ELSEVIER

Neurocomputing 17 (1997) 167–194

NEUROCOMPUTING

Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization

George Bebis*, Michael Georgiopoulos, Takis Kasparis

Department of Electrical & Computer Engineering, University of Central Florida, Orlando, FL 32816, USA

Received 23 January 1997; accepted 5 May 1997

Abstract

Recent theoretical results support that decreasing the number of free parameters in a neural network (i.e., weights) can improve generalization. These results have triggered the development of many approaches which try to determine an "appropriate" network size for a given problem. The main goal has been to find a network size just large enough to capture the general class properties of the data. In some cases, however, network size is not reduced significantly or the reduction is satisfactory but generalization is affected. In this paper, we propose the coupling of genetic algorithms with weight elimination. Our objective is not only to significantly reduce network size, by pruning larger size networks, but also to preserve generalization, that is, to come up with pruned networks which generalize as good or even better than their unpruned counterparts. The innovation of our work relies on a fitness function which uses an adaptive parameter to encourage reproduction of networks having small size and good generalization. The proposed approach has been tested using both artificial and real databases demonstrating good performance.

Keywords: Neural networks; Genetic algorithms; Weight elimination; Pruning

1. Introduction

One of the most critical problems in neural network design is the problem of choosing an appropriate network size for a given application [2, 3, 18]. Network size

* Correspondence address: Department of Mathematics and Computer Science, University of Missouri-St. Louis, 312 Computer Center Building, 8001 Natural Bridge Rd., St. Louis, MO 63121-4499, USA. Tel.: +1314 516-6342; fax: +1314 516-5400; e-mail: bebis@mayura.cs.umsl.edu.

0925-2312/97/$17.00 Copyright © 1997 Elsevier Science B.V. All rights reserved
PII S0925-2312(97)00050-7

in the case of layered, feed-forward, neural networks depends on the number of layers, the number of nodes per layer and the number of incoming connections onto a node (fan-in). Here, we express network size in terms of the number of connections in the network. In general, network size affects network complexity and learning time, but most importantly, it affects the generalization capabilities of the network; i.e., its ability to produce accurate results on data outside its training set. Given a set of training examples, there is probably a large number of different size networks that can learn to perform the underlying mapping (i.e., achieve a small enough training error). However, the generalization capabilities of each one of these networks will not be the same (i.e., the generalization error will be different from one network to the other). A network having a structure more complicated than necessary, "overfits" the training data, that is, it performs nicely on patterns included in the training set but performs very poorly on unknown patterns. On the other hand, a network having a structure simpler than necessary cannot give good results even for patterns included in its training set.

Major emphasis has been given in the last few years to the development of techniques which try to reduce network size by modifying not only the connection weights but also the network structure during training. These techniques can be divided into three main categories. The first category includes methods that start with a big network and gradually eliminate the unnecessary nodes or connections. These methods are called *pruning* [18, 22]. Pruning based methods remove unnecessary nodes or connections using a sensitivity based measure to indicate how the solution is affected from this change [21, 25, 30], or modifying the error function of the network in such a way that unnecessary connection weights are driven to zero during training (weight decay) [7, 19, 35, 23, 13]. The second category includes the *constructive* methods [22], which start with a small network and gradually add nodes or connections during training [27, 9, 10, 36, 1]. The last category includes the *weight sharing* methods [18]. In weight sharing, many connections share the same weight so that the number of free parameters in the network becomes much less than the number of connections. Hidden nodes are divided into groups and a local receptive field defines the input to each hidden node. Weight sharing is achieved by assigning an identical set of weights to each node within the same group [24, 32].

Although reducing network size is very important, it should be attempted under the assumption that generalization is not affected. In some studies where generalization was addressed, improvements were observed using artificial data sets only [19, 8, 34], while the same or even worse generalization has been reported in some other studies where real data sets were used [16, 23, 37, 15, 20]. Our interest in this paper is to improve the performance of weight pruning techniques. Weight pruning techniques are very sensitive to the selection of certain parameter values which determine when pruning should start and when it should stop. If pruning starts too early, the network might not be able to learn the desired mapping. On the other hand, if pruning stops early, it might not be possible to sufficiently prune the network. Also, if pruning does not stop at the right point, it might be possible to overprune and this will deteriorate generalization. Determining appropriate pruning parameter values to control the beginning and the end of the pruning process is usually done by trial and error.

Recently, there has been some work in order to determine these parameters in advance [26] using probabilistic approaches.

In this paper, we propose the coupling of genetic algorithms [11] and weight pruning. In particular, the weight elimination technique [35], a representative weight pruning technique and the most general of weight decay approaches, has been chosen to be coupled with the genetic algorithm. However, the framework of the approach we propose is more general and other pruning techniques can also be coupled with genetic algorithms. The goal of the proposed approach is to prune oversized networks without affecting generalization. Genetic algorithms are a class of optimization procedures inspired by the biological mechanisms of reproduction. Research on combining genetic algorithms and neural networks has lately attracted a lot of attention [38]. The most common approaches within this context use a genetic algorithm in order to: (1) select features used by a neural net; (2) select the parameters that control the learning in a neural net; (3) analyze the solution achieved by a neural net; (4) determine the neural network weights; and (5) determine the neural network architecture (see [38] for an excellent review). Our approach belongs to the last category; our primary interest is not only to determine an appropriate network size by pruning oversized networks but also to demonstrate that the pruned networks improve generalization. Some preliminary results of our work have been presented in [4].

Genetic algorithms operate iteratively on a population of structures, each one of which represents a candidate solution to the problem at hand. When genetic algorithms are used in the context of neural networks, an important issue to be addressed is how the architecture should be represented (encoded) into a structure (string of symbols) that can be handled by the genetic algorithm. Once an encoding scheme has been chosen, a number of networks are encoded to form the initial population. In our approach, we start with an oversized network and we consider a number of copies of this network with different initial weights and pruning parameters, to form the initial population. The encoding scheme we are using is a local scheme presented in [29]. On each iteration, a new population is formed by first applying a number of genetic operations (reproduction, crossover, and mutation) on the old population. Then, each member of the population is decoded into a legitimate network and is evaluated. The evaluation is performed by first training each network for a small number of epochs using weight elimination. Then, a fitness function is used to measure the performance of each member in the population. This process is repeated until the genetic algorithm converges to a final population. The criterion used for convergence is based on the average improvement of the genetic algorithm and is described in Section 4.

Choosing a "good" fitness function is probably the most critical issue in genetic algorithm design since it provides the mechanism for evaluating the members (encoded solutions) of a population. The improvement of a solution at future generations depends highly on the evaluations that the population members have received at past generations. Members assigned a bad evaluation are discarded, while members assigned a good evaluation survive in future populations. The innovation of the proposed approach relies on the use of a fitness function which takes into consideration both network size and generalization. In particular, during the generation of new

genetic populations, an adaptive parameter weights the importance of network size versus generalization, encouraging the reproduction of networks having good generalization and a relatively small size.

The motivation for using genetic algorithms is that as new populations are formed, the network structure of each member in a population changes in a different way, since each network is associated with different parameter values. Since the characteristics among different members in the population can be exchanged by applying the genetic operators, new, more powerful members may be discovered as evolution proceeds. As we mentioned before, choosing appropriate parameter values for the weight elimination technique (or for other weight pruning techniques) is a trial and error procedure. The goal of the proposed approach is to discover better solutions. using the same parameter settings chosen in the trial and error procedure. In other words, let us assume that we have decided to perform a number T of trials, using various parameter settings. Then, instead of training each network separately, we form a population of T networks, where each one of them is identical to one of the T independent initializations, and allow the genetic algorithm to form more powerful populations. Our experimental results, given in Section 4, demonstrate that following this approach, the genetic algorithm finally converges to a population whose best member is as good or better than the best solution found by training each network individually with weight elimination (using the same parameter values) or without weight elimination (unpruned network). Furthermore, the average generalization performance of our approach is always better.

The organization of the paper is as follows: Section 2 reviews the weight elimination technique and illustrates its sensitivity to the selection of certain parameter values which affect network size and generalization. Section 3 discusses the genetic algorithm approach. The network representation scheme, the genetic operators, and the fitness function utilized are presented in detail. The databases used, the experiments performed, and the results obtained are presented in Section 4. Finally, our conclusions are given in Section 5.

2. Reducing network size using weight elimination

Weight elimination is a general weight decay approach proposed by Weigend et al. [35]. It minimizes a modified error function which is formed by adding a penalty term to the original error function of the back-propagation algorithm. Specifically, the modified error function has the form

$$E_{\mathrm{WE}} = E_0 + \lambda_{\mathrm{WE}} E_1 = \sum_{\mu} k\,(t_k^{\mu} - o_k^{\mu})^2 + \lambda_{\mathrm{WE}} \sum_{i,\,j} \frac{w_{ij}^2/w_0^2}{1 + w_{ij}^2/w_0^2}, \tag{1}$$

where the sum over μ implies summation over all the training examples and the sum over k implies summation over all the output nodes.

The first term is the original error function which is simply the sum of the squared errors between the actual output values (o_k) and the desired output values (t_k). The

second term represents the complexity of the network as a function of the weight magnitudes relative to the parameter w_0. The factor λ_{WE} is a weighting factor which determines the importance of the network's complexity with respect to the network's performance over the training set. The choice of w_0 encourages the algorithm to find solutions with fewer large weights (w_0 small) or many small weights (w_0 large). The distinction between small and large weights is made with respect to the magnitude of w_0. To make this clear, let us consider the ratio $|w_i|/w_0$. If $|w_i|/w_0 \ll 1$, the cost of each weight approaches zero and the second term of Eq. (1) approaches zero as well. On the other hand, if $|w_i|/w_0 \gg 1$, then the cost of each weight approaches λ_{WE} and the second term of Eq. (1) is minimized when the algorithm converges to a solution with a few large weights.

There are several issues to be addressed during the implementation of the weight elimination technique. The first and probably most important issue is deciding when pruning should start affecting training. This issue is straightforward related to the choice of the weight factor λ_{WE}. When λ_{WE} is very small or zero, pruning does not affect training at all. When this parameter is large, small weights decay more rapidly than large ones. However, this parameter should not be chosen too large since all the weights will be driven to zero then. One way that this parameter is usually chosen is by performing a number of simulations using different values and choosing the value that gives the best performance on a validation set. However, experience has shown that better results can be achieved if this parameter can be determined adaptively during training. Weigend et al. [35] have proposed a procedure for this. Initially, λ_{WE} is set to zero. Then, it increases, decreases, or stays the same according to a methodology based on a number of parameters such as the error of the network on the training set, the average error (exponentially weighted over the past errors of the network on the training set), and a desired error provided externally by the user. The amount by which λ_{WE} must be increased or decreased is another parameter specified by the user. We have found that this procedure is not very robust. Here we have chosen an alternative way for determining λ_{WE}, motivated by [19]. Specifically, λ_{WE} is determined as follows:

$$\lambda_{WE} = \lambda_{0\_WE}\, e^{-\beta_{WE} E_{gen}}, \tag{2}$$

where λ_{0_WE} is a scaling factor (which was set equal to one in all of our experiments) and β_{WE} is a constant to be defined shortly. E_{gen} is an estimation of the generalization error of the network. When E_{gen} is large, λ_{WE} will be small and the second term in Eq. (1) will contribute almost nothing to the total error. When E_{gen} starts decreasing, the second term will start being more significant driving small weights to zero. A common way of estimating the generalization performance of a network during training is by using cross-validation [18]. In cross-validation, a validation set which is different from both the training and test sets is used to estimate the performance of the network during training. When the performance of the network on the validation set starts decreasing, this is an indication that the network starts overfitting the data. If G_{val} represents the ratio of the correctly classified patterns from the validation set over the total number of patterns in the validation set, then we define E_{gen} as $E_{gen} = 1 - G_{val}$. Updating G_{val} and λ_{WE} takes place in every epoch.

Another important issue to be addressed is when training, and consequently pruning, should stop. It has been reported in [35] that the exact stopping point is not very important. However, we demonstrate in the next section that this is not consistently true since it might lead us to overpruning, which can deteriorate generalization significantly. It is also important to decide when is the appropriate time to remove the connections associated with small weight values from the network. Obviously, we do not want to remove any connection, even if its corresponding weight value is very small, if the performance of the network on the training set is very poor. Usually, connections associated with weak weights are removed when the network has achieved some performance on the training set (i.e., the error over the training set has become smaller than a threshold [19]). In our implementation, connections are removed at the end of the training process. A weight w_{ij} is removed only if $|w_{ij}| < |w_0|$. It should be mentioned that approaches based on sensitivity analysis [30, 15] might be more appropriate in order to remove redundant weights. However, we have decided to use the above approach because it is simpler, faster, and it does not affect the ideas we are trying to demonstrate here. To account for the resulting increment of the error due to the elimination of the weaker weights, we train the pruned network for a few additional epochs using the standard back-propagation algorithm.

The modified version of weight elimination that we have described depends mainly on the choice of β_{WE}, w_0, and the weights assigned to the network in the beginning of the training process. The sensitivity of weight elimination on the selection of these parameters is considered in the next subsection through an experimental study. The conclusions of this experimental study are useful in the development of our proposed approach.

2.1. Sensitivity of weight elimination

Choosing appropriate parameter values for the weight elimination technique i involves some experimentation. Usually, we start with a network whose size has been chosen large enough to ensure convergence and we train it a number of times using different parameter values each time (i.e., different initial weights and different values for the parameters controlling weight elimination). Then, we choose the best solution in terms of size and generalization. In this subsection we present a number of experiments in order to investigate the dependence of weight elimination on the choice of these parameter values. Four different databases have been used in the experiments reported in this subsection: the Numbers, the Ionosphere, the Wine, and the Breast-cancer databases. Details about the databases are given in Section 4.

Initially we investigated the dependence of weight elimination on the stopping point used during training. Some experimental results obtained in [35] demonstrate that the exact point at which training must be terminated is not very important. In fact, generalization was shown to be good even for training times four times longer than the training time required to train a network without using weight elimination. However, a number of experiments performed by us using different data sets demonstrated that the stopping point is indeed important. Fig. 1 shows results obtained using the Wine database. The parameters used were $\beta_{WE} = 80$ and $w_0 = 1.0$.

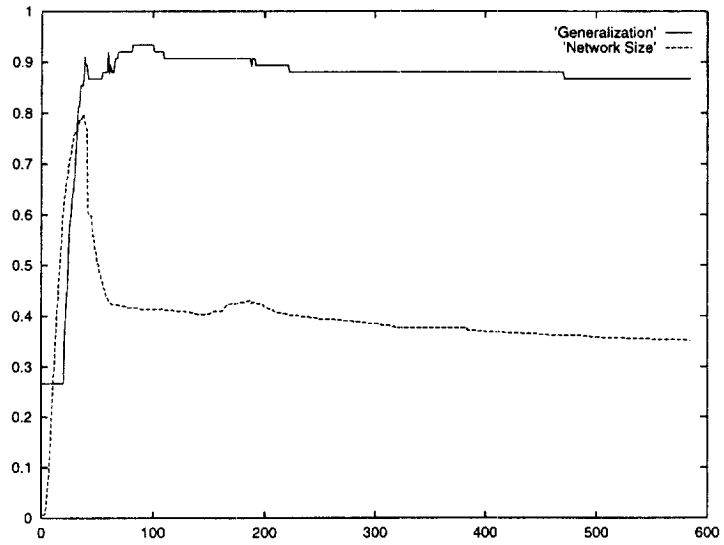


Fig. 1. An example illustrating the effects of overpruning. The continuous line represents the generalization performance of the network while the dashed line represents the percentage of connections whose absolute weight value is greater than $|w_0|$. The horizontal axis represents the number of epochs.

Obviously, generalization starts decreasing as soon as excessive weight elimination takes place. Thus, stopping weight elimination before overpruning occurs is important.

Next, we examined the sensitivity of weight elimination on the selection of the β_{WE} parameter. Thus, we fixed w_0 ($w_0 = 0.25$) and the initial weights and we performed 20 different experiments using different β_{WE} values each time (from $\beta_{WE} = 10$ to $\beta_{WE} = 100$ in increments of 5). Fig. 2 demonstrates the results for each one of the four databases. The horizontal axis represents different β_{WE} values. It is clear that different β_{WE}'s are more appropriate for different databases, making the choice of appropriate β_{WE} values problem dependent. For each database, there is a specific range of β_{WE} values which reduce network size significantly while good generalization can be achieved within this range. For the Numbers database, the best β_{WE}'s seems to be in the range of [20–30], for the Ionosphere database in the range of [15–50], for the Wine database in the range of [25–35] and for the Breast-cancer database in the range of [30–100].

In the next set of experiments, we tested the dependence of weight elimination on the selection of the w_0 parameter. For this reason, we fixed β_{WE} ($\beta_{WE} = 30$) and the initial weights and we performed 20 different experiments using different w_0 values each time (from $w_0 = 0.1$ to $w_0 = 1.0$ in increments of 0.05). Fig. 3 shows the results. The horizontal axis represents in this case different w_0 values. The results demonstrate that the choice of w_0's value is not quite problem dependent and that choosing w_0 close to 1.0 gives good results both in terms of network size and generalization.

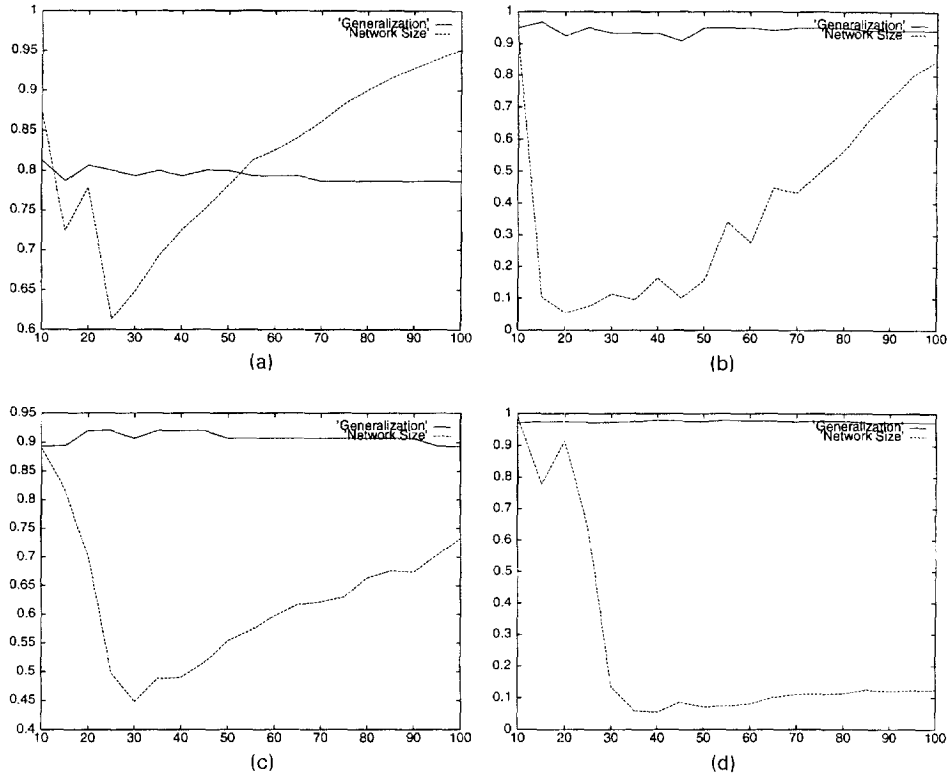


Fig. 2. Different β_{WE} values: (a) the Numbers database, (b) the Ionosphere database, (c) the Wine database, and (d) the Breast cancer database. The continuous line represents the generalization performance of the network while the dashed line represents the percentage of connections whose absolute weight value is greater than $|w_0|$. The horizontal axis represents different β_{WE} values.

This is also in agreement with [35] where it is suggested that choosing w_0 close to 1.0 is best for networks where activation functions in the range of [0.0–1.0] are used.

In the last set of the experiments, we tested the sensitivity of weight elimination on the selection of different initial weights. This time, we fixed β_{WE} and w_0 ($\beta_{WE} = 30$ and $w_0 = 0.25$) and we performed 20 different experiments using different random initial weights each time (randomly chosen in the range of $[-0.1$ to $0.1]$). Fig. 4 shows the results for the different databases used. The horizontal axis represents in this case the different experiments performed. It is clear that the choice of the initial weights is very critical since both generalization and network size are affected in an unpredictable way.

In conclusion, weight elimination depends mostly on the choice of the initial weights and the value of parameter β_{WE}. The choice of the value of the parameter w_0 does not seem to significantly affect the performance of the method and good results can be achieved by choosing w_0 close to 1.0.

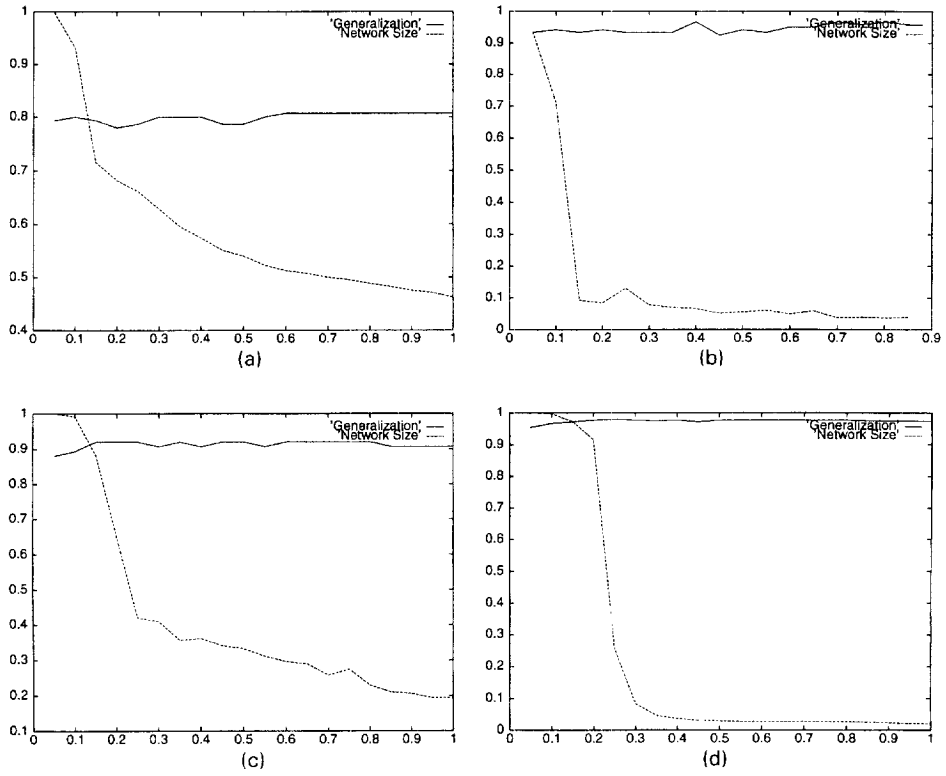


Fig. 3. Different w_0 values: (a) the Numbers database, (b) the Ionosphere database, (c) the Wine database, and (d) the Breast-cancer database. The continuous line represents the generalization performance of the network while the dashed line represents the percentage of connections whose absolute weight value is greater than $|w_0|$. The horizontal axis represents different w_0 values.

3. The genetic algorithm approach

As we indicate in the previous section, weight elimination is quite sensitive to the selection of certain parameters and differences in terms of generalization and network size can be observed by changing these parameter values. Finding good parameter values requires an extensive experimentation. Here, we propose not to perform each experiment independently. Instead of this, we form a population of networks by choosing the same exactly networks chosen for the individual experiments, and apply genetic algorithms. It should be clear that we do not choose new parameter values (initial weights and β_{WE}) for the networks in the population but we use the same choices we made for the independent experiments. Our experimental results demonstrate that this approach leads to as good or better solutions in terms of both network size and generalization. Most importantly, the solutions found by the genetic algorithm approach may correspond to parameter settings not chosen in the initial set of

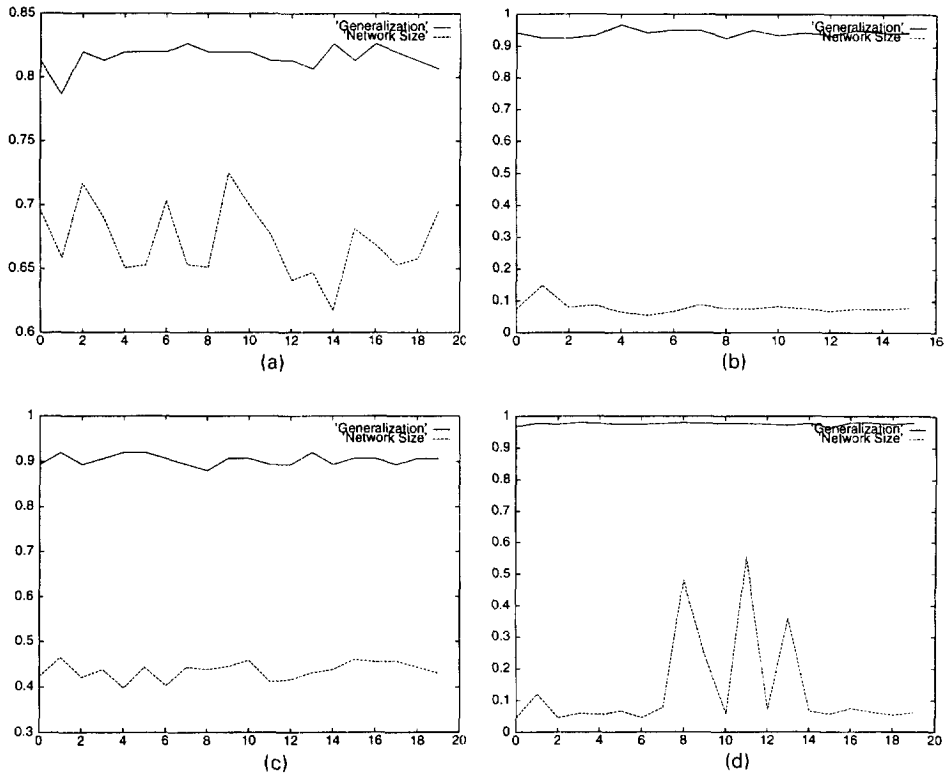


Fig. 4. Different initial weights: (a) the Numbers database, (b) the Ionosphere database, (c) the Wine database, and (d) the Breast cancer database. The continuous line represents the generalization performance of the network while the dashed line represents the percentage of connections whose absolute weight value is greater than $|w_0|$. The horizontal axis represents different experiments, using different set of initial weights.

networks. Consequently, training each network separately will not lead to the same solution found by the genetic algorithm approach.

Initially, we start with two-layer networks (i.e., one hidden and one output layer), having enough nodes in the hidden layer to ensure convergence. The reason we have restricted ourselves to two-layer networks is because *a single hidden layer feed-forward network with arbitrary sigmoid hidden layer activation functions can approximate arbitrarily well an arbitrary mapping from one finite dimensional space to another* [17]. After an oversized network has been chosen, we encode it into a structure that can be handled by the genetic algorithm and we create P copies of it, where P represents the population size. Each of these copies is assigned a different set of parameter values which are the initial weights and β_{WE}. This choice was based on the experimental results presented in Section 2.1, which indicated that these parameters affect generalization and network size the most. New populations are generated by applying the

genetic operators of reproduction, crossover and mutation on the weights of the networks.

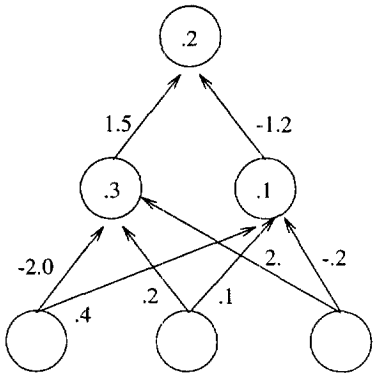
The fitness of each member is measured by first decoding it into a network. Then, we train the network for a number of epochs using weight elimination in order to record its performance in terms of generalization and size. The evaluation function consists of two terms: the first term returns an evaluation with respect to the generalization performance of the network, while the second term returns an evaluation with respect to its size. A factor λ_{GA} weights the importance of generalization versus network size. Each network in the population is associated with its own λ_{WE} parameter. After the first few generations members in the same population will have totally different characteristics. This variety in network sizes and generalization performances will allow the genetic algorithm to search and probably discover better solutions.

It should be emphasized that pruning takes place not only due to weight elimination, which is applied during the evaluation phase, but also due to the application of the genetic operators and in particular the crossover operator. When crossover is applied, groups of weights between different networks are exchanged, and as a result larger weights may be replaced by smaller weights. Thus, the solutions obtained by combining the genetic algorithm with weight elimination might not be possible to replicate using weight elimination by itself. Next, we describe the network representation scheme, the genetic operators, and the fitness evaluation function used.

3.1. Network representation scheme

There are two main categories of network representation schemes: the weak and the strong specification schemes [38]. The main advantage of the weak specification scheme is that the architecture can be represented globally in terms of the number of layers, the number of nodes, and the degree of connectivity from one layer to another. However, quite complex encoding and decoding procedures are required (see, e.g., [38]). On the other hand, the strong specification scheme uses a more local representation by representing each node and connection of the network directly. Very simple encoding and decoding procedures are used in this case (see, e.g., [14]). Since our approach considers a large predefined, two-layer network, with the objective to reduce the number of connections, we do not really require encoding of the number of layers and the number of nodes per layer. The quantity which is important to encode is the number of connections between successive layers. Thus, the strong specification scheme is adequate for our purpose.

A popular strong specification scheme has been proposed by Miller et al. [28], where the network architecture is represented as a connection matrix mapped directly into a bit-string. Although this scheme satisfies our requirements, it has the disadvantage that it creates very long strings. For example, assuming a network with N nodes, its connection matrix becomes a $N \times N$ array. The existence of a connection between node i and node j in the network is indicated by a nonzero element in the ith row and jth column of the connection matrix. By simply concatenating the rows the rows of the connection matrix, a bit-strings of length N^2 is obtained, slowing down the speed of



(0.3, 0.1, -2.0, 0.4, 0.2, 0.1, 0.2, -0.2, 0.2, 1.5, -1.2)

Fig. 5. An example of the encoding scheme. The internal thresholds associated with the nodes of the first layer are placed first in the string. Next, we record the weight values associated with the connections of the first layer. The same procedure is applied for the second layer.

the algorithm. Here, we have adopted another approach proposed by Montana and Davis [29]. According to this approach, the weights and biases of a network are encoded in a straightforward way as a string of real numbers. Decoding is also straightforward. Fig. 5 shows an example of this encoding scheme.

3.2. Genetic operators

The genetic operators used in this work are the most commonly used operators: the reproduction, the crossover, and the mutation operators. The purpose of the reproduction operator is to create a new population based on the evaluation (fitness) of the members of the old population. Each member of the old population produces a number of exact copies with the objective that the most fit members will produce the most copies. Our implementation uses the roulette wheel selection scheme described in [11]. This scheme allocates new members based on the ratio of a member's fitness to the population's average fitness. Experience has shown that this kind of reproduction can cause premature convergence, since members with high fitnesses can overpower a population, eliminating its diversity. For this reason, fitness scaling has also been implemented [11]. The purpose of fitness scaling is to control the number of copies that members with high fitness value will receive in future populations. This is because a few members with high fitness values in the initial populations can take over a significant proportion of the population in a single generation, affecting the diversity of the population seriously, leading to premature convergence. Linear scaling is a quite popular fitness scaling procedure and we have also used it in our study. Linear scaling computes the scaled fitness value as

$$f' = af + b,$$

where f is the fitness value and f' is the scaled fitness value. The coefficients a and b are calculated in each generation to ensure that the maximum value of the scaled fitness value is a small number, say, 1.2–2.0 times the average fitness value of the population. More details are given in [11]. In addition to fitness scaling, two more heuristics have been incorporated in our implementation: the elitism strategy and the generation gap [12]. The elitism strategy guarantees that the members with the best fitness value in a population will survive in the next generation. This is performed by sorting the members of a population according to their fitness value and choosing the best of them to survive intact in the next generation. The generation gap is a parameter which controls the percentage of the population to be replaced during each generation. Thus, if P is the population size and G is the generation gap, then $P(1\text{-}G)$ members are chosen to survive intact in the next generation. Usually, these members are chosen in random, however, in our implementation we choose the best $P(1\text{-}G)$ members. Both of the above heuristics seem to be quite beneficial.

Crossover is applied after reproduction. Traditionally, crossover works as follows: pairs of members are selected at random and portions of them are exchanged to form new members. Here, we are using a modified crossover operator which we call the *crossover_nodes* operator. The idea is to swap groups of weights feeding into the same node. The reason is quite plausible; each node in the network contributes to the solution that the network tries to find. Thus, weights feeding into a node serve a role in finding a solution for the problem at hand. Swapping weights arbitrarily may not make a lot of sense while swapping groups of weights feeding into nodes is more sensible. This operator has also been used in [28]. Although crossover is one of the most powerful genetic operators, it may have disruptive effects to the solution a network tries to find. This is because different nodes in a network play a different role, as mentioned above. The nodes in the first hidden layer for instance, serve as feature detectors, while the nodes at higher layers serve for knowledge representation. Swapping weights feeding nodes located at hidden layers higher than the first may be disruptive, since the internal knowledge representations between two different networks are probably quite different. However, swapping weights feeding nodes only located at the first hidden layer, may be less disruptive for the networks, since it can be considered as an exchange of feature detectors. We have therefore decided to use this modified operator, which we call the *crossover_first_layer_nodes* operator. Both of the operators have been tested in our experiments and results are reported in Section 4.

The last genetic operator used is the mutation operator. This operator picks randomly a member from the population and changes it slightly. In its simplest form, mutation changes the value of a weight by adding a small random value. Following our discussion regarding the *crossover_nodes* operator, the mutation operator used in this study does not change single weights but groups of weights feeding into a node. It should be clear that we do not change each weight by the same amount. In fact, a different small random value is added to each of the weights. This modified operator which we call the *mutate_nodes* operator, has also been used in other studies [28], leading to good performance.

3.3. Fitness evaluation

The choice of a fitness function is problem dependent and is probably the most critical issue in genetic algorithm design. When genetic algorithms are combined with neural networks, the most commonly used approach to evaluate the performance of a member in the population is to train the network represented by this member and record its mean squared error. This is quite inappropriate for our purpose, since it does not account for the network's generalization performance and size. To perform an evaluation based on network size and generalization, we have considered a fitness function having the following form:

$$E_{\text{fit}} = G_{\text{net\_val}} + \lambda_{\text{GA}} (1 - E_{\text{net\_size}}) . \tag{3}$$

The first term ($G_{\text{net\_val}} = 1 - E_{\text{net\_gen}}$) accounts for generalization, while the second term ($1 - E_{\text{net\_size}}$) accounts for the network size. The parameter λ_{GA} is a weighing factor which controls the importance of the two terms. If λ_{GA} is very small, the fitness of a member is mostly determined by its generalization performance only. However, when λ_{GA} assumes large values, both generalization, and size influence the fitness of a member. The value of the weighing factor λ_{GA} is determined adaptively, in a similar manner that λ_{WE} is determined in weight elimination. Specifically, λ_{GA} is determined as follows:

$$\lambda_{\text{GA}} = \lambda_{0\_\text{GA}} \, e^{-\beta_{\text{GA}} E_{\text{net\_gen}}} , \tag{4}$$

where λ_{0_GA} and β_{GA} are constants specified by the user, and $E_{\text{net\_gen}}$ denotes the generalization error of the network. The goal of the genetic algorithm is to find solutions which maximize the above fitness function. It is clear from the definition of the fitness function that reproduction favors members with good generalization performance and a relatively small network size. In early generations, network size does not play an important role in reproduction and the fittest members are the members which generalize best. However, in future generations both network size and generalization affect reproduction. Cross-validation is used for an estimation of $E_{\text{net\_gen}}$. As in Section 2, $E_{\text{net\_gen}}$ is defined to be $1 - G_{\text{net\_val}}$, where $G_{\text{net\_val}}$ is the generalization performance of a network over the validation set. The network size $E_{\text{net\_size}}$ is defined to be the number of effective connections of the network (connections whose associated absolute weight values are greater than $|w_0|$) over the total number of connections. Both $E_{\text{net\_gen}}$ and $E_{\text{net\_size}}$ take values between 0 and 1.

4. Simulations and results

In order to evaluate our approach, an extensive experimental study has been performed using one artificial and seven real databases. The experiments were run on SPARC-2 and SPARC-5 Workstations and the implementation of the algorithms was made using C. The total amount of cpu time needed to complete all the experiments was about three months. The real databases were selected from the collection of the databases distributed by the machine learning group at the University of California at

Table 1
Data sets used and network architectures chosen

Data set	Training set	Validation set	Test set	Classes	Architecture
Numbers	150	50	150	10	63-40-10
Ionosphere	200	31	120	2	34-30-2
Soybean	254	53	255	15	35-60-15
Breast-cancer	200	99	400	2	10-40-2
Wine	75	28	75	3	13-35-3
Iris	90	15	45	3	4-35-3
Balance	250	125	250	3	4-60-3
Cars	400	100	346	4	18-80-4

Irvine [31], while a similar artificial database to the one we use here has previously been utilized by us in a character recognition experiment [5]. For each problem, data was normalized in the interval [0,1]. Assuming that each datapoint within a data set consists of m features $x_1, x_2, \ldots, x_m$, normalization is performed using the following formula:

$$x_i' = \frac{x_i - \min_{x_i}}{\max_{x_i} - \min_{x_i}} \tag{5}$$

where x_i' is the normalized value of the ith feature, x_i is the original value, while $\min_{x_i}$ and $\max_{x_i}$ are the minimum and maximum values of the ith feature over the whole data set. Then data was divided into a training, a validation, and a test set. Details are provided in Table 1. Four approaches have been compared: the original back-propagation (BP), the back-propagation with weight elimination (BP_WE), the genetic algorithm approach using the *crossover_nodes* and *mutate_nodes* operators (GA_BP_WE), and the genetic algorithm approach using the *crossover_first_layer_nodes* and *mutate_nodes* operators (GA1_BP_WE).

For each problem considered, a two-layer network was chosen (see Table 1). The size of the networks chosen for each problem was considered to be big enough since we were able to successfully train smaller size networks for the same problems without any particular difficulty. In the case of the BP and BP_WE techniques, experimental results were obtained by running 20 experiments with each method for each database. For each experiment, a different set of initial weights was used (randomly chosen in the range of $[-0.1$ to $0.1]$). For comparison purposes, both the BP and BP_WE techniques used the same 20 initial weight configurations for each database. In the case of the BP_WE technique, we had also to choose values for the parameters β_{WE} and w_0 (see Eqs. (1) and (2)). The parameter w_0 was set to 1.0 as mentioned in Section 2. The range of appropriate β_{WE} values is usually different from database to database, as shown in Section 2.1. Since we do not usually have any *apriori* knowledge about the best range of β_{WE} values, in practice we choose various parameter settings and we perform simulations until we get some satisfactory results. Here, we have

chosen a different β_{WE} value (in the range of [10–100]) for each of the 20 experiments performed per database. To challenge the genetic algorithm approach (which uses the same β_{WE} values as we will explain later), the same 20 β_{WE} values have been used for all the databases.

To determine when to stop training in the case of the BP and BP_WE algorithms, we used the maximum output error over the training and validation sets. The maximum error over the validation set was computed by presenting all the validation examples to the network, recording the error associated with each output node, and choosing the maximum error over all the validation examples. Similarly we computed the maximum error over the training set. The algorithms were considered to have converged if the maximum output over the validation set at epoch $t + 1$ was greater than the maximum error at epoch t, and the maximum error over the training set at step $t + 1$ was less than 0.25. The last condition was added to our stopping criterion to ensure well trained networks. In the case of the weight elimination approach, we also set a maximum number of epochs, since training was rather long in some cases. The maximum number of epochs allowed was three times the average number of epochs needed by the BP approach to converge on the same problem (averaged over the 20 experiments performed using BP). After training had been completed, testing was performed by applying the following methodology. First, we apply a test example at the inputs of the network and we compute the difference between the two largest output values of the network. If that difference is less than a threshold H, the classification of the input pattern is rejected due to insufficient evidence. Otherwise, the maximum of the two largest values determines the classification of the input. In our implementation, H was set to 0.1.

The population size P of the genetic algorithm approach was set equal to 20, that is, equal to the number of individual experiments performed for each database. The architecture utilized for each database was the same as that utilized in the experimentation with the BP and BP_WE approaches. The initial population was formed by first encoding the initial network and then copying it P times (the initial encoding is going to be the same for each network since the same network architecture is used for all of them). The parameter values of each network (β_{WE}, w_0, and initial weights) were chosen exactly the same as those used in the individual experiments using the BP and BP_WE approaches. In other words, the setting of the initial population was exactly the same as the initial setting of the networks used in the 20 individual experiments performed for each database using the BP and BP_WE approaches. It should be mentioned that although we could have chosen the β_{WE} values from a much smaller range of values for each dataset (our experimental results of Section 2.1 show that given a set of initial weights, the optimimum range of β_{WE} values might be quite different from database to database), we have intentionally made the problem more difficult, in order to challenge the genetic algorithm approach, by choosing the β_{WE} values from the broader interval of [10–100] for all the databases. In fact, we use the same β_{WE} values for all the databases.

The evaluation of each network from the population was performed by first training each network for a number of epochs using the weight elimination technique. The number of epochs used to train each network was about 10% of the average

number of epochs required by the BP approach to converge for the same problem (average over 20 different experiments performed for each database using the BP approach). This decision was based on the ideas presented in [6]. After a network in the population has been trained for a number of epochs, we compute its classification performance (G_{net_val} - classification performance over the validation set) and its size (E_{net_size}-effective number of connections over the total number of connections). Connections with small weights are not removed before the genetic algorithm has converged, as for the BP_WE approach. After the genetic algorithm had converged, we removed all the weak connections and we trained each network for a few more epochs (3–4) using the BP approach to account for any error increase due to the removal of the weak connections. This is again the same procedure followed in the case of the BP_WE approach.

The convergence of the genetic algorithm was determined by considering the improvement I_n at each generation. The improvement I_n at the nth generation is defined as the average fitness at the nth generation over the average fitness at the $(n-1)$th generation. Using this definition, we can allow the formation of a new population at step $(n+1)$ if the improvement at step n is better or equal than the improvement at step $(n-1)$. To avoid early convergence, we have used a more robust criterion based on the average improvement A_n defined as follows:

$$A_n = \gamma A_{n-1} + (1-\gamma) I_n. \tag{6}$$

A_n is the average improvement at step n and γ is a constant usually chosen very close to 1.0. Here, γ was set equal to 0.9. A_0 can be computed by evaluating each member of the initial population before evolution begins. New populations are allowed to form as far as the average improvement keeps increasing, that is while $A_n \geq A_{n-1}$. After the genetic algorithm had converged, each member of the final population was tested on the test set, using the methodology described earlier in the case of the BP and BP_WE approaches.

In all the simulations performed the learning rate and momentum were both set equal to 0.1. Sigmoidal activation functions and weight updates after every pattern presentation were also used. The generation gap in the genetic algorithm approach was set equal to 0.9 while the parameter which determines the number of best fit copies in future populations (called *Cmult* in [11]) was set equal to 1.5. The crossover and mutation probabilities were chosen 0.6 and 0.001 correspondingly. λ_{0_WE} and λ_{0_GA} were set equal to 1 (we did not experiment with different values since both λ_{0_WE} and λ_{0_GA} are simply scaling constants). Different β_{GA} values (Eq. (4)) were used during our experimentation. The best solutions obtained correspond to β_{GA} values in the range of [1.0–10.0]. For β_{GA} values much greater than 10.0, we did not observe a great reduction in terms of network size. Although our initial goal was to use the same β_{GA} value for all the databases (see also the related discussion in the conclusion), we observed variations in our results using different β_{GA} values from database to database. However, the deviation of the results for different β_{GA} values within this range was not very large in most cases. The β_{GA} values that yielded the best results for each database are given in the next subsections.

The experimental results for each one of the databases are presented next. For each database, the results are summarized in a table where the first column indicates the method used while the next columns indicate the performance of the method on the training and test sets as well as the reduction in network size achieved by each method. For each approach, we report the average performance and standard deviation as well as the best and worst solutions found. It is important for the reader to keep in mind that the criterion used for choosing the best and worst networks is their performance on the test set (columns eight and nine). The best and worst results in the case of the training set (columns four and five) correspond to the performance of the best and worst networks (in terms of test set) on the training set. Similarly, the best and worst number of weights (last two columns) correspond to the size of the best and worst networks (in terms of the test set) found. Thus, the reader should not be confused when, for example, the size of a network reported under the "best" column in the network size field (column before the last), is larger than the size of the network reported under the "worst" column (last column). This is because the larger size network has better performance on the test set than the smaller size network. The same applies when the performance of the networks on the train set is considered. The reader though can get an idea of the actual best-worst network sizes and performances on the training set by considering the reported averages and standard deviations for each case.

4.1. Numbers database

This is an artificial database which consists of noisy versions of machine printed numbers, digitized in a 7×9 grid. There are 10 classes. The training set consists of 150 examples, the validation set of 50 examples and the test set of 150 examples. The architecture chosen for this experiment was a fully connected two-layer network with 63 nodes in the input layer, 40 nodes in the hidden layer and 10 nodes in the output layer. The total number of weights and biases for this architecture is 2970. Table 2 illustrates the results.

The best solution was obtained by the GA1_BP_WE method (85.3% correct on the test set), while the best solution obtained by the GA_BP_WE method was also better than the best solutions obtained by the BP and BP_WE methods. Note also that the

Table 2
Results using the Numbers database

Method	Train				Test				Network size			
	ave	sd	best	worst	ave	sd	best	worst	ave	sd	best	worst
BP	1.0	0.0	1.0	1.0	0.813	0.01	0.833	0.787	2970.0	0.0	2970	2970
BP_WE	1.0	0.0	1.0	1.0	0.815	0.02	0.833	0.793	1926.4	1303.5	912	1872
GA_BP_WE	1.0	0.0	1.0	1.0	0.82	0.019	0.84	0.807	912.2	49.4	949	882
GA1_BP_WE	1.0	0.0	1.0	1.0	0.835	0.03	0.853	0.8	1013.9	220.9	922	1267

Table 3
Results using the Ionosphere database

Method	Train				Test				Network size			
	ave.	sd	best	worst	ave.	sd	best	worst	ave.	sd	best	worst
BP	1.0	0.0	1.0	1.0	0.936	0.01	0.942	0.925	1112.0	0.0	1112	1112
BP_WE	0.988	0.07	0.985	1.0	0.943	0.026	0.960	0.925	264.9	446.0	37	187
GA_BP_WE	0.977	0.01	0.99	0.97	0.974	0.005	0.975	0.967	87.2	8.7	83	79
GA1_BP_WE	0.972	0.01	0.985	0.95	0.973	0.01	0.983	0.958	68.5	32.2	70	43

worst performance solutions obtained by the genetic approaches are a little better than the worst solutions found by the other two methods. In terms of network size, the best solutions obtained by the genetic algorithm approaches were comparable to the best solutions obtained by the BP and BP_WE approaches. On the average, the size of the networks obtained by the genetic approaches is much smaller (see also the corresponding standard deviations). The best results in the case of both GA_BP_WE and GA1_BP_WE were obtained with $\beta_{GA} = 2.0$.

4.2. *Ionosphere database*

This database consists of radar data. It contains 2 classes and 351 instances. The number of attributes is 34. The database is distributed into two different files, a training file including 200 instances and a test file including 151 instances. In order to create a validation set we split the test file into two different files. The first consisted of 120 examples and was our actual test set, while the second consisted of 31 examples and was our validation set. The architecture chosen for this experiment was a fully connected, two-layer network with 34 nodes in the input layer, 30 nodes in the hidden layer and 2 nodes in the output layer. The total number of weights and biases for this architecture is 1112.

The results obtained are illustrated in Table 3. The GA1_BP_WE approach has improved generalization by 2.3% and the GA_BP_WE approach by 1.5% (best solutions). It should be mentioned that the best solution obtained by the BP_WE method has almost half the weights of the best solutions found by the GA_BP_WE and GA1_BP_WE approaches. However, this may have caused by overpruning, which may have prevented BP_WE from further improving its generalization performance. On the average, the network sizes obtained by the genetic approaches are again much smaller. The best results in the case of GA_BP_WE method were obtained using $\beta_{GA} = 2.0$, while in the case of GA1_BP_WE using $\beta_{GA} = 10.0$.

4.3. *Soybean database*

The Soybean disea ses database is well known in the field of machine learning. There are different versions of this database. Here we have used the soybean-large

Table 4
Results using the Soybean database

Method	Train				Test				Network size			
	ave.	sd	best	worst	ave.	sd	best	worst	ave.	sd	best	worst
BP	1.0	0.0	1.0	1.0	0.812	0.01	0.830	0.79	3075.0	0.0	3075	3075
BP_WE	0.999	0.006	1.0	1.0	0.816	0.002	0.827	0.80	2209.6	1335.8	2213	2493
GA_BP_WE	0.998	0.004	1.0	1.0	0.823	0.015	0.839	0.81	1203.9	235.6	1116	1460
GA1_BP_WE	1.0	0.0	1.0	1.0	0.833	0.004	0.835	0.83	1323.0	63.7	1309	1348

data set which consists of 562 instances. The number of attributes is 35 while the number of classes is 15. For the training set we selected 254 examples, 53 examples were selected for the validation set, and the rest 255 examples were put aside for the test set. The architecture chosen for this experiment was a fully connected network again with 35 nodes in the input layer, 60 nodes in the hidden layer, and 15 nodes in the output layer. The total number of weights and biases for this architecture is 3075. The results shown in Table 4 demonstrate that the generalization performance of the best genetic solutions is very close to these of the BP and BP_WE approaches.

However, the size of the networks obtained by the genetic approaches are much smaller than the size of the networks obtained using the BP_WE approach. This may have been caused by an early stopping of the pruning process in the BP_WE approach; it may also be the reason that BP_WE did not achieve a good generalization. Note also that BP (unpruned network) generalizes better than BP_WE. The best results in the case of GA_BP_WE were obtained with $\beta_{GA} = 4.0$, while in the case of GA1_BP_WE using $\beta_{GA} = 10.0$.

4.4. Breast-cancer database

This database consists of 699 examples. The number of attributes is 10 while the number of classes is 2. Since this database is distributed again into a single file, we split the data into three different sets: a training set (200 instances), a validation set (99 instances), and a test set (400 instances). The architecture chosen for this experiment was a fully connected network with 10 nodes in the input layer, 40 nodes in the hidden layer, and 2 nodes in the output layer. The total number of weights and biases for this architecture is 522. The genetic algorithm approaches have here also shown some improvement in terms of generalization. The results are shown in Table 5. It is worth noting that here the worst performance achieved by the BP_WE method was only 25%. Note also the standard deviation of the performance of the BP_WE method on the test set (0.553), due to excessive overpruning (in the worst solution found, only 9 weights remained after pruning). However, the genetic approaches avoided such solutions (the generalization performance of the worst solutions found by both approaches was 72%). The size of the best networks obtained are almost the same in

Table 5
Results using the Breast-cancer database

Method	Train				Test				Network size			
	ave.	sd	best	worst	ave.	sd	best	worst	ave.	sd	best	worst
BP	1.0	0.0	1.0	1.0	0.957	0.007	0.97	0.94	522.0	0.0	522	522
BP_WE	0.926	0.256	0.975	0.9	0.865	0.553	0.975	0.25	17.3	14.0	19	9
GA_BP_WE	0.930	0.260	0.98	0.56	0.950	0.160	0.98	0.72	22.1	16.1	16	14
GA1_BP_WE	0.930	0.260	0.975	0.56	0.940	0.160	0.98	0.72	17.2	6.2	13	14

all methods. The best results in the case of both GA_BP_WE and GA1_BP_WE were obtained using $\beta_{GA} = 2.0$.

4.5. *Wine database*

The Wine database is a relatively "easy" to train database. It consists of 178 instances. The number of attributes is 13 while the number of classes is 3. The data are distributed into a single file and thus we split it into three different files: a training file (75 instances), a validation file (28 instances), and a test file (75 instances). The architecture chosen for this experiment was a fully connected network with 13 nodes in the input layer, 35 nodes in the hidden layer, and 3 nodes in the output layer. The total number of weights and biases for this architecture is 598. Significant generalization improvement has been achieved by the genetic approaches as is illustrated in Table 6. The GA_BP_WE approach has improved generalization by 2.7%, while the GA1_BP_WE approach by 1.4% (best solutions). The best network sizes are almost the same for the BP_WE and the genetic approaches, while a much smaller average network size was obtained by the genetic approaches. The best results in the case of both GA_BP_WE and GA1_BP_WE were obtained using $\beta_{GA} = 2.0$.

4.6. *Iris database*

The Iris database is perhaps the best known database in the pattern recognition literature. It contains 3 classes of 50 instances each (total number of instances is 150), where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other. The number of attributes is 4, all of which are real valued. Since this database is distributed into a single file, we have randomly chosen 90 instances for training, 15 instances for validation, and the rest 45 instances for testing. The architecture chosen for this experiment was a fully connected network with 4 nodes in the input layer, 35 nodes in the hidden layer, and 3 nodes in the output layer. The total number of weights and biases for this architecture is 283. The results illustrated in Table 7 show that the generalization performance of all the methods is perfect (100%). However, the average size of the networks obtained using the genetic approaches is much smaller than the average size

Table 6
Results using the Wine database

Method	Train				Test				Network size			
	ave.	sd	best	worst	ave.	sd	best	worst	ave.	sd	best	worst
BP	1.0	0.0	1.0	1.0	0.919	0.006	0.92	0.907	598.0	0.0	598	598
BP_WE	1.0	0.0	1.0	1.0	0.907	0.027	0.933	0.88	183.0	159.35	56	238
GA_BP_WE	1.0	0.0	1.0	1.0	0.948	0.028	0.96	0.933	67.95	12.9	55	74
GA1_BP_WE	1.0	0.0	1.0	1.0	0.91	0.035	0.947	0.89	81.56	11.65	81	89

Table 7
Results using the Iris database

Method	Train				Test				Network size			
	ave.	sd	best	worst	ave.	sd	best	worst	ave.	sd	best	worst
BP	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	283.0	0.0	283	283
BP_WE	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	105.6	97	36	196
GA_BP_WE	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	36.2	2.1	34	36
GA1_BP_WE	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	34.2	1.2	33	35

of the networks obtained by the BP_WE approach. The best results in the case of both GA_BP_WE and GA1_BP_WE were obtained using $\beta_{GA} = 2.0$

4.7. Balance database

This database consists of 625 instances. The number of attributes is 4 while the number of classes is 3. Since this database is also distributed into a single file, we have randomly chosen 250 instances for training, 125 instances for validation, and the rest 250 instances for testing. The architecture chosen for this experiment was a fully connected network with 4 nodes in the input layer, 60 nodes in the hidden layer, and 3 nodes in the output layer. The total number of weights and biases for this architecture is 483. We have not achieved great generalization improvements with this database. As shown in Table 8, the genetic approaches have comparable generalization with the BP and BP_WE approaches. In terms of network size, both of the genetic approaches have shown great improvements. The best results in the case of GA_BP_WE were obtained using $\beta_{GA} = 4.0$ while in the case of GA1_BP_WE using $\beta_{GA} = 2.0$.

4.8. Cars database

This database consists of 846 instances. The number of attributes is 18 and the number of classes is 4. The database is distributed into a single file. A training file was

Table 8
Results using the Balance database

Method	Train				Test				Network size			
	ave.	sd	best	worst	ave.	sd	best	worst	ave.	sd	best	worst
BP	1.0	0.0	1.0	1.0	0.904	0.006	0.908	0.892	483.0	0.0	483	483
BP_WE	0.99	0.036	1.0	0.952	0.895	0.03	0.908	0.864	297.35	271.64	252	90
GA_BP_WE	1.0	0.0	1.0	1.0	0.908	0.0	0.908	0.908	73.67	2.5	72	78
GA1_BP_WE	1.0	0.0	1.0	1.0	0.908	0.004	0.912	0.904	72.25	4.27	71	66

Table 9
Results using the Cars database

Method	Train				Test				Network size			
	ave.	sd	best	worst	ave.	sd	best	worst	ave.	sd	best	worst
BP	1.0	0.0	1.0	1.0	0.78	0.02	0.795	0.74	1844.0	0.0	1844	1844
BP_WE	0.99	0.01	1.0	1.0	0.77	0.025	0.792	0.74	1549.8	817.55	1804	1180
GA_BP_WE	0.99	0.015	0.988	0.998	0.79	0.025	0.81	0.775	975.95	262.1	1026	968
GA1_BP_WE	0.98	0.03	0.995	0.975	0.777	0.02	0.795	0.763	1078.0	295.5	1132	906

created by choosing 400 instances. For the validation set we chose 100 instances, while the rest 346 instances were kept for the test set. The architecture chosen for this experiment was a fully connected network with 18 nodes in the input layer, 80 nodes in the hidden layer, and 4 nodes in the output layer. The total number of weights and biases for this architecture is 1844.

The results are illustrated in Table 9. Note that BP_WE's generalization performance is a little worst than BP's performance (both on the average and in terms of the best solution found). However, the genetic approaches have retained good generalization. In particular, GA_BP_WE has found a solution which is better (by 1.8%) that the best solution found by BP_WE. Note also that the GA_BP_WE approach has found solutions with a smaller network size. The GA1_BP_WE approach has not improve generalization it has given better results than the BP_WE approach in terms of network size. The best results in the case of GA_BP_WE were obtained with $\beta_{GA} = 1.0$, while in the case of GA1_BP_WE were obtained with $\beta_{GA} = 4.0$.

4.9. Discussion

Table 10 presents a summary of the best results obtained through our experimentation. The first column presents the database used while the next columns present the best network solutions found for each of the four methods we considered in our experiments. In particular, for each method we report the best generalization

Table 10
Comparison of the best solutions obtained

Data set	Generalization and network size of best solutions							
	BP		BP_WE		GA_BP_WE		GA1_BP_WE	
	test	size_reduction	test	size_reduction	test	size_reduction	test	size_reduction
Numbers	0.8333	0%	0.833	69.3%	0.84	68%	0.853	68.96%
Ionosphere	0.942	0%	0.96	96.67%	0.975	92.54%	0.983	93.7%
Soybean	0.83	0%	0.827	28%	0.839	63.7%	0.835	57.43%
Breast-cancer	0.97	0%	0.975	96.36%	0.98	96.93%	0.98	97.51%
Wine	0.92	0%	0.933	90.6%	0.96	90.6%	0.947	86.45%
Iris	1.0	0%	1.0	87.28%	1.0	87.98%	1.0	88.34%
Balance	0.908	0%	0.908	47.83%	0.908	85.1%	0.912	85.3%
Cars	0.795	0%	0.792	2.1%	0.81	44.36%	0.795	38.6%

performance achieved and the reduction in network size associated with this solution. It is obvious that the pruned networks obtained by the combination of genetic algorithms and weight elimination have the same or better generalization capabilities. In terms of network size, the results are much better in many cases (e.g., in the case of the Soybean database). In some cases, the solutions obtained were a little worse in terms of network size (Numbers and Ionosphere databases) but they had better generalization performance which is more critical. The use of the *crossover_first_layer_nodes* operator seems to be beneficial in many cases. However, our experimental results do not give us enough evidence to conclude that it is always superior than the *crossover_nodes* operator. In fact, the GA_BP_WE approach gave better results than the GA1_BP_WE approach in some cases (e.g., in the case of the Wine database). In the case of the GA_BP_WE approach, the best β_{GA} values were rather small in the range of [1.0–4.0]. In the case of the GA1_BP_WE approach, best results were obtained using large β_{GA} values (around 10.0 in some cases).

The proposed approach can be improved in a number of ways. First of all, we believe that the size of the validation set is very important. In all of the experiments performed, the validation sets used were rather small. However, we think that results can be further improved by using larger size validation data sets, when a sufficient number of data is available of course. This is because the evaluation of a network during evolution is strongly based on its generalization performance, which is estimated using the validation set. If the network generalizes well then its network size starts having some importance in the fitness evaluation, otherwise the size does not have any contribution. Obviously, if the validation set is smaller than appropriate for obtaining a good "estimate" of the generalization performance of the network, bad results can be obtained. If for example, the "true" generalization of the network is bad while the "estimated" generalization of the network is good, then the fitness evaluation will mostly favor networks having bad generalization and small network size. However, encouraging network size reduction before the network has achieved some

good generalization performance might lead to small size networks having a very poor generalization.

Another way to improve the proposed approach is by increasing the population size. In our experimentation, we set the population size equal to 20 in order to keep the time requirements of our experiments within some reasonable limits, given the fact that we are dealing with a large collection of databases. However, larger size populations (~ 50 or more) might allow the genetic algorithm to discover even better solutions. Of course, the need of using faster machines to run the experiments is inevitable. The above two improvements might also make the choice of appropriate β_{GA} values less data dependent. The consequence of this is very important. Given a problem to solve, we would be able to obtain a good solution (both in terms of network size and generalization) without trying different β_{GA} values for each database.

5. Conclusions

Pruning techniques represent a broad class of methods which try to restrict the number of free parameters in a network (i.e., weights) in order to improve generalization. Although pruning techniques reduce network size, they do not always improve generalization. In fact, smaller size networks obtained by applying weight pruning may have the same or even worst generalization performance than that of their unpruned counterparts. In this paper we proposed the coupling of genetic algorithms with weight elimination. Weight elimination is a well known pruning technique and the most general among the weight decay techniques. The objective of our work is to provide a method not only for reducing network size but also for preserving generalization.

An extensive experimental study involving one artificial and seven real databases has demonstrated that the coupling of genetic algorithms with weight elimination is a very promising approach. Actually, the framework of our approach is more general since other pruning techniques can be also coupled with genetic algorithms. Weight elimination depends on a number of parameter values whose choice can significantly affect the results. We show that small size networks can be obtained; however, the generalization performance of these networks is not always satisfactory. On the other hand, the networks obtained by the proposed approach not only are small in size but they also have good generalization capabilities.

The success of the proposed approach is mainly due to the interchange of information that takes place during evolution. Starting with a population of networks having various parameter settings, the genetic algorithm was capable of finding good solutions (both in terms of size and generalization) than the solutions found by training each one of the networks separately using the same parameter settings. To demonstrate the capabilities of the genetic algorithm approach, we used the same parameter settings for all the databases used. Although it was shown (Section 2) that the optimal parameter value ranges are data dependent, the genetic algorithm managed to find good solutions in all the cases.

Acknowledgements

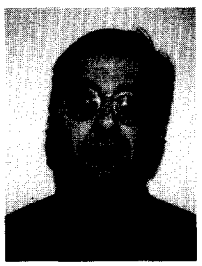
This research was supported in part by a grant from Harris Corporation.

References

[1] T. Ash, Dynamic node creation, Connection Sci. 1 (4) (1989) 365–375.
[2] E. Baum, D. Haussler, What size net gives valid generalization, Neural Comput. 1 (1989) 151–160.
[3] G. Bebis, M. Georgiopoulos, Feed-forward neural networks: why network size is so important, IEEE Potentials (Oct/Nov.) (1994) 27–31.
[4] G. Bebis, M. Georgiopoulos, T. Kasparis, Coupling weight elimination and genetic algorithms, in: Internat. Conf. on Neural Networks (ICNN-96), vol. 2, Washington, DC, June 1996, pp. 1115–1120.
[5] G. Bebis, G. Papadourakis, Implementation of character recognition using neural networks and traditional classifiers, Proc. NEURONET Internat. Symp. on Neural Networks and Neural Computing, Prague, 1990, pp. 33–36.
[6] M. Caudill, Evolutionary neural network, AI Expert, (March) (1991) 29–33.
[7] Y. Chauvin, A back-propagation algorithm with optimal use of hidden units, Adv. Neural Inform. Process. Systems 1 (1989) 519–526.
[8] J. Depenau, M. Moller, Aspects of generalization and pruning, in: Proc. World Congress on Neural Networks, vol. III, 1994, pp. 504–509.
[9] S. Fahlman, C. Lebiere, The Cascade-Correlation learning architecture, Adv. Neural Inform. Process. Systems 2 (1990) 524–532.
[10] M. Frean, The Upstart algorithm: a method for constructing and training feed-forward networks, Neural Comput. 2 (1990) 198–209.
[11] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[12] J. Grefenstette, Optimization of control parameters for genetic algorithms IEEE Trans. Systems Man Cybernet. 16(1) (1) (1986) 122–128.
[13] S. Hanson, L. Pratt, Comparing biases for minimal network construction with back-propagation, Adv. Neural Inform. Process. Systems 1 (1989) 177–185.
[14] S. Harp, T. Samad, A. Guha, Designing application-specific neural networks using the genetic algorithm, Adv. Neural Inform. Process. Systems 2 (1990) 447–454.
[15] B. Hassibi, D. Stork, Second order derivatives for network pruning: optimal brain surgeon, Adv. Neural Inform. Process. Systems 5 (1993)
[16] Y. Hirose, K. Yamashita, S. Hijiya, Back-propagation algorithm which varies the number of hidden units, Neural Networks 4 (1991) 61–66.
[17] K. Hornik, M. Stinchombe, Multilayer feed-forward networks are universal approximators, in: H. White et al. (Eds.), Artificial Neural Networks: Approximation and Learning Theory, Blackwell Press, Oxford, 1992.
[18] D. Hush, B. Horne, Progress in supervised neural networks, IEEE Signal Process. Mag. (Jan.) (1993) 8–39.
[19] C. Ji, R. Snapp, D. Psaltis, Generalizing smoothing constraints from discrete samples, Neural Comput. 2 (1990) 188–197.
[20] R. Kamimura, S. Nakanishi, Weight-decay as a process of redundancy reduction, in: Proc. World Congress on Neural Networks, vol. III, 1994, pp. 486–489.
[21] E. Karnin, A simple procedure for pruning back-propagation trained neural networks, IEEE Trans. Neural Networks, 1 (2) (1990) 239–242.
[22] J. Hertz, A. Krogh, R. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley, Reading, MA, 1991.
[23] A. Krogh, J. Hertz, A simple weight decay can improve generalization, Adv. Neural Inform. Process. Systems 4 (1992) 950–957.

[24] Y. Le Cun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, L. Jackel, Back-propagation applied to handwritten zip code recognition, Neural Comput. 1 (1989) 541–551.
[25] Y. Le Cun, J. Denker, S. Solla, Optimal brain damage, Adv. Neural Inform. Process. Systems 2 (1990) 598–605.
[26] D. MacKay, A practical bayesian framework for backpropagation networks, Neural Comput. 4 (1992) 448–472.
[27] M. Mezard, J. Nadal, Learning in feed-forward layered networks: the tiling algorithm, J. Phys. A 22 (1989) 2191–2204.
[28] G. Miller, P. Todd, S. Hegde, "Designing neural networks using genetic algorithms", 3rd Internat. Conf. on Genetic Algorithms, pp. 379–384.
[29] D. Montana, L. Davis, Training feed-forward neural networks using genetic algorithms, in: Proc. 11th Internat. Joint Conf. on Artificial Intelligence, 1989, pp. 762–767.
[30] M. Mozer, P. Smolensky, Skeletonization: a technique for trimming the fat from a network via relevance assessment, Adv. Neural Inform. Process. Systems 1(1989) 105–115.
[31] P. Murphy, D. Aha, UCI Repository of machine learning databases, [Machine-readable data repository], University of California, Department of Information and Computer Science, Irvine, CA, 1994.
[32] S. Nowlan, G. Hinton, Simplifying neural networks by soft weight sharing, Neural Comput. 4 (4) (1992) 473–493.
[33] J. Sietsma, R. Dow, Creating artificial neural networks that generalize, Neural Networks 4 (1991) 67–79.
[34] H.H. Thodberg, Improving generalization of neural networks through pruning, Internat. J. Neural Systems 1 (4) (1991) 317–326.
[35] A. Weigend, D. Rumelhart, B. Huberman, Generalization by weight elimination with application to forecasting, Adv. Neural Inform. Process. Systems 3 (1991) 875–882.
[36] M. Wynne-Jones, Node splitting: a constructive algorithm for feed-forward neural networks, Adv Neural Inform. Process. Systems 4 (1992) 1072–1079.
[37] Y. Yang, V. Honavar, Experiments with the cascade-correlation algorithm, Technical Report, Dept. of Computer Science, Iowa State University, Ames, IA.
[38] X. Yao, A review of evolutionary artificial neural networks, Internat. J. Intelligent Systems 8 (1993) 539–567.

George Bebis received the B.S. degree in Mathematics and the M.S. degree in Computer Science from the University of Crete, Greece, in 1987 and 1991, respectively, and the the Ph.D. degree in Electrical and Computer Engineering from the University of Central Florida, Orlando, in 1996.Currently, he is a Visiting Assistant Professor in the Department of Mathematics and Computer Science at the University of Missouri, St. Louis. His research interests include computer vision, image processing, neural networks, and genetic algorithms.



Michael Georgiopoulos received the Diploma in electrcial engineering from the National Technical University of Athens, Greece, in 1981, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Connecticut, Storrs, in 1983 and 1986, respectively. In 1987, he joined the University of Central Florida, Orlando, where he is currently an Associate Professor at the Department of Electrical and Computer Engineering. His research interests are in the area of pattern recognition, neural networks, fuzzy logic and genetic algorithms. He is also interested in applications of the above technologies in prediction, electromagnetics, signal/image processing and simulation. Dr. Georgiopoulos is a member of the IEEE, International Neural Network Society, and a member of the Technical Chamber of Greece.

Takis Kasparis received the Diploma of Electrical Engineering from the National Technical University of Athens, Greece in 1980, and the MEEE and Ph.D. degrees in Electrical Engineering from the City College of New York in 1982 and 1988. From 1985 until 1989 he was an electronics consultant for various firms in the New York area designing various single board computers and other types of digital hardware. In 1989 he joined the Electrical Engineering Department of the University of Central Florida, Orlando, where he is presently an associate professor. His main research areas are non-linear adaptive median filters for image filtering and restoration, texture analysis and segmentation problems, computer vision and digital signal processing for communications and audio processing. He has published over 50 papers in various scientific journals and conference proceedings.