# Evolutionary Algorithms for Neural Network Design and Training

Jürgen Branke
University of Karlsruhe
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren
D-76128 Karlsruhe, Germany
Email: branke@aifb.uni-karlsruhe.de

January 1995

**Abstract**

Neural networks and genetic algorithms are two relatively young research areas that were subject to a steadily growing interest during the past years. Both models are inspired by nature, but whereas neural networks are concerned with learning of an individual (phenotypic learning), evolutionary algorithms deal with a population's adaptation to a changing environment (genotypic learning).

This paper focuses on the intersection of neural networks and evolutionary computation, namely on how evolutionary algorithms can be used to assist neural network design and training. The purpose of the paper is to set forth the general considerations that have to be made when designing an algorithm in this area and to give an overview on how researchers addressed these issues in the past.

**Keywords:** artificial neural networks, evolutionary algorithms, evolutionary training, evolutionary design.

## 1 Introduction

Apart from tasks that mainly depend on simple arithmetic, the brain is far superior compared to common digital computation: the brain is fault tolerant, can interpret imprecise information, adapt to new situations etc.

Naturally, scientists in the early 1940s [49] tried to use knowledge gained in neural biology when they were looking for improvement of conventional computing. As a result, in analogy to the brain, an artificial neural network is composed of many very simple calculating units (also called neurons) that are connected to form a network. The structure of the network determines whether one neuron may influence another. Weights assigned to each connection specify the extent of possible influence.

[0]Technical Report No. 322, University of Karlsruhe, Institute AIFB.
Also published in: Jarmo TÅlander, ed; Procceedings of the 1st Nordic Workshop on Genetic Algorithms and its Applications. Vaasa, Finland, 1995

Despite the fact that the single neurons are extremely simple, the network as a whole is very powerful. Indeed it has been shown that artificial neural networks, if large enough, can approximate arbitrary continuous functions. (For introductory literature to the field of neural networks, the reader is referred to e.g. [31]).

But although knowing there exists a suitable network for a specific problem is important, finding it proved to be difficult. Although there exist some algorithms to set the weights by learning from presented input/output examples given a fixed topology, these algorithms often get stuck in local minima. To lead to good results, they strongly depend on problem specific parameter settings and on the topology of the network.

To determine a good or perhaps optimal topology is even more difficult and has already been labeled a "black art" [51]. Indeed most often today, an appropriate structure is created by intuition and time consuming trial and error.

To use evolutionary algorithms as a global and very broad search procedure to assist neural network design and training seems to be a straightforward idea, especially since the real brain also is somehow the result of biological evolution.

Evolutionary algorithms (EAs) are a class of probabilistic adaptive algorithms based on the principles of biological evolution. The three major forms of these algorithms currently are Genetic Algorithms, Evolution Strategies and Evolutionary Programming. For a comparison of these representatives see for example [19, 20, 33].

In general, evolutionary algorithms distinguish themselves from similar search algorithms by working on a population of individuals each representing a possible (trial) solution to the problem. Each individual is assigned a "fitness score" according to how good a solution to the problem it is.

The algorithm's three main operators are selection, recombination and mutation. Only individuals that are competitive (according to their fitness value) get the possibility to survive long enough to produce offspring by recombination and mutation and thus to transfer their genetic material to the next generation.

By considering many points in the search space simultaneously, evolutionary algorithms reduce the risk of converging to local optima. Although they use probabilistic rules to guide their search, by favoring the mating of the fitter individuals, the most promising areas in search space are explored.

Past experiences demonstrated that evolutionary algorithms represent effective and robust search algorithms that allow to quickly locate areas of high quality solutions even if the search space is very large and complex. This quality makes these algorithms well suited to artificial neural network design and training where the search space is infinite, highly dimensional and multimodal [74].

The user who wants to apply evolutionary algorithms to solve a specific problem has at least to provide problem specific knowledge in the following form:

- *representation:* evolutionary algorithms work with "genetic" representations of trial solutions, usually in form of a string of real or integer numbers. The user has to provide a suitable representation and a function that maps genetic representation into phenotypic trial solutions.

- *performance:* a function has to be provided that associates a performance value with each individual. The performance should reflect how good or how useful the individual is to solve the considered problem.

- *creation of offspring:* the user has to specify operators (e.g. crossover or mutation) that allow the creation of new individuals given one or two parent individuals. Very often these operators need repair functions to ensure that the offspring is a valid trial solution, or they include local hill climbing to speed up the local fine tuning.

In recent years, various schemes for combining evolutionary algorithms and neural networks have been proposed and there is a large body of literature on the field. This article attempts to give an overview on the subject and to point out the general considerations that have to be made when working in that area.

The focus lies on problem specific aspects, not so much on the applications or test problems used or GA-design features that do not directly relate to the neural network problem.

Also, the article does not claim to be complete. For supplementary overviews on the subject, the reader is referred to [61, 65, 74] or [7] (taxonomy and guide to literature). A bibliography with 510 related articles can be found in [2].

In most cases, Evolutionary Algorithms have been used to either train the network or to find a suitable topology. This is reflected in the outline of the paper:

Section 2 is concerned with evolutionary approaches to find the weights for a network, whereas Section 3 focuses on using evolutionary algorithms to find the network topology. In Section 4, evolutionary algorithms to simultaneously develop weights and structure are covered. Further interesting approaches that do not fit into these categories are dealt with in Section 5.

In all of the above sections, it is tried to point out different possibilities to deal with the issues of representation, performance evaluation and reproduction operators.

Finally, Section 6 is concerned with a problem that frequently arises when applying evolutionary algorithms to neural networks, the so called *competing conventions problem.*

The paper concludes with some general remarks.

## 2 Evolutionary Algorithms for Neural Network Training

Given a fixed topology, specifying the weights of a network can be seen as an optimization process with the goal to find a set of weights that minimizes the network's error on the training set. Unfortunately, the problem's error surface is highly dimensional and usually contains many local minima.

Nevertheless, the most widely used algorithm for that problem is the backpropagation algorithm [62] which is a local gradient search method. As such, it is prone to get stuck in local minima and it needs gradient information. Also, the success of backpropagation methods very much depends on good, problem specific parameter settings.

On the other hand, evolutionary algorithms usually avoid local minima by searching in several regions simultaneously (working with a population of trial solutions). And the only information they need is some performance value that determines how good a given set of weights is (no gradient information). Furthermore, evolutionary algorithms place no restrictions on network topology because they do not require backward propagation of an error signal.

Thus, to apply evolutionary algorithms seems to be advantageous at least to problems where gradient information is difficult to obtain, e.g. to recurrent networks, to networks with non differentiable transfer functions or non differentiable optimality criteria, to product neurons [34] or threshold neurons.

One drawback of genetic algorithms is that they seem to have difficulties to fine tune the parameters [39]. And there is the problem of competing conventions discussed in Section 6.

So far, evolutionary algorithms in this area do not seem to be competitive with improved gradient decent methods like quickprop or cascade correlation, see [39, 65].

For some work in that area see e.g. [13, 34, 52, 69, 76, 77, 78, 81].

In [15, 41], recurrent nets are trained.

## 2.1 Representation

The straightforward genotype representation is simply a concatenation of all the network's weights in a string.

Since the standard single point crossover operator is more likely to disrupt genes that are far apart on the chromosome than to disrupt genes located close to each other, it may be useful to place functional units tightly together on the genotype. For that reason, Thierens et al. [69] suggest to place the incoming and outgoing weights of a hidden node next to each other. Yoon et al. [81] place side by side all incoming weights of a node, and all nodes of a layer.

Taking this idea one step further one might consider a whole functional unit as one gene, or as the smallest unit of a genotype the reproduction operators act upon. See Section 2.3 for some ideas in this direction.

Perhaps the most fundamental decision is the choice between binary and real valued encoding. Whereas standard genetic algorithms use binary encoding (e.g. [13, 34, 76]), real valued encoding seems to become dominating in the current literature (e.g. [41, 52, 69, 81]). For larger problems, binary encodings result in very large strings (thousands of bits) which can slow down the evolution process [3, 78]. Maniezzo [45, 46] (optimizing weights and structure at the same time) used binary encoding, but additionally included a granularity parameter in the genotype to simultaneously optimize how many bits should be used to encode one weight value.

## 2.2 Performance Evaluation

The quality of a set of weights usually is defined as the corresponding network's mean squared error (mse) on a given training set. Since the goal is to minimize this error, it does not directly translate into a fitness value which have to be the larger the better the individual. One way to handle this transformation is to use rank based selection [34, 77]. Alternatively, one can transform the error by using $\frac{1}{mse}$ [47] or $\frac{1}{1+mse}$ [34] as fitness value. If some maximum mse is known, max_mse - mse is yet another alternative [51].

As no gradient information is needed, nondifferentiable functions like the percentage of correct answers (classification accuracy) on the training set could also be used. However, it is advantageous if the performance measure is continuous, not just discrete, since this allows the genetic algorithm to better discriminate between the performance of different individuals.

### 2.3 Reproduction Operators

Of course, on a population of weight strings, conventional crossover and mutation would be applicable. However, it has already been suggested in Section 2.1 that one has to decide on which level the operators are allowed to act on, i.e. on what constitutes a gene. For the training of a recurrent XOR-network, Kohlmorgen et al. [41] concluded from their experiments that combining all outgoing weights of a unit to one gene was most useful. In several approaches [52, 53], the collection of ingoing weights to a unit is considered as one gene (and not disrupted by crossover). Thierens et al. [69] have the crossover operator exchange hidden units with all incoming and outgoing weights in a feed-forward-network with only one hidden layer.

Montana and Davis [52] did not find significant differences between an operator that always transferred all incoming weights of a node to the offspring and one that worked on single weight values. However, they found a mutation operator that acts simultaneously on all ingoing weights of a node to be more helpful than an operator acting on single weight values.

In a multi-layer network, Yoon et al. [81] found it advantageous to fix the weights of some layers (res. the corresponding parts of the genotype) during the evolution. That is, while at the beginning the whole genotype is subject to mutation and crossover, later on, parts of the genotype are kept constant.

### 2.4 Hybrid Approaches

Since standard evolutionary algorithms are very efficient for global search, but relatively slow in local fine tuning, it seems natural to integrate back-propagation, a local gradient method, as local hill climber.

The rationale behind this is that it restricts the search space to locally fit individuals.

How those hybrid approaches work in practice is still unclear. In [8] and [30] successful results are reported, whereas Kitano's experiments [39] suggest that hybrid GA/BP approaches do not provide any advantage over a randomly initialized, multiple application of quickprop, at least for shallow networks and easy fitness functions.

As reported in [30], a strategy that switches from the evolutionary algorithm to back-propagation after convergence might be advantageous for large networks.

## 3 Evolutionary Algorithms to Determine the Structure of a Neural Network

A neural network's structure greatly influences its performance. Most neural networks today have one or two fully connected hidden layers. But this might not be the best possible choice: it could well be appropriate to use more hidden layers, partially connected or with direct connections from input to output [75, 78].

If the topology is too small (in terms of units and connections) the network might not be able to represent or even learn the desired input/output mapping. On the other hand, if it is too large, the network very often generalizes poorly to inputs previously unseen (for some theoretical aspects on generalization see e.g. [1]).

Besides, the topology influences speed and accuracy of backpropagation learning, fault tolerance and representational comprehensibility.

But although structure is a very important aspect in neural network design, it is not only still impossible to determine an optimal structure for a given problem, it is even impossible to prove that a given structure is optimal.

And apart from some construction/destruction heuristics (e.g. [18, 58, 67]) and some rules of thumb, today most often a suitable structure is created by intuition and time consuming trial and error.

Evolutionary algorithms there seem to be a promising approach to perform an automated search method for an optimal topology.

For some work in this area see for example [16, 29, 28, 38, 44, 51, 66, 72].

## 3.1 Representation

Representing the structure of a neural network is not as straightforward. Considerations have to be made as to

- whether the optimal or at least many quasi-optimal solutions can be represented,
- whether flawed or meaningless network structures should be excluded,
- whether the reproduction operators yield valid offspring and
- how the representation scales up to larger networks.

Ideally we would like a genetic space of network structures which does not contain any unviable network genotypes, but which spans the space of all potentially useful network genotypes.

The tradeoff between expressive power and the exclusion of flawed or meaningless networks is difficult: neither does one wish to overly constrain the architecture, nor should it grow out of control.

Basically there are two general paradigms to design a representation of a network's structure: the strong, direct or low-level encoding [51, 66, 72] and the weak, indirect or high-level encoding [29, 28, 38, 44].

Low level encodings specify each connection individually, while high level encodings are more like growth rules, that may specify many units and connections simultaneously, perhaps stochastically. Examples of both schemes are given in Figure 1 and 2.

In most cases, direct encodings specify the connections only. To remove a whole unit from the network the algorithm would have to remove all ingoing or outgoing weights. An exemption is for example [60] where the GA determines the number of hidden layers and the number of nodes in each hidden layer with the layers being fully connected. A rather restricted view is taken by Arena et al. [5, 6]. In the first paper, only the number of hidden units in a one hidden layer network, and in the second paper the distribution of a fixed number of hidden units into layers is optimized.

The first high level encoding scheme has probably been proposed by Harp, Samad and Guha [28, 29]. They specify three-dimensional areas and for each area the number of nodes and the connection densities to some other areas. In addition, the backpropagation learning rate is a parameter of an area. Note that this representation allows variable string

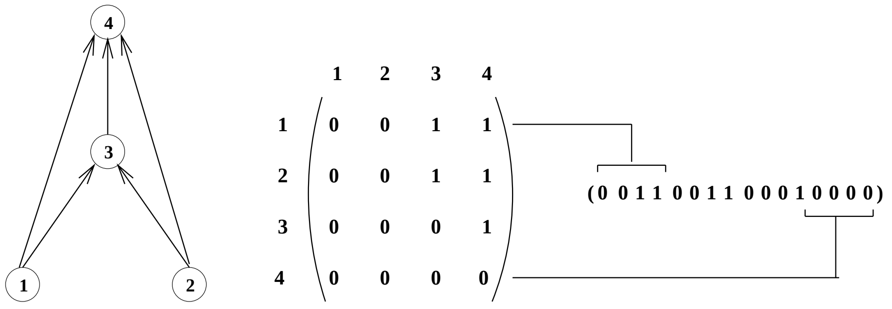

Figure 1: Example of a low level encoding as proposed in [51]. The chromosome directly translates into the adjacency matrix, after which the network can be constructed.

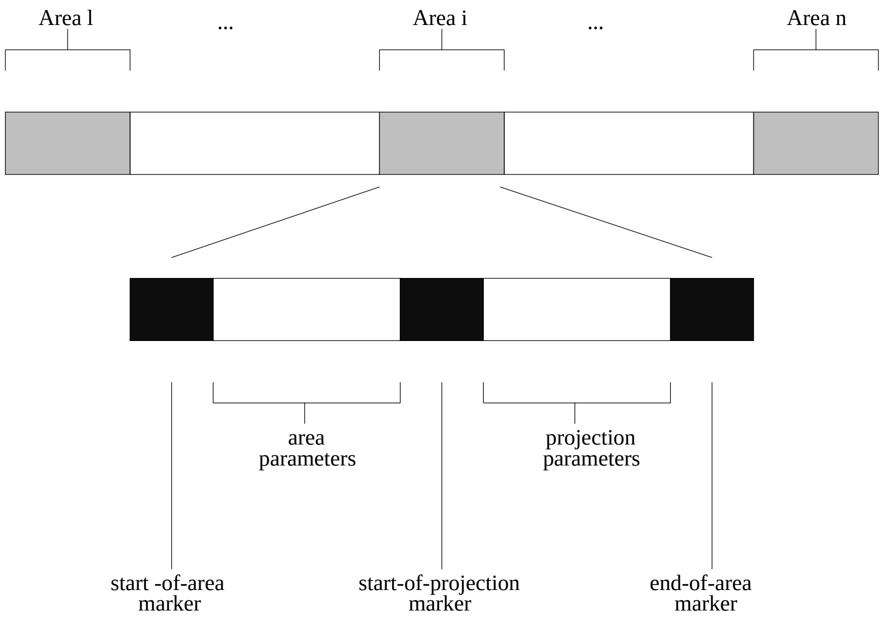

Figure 2: Example of a high level encoding according to [38]. Here, each area defines a set of neurons, their spatial arrangement and their connections to other areas (density and projection field).

length. Mandischer [45] proposed a somewhat related representation, where for every layer receptive as well as projective connections are specified on the chromosome.

Another high level encoding scheme is the graph generation system used by Kitano [38]. Here, the genetic algorithm is used to develop the rules of a context free and deterministic graph grammar according to which the network structure can be generated. Kitano's experimental results indicate that this encoding is almost not affected by the network size and that the nets created are very regular.

Voigt et al. [73] used a stochastic graph generation grammar and had an evolution strategy adapt the control parameters.

The obvious drawback of the low level encoding is the explosive increase in genotype length as the network grows. Many researchers conclude that therefore this encoding has bad scaling properties and is useful for relatively small networks only (e.g. [38]). The maximum topology and thus the space of topologies to be searched is limited by the user. On the one hand, that prevents the network from growing to an arbitrarily large size and it allows to evolve networks with special connectivity patterns by constraining the representation. On the other hand, this may exclude the fittest structures from the search space.

In high level encodings, not every structure is equally probable, but usually regular networks are favored. Whereas sometimes this is desired, one has to be aware of the human design bias that might exclude fit structures from the search.

Researchers using high level encodings usually explain that with their better scalability, desired regularities and biological plausibility.

No matter which encoding is used, similar to the case when evolving the weights, one may try to pack genes comprising functional units closely together on the chromosome.

## 3.2 Performance Evaluation

It is yet impossible to evaluate a network structure's quality directly. Only after having been trained, the network as a whole can be evaluated and the result can be used as an estimation of the structure's quality.

Since the result may depend on the training algorithm and since several runs usually lead to different local minima, this estimate can be biased, or at least be noisy. However, evolutionary algorithms proved to be relatively noise tolerant in practice.

Usually, the net is trained for some fixed number of epochs and then the performance on the training set is measured. Most common performance measure is the mean squared error [66], but also the integral of the error over the epochs has been used [29, 28].

A very big advantage of evolutionary algorithms compared to other optimization methods is that heuristics can be incorporated into the evaluation. For example, the evaluation function may prefer smaller networks as those are expected to exhibit a better generalization behavior. Whitley, Starkweather and Bogart [75, 78] introduced a bias towards smaller networks by allocating the more training time the smaller the network. That way, the smaller nets are not actually rewarded unless they are able to exploit the opportunity. Other researchers [17, 14] as well successfully applied this reward scheme.

Dodd [16] used a combination of reached accuracy and network complexity for evaluation, Harp et al. [28] additionally included the speed of learning.

In [40], Kitano extended his earlier work [38] to include the evolution of the initial weight distribution for backpropagation. This approach lies somewhere in the middle of Sections 3 and 4 as it includes evolution of initial weights, but still uses backpropagation to train the networks.

In several approaches [6, 12, 45, 47, 60, 80] the net, after having been trained on a training set, is evaluated on a separate "evaluation set" to determine the fitness of the structure. Dodd et a.l. [17] used a convex combination of the net's performance on evaluation and training set as performance measure. By using different sets for training and evaluation, it is hoped to evolve structures with better generalization abilities. However, for a fair comparison the evaluation set should be considered as an additional training set as it is used to find the optimized network. To test the generalization abilities, yet another test set would be necessary. Whether it is actually beneficial to divide the available training set into one set used for backpropagation learning and one used for evaluation remains to be seen.

As each evaluation of an individual involves training, the computational cost is quite large. Therefore, and since evolutionary algorithms are inherently parallel, it can be expected that parallel implementations will gain ground [17, 68].

### 3.3 Reproduction Operators

The most common operators are simple or uniform crossover and mutation that changes single values.

As Utrecht and Trint [71] point out, a good mutation operator should adhere to the principle of strong causality, i.e. it should in most cases cause small differences in quality. Also, it should allow short transition paths between any pair of structures. In their paper, Utrecht and Trint propose a number of heuristic structure mutation operators and test them for these two qualities.

Schiffmann [66] restricts mutation to adding or deleting weak connections.

The crossover operator in [51] can only exchange whole sets of incoming connections to a unit. But although this intuitively makes sense (transfering whole functional units), the operator did not prove to be advantageous in experiments reported by [72].

Some specific representations, of course, need suitable operators that take into account the representation's peculiarities.

### 3.4 Backpropagation Control Parameters

It has already been mentioned that evaluating a structure involves training the corresponding network and thus requires large computational power. Obviously, speeding up training would greatly reduce the required effort. Thus, many researchers try to optimize back-propagation parameters by the genetic algorithm along with the optimization of the structure.

To do this, the backpropagation parameters most often are just added to the chromosome as additional values.

The most common parameters to be optimized are learning rate [8, 29, 28, 47, 60, 80], momentum [8, 47, 80] and initial weight range [8, 47].

In the approach by Harp et al. [28] the learning rate can vary among groups of neurons and exponentially decays during the training according to a slope parameter also set by the genetic algorithm. The learning rates found by the EA seem to be high compared to human choices [28].

Further parameters to be optimized by an evolutionary algorithm could be the activation function, the learning strategy, a weight decay term [47] or the number of training epochs [47].

## 4 Evolutionary Algorithms to Simultaneously Determine Weights and Structure of a Neural Network

Instead of using back-propagation to train the networks over and over again, it seems to be a valid idea to have the evolutionary algorithm search for structure and weights simultaneously [11, 12, 14, 32, 36, 48, 63].

The works of Potter [56] and Karunanithi et al. [35] also use an evolutionary algorithm to determine structure and weights of the network. However, their approaches are inspired by Fahlman's cascade-correlation algorithm that starts with a minimal network and dynamically builds a suitable cascade structure by training and installing one hidden unit at a time until the problem is successfully learned. Thus the structure is not directly optimized by the evolutionary algorithm but rather a result of the cascade algorithm.

In [4, 9, 64, 68, 70] the weights and topology of recurrent neural networks are determined, Zhang [82] optimizes Sigma-Pi networks

A quite unusual approach has been proposed by Oliker et al. [55] where the search for the optimal neural network is done separately for every single neuron, i.e. separately in different genetic algorithms working together to finally build an optimal network structure. The underlying idea is that the search space is drastically reduced in comparison to genetic algorithms that are responsible for the whole network. The proposed evaluation scheme is rather complex and takes into account the network's error and an estimation for the network's convergence capabilities. To reduce the complexity of the evaluation, the networks have been restricted to feed-forward topologies with binary linear threshold units.

As to representation, fitness evaluation and genetic operators, most of the aspects presented in Sections 2 and 3 are also valid for this Section and shall not be repeated. However, there are some special facets that shall be described in the following.

### 4.1 Representation

As in the case with mere topology optimization, here again direct or low level encodings (e.g. [11, 12, 14, 48, 68, 70]) and high level encodings (e.g. [21]) can be distinguished.

An example of a typical low level encoding is given in Figure 3

Koza and Rice [42] also used a direct encoding, but theirs is in terms of Lisp-S-expressions.

Saha and Christensen [63] chose as representation a fixed length binary string with a predefined space per neuron on which in- or outgoing connections from or to other neurons as well as their weight could be specified explicitly.

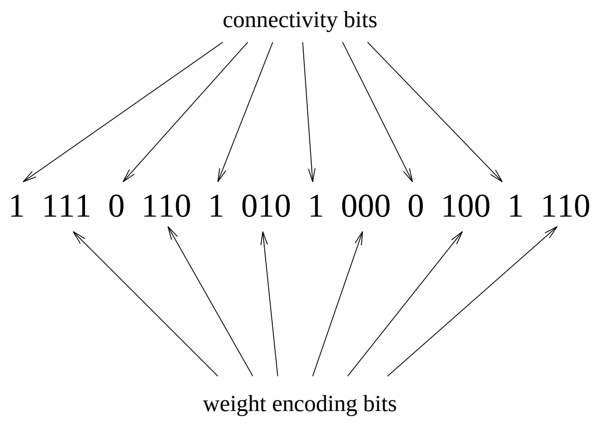

Figure 3: Low level encoding for structure and weights of a neural network as suggested by [46]. Each connection of a maximal network structure is specified by two parameters: the connectivity bit (existence/non-existence) and the connection weight.

Sometimes (e.g. [48, 68, 70]), only two bits are used to encode a weight which is then restricted to be either excitatory (1), exhibitory (-1) or non-existing.

Inspired by Kitano [38], Gruau [21] used a grammatical encoding of structure and weights (restricted to $\pm$ 1 for boolean functions). But instead of rewriting matrix elements, Gruau proposed a cell rewriting process. Koza [43] suggested an approach that uses LISP-S-expressions for representation. In both cases, the chromosomes used can be depicted as syntactic trees similar to those in Genetic Programming (cf. Koza [42]) and the same specialized recombination operators exchanging subtrees of the individuals are applied.

Dasgupta and McGregor [14] designed a two-level representation where the front part of a chromosome (high level) encodes the connectivity pattern and the rest of the chromosome (low level) represents the weights and biases. High level genes can activate sets of lower level genes, i.e. if a connection is labeled non-existent, the weight of that connection, though still in the chromosome, is not used. Maniezzo [46] also used connectivity bits to indicate whether a connection would be used, but he located them directly in front of the corresponding weight bits on the chromosome.

The aspect of encoding functional units together on the chromosome is addressed in Marti [48], where the ordering of the links on the genotype is optimized by an additional ”outer” genetic algorithm.

## 4.2 Performance Evaluation

Optimizing recurrent networks, Bornholdt and Graudenz [9] included the number of update cycles until the network reaches a stable state in the evaluation of their genetically constructed networks.

Kendall and Hall [36, 37] used an estimation for the description length (according to the Minimum Description Length Principle, [59]) for evaluation. This takes into account the network structure, the distribution of weights and the residual error on the training set.

Another complexity measure based on the number of weights, units and layers is considered for evaluation in [82].

### 4.3 Reproduction Operators

Angeline et al. [4] have the severity of weight mutations dictated by how close the network is to being a solution for the task. For structural mutations, they try to avoid radical jumps by initializing new links with zero and only adding hidden units without connections to other units (links can be added by future structural mutations). Deletion of nodes and links is done without any further modifications.

To reduce the computation time Braun and Zagorski [12] and Braun and Weisbrod [11] suggest to inherit the weights from the parents but to train them with backpropagation to a local optimum. This approach stands somewhat between those of Section 3 and Section 4 as for weight finding backpropagation and genetic algorithm work together. All the weights optimized by backpropagation are inherited to the offspring along with the connections. Seen from a genetic viewpoint, this restricts the search space to locally fit individuals. Seen from a back-propagation standpoint, initializing the offspring with parent weights drastically reduces the training time. In their experiments with this strategy, Braun and Zagorski not only observed a speedup of 1 to 2 orders of magnitude but also noted that the found structures could not be trained from scratch, but only with the inherited initialization. In addition to back-propagation, Braun and Zagorski [12] used weight elimination and pruning of small connections as local hill climbers.

The crossover operator suggested by Braun and Weisbrod [11] assigns to the offspring every connection that exists in both parents. If the connection is only used by one parent net, the offspring will get that connection with some user-specified probability. The weight of inherited connections is a user defined fraction of the relating parent weight. Of course this crossover operator needs to be seen together with the method to take care of the competing conventions problem (cf. Section 6).

To soften the rather strong effect of unit mutations, Braun and Zagorski [12] initialized new units with small random weights, preferably deleted units with few connections and "bypassed" units (introducing connections from predecessors to successors) before deleting them.

## 5 Further Work

Three authors suggested training algorithms for one-hidden-layer networks in which the input to hidden connections are determined by an evolutionary algorithm whereas the hidden to output connections are trained by a perceptron learning algorithm.

- In the approach proposed by Wilson [79], the hidden units compute the AND function. A genetic algorithm is used to determine the connectivity of input to hidden units where a connection can be non existent or, if it exists, either transfer the input signal or its complement.

- Munro [53] uses real-valued weight encodings. It is ensured that all hidden units actually discriminate between the input patterns in the training set (i.e. do not respond identically for every input pattern). If this is not the case, the thresholds are adapted.

- Obradovic and Srikumar [54] use a genetic algorithm to incrementally construct a set of hyper planes (corresponding to hidden units) which partition the training set into regions such that almost all training examples belonging to the same region are of one class. These regions are called "resolved" and the training examples in those regions are ignored for the construction of further hyper planes. Hyper planes are added one after the other by subsequent genetic algorithms. For evaluation of trial hyper planes, the percentage of correctly classified training examples by the hyper plane is taken. Rather uncommon is the representation: for an m dimensional input space, each individual consists of m concatenated binary substrings of equal length. Each substring encodes a point on the line between a positive and a negative training example of the same region. The m substrings together define the hyper plane from which the corresponding hidden unit weights can then be constructed. After all regions have been resolved, the connections from hidden to output layer are determined by a pocket algorithm which is similar to simple perceptron learning. It can be shown that the proposed algorithm will always converge.

Whitley, Starkweather and Bogart [75, 78] used a genetic algorithm to prune networks. Starting with a fully connected and already trained (starting) network, the genetic algorithm was used to find links that could be eliminated. The representation was direct and contained a zero or one for every possible link. Considerable training time was saved by initializing the pruned network with the weights from the starting network. Selective pressure towards smaller networks was introduced by allowing more training cycles for smaller networks. Hancock [25] made similar experiments but gradually added noise to the starting weights in order to obtain nets that are capable of learning from random weights.

An approach to reduce the number of different weights in the network (i.e. to introduce weightsharing) by means of a genetic algorithm has been suggested by Branke et al. [10].

Ari Hämäläinen developed a genetic algorithm to determine the topology of a self-organizing (Kohonen-) map [22]. In his fitness function he included a measure of the disorder of the map.

Alba et al. [3] suggested a 3-level genetic algorithm: the top level is used to determine an appropriate number of nodes in each layer, the intermediate level to find a suitable connectivity and the lowest level to set the weights of the network. Each level uses the next lower level for evaluation which of course makes the whole procedure extremely time consuming.

Merelo et al. [50] devised a two layer network for classification problems where the first (hidden) layer is trained following a competitive learning algorithm and the second layer is trained by perceptron learning. The genetic algorithm is used to find learning parameters, the number of units in the first layer and a set of initial weights for the network.

Happel and Murre [27] focus on modularity as a basic design principle. Instead of forming the structure from single neurons, in that paper the evolutionary algorithm arranges "categorization and learning modules" (CALMs) and specifies their interconnectivity pattern. The smallest of such modules already contains 6 neurons with a fixed internal structure.

## 6 Competing Conventions Problem

A neural network property that makes neural network optimization difficult for evolutionary algorithms is that the same functional mapping from input to output can be implemented by a number of different neural networks. The group of functionally equivalent but structurally different networks can be defined by two simple transformations [23, 69].

*First,* permuting the hidden units of a network does not alter the function of the network. Consider the two simple neural networks in Figure 4. The only difference between them is that the two hidden nodes A and B are exchanged. Clearly, both networks implement the same functional mapping. With n hidden neurons, a total of n! functionally equivalent networks are defined by this transformation.

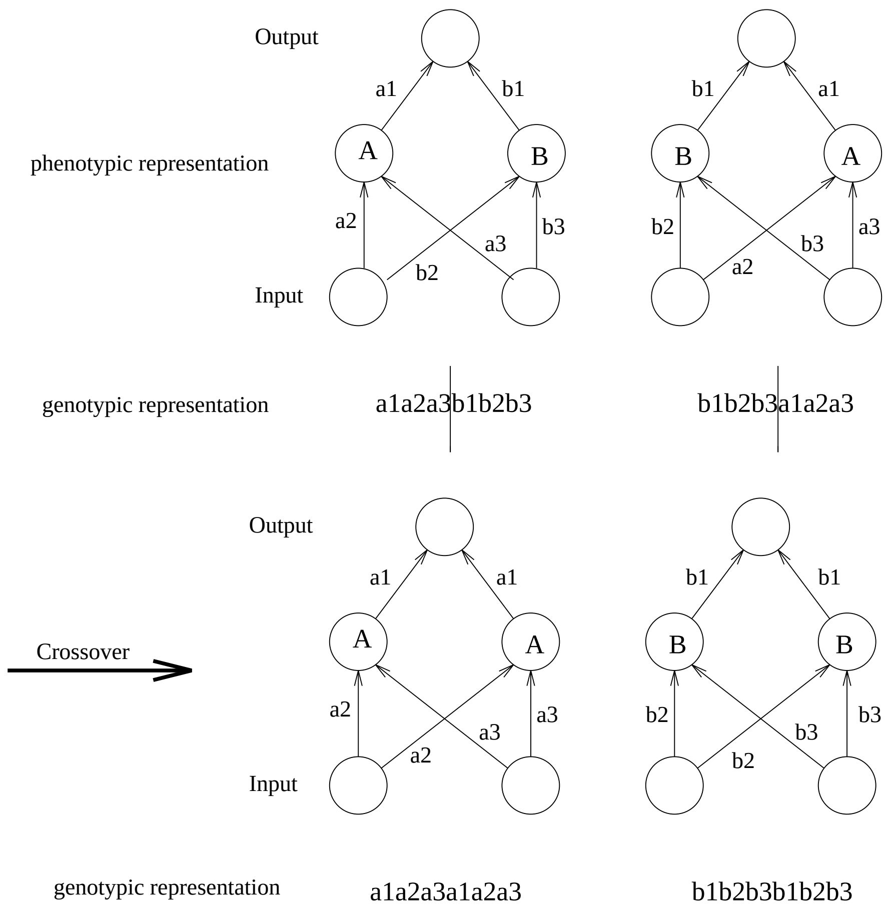

Figure 4: Phenotypic and genotypic representation of two functionally equivalent but structurally different neural networks (top). If a crossover operator is applied to them, the output is likely to be useless (bottom), since it probably contains more than one copy of the same neuron.

*Second,* if the activation function is odd, flipping the signs of all incoming and outgoing weights of a hidden node has no effect on the overall function of the network. As one can choose any combination of the n hidden neurons to flip their weight signs, there are $\sum_{i=0}^{n} \binom{n}{i} = 2^n$ structurally different but functionally identical networks generated by this transformation.

The problem is that the evolutionary algorithm knows nothing about those symmetries. It works on genotypes which represent the structural definition of a network. Structurally different networks are represented by different genotypes, even if the functional mapping they define is the same. If the crossover operator is applied to two functionally similar but structurally very different (different conventions) parent networks, it very often yields totally inappropriate offspring as is illustrated in Figure 4.

From another viewpoint the problem can be seen as an explosion of the search space: each convention (e.g. ordering of the hidden units) is an extra region in the search space, and only recombination of individuals of the same region is promising. Figure 5 illustrates this view: assume there are two networks similar to those depicted in Figure 4. Assume further network X has perfectly learned role A, but not role B, whereas network Y has perfectly learned role B, but not A. Without the competing conventions problem, chances would be high that crossover produced a perfect network. But due to different conventions (node A is the left unit / node A is the right unit) the search space is increased drastically and the efficiency of the crossover operator is severely effected.

The situation is actually worse than this, because there is often more than one way of solving a problem. One net might have roles A and B, another, roles C and D. Crossover will now combine imcompatible roles which will lead to problems similar to competing conventions.

It is yet unclear how severe the competing conventions problem does affect the genetic algorithm's search abilities. Theoretically, the problem scales exponentially ($n!2^n$) with the number of hidden units. Hancock [24, 26] suggests that it may be less of a problem than one might suppose. Most probably, the effect will depend on parameters like network size and population size, and it may be reasonable to avoid it if possible.

The simplest way to avoid the problem is to not use the crossover operator at all [4, 9, 12]. Using smaller population sizes and more aggressive selection and mutation reduces the risk of exploring several competing conventions in parallel. Braun and Weisbrod [11] tried to prevent permuted internal representations by making long connections less probable than short connections and thus preferring for each functional mapping the structural mapping with the shortest connection lengths. Radcliffe [57] suggested a matching recombination operator based on the pattern of connections of the hidden units. This and some similar recombination operators are critically compared in [24] (see also [72] for some experiments). Montana and Davis [52] matched hidden units before crossover by their responses to a number of trial inputs applied to the net. In Munro [53], the crossover operator exchanges hidden units that have similar but nonidentical response patterns to the training set. Alba [3] used the Hamming distance of two individuals to prevent crossover between two very different individuals. Thierens et al. [69] argue that the position of the hyper plane defined by a hidden neuron is predominantly determined by the weight signs. Therefore, prior to crossover, they reorder the genetic string of the parents such that the genes of hidden neurons with a similar amount of positive and negative weights are at the same position in the gene string. Also, to break the weight sign symmetry, they simply flip the signs of the incoming and outgoing weights of a hidden neuron whenever the neuron has less positive than negative weights.

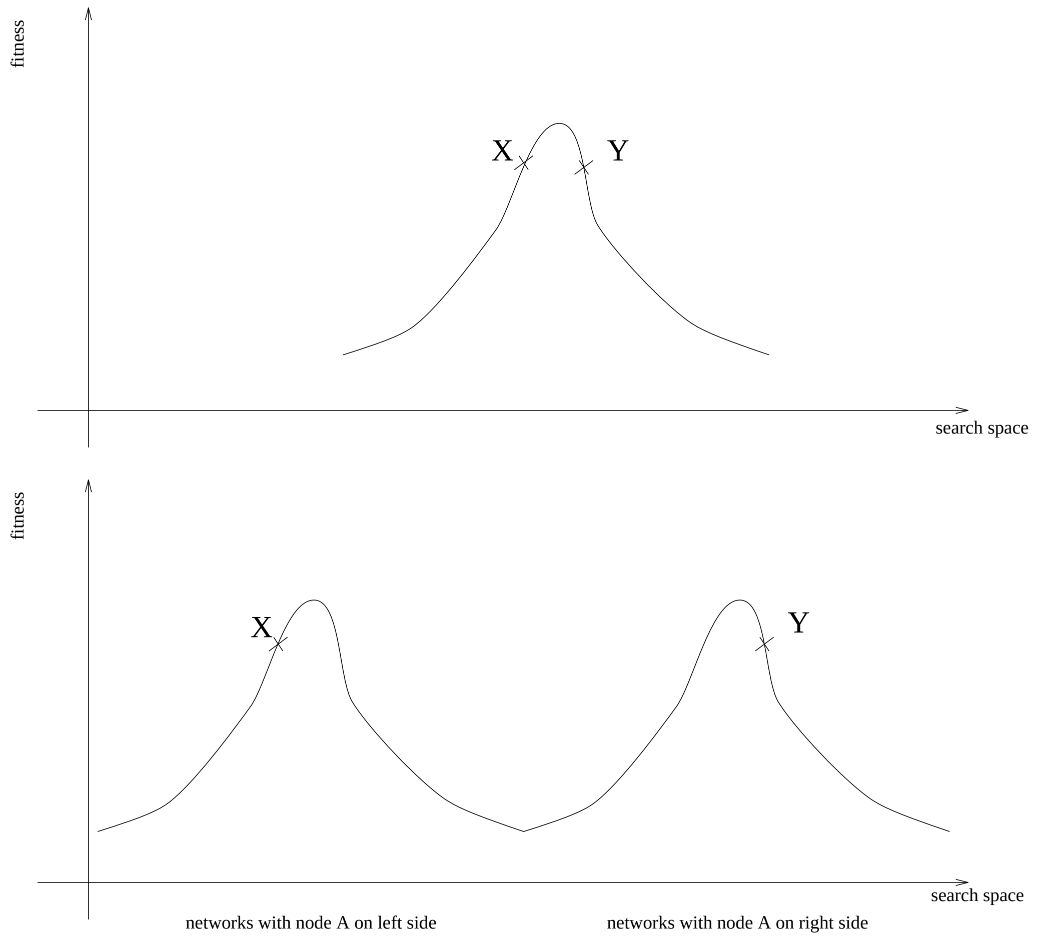

Figure 5: Fitness landscape with (bottom) and without (top) competing conventions.

Using genetic algorithms to set the weights in the cascade correlation algorithm automatically solves the permutation problem because the hidden units are trained and added one at a time [35, 56].

## 7 Conclusion

An overview on the important aspects when using evolutionary algorithms to neural network design and training has been presented. A number of different approaches has been reported, but a qualitative comparison has been omitted: there is not enough reliable experimental data to make definite statements on which method is preferable.

Probably, the most promising areas are those of topology optimization and those of finding the weights if no gradient information is available, simply because no other reliable method exists to solve those problems.

Especially for topology optimization, large computing power is needed. Thus, parallel implementations will become more and more important as the networks grow from toy problems to real world applications.

Altogether, work in the cross-section of evolutionary algorithms and neural networks bears great opportunities - but needs to become better founded and comparable.

## References

[1] Y. S. Abu-Mostafa. The vapnik-chervonenkis dimension: Information versus complexity in learning. *Neural Computation*, 1:312–317, 1989.

[2] J. T. Alander. An indexed bibliography of neural networks and genetic algorithms: Years 1985-1994. Available via ftp at ftp.uwasa.fi: cs/report94-1/gaNNbib.ps.Z, 1995.

[3] E. Alba, F. Aldana, and J. M. Troya. Full automatic ANN design: a genetic approach. In J. Mira, J. Cabentany, and A. Prieto, editors, *Proceedings of the International Workshop on Artificial Neural Networks*, pages 399–404. Springer Verlag, June 1993.

[4] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, January 1994.

[5] P. Arena, R. Caponetto, L. Fortuna, and M. G. Xibilia. Genetic algorithms to select optimal neural network topology. In *Proceedings of the 35th Midwest Conference on Circuits and Systems*, pages 1381–1383, 1992.

[6] P. Arena, R. Caponetto, L. Fortuna, and M. G. Xibilia. M. l. p. optimal topology via genetic algorithms. In *Proceedings of the Conference on Artificial Neural Nets and Genetic Algorithms*, pages 670–674. Springer Verlag, 1993.

[7] K. Balakrishnan and V. Honavar. Evolutionary design of neural architectures - preliminary taxonomy and guide to literature. Technical Report CS TR #95-01, Artificial Intelligence Group, Iowa State University, January 1995.

[8] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. Technical Report CS90-174, Computer Science and Engineering Department, UCSD (La Jolla), 1989.

[9] S. Bornholdt and D. Graudenz. General asymmetric neural networks and structure design by genetic algorithms. *Neural Networks*, 5:327–334, 1992.

[10] J. Branke, U. Kohlmorgen, and H. Schmeck. A distributed genetic algorithm improving the generalization behavior of neural networks. In Nada Lavrac and Stefan Wrobel, editors, *Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning*, Lecture Notes in Artificial Intelligence 914, pages 107–121. Springer Verlag, 1995.

[11] H. Braun and J. Weisbrod. Evolving neural feedforward networks. In *Proceedings of the Conference on Artificial Neural Nets and Genetic Algorithms*, pages 25–32. Springer Verlag, 1993.

[12] H. Braun and P. Zagorski. Enzo-ii - a powerful design tool to evolve multilayer feed forward networks. In *Proceedings of the first IEEE conference on evolutionary computation*, volume 2, pages 278–283, June 1994.

[13] T. P. Caudell and C.P. Dolan. Parametric connectivity: Training of constrained networks using genetic algotithms. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 370–374, Arlinghton, 1989.

[14] D. Dasgupta and D. R. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms on Neural Networks*, pages 87–96, 1992.

[15] H. de Garis. Building nanobrains with genetically programmed neural network modules. In *Proceedings of the International Joint Conference on Neural Networks*, pages 511–516, 1990.

[16] N. Dodd. Optimisation of network structure using genetic techniques. In *Proceedings of the International Joint Conference on Neural Networks*, pages 965–970, 1990.

[17] N. Dodd, D. Macfarlane, and C. Marland. Optimisation of artificial neural network structure using genetic techniques implemented on multiple transputers. In *Proceedings of the Conference on Transputing*, pages 687–700, 1991.

[18] S. E. Fahlmann. The cascade correlation learning architecture. Technical report, 1988.

[19] D. B. Fogel. On the philosophical differences between evolutionary algorithms and genetic algorithms. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, 1993.

[20] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transaction on Neural Networks*, 5(1):3–14, January 1994.

[21] F. Gruau. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In Whitley and Schaffer, editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 55–74, June 1992.

[22] A. Hämäläinen. Using genetic algorithms in self-organizing map design. In *to appear in: Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, April 1995.

[23] P. J. B. Hancock. *Coding Strategies for Genetic Algorithms and Neural Nets.* PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1992.

[24] P. J. B. Hancock. Genetic algorithms and premutation problems: a comparison of recombination operators for neural net structure specification. In *Proceedings of the IEEE Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 108–122, 1992.

[25] P. J. B. Hancock. Pruning neural nets by genetic algorithms. In Aleksander, I. and Taylor, J.G., editors, *Proceedings of the International Conference on Artificial Neural Networks*, pages 991–994. Elsevier, 1992.

[26] P. J. B. Hancock. Recombination operators for the design of neural nets by genetic algorithm. In Maenner, R., and Manderick, B., editors, *Proceedings of the Conference on Parallel Problem Solving from Nature*, volume 2, pages 441–451. Elsevier Science Publishers B. V., 1992.

[27] B. L. M. Happel and J. M. J. Murre. Design and evolution of modular neural network architectures. *Neural Networks*, 7(6/7):985–1004, 1994.

[28] S. A. Harp, T. Samad, and Guha A. Towards the genetic synthesis of neural networks. In J. D. Schaffer, editor, *Procedings of the 3rd International Conference on Genetic Algorithms*, pages 360–369, 1989.

[29] S. A. Harp, T. Samad, and A. Guha. Designing application-specific neural networks using genetic algorithm. In *Advances in Neural Information Processing Systems II.* Touretzky, D. S., 1989.

[30] J. Heistermann. Different learning algorithms for neural networks - a comparative study. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature, Workshop Proceedings*, pages 368–396. Springer Verlag, 1994.

[31] J. Hertz, A. Kogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[32] K. J. Hintz and J. J. Spofford. Evolving a neural network. In *Proceedings of the 5th IEEE International Symposium on Intelligent Control*, pages 479–484, Philadelphia, PA, September 1990.

[33] F. Hoffmeister and T. Bäck. Genetic algorithms and evolution strategies - similarities and differences. In H.-P. Schwefel and R. Maenner, editors, *Parallel Problem Solving from Nature - Proceedings of the 1st Workshop PPSN*, pages 447–461. Springer, October 1991.

[34] D. J. Janson and J. F. Frenzel. Training product unit neural networks with genetic algorithms, 1993.

[35] N. Karunanithi, R. Das, and D. Whithley. Genetic cascade learning for neural networks. In Schaffer and Whitley, editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 134–144, 1992.

[36] G. D. Kendall and H. J. Hall. Improving generalisation with Ockham's networks: Minimum description length networks. In *Proceedings of the 3rd International Conference on Artificial Neural Networks*, May 1993.

[37] G. D. Kendall and T. J. Hall. Optimal network construction by minimum descreption length. *Neural Computation*, 1993.

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

[39] H. Kitano. Empirical studies on the speed of convergence of neural network training using genetic algorithms. In *Proceedings AAAI*, pages 789–795, 1990.

[40] H. Kitano. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physika D*, (75):225–238, 1994.

[41] U. Kohlmorgen, H. B. Penfold, and H. Schmeck. Deriving application-specific neural nets with a massively parallel genetic algorithm. In *this volume.*

[42] J. R. Koza. *Genetic Programming.* MIT Press, 1991.

[43] J. R. Koza and J. P. Rice. Genetic generation of both the weights and architecture for a neural network, 1991.

[44] M. Mandischer. Representation and evolution of neural networks. In *Proceedings of the Conference on Artificial Neural Nets and Genetic Algorithms*, pages 643–649. Springer Verlag, 1993.

[45] V Maniezzo. Searching among search spaces: hastening the genetic evolution of feedforward networks. In *Proceedings of the Conference on Artificial Neural Nets and Genetic Algorithms*, pages 635–642. Springer Verlag, 1993.

[46] V. Maniezzo. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on Neural Networks*, 5(1), January 1994.

[47] S. J. Marshall and R. F. Harrison. Optimization and training of feedforward neural networks by genetic algorithms. In *Proceedings of the 2nd International Conference on Artificial Neural Networks*, pages 39–43, 1991.

[48] L. Marti. Genetically generated neural networks II; searching for an optimal representation. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 221–226, 1992.

[49] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[50] J. J. Merelo, M. Paton, A. Canas, A. Prieto, and F. Moran. Optimization of a competitive learning neural network by genetic algorithms. In J. Mira, J. Cabestany, and A. Prieto, editors, *Proceedings of the International Work shop on Artifical Neural Networks*, pages 185–192. Springer-Verlag, June 1993.

[51] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In J. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 379–384, Arlington, 1989.

[52] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Inteligence*, pages 762–767, 1989.

[53] P. W. Munro. Genetic search for optimal representations in neural networks, 1991.

[54] Z. Obradovic and R. Srikumar. Evolutionary design of application tailored neural networks. In *Proceedings of the first IEEE conference on evolutionary computation*, volume 1, pages 284–289, June 1994.

[55] S. Oliker and M. Furst. A distributed genetic algorithm for neural network design and training. *Complex systems*, 6:459–477, 1992.

[56] M. A. Potter. A genetic cascade-correlation learning algorithm. In Schaffer and Whitley, editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 123–133, 1992.

[57] N. Radcliffe. Genetic set recombination and its application to neural network topology optimization. Technical Report EPCC-TR-91-21, Edinburgh Parallel Computing Centre, University of Edinburgh, 1991.

[58] R. Reed. Pruning algorithms - a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, September 1993.

[59] J. Rissanen. Modelling by shortest data description. *Automatica*, (14):465–471, 1978.

[60] P. Robbins, A. Soper, and K. Rennolls. Use of genetic algorithms for optimal topology determination in back propagation neural networks. In *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 726–730. Springer Verlag, 1993.

[61] M. Rudnick. A bibliography of the intersection of genetic search and artificial neural networks. Technical report, Oregon Graduate Institute, Department of Computer Science and Engineering, January 1990.

[62] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, (323):533–536, 1986.

[63] S. Saha and J. P. Christensen. Genetic design of sparse feedforward neural networks. *Information Sciences*, (79):191–200, 1994.

[64] J. Santos and R. J. Duro. Evolutionary generation and training of recurrent artificial neural networks. In *Proceedings of the first IEEE conference on Evolutionary Computation*, volume 2, pages 759–763, June 1994.

[65] 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 Whitley and Schaffer, editors, *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, June 1992.

[66] W. Schiffmann, M. Joost, and R. Werner. Performance evaluation of evolutionarily created neural network topologies. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 274–283. Springer Verlag, October 1990.

[67] F. J. Smieja. Neural network constructive algorithms. trading generalization for learning efficiency? Arbeitspapiere der GMD 636, GMD, 1992.

[68] P. Spiessens and J. Torreele. Massively parallel evolution of recurrent networks: An approach to temporal processing. In Varela, F., and Bourgine, P., editors, *Proceedings of the first Conference on Artificial Life*, pages 70–77, 1991.

[69] D. Thierens, J. Suykens, J. Vandewalle, and B. De Moor. Genetic weight optimization of a feedforward neural network controller. In *Proceedings of the Conference on Artificial Neural Nets and Genetic Algotithms*, pages 658–663. Springer Verlag, 1993.

[70] J. Torreele. Temporal processing with recurrent networks: An evolutionary approach. In Belew, R.K. and Booker, L. B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 555–561. Morgan Kaufmann, San Diego, CA, USA, 1991.

[71] U. Utrecht and K. Trint. Mutation operators for structure evolution of neural networks. In R. Maenner Y. Davidor, H.-P. Schwefel, editor, *Parallel Problem Solving from Nature, Workshop-Proceedings*, pages 492–501. Springer, 1994.

[72] E. van Wanrooij. Evolving sequential neural networks for time series forecasting. Master's thesis, Department of Computer Science, University of Utrecht, Netherlands, November 1994.

[73] H. M. Voigt, J. Born, and I. Santibanez-Koref. Evolutionary structuring of artificial neural networks. Technical report, Technical University Berlin, Bio- and Neuroinformatics Research Group, 1993.

[74] G. Weiss. Neural networks and evolutionary computation. part i; hybrid approaches in artificial intelligence. In *Proceedings of the first IEEE Conference on Evolutionary Computation*, pages 268–277, June 1994.

[75] D. Whitley and C. Bogart. The evolution of connectivity: Pruning neural networks using genetic algorithms, 1990.

[76] D. Whitley and T. Hanson. Optimizing neural networks using faster, more accurate genetic search. In J. D. Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 391–395, 1989.

[77] D. Whitley and T. Starkweather. Optimizing small neural networks using a distributed genetic algorithm, 1990.

[78] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, (14):347–361, 1990.

[79] S. W. Wilson. Perceptron redux: Emergence of structure. *Physica D*, (42):249–256, 1990.

[80] F. Wong. Genetically optimized neural networks. Technical report, NIBS Pte Ltd., Singapore, 1994.

[81] B. Yoon, D. J. Holmes, G. Langholz, and A. Kandel. Efficient genetic algorithms for training layered feedforward neural networks. *Information Sciences*, 76:67–85, 1994.

[82] B.-T. Zhang. Effects of occam's razor in evolving sigma-pi neural nets. In R. Maenner Y. Davidor, H.-P. Schwefel, editor, *Parallel Problem Solving from Nature, Workshop Proceedings*, pages 462–471. Springer Verlag, 1994.