Training Neural Networks with GA Hybrid Algorithms

Enrique Alba and J. Francisco Chicano

Departamento de Lenguajes y Ciencias de la Computación
University of Málaga, SPAIN
{eat,chicano}@lcc.uma.es

Abstract. Training neural networks is a complex task of great importance in the supervised learning field of research. In this work we tackle this problem with five algorithms, and try to offer a set of results that could hopefully foster future comparisons by following a kind of standard evaluation of the results (the Prechelt approach). To achieve our goal of studying in the same paper population based, local search, and hybrid algorithms, we have selected two gradient descent algorithms: Backpropagation and Levenberg-Marquardt, one population based heuristic such as a Genetic Algorithm, and two hybrid algorithms combining this last with the former local search ones. Our benchmark is composed of problems arising in Medicine, and our conclusions clearly establish the advantages of the proposed hybrids over the pure algorithms.

1 Introduction

The interest of the research in Artificial Neural Networks (ANNs) resides in the appealing properties they exhibit: adaptability, learning capability, and ability to generalize. Nowadays, ANNs are receiving a lot of attention from the international research community with a large number of studies concerning training, structure design, and real world applications, ranging from classification to robot control or vision [1].

The neural network training task is a capital process in supervised learning, in which a pattern set made up of pairs of inputs plus expected outputs is known beforehand, and used to compute the set of weights that makes the ANN to learn it. One of the most popular training algorithms in the domain of neural networks is the Backpropagation (or generalized delta rule) technique [2], a gradient-descent method. Other techniques such as evolutionary algorithms (EAs) have been also applied to the training problem in the past [3,4], trying to avoid the local minima that so often appear in complex problems. Although training is a main issue in ANN's design, many other works are devoted to evolve the layered structure of the ANN or even the elementary behavior of the neurons composing the ANN. For example, in [5] a definition of neurons, layers, and the associated training problem is analyzed by using parallel genetic algorithms; also, in [6] the architecture of the network and the weights are evolved by using the EPNet evolutionary system. It is really difficult to perform a revision of this

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 852–863, 2004.

topic; however, the work of Yao [7] represents an excellent starting point to get acquired of the research in training ANNs.

The motivation of the present work is manyfold. First, we want to perform a standard presentation of results that promotes and facilitates future comparisons. This sounds common sense, but it is not frequent that authors follow standard rules for comparisons such as the structured Prechelt's set of recommendations [8], a "de facto" standard for many ANN researchers. A second contribution is to include in our study, not only the well known Genetic Algorithm (GA) and Backpropagation algorithm, but also the Levenberg-Marquardt (LM) approach [9], and two additional hybrids. The potential advantages coming from an LM utilization merit a detailed study. We have selected a benchmark from the field of Medicine, composed of three classification problems: diagnosis of breast cancer, diagnosis of diabetes in Pima Indians, and diagnosis of heart disease.

The remainder of the article is organized as follows. Section 2 introduces the Artificial Neural Network computation model. Next, we give a brief description of the algorithms under analysis (Section 3). The details of the experiments and their results are shown in Section 4. Finally, we summarize our conclusions and future work in Section 5.

2 Artificial Neural Networks

Artificial Neural Networks are computational models naturally performing a parallel processing of information [10]. Essentially, an ANN can be defined as a pool of simple processing units (*neurons*) which communicate among themselves by means of sending analog signals. These signals travel through weighted connections between neurons. Each of these neurons accumulates the inputs it receives, producing an output according to an internal activation function. This output can serve as an input for other neurons, or can be a part of the network output. In Fig. 1 left we can see a neuron in detail.

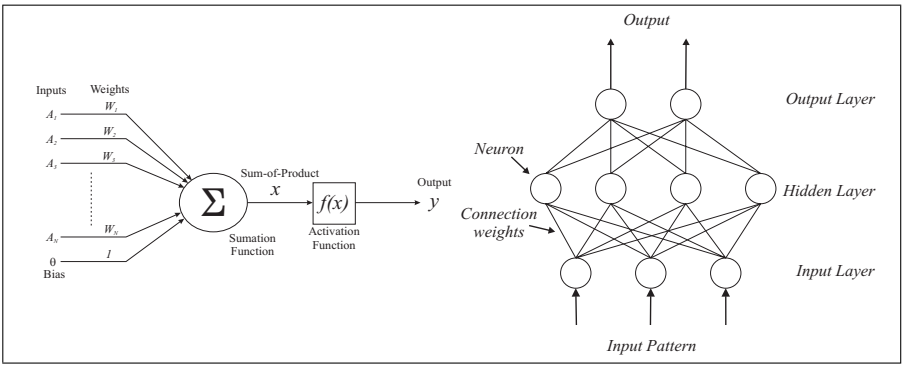

Fig. 1. An artificial neuron (left) and a multilayer perceptron (right)

There is a set of important issues involved in the ANN design process. As a first step, the architecture of the network has to be decided. Initially, two major options are usually considered: *feedforward* networks and *recurrent* networks (additional considerations regarding the *order* of the ANN exist, but are out of our scope). The feedforward model comprises networks in which the connections are strictly feedforward, i.e., no neuron receives input from a neuron to which the former sends its output, even indirectly. The recurrent model defines networks in which feedback connections are allowed, thus making the dynamical properties of a capital importance. In this work we will concentrate on the first and simpler model: the feedforward networks. To be precise, we will consider the so-called *multilayer perceptron* (MLP) [11], in which units are structured into ordered layers, and connections are allowed only between adjacent layers in an input-to-output sense (see Fig. 1 right).

For any MLP, several parameters such as the number of layers and the number of units per layer must be defined. After having done this, the last step in the design is to adjust the weights of the network, so that it produces the desired output when the corresponding input is presented. This process is known as *training* the ANN or *learning* the network weights. Network weights comprise both the previously mentioned connection weights, as well as a *bias* term for each unit. The latter can be viewed as the weight of a constant saturated input that the corresponding unit always receives. As initially stated, we will focus on the learning situation known as *supervised* training, in which a set of input/desired-output patterns is available. Thus, the ANN has to be trained to produce the desired output according to these examples. The input and output of the network are both real vectors in our case.

In order to perform a supervised training we need a way of evaluating the ANN output error between the actual and the expected output. A popular measure is the Squared Error Percentage (SEP). We can compute this error term just for one single pattern or for a set of patterns. In this last case, the SEP is the average value of the patterns individual SEP. The expression for this global SEP is:

$$SEP = 100 \cdot \frac{o_{max} - o_{min}}{P \cdot S} \sum_{p=1}^{P} \sum_{i=1}^{S} (t_i^p - o_i^p)^2 \quad . \tag{1}$$

where t_i^p and o_i^p are, respectively, the i-th components of the expected vector and the actual current output vector for the pattern p; o_{min} and o_{max} are the minimum and maximum values of the output neurons, S is the number of output neurons, and P is the number of patterns.

In classification problems, we could use still an additional measure: the Classification Error Percentage (CEP). CEP is the percentage of incorrectly classified patterns, and it is a usual complement to any of the other two (SEP or the well-known MSE) raw error values, since CEP reports in a high-level manner the quality of the trained ANN.

3 The Algorithms

We use for our study several algorithms to train ANNs: the Backpropagation algorithm, the Levenberg-Marquardt algorithm, a Genetic Algorithm, a hybrid between Genetic Algorithm and Backpropagation, and a hybrid between Genetic Algorithm and Levenberg-Marquardt. We briefly describe them in the following paragraphs.

3.1 Backpropagation

The Backpropagation algorithm (BP) [2] is a classical domain-dependent technique for supervised training. It works by measuring the output error, calculating the gradient of this error, and adjusting the ANN weights (and biases) in the descending gradient direction. Hence, BP is a gradient-descent local search procedure (expected to stagnate in local optima in complex landscapes).

First, we define the squared error of the ANN for a set of patterns:

$$E = \sum_{p=1}^{P} \sum_{i=1}^{S} (t_i^p - o_i^p)^2 \quad . \tag{2}$$

The actual value of the previous expression depends on the weights of the network. The basic BP algorithm (without momentum in our case) calculates the gradient of E (for all the patterns in our case) and updates the weights by moving them along the gradient-descendent direction. This can be summarized with the expression $\Delta \mathbf{w} = -\eta \nabla E$, where the parameter $\eta > 0$ is the learning rate that controls the learning speed. The pseudo-code of the BP algorithm is shown in Fig. 2.

```
InitializeWeights;
while not StopCriterion do
    for all i,j do
        w_ij := w_ij - η ∂E/∂w_ij ;
    endfor;
endwhile;
```

Fig. 2. Pseudo-code of the BP algorithm

3.2 Levenberg-Marquardt

The Levenberg-Marquardt algorithm (LM) [9] is an approximation to the Newton method used also for training ANNs. The Newton method approximates the error of the network with a second order expression, which contrasts to the Backpropagation algorithm that does it with a first order expression. LM is popular in the ANN domain (even it is considered the first approach for an unseen

MLP training task), although it is not that popular in the metaheuristics field. LM updates the ANN weights as follows:

$$\Delta \mathbf{w} = -\left[\mu I + \sum_{p=1}^{P} J^p(\mathbf{w})^T J^p(\mathbf{w})\right]^{-1} \nabla E(\mathbf{w}) \ . \tag{3}$$

where $J^p(\mathbf{w})$ is the Jacobian matrix of the error vector $\mathbf{e}^p(\mathbf{w})$ evaluated in $\mathbf{w}$, and I is the identity matrix. The vector error $\mathbf{e}^p(\mathbf{w})$ is the error of the network for pattern p, that is, $\mathbf{e}^p(\mathbf{w}) = \mathbf{t}^p - \mathbf{o}^p(\mathbf{w})$. The parameter μ is increased or decreased at each step. If the error is reduced, then μ is divided by a factor β, and it is multiplied by β in other case. Levenberg-Marquardt performs the steps detailed in Fig. 3. It calculates the network output, the error vectors, and the Jacobian matrix for each pattern. Then, it computes $\Delta\mathbf{w}$ using (3) and recalculates the error with $\mathbf{w} + \Delta\mathbf{w}$ as network weights. If the error has decreased, μ is divided by β, the new weights are maintained, and the process starts again; otherwise, μ is multiplied by β, $\Delta\mathbf{w}$ is calculated with a new value, and it iterates again.

```
InitializeWeights;
while not StopCriterion do
    Calculates e^p(w) for each pattern;
    e1 := sum_{p=1}^{P} e^p(w)^T e^p(w);
    Calculates J^p(w) for each pattern;
    repeat
        Calculates Δw;
        e2 := sum_{p=1}^{P} e^p(w + Δw)^T e^p(w + Δw);
        if (e1 <= e2) then
            μ := μ * β;
        endif;
    until (e2 < e1);
    μ := μ/β;
    w := w + Δw;
endwhile;
```

Fig. 3. Pseudo-code of the LM algorithm

3.3 Genetic Algorithm

A GA [12] is a stochastic general search method. It proceeds in an iterative manner by generating new populations of individuals from the old ones. Every individual is the encoded (binary, real, etc.) version of a tentative solution. The canonical algorithm applies stochastic operators such as selection, crossover, and mutation on an initially random population in order to compute a new population. In *generational* GAs all the population is replaced with new individuals. In *steady-state* GAs (used in this work) only one new individual is created and

it replaces the worst one in the population if it is better. The pseudo-code of the GA we are using here can be seen in Fig. 4. The search features of the GA contrast with those of the BP and LM in that it is not trajectory-driven, but population-driven. The GA is expected to avoid local optima frequently by promoting exploration of the search space, in opposition to the exploitative trend usually allocated to local search algorithms like BP or LM.

```
t := 0;
Initialize:        P(0) := {a_1(0), ..., a_μ(0)} ∈ I^μ;
Evaluate:          P(0) : {Φ(a_1(0)), ..., Φ(a_μ(0))};
while ι(P(t)) ≠ true do        //Reproductive loop
      Select:      P'(t) := s_Θs(P(t));
      Recombine:   P''(t) := ⊗_Θc(P'(t));
      Mutate:      P'''(t) := m_Θm(P''(t));
      Evaluate:    P'''(t) : {Φ(a'''_1(t)), ..., Φ(a'''_λ(t))};
      Replace:     P(t+1) := r_Θr(P'''(t) ∪ Q);
      t := t + 1;
endwhile;
```

Fig. 4. Pseudo-code of a Genetic Algorithm

3.4 Hybrid Algorithms

Here, the hybridization refers to the inclusion of problem-dependent knowledge in a general search template [13,14]. We can distinguish two kinds of hybridization: *strong* and *weak* hybridization. In the first one, the knowledge is included using specific operators or representations. In the latter, several algorithms are combined somehow. In this last case, an algorithm can be used to improve the results of another one separately or it can be used as an operator of the other.

The hybrid algorithms that we use in this work are combinations of two algorithms (weak hybridization), where one of them acts as an operator in the other. We combine a GA with the BP algorithm (GABP), and a GA with LM (GALM). In both cases the problem-specific algorithm (BP and LM) is used as a mutation-like operation of the general search template (GA). Therefore, GAxx is a GA (Fig. 4) in which the mutation has been replaced by the "xx" algorithm that is applied with probability p_t.

4 Empirical Study

After discussing the algorithms, we present in this section the experiments performed and their results. The benchmark for training and the parameters of the algorithms are presented in the next subsection. The analysis of the results is shown in Subsection 4.2.

4.1 Computational Experiments

We tackle three classification problems. These problems consist in determining the class that a certain input vector belongs to. Each pattern from the training pattern set contains an input vector and its desired output vector. These vectors are formed by real numbers. However, in classification problems, the output of the network must be interpreted as a class. Such interpretation can be performed in different ways [8]. One of them consists in assigning an output neuron to each class. When an input vector is presented to the network, the network response is the class associated with the output neuron with the larger value. This method is known as *winner-takes-all* and it is employed in this work.

The instances solved here belong to the PROBEN1[1] benchmark [8]: Cancer, Diabetes, and Heart. We now briefly detail them:

- **Cancer**: Diagnosis of breast cancer. Classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. There are 699 examples that were obtained by Dr. William H. Wolberg at the University of Wisconsin Hospitals, Madison [15,16,17,18].
- **Diabetes**: Diagnose diabetes of Pima Indians. Based on personal data and the results of medical examinations, decide whether a Pima Indian individual is diabetes positive or not. There are 768 examples from the *National Institute of Diabetes and Digestive and Kidney Diseases* by Vincent Sigillito [19].
- **Heart**: Predict heart disease. Decide whether at least one of four major vessels is reduced in diameter by more than 50%. This decision is made based on personal data and results of medical examinations. There are 920 examples from four different sources: Hungarian Institute of Cardiology in Budapest (Andras Janosi, M.D.), University Hospital of Zurich in Switzerland (William Steinbrunn, M.D.), University Hospital of Basel in Switzerland (Mathhias Pfisterer, M.D.), V.A. Medical Center of Long Beach and Cleveland Clinic Foundation (Robert Detrano, M.D., Ph.D.) [20,21].

The structure of the MLP used for any problem accounts for three layers (input-hidden-output) having six neurons in the hidden layer. The number of neurons in the input and output layers depends on the concrete instance. The activating function of the neurons is the sigmoid function. Table 1 summarizes the network architecture for each instance.

To evaluate an ANN, we split the pattern set into two subsets: the training one and the test one. The ANN is trained with all the algorithms by using the training pattern set, and then it is evaluated on the unseen test pattern set. The training set for each instance is approximately made of the first 75% of the examples, while the last 25% constitutes the test set. The exact number of patterns for each instance is presented in Table 1 to ease future comparisons.

After presenting the problems, we now turn to describe the parameters for the algorithms (Table 2). To get the parameters of the pure algorithms we performed

[1] Available from `ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz`.

Table 1. MLP architecture and patterns distribution for all instances

Instance	Architecture	Patterns	
		Training	Test
Cancer	9 - 6 - 2	525	174
Diabetes	8 - 6 - 2	576	192
Heart	35 - 6 - 2	690	230

some preliminary experiments and defined those with the best results. The hybrid algorithms GABP and GALM use the same parameters as their elementary components. However, the mutation operator of the GA is not applied; instead, it is replaced by BP or LM, respectively. The BP and LM are applied with an associated probability p_t only to one individual generated after recombination at each iteration. When applied, BP/LM only performs one single *epoch*.

Table 2. Parameters for the algorithms

		BC	DI	HE
BP	Epochs	1000	1000	500
	η	0.01	0.01	0.001
LM	Epochs	1000	1000	500
	μ	0.001	0.001	0.001
	β	10	10	10
GA	Population size	64		
	Selection	Roulette (2 inds.)		
	Recombination	SPX ($p_c = 1.0$)		
	Mutation	Bit-Flip ($p_m = 1/lenght$)		
	Replacement	Elitist		
	Stop criterion	1064 evals.		
GAxx	p_t	1.0	1.0	0.5
	Epochs of xx	1	1	1

As to the representation of the individuals, the weights are encoded as binary vectors. These vectors allocate 16 bit substrings to represent a real value in the interval $[-1, +1]$. The weights associated to any link arriving to a neuron (and the neuron bias) are placed together in the chromosome.

Finally, we need to define a fitness function to guide the search of the GA (either pure or hybrid). The fitness function (to be maximized) is the inverse of the SEP for the training set.

4.2 Analysis of the Results

In this section we present the results obtained after the application on the three instances of the five algorithms. We report the mean and the standard deviation of the CEP for the test pattern set after performing 50 independent runs. Table 3 and Fig. 5 show the results.

Table 3. Results of the ANN training

CEP(%)		BP	LM	GA	GABP	GALM
Cancer	$\overline{\mathbf{x}}$	0.91	3.17	16.76	1.43	0.02
	$\sigma_{\mathbf{n}}$	0.28	1.29	6.15	4.87	0.11
Diabetes	$\overline{\mathbf{x}}$	21.76	25.77	36.46	36.46	28.29
	$\sigma_{\mathbf{n}}$	0.38	3.26	0.00	0.00	1.15
Heart	$\overline{\mathbf{x}}$	27.41	34.73	41.50	54.30	22.66
	$\sigma_{\mathbf{n}}$	1.48	3.68	14.68	20.03	0.82

A first conclusion is that the GA obtains always a higher CEP than BP, LM and the hybrids (except for Heart and GABP). This is not a surprising fact, since the GA performs a rather explorative search in this kind of problems. BP is slightly more accurate than LM for all the instances, what we did not expect after the accurate behavior of LM in other studies.

With respect to the hybrid algorithms, the results do confirm our hypothesis of work: GALM is more accurate than GABP. In fact, this is noticeable since BP performed better than LM. Of course, we are not saying that this holds for any ANN training problem. However, we do state a clear claim after these results, i.e., GABP has received "too much" attention from the community, while maybe GALM could have worked out lower error percentages. To help the reader we also display these results in a graph in Fig. 5.

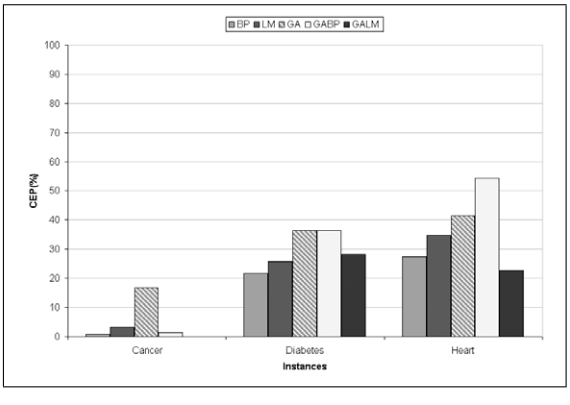

Fig. 5. Comparison among the algorithms (CEP)

We have traced the evolution of each algorithm for the Cancer instance to better explain how the different algorithms work (Fig. 6). We measure the SEP of the network in each epoch of the algorithm. For population-based algorithms (GA, GABP and GALM) we trace the SEP of the best fitness network. Each trace line represents the average SEP over 50 independent runs. We can observe that LM is the faster algorithm, followed by BP, what confirms intuition on

the velocity of local search compared to GAs and hybrids. BP an LM clearly stagnate before 200 epochs in a solution. The GA is the slowest algorithm, and its hybridization with BP, and especially with LM, shows an acceleration of the evolution. An interesting observation is that the algorithms with the lowest SEP (BP and LM) do not always get the lowest CEP (best classification) for the test patterns. For example, GALM, which exhibits the lowest CEP, has only a modest value of SEP in the training process. This is due to the overtraining of the network in the BP and the LM algorithms, and confirms the necessity of reporting both, ANN errors and classification percentages in this field of research.

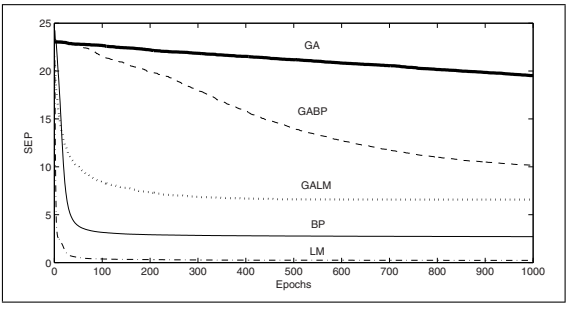

Fig. 6. Average evolution of SEP for the algorithms on the Cancer instance

There are many interesting works related to neural network training that also solve the instances tackled here. But unfortunately, some of the results are not comparable with ours, because they use a different definition of the training and test sets; this is why we consider a capital issue to adhere to any standard way of evaluation like the one proposed by Prechelt [8]. However, we did find some works for meaningful comparisons.

For the Cancer instance we find that the best mean CEP [22] is 1.1%, which represents a lower accuracy compared to our 0.02% obtained with the GALM hybrid. In [23], a CEP close to 2% for this instance is achieved, while our GALM is one hundred times more accurate. The mentioned work uses 524 patterns for the training set and the rest for the test set, that is, almost exactly our configuration with only one pattern changed (a minor detail), and therefore the results can be compared. The same occurs for the work of Yao and Liu [6], where their EPNet algorithm works out neural networks of a lower quality (1.4% of CEP).

For the Diabetes instance, a CEP of 30.11% is reached in [24] (outperformed by our BP, LM, and GALM) with the same network architecture as in our work. In [6] we found for this instance a 22.4% of CEP (outperformed by our BP with a 21.76%).

Finally, in [24] we found a 45.71% of CEP for the Heart instance using the same architecture. In this case, all our algorithms outperform their CEP measure (except GABP).

In summary, while we have found some of the more accurate results for the three instances, it is still needed to get ahead on other instances, always keeping in mind the importance of reporting results in a standardized manner.

5 Conclusions

In this work we have tackled the neural network training problem with five algorithms: two well-known problem-specific algorithms such as Backpropagation and Levenberg-Marquardt, a general metaheuristic such as a Genetic Algorithm, and two hybrid algorithms combining the Genetic Algorithm with the problem-specific techniques. To compare the algorithms we solve three classification problems from the domain of Medicine: the diagnosis of breast cancer, the diagnosis of diabetes in the Pima Indians, and the diagnosis of heart disease.

Our results show that the problem-specific algorithms (BP and LM) get lower classification error than the genetic algorithm, and thus confirm numerically what intuition can only suggest. The hybrid algorithm GALM outperforms in two of the three instances the classification error of the problem-specific algorithms. This makes GALM look as a promising algorithm for neural network training. On the other hand, many of the classification errors obtained in this work are below those found in the literature, what represents a cutting-edge result. As a future work we plan to add new algorithms to the analysis, and to apply them to more instances, especially in the domain of Bioinformatics.

Acknowledgments. This work has been partially funded by the Ministry of Science and Technology and FEDER under contract TIC2002-04498-C05-02 (the TRACER project, `http://tracer.lcc.uma.es`).

References

1. Alander, J.T.: Indexed Bibliography of Genetic Algorithms and Neural Networks. Technical Report 94-1-NN, University of Vaasa, Department of Information Technology and Production Economics (1994)
2. Rumelhart, D., Hinton, G., Williams, R.: Learning Representations by Backpropagation Errors. Nature **323** (1986) 533–536
3. Cotta, C., Alba, E., Sagarna, R., Larrañaga, P.: Adjusting Weights in Artificial Neural Networks using Evolutionary Algorithms. In Larrañaga, P., Lozano, J., eds.: Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, Kluwer Academic Publishers (2001) 357–373
4. Cantú-Paz, E.: Pruning Neural Networks with Distribution Estimation Algorithms. In Erick Cantú-Paz et al., ed.: GECCO 2003, LNCS 2723, Springer-Verlag (2003) 790–800
5. Alba, E., Aldana, J.F., Troya, J.M.: Full Automatic ANN Design: A Genetic Approach. In Mira, J., Cabestany, J., Prieto, A., eds.: New Trends in Neural Computation, Springer-Verlag (1993) 399–404
6. Yao, X., Liu, Y.: A New Evolutionary System for Evolving Artificial Neural Networks. IEEE Transactions on Neural Networks **8** (1997) 694–713

7. Yao, X.: Evolving Artificial Neural Networks. Proceedings of the IEEE **87** (1999) 1423–1447
8. Prechelt, L.: PROBEN1 — A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical Report 21, Fakultät für Informatik Universität Karlsruhe, 76128 Karlsruhe, Germany (1994)
9. Hagan, M.T., Menhaj, M.B.: Training Feedforward Networks with the Marquardt Algorithm. IEEE Transactions on Neural Networks **5** (1994)
10. McClelland, J.L., Rumelhart, D.E.: Parallel Distributed Processing: Explorations in the Microstructure of Cognition. The MIT Press (1986)
11. Rosenblatt, F.: Principles of Neurodynamics. Spartan Books, New York (1962)
12. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, Michigan (1975)
13. Davis, L., ed.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
14. Cotta, C., Troya, J.M.: On Decision-Making in Strong Hybrid Evolutionary Algorithms. Tasks and Methods in Applied Artificial Intelligence, Lecture Notes in Artificial Intelligence **1415** (1998) 418–427
15. Bennett, K.P., Mangasarian, O.L.: Robust Linear Programming Discrimination of Two Linearly Inseparable Sets. Optimization Methods and Software **1** (1992) 23–34
16. Mangasarian, O.L., Setiono, R., Wolberg, W.H.: Pattern Recognition via Linear Programming: Theory and Application to Medical Diagnosis. In Coleman, T.F., Li, Y., eds.: Large-Scale Numerical Optimization. SIAM Publications, Philadelphia (1990) 22–31
17. Wolberg, W.H.: Cancer Diagnosis via Linear Programming. SIAM News **23** (1990) 1–18
18. Wolberg, W.H., Mangasarian, O.L.: Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology. In: Proceedings of the National Academy of Sciences. Volume 87., U.S.A (1990) 9193–9196
19. Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., Johannes, R.S.: Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus. In: Proceedings of the Twelfth Symposium on Computer Applications in Medical Care, IEEE Computer Society Press (1988) 261–265
20. Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., Froelicher, V.: International Application of a New Probability Algorithm for the Diagnosis of Coronary Artery Disease. American Journal of Cardiology (1989) 304–310
21. Gennari, J.H., Langley, P., Fisher, D.: Models of Incremental Concept Formation. Artificial Intelligence **40** (1989) 11–61
22. Ragg, T., Gutjahr, S., Sa, H.: Automatic Determination of Optimal Network Topologies Based on Information Theory and Evolution. In: Proceedings of the 23rd EUROMICRO Conference, Budapest, Hungary (1997)
23. Land, W.H., Albertelli, L.E.: Breast Cancer Screening Using Evolved Neural Networks. In: IEEE International Conference on Systems, Man, and Cybernetics, 1998. Volume 2., IEEE Computer Society Press (1998) 1619–1624
24. Erhard, W., Fink, T., Gutzmann, M.M., Rahn, C., Doering, A., Galicki, M.: The Improvement and Comparison of Different Algorithms for Optimizing Neural Networks on the MasPar MP-2. In Heiss, M., ed.: Neural Computation – NC'98, ICSC Academic Press (1998) 617–623