Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons

W. Schiffmann, M. Joost, R. Werner *

Abstract

In this paper we present a new approach for automatic topology optimization of backpropagation networks. It is based on a genetic algorithm. In contrast to other approaches it allows that two networks with different number of units can be crossed to a new valid "child" network. We applied this algorithm to a medical classification task, which is extremely difficult to solve. The results confirm, that optimization make sence, because the generated network outperform all fixed topologies.

1 Introduction

As Minsky and Papert [1] have shown the XOR–problem cannot be solved without a hidden layer. A learning rule which is able to train this kind of networks was developed by Rumelhart et al. [2]. It is known as *backpropagation* (BP) and it is one of the most often used neural network paradigms. Hornik [3] proved that every function can be approximated by a neural network with just *one* hidden layer. By adding "enough" hidden units the approximation error can be made as small as required. This can be compared with a look–up table which stores one output value/vector per hidden unit. But, this approach doesn't take into account that the network should be able to generalize. It just tries to fit the training data.

2 Automatic Topology Search

BP is working on a given network architecture which is made up by a specific partition of units over layers and by a particular connectivity pattern. Choosing an appropriate topology for a given problem depends on personal experience of the human designer. Almost always a fully interconnected architecture is used. But what is the optimal number of units und their organization into layers?

With the exception of some simple task, e.g. the XOR–problem, humans cannot foresee the optimal network topology. Thumb rules like *the harder the problem the more units you need* are of little practical use to the design problem. Manual network design is something of a black art [4] and it can be assumed that most of such network designs are not optimal. In [5] the impact of network topology on the speed and performance of BP trained networks was demonstrated. In order to adapt the network topology to the problem at hand we proposed an *automatic* design procedure which is based on *Genetic Algorithms* (GAs) (see also [6]).

Later on, we evaluated the performance and generalization behaviour of automatic generated network architectures with respect to character recognition tasks [7]. While our first approaches used just mutation to modify the network topology we are now able to cross two architectures as well.

2.1 Possible partitions

Suppose a wizard would tell us that h hidden units and l hidden layers are necessary for a given problem. Now, we have to distribute the hidden units into these layers. We will derive a recursive formula which allows to compute the number of possible partitions $p(h, l)$. It is evident that

$$p(h, l) = 1 \quad \text{if} \quad l = 1 \quad \text{or if} \quad h = l \tag{1}$$

If $h < l$ then $p(h, l) = 0$ because empty partitions are not reasonable. If $h > l$ we get

$$p(h, l) = p(h-1, l-1) + p(h-1, l) \tag{2}$$

By this formula we can compute the number of possible partitions from that number of a less complex

*University of Koblenz, Physics Department, Rheinau 3-4, D-5400 Koblenz, e-mail: schiff@infko.uni-koblenz.de. This work is supported by the *Deutsche Forschungsgemeinschaft* (DFG) as part of the project *FE–generator* (grant Schi 304/1–1)

$\frac{h}{l}$	10	20	30	40	50	60	70	80	90	100
1	1	1	1	1	1	1	1	1	1	1
2	9	19	29	39	49	59	69	79	89	99
3	36	171	406	741	1176	1711	2346	3081	3916	4851
4	84	969	3654	9139	18424	32509	52394	79079	113564	156849
5	126	3876	23751	82251	211876	455126	864501	1502501	2441626	3764376

Table 1. Number of possible partitions if h hidden units should be distributed over l (non empty) layers

architecture with $h-1$ units. The first term counts the number of partitions if the boundary of an additional layer seperates the additional unit itself. The second term means that the additional unit is placed into the last hidden layer of a less complex architecture which has already l layers. As Table 1 illustrates the number of possible partitions —even for moderate numbers of hidden units— increases to astronomical values.

2.2 Possible connections

If we have decided for a specific partition we are confronted with the problem of optimizing connectivity. For h hidden units the number of possible connections is limited by two extreme topologies which are fully interconnected from input layer (m units) to output layer (n units):

1. A topology that has as much as possible hidden layers with one hidden unit each. We will refer to that topology as TALL.
2. A topology that has just one hidden layer. It forms a look–up table and we will refer to it as WIDE.

While the TALL–architecture contains

$$C_T = \frac{h^2 - h}{2} + h \cdot (m+n) + m \cdot n \qquad (3)$$

connections, the WIDE–architecture has only

$$C_W = h \cdot (m+n) + m \cdot n \qquad (4)$$

connections. Table 2 compares these two topologies to each other for various values of hidden units. Additionally the fraction of C_W/C_T is given. It can be used as a measure of connectivity.

In most practical applications neither the TALL– nor the WIDE–architecture will be best suited. Thus, we have to find a connectivity pattern which is adapted to a particular task. Because we (humans) cannot comprehend the effects of modifications in topology, we have to provide methods for topology optimization which operate automatically.

h	C_T	C_W	C_W/C_T
1	87	87	1.000000
25	963	663	0.688474
50	2488	1263	0.507637
75	4638	1863	0.401682
100	7413	2463	0.332254
200	24763	4863	0.196382
300	52113	7263	0.139370

Table 2. Number of possible connections and minimum connectivity for the two extreme architectures (m=21, n=3, see also section *Simulation Results*)

3 Applications of GAs to Neural Networks

It should be noted that GAs can be applied to neural networks in two different ways:

1. Optimizing connection weights
2. Optimizing network topology

3.1 GAs for weight adjustment

In the first case the GA works at continuous parameters. Results concerned with this approach can be found in [8] – [16]. An exciting discussion of evolutionary training methods is provided by [17]. It can be summarized that the mentioned approaches differ mainly in three ways:

1. number representation
2. genetic operators used and
3. parent–offspring replacement strategies

If the number of connections is high it takes a lot of time to train a network by means of GAs. As simulations show, just less complex networks (<50 units) can be trained by this method. The speed of convergence can be increased if the rate of mutation is inverse to the diversity in the population of networks

[9]. GAs for weight adjustment implement a parallel search in weight space. Thus, they are able to find approximative solutions in short time. In contrast, GAs have trouble to get an exact solution. In order to solve this problem, one can combine GAs with BP. The starting weight vector for BP is determined by a preceding GA training run. See [18] for details of that approach.

3.2 GAs for topology optimization

Here, learning is done by BP or any other well known learning procedure. Because discrete parameters must be optimized the performance function is undifferentiable. Thus, gradient methods are not applicable. As we have seen the search space of all possible network architectures is vast and noisy. Because it depends on the random initial weights the performance of a specific architecture must be viewed as a random variable. It is also *deceptive* and *multimodal.* These features are concerned with the effects of small changes of the parameters and the objective function. *Deceptive* means that similar network architectures can have different performance. On the other hand, different network architectures show similar performance. Hence, the search space is *multimodal.*

As pointed out in [4] such complex spaces cannot be explored efficiently be enumerative, random or even heuristic knowledge–guided search methods. In contrast, the adaptive features of GAs (building blocks, step–width control by crossover) provide a more robust and faster search procedure. Additionally, it is easy to speed–up the genetic search by means of parallel processing.

Before we discuss the differences between several approaches we want to give a basic GA for topology optimization which is common to all these approaches. It is assumed that two representations of the networks are distinguished:

1. *genotypes* which are modified by the GA's operators (mutation, crossover)
2. *phenotypes* which are trained by a conventional learning procedure (e.g. BP) used for performance evaluation or selection

The basic GA comprises four steps:

1. Initialize a population of random starting architectures
2. Select, cross(over) and mutate these architectures
3. Train the networks for a given number of epochs
4. Stop if the desired performance is achieved; else proceed with step 2

3.3 Representation of genotypes

Roughly two basic representation schemes can be distinguished:

1. low–level genotypes
2. high–level genotypes

While the first one is transparent and easy to use, there are two variants of the second representation scheme. Low–level genotypes directly code the network topology. Each unit and each connection is specified separately. This "blueprint" approach is used by Miller et al. [4], Schiffmann et al. [7] and Dodd [19].

High–level genotypes are more complex coded representations of network architectures. They can be further divided into *parametric* and *recipe* genotypes. Examples for the use of parametric genotypes are found in [20] – [23]. Here, the networks are splited into modules of units which are specified by parameters and which are coupled by parametric connectivity patterns. Even through this representation is more compact and thus well suited to code large network architectures, it is difficult to choose the *relevant* parametric shapes. It should be noted that in using parametric genotypes the search space is confined to a specific subspace.

Similar considerations apply to the recipe genotype representation. Here, the architecture is specified by growth rules [24] and [25] or by sentences of a formal language [26]. The last approach is called *genetic neural networks.* Even though these approaches are mostly biological plausible, one has to commit to primitives used for the rules. Further, the choice of specific primitives hardly influences the application of genetic operators.

Usually neural network applications are limited to network sizes of about 1000 units. Often, research applications are confined to smaller architectures (e.g. XOR– or encoder–problem). Thus, in order to evaluate the properties of GAs for topology optimization the low–level representation is well suited and sufficient. Because it is easy and straightforward to apply genetic operators to the blueprints, we decided to use it to code our network architectures.

3.4 Genetic Operators

As pointed out in [17] the genetic operators must produce *correct* and *complete* offsprings. These requirements are easy to satisfy with blueprints. An efficient blueprint representation can be achieved by using a list where the connectivity of individual units is registered. In order to use this data structure together with BP, the list must be chained in both directions. For the sake of simplicity, this is not shown in Figure 1.

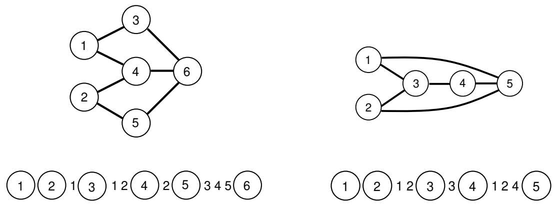

Fig. 1. Two network architectures and its corresponding blueprint representations

Blueprint coding works in two steps. Starting with the input layer, the units are numbered. Then, for each of the units the numbers of its preceding units are registered in the list. Note that topologically equivalent architectures can have different blueprint representations.

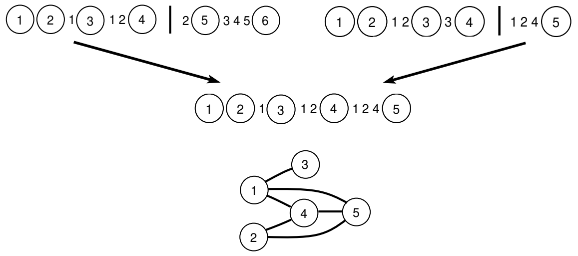

Fig. 2. Application of our crossover operator to the networks in the previous figure. The isolated unit can be removed.

Crossover is done by choosing a common cross point and joining two sub–lists of the blueprints. The cross point must be positioned before the last output unit of that network which has the lower number of units. If this constraint is satisfied it is guaranteed that the produced offspring operates on the same interface units (input and output) as its parents. So we get an *useful* network (see Figure 2). In this way, we have defined a *correct* crossover operator which is able to cross networks of *arbitrary* size.

In order to guarantee that the offspring isn't identical with one of the parent networks we must position the cross point behind the last input unit. Nevertheless we cannot guarantee that always *new* architectures arise. Even though we get useful networks it could happen that isolated or fixed hidden units arise. The output of *isolated* units isn't used by any other unit. *Fixed* units don't have connections to preceding units. Its weighted (constant) output can be substituted by the bias units and its connection weights. Both kinds of useless* units can be eliminated without affecting the networks functionality. In this way the shrunk offspring networks can become smaller than the smallest parent network. On the other side, it isn't possible that the number of units becomes larger than the biggest parent network. However, the number of connections can grow when the described crossover operator is applied.

4 Simulation Results

A description of implementation details can be found in [27] and [28], which are available via ftp†.

XOR–Problem

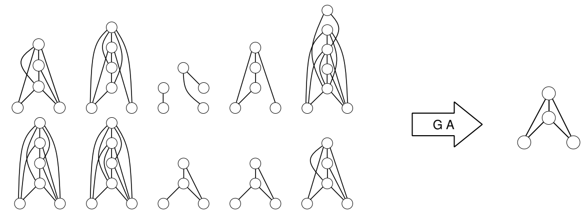

Fig. 3. The starting nets for the XOR Problem. All nets were made by chance.

To test our genetic algorithm, we tried to generate an optimal net for the XOR problem. Ten different nets were made by chance and so-called evolution servers were started on several computers (6 x Sun4, 4 x NeXT). The number of the training epochs per net was fixed on 1000 passes. After a short time of approximately five minutes the genetic algorithm converged to the well known topology for the XOR Problem (Figure 3).

*information is neither combined nor processed

†archive.cis.ohio-state.edu (128.146.8.52) pub/neuroprose/schiff.gann.ps.Z and pub/neuroprose/schiff.bp_speedup.ps.Z.

Thyroid data

The thyroid data are measurements of the thyroid gland. Each measurement vectors consists of 21 values — 15 binary and 6 analog ones. Each pattern is labeled by a class name which corresponds to the hyper–, normal–, and subnormal function of the thyroid gland. Since over 92% of all patients have a normal function, a useful classifier must be significantly better than 92% correct classifications. The training set consists of 3772 measurements and again 3428 measurements are available for testing. This classification problem is difficult to solve by neural nets. Several fixed topologies were tested. Figure 4 shows the course of the error during the training of these fixed topologies. It can be seen that several thousand learning passes are necessary to archieve a reasonable classifier.

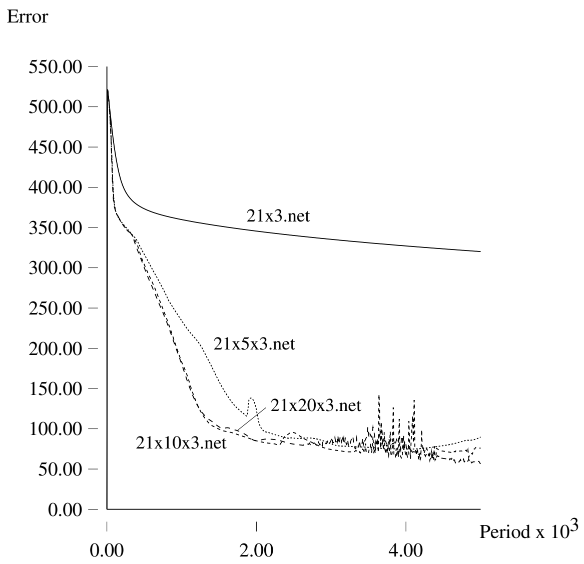

Fig. 4. The error during the training of the thyroid data with fixed topologies. The name of a net specifies its architecture. For example the name 21x10x3 indicates that the net consists of 21 input units, 3 output units and 10 hidden units. All units are fully interconnected.

In order to accelerate the evolution process, we don't use the complete training set. Instead of that, 713 measurements were chosen out of it. These selected patterns are the hardest ones of the training set because they are drawn from the boundary region between two categories. All nets were trained only with this reduced set. The training period during the evolution was fixed on 500 passes. Learning rate and momentum were adjusted constantly to 0.01 and 0.9 as well.

	#units	#Weights	Connectivity
n_1.net	54	110	9.0%
n_2.net	34	270	22.4%
n_3.net	49	461	47.9%
n_4.net	34	32	9.2%
n_5.net	39	378	71.6%
n_6.net	54	174	14.3%
n_7.net	24	40	63.5%
n_8.net	39	433	82.0%
n_9.net	31	233	92.5%
n_0.net	34	238	31.6%

Table 3. Characteristics of the starting nets for the thyroid task.

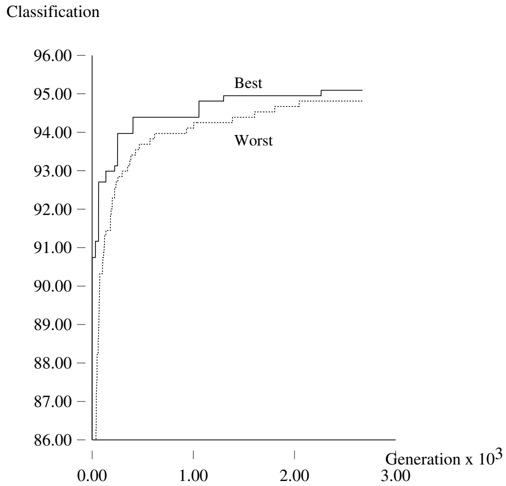

Fig. 5. The achieved classification performance during the evolution process. Only the classification performance of the best and the worst net of the population is shown.

To start the evolution process we designed 10 different nets by chance. These nets have various numbers of units and connections (Table 3). The achieved classification performance during the evolution process is shown in Figure 5. Only the values of the best and the worst net of each generation are represented. To get these well adapted networks the evolution needs about 72 hours. It can be seen that the best net reaches a classification performance of more than 95% correct classified patterns.

The development of the network's complexity is de-

	#Units	#Weights	Connectivity	reduced training set	whole training set	test set
fixed nets						
21x5x3.net	29	183	94.8%	76.6%	95.0%	94.3%
21x10x3.net	34	303	87.1%	76.9%	95.5%	94.8%
21x20x3.net	44	543	74.1%	71.1%	94.4%	93.9%
evol. nets						
n_123.net	48	285	31.1%	94.8%	98.3%	97.3%
n_125.net	49	283	29.4%	95.0%	98.1%	96.9%
n_130.net	49	278	28.9%	94.8%	98.3%	97.4%
n_136.net	48	277	30.3%	94.8%	98.2%	97.1%
n_147.net	49	276	28.7%	95.0%	98.4%	97.3%
n_149.net	50	279	27.6%	95.0%	98.1%	97.2%
n_152.net	50	278	27.5%	95.0%	98.2%	97.0%
n_154.net	49	286	29.7%	95.1%	98.3%	97.3%
n_155.net	50	279	27.6%	95.0%	98.6%	97.4%
n_156.net	50	278	27.5%	94.8%	98.4%	97.5%

Table 4. Comparison between the fixed and the generated networks. The evolved architectures archive a better performance although they are smaller.

monstrated in Figure 6. It can bee seen that at first big topologies are preferred because of the better results. Later the algorithm produced smaller architectures.

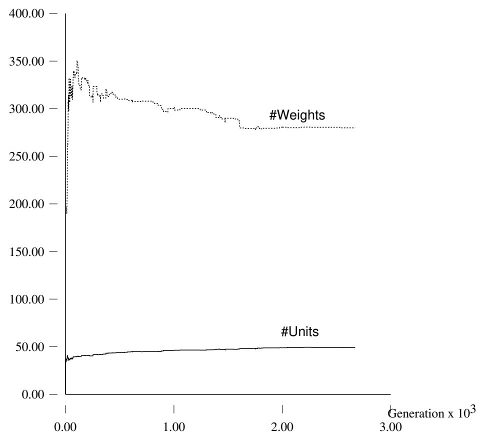

Fig. 6. The development of the net size during the evolution process.

Table 4 compares the results of the nets created by evolution with the results of the fixed architectures. To obtain comparable results, every fixed net was also trained 500 epochs with the reduced training set. The table represents the achieved classification performance with respect to the reduced training set, the whole training set, and the test set. Note, that the nets achieve a better classification performance on the complete training set. This is, because the extracted set consists only of the hardest samples of the training set. Furthermore, the table shows the number of units and connections in order to compare the network's complexity. It should be taken into account that the number of the connections is especially decisive for the size of a net. The adapted nets are more efficient although they are smaller. They train faster and generalize better. To achieve a similar classification performance with the fixed nets, many additional learning passes would be necessary.

Finally, we investigated the training behavior of the created nets with respect to the complete training set. Figure 7 shows the error of selected nets during the learning passes. The illustration demonstrates that the error starts to oscillate. This behavior is typical for all generated nets. Nevertheless, the error of the adapted nets is below the error of the fixed nets.

Table 5 compares the results of the trained nets. It shows that the adapted nets have a some better result concerning the training set. But with respect to the generalization behavior they are, however, superior to the fixed architectures ($> 1\%$). This can be explained by the fact that evolution finds feature extracting cells.

5 Conclusion and further work

Unfortunately, the generated nets show oscillations in the course of error. These oscillations start typically after the number of epochs which was used for the

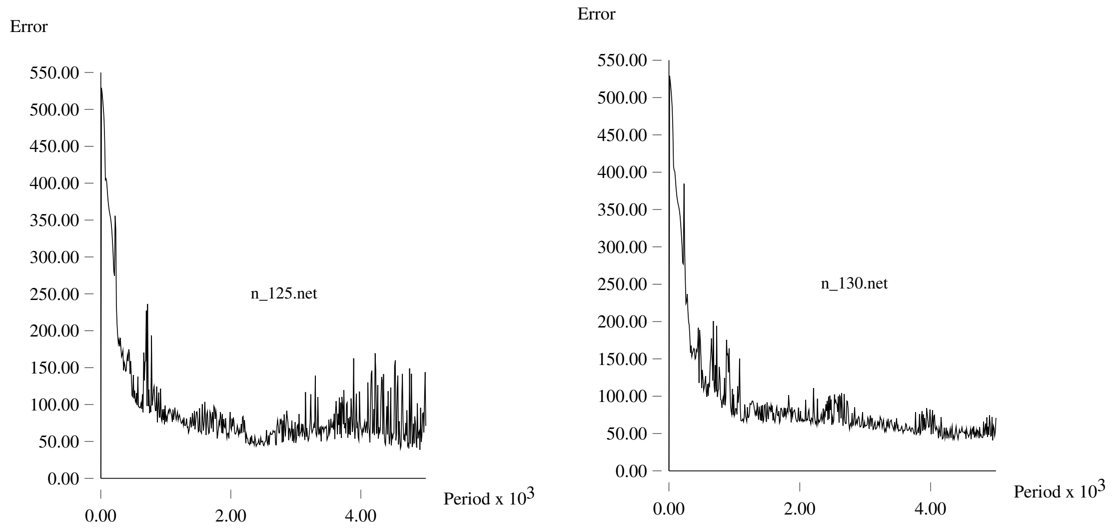

Fig. 7. Error during the training of the complete thyroid data with the evolved networks

	training set	test set
21x5x3.net	99.0%	97.4%
21x10x3.net	99.1%	97.3%
21x20x3.net	99.3%	97.4%
n_123.net	99.5%	98.2%
n_125.net	99.4%	98.6%
n_130.net	99.4%	98.4%
n_147.net	99.4%	98.2%
n_152.net	99.4%	98.5%
n_154.net	99.4%	98.4%
n_155.net	99.4%	98.6%
n_156.net	99.4%	98.4%

Table 5. Comparison between the fixed and the generated networks which are trained with the whole training set.

genetic algorithm (here at 500 epochs). The genetic algorithm generates networks which can be trained quickly up to this epoch. The training behavior of the nets after this number of epoch is uncertain. In order to suppress these oscillations, a learning rate adaption should be used.

Furthermore, we have found that the number of training epochs can be reduced by using quickprop [29]. We plan to implement this training procedure in our algorithm. In order to improve the generalization behavior of the nets, the performance of the produced nets should be determined with respect to the test set. The generalization behavior must then be determined on a third test (test) pattern set. In the case of the thyroid data the nets could be trained with the reduced training set and the quality measure should be determined with the complete training set. Finally, the test set could be used to determine the generalization performance.

6 Acknowledgements

We are grateful to K.–H. Staudt for implementation of the network editor. By means of this tool we are able to visualize and analyse the generated topologies. This work is supported by the *Deutsche Forschungsgemeinschaft* (DFG) as part of the project *FE–generator* (grant Schi 304/1–1).

7 References

[1] Minsky and Papert, *Perceptrons*, MIT Press, 1969

[2] Rumelhart D.E., Hinton G.E. and Williams R.J., *Learning internal representations by error propagation*, in Parallel Distributed Processing: *Explorations in the Microstructures of Cognition*, Vol.I, MIT Press, pp. 318–362, 1986

[3] Hornik K., Stinchcombe M. and White H., *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, Vol. 2, pp. 359–366, Pergamon Press, 1989

[4] Miller G., Todd P.M. and Hegde S.U., *Designing Neural Networks Using Genetic Algorithms*,Proc. of the third Intern. Conf. on Genetic Algorithms (ICGA), pp. 379–384, San Mateo (CA), 1989,

[5] Schiffmann W.H. and Mecklenburg K., *Genetic Generation of Backpropagation Trained Neural Networks*, Proc. of Parallel Processing in Neural Systems and Computers (ICNC), Eckmiller R. et al. (Eds.), pp. 205–208, Elsevier, 1990

[6] Schiffmann W.H., *Selbstorganisation neuronaler Netze nach den Prinzipien der Evolution*, Fachbericht 7/1989, Universität Koblenz

[7] Schiffmann W.H., Joost M. and Werner R., *Performance Evaluation of Evolutionarily Created Neural Network Topologies*, Proc. of Parallel Problem Solving from Nature, Schwefel H.P. and Maenner R. (Eds.), pp. 274–283, Lect. Notes in Computer Science, Springer, 1991

[8] Whitley D., *Applying Genetic Algorithms to Neural Net Problems*, Neural Networks, Vol. 1, p. 230, 1988

[9] Whitley D. and Hanson T., *Optimizing Neural Networks Using Faster, More Accurate Genetic Search*, Proc. of the third Intern. Conf. on Genetic Algorithms (ICGA), pp. 391–396, San Mateo (CA), 1989

[10] Whitley D. and Bogart C., *The Evolution of Connectivity: Pruning Neural Networks Using Genetic Algorithms*, Proc. of the intern. Joint Conf. on Neural Networks, Vol. I, pp. 134–137, 1990

[11] Whitley D., Starkweather and Bogart C., *Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity*, Parallel Computing Vol. 14, pp. 347–361, Elsevier, 1990

[12] Whitley D., *The GENITOR Algorithm and Selection Pressure: Why Rank–Based Allocation of Reproductive Trials is Best*, Proc. of the third Intern. Conf. on Genetic Algorithms (ICGA), pp. 116–121, San Mateo (CA), 1989

[13] Heistermann J., *Parallel Algorithms for Learning in Neural Networks with Evolution Strategy*, Parallel Computing, Vol. 12, 1989

[14] Heistermann J., *Learning in Neural Nets by Genetic Algorithms*, Proc. of Parallel Processing in Neural Systems and Computers (ICNC), Eckmiller R. et al. (Eds.), pp. 165–168, Elsevier, 1990

[15] Montana D. and Davis C., *Training Feedforward Neuronal Networks Using Genetic Algorithms*, Technical Report, BBN Systems and Technologies Inc., Cambridge (MA), 1989

[16] de Garis H., *Genetic Programming — Modular Neural Evolution for Darwin Machines*, Proc. of the Intern. Joint Conf. on Neural Networks, Vol I, pp. 194–197, 1990

[17] Weiss, G., *Combining neural and evolutionary learning: Aspects and approaches*, Report FKI–132–90, Technische Universität München, 1990

[18] Kitano H., *Empirical Studies on the Speed of Convergence of Neural Network Training Using Genetic Algorithms*, Proc. of the National Conf. of the American Association of Artificial Intelligence (AAAI), pp. 789–795, 1990

[19] Dodd N., *Optimization of Network Structure Using Genetic Techniques*, Proc. of the Intern. Conf. on Neural Networks, pp. 693–696, Paris, 1990

[20] Lehar S. and Weaver J., *A Developmental Approach to Neural Network Design*, Proc. of the IEEE Intern. Conf. on Neural Networks, Vol. I, pp. 97–104, 1987

[21] Harp S.A., Samad T. and Guha A., *Towards the Genetic Synthesis of Neural Networks*, Proc. of the third Intern. Conf. on Genetic Algorithms (ICGA), pp. 360–369, San Mateo (CA), 1989

[22] Harp S.A., Samad T. and Guha A., *The Genetic Synthesis of Neural Networks*, Technical Report CSDD–89–14852–2, Honeywell, Golden Valley (MN), 1989

[23] Merrill J. and Port R., *Fractally Configured Neural Networks*, Neural Networks, Vol. 4, pp.53–60, 1991

[24] Mjolsness E. and Sharp D.H., *A Prelimiary Analysis of Recursively Generated Networks*, in Denker J. (Eds.): *Neural Networks for Computing*, Snowbird (Utah), 1986

[25] Mjolsness E., Sharp D.H. and Alpert B.K., *Recursively Generated Neural Networks*, Proc. of the IEEE Intern. Conf. on Neural Networks, Vol. III, pp. 165–171, 1987

[26] Mühlenbein H. and Kindermann J., *The Dynamics of Evolution and Learning — Towards Genetic Neural Networks*, in Pfeifer et al. (Eds.): *Connectionism in Perspective*, Elsevier, 1989

[27] Schiffmann W.H., Joost M. and Werner R. *Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons*, Technical Report 15/1992, University of Koblenz

[28] Schiffmann W.H., Joost M. and Werner R. *Synthesis and Performance Analysis of Multilayer Neural Network Architectures*, Technical Report 16/1992, University of Koblenz

[29] Fahlman S.E., *An Empirical Study of Learning Speed in Back–Propagation Networks*, Technical Report, Carnegie–Mellon University, CMU–CS–88–162, 1988