



ELSEVIER

Neurocomputing 35 (2000) 149–163

NEUROCOMPUTING

www.elsevier.com/locate/neucom

# G-Prop: Global optimization of multilayer perceptrons using GAs

P.A. Castillo<sup>a,\*</sup>, J.J. Merelo<sup>a</sup>, A. Prieto<sup>a</sup>, V. Rivas<sup>b</sup>, G. Romero<sup>a</sup>

<sup>a</sup>*Departamento de Arquitectura y Tecnología de Computadoras, Facultad de Ciencias, Campus Fuentenueva, s/n, E. 18071, Universidad de Granada, Spain*

<sup>b</sup>*Departamento de Informática de la Universidad de Jaén, Escuela Politécnica Superior, Universidad de Jaén, Avda. Madrid, 35, E. 23071, Jaén, Spain*

Received 7 June 1999; accepted 13 April 2000

---

## 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 back-propagation (BP), this means finding appropriate layer size and initial weights. This paper proposes a method (G-Prop, genetic backpropagation) 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 changes the number of neurons in the hidden layer through the application of specific genetic operators. G-Prop 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 algorithm to several real-world and benchmark problems shows that MLPs evolved using G-Prop are smaller and achieve a higher level of generalization than other perceptron training algorithms, such as Quick-Propagation or RPROP, and other evolutive algorithms, such as G-LVQ. © 2000 Elsevier Science B.V. All rights reserved.

*Keywords:* Genetic algorithms; Neural networks; Optimization; Learning; Generalization

---

## 1. Introduction and state of the art

Using artificial neural networks (ANNs) requires establishing the structure in layers and connections between them, the parameters (such as initial weights) and a set of learning constants. As training mechanism, an iterative gradient descent algorithm

---

\* Corresponding author.

E-mail address: todos@geneura.ugr.es (P.A. Castillo).

designed to step by step minimize the difference between the actual output vector of the network and the desired output vector, such as BP in its different versions (like, for instance QuickProp by Fahlman et al. [12], and RPROP by Riedmiller and Braun [37,38]) is widely used as training mechanism, as well as other evolutionary approaches [6,23,27,29,47]. However, even as these BP methods are successfully used in many fields, especially for pattern recognition, due to its learning ability, it does encounter certain difficulties in practice: (1) convergence tends to be extremely slow and very dependent on the initial weights; (2) convergence to the global optimum is not guaranteed; (3) learning constants and hidden layer size must be guessed heuristically or through systematic tests.

Convergence can be speeded up through the use of gradient following techniques [13,38], but (2) and (3) are more difficult to tackle, specially since the learning parameters and network size depend on the problem to solve. Thus the problem of automatic MLP parameter setting and optimization remains an open question.

There are two ways of approaching the optimization of BP parameters for certain problem: incremental/decremental (see Alpaydim et al. [1] for an interesting review) or genetic algorithms (see either Yao [45,46] or Balakrishnan et al. [2] for an interesting review).

- *Incremental algorithms*, such as *Cascade Correlation* by Fahlman and Lebière [12], and the *Tiling and Perceptron Cascade Algorithm* presented by Parekh et al. in Ref. [32], are based on adding hidden neurons to a network of minimum size until it reaches the required precision. A sequential orthogonal approach to the building and training of single hidden layer neural networks is also presented by Zhang in [48]. This method starts with a single hidden neuron and increases its number until the error is sufficiently small, or as in the method proposed by Rathbun et al. [36] constructs MLP from scratch, adding hidden units until the algorithm achieves the required accuracy, there is also a method to prune the network. One problem of these methods is that once the hidden neurons have been added they cannot be suppressed to reduce the size (the redundant information stored in the weights is never eliminated) and huge ANNs are usually obtained. Furthermore, since weights of existing neurons are frozen, added ones are usually inefficient feature detectors, so the algorithm has to add even more units to improve the obtained results. In general, adding new units lead to overfitting: Hwang et al. demonstrated [18] that the Cascade Correlation algorithm does not generalize well in some classification and regression problems.
- *Decremental algorithms* such as those presented by Jasic et al. [19] and Pelillo et al. [33], usually called pruning methods, are based on taking a network with an initially high number of connections, and then eliminating them one by one, to obtain a smaller network but with the same (or better) classification ability. These algorithms work by searching redundant noncontributing or duplicate nodes. After pruning, ANN must usually be re-trained. The problem with these methods is that they start with excessively big networks, which slows down training. However, depending on the selection criteria used to select the neurons to be pruned, it is

possible to obtain a good solution, if the right units to eliminate and the elimination order are guessed correctly. Another approach is to set some of the weights to zero as is done in *Optimal Brain Damage*, by Le Cun et al. [7], and *Optimal Brain Surgeon*, by Hassibi et al. [16]. The problem presented by this method is that near co-linearities can mean that if one weight is set to zero, the standard errors for the others are drastically reduced, so it may be unsafe to set more than one weight to zero at a time. 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* provide an alternative for this task of controlling the complexity by adjusting the number of weights of the ANN. They are an example of a *hybrid approach* of a GA and BP to train ANN. The Baldwin effect may arise in evolutionary computation when a genetic algorithm is used to evolve a population of individuals that also employ a local search algorithm [17]. In computational terms, in a first step of the Baldwin effect, local search smooths the fitness landscape, which can facilitate evolutionary search. In a second step, as more optimal individuals arise in the population, there is selective pressure for reduction in local search, driven by the intrinsic costs associated with the search. GAs can be applied to design ANN in several ways [25,45].

- *Search for the optimal set of weights* of a pre-established topology net, as did Topchy et al. in [40], where a GA is used to evolve a population of ANN by encoding the parameters of the hidden layer into binary strings. This approach uses a pre-established connectivity and number of neurons, and it does not allow to search other topologies. De Falco et al. propose a method [9] based on an evolutionary approach to provide the optimal set of synaptic weights of the network. Some authors use binary encoding of weights into individuals [44] while others use dynamic encoding, like Gray encoding [39]. Keesing and Stork [20] proposed a method that combines a GA and neural networks to solve a simple pattern recognition problem, which takes advantage of the Baldwin effect [3] to increase the rate of evolution: too little of too much learning leads to poorer evolution than does an intermediate amount of learning. Main problem with this approach is that it concentrates on optimizing only a part of the neural net, disregarding the rest: hidden layer size and learning constants.
- *Search over topology space*, as did White et al. in [42], where a GA, using a population initialized with different hidden layer sizes, is presented. In this approach, each individual is a complete neural network, and the allele is a hidden or output node with its associated input links. The main problem of this method is that it only search using sizes obtained during the initialization phase. Another method to determine the architecture is presented by Miller et al. in [29], where the network connection structure is mapped onto a binary adjacency matrix called *Miller—Matrix* (MM) describing ANN architecture, but matrix representation of the network: it can generate incorrect network structures (feedback connections) and very long codes for larger networks and it works only with fixed size nets. Bebis

et al. [4] propose the coupling of GAs with weight elimination [41] to search the architecture by pruning oversized networks. This method begins with big networks and, using the GA, searches for the best set of parameters to apply the weight elimination method. The method proposed by Yao and Liu [23,47] combines the search for the optimal set of weights and the search for the optimal topology of modular neural nets. They use a GA to evolve individual modules and integrate them in the same evolutionary process. De Falco et al. present a method [8] 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. A problem of using a matrix representation is that some networks can generate big individuals. To avoid this problem, some methods of indirect coding have been proposed, as did Kitano [22], Harp et al. [15], Dodd et al. [10] and Gruau [14].

- *Search for the optimal learning parameters*, including weights, having pre-established the number of neurons and the connectivity between them, as did Merelo et al. [27], for multilayer competitive learning neural nets, where a method that codifies the weights and learning parameters onto chromosomes is presented. Another example is the method presented by Petridis et al. [34] where both weights and learning parameters are represented as bit strings to be evolved. Another approach that searches for the learning parameters of the net, based on Simulated Annealing is proposed by Castillo et al. [6].
- *Genetic approaches* that modify the BP algorithm, as did Kinnebrock in [21], where a mutation operator that adds or subtracts an amount to the weight values after each iteration is presented. The disadvantage of this method is that such modifications do not necessarily improve the net classification error.

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 over a subset of all possible parameters.

The aim of this paper is to present an algorithm that searches over the initial weights and hidden layer size of a MLP, based on a GA and BP. It will be proved that it obtains better results than the BP alone, and it only needs to set the learning constant by hand (it obviously needs to set GA constants, but is robust enough to obtain good results under the default parameter settings). 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. As genetic operators, mutation, multi-point crossover, addition, elimination and substitution of hidden units have been selected. Thus, the GA searches and optimizes the architecture (number of hidden units) and the initial weight setting for that architecture. Unlike other approaches [47], the hidden layer maximum size is not bounded in advance.

G-Prop main contributions to the field of neurocomputing are:

- A GA is applied directly to a population of MLP, instead of some codification of the network. Thus a lineal or binary representation is not needed in order to evolve the population.
- GA is used to modify the initial weights only, while BP is used to train from those weights. This makes a clean division between global and local search.
- The hidden layer size is searched throughout the application of some new GA operators: substitution, addition and elimination. These operators are developed to perform incremental (adding hidden neurons) or decremental (pruning hidden neurons) learning.
- The program does not takes much more time than plain BP to find a better solution.

The remainder of this paper is structured as follows: Section 2 presents the G-Prop 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

The designed algorithm is specified in the following pseudocode:

- (1) Generate the initial population with random weight values and hidden layer sizes uniformly distributed from 2 to a maximum of MAX (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 mate them, using mutation, crossover, addition, elimination and substitution of hidden neurons.
  - (c) Replace the  $n$  worst individuals by the new ones.
- (3) Use the best individual to obtain the testing error.

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. In G-Prop 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 (measured as the number of elements incorrectly classified) 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; besides, it improves generalization.

A *steady-state* [43] 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 mate,

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 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: this is made possible by the evolvable|evolutionary objects (EO) library philosophy (see next section), according to which any object with a fitness can be evolved. Moreover, this agrees with the spirit of Michalewicz: Genetic Algorithm plus Data Structures equal Evolution Programs [30]. The genetic operators act directly on the ANN object, but only *initial weights* are subjected to evolution, not the weights obtained after training (a clone of the MLP is created and trained 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 deal with hidden layer neurons and weights to and from it as a 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 [21], 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  $[-0.1, 0.1]$ . 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* operator carries out the multipoint cross-over between two chromosome nets, so that two networks are obtained whose hidden layer neurons are a mixture of the hidden layer neurons of both parents: some hidden neurons along with their *in* and *out* connections, from each parent make one offspring and the remaining hidden neurons make the other one.
- 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 necessary, adding new hidden units. At the same time, it raises the dilemma of overfitting: small networks generalize well, but they are slow learning, whereas big networks are fast learning, but generalize badly [4,5].
- 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 [4,19,33]. 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 with 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* [12]. This algorithm, together with RPROP [37,38] BP variant, is one of the best avoiding local minima (one of the problems of BP) because the magnitude of the change in the weights (the step size) is not a function of the magnitude of the gradient. The QuickProp algorithm, together with GA, improves the probability of avoiding local minima.

### 3. Experiments and results

We used EO library as a toolbox to develop G-Prop, due to the facility that this library offers to evolve any object with a fitness function and use anything as a genetic operator. EO 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 is available at <http://geneura.ugr.es/~pedro/G-Prop.htm>.

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

In the run shown here, in generation 1, a MLP with the same classification accuracy (97.7%) but fewer hidden neurons (3) is found; at generation 2, a perceptron with better classification accuracy (98.2%) but more hidden neurons (4) is found; later on during the simulation, a perceptron with the same accuracy but fewer hidden neurons (3) is selected as the winner.

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

The tests used to assess the accuracy of G-Prop and other must be chosen carefully, because some tests (exclusive-or problem) are not suitable for certain capacities of the BP algorithm, such as generalization [11]. Our opinion, along with Prechelt [35], is that to test an algorithm, at least two real-world problems should be used. This is why comparisons with other GA + NN algorithms are not included in this paper: in other published papers [14,22,29,42], they were applied to toy or non-public available problems.

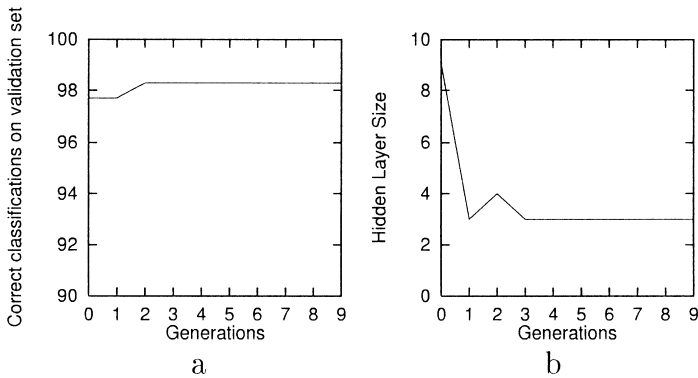


Fig. 1. Evolution of the classification accuracy of G-Prop in terms of hits percentage (a) and of the hidden layer size (b) during a typical run. At generation 1, a MLP with same classification accuracy (97.7%) but fewer hidden neurons (3) is found; at generation 2, a perceptron with better accuracy (98.2%) but more hidden neurons (4) is found; later on during the simulation a perceptron with the same accuracy but fewer hidden neurons (3) takes its place.

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 shown result.

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 on a run (about 120 MLPs), using the *same topology* 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. The results which have an error greater than five times the error average are not taken into account to obtain the final results. The RPROP results, quoted here come from Prechelt [35]. The basic principle of RPROP is to eliminate the influence of the size of the partial derivative on the weight step. As a consequence, only the sign of the derivative is considered to indicate the direction of the weight update.

**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 Wolberg [24]. An exhaustive report, by Prechelt, on this dataset (and others) is given in [35]. 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 hexameric

helicase: the large T antigen of Simian Virus 40 [26]. 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 training/test set has been used in [28] to classify images of the large T antigen of Simian Virus 40. The set consists of 197 examples, divided into the training set (78 examples), validating set (60) and testing set (59). Each example consists of 25 inputs (the 25 blocks of the image to be classified), and the output: 0 is side view and 1 is top view.

The tests were carried out, using the described benchmarks, with our own version of QP (available as part of the G-Prop package) and the proposed method, G-Prop.

For the *Cancer* problem the GA was executed for 10 generations on a population of 20 individuals (neither too many generations nor a very large population, in order to avoid long runs), with a mutation probability of 0.5 (which was found empirically to generate better individuals than a lower probability) and 2 crossing points for the crossover operator. In each generation, 50% of the population is replaced by new individuals (to prevent the population from remaining too static and perhaps not evolving at all). The individuals, MLP with 9 inputs, 2 outputs and a number of hidden units between 2 and 20, were evaluated with 100 epochs and a suitable learning coefficient for this problem, where a MLP can easily fall into a local minimum (0.007). Each execution with these parameters took several minutes.

To show the improvement of a GA over purely random search, the networks were trained for 1000 epochs. The results obtained for the first test (% of error in test), compared with those obtained by Prechelt [35], are shown in Table 1.

Table 1  
Results of evaluating QP and G-Prop for the Cancer problem<sup>a</sup>

Cancer	Error% $\pm$ Std Dev	Hidden units $\pm$ Std Dev (Parameters)
Cancer 1	QP	17 $\pm$ 17
		13 $\pm$ 16
	G-Prop	1.0 $\pm$ 0.5
	Prechelt	1.149
		3 (33)
		9 (99)
		3.2 $\pm$ 0.8 (35.2)
		4 + 2 (48)
Cancer 2	QP	15 $\pm$ 14
		27 $\pm$ 13
	G-Prop	4.4 $\pm$ 0.4
	Prechelt	5.747
		7 (77)
		12 (132)
		6.7 $\pm$ 2.3 (73.7)
		6 + 6 (112)
Cancer 3	QP	13 $\pm$ 15
		18 $\pm$ 17
	G-Prop	3.0 $\pm$ 0.7
	Prechelt	2.299
		4 (44)
		8 (88)
		4.3 $\pm$ 1.7 (47.3)
		4 + 4 (60)

<sup>a</sup>RPROP results are taken from [35]. This table shows the average error rate and the average size of nets, as the number of parameters (number of weights of the MLP) and as the number of hidden units.

In general, G-Prop obtains MLPs with a slightly lower generalization error than other methods (QP and RPROP), except in Cancer 3, for which Prechelt [35], using the RPROP variation of the BP learning algorithm, presents better results; however he does not mention standard deviation (although G-Prop obtains smaller sized networks, 4.3 neurons, with a similar classification error,  $3.0 \pm 0.7$ ). On average G-Prop is slightly better than RPROP, both in error rates and net size.

Yao and Liu present a method to evolve modular neural networks and the results obtained on the cancer problem in [23]. However, the results they present are better on classification accuracy ( $0.037 \pm 0.012$ ), the network sizes obtained are greater ( $13.1 \pm 1.2$ ) than those obtained using G-Prop. Moreover, we have not included Yao's results in Table 1 because we have used three different cancer sets to test G-Prop against QP and RPROP, while Yao and Liu use only one of them. Thus, although Yao and Liu also use the cancer problem, we do not know against which cancer problem we have to compare to.

For the *DNA helicases* problem, the GA was executed for 10 generations on a population of 20 individuals, with a probability of mutation of 0.5 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 a suitable learning coefficient for this problem, where a MLP can easily fall into a local minimum (0.07). 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 [28].

In the *DNA helicases* problem, it is evident that G-Prop outperforms other methods: G-LVQ [28] takes around 20 generations to achieve an error of  $18 \pm 2$ , while G-Prop achieves an error of  $6.10 \pm 3.01$ .

As it has been shown, in both problems the standard deviation for the G-Prop algorithm is much smaller than that for QP. This further supports the notion that QP may often be trapped in various different local minima whereas G-Prop can avoid

Table 2

Results of evaluating QP and G-Prop on the classification of different views of large T antigen of Simian Virus 40, and the results obtained with G-LVQ [28]

DNA Helicases	Error% $\pm$ Std Dev	Hidden units $\pm$ Std Dev (parameters)
QP	$9 \pm 10$	7 (189)
	$9 \pm 9$	11 (297)
G-Prop	$6 \pm 3$	$7.3 \pm 3.9$ (197.1)
G-LVQ	20 generations	$18 \pm 2$
	50 generations	$13 \pm 3$
	100 generations	$15 \pm 5$
		$2.3 \pm 0.4$ (59.5)
		$2.2 \pm 0.3$ (57)
		$2.2 \pm 0.3$ (57)

many of such traps. Moreover, Prechelt [35] does not mention the standard deviation value, but it can be seen that G-Prop obtains much smaller values than those obtained using G-LVQ [28].

Each run of the proposed method takes about 3 min on a Pentium 233MMX, using the parameters described above, while each run of QP takes about 1 min; thus, it is evident that is less time-consuming apply G-Prop than to train as many randomly-initialized MLPs using QP.

Summing up, these experiments show that G-Prop avoids the task of setting MLP parameters, finding them as a result, and besides, obtains smaller networks and with less error than other procedures.

#### 4. Conclusions

This paper presents G-Prop, an algorithm to train MLPs based on GA and BP. Experiments prove that the proposed method achieves better results than other non-evolutionary variants and, besides, obtains network size as a result.

In particular, the proposed algorithm (G-Prop) 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.

The idea of the approach presented here is to use an MLP population that will be evaluated to obtain its classification ability. This value, together with the hidden layer size, is used to determine fitness. Using the individuals with highest fitness, after having applied the designed genetic operators, a new population is obtained.

The GA evaluates each individual taking into account the classification capacity and the hidden layer size. The weights of the individuals/nets of the first generation are generated randomly, and these are the weights that the GA will evolve using operators that alter the ANN size. This strategy attempts to avoid Lamarckism (i.e., that individuals inherit the trained weights from their parents), but, at the same time, it is a good strategy to avoid local minima.

Several benchmarks (cancer 1, 2 and 3, and DNA helicases) have been used to test the proposed algorithm and to compare it with others [28,38]. 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 [13,33]. The QP algorithm will also be used as an operator in the GA, as suggested in [31], and it will be used to check the Baldwin effect on the rate of evolution, as it is proposed by Keesing and Stork [20]. Finally, the presented algorithm will be applied to solve other real world problems.

## 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.

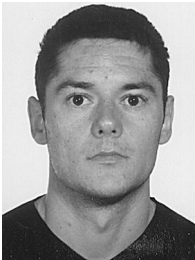
The authors are grateful to anonymous referees for their constructive comments and advice about the first revision of the paper.

## References

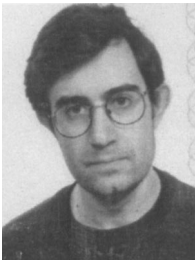
- [1] E. Alpaydim, GAL: networks that grow when they learn and shrink when they forget, *Int. J. Pattern Recogn. Artif. Intell.* 8 (1) (1994) 391–414.
- [2] K. Balakrishnan, V. Honavar, Evolutionary design of neural architectures – a preliminary taxonomy and guide to literature, Technical Report, CS-TR 95-01, AI Research Group, January 1995.
- [3] J.M. Baldwin, A new factor in evolution, *Am. Nat.* 30 (1896) 441–451.
- [4] G. Bebis, M. Georgiopoulos, T. Kasparis, Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization, *Neurocomputing* 17 (1997) 167–194.
- [5] I. Bellido, G. Fernandez, Backpropagation growing networks: towards local minima elimination, *Lectures Notes in Computer Science*, Springer, Berlin, Vol. 540, 1991, pp. 130–135.
- [6] P.A. Castillo, J. González, J.J. Merelo, V. Rivas, G. Romero, A. Prieto, SA-Prop: optimization of multilayer perceptron parameters using simulated annealing, *Lectures Notes in Computer Science*, Springer, Berlin.
- [7] Y.L. Cun, J.S. Denker, S.A. Solla, Optimal brain damage, in: D.S. Touretzky (Ed.), *Neural Information Systems 2*, Morgan-Kaufman, Los Altos, CA, 1990, pp. 598–605.
- [8] 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.
- [9] I. De Falco, A. Iazzetta, P. Natale, E. Tarantino, Evolutionary neural networks for nonlinear dynamics modeling, in: *Parallel Problem Solving from Nature 98. Lectures Notes in Computer Science*, Vol. 1498, Springer, Berlin, 1998, pp. 593–602.
- [10] N. Dodd, D. Macfarlane, C. Marland, Optimisation of artificial neural networks structure using genetic techniques implemented on multiple transputers, *Proceedings of Transputing 91*, 1991.
- [11] S. Fahlman, An empirical study of learning speed in back-propagation networks, Technical Report, Carnegie Mellon University, 1988.
- [12] S.E. Fahlman, in: *Faster-learning variations on back-propagation: an empirical study*, *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, Los Altos, CA, 1988.
- [13] S. Fahlman, C. Lebière, The cascade-correlation learning architecture, in: D.S. Touretzky (Ed.), *Neural Information Systems 2*. Morgan-Kaufman, Los Altos, CA, 524–532, 1990.
- [14] F.C. Gruau, Cellular encoding of genetic neural networks, Technical Report, LIP-IMAG Ecole Normale Supérieure de Lyon, 46 Allée d'Italie 69007, Lyon, France, 1992.
- [15] S.A. Harp, T. Smad, A. Guha, Towards the genetic synthesis of neural networks. in: J.D. Schaffer (Ed.), *Third International Conference on Genetic Algorithms*, Morgan Kaufman, Los Altos, CA, 1989, pp. 360–369.
- [16] B. Hassibi, D.G. Stork, G. Wolff, T. Watanabe, Optimal Brain Surgeon: extensions and performance comparisons, *NIPS6*, 1994, pp. 263–270.
- [17] G.E. Hinton, S.J. Nowlan, How learning can guide evolution, *Complex Systems* 1 (1987) 495–502.
- [18] J. Hwang, S. You, S. Lay, I. Jou, The cascade-correlation learning: a projection pursuit learning perspective, *IEEE Trans. Neural Networks* 7 (2) (1996) 278–289.
- [19] T. Jasic, H. Poh, Analysis of pruning in backpropagation networks for artificial and real world mapping problems, *Lectures Notes in Computer Science*, Springer, Berlin, Vol. 930, 1995, 239–245.
- [20] R. Keesing, D.G. Stork, Evolution and learning in neural networks: the number and distribution of learning trials affect the rate of evolution, *Adv. Neural Inform. Process. Systems* 3 (1991) 805–810.

- [21] W. Kinnebrock, Accelerating the standard backpropagation method using a genetic approach, *Neurocomputing* 6 (1994) 583–588.
- [22] H. Kitano, Designing neural networks using genetic algorithms with graph generation system, *Complex Systems* 4 (1990) 461–476.
- [23] Y. Liu, X. Yao, Evolving modular neural networks which generalise well, in: M. Sugisake (Ed.), *Proceedings of the IEEE International Conference on Artificial Life and Robotics, AROBIII 98*, Vol. 2, 1998, pp. 736–739.
- [24] O.L. Mangasarian, R. Setiono, W.H. Wolberg, Pattern recognition via linear programming: Theory and application to medical diagnosis, in: T.F. Coleman, Yuying Li (Eds.), *Large-scale Numerical Optimization*, SIAM Publications, Philadelphia, 1990, pp. 22–30.
- [25] F.J. Marín, F. Sandoval, Diseño de redes neuronales artificiales mediante algoritmos genéticos, *Computación neuronal*, Universidad de Santiago de Compostela, 1995, pp. 385–424.
- [26] C.S. Martin, C. Gruss, J.M. Carazo, Six molecules of SV40 large t antigen assemble in a propeller-shaped particle around a channel, *J. Molecular Biol.* 1997, in press.
- [27] J.J. Merelo, M. Patón, A. Cañas, A. Prieto, F. Morán, Optimization of a competitive learning neural network by genetic algorithms, *Lectures Notes in Computer Science*, Springer, Berlin, Vol. 686, 1993, pp. 185–192.
- [28] J.J. Merelo, A. Prieto, F. Morán, R. Marabini, J.M. Carazo, Automatic classification of biological particles from electron-microscopy images using conventional and genetic-algorithm optimized learning vector quantization, *Neural Process. Lett.* 8 (1998) 55–65.
- [29] G.F. Miller, P.M. Todd, S.U. Hegde, Designing neural networks using genetic algorithms, in: J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, 1989, pp. 379–384.
- [30] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Third extended Edition, Springer, Berlin, 1996.
- [31] D.J. Montana, L. Davis, Training feedforward neural networks using genetic algorithms, *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1989, pp. 762–767.
- [32] R. Parekh, J. Yang, V. Honavar, 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.
- [33] M. Pelillo, A. Fanelli, A method of pruning layered feed-forward neural networks, *Lectures Notes in Computer Science*, Vol. 686, Springer, Berlin, 1993, 278–283.
- [34] V. Petridis, S. Kazarlis, A. Papaikonomou, A. Filelis, A hybrid genetic algorithm for training neural networks, *Artif. Neural Networks* 2 (1992) 953–956.
- [35] L. 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.
- [36] T. Rathbun, S. Rogers, M. DeSimio, M. Oxley, MLP iterative construction algorithm, *Neurocomputing* 17 (1997) 195–216.
- [37] M. Riedmiller, RPROP: description and implementation details, Technical Report, University of Karlsruhe, 1994.
- [38] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the RPROP algorithm, in: H. Ruspini (Ed.), *Proceedings of the ICNN93*, San Francisco, 1993, pp. 586–591.
- [39] N.N. Schraudolph, R.K. Belew, Dynamic parameter encoding for genetic algorithms, *Machine Learning* 9 (1) (1992) 9–21.
- [40] A.P. Topchy, O.A. Lebedko, V.V. Miagkikh, Fast learning in multilayered neural networks by means of hybrid evolutionary and gradient algorithms. *Proceedings of IC on Evolutionary Computation and its Applications*, Moscow, 1996, to appear.
- [41] A. Weigend, D. Rumelhart, B. Huberman, Generalization by weight elimination with application to forecasting, *Advanced Neural Information Processing Systems* 3 (1991) 875–882.
- [42] D. White, P. Ligomenides, GANNet: a genetic algorithm for optimizing topology and weights in neural network design, *Lectures Notes in Computer Science*, Vol. 686, Springer, Berlin, 1993, pp. 322–327.

- [43] D. Whitley, The GENITOR Algorithm and Selection Pressure: Why rank-based allocation of reproductive trials is best, in: J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufman, Los Altos, CA, 1989, pp. 116–121.
- [44] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithms and neural networks: Optimizing connections and connectivity, *Parallel Computing* 14 (1993) 347–361.
- [45] X. Yao, A review of evolutionary artificial neural networks, Technical Report, CSIRO, 1992.
- [46] X. Yao, A review of evolutionary artificial neural networks, *Int. J. Intell. Systems* 8 (4) (1993) 539–567.
- [47] X. Yao, Y. Liu, Towards designing artificial neural networks by evolution, *Applied Mathematics and Computation* 91 (1) (1998) 83–90.
- [48] J. Zhang, A. Morris, A sequential learning approach for single hidden layer neural networks, *Neural Networks* 11 (1997) 65–80.



**Pedro Angel Castillo Valdivieso** received B.Sc. degree in Computer Science from the University of Granada in 1997. He has been working as Technical Assistant for EvoNet and as Lecturer at the Jaén University, Spain. His main interests in research are in neural networks and evolutionary computation. His Ph.D. thesis, which will be presented in June 2000, is related to the optimization of artificial neural networks using evolutionary algorithms. He is currently working under a fellowship from the regional government at the Computer Architecture and Technology Department of the University of Granada, Spain.



**Juan J. Merelo** received B.Sc. degree in Theoretical Physics from the University of Granada in 1988 and a Ph.D. from the same University in 1994. He has been visiting researcher at Santa Fe Institute, Politecnico Torino, RISC-Linz, University of Southern California, and Université Paris-V. His main interests in research are in neural networks, genetic algorithms and artificial life. He is currently Associate Professor at the Computer Architecture and Technology Department of the University of Granada, Spain.



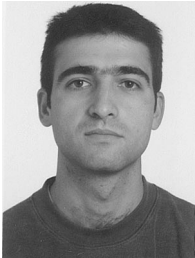
**Alberto Prieto** received B.Sc. degree in electronic physics in 1968 from the Complutense University, Madrid and his Ph.D. degree from the University of Granada, Spain, in 1976, obtaining the award of Ph.D. Dissertations and the Citema Foundation National Award. From 1969 to 1970 he was at the Centro de Investigaciones Técnicas de Guipuzcoa and at the E.T.S.I Industriales of San Sebastián. From 1971 to 1984 he was Director of the Computer Centre and from 1985 to 1990 Dean of the Computer Science and Technology studies of the University of Granada. He is currently Full Professor and Director of the Department of Computer Architecture and Technology at the University of Granada. He has been a visiting researcher in different foreign centres including the University of Rennes (1975, Prof. Boulaye), the Open University (UK) (1987, Prof. S.L. Hurst), the Institute National Polytechnique of Grenoble (1991, Profs. J. Hérault and C.

Jutten), and the University College of London (1991–92, Prof. P. Treleaven). His research interests are in the area of intelligent systems.

Prof. Prieto was the Organization Chairman of the International Workshop on Artificial Neural Networks (IWANN'91), Granada, September 17–19, 1991, and of the 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-inspired Systems (MicroNeuro'99) Granada, Spain, April 7–9, 1999. He was nominated member of the IFIP WG 10.6 (Neural Computer Systems) and Chairman of the Spanish RIG of the IEEE Neural Networks Council.



**Victor Rivas** is working as Lecturer at the Department of Computer Science at the University of Jaen, Spain, after having received B.Sc. degree in Computer Science in 1994 from the University of Granada. He is currently doing his Ph.D. thesis, which is related with evolutionary optimization of Radial Basis Function Neural Networks. He was invited by Dr. Mark Schoenauer in October, 1998 to spent two months in the Ecole Polytechnique, in Paris.



**Gustavo Romero López** received B.Sc. degree in Computer Science from the University of Granada in 1997. He has been working as programmer for CICYT project BIO96-0895. He is currently working under a fellowship from the regional government at the Computer Architecture and Technology Department of the University of Granada, Spain. His main interests in research are in neural networks and evolutionary computation. His Ph.D. thesis is related to evolutionary computation visualizaton.