Genetic Programming of Minimal Neural Nets Using Occam's Razor[*]

Byoung-Tak Zhang **Heinz Mühlenbein**
Artificial Intelligence Research Division
German National Research Center for Computer Science (GMD)
Schloss Birlinghoven, D-5205 Sankt Augustin 1, Germany
e-mail: zhang@gmd.de, muehlen@gmd.de

Abstract

A genetic programming method is investigated for optimizing both the architecture and the connection weights of multilayer feedforward neural networks. The genotype of each network is represented as a tree whose depth and width are dynamically adapted to the particular application by specifically defined genetic operators. The weights are trained by a next-ascent hillclimbing search. A new fitness function is proposed that quantifies the principle of Occam's razor. It makes an optimal trade-off between the error fitting ability and the parsimony of the network. We discuss the results for two problems of differing complexity and study the convergence and scaling properties of the algorithm.

1 INTRODUCTION

Optimization of neural network architectures for particular applications is important because the speed and accuracy of learning and performance are dependent on the network complexity, i.e. the type and number of units and connections, and the connectivity of units. For example, a network having a large number of adjustable connections tends to converge fast, but it usually leads to overfitting of the training data. On the other hand, a small network will achieve a good generalization if it converges, it needs, however, generally a large amount of training time (Abu-Mostafa 1989). Therefore, the size of the network should be as small as possible, but sufficiently large to ensure a fast convergence to the training set.

Genetic algorithms have been used to design network architectures. The network design problem can be regarded as a search for an architecture which fits best to the specified task according to some explicit fitness criteria. Mühlenbein and Kindermann (1989) discuss general problems for evolving genetic neural networks. Harp *et al.* (1989) and Miller *et al.* (1989) suggest representation schemes in which the anatomical properties of the network structure are encoded as bit-strings. Similar representation has also been used by Whitley *et al.* (1990) to prune unnecessary connections. Kitano (1990) and Gruau (1992) describe encoding schemes in which a network configuration is indirectly specified by a graph generation grammar which evolves by genetic algorithms. All these methods use the backpropagation algorithm (Rumelhart *et al.* 1986) to train the weights of the network. Koza (1990) provides an alternative approach to representing neural networks, under the framework of so-called genetic programming, which enables modification not only of the weights but also of the architecture for a neural network. However, this method does not find a network of minimal complexity.

In this paper we describe a new genetic programming method for constructing minimal neural networks. Using an Occam's razor in the fitness function, the method prefers a simple network architecture to a complex one. The weights are trained not by backpropagation, but by a next-ascent hillclimbing search. The breeder genetic algorithm BGA (Mühlenbein *et al.* 1993) is used for evolving optimal networks.

The paper is organized as follows. In Section 2, the fitness function for the genetic search of minimal complexity solutions is derived. The representation scheme and genetic operators as well as the control algorithm for adapting the architectures and the weights are described in Section 3. Section 4 shows the experimental results, which is followed by an analysis of fitness landscapes in Section 5, and discussions in Section 6.

[*]Proc. of the Fifth Int. Conf. on Genetic Algorithms (ICGA-93), Morgan Kaufmann, 1993, pp. 342–349.

2 QUANTIFYING OCCAM'S RAZOR

Occam's razor states that unnecessarily complex models should not be preferred to simpler ones. This section gives a quantitative Occam's razor for constructing minimal complexity neural networks by genetic algorithms.

In defining minimality, it is important that the network be able to approximate at least the training set to a specified performance level. A small network should be preferred to a large network *only if* both of them achieve a comparable performance. Otherwise, the algorithm would not reduce the approximation error, preferring smaller networks which can not be powerful enough to solve the task. So the first term of the fitness function of an individual network should be the error function. The error function commonly used for the data set $D = \{(x_i, y_i) \mid i = 1, ..., N\}$ of N examples is the sum of squared errors between the desired and actual outputs:

$$E(D|W,A) = \sum_{i=1}^{N} E(y_i|x_i, W, A) \qquad (1)$$

with

$$E(y_i|x_i, W, A) = \sum_{j=1}^{m} (y_{ij} - o_j(x_i; W, A))^2 . \qquad (2)$$

Here y_{ij} denotes the jth component of the ith desired output vector y_i, and $o_j(x_i; W, A)$ denotes the j-th actual output of the network with the architecture A and the set of weights W for the i-th training input vector x_i.

The complexity of a neural network architecture is dependent on the task to be learned and can be defined in various ways, depending on the application. In general the number of free parameters (or adjustable weights) of the network should be minimal, since this is one of the most important factors determining the speed and accuracy of the learning. Additionally, large weights should in general be penalized (regularization), in the hope of achieving a smoother or simpler mapping (Poggio and Girosi 1990; MacKay 1992). We define the complexity, C, of a network as

$$C(W|A) = \sum_{k=1}^{K} w_k^2 \qquad (3)$$

where K is the number of free parameters. Notice that K can be arbitrarily large, because we fit the architectures too. In the case of binary weights, C reduces to the number of synaptic connections. This complexity measure might be extended by additional cost terms, such as the number of layers when the application requires a fast execution of the trained network.

The combined fitness function which we try to *minimize* is defined as

$$F(D|W,A) = \alpha C(W|A) + \beta E(D|W,A) \qquad (4)$$

where α and β are constants for the trade-off between error fitting and complexity reduction. This fitness function has an elegant probabilistic interpretation for the learning process: according to the Bayesian framework, minimizing F is identical to finding the most probable network with architecture A and weights W (Sorkin 1983; Tishby *et al.* 1989).

To see this, let us define the following. Let D be the data set for the function $\gamma : X \rightarrow Y$, i.e.

$$D = \{(x_i, y_i) \mid x_i \in X, \ y_i \in Y, \ y_i = \gamma(x_i), \ i = 1...N\}.$$

Then a model $\mathcal{M}$ of the function γ is an assignment to each possible pair (x, y) of a number $P(y|x)$ representing the hypothetical probability of y given x. That is, a network with specified architecture A and weights W is viewed as a model $\mathcal{M} = \{A, W\}$ predicting the outputs y_i as a function of input x_i in accordance with the probability distribution:

$$P(y_i|x_i, W, A) \quad = \quad \frac{\exp(-\beta E(y_i|x_i, W, A))}{Z(\beta)} \qquad (5)$$

where β is a positive constant which determines the sensitivity of the probability to the error value. $Z(\beta) = \int \exp(-\beta E(y_i|x_i, W, A))dy$ is a normalizing constant. Under the assumption of the Gaussian error model, i.e. if the true output is expected to include additive Gaussian noise with standard deviation σ, we have

$$P(y_i|x_i, W, A) \quad = \quad \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{E(y_i|x_i, W, A)}{2\sigma^2}\right) \qquad (6)$$

with $\beta = \frac{1}{2\sigma^2}$ and $Z(\beta) = \sqrt{2\pi}\sigma$.

A prior probability is assigned to alternative network model written in the form:

$$P(W|A) \quad = \quad \frac{\exp(-\alpha C(W|A))}{Z(\alpha)} \qquad (7)$$

where $Z(\alpha) = \int \exp(-\alpha C(W|A))d^K W$ is a measure of the characteristic network complexty. The posterior probability of the network model is then:

$$P(W|D, A) = \frac{\exp(-\alpha C(W|A) - \beta E(D|W, A))}{Z(\alpha, \beta)} \qquad (8)$$

with $Z(\alpha, \beta) = \int \exp(-\alpha C(W|A) - \beta E(D|W, A))d^K W$.

Now let $-I(\mathcal{M})$ be the log of the prior probability of the model $\mathcal{M}$, i.e.

$$I(\mathcal{M}) = \log P(W|A)^{-1}. \tag{9}$$

Let $-I(D|\mathcal{M})$ be the log probability of D according to $\mathcal{M}$:

$$I(D|\mathcal{M}) = \sum_{i=1}^{N} \log P(y_i|x_i, W, A)^{-1}. \tag{10}$$

Then the probability that both $\mathcal{M}$ is true and D occurs is

$$p(\mathcal{M}) = \exp(-I(D, \mathcal{M})) \tag{11}$$

where

$$I(D, \mathcal{M}) = I(\mathcal{M}) + I(D|\mathcal{M}). \tag{12}$$

As is well known this p results as the posterior probability of $\mathcal{M}$ and the model which maximizes $p(\mathcal{M})$ would be the best fit. For most real applications, $I(D, M)$ can not be computed exactly because the involved probabilities are not known. But it is easily seen that minimization of the fitness function (4) approximates maximization of $p(\mathcal{M})$ under the assumption (6).

3 GENETIC BREEDING OF MINIMAL NEURAL NETS

3.1 BREEDER GENETIC ALGORITHM

For the evolution of minimal neural networks we use the breeder genetic algorithm BGA of Mühlenbein *et al.* (1993). In contrast to the usual GA's model of natural evolution, the BGA models rational selection performed by human breeders. The BGA maintains a population $\mathcal{P}$ consisting of M individuals of neural networks. Each network of the initial population, $\mathcal{P}(0)$, is generated with a random number of layers. The receptive field of each neural unit and its width are also chosen randomly.

The t-th population, $\mathcal{P}(t)$, is created from $\mathcal{P}(t-1)$ in three steps: selection, hillclimbing, and recombination. In the selection step, τ% of the most fit individuals in $\mathcal{P}(t-1)$ are accepted into the mate set $\mathcal{S}$. Then each individual in $\mathcal{S}$ undergoes a hillclimbing search where the weights of the network are adapted by mutation. This results in the revised mate set $\mathcal{S}'$. The recombination phase repeatedly selects two random parent individuals in $\mathcal{S}'$ to mate and generate two offspring, until the population size amounts to M.

A new population is generated repeatedly until an acceptable solution is found or the variance falls below a specified limit value V_{min}, i.e.

$$\frac{1}{M} \sum_{i=1}^{M} \left(F(i) - \bar{F}\right)^2 \leq V_{min} \tag{13}$$

where $\bar{F}$ is the average fitness of the individuals in $\mathcal{P}(t)$. If a solution is found, the algorithm stops. If a solution is not found but the population has converged to a local minimum, then the algorithm starts again by initializing a new population.

3.2 REPRESENTATION

For the experiments we have used McCulloch-Pitts neurons. The McCulloch-Pitts neuron is a binary device, i.e. it can be in only one of two possible states. Each neuron has a threshold. The neuron can receive inputs from excitatory and/or from inhibitory synapses. The neuron becomes active if the sum of weighted inputs exceeds its threshold. If it does not, the neuron is inactive. Formally, the neurons used in this work has the threshold activation function:

$$y_j = \begin{cases} 1 & \text{if } \sum_i w_{ji} x_i > \theta_j \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where w_{ji} is the connection weight from unit i to unit j and θ_j denotes the threshold value for unit j. Despite their simplicity, McCulloch-Pitts neurons are very powerful. In fact, it can be shown that any finite logical expression can be realized by them (McCulloch and Pitts 1943).

Figure 1 describes the grammar for generating a feedforward network of n inputs and m outputs. A network is represented as a set of m trees, each corresponding to one output unit. In the grammar, the nonterminal symbol Y is used to represent a neural unit having a threshold of θ and r weights. The integer r indicates the receptive field width of the unit. Each connection weight is represented as a nonterminal node W consisting of a symbol 'W', a weight value w, followed by a nonterminal symbol indicating recursively another neural unit Y or an external input unit X. An external input is described by a symbol 'X' followed by an integer i denoting the index of the input unit.

In the first experiments we used binary thresholds. McCulloch-Pitts neurons allow integer thresholds. Networks with binary thresholds can realize networks with integer thresholds by using additional neurons. Similarly, integer weights can also be realized by neurons using binary weights. The number of weights and units is usually reduced if the genotype is transformed into a network of integer values. This is illustrated in

Figure 3: Crossover operation

Unlike the mutation, the crossover operator adapts the size and shape of the network architecture. A crossover operation starts by choosing two parent individuals which are chosen randomly from the mate set. Actual crossover of two individuals, i and j, is done on their genotypical representations s_i and s_j. The nodes in the tree are numbered according to the depth-first search order and crossover sites c_i and c_j are chosen at random with the following conditions:

$$1 \le c_i \le Size(s_i) \quad \text{and} \quad 1 \le c_j \le Size(s_j).$$

Here, the length of an individual, $Size(s_i)$, is defined as the total number of units and weights.

Given the crossover points, the subtrees of two parent individuals, s_i and s_j, are exchanged to form two offspring s_i' and s_j' (Figure 3). The label of the nodes c_i and c_j must belong to the same class, i.e. either both Y-type or both W-type nodes. The number of arguments of each operator plays no role because the syntactically correct subtree under the node c_i and c_j is completely replaced by another syntactically correct expression.

4 EXPERIMENTAL RESULTS

The convergence and scaling properties of the method were studied on two classes of problems with different difficulty: majority and parity. The majority function of n inputs (n odd) returns a 1 if more than half of

Figure 4: Solutions for 4-parity problem (a) minimal (b) discovered

growth and pruning is repeated to fit errors on one hand and to minimize the complexity of the network on the other hand. The corresponding evolution of the fitness values of the best individuals in each generation is depicted in Figure 6. It is interesting to notice that the global behavior of this optimization method is comparable with the group method of data handling (GMDH) in which additional terms are incrementally added to the existing polynomial approximator to achieve a minimal description length model of a complex system (Ivakhnenko 1971).

In general, the results are encouraging. For large size problems of some class, however, the convergence was very slow. A simple optimization method does not exist which performs better than any other optimization method for a reasonable large class of binary functions of size n. To be powerful, every sophisticated optimization method has to be tuned to the application (Mühlenbein 1993). In order to speed up the genetic search, an analysis of the fitness landscape has to be made.

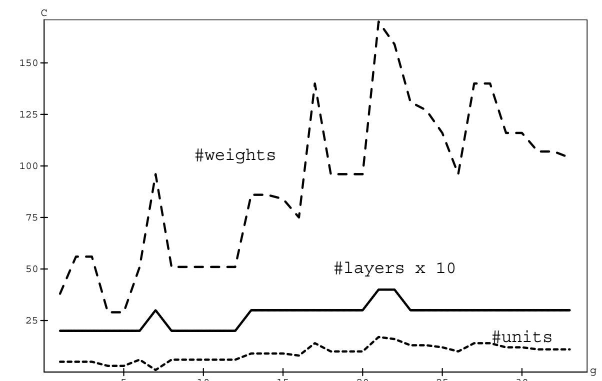

Figure 5: The evolution of network complexity in terms of the number of weights, layers, and units for the best individual in each generation. Growth and pruning is repeated to find an optimal complexity which is parsimonious but large enough to solve the problem.

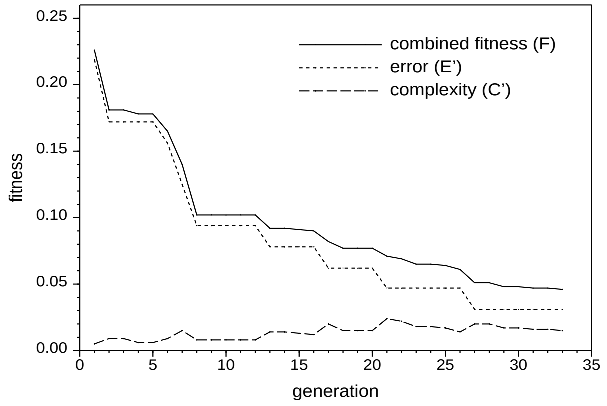

Figure 6: The evolution of the network fitness F decomposed into the normalized error E' and the extended complexity C'. In spite of a fixed Occam factor, the relative importance of the complexity term increases as evolution proceeds.

5 ANALYSIS OF FITNESS LANDSCAPES

The number of local optima, their distribution and the basins of attraction are some of the variables necessary to describe a fitness landscape. For the evaluation of search strategies more specific questions have to be answered:

- How do local optima vary with the fitness?
- How many local optima are there with respect to 1-mutant moves?

These questions have been studied on two problems: XOR and OR function of two inputs. For each problem we analysed two search spaces of different dimension. One was a feedforward network of 2-2-1 architecture which has 9 free parameters (6 binary weights plus 3 binary thresholds). The other search space was a 2-3-1 architecture having 13 free parameters (9 binary weights plus 4 binary thresholds). The 2-2-1 architecture is known as minimal for solving the XOR problem, while the minimal architecture for the OR problem is a 2-1 architecture (no hidden units). So the OR network has an excessive degree of freedom. In describing the landscapes, we should focus on the statistical characteristics of them because the spaces are too large to list all the details. For the analysis, the fitness function consisted of the error term only; the coefficient α in (4) was set to zero.

The fitness distributions are shown in Table 2. Notice that each of the XOR and OR networks has two binary inputs, resulting in four input-output pairs. Hence a specific network can have only one of five fitness values (0.0 in case of all four examples are classified correctly, 0.25 if one example is classified incorrectly, and so on.), ignoring the complexity term. The analysis shows that the XOR-9 network has only two (0.4%) isolated global optima, while the OR-9 net has fifteen (2.9%) optima. Growth of the dimension from 9 to 13 increases the proportion of optima of XOR by 0.2%, but reduced that of OR by 0.2%. The table shows also that the fitness of OR-9 is more uniformly distributed than that of XOR-9, suggesting that a search step in the OR network space would get more information than a step in the XOR space.

Table 2: Fitness distribution

	XOR		OR	
	$d = 9$	$d = 13$	$d = 9$	$d = 13$
$F(i) = 0.00$	0.004	0.006	0.029	0.027
$F(i) = 0.25$	0.125	0.121	0.301	0.281
$F(i) = 0.50$	0.805	0.767	0.140	0.165
$F(i) = 0.75$	0.002	0.097	0.512	0.501
$F(i) = 1.00$	0.004	0.009	0.018	0.026

To see how the local optima vary, we computed the probability of an individual i finding a better, same, and worse fit neighbor n by a *single* mutation, respectively (Table 3). Here, a better fit neighbor n of i means $F(n)$ is smaller than $F(i)$, since we attempt to minimize the fitness function. The table shows, for instance, that for XOR-9 the probability of finding a

Table 3: Fitness distribution of neighbors

	XOR		OR	
	$d = 9$	$d = 13$	$d = 9$	$d = 13$
$P[F(n) < 0.00]$	0.000	0.000	0.000	0.000
$P[F(n) = 0.00]$	0.000	0.192	0.296	0.380
$P[F(n) > 0.00]$	1.000	0.808	0.704	0.620
$P[F(n) < 0.25]$	0.024	0.028	0.035	0.034
$P[F(n) = 0.25]$	0.431	0.488	0.290	0.718
$P[F(n) > 0.25]$	0.545	0.484	0.350	0.248
$P[F(n) < 0.50]$	0.084	0.075	0.360	0.282
$P[F(n) = 0.50]$	0.860	0.853	0.290	0.402
$P[F(n) > 0.50]$	0.056	0.072	0.350	0.316
$P[F(n) < 0.75]$	0.708	0.559	0.170	0.155
$P[F(n) = 0.75]$	0.264	0.397	0.812	0.822
$P[F(n) > 0.75]$	0.028	0.044	0.018	0.023
$P[F(n) < 1.00]$	1.000	0.820	0.802	0.650
$P[F(n) = 1.00]$	0.000	0.180	0.198	0.350
$P[F(n) > 1.00]$	0.000	0.000	0.000	0.000

better neighbor is only 8.4% if the fitness of the individual is 0.5. For OR, the corresponding probability is 36.0%. It can also be seen in both tables that the increase of the dimensionality of the search space from 9 to 13 leads to a change in the fitness distributions and landscapes, meaning the modification of network architecture can make it easier to train the weights.

We also computed the probability of a configuration finding a better fit neighbor by steepest-descent hillclimbing, i.e. by looking at *all* its neighbors at Hamming distance 1. Not surprisingly for this kind of landscape, one has for XOR a less than 50% chance of finding a better configuration. For OR, the probability is about 70%. This means steepest-descent hillclimbing would be effective for OR, but not for XOR. This explains in part why our experiments showed a good scaling property for the majority function (a kind of OR) in comparison to the parity problem (whose smallest size is XOR).

6 FUTURE WORK

We have presented an evolutionary method for optimizing both the network architecture and the weights at the same time. The method uses trees to represent a feedforward network whose size and topology are dynamically adapted by genetic operators. A new fitness function has been proposed which proved to work well in combination with the breeder genetic algorithm. Experimental results have shown that, given enough resources, the method finds minimal complexity networks with respect to the representation scheme used. As opposed to conventional learning algorithms for neural networks, the genetic programming method makes relative few assumptions on the structure of the search space. Thus, the same method described above can also be used to breed networks of radial basis functions, sigma-pi units, or any mixture of them, instead of the threshold or sigmoid units. The potential for evolving neural architectures that are customized for specific applications is one of the most interesting properties of genetic algorithms. The present work can be extended in three directions.

First, the information about the fitness landscape can be used to speed up convergence. As was shown, the fitness landscapes are characterized by large plateaus. The basin of attraction of the global optimum is fairly small. We have also seen that the fitness landscapes are changed by modifying the architectures. It is expected that fitness landscapes will generally have large plateaus as the network complexity approaches to a minimum, which makes it difficult for a hillclimber to reach the minimum. A possible method of accelerating the convergence speed would be to start with larger networks (than are supposed to be minimal) and to let the network be pruned by the Occam factor.

A second improvement involves the encoding of neural nets in chromosomes. Although the current strong or direct representation scheme worked well for many problems, a scaling problem was observed in cases that the problem requires a large network or the magnitude of weights grows large. An alternative representation scheme might be a weak or indirect mapping. For example, Gruau (1992) describes a method that uses a graph grammar for generating connection matrices of networks. This reduces the chromosome size and so it can find more regular connectivity patterns relative efficiently. However, they necessarily involve severe constraints on the network search space and cannot be useful for finding a minimal complexity network of arbitrary topology. In addition, in the weak specification scheme the genotype must be converted to the phenotype every time the weights are trained and/or the fitness of an individual is evaluated, what is not needed in the strong specification scheme. We are looking for a more compact representation scheme which exploits the advantages of both the direct and the indirect encoding.

A third future work concerns the study of other factors, for instance the effect of training set, on convergence speed and generalization performance of the algorithm. The genetic programming involves a time-consuming process of evaluating training examples. Although we have used in the experiments all possible examples to ensure a 100% accuracy, usual applications of neural networks do not require a perfect generalization, but a reasonable performance (e.g. 95%). In this case, the fitness evaluation time can be saved enormously, if we have an efficient method for selecting examples critical to specific tasks (Zhang and Veenker 1991a; Zhang 1992). The integration of active data selection to the genetic programming should improve the efficiency and scaling property of the method described above.

Although we have focused in this paper on the application aspect of genetic algorithms, neural net optimization provides a very interesting problem worthy of theoretical study from the genetic algorithm point of view. For example, the problem we discussed had to handle variable length of chromosomes by which the fitness landscape is modified during evolution. This kind of problem is contrasted with usual applications of genetic algorithms in which the search space is fixed.

Acknowledgements

The authors thank Jörg Kindermann, Frank Śmieja, Dirk Schlierkamp-Voosen, and the other members of the learning systems research group of the GMD for useful comments on the draft versions of this paper. This work was supported by the Real World Computing Programme under the project SIFOGA.

References

Y. S. Abu-Mostafa (1989). The Vapnik-Chervonenkis dimension: information versus complexity in learning. *Neural Computation*, **1**(3):312–317.

F. Gruau (1992). Genetic synthesis of boolean neural networks with a cell rewriting developmental process. Tech. Rep., Laboratoire de l'Informatique du Parallélisme.

S. A. Harp, T. Samad, and A. Guha (1989). Towards the genetic synthesis of neural networks, *Proc. ICGA-89*, 360–369. Morgan Kaufmann.

A. G. Ivakhnenko (1971). Polynomial theory of complex systems. *IEEE Trans. Sys. Man and Cybern.*, SMC-**1**(4):364–378.

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

J. R. Koza (1990). *Genetic Programming: A paradigm for genetically breeding populations of computer programs to solve problems.* Tech. Rep. STAN-CS-90-1314, Dept. of Computer Science, Stanford Univ., CA.

D. J. C. MacKay (1992). *Bayesian methods for adaptive models.* Ph.D. thesis, Caltech, Pasadena, CA.

W. S. McCulloch, and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophysics*, **5**:115–133.

G. F. Miller, P. M. Todd, and S. U. Hegde (1989). Designing neural networks using genetic algorithms. *Proc. ICGA-89*, 379–384. Morgan Kaufmann.

H. Mühlenbein (1993). Evolutionary algorithms: theory and applications. To appear in E. H. L. Aarts, and J. K. Lenstra (eds.), *Local Search in Combinatorial Optimization*, Wiley.

H. Mühlenbein, and J. Kindermann (1989). The dynamics of evolution and learning—Towards genetic neural networks. In R. Pfeifer *et al.* (eds.), *Connectionism in Perspective*, 173–197, North-Holland.

H. Mühlenbein, and D. Schlierkamp-Voosen (1993). Predictive models for the breeder genetic algorithm I: continuous parameter optimization. *Evolutionary Computation*, **1**(1).

T. Poggio, and F. Girosi (1990). Networks for approximation and learning. *Proc. IEEE*, **78**(9):1481–1497.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams (1986). Learning internal representations by error-propagation. In D. E. Rumelhart, and J. L. McClelland (eds.), *Parallel Distributed Processing*, Vol. I, 318–362. MIT Press.

R. Sorkin (1983). A quantitative Occam's razor. *Int. J. Theor. Phys.*, **22**(12):1091–1104.

N. Tishby, E. Levin, and S. A. Solla (1989). Consistent inference of probabilities in layered networks: predictions and generalization. *Proc. Int. Joint Conf. Neural Networks*, Vol. II, 403–409. IEEE.

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

B. T. Zhang (1992). *Learning by Genetic Neural Evolution.* (in German), Sankt Augustin, Infix-Verlag. Also available as Informatik Berichte No. 93, Institut für Informatik, Universität Bonn.

B. T. Zhang, and G. Veenker (1991a). Focused incremental learning for improved generalization with reduced training sets. In T. Kohonen *et al.* (eds.), *Artificial Neural Networks: Proc. ICANN-91*, Vol. I, 227–232. Elsevier.

B. T. Zhang, and G. Veenker (1991b). Neural networks that teach themselves through genetic discovery of novel examples. In *Proc. IJCNN-91*, Vol. I, 690–695. IEEE.