Using Genetic Engineering to Find Modular Structures and Activation Functions for Architectures of Artificial Neural Networks

Christoph M. Friedrich[1] and Claudio Moraga[2]

[1] University of Witten/Herdecke; Institute for Technology Development and Systems Analysis; Alfred-Herrhausen Str. 50; D-58448 Witten, Germany; E-mail: chris@uni-wh.de; URL: http://www.tussy.uni-wh.de/~chris

[2] University of Dortmund; Department of Computer Science; D-44221 Dortmund; Germany; E-mail: moraga@ls1.informatik.uni-dortmund.de

Abstract. An Evolutionary Algorithm is used to optimize the architecture and activation functions of an Artificial Neural Networks (ANN). It will be shown that it is possible, with the help of a graph-database and Genetic Engineering, to find modular structures for these networks. Some new graph-rewritings are used to construct families of architectures from these modular structures. Simulation results for two problems are given. An analysis of the data in the database suggest the usage of symmetric activation functions.

1 Introduction

One of the major problems using ANN's is the design of their architecture. The architecture of an ANN greatly influences its learning capability and generalization ability. Beside of constructive and pruning techniques, Evolutionary Algorithms have been suggested by many scientists to find good performing architectures for Artificial Neural Networks. For a summary of recent results reading of [2] is suggested. Most of these methods have the problem of scalability. In [4] a method was proposed to find building-blocks for architectures of Artificial Neural Networks and to create families of modular architectures. In this paper an extension of this method is presented that additionally to the architecture optimizes the kind of activation functions of the neurons. Furthermore new graph-rewriting methods are suggested and tested.

2 Cellular Encoding

A method to optimize the architecture and weights of boolean neural networks, where the weights are restricted to the values -1 and 1, was suggested

* Appeared in: Computational Intelligence, Theory and Applications; Proceedings of the 5th Fuzzy Days; (LNCS 1226); Pages 150-161, Dortmund, Germany (April 1997)

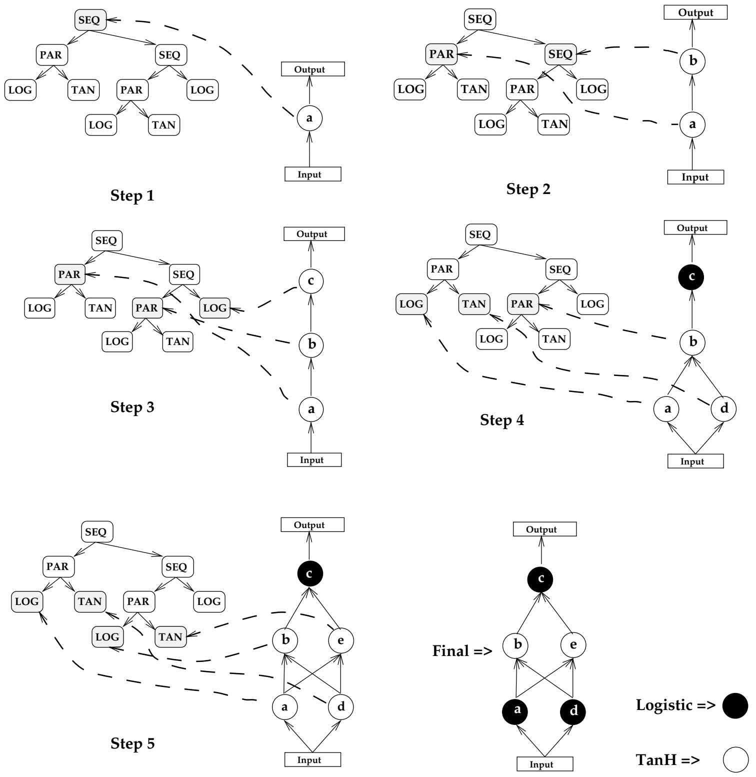

Fig. 1. Six steps to develop an 2-2-1 network with mixed activation functions

by Gruau [5] who named it Cellular Encoding. In this encoding technique, the information to develop an architecture is obtained by interpreting the information from a grammar-tree. The nodes of this tree encode information about graph rewriting operations. The development of a neural network starts with a graph consisting of one node (cell), having ingoing connections from the input area and outgoing connections to the output area. Every cell has a reading head pointing to one node of the grammar-tree (cellular code). The nodes of the cellular code contain symbols defining the graph rewriting operation. This technique is comparable to a Turing machine. Instead of writing to a tape in Cellular Encoding the cells are changed. The operation #par: for example symbolizes a parallel cell division, both cells inherit the input and output connections from the mother cell. The reading heads of the new created cells are moved to the left and right subtree of the grammar-tree. For biological plausibility, the development of the cells should be parallel. This can be achieved by using a FIFO Queue. The development of a network ends, if all cells evaluate the operations, located as terminals at the leaves of the grammar-tree. The evaluation of the terminal #log: for example determines the final activation function of the node. Figure 1 shows six steps to develop the architecture of an XOR network with mixed activation functions.

As mentioned above, the cellular code is given as a grammar-tree. In this case it is possible to use Genetic Programming [8] to optimize the cellular code. Genetic Programming is a Genetic Algorithm, that uses trees instead of Bitstrings as representation. The recombination operator crossover is realized by exchanging the subtrees of two parent individuals. The mutation operator brings some variations into the genome and is realized by inclusion of randomly initialized trees.

Gruau has proved some properties of his encoding technique, among them are completeness, compactness, closure, modularity and scalability. He got these properties by using a recursion operator that makes it possible to repeat parts of his cellular Code. In [6] he used Automatic Defined Functions (ADF) to get modular architectures.

3 Modified Cellular Encoding

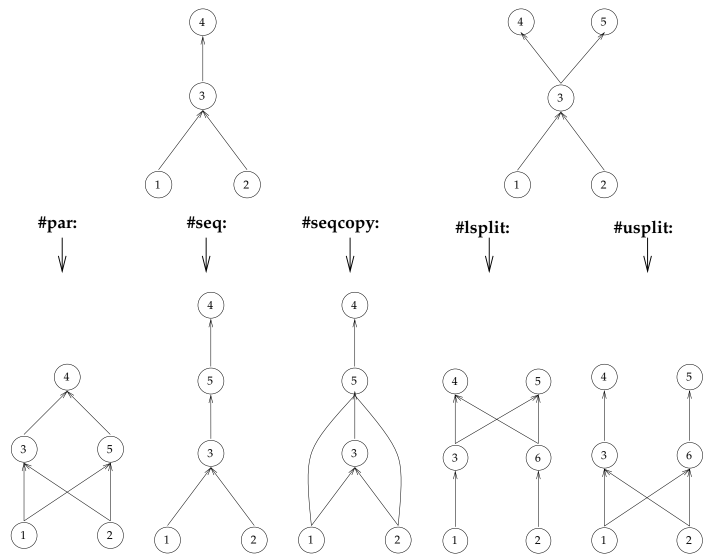

Fig. 2. The used rewriting operations for the modified Cellular Encoding

For the optimization of the architecture of ANN's with real-valued weights, the Cellular Encoding method was modified. Some new Development operators were developed and tested. The set of operators {#par:, #seq:, #seqcopy:, #usplit:, #lsplit:} was used to develop the architectures. Figure 2 shows the effects of rewriting the hidden node of a network with the development operators. In contrast to some work of Gruau in this work the input and output nodes are not coded in the cellular code and are given by the problem. As an extension to [4], the set of terminals {#tanh:, #tanhDiv2: #log: #gaussian:} was used to develop architectures with mixed activation functions. Following activation functions were used:

$$f_{\#tanh:}(x) = tanh(x) \tag{1}$$

$$f_{\#tanhDiv2:}(x) = tanh(\frac{x}{2}) \tag{2}$$

$$f_{\#log:}(x) = \frac{1}{1+e^{-x}} \tag{3}$$

$$f_{\#gaussian:}(x) = e^{-x^2} \tag{4}$$

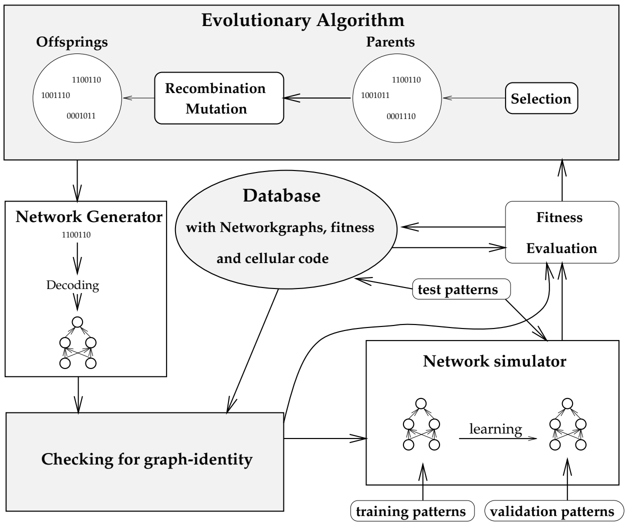

Fig. 3. Modified Cellular Encoding using a graph-database

One property of the created networks should be their correctness. A created graph is a correct architecture for a feedforward neural network, if it contains only feedforward connections and if all hidden nodes are on a path from an input node to an output node. All operators were designed to hold this property, so only correct network graphs could be created. Different to Gruau's work, it is possible with the operator #seqcopy:, to create networks with shortcut connections.

The second modification to the evolutionary development cycle was the use of a graph-database. This database contains the cellular code, network graphs and fitness parameters like the number of necessary epochs for learning, classification error on the learning, validation and test set and other parameters of the networks. Figure 3 shows the working structure of the evolutionary cycle.

Before the fitness calculation of a developed network, it is checked, if the fitness of this network was calculated five* times before. If not, the fitness is calculated again and the results are saved in the database. The resulting fitness of a network is the mean of all fitness evaluations of this network. This method allows it to minimize the fitness distortion resulting from the random initialization of the weights. This increases the validity and robustness of the obtained results. Several authors document only solutions which are results of a single fitness evaluation. In this work, only results are given for architectures tested five times.

3.1 Genetic Engineering

Optimizing the architecture of an ANN using Evolutionary Algorithms leads to proper performing architectures for problems. But it gives no insight into the problem, how good architectures are built or what activation functions should be used in general. It would be more interesting to find building principles for good architectures to create modular architectures.

In his work on Genetic Programming, Koza [8] suggested Automatic Defined Functions as a possible solution to this problem. Unfortunately for the task of architecture Optimization the fitness evaluation is very computing intensive, so it is not possible to use big population sizes and many epochs. Especially for Genetic Programming, where the representation of a problem is a grammar-tree, Altenberg [1] suggested a method to find good modular structures that are subtrees of the representation. He called this technique *Genetic Engineering*. It depends on the assumption, that some parts of the genetic program have a higher impact on the complete fitness of the genetic program than others. It is a problem to attach a fitness value to sub-programs, because it is only possible to obtain the fitness of the complete phenotype.

In this work the fitness of modular structures of the genetic programs were found by statistically analyzing the cellular code in the graph-database, which was build during the formerly described optimization process. This makes it possible to obtain information about several optimization runs that start with different populations. The fitness of a subtree is equivalent to the frequency of this subtree in the database. The found modular structures are called modules and noted as a preorder traversal of the subtree.

* The number five was used as a compromise between statistical evaluation possibility and necessary computing time. It would be better to use the fitness distribution as information.

3.2 Graph-Rewriting

Having found good modular structures for neural network architectures, it would be interesting to see how architectures built of this modules perform. One possibility is to include these subtrees as encapsulated functions in the next evolutionary optimization process.

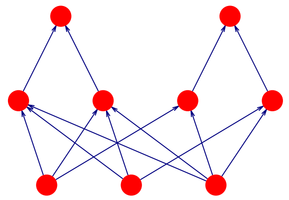

Fig. 4. Resulting network of $(1,1)_{\#par:}$-rewriting of a 3-1-2 network with the module #usplit:#par:#lsplit:

In contrast to this, in this work some special graph-rewriting methods were used. The $(i,k)_{op}$-rewriting of a network graph with a module starts with a fully connected $n-1-m$ network, where n, m are determined by the problem. Then the hidden node will be rewritten i-times with the development-operator *op*. The resulting hidden nodes are rewritten k times with the rewriting operations defined by the module. For $k = 0$ and $op = \#par$: this results in standard architectures with one hidden layer and i hidden nodes. For *op*=#seqcopy: this results in deep networks like that of the Cascade Correlation algorithm [3]. With this method it is possible to create families of architectures consisting of good performing modules. Figure 4 shows an example of a network created by $(1,1)_{\#par:}$-rewriting of a 3-1-2 network with the module #usplit:#par:#lsplit:. The hidden node will be first rewritten with a #usplit: operation and the resulting nodes will be rewritten with the operation #par: or #lsplit:.

4 Experimental Results

The data for the first experiment are taken from the benchmark suite from Prechelt [11]. The task for the diabetes problem is to classify from some diagnostic data (e.g. blood pressure, result of glucose tolerance test etc.), whether a female Pima indian is diabetes positive or not. This problem with 8 input and 2 output parameters is difficult, because some values for the input patterns are not available and set to zero. This decepts the classification algorithm, but is realistic for real world tasks. In contrast to [4] in this work cross validation with 3 datasets (576/96/96) and weight-decay was used to find good performing architectures. This improvement makes it possible to obtain better results than in that paper. Additionally there is no bias towards the test set by the evolutionary algorithm, because the fitness function does not use the test set. Prechelt and Michie et al. [10] found networks with an classification error on the test set of 24.8 %. The learning set was learned with RPROP up to 500 epochs, but usually cross validation stopped learning before reaching this maximum. As classification method WTA (winner takes all) was used. In the first tests only $tanh(x)$ was used as the activation function. In a second test, mixed activation functions were allowed. The population size for the Genetic Algorithm was 100. The fitness function was a linear combination of the classification error on the validation set (factor 1) and the learning set (factor 0.3).

Table 1. Classification error for the diabetes problem with the module #par:#usplit:#usplit:

Architecture	Size	classification error			Epochs
	Nodes/Weights	learn set	vali. set	test set	
Best Evo	33/148	15.0 %	22.4 %	23.1 %	180
Best Evo Mixed	21/95	17.0 %	21.3 %	23.8 %	120
$(13,0)_{\#par:}$	23/130	23.8 %	24.5 %	25.0 %	145
$(5,1)_{\#par:}$	26/160	20.1 %	23.1 %	23.1 %	212
$(2,1)_{\#seqcopy:}$	18/100	20.3 %	21.8 %	25.6 %	135

Several optimization runs with and without mixed activation functions were made. Altogether about 120,000 architectures were tested, this needed about 20 days of an UltraSparc computing time. This is much time, but it is possible to parallelize the evolutionary algorithm. The application of Genetic Engineering on the data in the database found the module #par:#usplit:-#usplit: with the highest frequency. Table 1 summarizes the results. All results are the mean of five different runs from the best members of any graph-rewriting family. Figure 5 shows the best architecture with mixed activation functions found by the evolutionary algorithm. It can be seen, that the evolutionary algorithm found a network with a strict hierarchical structure.

It can be concluded that for this problem, the evolutionary algorithm found networks which are better than standard architectures and networks found in literature. The performance of networks with mixed activation functions is comparable to that of uniform $tanh(x)$ activation functions but their architecture is smaller. The architectures built by $(i,1)_{\#par:}$-rewriting with the best found module show better performance than standard architectures. The architecture built by $(2,1)_{\#seqcopy:}$-rewriting (compare Figure 6) gave good results on the learning and validation set, but the bias towards the patterns in the validation set leads to bad results for the test set. All other rewriting classes except the #seq:-rewritings gave results comparable to standard architectures.

The second tested problem was the well known two-spirals problem [9]. The task is to learn to discriminate between two sets of training points which lie on two distinct spirals in the x-y plane. These spirals coil three times around the origin and around one another. This problem is very difficult for standard backpropagation networks and only Cascade Correlation [3] gave

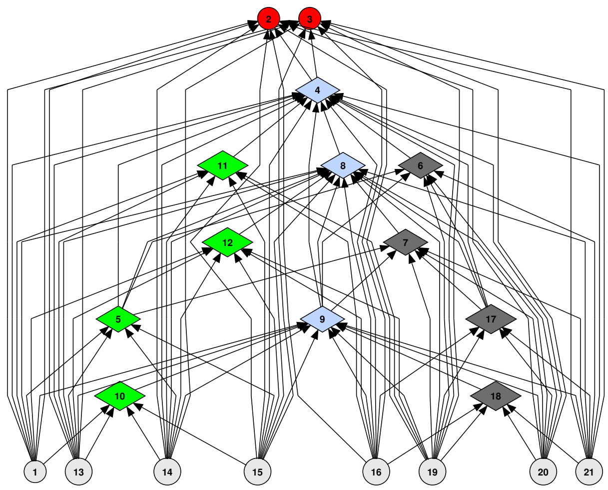

Fig. 5. Best evolved network for the diabetes problem with mixed activation functions

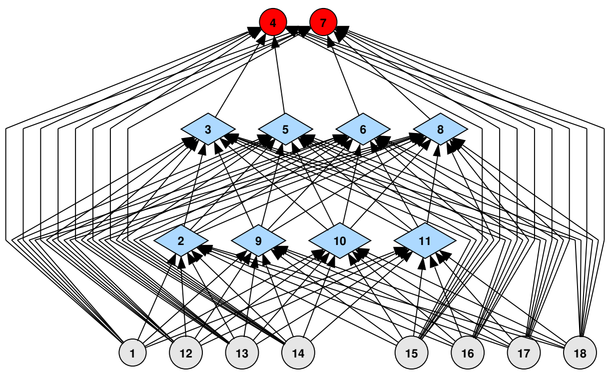

Fig. 6. network created by $(2,1)_{\#seqcopy:}$-rewriting with the module #par:#usplit:#usplit: for the diabetes problem

good results, their network needed about 1700 epochs to solve this problem. In this work the output activation function was the $tanh(x)$ function and as classification method the correctness of the sign of the outputs was used. This is a less strict classificaton method than the 402040 technique, but good for boolean classification tasks. For all evolutionary created networks the number of hidden nodes was bounded at 40 and RPROP was used to present the learning set up to 3000 epochs.

The first spirals test was the usage as an approximation problem with the classic 194 points and no cross validation. The evolutionary algorithm with the $tanh(x)$ activation function found a network with 25 nodes and 116 weights that approximates, in the mean of five runs, the spirals in 1788 Epochs. It should be noted, that not all five runs were successful in learning all points. The application of Genetic Engineering found the module #seqcopy:#usplit:#usplit:. The family of architectures build by $(i,1)_{\#par:}$-rewriting gave only results comparable to that of standard architectures with comparable size. Whether this modular architectures nor standard architectures with up to 40 nodes could learn this function in 3000 epochs.

Table 2. Classification error for the second spirals test with the module #par:#lsplit:#par:

Architecture	Size	classification error			Epochs
	Nodes/Weights	learn set	vali. set	test set	
Best Evo	39/123	3.3 %	31.4 %	40.2 %	2157
Best Evo Mixed	18/55	5.2 %	38.8 %	42.7 %	1728
$(40,0)_{\#par:}$	43/122	5.1 %	47.0 %	52.0 %	2736
$(8,1)_{\#par:}$	35/82	7.2 %	41.3 %	47.8 %	2423
$(7,1)_{\#seqcopy:}$	31/300	2.5 %	45.2 %	49.3 %	1008

In the second spirals test, the network was learned with 194 points and tested on 584 unseen points of the spirals to measure generalization ability. The validation and test set consists of 292 points each. This time experiments with and without mixed activation functions were made. The fitness function for the evolutionary algorithm was a linear combination of classification error on the validation set (factor 1) and the classification error on the learning set (factor 0.3). The learning set was learned with RPROP up to 3000 epochs, but usually cross validation stopped learning before.

Again Several runs with and without mixed activation functions were made. Altogether about 60,000 architectures were tested. The application of Genetic Engineering on the data in the database found the module #par:-#lsplit:#par: with the highest frequency. Table 2 summarizes the results for the second spirals test. The presented results are from the best networks in

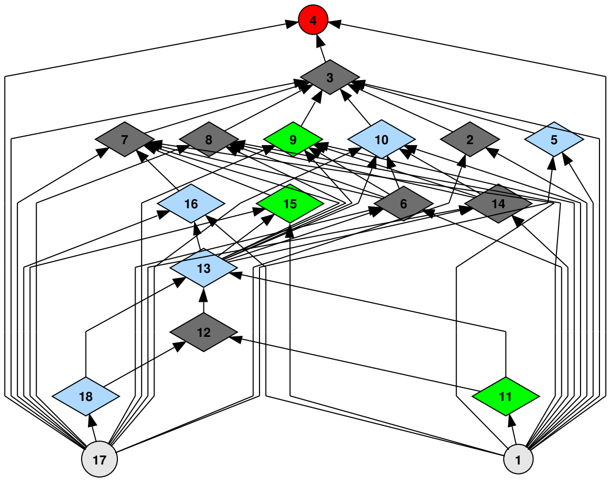

Fig. 7. best evolutionary found network with mixed activation functions for the second spirals test

every graph-rewriting class. Figure 7 shows the network graph of the best evolutionary found network with mixed activation functions.

It could be shown, that networks created by the modified Cellular Encoding technique or by graph-rewriting have better generalization abilities than standard architectures of comparable size. Evolutionary created networks with mixed activation functions show comparable performance than uniform $tanh(x)$ activation functions, but again architectures are smaller. It could be seen that the two spirals problem is a very hard problem for standard feedforward networks. The architectures tend to memorize the learning set, so big network graphs with many weights (degrees of freedom) were mainly created. No network architecture shows satisfying generalization abilities for this problem.

The database was statistically analyzed to get information about the used activation functions. Table 3 gives an overview about the used activation functions. It can be concluded, that for the tested problems the usage of symmetric activation functions (not gaussian) gives better results for evolutionary created networks.

Table 3. Percentages of the used activation functions in all evolved network graphs with mixed activation function

Problem	#gaussian:	#tanH:	#tanHdiv2:	#logistic:
Diabetes	7.9 %	42.8 %	27.9 %	21.4 %
Spirals	1.4 %	46.6 %	11.8 %	40.2 %

An analysis of the last population of every evolutionary optimization process shows a bias towards architectures that have good classification performance on the validation set, but worse classification on the unseen test set. Therefore it is necessary to be very careful to choose pattern sets that are representative and big enough for the problem domain and to adjust the parameters for cross validation and the evolutionary process.

5 Conclusion and Outlook

It was shown that it is possible to find good performing architectures for non-boolean ANN's, with a modified Cellular Encoding method. All evolved architectures show better performance for the optimized criteria than comparable standard architectures or architectures found in literature. With the help of a graph-database it was furthermore possible to find modular structures for the architecture of ANN's. Architectures created with a graph-rewriting method from this modular structures show better performance than standard architectures. The usage of mixed activation functions in an architecture show similar performance than uniform activation functions. From data in the database it can be concluded, that symmetric activation functions result in better performance. This technique can be used as an alternative to Automatic Defined Functions for problems that restrict the number of possible fitness evaluations. Using this method, it is possible to create a library of good modular structures for problem-specific modular network architectures.

In later works the possibility should be tested, if the modules found by Genetic Engineering could be used for the mixture of expert networks architecture proposed in [7]. Another possibility will be the usage in Cascade Correlation networks, where not nodes but modules will be cascaded.

References

1. L. Altenberg. The evolution of evolvability in genetic programming. In K. E. Kinnear, editor, *Advances in Genetic Programming*. MIT Press, 1994. http://pueo.mhpcc.edu/ altenber/PAPERS/Papers2.html.
2. J. Branke. Evolutionary algorithms for neural network design and training. In *Proceedings of the 1st Nordic Workshop on Genetic Algorithms and its Applications*, 1995. ftp://ftp.aifb.uni-karlsruhe.de/pub/jbr/Vaasa.ps.
3. S. Fahlmann and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Systems -2*, pages 525–532. Morgan Kaufmann, 1990.
4. C. M. Friedrich and C. Moraga. An evolutionary method to find good building-blocks for architectures of artificial neural networks. In *Proceedings of the Sixth International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU '96)*, pages 951–956, Granada, Spain, 1996. ftp://archive.cis.ohio-state.edu/pub/neuroprose/friedrich.ipmu96.ps.Z.

5. F. Gruau. Cellular encoding of genetic neural networks. Technical Report 92-21, Ecole Normale Superieure de Lyon, Institut IMAG, 1992. ftp://lip.ens-lyon.fr/pub/Rapports/RR/RR92/RR92-21.ps.Z.
6. F. Gruau. Genetic synthesis of modular neural networks. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 318–325. Morgan Kaufmann, 1993.
7. R. A. Jacobs, M. I. Jordan, and A. G. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. Technical Report COINS Technical Report 90-27, University of Massachusetts, Amherst, MA, 1990. ftp://archive.cis.ohio-state.edu/pub/neuroprose/jacobs.modular.ps.Z.
8. J. R. Koza. *Genetic programming, on the programming of Computers by means of natural selection*. MIT Press, 1992.
9. K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In *Proceedings of the 1988 Connectionist Models Summer School*, 1988.
10. D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, neural and statistical classification*. Ellis Horwood Ltd., 1994.
11. Lutz Prechelt. PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, September 1994. ftp://ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.Z.