G-Prop-II: Global Optimization of Multilayer Perceptrons using GAs

P.A. Castillo and V. Rivas
Computer Science Dept.
Avda. Madrid 35
E. 23071 Jaén (Spain)
geneura@kal-el.ugr.es

J.J. Merelo, J. González, A. Prieto and G. Romero
Dept. of Architecture and Computer Technology
Campus de Fuentenueva s/n
E. 18071 Granada (Spain)

Abstract- A general problem in model selection is to obtain the right parameters that make a model fit observed data. For a Multilayer Perceptron (MLP) trained with Backpropagation (BP), this means finding the right hidden layer size, appropriate initial weights and learning parameters. This paper proposes a method (G-Prop-II) that attempts to solve that problem by combining a genetic algorithm (GA) and BP to train MLPs with a single hidden layer. The GA selects the initial weights and the learning rate of the network, and changes the number of neurons in the hidden layer through the application of specific genetic operators. G-Prop-II combines the advantages of the global search performed by the GA over the MLP parameter space and the local search of the BP algorithm.

The application of the G-Prop-II algorithm to several real-world and benchmark problems shows that MLPs evolved using G-Prop-II are smaller and achieve a higher level of generalization than other perceptron training algorithms, such as QuickPropagation or RPROP, and other evolutive algorithms, such as G-LVQ. It also shows some improvement over previous versions of the algorithm.

1 Introduction and State of the Art

Whatever the application using them, artificial neural networks (ANNs) must be designed (structured in layers and correctly connected), their initial parameters established and then trained. BP in its different versions is widely used as training mechanism; examples include QuickProp (QP) [1] and RPROP (Riedmiller and Braun [2]), as well as other evolutionary based approaches [3, 4, 5, 6]. Whichever method is chosen, the training mechanism is an iterative gradient descent algorithm designed to step by step minimize the difference between the actual output vector of the network and the desired output vector. Although this method is used successfully in many fields, especially for pattern recognition, due to its learning ability, it does encounter certain difficulties in practice: 1) the convergence tends to be extremely slow; 2) convergence to the global optimum is not guaranteed; 3) learning constants must be guessed heuristically.

Every method has also got a set of parameters which have to be adjusted depending on the problem to solve, on the type of network (number of hidden units to use) and on the training/test sets. This leaves the problem of automatic MLP parameter setting and optimization open.

There are two ways of approaching the optimization problem parameters: Incremental / decremental (see [7], by Alpaydm et al. for a good review) or genetic algorithms (see [8], by Yao for a good review).

Incremental algorithms, such as *Cascade Correlation* by Fahlman and Lebière [9], the *Tiling and Perceptron Cascade* by Parekh et al. [10], or the methods proposed by Zhang [11] or Rathbun et al. [12], are based on adding hidden neurons to a network of minimum size until it reaches the required precision.

Decremental algorithms (pruning methods), such as those presented by Jasic et al. in [13] and Pelillo et al. in [14], are based on taking a big network and then eliminating hidden layer neurones one by one, to obtain a smaller network. Other approaches set some of the weights to zero as is done in *Optimal Brain Damage* (OBD), by Le Cun et al. [15], and *Optimal Brain Surgeon*, by Hassibi et al. [16]. Hintz-Madsen et al. [17] proposes a method for construction of neural classifiers based on regularization and OBD pruning.

Evolutionary neural networks provide an alternative for this task of controlling the complexity by adjusting the number of weights of the ANN. This is an example of a *hybrid aproach* of a GA and BP to train ANN. The use of GAs to design ANN can be applied in several ways:

Search for the optimal set of weights of a pre-established topology net, as did Topchy et al. in [18], where a GA is used to evolve a population of ANN by encoding the parameters of the hidden layer into binary strings. De Falco et al. propose a method [19] based on an evolutionary approach to provide the optimal set of synaptic weights of the network.

Several authors *search over topology space*, as did White et al. in [20], Miller et al. in [6], or Bebis et al. [21] that proposes the couple between GAs and weight elimination. The methods proposed by Yao and Liu [5] and Castillo et al. [4] combine the search for the optimal set of weights and the search for the optimal topology, using a GA and BP. De Falco et al. present a method [22] based on an evolutionary approach to face the optimization of the design of a neural network architecture and the choice of the best learning method.

Other approaches *search for the optimal learning parameters*, having pre-established the number of neurons and the connectivity between them, as did Merelo et al. in [3], Petridis et al. in [23] or Castillo et al. in [24], where an approach based on Simulated Annealing and BP is presented.

Both decremental and incremental algorithms are gradient descent optimization methods, so they suffer the problem that they may reach the closest local minimum to the search space point where the method began. Evolutionary neural networks are a more efficient way of searching, but still, they search in a subset of all possible parameters.

The aim of this paper is to present an algorithm to tune learning parameters, set the initial weights and hidden layer size of a MLP, based on a GA and BP, that obtains better results than the BP alone. We intend to make use of the capacity of both algorithms: the ability of the GA to find a solution close to the global optimum, and the ability of the BP to tune a solution and reach the nearest local minimum by means of local search from the solution found by the GA. Instead of using a pre-established topology, the population is initialized with different hidden layer sizes, with some specific operators designed to change it. Thus, the GA searches and optimizes the architecture (number of hidden units), the initial weight setting for that architecture and the learning rate for that net.

The remainder of this paper is structured as follows: Section 2 presents the G-Prop-II algorithm, followed by the architecture and genetic operators. Section 3 describes the results obtained, followed by a brief conclusion in Section 4.

2 The genetic algorithm in G-PROP-II

The designed algorithm is specified in the following pseudocode:

1. Generate the initial population with random weight values and hidden layer sizes ranging from 2 to a maximum of 20. This is needed in order to have a diversity of sizes.

2. Repeat for g generations:

   (a) Evaluate the new individuals: train them using the training set and obtain their fitness according to the number of correct classifications on the validation set and the hidden layer size.

   (b) Select the n best individuals in the population, according to their fitness, and apply genetic operators to them.

   (c) Replace the n worst individuals by the new ones.

3. Use the best individual to obtain the testing error.

In G-Prop-II the *fitness function* is given by the number of hits when carrying out the test after the training, and in the case of two individuals with identical classification error, the best is the one that has a hidden layer with fewer neurons. This implies greater speed when training and classifying and facilitates its hardware implementation. The classification accuracy or number of hits is obtained by dividing the number of hits between the total number of examples in the testing set.

A *steady state* [25] algorithm was used because it was empirically found to be faster at obtaining solutions than other selection algorithms. For each generation, the best n individuals of the population, those whose fitness is highest, are chosen to reproduce, using the genetic operators. The offspring replace the n worst individuals of the current generation.

In principle, evolved MLPs should be codified into chromosomes to be handled by the genetics operators of the GA. But G-Prop-II uses no binary codification; instead, the initial parameters of the network are evolved using specific genetic operators (see below), such as mutation, crossover and the substitution, addition and elimination of hidden neurons, which is made possible by the **EO** (Evolvable|Evolutionary Objects) library philosophy (see next section): any object with a fitness can be evolved. Moreover, this agrees with the spirit of Z. Michalewicz: $GA + DS = EP$ (GA plus Data Structures equal Evolution Programs) [26] . The genetic operators act directly on the ANN object, but only *initial weights* and the *learning rate* are subjected to evolution, not the weights obtained after training (a clone of the MLP is created to compute its fitness function, thus the initial weights remain unchanged in the original MLP). The "genetic atom" is a hidden layer neuron; most operators treat hidden layer neurons and weights to and from it as an unit.

Five genetic operators are used to change MLP. Besides the percentage of mutation and the number of crossing points, the application priority for each operator has to be specified to indicate the number of individuals generated by each genetic operator. The tests have been done using a higher priority for mutation and the same level for the remaining operators.

The *mutation* operator modifies the weights of certain neurons, at random, depending on the application rate. It is based on the algorithm presented in [27], which modifies the weights of the network after each epoch of network training, adding or subtracting a small random number that follows *uniform* distribution with the interval $[-0.1, 0.1]$. The learning rate is modified by adding a small random number that follows *uniform* distribution in the interval $[-0.05, 0.05]$. This operator was used with an application probability of 40%, that is 40% of weights are changed, which was found empirically to obtain better results than did lower probabilities.

The *CrossOver* interchanges hidden layer neurons between both parents. The learning rate is swapped between the two nets.

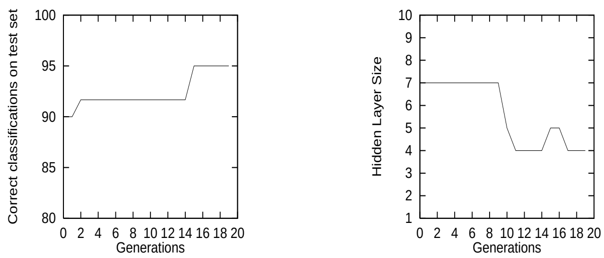
The *Addition* operator and the following (Elimination) attempt to solve one of the main problems of BP and its variants: the difficulty in guessing the number of the hidden layer neurons. By adding hidden neurons it is not necessary to set the size of the GA search space. This operator is intended to perform incremental learning: it starts with a small structure and increments it, if neccesary, adding new hidden units.

The *Elimination* operator eliminates one hidden neuron at random. This operator is intended to perform decremental learning: it prunes certain nodes to obtain better results in generalization and a smaller network [13, 14, 21]. Thus, to a certain extent, the networks are prevented from growing too much.

The *Substitution* operator is applied at a low priority and replaces one hidden layer neuron at random by a new one, initialized with random weights. This operator may be considered a kind of mutation that affects only one gene.

The algorithm was run for a fixed number of generations. When evaluating each individual of the population to obtain its fitness a limit of epochs was established.

We have used the BP variant known as the perceptron training algorithm *QuickProp* [1]. This algorithm is one of the best avoiding local minima, one of the problems of BP. This algorithm, together with GA, improves the probability of avoiding local minima.



a b

Figure 1: Evolution of the classification error of G-Prop-II in terms of hit percentages and hidden layer size during a typical run on the *DNA Helicases* set. At generation 2, a MLP with better classification accuracy but more hidden neurons (7) is found; at generation 10, a perceptron with the same accuracy and fewer hidden neurons (5) is found; at generation 11, a MLP with the same accuracy and 4 hidden neurons is found; at generation 15, a MLP with better classification accuracy but more hidden neurons (5) is found; later on during the simulation (generation 17) a perceptron with the same accuracy but fewer hidden neurons (4) takes its place.

3 Experiments and results

We used **EO** library as a toolbox to develop G-Prop-II, due to the facility that this library offers to evolve any object with a fitness function and use anything as genetic operator. It is a C++ toolbox which defines interfaces for many classes of algorithms used in evolutionary computation and, at the same time, provides some examples that use those interfaces. It is available at *http://geneura.ugr.es/˜jmerelo/EO.html*. One of the objectives of using EO is to make it easier to reproduce results, which is why the source for all objects evolved with EO should also be made available, together with the articles that mention them. G-Prop-II is available at *http://krypton.ugr.es/˜pedro/G-Prop.htm*

First, we show the dynamics of G-Prop-II as a genetic algorithm and then compare it with other methods. The evolution of classification error and hidden layer size during a typical run are shown in figure 1. As is shown in the figure, each of these quantities is usually optimized in turn: classification error is optimized first, since it is given higher priority in fitness evaluation; then, while keeping the same classification accuracy, G-Prop-II optimizes size.

In the run shown here, in generation 6, a MLP with better classification accuracy but more hidden neurons (7) is found; at generation 10, a perceptron with the same accuracy and fewer hidden neurons (5) is found; at generation 11, a MLP with the same accuracy and 4 hidden neurons is found; at generation 15, a MLP with better classification accuracy but more hidden neurons (5) is found; later on during the simulation (generation 17) a perceptron with the same accuracy but fewer hidden neurons (4) is selected as the winner.

G-Prop-II was run 20 times with the same parameter settings and a different random initialization for each benchmark used.

The tests were applied as follows: each data set was divided into three disjoint parts, for training, validating and testing. Thus, in order to obtain the fitness of an individual, the MLP is trained with the training set and its fitness is established from the classification error with the validating set. Once the GA is finished (when it reaches the limit of generations), the classification error with the testing set is calculated: this is the result shown.

To obtain the results with the program that implements QP, the methodology used was to train as many MLPs as were trained by G-Prop-II on a run (about 200 MLPs), using the *same topology* and *learning parameter* of the best net found by the proposed method, for a particular benchmark. Then the validating set is used to obtain the validating error for each MLP; once the best MLP is found, the testing error is obtained on the testing set.

Cancer. This dataset is from the UCI machine learning dataset "Wisconsin breast cancer database". This breast cancer database was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg [28]. An exhaustive report, by Prechelt, on this dataset (and others) is given in [29]. Each sample has 10 attributes plus the class attribute: Sample code number, Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, Mitoses, Class (0 for benign, 1 for malignant). The class distribution in the original set is the following: 65.5% Benign and 34.5% Malignant.

DNA Helicases. This is a problem of electron microscopy image classification. The object of the classification is to study the structure of a representative hexametric helicase: the large T antigen of Simian Virus 40. When observed in the electron microscope, large T antigen preparations mainly show either a characteristic roughly circular view with a stain penetrating region in the centre (which we will call a top view) or a view with a rectangular shape (side view). This

Cancer		Error $\pm\ \sigma$	Size $\pm\ \sigma$ (params.)	Learning P. $\pm\ \sigma$
Cancer 1	QP	4 ± 2	3 (33)	0.072
	G-Prop-II	1.0 ± 0.5	3 ± 1 (33)	0.039 ± 0.017
	G-Prop	1.0 ± 0.5	3.2 ± 0.8 (33)	-
	SA-Prop	1.1 ± 0.5	6 (66)	0.18 ± 0.09
	Prechelt	1.149	4+2 (48)	-
Cancer 2	QP	7 ± 2	7 (77)	0.032
	G-Prop-II	4 ± 1	7 ± 3 (77)	0.035 ± 0.014
	G-Prop	4.4 ± 0.4	7 ± 2 (77)	-
	SA-Prop	4.2 ± 0.6	12 (132)	0.12 ± 0.09
	Prechelt	5.747	8+4 (112)	-
Cancer 3	QP	4 ± 1	4 (44)	0.034
	G-Prop-II	3.0 ± 0.9	4 ± 2 (44)	0.038 ± 0.018
	G-Prop	3.0 ± 0.7	4 ± 2 (44)	-
	SA-Prop	3.2 ± 0.9	8 (88)	0.11 ± 0.08
	Prechelt	2.299	4+4 (60)	-

Table 1: Results of evaluating QP and G-Prop-II for the Cancer problem, and the results obtained by Prechelt [29] and Castillo et al. [4, 24]. This table shows the average error rate, the average size of nets as the number of hidden units, and the learning parameter found. The learning rate used to obtain the results with QP is the value of that found by G-Prop-II in the best net. The hidden layer size is expressed, enclosed in parentheses, in terms of number of parameters of the net, that is, the number of weights of the net.

DNA Helicases		Error $\pm\ \sigma$	Size $\pm\ \sigma$ (params.)	Learning P. $\pm\ \sigma$
QP		6 ± 3	7 (189)	0.035
G-Prop-II		4 ± 2	1.9 ± 0.7 (54)	0.05 ± 0.02
G-Prop		6.10 ± 3.01	7 ± 4 (189)	0.04 ± 0.03
SA-Prop		5 ± 3	7 (189)	0.16 ± 0.06
G-LVQ	20 generations	18 ± 2	2.3 ± 0.4 (59)	-
	50 generations	13 ± 3	2.2 ± 0.3 (57)	-
	100 generations	15 ± 5	2.2 ± 0.3 (57)	-

Table 2: Results of evaluating QP and G-Prop-II on the classification of different views of large T antigen of Simian Virus 40 (error on test set, hidden layer size and the learning parameter found), and the results obtained with G-LVQ [30], G-Prop [4] and SA-Prop [24]. The hidden layer size is expressed, enclosed in parentheses, in terms of number of parameters of the net, that is, the number of weights of the net.

In general, G-Prop-II obtains MLPs with a lower generalization error than other methods (SA-Prop [24] and RPROP), except in Cancer 3, for which Prechelt [29] presents similar results, taking into account that he does not mention standard desviation (although G-Prop-II obtains smaller sized networks, 4 neurons, with a similar classification error, 3.0 ± 0.9). The results obtained with G-Prop-II are very similar to those obtained with G-Prop [4] in this particular problem; however, it should be noticed that G-Prop-II searches for the learning constant, while G-Prop needs an extra effort to guess a suitable learning constant to train the networks.

For the *DNA helicases* problem, the GA was executed for 20 generations on a population of 20 individuals, with a probability of mutation of 0.4 and 2 crossing points for the crossover operator. Each generation, 50% of the population is replaced by new individuals. The individuals, MLP with 25 inputs, 2 outputs and a number of hidden units between 2 and 15, were evaluated with 100 epochs and an initial learning coefficient of 0.03. Each execution with these parameters took several minutes.

Table 2 shows the results obtained with both algorithms and compares them to the ones shown in [30].

In the *DNA helicases* problem, it is evident that G-Prop-II outperforms other methods: G-LVQ [30] takes arround 20 generations to achieve an error of 18 ± 2, while G-Prop-II achieves an error of 4 ± 2; and compared to G-Prop or SA-Prop, the proposed method obtains better results in errors and net sizes.

The optimal values of the learning parameters are between 0.034 and 0.05. This result is not generalizable to all kind of perceptrons, since it is the change of this value and the adaption of the initial weights that minimizes the error rate, as can be seen in the results.

4 Conclusions

This paper presents G-Prop-II, an algorithm to train MLP based on GA and BP. Experiments prove that the proposed method achieves better results than other non-evolutionary variants and besides, obtains network size and learning parameters as a result. It is also an improvement over previous variants of the method.

In particular, the proposed algorithm (G-Prop-II) obtains a much higher degree of generalization (that is, error on a previously unseen test set) than that achieved by other BP algorithms such as QP or RPROP, minimizing the number of hidden units as the second optimization criterion. This is achieved by evolving the initial weights and the learning parameters, and by using operators that alter the ANN size. This strategy attempts to avoid Lamarckism (i.e., that individuals inherit the trained weights from their parents). It is also another step towards full automation of MLP design, since it also sets a value for the learning constant.

Several benchmarks (cancer 1, 2 and 3, and DNA helicases) have been used to test the proposed algorithm and to compare it with others, [29, 30, 4, 24]. The results show that the GA obtains a MLP whose classification accuracy is better than that obtained by training a MLP using only conventional procedures.

Future work will extend the presented method and will include the development of new genetic operators and the improvement of those described here, to perform incremental learning (*pruning* and *addition* operators) from the approach presented in [9] and [14]. The QP algorithm will also be used as an operator in the GA, as suggested in [31] and [5], and the presented algorithm applied to solve other real world problems.

5 Acknowledgements

This work has been supported in part by the CICYT project BIO96-0895 (Spain), the DGICYT project PB-95-0502 and the FEDER I+D project 1FD97-0439-TEL1.

Bibliography

[1] S.E. Fahlman. Faster-Learning Variations on Back-Propagation: An Empirical Study. *Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann*, 1988.

[2] M. Riedmiller and H. Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *In Ruspini, H., (Ed.) Proc. of the ICNN93, San Francisco, pp. 586-591*, 1993.

[3] J.J. Merelo; M. Patón; A. Canas; A. Prieto and F. Morán. Optimization of a competitive learning neural network by genetic algorithms. *IWANN93. Lectures Notes in Computer Science, vol. 686, 185-192*, 1993.

[4] P.A. Castillo; J. González; J.J. Merelo; V. Rivas; G. Romero; A. Prieto. G-Prop: Global Optimization of Multilayer Perceptrons using GAs. *Submitted to Neurocomputing*, 1998.

[5] Xin Yao and Yong Liu. Towards Designing Artificial Neural Networks by Evolution. *Applied Mathematics and Computation, 91(1):83-90*, 1998.

[6] G.F. Miller; P.M Todd and S.U. Hegde. Designing neural networks using genetic algorithms. *In J.D.Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, pages 379-384, San Mateo, 1989*, 1989.

[7] Ethem Alpaydm. GAL: Networks that grow when they learn and shrink when they forget. *International Journal of Pattern Recognition and Artificial Intelligence , 8(1), 391-414*, 1994.

[8] Xin Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8(4):539–67, April 1993.

[9] S. Fahlman and C. Lebière. The Cascade-Correlation Learning Architecture. *Neural Information Systems 2. Touretzky, D.S. (ed) Morgan-Kauffman, 524-532*, 1990.

[10] R. Parekh; J. Yang and Honavar V. Constructive neural network learning algorithms for multi-category real-valued pattern classification. Technical Report ISU-CS-TR-97-06, Department of Computer Science, Iowa State University., 1997.

[11] J. Zhang; A. Morris. A sequential learning approach for single hidden layer neural networks. *Neural Networks 11 (1997) 65-80*, 1997.

[12] T. Rathbun; S. Rogers; M. DeSimio; M. Oxley. MLP iterative construction algorithm. *Neurocomputing 17 (1997) 195-216*, 1997.

[13] T. Jasic and H. Poh. Analysis of Pruning in Backpropagation Networks for Artificial and Real World Mapping Problems. *IWANN95. Lectures Notes in Computer Science, vol. 930, 239-245*, 1995.

[14] M. Pelillo and A. Fanelli. A Method of Pruning Layered Feed-Forward Neural Networks. *IWANN93. Lectures Notes in Computer Science, vol. 686, 278-283*, 1993.

[15] Y. Le Cun; J.S. Denker; S.A. Solla. Optimal brain damage. *Neural Information Systems 2. Touretzky, D.S. (ed) Morgan-Kauffman, pp. 598-605*, 1990.

[16] B. Hassibi; D.G. Stork; G. Wolff; T. Watanabe. Optimal Brain Surgeon: extensions and performance comparisons. *In NIPS6, pp. 263-270*, 1994.

[17] Mads Hintz-Madsen; Lars Kai Hansen; Jan Larsen; Morten With Pedersen; Michael Larsen. Neural classifier construction using regularization, pruning and test error estimation. *Neural Networks 11, 1659-1670*, 1998.

[18] A.P. Topchy; O.A. Lebedko; V.V. Miagkikh. Fast learning in multilayered neural networks by means of hybrid evolutionary and gradient algorithms. *to appear in Proc. of IC on Evolutionary Computation and Its Applications, Moscow*, 1996.

[19] I. De Falco; A. Iazzetta; P. Natale; E. Tarantino. Evolutionary Neural Networks for Nonlinear Dynamics Modeling. *PPSN98. Lectures Notes in Computer Science, vol. 1498, 593-602*, 1998.

[20] David White and Panos Ligomenides. GANNet: A Genetic Algorithm for Optimizing Topology and Weights in Neural Network Design. *IWANN93. Lectures Notes in Computer Science, vol. 686, 322-327*, 1993.

[21] G. Bebis; M. Georgiopoulos; T. Kasparis. Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization. *Neurocomputing 17 (1997) 167-194*, 1997.

[22] I. De Falco; A. Della Cioppa; A. Iazzetta; P. Natale; E. Tarantino. Optimizing Neural Networks for Time Series Prediction. *Third World Conference on Soft Computing (WSC3), June 1998.*, 1998.

[23] V. Petridis; S. Kazarlis; A. Papaikonomu and A. Filelis. A hybrid genetic algorithm for training neural networks. *Artificial Neural Networks, 2, 953-956*, 1992.

[24] P.A. Castillo; J. González; J.J. Merelo; V. Rivas; G. Romero; A. Prieto. SA-Prop: Optimization of Multilayer Perceptron Parameters using Simulated Annealing. *Submitted to IWANN99*, 1998.

[25] D. Whitley. *The GENITOR Algorithm and Selection Presure: Why rank-based allocation of reproductive trials is best.* in J.D. Schaffer (Ed.), Proceedings of The Third International Conference on Genetic Algorithms, Morgan Kauffmann, Publishers, 116-121, 1989.

[26] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs , Third, Extended Edition*. Springer-Verlag, 1996.

[27] Werner Kinnebrock. Accelerating the standard backpropagation method using a genetic approach. *Neurocomputing, 6, 583-588*, 1994.

[28] O. L. Mangasarian; R. Setiono and W.H. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. *Large-scale numerical optimization, Thomas F. Coleman and Yuying Li, editors, SIAM Publications, Philadelphia 1990, pp 22-30*, 1990.

[29] 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.

[30] J.J. Merelo; A. Prieto; F. Morán; R. Marabini and J.M Carazo. Automatic Classification of Biological Particles from Electron-microscopy Images Using Conventional and Genetic-algorithm Optimized Learning Vector Quantization. *Neural Proccessing Letters 8: 55-65, 1998*, 1998.

[31] D.J. Montana; L. Davis. Training feedforward neural networks using genetic algorithms. *Proc. 11th Internat. Joint Conf. on Artificial Intelligence, 762-767*, 1989.