An Iterative Pruning Algorithm for Feedforward Neural Networks

Giovanna Castellano, Anna Maria Fanelli, *Member, IEEE,* and Marcello Pelillo, *Member, IEEE*

***Abstract*— The problem of determining the proper size of an artificial neural network is recognized to be crucial, especially for its practical implications in such important issues as learning and generalization. One popular approach tackling this problem is commonly known as pruning and consists of training a larger than necessary network and then removing unnecessary weights/nodes. In this paper, a new pruning method is developed, based on the idea of iteratively eliminating units and adjusting the remaining weights in such a way that the network performance does not worsen over the entire training set. The pruning problem is formulated in terms of solving a system of linear equations, and a very efficient conjugate gradient algorithm is used for solving it, in the least-squares sense. The algorithm also provides a simple criterion for choosing the units to be removed, which has proved to work well in practice. The results obtained over various test problems demonstrate the effectiveness of the proposed approach.**

***Index Terms*— Feedforward neural networks, generalization, hidden neurons, iterative methods, least-squares methods, network pruning, pattern recognition, structure simplification.**

I. Introduction

DESPITE many advances, for neural networks to find general applicability in real-world problems, several questions must still be answered. One such open question involves determining the most appropriate network size for solving a specific task. The network designer's dilemma stems from the fact that both large and small networks exhibit a number of advantages. When a network has too many free parameters (i.e., weights and/or units) not only is learning fast [2]–[5], but local minima are more easily avoided [6]. In particular, a theoretical study [7] has shown that when the number of hidden units equals the number of training examples (minus one), the backpropagation error surface is guaranteed to have no local minima. Large networks can also form as complex decision regions as the problem requires [8] and should exhibit a certain degree of fault tolerance under damage conditions (however, this appears not to be as obvious as might intuitively have been expected [9]). On the other hand, both theory [10] and experience [11]–[13] show that networks with few free parameters exhibit a better generalization performance, and this is explained by recalling the analogy between neural network learning and curve fitting. Moreover, knowledge embedded in small trained networks is presumably easier to interpret and thus the extraction of simple rules can hopefully be facilitated [14]. Lastly, from an implementation standpoint, small networks only require limited resources in any physical computational environment.

To solve the problem of choosing the right size network, two different incremental approaches are often pursued (e.g., [1], [15], [16] and references therein). The first starts with a small initial network and gradually adds new hidden units or layers until learning takes place. Well-known examples of such growing algorithms are cascade correlation [17] and others [18]–[20]. The second, referred to as *pruning*, starts with a large network and excises unnecessary weights and/or units. This approach combines the advantages of training large networks (i.e., learning speed and avoidance of local minima) and those of running small ones (i.e., improved generalization) [21]. However it requires advance knowledge of what size is "large" for the problem at hand, but this is not a serious concern as upper bounds on the number of hidden units have been established [22]. Among pruning algorithms there are methods that reduce the excess weights/nodes during the training process, such as penalty term methods [23]–[25] and the gain competition technique [26], and methods in which the training and pruning processes are carried out in completely separate phases.[1] The latter approach is exemplified by Sietsma and Dow's two-stage procedure [28], [29], Mozer and Smolensky's skeletonization technique [32], the optimal brain damage (OBD) algorithm [30] and the optimal brain surgeon (OBS) [31]. These post-training pruning procedures do not interfere with the learning process [3], but they usually require some retraining to mantain the performance of the original network.

In this paper, a novel posttraining pruning method for arbitrary feedforward networks is proposed, which aims to select the optimal size by gradually reducing a large trained network. The method is based on the simple idea of iteratively removing hidden units and then adjusting the remaining weights with a view to maintaining the original input–output behavior. This is accomplished by imposing that, at each step, the net input of the units fed by the unit being removed be approximately the same as the previous one, across the entire training set. This amounts to defining a system of linear equations that we solve in the least-squares sense using an efficient preconditioned conjugate gradient procedure. Although the approach does not itself provide a criterion for choosing the units to be

Manuscript received May 16, 1994; revised February 13, 1995, January 16, 1996, and October 8, 1996.

G. Castellano is with the Istituto di Elaborazione dei Segnali e delle Immagini, Consiglio Nazionale delle Ricerche, 70126 Bari, Italy.

A. M. Fanelli is with the Dipartimento di Informatica, Universita' di Bari, 70126 Bari, Italy.

M. Pelillo is with the Dipartimento di Matematica Applicata e Informatica, Universita' "Ca' Foscari" di Venezia, 30173 Venezia Mestre, Italy.

Publisher Item Identifier S 1045-9227(97)01755-4.

[1] This idea has been proved to be quite effective in the analogous problem of determining the proper dimension of a classification tree, thereby improving its generalization performance [27].

removed, a computationally simple rule has been derived directly from the particular class of least-squares procedures employed, and has proved to work well in practice. Besides sharing the advantages of posttraining pruning procedures, the one proposed here exhibits a number of additional features. First, in contrast with many existing algorithms (e.g., [28], [29], [34], [35]), ours does not make use of any working parameter and this frees the designer from a lengthy, problem-dependent tuning phase. Second, the proposed algorithm does not require any retraining phase after pruning, like the OBS procedure [31], but it requires far less computational effort (see Section III-D for details). Finally, although we shall focus primarily on hidden unit removal in feedforward networks, the approach presented here is quite general and can be applied to networks of arbitrary topology [36] as well as to the elimination of connections.

Some algorithms bear similarities with the proposed one. First, the basic idea of removing redundant hidden units and properly adjusting remaining weights was proposed by Sietsma and Dow [28], [29] who heuristically derived a two-stage pruning procedure for layered networks. While their primary goal was to devise specific rules for locating redundant units, we focus mainly on directly solving a linear system without explicitly taking into account the redundancy of individual units. Furthermore, as described in Section IV, their stage-one pruning rules are but special consistency conditions for the system we solve, and the remaining weights update corresponds to a particular solution of that system. Second, the Frobenius approximation reduction method (FARM) [37] selects the set of hidden units in a layered network so as to preserve, like in our approach, the original training-set behavior. Our algorithm, however, differ significantly as regards how both the units to be removed are selected and the weights of the reduced networks are derived. Further, for FARM to be applicable, the optimal number of hidden units must be determined in advance. This can be carried out by a computationally expensive singular value decomposition (SVD) procedure [38], [39], but it may fail to detect linear dependencies among hidden neurons even in very redundant networks [9]. Finally, system formulations similar to ours were used in [3] and [40], but for different goals.

The definitions and notations used in this paper are introduced in Section II. In Section III we formulate the pruning problem in terms of solving a system of linear equations and derive the pruning algorithm. In Section IV the relations between the proposed pruning approach and that of Sietsma and Dow are pointed out. In Section V experimental results on different test problems are presented. Finally, Section VI gives the summary and conclusions.

II. Definitions and Notations

Since the pruning algorithm presented in this paper can be applied to arbitrary feedforward networks, not necessarily layered or fully connected, some definitions and notations must be introduced. A feedforward artificial neural network can be represented by an acyclic weighted directed graph $N = (V, E, w)$ where $V = \{0, 1, \cdots, n\}$ is a set of $n + 1$ units (or neurons), $E \subseteq V \times V$ is a set of connections, and w: $E \rightarrow R$ is a function that assigns a real-valued weight $w(i, j)$ to each connection $(i, j) \in E$; positive weights correspond to excitatory connections and negative weights to inhibitory connections. In the following, the more familiar notation w_{ij} will be used instead of $w(i, j)$.

Each unit $i \in V$ is associated with its own *projective field*

$$P_i = \{j \in V : (i, j) \in E\} \tag{1}$$

which represents the set of units that are fed by unit i, and its own *receptive field*

$$R_i = \{j \in V : (j, i) \in E\} \tag{2}$$

which is the set of units that feed unit i. In the special case of layered fully connected networks, the receptive and projective fields of a given unit are simply its preceding and succeeding layers, if any, respectively. In the following, we will denote the cardinality of P_i by p_i and the cardinality of R_i by r_i. The set of units V is divided into three subsets: the set of input units V_I, having an empty receptive field, the set of output units V_O, having an empty projective field, and the set of hidden units V_H. As usual, it is assumed that a particular input unit (here labeled by zero) works as a bias unit which is permanently clamped at $+1$ and is connected to any noninput unit. Fig. 1 shows an example of feedforward network architecture and illustrates the notations introduced above.

The network operates as follows. Input units receive from the external environment an activity pattern which is propagated to all units in the corresponding projective fields. Every noninput unit $i \in V_H \cup V_O$, in turn, receives from its own receptive field R_i a net input given by

$$\xi_i = \sum_{j \in R_i} w_{ji} y_j \tag{3}$$

where y_j represents the output value of unit j, and sends to its projective field P_i an output signal equal to

$$y_i = f(\xi_i) \tag{4}$$

f being an arbitrary differentiable activation function. The process continues until the output units are reached and their outgoing signals are thus taken to be the actual response of the network. A common choice for the function f is the logistic function $f(x) = 1/(1 + e^{-x})$ but no restriction is placed on the type of activation function used.

III. The Algorithm

A. Problem Formulation

Our approach to network minimization consists of successively removing hidden units after the network has been trained for satisfactory performance. It is assumed that learning is carried out over a sample of M training patterns by means of an arbitrary learning procedure (note that the pruning algorithm developed in the following is completely independent of the particular training procedure).

Assume that hidden unit $h \in V_H$ has somehow been identified as a candidate for removal (later on, we shall address

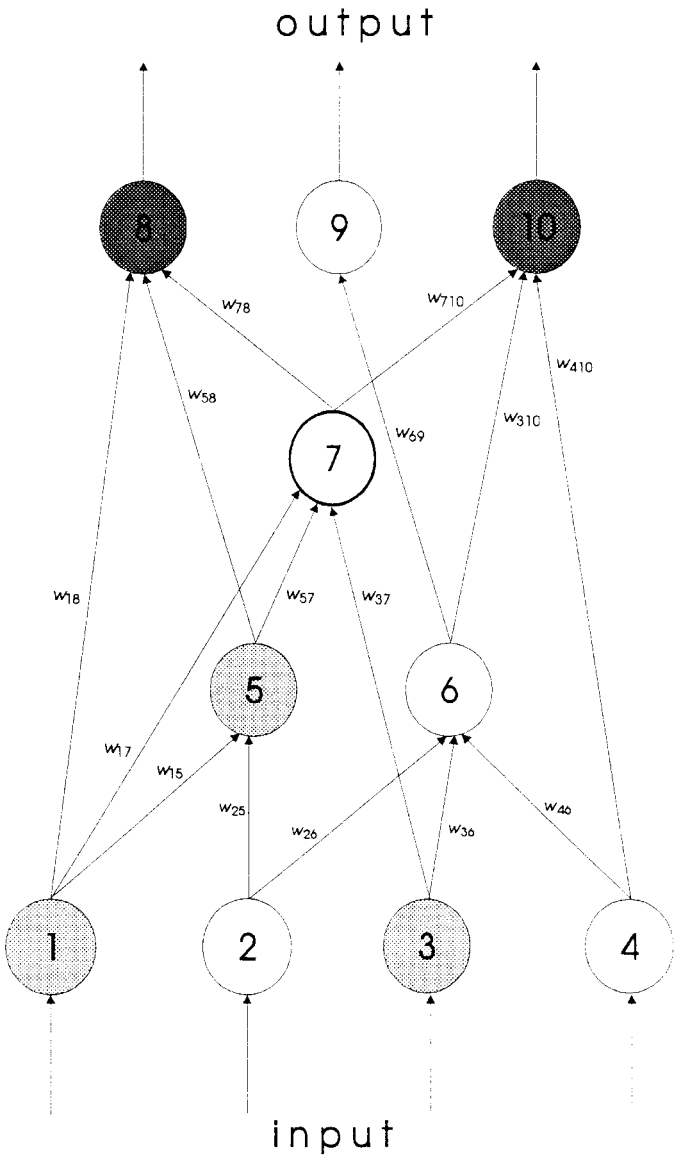

Fig. 1. Example of feedforward network architecture to illustrate notations (the bias unit zero has not been included to make representation easier). Here, the set of input units is $V_I = \{1, 2, 3, 4\}$, the set of hidden units is $V_H = \{5, 6, 7\}$, and the set of output units is $V_O = \{8, 9, 10\}$. As an example, the receptive field of node 7 is $R_7 = \{1, 3, 5\}$ (light-gray units) whereas its projective field is $P_7 = \{8, 10\}$ (dark-gray units).

the question of how to choose the units to be excised). First, the elimination of unit h involves removing all its incoming and outgoing connections. More precisely, this amounts to stating that the new pruned network will have the following set of connections:

$$E_{\text{new}} = E_{\text{old}} - (\{h\} \times P_h \cup R_h \times \{h\})$$

where E_{old} is the connection set of the preceding unpruned network.

Removing the in/out h's connections is not the whole story. Our approach to network pruning consists first of removing unit h and then appropriately adjusting the weights incoming into h's projective field so as to preserve the overall network input/output behavior of the training set. To be more precise, let $i \in V$ be a unit of h's projective field P_h. Its net input upon presentation of pattern $\mu \in \{1, \cdots, M\}$ is given by

$$\xi_i^{(u)} = \sum_{j \in R_i} w_{ji} y_j^{(\mu)}$$

where $y_j^{(\mu)}$ denotes the output of unit j corresponding to pattern μ. After removal of h, unit i will take its own input from $R_i - \{h\}$.[2] In order to maintain the original network behavior, we attempt to adjust the remaining weights incoming into node i, i.e., the w_{ji}'s for all $j \in R_i - \{h\}$, so that its new net input remains as close as possible to the old one, for all the training patterns (see Fig. 2). This amounts to requiring that the following relation holds:

$$\sum_{j \in R_i} w_{ji} y_j^{(\mu)} = \sum_{j \in R_i - \{h\}} (w_{ji} + \delta_{ji}) y_j^{(\mu)} \tag{5}$$

for all $\mu = 1 \cdots M$ and $i \in P_h$, where the δ_{ij}'s are appropriate adjusting factors to be determined. Simple algebraic manipulations yield

$$\sum_{j \in R_i - \{h\}} \delta_{ji} y_j^{(\mu)} = w_{hi} y_h^{(\mu)} \tag{6}$$

which is a (typically overdetermined) system of Mp_h linear equations in the $\kappa_h = \Sigma_{i \in P_h} (r_i - 1)$ unknowns $\{\delta_{ji}\}$. Observe that κ_h represents the total number of connections incoming into h's projective field after unit h has been removed.

It is convenient to represent system (6) in a more compact matrix notation. To do so, consider for each unit $i \in P_h$ the M-vector $\overline{y}_i$ composed of the output values of unit i upon presentation of the M training patterns

$$\overline{y}_i = (y_i^{(1)}, \cdots, y_i^{(M)})^T.$$

Also, let $Y_{i,h}$ denote the $M \times (r_i - 1)$ matrix, whose columns are the output vectors of i's new receptive field $R_i - \{h\}$, that is,

$$Y_{i,h} = [\overline{y}_{j_1} \quad \overline{y}_{j_2} \quad \cdots \quad \overline{y}_{j_{r_i - 1}}] \tag{7}$$

where the indexes j_k, for all $k = 1 \cdots r_i - 1$, vary in $R_i - \{h\}$. Now, we have to solve the p_h disjoint systems

$$Y_{i,h} \overline{\delta}_i = \overline{z}_{i,h} \tag{8}$$

for every $i \in P_h$, where $\overline{\delta}_i$ is the unknown vector, and

$$\overline{z}_{i,h} = w_{hi} \overline{y}_h. \tag{9}$$

Finally, putting these systems together, we obtain

$$Y_h \overline{\delta} = \overline{z}_h \tag{10}$$

where

$$Y_h = \text{diag}(Y_{i_1,h}, Y_{i_2,h}, \cdots, Y_{i_{p_h},h}) \tag{11}$$

$$\overline{\delta} = (\overline{\delta}_{i_1}^T, \overline{\delta}_{i_2}^T, \cdots, \overline{\delta}_{i_{p_h}}^T)^T \tag{12}$$

and

$$\overline{z}_h = (\overline{z}_{i_1,h}^T, \overline{z}_{i_2,h}^T, \cdots, \overline{z}_{p_h,h}^T)^T. \tag{13}$$

Here, the indexes i_k's $(k = 1 \cdots p_h)$ vary in P_h.

[2] For convenience, it is assumed here that the following condition is always satisfied $\forall i \in P_h$: $R_i - \{0, h\} \neq \emptyset$, which means that, after removing node h, all the units of its projective field will receive at least one input other than the bias signal.

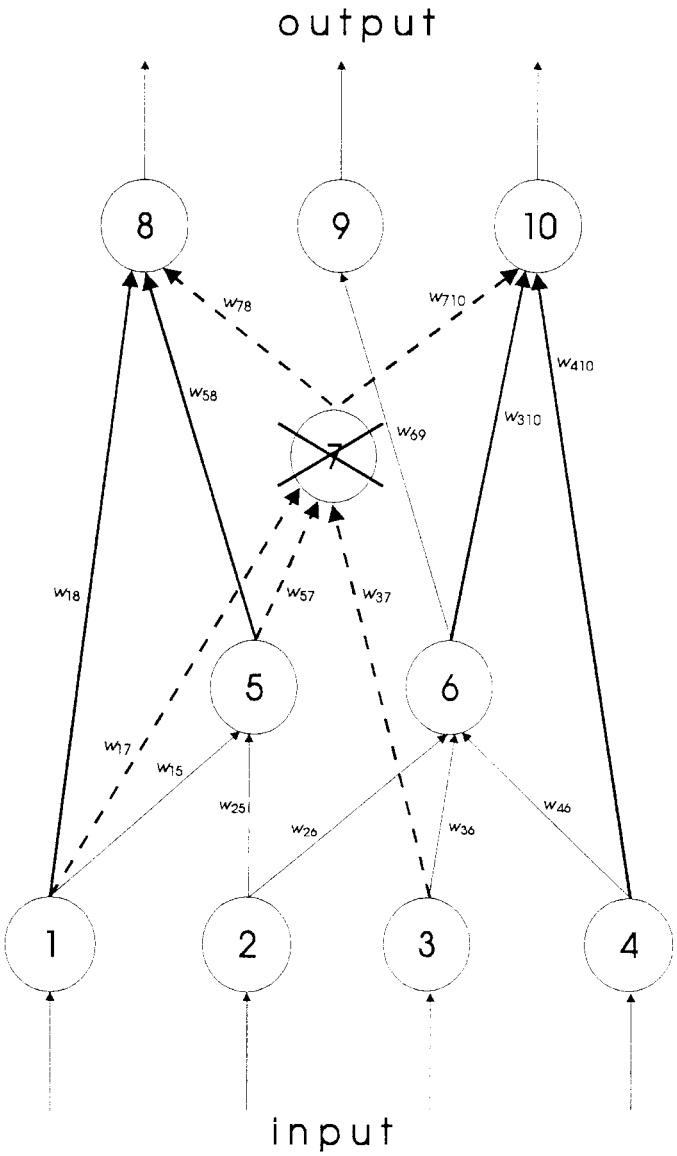

Fig. 2. Illustration of the pruning process. Unit 7 has been selected to be removed. After pruning, all its incoming and outgoing connections (dashed lines) will be excised and all the weights incoming into its projective field (thick lines) will be properly adjusted.

To summarize, the core of the proposed pruning approach consists of solving the linear system defined in (10).[3] This will be accomplished in the least-squares sense, which amounts to solving the following problem:

$$\text{minimize}||\overline{z}_h - Y_h\overline{\delta}||_2. \tag{14}$$

Since we start with larger than necessary networks, linear dependencies should exist among the output vectors of the hidden units, and the matrix Y_h is therefore expected to be rank deficient, or nearly so. This implies that an infinite number of solutions typically exist for problem (14) and in such cases the one having the lowest norm is generally sought. Because of the particular meaning of our unknown vector $\overline{\delta}$, however, we deviate from this practice and will be satisfied with any solution vector, regardless of its length.

We emphasize that, although it was developed to prune off hidden units, the approach presented here can be applied to remove single connections as well. Supposing that connection $(h,i) \in E$ is to be removed, the idea is to distribute the weight w_{hi} over the connections incoming into unit i, in such a way that i's net input remains unchanged across the training set. It is easy to see that this condition is equivalent to solving a single system of M equations and $r_i - 1$ unknowns that is identical to (8).

B. Weight Adjustment by a Conjugate Gradient Least-Squares Method

To solve the least-squares problem (14), standard QR factorization methods with column pivoting or the SVD technique can be employed [41]. These methods, however, turn out to be impractical for large and sparse coefficient matrices, and iterative conjugate gradient (CG) methods, especially when used in conjunction with appropriate preconditioning techniques [41]–[43], are advantageous in this case. Moreover, CG-methods are particularly attractive because they do not depend upon parameters that are difficult to choose. In this work we made use of a general preconditioned CG-method derived by Björck and Elfving in [44], which proved to be extremely efficient thanks not only to its modest computational cost per step, but also to its fast rate of convergence. This section is devoted to a brief description of Björck and Elfving's so-called CGPCNE algorithm,[4] as applied to problem (14).

Consider our original system $Y_h\overline{\delta} = \overline{z}_h$, as defined in (10)–(13). For notational convenience the index h, which denotes the unit being removed, will be suppressed. Write the matrix Y_{TY} as

$$Y^TY = L + D + L^T \tag{15}$$

and recall from the previous section that κ denotes the number of columns of Y. Here, D is a $\kappa \times \kappa$ diagonal matrix whose nonzero elements are defined to be

$$(D)_{jj} = ||Y(:,j)||_2^2 \tag{16}$$

(the notation $Y(:,j)$ indicates the jth column of matrix Y), and L is a strictly lower triangular matrix constructed as

$$(L)_{jk} = Y(:,j)^TY(:,k), \qquad j > k. \tag{17}$$

Now, consider the normal system $Y^TY\overline{\delta} = Y^T\overline{z}$, and let C_ω denote the following preconditioning matrix:

$$C_\omega = (D + \omega L)D^{-1/2} \tag{18}$$

where ω represents a relaxation parameter in the interval (0, 2) used to control the convergence rate of the algorithm. Making the change of variables $\overline{\gamma} = C_\omega^T\overline{\delta}$, we finally arrive at the preconditioned normal system

$$C_\omega^{-1}Y^TYC_\omega^{-T}\overline{\gamma} = C_\omega^{-1}Y^T\overline{z}. \tag{19}$$

[3] Instead of directly solving system (10), one can independently solve the p_h disjoint systems (8). Here, the former approach was preferred because of the moderate dimensions of the networks considered. When large networks are involved, the latter approach can well be pursued.

[4] The acronym CGPCNE is not explicitly defined in [44]; however, it should stand for "conjugate gradient preconditioned normal equation."

Applying the CG-method to (19), the CGPCNE algorithm for solving problem (14) results. It begins with an initial tentative solution $\overline{\delta}_0 \in$ range (Y^T) and iteratively produces a sequence of points $\{\overline{\delta}_k\}$ in such a way that the residuals

$$\rho(\overline{\delta}_k) = \|\overline{z} - Y\overline{\delta}_k\|_2^2 \tag{20}$$

are monotonically decreased, i.e., $\rho(\overline{\delta}$ for all $k \geq 0 \rho(\overline{\delta}_k + 1) < \rho(\overline{\delta}_k)$.

Algorithm 1: CGPCNE Least-Squares Algorithm
1) $\overline{r}_0 := \overline{z} - Y\overline{\delta}_0$
2) $\overline{s}_0 := C_\omega^{-1} Y^T \overline{r}_0$
3) $\overline{p}_0 := \overline{s}_0$
4) $k := 0$
5) repeat
   6) $\overline{q}_k := Y C_\omega^{-T} \overline{p}_k$
   7) $\alpha_k := \|\overline{s}_k\|_2^2 / \|\overline{q}_k\|_2^2$
   8) $\overline{r}_{k+1} := \overline{r}_k - \alpha_k \overline{q}_k$
   9) $\overline{s}_{k+1} := C_\omega^{-1} Y^T \overline{r}_{k+1}$
   10) $\beta_k := \|\overline{s}_{k+1}\|_2^2 / \|\overline{s}_k\|_2^2$
   11) $\overline{p}_{k+1} := \overline{s}_{k+1} + \beta_k \overline{p}_k$
   12) $\overline{\delta}_{k+1} := \overline{\delta}_k + \alpha_k C_\omega^{-T} \overline{p}_k$
   13) $k := k + 1$
14) until $\|\overline{\delta}_k - \overline{\delta}_{k-1}\|_2 < \epsilon$, /* where ϵ is a small predetermined constant *.

Björck and Elfving showed that the vectors $\overline{t} = C_\omega^{-T} \overline{p}$ and $\overline{q} = Y C_\omega^{-T} \overline{p}$ can be computed efficiently according to the following iterative procedure (where $d_j = (D)_{jj}$).

Algorithm 2
1) $\overline{u}^{(k)} := \overline{0}$
2) for $j := k$ downto 1 do
   3) $t_j := d_j^{-1/2} p_j - \omega d_j^{-1} Y(:, j)^T \overline{u}^{(j)}$
   4) $\overline{u}^{(j-1)} := \overline{u}^{(j)} + t_j Y(:, j)$
5) endfor
6) $\overline{q} := \overline{u}^{(0)}$.

Analogously, the following algorithm calculates $\overline{s} = C_\omega^{-1} Y^T \overline{r}$.

Algorithm 3
1) $\overline{u}^{(1)} := \overline{r}$
2) for $j := 1$ to κ do
   3) $s_j := d_j^{-1/2} Y(:, j)^T \overline{u}^{(j)}$
   4) $\overline{u}^{(j+1)} := \overline{u}^{(j)} - (\omega d_j^{-1/2} s_j) Y(:, j)$
5) endfor.

Therefore, the above vectors can be computed with only two sweeps through the columns of the matrix Y, so that the overall computational complexity of each CGPCNE cycle turns out to be in the order of $nz(Y)$, where $nz(Y)$ denotes the number of nonzero elements of Y. Analyzing the structure of Y from (11), it is readily seen that $nz(Y) \leq M\kappa$ and, in turn, κ is smaller than ν, the total number of connections in the network. Consequently, each CGPCNE cycle requires a number of operations which is roughly proportional to $M\nu$ and this is exactly the computational complexity of one epoch of the feedforward backpropagation algorithm [15] (observe that one CGPCNE cycle "sees" all the training patterns at once, and therefore it should be compared with one backpropagation epoch). However, unlike backpropagation, the CGPCNE algorithm contains dependencies which make it less suitable for parallel implementation. Fortunately, this does not appear to be a serious problem because, as seen in the experimental section, the number of CGPCNE cycles needed to find a solution is typically low.

C. Choosing the To-Be-Removed Units

One of the fundamental concerns in any pruning algorithm is how best to select the units to be eliminated. One possible strategy is to make use of some relevance or sensitivity measure to quantify the contribution that individual nodes make in solving the network task, and then to select the less relevant units as those to be removed (see, e.g., [32]). This choice, however, has nothing to do with the basic idea underlying the proposed method, as there is in general no guarantee that selecting the less relevant units will actually make system (10) consistent, or nearly so, which is clearly our ultimate goal.

Ideally, the most appropriate choice would be to eliminate among all the hidden units the one that, after solving the corresponding problem (14), results in the smallest *final* residual. This guarantees that the removal of that unit and the corresponding updating of the weights will have a minimal effect on the network's input–output behavior. This brute-force approach, however, involves solving as many least-squares problems as there are hidden units and would become impractical even for moderate problem dimensions. Fortunately, the specific method we use to solve problem (14) exhibits a nice property that naturally suggests a suboptimal selection criterion. Namely, recalling that CG-methods monotonically decrease the residuals $\rho_h(\overline{\delta}_k) = \|\overline{z} - Y_h\overline{\delta}_k\|_2^2$ at each time step, a reasonable approximation to the globally optimal approach mentioned above is to choose the unit h to be eliminated in such a way that the *initial* residual

$$\rho_h(\overline{\delta}_0) = \|\overline{z}_h - Y_h\overline{\delta}_0\|_2^2 \tag{21}$$

be the smallest among all the hidden units or, more formally,

$$h = \arg\min_{h \in V_H} \rho_h(\overline{\delta}_0). \tag{22}$$

Since the initial solution $\overline{\delta}_0$ is typically chosen to be the null vector, and recalling the meaning of $\overline{z}_h$ from (9) and (13), rule (22) becomes

$$h = \arg\min_{h \in V_H} \sum_{i \in \Gamma_h} w_{hi}^2 \|\overline{y}_h\|_2^2. \tag{23}$$

Along the same lines, in the case of weight elimination, the rule for detecting the to-be-removed connections would become

$$(h, i) = \arg\min_{(h,i) \in E} w_{hi}^2 \|\overline{y}_h\|_2^2. \tag{24}$$

It should be clear that the proposed criterion is by no means intended to be the *globally* optimal choice because there is no guarantee that starting from the smallest initial residual will result in the smallest final residual. Moreover, these selection

rules suffer from the same disadvantage as the sensitivity methods [21]. Namely, since the decision as to which unit/weight to remove is based mainly on the output of individual nodes, we could fail to identify possible correlations among units. To avoid this problem, combined selection criteria that take into account some correlation measure could well be employed, without altering the nature of the proposed pruning algorithm. Despite these caveats, in the present work we prefer this criterion not only because of the encouraging results it yielded in practice but also for its operational simplicity.

As a final remark, we note that the preceding selection rules have an interesting interpretation which comes from rather different considerations. In fact, as suggested in [45], the quantity $(w_{hi}y_h^{(\mu)})^2$ can be regarded as a measure of the synaptic activity of the connection between units h and i upon presentation of pattern μ. By averaging across all training patterns we obtain $1/M \ \Sigma_\mu (w_{hi}y_h^{(\mu)})^2 = w_{hi}^2 ||\overline{y}_h||_2^2/M$. Accordingly, the total synaptic activity of unit h can be defined by summing the above quantities across all the units in h's projective field and this is proportional to $\Sigma_{i\in P_h} w_{hi}^2 ||\overline{y}_h||_2^2$ (a nearly identical measure was independently proposed in [46], in an attempt to quantify the "goodness" of individual hidden units). Therefore, rules (23) and (24) can also be interpreted as criteria that select, respectively, the units and connections having the smallest synaptic activity.

D. Definition of the Pruning Algorithm

We now define precisely the pruning algorithm. Starting with an initial trained network $N^{(0)} = (V^{(0)}, E^{(0)}, w^{(0)})$, the algorithm iteratively produces a sequence of smaller and smaller networks $\{N^{(k)}\}$ by first identifying the unit h to be removed, and then solving the corresponding system (10) to properly adjust the remaining weights. The process is iterated until the performance of the reduced networks falls below the designer's requirements. More explicitly, the algorithm can be written as follows.

Algorithm 4: Proposed Pruning Algorithm

1) $k := 0$
2) repeat
   3) identify excess unit $h \in V_H^{(k)}$ in network $N^{(k)}$ according to rule (23)
   4) apply the *CGPCNE* algorithm to find a $\overline{\delta}$ that solves problem (14)
   5) construct $N^{(k+1)} = (V^{(k+1)}, E^{(k+1)}, w^{(k+1)})$ as follows:
      $V^{(k+1)} := V^{(k)} - \{h\}$
      $E^{(k+1)} := E^{(k)} - (\{h\} \times P_h^{(k)} \cup R_h^{(k)} \times \{h\})$
      $$w_{ji}^{(k+1)} := \begin{cases} w_{ji}^{(k)}, & \text{if } i \notin P_h \\ w_{ji}^{(k)} + \delta_{ji}, & \text{if } i \in P_h \end{cases}$$
   6) $k := k + 1$
7) until the "performance" of $N^{(k)}$ deteriorates excessively.

Note that the iterative nature of the above procedure allows the network designer to define appropriate performance measures depending on his own requirements (e.g., if separate training and validation sets are available, the network's performance can be measured as the error rate over validation data, in an attempt to improve generalization). Moreover, after pruning, the final network $N^{(k)}$ need not be retrained because the updating of the weights is embedded in the pruning algorithm itself.

Finally, we point out that the overall computational cost of each iteration of our pruning algorithm depends mainly on Step 4) which, as seen in Section III-B, requires a number of operations per step roughly proportional to $M\nu$, ν being the total number of connections in the network. Also, the number of cycles performed by the *CGPCNE* procedure, like any CG-based least-squares method [42], [43], turns out to be very low and typically far less than either M or ν, so that the overall computational complexity of each pruning step can be approximately $O(M\nu)$. It may be interesting to compare this computational cost with that of the OBS procedure [31] which, like our algorithm, does not demand further retraining sessions after pruning. Whereas OBS takes $O(M\nu^2)$ time to remove one single weight and update the remaining ones, we are able to remove a processing unit with all its incoming and outgoig connections and adjust the remaining weights in $O(M\nu)$ time (a major difference between the two procedures is that, unlike our method, OBS updates all the weights in the network). Furthermore, if our procedure were employed to remove a single connection, say $(h, i) \in E$, its computational complexity would scale to $O(Mr_i)$, where r_i is the number of connections incoming into unit i; since $r_i \ll \nu^2$, our pruning algorithm is far less computationally expensive than OBS.

IV. Relations with Sietsma and Dow's Pruning Approach

In this section we show how the linear system formulation developed in this paper can lead to an alternative *manual* approach to network pruning, of which Sietsma and Dow's (S&D) algorithm [28], [29] is the best-known representative (see also [35], [47]). Specifically, rather than trying to directly solve system (10), one can study specific consistency conditions for it, thereby deriving rules for locating and removing redundant units. This is basically the approach made by Sietsma and Dow, although they were apparently unaware of the linear system formulation underlying their idea. Starting from simple heuristic considerations, they developed the following simple rules for pruning layered networks.

1) If the output of hidden unit h is constant over the whole training set, i.e., $y_h^{(\mu)} = a, \mu = 1 \cdots M$, then remove h and, for each unit i in the succeeding layer (which is h's projective field), add to its bias value w_{0i} the quantity $w_{hi}a$.
2) If unit h gives the same output as a second unit k across the entire training set, i.e., $y_h^{(\mu)} = y_k^{(\mu)}$ for all $\mu = 1 \cdots M$, then remove h and adjust k's outgoing weights as $w_{ki} \leftarrow w_{ki} + w_{hi}$.
3) If unit h gives opposite output values as a second unit k, i.e., $y_h^{(\mu)} = 1 - y_k^{(\mu)}, \mu = 1 \cdots M$, then remove h, adjust k's outgoing weights as $w_{ki} \leftarrow w_{ki} - w_{hi}$, and, for each i in the next layer, add to its bias value w_{0i} the quantity w_{hi}.

We now demonstrate how S&D's rules can easily be derived within our framework. In particular, if any of the previous conditions are fulfilled then all the subsystems $Y_{i,h}\overline{\delta}_i = \overline{z}_{i,h}, i \in P_h$, defined in (8) are consistent, and so therefore will be system (10). Moreover, the way in which rules 1)–3) update the remaining weights after pruning corresponds to a specific solution of that system. In fact, all the matrices $Y_{i,h}$'s defined in (7) will always contain a column consisting of all "1's", which corresponds to the output of the bias unit. When the output of unit h is constant (rule 1), the vector $\overline{z}_{i,h}$ (for all $i \in P_h$) becomes $w_{hi}(a, a, \cdots, a)^T$; by simply setting $\delta_{0i} = w_{hi}a$ and all the other components of $\overline{\delta}_i$ at zero, $\overline{z}_{i,h}$ can therefore be written as a linear combination of the columns of $Y_{i,h}$, which means that $Y_{i,h}\overline{\delta}_i = \overline{z}_{i,h}$ is consistent, for any $i \in P_h$. Similar arguments apply to rule 2) where the vector $\overline{z}_{i,h}$ now becomes proportional to the column in $Y_{i,h}$ corresponding to unit k; in this case, too, system (8) is consistent and a solution can be obtained by setting $\delta_{ki} = w_{hi}$ and all other components at zero. Finally, rule 3) can be derived analogously by observing that the vectors $\overline{z}_{i,h}$'s can be obtained as a linear combination of the columns of the $Y_{i,h}$'s corresponding to the "0" (bias) and k units.

It is clear that the major drawback of S&D's approach is its simplicity: in real-world applications, in fact, hidden units are unlikely to be exactly correlated and the algorithm may thus fail to detect redundant elements. Additionally, although rules 1)–3) are particularly simple to implement in networks of threshold units, in the practical case of nonlinear activation functions, they must be translated into more precise criteria. This involves determining a number of problem-dependent threshold parameters whose choice is problematic: small threshold values, in fact, typically lead to removing very few redundant units, while large values result in too many excised nodes, thereby seriously worsening the performance of the network. Due to these approximations, a further slow retraining stage is generally required after pruning and we found experimentally that sometimes the retraining process may even fail to converge. The approach to network pruning pursued in this paper offers a general and systematic method for reducing the size of a trained network which contrasts sharply with the heuristic, *ad hoc* procedure developed by Sietsma and Dow: ours, in fact, is not based on any simplifying assumption, does not depend on any working parameter, and does not demand additional retraining sessions.

V. Experimental Results

To test the effectiveness of our pruning algorithm, several simulations were carried out over different problems. In all the experiments presented, fully connected neural networks with one hidden layer were considered. For each test problem, ten independent networks (denoted with A to J), with weights randomly generated from a standard Gaussian distribution, were trained by the backpropagation algorithm [2], [6] ($\eta = 1.0$ and $\alpha = 0.7$). Each trained network was reduced by applying our algorithm, where the CGPCNE procedure was iterated, with relaxation parameter ω fixed at 1, until the distance between two successive solutions became smaller than 10^{-8}.

For comparison, the S&D pruning algorithm was implemented with a fine-tuned choice of parameters. Specifically, a hidden unit was regarded as having constant output (rule 1) when the variance of its output vector is lower than a threshold ϵ_1; two hidden units h and k had the same output values (rule 2) when the normalized distance $\|\overline{y}_h - \overline{y}_k\|^2/M$ is less than a threshold ϵ_2, and the same condition was applied to $\overline{y}_h$ and $\sim\overline{y}_k$ for locating anti-parallel units (rule 3). The choice of $\epsilon_1 = 10^{-2}$ and $\epsilon_2 = 10^{-1}$ appeared to be near-optimal for the considered problems. Also, following S&D's practice [29], before detecting the units to be removed, output values less than 0.35 and greater 0.65 were approximated to zero and one, respectively. Units were removed in the following order, shown to be the best: first constant-output units, then parallel units, and finally antiparallel units.

To evaluate the behavior of both methods, three different measures were adopted: 1) the number of hidden nodes in the reduced networks; 2) the recognition rate, measured as the proportion of examples for which all network output values differed from the corresponding target by less than 0.5; and 3) the usual mean-squared error (MSE).

A. Parity and Symmetry Problems

To assess the performance of the proposed method, the well-known parity and symmetry tasks [2] were chosen, as the near-optimal number of hidden units required to achieve their solution is known. In the case of the parity task, this is equal to the number of input units (but, see [48] for smaller solution networks), while the symmetry problem can be solved with only two hidden nodes, whatever the length of the input string. In both series of experiments, ten randomly initialized 4-10-1 networks were trained until, for each pattern in the training set, all the network output values differed from the corresponding targets by less than 0.05.

Then, to evaluate the behavior of our pruning procedure under "stressed" conditions, each trained network was pruned (regardless of its performance) until the hidden layer contained exactly one unit. The average results are shown in Fig. 3 which plots, for both problems, the evolution of the recognition rate and the MSE during the pruning process. As can be seen, our algorithm exhibits very steady behavior in the first stages of pruning, whereas, as may be expected, it performs poorly when near-optimal size is approached. Also, as shown in Fig. 4, the number of CGPCNE iterations required to find a solution for each pruning step is extremely small and decreases linearly with the number of hidden units. This is in agreement with theoretical and experimental results reported in [42] and [43].

However, for the proposed procedure to produce useful small networks, a stopping criterion should be specified, which takes into account the performance of the reduced networks. For the two tasks at hand, the following stopping rule was adopted, aiming to obtain networks with the same training-set performance as the original ones. At each pruning step, the performance of the reduced network, measured as the recognition rate over the training set, was compared with that of the *original* network. If a deterioration of 1% or more was observed, then pruning was stopped and the previous reduced

TABLE I
NUMERICAL RESULTS FOR THE PARITY PROBLEM

	no. hidden units		recognition rate (%)		MSE	
net	proposed method	S&D	proposed method	S&D	proposed method	S&D
A	5	3	100	87.5	0.00	0.12
B	4	5	100	93.7	0.02	0.06
C	5	4	100	93.7	0.00	0.06
D	5	6	100	100	0.00	0.00
E	5	6	100	100	0.01	0.00
F	5	7	100	100	0.00	0.00
G	5	5	100	100	0.00	0.00
H	5	4	100	93.7	0.00	0.06
I	5	6	100	100	0.00	0.00
J	5	5	100	88.0	0.00	0.12
average	4.9	5.1	100	95.66	0.003	0.044

TABLE II
NUMERICAL RESULTS FOR THE SYMMETRY PROBLEM

	no. hidden units		recognition rate (%)		MSE	
net	proposed method	S&D	proposed method	S&D	proposed method	S&D
A	3	6	100	100	0.01	0.00
B	3	9	100	100	0.01	0.00
C	4	6	100	100	0.00	0.01
D	4	8	100	100	0.00	0.00
E	2	7	100	100	0.02	0.00
F	3	4	100	100	0.00	0.00
G	4	6	100	87.5	0.02	0.10
H	4	6	100	93.7	0.01	0.02
I	4	7	100	100	0.02	0.01
J	5	7	100	93.7	0.01	0.04
average	3.6	6.6	100	97.49	0.008	0.018

network was retained as the final one. The results obtained by our pruning procedure with the above stopping criterion, and the S&D pruning algorithm with no retraining phase, are summarized in Table I for parity and Table II for symmetry. As can be observed, our procedure appears extremely robust since all trials resulted in a near-optimal solution network with perfect 100% recognition rate, irrespective of the initial conditions. The S&D method, by contrast, exhibits much more unstable behavior with poorer recognition performance of the pruned networks, that were therefore later retrained. We found that in the case of the parity problem five out of ten trials failed to converge within 3000 epochs. The remaining five required a median number of 24 (ranging from zero to 119). In the symmetry case, instead, all the retraining trials converged to a solution in a median number of 48 epochs.

Finally, to compare the time required to train and prune a large solution network with that for directly training a small network, ten independent 4-5-1 and 4-4-1 networks were trained for parity and symmetry respectively; the size of these networks corresponds to the "median" sizes found by our procedure. Tables III and IV show the median number of epochs required to train such small networks, for parity and symmetry, respectively, and the median number of backpropagation as well as CGPCNE cycles (which, as seen in Section III-B, have comparable computational complexity) needed to train the original 4-10-1 networks and then reduce them to the median size. As can be observed, in any case the overall time required to train a large network and then prune it to a small size compares very favorably with that of simply training a small network. In addition, for initially small networks, convergence is not always guaranteed; in fact, seven out of ten parity trials did not converge within 3000 epochs.

B. A Simulated Pattern Recognition Task

A classification task nearer to real-world problems, suggested in [49] and used as a test problem by many authors

TABLE III
MEDIAN NUMBER OF BACKPROP/CGPCNE CYCLES REQUIRED BOTH TO TRAIN A SMALL NETWORK AND TO TRAIN-AND-PRUNE A LARGE NETWORK, FOR THE PARITY PROBLEM

architecture	failure (%)	BP epochs	pruning cycles	total
4-5-1	70	1228	-	1228
4-10-1	0	650	45	695

TABLE IV
MEDIAN NUMBER OF BACKPROP/CGPCNE CYCLES REQUIRED BOTH TO TRAIN A SMALL NETWORK AND TO TRAIN-AND-PRUNE A LARGE NETWORK, FOR THE SYMMETRY PROBLEM

architecture	failure (%)	BP epochs	pruning cycles	total
4-4-1	0	299	-	299
4-10-1	0	194	51	245

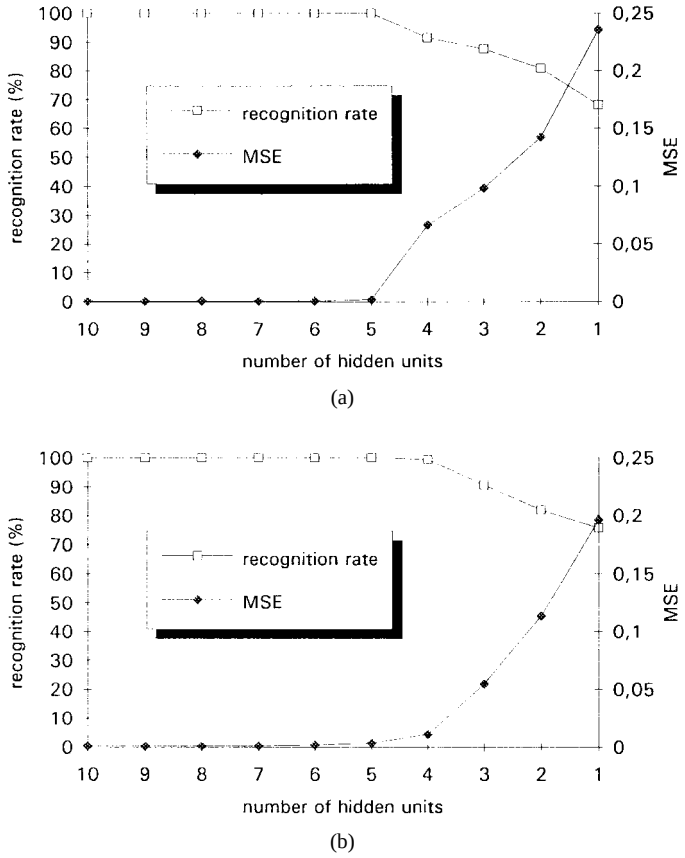

Fig. 3. Behavior of the recognition rate and the MSE during the pruning process. (a) Parity problem; (b) symmetry problem.

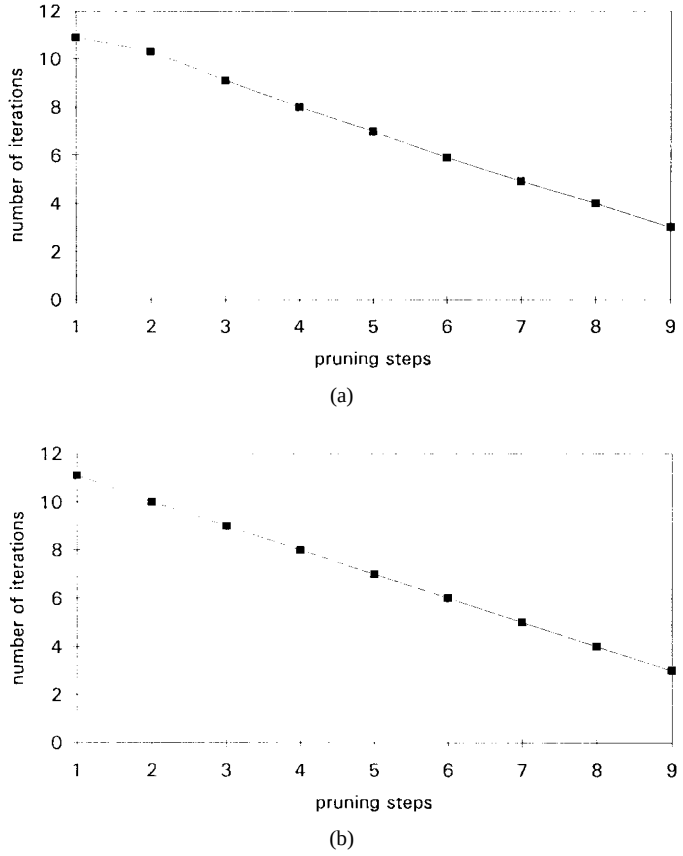

Fig. 4. Number of CGPCNE cycles at each pruning step. (a) Parity problem; (b) symmetry problem.

[50]–[52], was chosen to test the generalization performance of the networks reduced by our procedure. This is a two-class simulated pattern recognition problem with a two-dimensional continuous feature space, and known probability distributions. As shown in Fig. 5, class 1 is a single Gaussian distribution, while class 2 is an equal likelihood mixture of two Gaussians. In the experiments reported here the values $\mu_x = 2.30\sigma, \mu_y = 2.106\sigma$, and $\sigma = 0.2$ were used, which correspond to "mixture 2" data used by others [50], [51].

A training set of 200 samples, and two separate 1000-sample data sets, one for validation and the other for testing, were randomly generated from the Gaussian distributions for both classes with equal probability. Due to overlapping between classes, perfect learning cannot be achieved. Accordingly, ten randomly initialized 2-10-1 networks were trained for 1000 epochs. After training, the pruning procedure was applied. Fig. 6 shows the results of running the algorithm until all but one unit remained in the hidden layer; the performance over

TABLE V
NUMERICAL RESULTS ON TRAINING AND VALIDATION SETS FOR THE PATTERN RECOGNITION PROBLEM

			on training set						on validation set					
	no. hidden units		recognition rate (%)			MSE			recognition rate (%)			MSE		
	proposed			proposed			proposed			proposed			proposed	
net	method	S&D	Original	method	S&D	Original	method	S&D	Original	method	S&D	Original	method	S&D
A	4	6	96.5	93.0	97.0	0.03	0.06	0.03	92.6	94.0	92.8	0.057	0.052	0.057
B	1	6	97.5	93.5	96.0	0.02	0.06	0.04	92.0	93.7	93.7	0.065	0.064	0.059
C	4	4	96.0	95.0	95.0	0.03	0.03	0.03	94.3	94.2	94.9	0.042	0.041	0.040
D	3	5	97.0	94.0	96.5	0.03	0.04	0.04	92.4	94.4	93.3	0.056	0.044	0.053
E	1	4	97.0	94.5	94.0	0.03	0.05	0.04	91.8	93.1	93.1	0.061	0.067	0.049
F	4	6	96.5	94.0	96.5	0.03	0.05	0.03	92.7	94.4	92.7	0.055	0.050	0.056
G	5	5	97.0	97.0	97.0	0.03	0.03	0.03	94.7	94.6	94.2	0.045	0.044	0.047
H	4	6	96.5	94.5	95.5	0.03	0.06	0.04	93.1	94.4	92.6	0.060	0.059	0.065
I	2	5	95.5	92.5	95.5	0.03	0.06	0.03	93.0	93.6	94.9	0.047	0.063	0.041
J	1	4	97.0	94.5	93.5	0.02	0.05	0.05	91.8	93.2	93.2	0.061	0.067	0.049
average	2.9	5.1	96.65	94.25	95.65	0.027	0.049	0.035	92.84	93.96	93.54	0.055	0.055	0.052

both the training and the validation set, averaged over the ten trials, is reported. It can be observed how the algorithm exhibits a very stable behavior, both in terms of recognition rate and MSE and even with just one hidden unit performance deterioration is negligible. In Fig. 7, the (average) number of CGPCNE cycles is plotted as a function of the size of the hidden layer. As in both the previous logical tasks, linear behavior can be observed.

In practical real-world applications, a better generalization performance is more important than optimal behavior over the training data. As suggested by many authors [53], [54], one way to improve generalization and thereby avoid overfitting consists of stopping the learning stage as soon as a deterioration of the network performance over a separate validation set is observed. Recently, this approach has proved to be effective when combined with some pruning process [55]. Accordingly, the following rule was employed to stop pruning. Whenever a hidden unit was removed, the recognition rate of the new smaller network was evaluated over the validation set, and then compared with the performance of the *previous* network. If a deterioration of 1% or more was observed, then pruning was stopped and the previous reduced network was taken as the final one.

Table V summarizes the results obtained by applying our pruning with the above stopping condition as well as the S&D results. It is clear that our procedure yields a better performance than S&D procedure, in terms of network size reduction. Comparable results, instead, were obtained in terms of recognition rate over both the training and validation set. The original networks performed better than the pruned ones over the training set, due to the particular rule employed to stop pruning, which does not take into account the performance of the reduced networks over the training data as pruning proceeds. Besides, it is well known that good generalization results are typically achieved when performance over the learning set is nonoptimal [53]–[55]. On the other hand, when the performance on the training set is important, alternative criteria can be employed to stop pruning, in order to avoid loss of accuracy on training data.

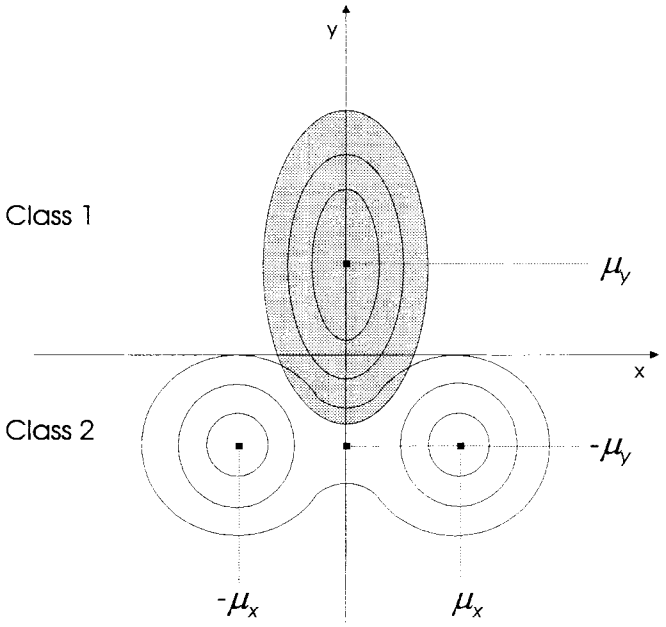

Fig. 5. Gaussian densities used in the pattern recognition example.

Next, we evaluated the generalization ability of the reduced networks with respect to the original ones. The statistics of the performance measures, number of hidden units and recognition rate computed over the test set, are listed in Table VI. As can be seen, the networks reduced by our procedure generalize slightly better than the original ones as well as the ones reduced by S&D.

Finally, we point out that the generalization results of the networks reduced by our pruning algorithm are not only superior to those found by Holt [51] under experimental conditions similar to ours (i.e., same classification problem, equal training and test set sizes, network architecture, and learning stopping criterion), but are comparable with the best results he obtained using an alternative cost function aiming to improve the generalization performance. Moreover, the median size of our reduced networks (2-3-1) coincides with the minimal network size required for achieving the best

TABLE VI
GENERALIZATION RESULTS FOR THE PATTERN RECOGNITION PROBLEM

	no. hidden units		recognition rate (%)	
	average	standard deviation	average	standard deviation
original	10	0	92.61	0.74
proposed method	2.9	1.52	93.36	0.691
S&D	5.1	0.88	93.04	0.538

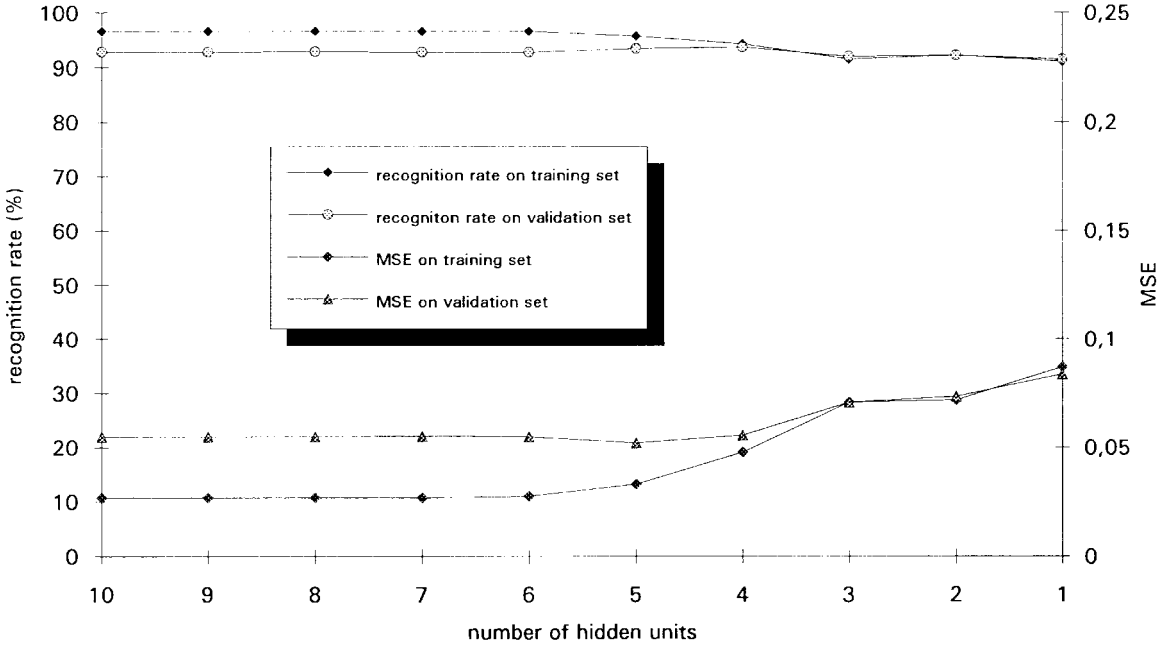

Fig. 6. Behavior of the recognition rate and the MSE during the pruning process for the pattern recognition problem. Both the behavior over the training and validation sets are displayed.

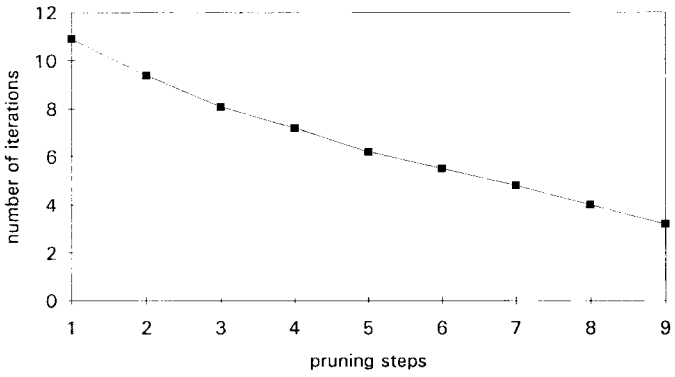

Fig. 7. Number of CGPCNE cycles at each pruning step for the pattern recognition problem.

generalization results for this problem, as claimed in [51] and [49]. This demonstrates once again that the proposed

pruning procedure is able to find the most appropriate network architecture to solve a given problem.

VI. CONCLUSIONS

This paper describes a method for reducing the size of trained feedforward neural networks, in which the key idea consists of iteratively removing hidden units and then adjusting the remaining weights in such a way as to preserve the overall network behavior. This leads to formulating the pruning problem in terms of defining a system of linear equations that we solve with a very efficient conjugate gradient least-squares procedure. A simple and effective rule is also derived which proves to work well in practice, and turns out to be closely related with a selection rule derived elsewhere in a different context. However, alternative selection rules can be adopted as well, without altering the method as a whole. A nice feature is that no parameter needs to be set, and this contrasts with most existing pruning procedures. Furthermore, the iterative nature of the algorithm permits the network designer to monitor the behavior of the reduced networks at each stage of the pruning process, so as to define his own stopping criterion. The experimental results obtained prove that the method does a very good job of reducing the network size while preserving excellent performance, without requiring the additional cost of a retraining phase. In addition, the time required to train a small network is typically much longer than that needed to train a large network and then reduce its size by means of our pruning procedure. The results of the experimental comparison with a well-known pruning procedure show that the proposed

one yields better results in terms of network size, performance, and robustness.

Although we have focused primarily on the problem of removing units, our approach is naturally applicable to the elimination of single connections as well. Our choice was motivated by the observation that computational nodes represent the "bottleneck" through which information in a neural network is conveyed, and are therefore more important than individual connections [26]. Moreover, node pruning algorithms are more efficient, although less accurate, than weight elimination methods. However a "coarse-to-fine" approach could be pursued by removing first units and then, when no unit can be further excised, single connections. Finally, we emphasize that the proposed pruning approach is far more general than we have presented here and can be applied to networks of arbitrary topology.

REFERENCES

[1] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Mag.*, vol. 10, pp. 8–39, 1993.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing—Vol. 1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[3] S. Y. Kung and J. N. Hwang, "An algebraic projection analysis for optimal hidden units size and learning rates in backpropagation learning," in *Proc. IEEE Int. Conf. Neural Networks*, San Diego, CA, vol. 1, 1988, pp. 363–370.

[4] D. C. Plaut and G. E. Hinton, "Learning sets of filters using backpropagation," *Comput. Speech Language*, vol. 2, pp. 35–61, 1987.

[5] D. J. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-36, pp. 1162–1168, 1988.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by backpropagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[7] X.-H. Yu, "Can backpropagation error surface not have local minima," *IEEE Trans. Neural Networks*, vol. 3, pp. 1019–1021, 1992.

[8] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE Acoust., Speech, Signal Processing Mag.*, vol. 4, pp. 4–22, 1987.

[9] M. D. Emmerson and R. I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Trans. Neural Networks*, vol. 4, pp. 788–793, 1993.

[10] E. B. Baum and D. Haussler, "What size net gives valid generalization?" *Neural Computa.*, vol. 1, pp. 151–160, 1989.

[11] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield, "Large automatic learning, rule extraction, and generalization," *Complex Syst.*, vol. 1, pp. 877–922, 1987.

[12] Y. Le Cun, "Generalization and network design strategies," in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman-Soulie, and L. Steels, Eds. Amsterdam: Elsevier, 1989, pp. 143–155.

[13] Y. Chauvin, "Generalization performance of overtrained backpropagation networks," in *Neural Networks—Proc. EURASIP Wkshp. 1990*, L. B. Almeida and C. J. Wellekens, Eds. Berlin: Springer-Verlag, 1990, pp. 46–55.

[14] G. G. Towell, M. K. Craven, and J. W. Shavlik, "Constructive induction in knowledge-based neural networks," in *Proc. 8th Int. Wkshp. Machine Learning*, L. A. Birnbaum and G. C. Collins, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 213–217.

[15] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation.* Redwood City, CA: Addison-Wesley, 1991.

[16] Y. H. Hu, "Configuration of feedforward multilayer perceptron network," unpublished manuscript, 1993.

[17] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

[18] S. I Gallant, "Optimal linear discriminants," in *Proc. 8th Int. Conf. Pattern Recognition*, Paris, France, 1986, pp. 849–852.

[19] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Sci.*, vol. 1, no. 4, pp. 365–375, 1989.

[20] M. Mézard and J.-P. Nadal, "Learning in feedforward layered networks: The Tiling algorithm," *J. Phys. A*, vol. 22, pp. 2191–2204, 1989.

[21] R. Reed, "Pruning algorithms—A review," *IEEE Trans. Neural Networks*, vol. 4, pp. 740–747, 1993.

[22] S. C. Huang and Y. F. Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 47–55, 1991.

[23] G. E. Hinton, "Connectionist learning procedures," *Artificial Intell.*, vol. 40, no. 1, pp. 143–150, 1989.

[24] Y. Chauvin, "A backpropagation algorithm with optimal use of hidden units," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 519–526.

[25] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination with application to forecasting," in *Advances in Neural Information Processing Systems 3*, R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, pp. 875–882.

[26] J. K. Kruschke, "Creating local and distributed bottlenecks in hidden layers of backpropagation networks," in *Proc. 1988 Connectionist Models Summer School*, D. S. Touretzky, G. E. Hinton, and T. J. Sejnowski, Eds. San Mateo, CA: Morgan Kaufmann, 1988, pp. 120–126.

[27] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984.

[28] J. Sietsma and R. J. F. Dow, "Neural net pruning: Why and how," in *Proc. IEEE Int. Conf. Neural Networks*, San Diego, CA, vol. 1, 1988, pp. 325–333.

[29] ———, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, pp. 67–79, 1991.

[30] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretsky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 598–605.

[31] B. Hassibi and D. G. Stork, "Second-order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 164–171.

[32] M. C. Mozer and P. Smolensky, "Using relevance to reduce network size automatically," *Connection Sci.*, vol. 1, no. 1, pp. 3–16, 1989.

[33] E. D. Karnin, "A simple procedure for pruning backpropagation trained neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 239–242, 1990.

[34] A. Burkitt, "Optimization of the architecture of feedforward neural networks with hidden layers by unit elimination," *Complex Syst.*, vol. 5, pp. 371–380, 1991.

[35] F. L. Chung and T. Lee, "A node pruning algorithm for backpropagation networks," *Int. J. Neural Syst.*, vol. 3, no. 3, pp. 301–314, 1992.

[36] G. Castellano, A. M. Fanelli, and M. Pelillo, "Pruning in recurrent neural networks," in *Proc. Int. Conf. Artificial Neural Networks (ICANN'94)*, Sorrento, Italy, 1994, pp. 451–454.

[37] S. Y. Kung and Y. H. Hu, "A Frobenius approximation reduction method (FARM) for determining optimal number of hidden units," in *Proc. Int. J. Conf. Neural Networks*, Seattle, WA, vol. 2, 1991, pp. 163–168.

[38] Q. Xue, Y. H. Hu, and W. J. Tompkins, "Analyzes of the hidden units of backpropagation model by singular value decomposition (SVD)," in *Proc. Int. J. Conf. Neural Networks*, Washington, D.C., vol. 1, 1990, pp. 739–742.

[39] Y. H. Hu, Q. Xue, and W. J. Tompkins, "Structural simplification of a feedforward multilayer perceptron artificial neural network," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Toronto, Canada, 1991, pp. 1061–1064.

[40] M. A. Sartori and P. J. Antsaklis, "A simple method to derive bounds on the size and to train multilayer neural networks," *IEEE Trans. Neural Networks*, vol. 2, pp. 467–471, 1991.

[41] G. H. Golub and C. F. Van Loan, *Matrix Computations.* Baltimore, MD: Johns Hopkins Univ. Press, 1989.

[42] A. Björck, "Methods for sparse linear least-squares problems," in *Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, Eds. New York: Academic, 1976, pp. 177–199.

[43] I. S. Duff, "A survey of sparse matrix research," *Proc. IEEE*, vol. 65, no. 4, pp. 500–535, 1977.

[44] A. Björck and T. Elfving, "Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations," *BIT*, vol. 19, pp. 145–163, 1979.

[45] G. Orlandi, F. Piazza, A. Uncini, E. Luminari, and A. Ascone, "Dynamic pruning in artificial neural networks," in *Parallel Architectures and Neural Networks*. E. R. Caianello, Ed., Singapore: World Scientific, 1991, pp. 199–208.

[46] K. Murase, Y. Matsunaga, and Y. Nakade, "A backpropagation algorithm which automatically determines the number of association units," in *Proc. Int. J. Conf. Neural Networks*, Singapore, pp. 783–788.
[47] T.-C. Lee, *Structure Level Adaptation for Artificial Neural Networks.* Boston, MA: Kluwer, 1991.
[48] A. Sperduti and A. Starita, "Speed up learning and network optimization with extended backpropagation," *Neural Networks*, vol. 6, pp. 365–383, 1993.
[49] L. Niles, H. Silverman, J. Tajchman, and M. Bush, "How limited training data can allow a neural network to outperform an optimal statistical classifier," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, Glasgow, Scotland, vol. 1, 1989, pp. 17–20.
[50] P. Burrascano, "Learning vector quantization for the probabilistic neural network," *IEEE Trans. Neural Networks*, vol. 2, pp. 458–461, 1991.
[51] M. J. J. Holt, "Comparison of generalization in multilayer perceptrons with the log-likelihood and least-squares cost functions," in *Proc. 11th Int. Conf. Pattern Recognition*, The Hague, The Netherlands, vol. 2, 1992, pp. 17–20.
[52] W. T. Lee and M. F. Tenorio, "On an asymptotically optimal adaptive classifier design criterion," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 15, pp. 312–318, 1993.
[53] N. Morgan and H. Bourlard, "Generalization and parameter estimation in feedforward nets: Some experiments," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 630–637.
[54] A. S. Weigend, B. A. Huberman, and D. E. Rumelhart, "Predicting the future: A connectionist approach," *Int. J. Neural Syst.*, vol. 1, no. 3, pp. 193–209, 1990.
[55] W. Zinnoff, F. Hergert, and H. G. Zimmermann, "Improving model selection by nonconvergent methods," *Neural Networks*, vol. 6, pp. 771–783, 1993.

Giovanna Castellano was born in Bari, Italy, on June 27, 1969. She received the "Laurea" degree with honors in computer science from the University of Bari, Italy in 1993, with a thesis on structural optimization of artificial neural networks.

From 1993 to 1995 she worked at the Computer Science Department of the University of Bari for research in the field of artificial neural networks. Since March 1995, she has been a Researcher at the Institute for Signal and Image Processing with a scholarship under a grant from the "Consiglio Nazionale delle Ricerche." Her current research interests include artificial neural networks, fuzzy systems, neuro-fuzzy modeling, intelligent control using fuzzy logic, robotics, and autonomous systems.

Anna Maria Fanelli (M'89) was born in Bari, Italy, on June 29, 1949. She received the "Laurea" degree in physics from the University of Bari, Italy, in 1974.

From 1975 to 1979, she was full-time Researcher at the Physics Department of the University of Bari, Italy, where she became Assistant Professor in 1980. In 1985 she joined the Department of Computer Science at the University of Bari, Italy, as Professor of Computer Science. Currently, she is responsible for the courses "computer systems architectures" and "neural networks" at the degree course in computer science. Her research activity has involved issues related to pattern recognition, image processing, and computer vision. Her work in these areas has been published in several journals and conference proceedings. Her current research interests include artificial neural networks, genetic algorithms, fuzzy systems, neuro-fuzzy modeling, and hybrid systems.

Prof. Fanelli is a Member of the System, Man, and Cybernetics Soceity and the International Neural Network Society. She is also on the editorial board of the journal *Neural Processing Letters.*

Marcello Pelillo (M'92) was born in Taranto, Italy, on June 1, 1966. He received the "Laurea" degree with honors in Computer Science from the University of Bari, Italy, in 1989.

From 1988 to 1989 he was at the IBM Scientific Center in Rome, where he was involved in studies on natural language and speech processing. In 1991 he joined the Department of Computer Science at the University of Bari, Italy, as Assistant Professor. Since 1995 he has been with the Department of Applied Mathematics and Computer Science at the University of Venice "Ca' Foscari." In 1995 and 1996 he was a Visiting Professor at the Department of Computer Science of the University of York, U.K., and in 1995 he visited the Center for Intelligent Machines of the Department of Electrical Engineering at the McGill University, Montréal, Canada. His research interests include computer vision, neural networks, and pattern recognition, where he has published over 40 papers in refereed journals and conference proceedings.

Prof. Pelillo is currently the Program Cochair of the *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition* that will be held in Venice, Italy, in May 1997. He is a Member of the IEEE Computer Society and the Pattern Recognition Society.