting developmental process.

In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages

55–74. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[108] Y. Doi.

Morphogenesis of Life Forms.

Saiensu-sha, 1988.

[109] S. S. Wilson.

Teaching network connectivity using simulated annealing on a massively parallel processor.

Proceedings of IEEE, 79:559–566, 1991.

[110] H. H. Szu and R. L. Hartley.

Nonconvex optimization by fast simulated annealing.

Proceedings of IEEE, 75:1538–1540, 1987.

[111] G. Mani.

Learning by gradient descent in function space.

In *Proc. of IEEE Int'l Conf. on System, Man, and Cybernetics*, pages 242–247, Los Angeles, CA, 1990.

[112] D. R. Lovell and A. C. Tsoi.

The performance of the Neocognitron with various S-cell and C-cell transfer functions.

Intelligent Machines Lab., Dept. of Elec. Eng., Univ. of Queensland, April 1992.

[113] B. DasGupta and G. Schnitger.

Efficient approximation with neural networks: a comparison of gate functions.

Technical report, Dept. of Computer Sci., Pennsylvania State Univ., University Park, PA 16802, 1992.

[114] D. G. Stork, S. Walker, M. Burns, and B. Jackson.

Preadaptation in neural circuits.

In *Proc. of Int'l Joint Conf. on Neural Networks, Vol. I*, pages 202–205, Washington, DC, 1990. Lawrence Erlbaum Associates, Hillsdale, NJ.

[115] M. N. Narayanan and S. B. Lucas.

A genetic algorithm to improve a neural network to predict a patient's response to Warfarin.

Methods of Information in Medicine, 32:55–58, 1993.

[116] Z. Guo and R. E. Uhrig.

Using genetic algorithms to select inputs for neural networks.

In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 223–234. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[117] F. Z. Brill, D. E. Brown, and W. N. Martin.

Fast genetic selection of features for neural network classifiers.

IEEE Transactions on Neural Networks, 3:324–328, 1992.

[118] L. S. Hsu and Z. B. Wu.

Input pattern encoding through generalised adaptive search.

In D. Whitley and J. D. Schaffer, editors, *Proc. of the Int'l Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 235–247. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[119] E. J. Chang and R. P. Lippmann.

Using genetic algorithms to improve pattern classification performance.

In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems (3)*, pages 797–803. Morgan Kaufmann, San Mateo, CA, 1991.

[120] B.-T. Zhang and G. Veenker.

Neural networks that teach themselves through genetic discovery of novel examples.

In *Proc. of 1991 IEEE International Joint Conference on Neural Networks (IJCNN'91 Singapore)*, volume 1, pages 690–695. IEEE Press, New York,

NY, 1991.

[121] A. Artola, S. Broecher, and W. Singer.

Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex.

Nature, 347:69–72, 1990.

[122] P. J. B. Hancock, L. S. Smith, and W. A. Phillips.

A biologically supported error-correcting learning rule.

In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Proc. of Int'l Conf. on Artificial Neural Networks — ICANN-91, Vol. 1*, pages 531–536. North-Holland, Amsterdam, 1991.

[123] J. Maynard Smith.

When learning guides evolution.

Nature, 329:761–762, 1987.

[124] G. E. Hinton and S. J. Nowlan.

How learning can guide evolution.

Complex Systems, 1:495–502, 1987.

[125] R. K. Belew.

Evolution, learning and culture: computational metaphors for adaptive algorithms.

Technical Report #CS89-156, Computer Science & Engr. Dept. (C-014), Univ. of California at San Diego, La Jolla, CA 92093, USA, September 1989.

[126] S. Nolfi, J. L. Elman, and D. Parisi.

Learning and evolution in neural networks.

Technical Report CRT-9019, Center for Research in Language, University of California, San Diego, La Jolla, CA 92093-0126, U.S.A., July 1990.

[127] H. Mühlenbein and J. Kindermann.

The dynamics of evolution and learning — towards genetic neural networks.

In R. Pfeifer et al., editor, *Connectionism in Perspective*, pages 173–198. Elsevier Science Publishers B.V., Amsterdam, 1989.

[128] H. Mühlenbein.

Adaptation in open systems: learning and evolution.

In J. Kindermann and C. Lischka, editors, *Workshop Konnektionismus*, pages 122–130. GMD, Postfach 1240, D-5205 St., Augustin, Germany, 1988.

[129] J. Paredis.

The evolution of behavior: some experiments.

In J.-A. Meyer and S. W. Wilson, editors, *Proc. of the First Int'l Conf. on Simulation of Adaptive Behavior: From Animals to Animats*. MIT Press, Cambridge, MA, 1991.

[130] D. J. Chalmers.

The evolution of learning: an experiment in genetic connectionism.

In D. S. Touretzky, J. L. Elman, and G. E. Hinton, editors, *Proceedings of the 1990 Connectionist Models Summer School*, pages 81–90. Morgan Kaufmann, San Mateo, CA, 1990.

[131] Y. Bengio and S. Bengio.

Learning a synaptic learning rule.

Technical Report 751, Départment d'Informatique et de Recherche Opérationelle, Université de Montréal, Canada, November 1990.

[132] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei.

On the optimization of a synaptic learning rule.

In *Preprints of the Conference on Optimality in Artificial and Biological Neural Networks*, Univ. of Texas, Dallas, Feb. 6–8, 1992.

[133] J. F. Fontanari and R. Meir.

Evolving a learning algorithm for the binary perceptron.

Network, 2:353–359, 1991.

[134] D. H. Ackley and M. S. Littman.

Interactions between learning and evolution.

In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X*, pages 487–509, Reading, MA, 1991. Addison-Wesley.

[135] J. Baxter.

The evolution of learning algorithms for artificial neural networks.

In D. Green and T. Bossomaier, editors, *Complex Systems*, pages 313–326. IOS Press, Amsterdam, 1992.

[136] D. Crosher.

The artificial evolution of a generalized class of adaptive processes.

In X. Yao, editor, *Preprints of AI'93 Workshop on Evolutionary Computation*, pages 18–36, November 1993.

[137] R. A. Jacobs.

Increased rates of convergence through learning rate adaptation.

Neural Networks, 1:295–307, 1988.

[138] B. Widrow and M. E. Hoff.

Adaptive switching circuits.

In *1960 IRE WESTCON Convention Record*, pages 96–104. IRE, New York, NY, 1960.

[139] D. Parasi, F. Cecconi, and S. Nolfi.

Econets: neural networks that learn in an environment.

Network, 1:149–168, 1990.

[140] X. Yao.

The evolution of connectionist networks.

In T. Dartnall, editor, *Artificial Intelligence and Creativity*, pages 233–243. Kluwer Academic Publishers, Dordrecht, 1994.

[141] G. Weiss.

Combining neural and evolutionary learning: aspects and approaches.

Technical Report FKI-132-90, Institut für Informatik, Technische Universität München, May 1990.