Neural Comput & Applic (2013) 22:71–84
DOI 10.1007/s00521-011-0748-6

CONT. DEV. OF NEURAL COMPT. & APPLN.

A novel weight pruning method for MLP classifiers based on the MAXCORE principle

Cláudio M. S. Medeiros · Guilherme A. Barreto

Received: 16 February 2011 / Accepted: 17 September 2011 / Published online: 14 October 2011

Abstract We introduce a novel weight pruning methodology for MLP classifiers that can be used for model and/or feature selection purposes. The main concept underlying the proposed method is the *MAXCORE principle*, which is based on the observation that relevant synaptic weights tend to generate higher correlations between error signals associated with the neurons of a given layer and the error signals propagated back to the previous layer. Nonrelevant (i.e. prunable) weights tend to generate smaller correlations. Using the MAXCORE as a guiding principle, we perform a cross-correlation analysis of the error signals at successive layers. Weights for which the cross-correlations are smaller than a user-defined error tolerance are gradually discarded. Computer simulations using synthetic and real-world data sets show that the proposed method performs consistently better than standard pruning techniques, with much lower computational costs.

Keywords MLP classifier · Backpropagation algorithm · Weight pruning · Feature selection

C. M. S. Medeiros
Department of Industry, Federal Institute of Ceará,
Av. Treze de Maio, 2081—Campus of Benfica, Fortaleza,
Ceará CEP 60040-531, Brazil
e-mail: claudiosa@ifce.edu.br

G. A. Barreto (✉)
Department of Teleinformatics Engineering,
Federal University of Ceará, Av. Mister Hull,
S/N—Campus of Pici, Center of Technology,
CP 6005, Fortaleza, Ceará CEP 60455-970, Brazil
e-mail: guilherme@deti.ufc.br

1 Introduction

Even though it has passed two decades and a half since the rediscovery of the backpropagation algorithm in the mid-1980s, and despite all the available literature on the MLP network, a beginner soon becomes aware of the difficulties in finding a suitable architecture for real-world applications. In fact, this is a hard task also for an experienced practitioner. An architecture that is too small will not be able to learn from data properly, no matter what training algorithm is used for this purpose. An architecture with too many input units and hidden layer/neurons are prone to learn undesirable characteristics (e.g. noise) of the training data.

Hence, a crucial step in the design of a MLP is related with the *network model selection* problem [5]. This problem is still a research topic of interest [8, 10, 13, 20, 28, 30, 33] and can be roughly defined as the task of finding the smallest architecture that generalizes well, making good predictions for new data. Generalization can be assessed by changing the number of adjustable parameters (weights and biases) associated with input units and hidden/output neurons. Among the several ways to implement this in practice, we list the following four as possibly the most common approaches.

1.1 Exhaustive search plus early stopping

Early stopping of training is a method that aims to prevent overtraining due to oversized network, noisy training examples, or a small training set [7]. The performances of several networks having different number of features, hidden layers, and hidden neurons are evaluated during training on an independent validation set. Training of each network is stopped as soon as its generalization error begins to increase. The optimal architecture is the one providing the smallest generalization error.

1.2 Constructive algorithms

Network training starts with a small number of hidden neurons and add neurons during the training process, with the goal of arriving at an optimal network structure [1, 21]. This is the approach behind the Cascade-Correlation network [9, 12, 14, 17, 31].

1.3 Pruning algorithms

Network training can start with a relatively large number of hidden neurons and then, based on a measure of weight relevance, prune out the least significant connections, either by removing individual weights or by removing complete units [6, 11, 24]. This is the approach followed by the *Optimal Brain Surgeon* [15, 29] (OBS) and the *Weight Decay and Elimination* (WDE) [23, 32]. The OBS performs weight pruning based on the so-called *weight saliencies* (see the "Appendix"). The WDE algorithm originates from a regularization method that modifies the error function by the introduction of a term that penalizes large weights.

1.4 Evolutionary computation

Several authors have used genetic algorithms [2, 27, 34] and other evolutionary algorithms, such as particle swarm optimization [35], for determining the best network topology for a given problem, including the number of hidden layers, the number of neurons per hidden layer, and the number of relevant input features, connection weights, and biases.

The previous model selection methods require significant computational efforts. For example, even if we restrict the exhaustive search to a specific class of architectures (e.g. one-hidden-layered MLP), it is still a burdensome task. Evolutionary algorithms are slow by their very population-based nature and usually cannot be used for real-time purposes. The OBS algorithm demands the inversion of the Hessian matrix of the error function, which is a computationally intensive task [4]. The WDE algorithm requires the specification of a user-defined regularization parameter, which is generally set up by cross-validation.

In this paper, we introduce an efficient method for pruning unnecessary weights of a trained MLP classifier without the need of matrix inversions and any additional regularization parameter. The proposed method is based on the correlation analysis between the errors of the output neurons and the errors backpropagated to the hidden neurons. Repeated application of the method leads eventually to the complete elimination of all connections arriving to (or emanating from) a given neuron. Computer simulations using synthetic and real-world data sets are carried out to compare the proposed method with OBS and WDE algorithms on pattern classification and feature subset selection tasks.

The remainder of the paper is organized as follows. In Sect. 2, we briefly review the backpropagation algorithm. In Sect. 3, we introduce the MAXCORE principle as the guiding principle behind the proposal of the weight pruning method to be described in Sect. 4. Comprehensive computer simulations are then presented in Sect. 5 for model selection tasks and in Sect. 5 for feature selection tasks. The computational cost of the proposed weight pruning method is discussed in Sect. 7. We conclude the paper in Sect. 8 with a summary of the achievements and suggestions for further developments.

2 The backpropagation algorithm

In this section, we describe the bascis of the backpropagation algorithm for training a one-hidden-layered MLP. This learning algorithm requires two passes of computation: a forward pass and a backward pass. During the forward pass activations and outputs are computed on a neuron-by-neuron basis, while the weights remain unaltered. At iteration t, the activation of the ith hidden neuron, $i = 1, 2, \ldots, Q$, is given by

$$u_i^{(h)}(t) = \sum_{j=1}^{P} w_{ij}(t)x_j(t) - \theta_i(t) = \sum_{j=0}^{P} w_{ij}(t)x_j(t), \tag{1}$$

where w_{ij} is the synaptic weight connecting the jth input unit to the ith hidden neuron, $\theta_i(t)$ is the bias of the ith hidden neuron, $Q(2 \leq Q < \infty)$ is the number of hidden neurons, and P is the dimension of the input vector (excluding the threshold). For simplifying notation, we set $x_0(t) = -1$ and $w_{i0} = \theta_i^{(h)}(t)$.

The output of the ith hidden neuron is then defined by

$$y_i^{(h)}(t) = \varphi_i\left[u_i^{(h)}(t)\right] = \varphi_i\left[\sum_{j=0}^{P} w_{ij}(t)x_j(t)\right], \tag{2}$$

where $\varphi_i(\cdot)$ is usually a sigmoidal function. Similarly, the output values of the output neurons are given by

$$y_k^{(o)}(t) = \varphi_k\left[u_k^{(o)}(t)\right] = \varphi_k\left[\sum_{i=0}^{Q} m_{ki}(t)y_i^{(h)}(t)\right], \tag{3}$$

where m_{ki} is the synaptic weight connecting the ith hidden neuron to the kth output neuron ($k = 1, \ldots, M$), and $M \geq 1$ is the number of output neurons. Again, for the purpose of simplifying notation, we set $y_0(t) = -1$ and $m_{k0} = \theta_k^{(o)}(t)$, where $\theta_k^{(o)}(t)$ is the threshold of the kth output neuron.

The backward pass starts at the output layer by propagating the error signals from the output layer toward the hidden layer. For that, we need first to compute the error

value $e_k^{(o)}(t)$ generated by each output neuron at current iteration t

$$e_k^{(o)}(t) = d_k(t) - y_k^{(o)}(t), \quad k = 1, \ldots, M \tag{4}$$

where $d_k(t)$ is the target value for the kth output neuron. The error $e_k(t)$ is multiplied by the derivative $\varphi_k'\left[u_k^{(o)}(t)\right] = \partial\varphi_k/\partial u_k^{(o)}$ before being propagated to the hidden layer:

$$\delta_k^{(o)}(t) = \varphi_k'\left[u_k^{(o)}(t)\right] e_k^{(o)}(t). \tag{5}$$

The quantity resulting from this multiplication is known as the *local gradient* of the kth output neuron. Similarly, the local gradient $\delta_i^{(h)}(t)$ of the ith hidden neuron is computed as

$$\begin{aligned}\delta_i^{(h)}(t) &= \varphi_i'\left[u_i^{(h)}(t)\right] \sum_{k=1}^{M} m_{ki}(t)\delta_k^{(o)}(t) \\ &= \varphi_i'\left[u_i^{(h)}(t)\right] e_i^{(h)}(t), \quad i = 0, \ldots, Q,\end{aligned} \tag{6}$$

where the term $e_i^{(h)}(t)$ plays the role of a *backpropagated* or *projected* error signal for the ith hidden neuron.

Finally, the synaptic weights of the output neurons are updated according to the following rule

$$m_{ki}(t+1) = m_{ki}(t) + \eta\delta_k^{(o)}(t)y_i^{(h)}(t), \quad i = 0, \ldots, Q, \tag{7}$$

where $0 < \eta \ll 1$ is the learning rate. The weights of the hidden neurons are, by their turn, adjusted through a similar learning rule

$$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_i^{(h)}(t)x_j(t), \quad j = 0, \ldots, P. \tag{8}$$

One complete presentation of the entire training set during the learning process is called an *epoch*. Many epochs may be required until the convergence of the backpropagation algorithm is verified. Thus, it is good practice to randomize the order of presentation of training examples from one epoch to the next, in order to make the search in the weight space stochastic over the learning cycles.

The simplest way of evaluating convergence is through the average squared error

$$\begin{aligned}\varepsilon_{\text{train}} &= \frac{1}{2N}\sum_{t=1}^{N}\sum_{k=1}^{M}\left[e_k^{(o)}(t)\right]^2 \\ &= \frac{1}{2N}\sum_{t=1}^{N}\sum_{k=1}^{M}\left[d_k(t) - y_k^{(o)}(t)\right]^2,\end{aligned} \tag{9}$$

computed at the end of a training run using the training data vectors. If it falls below a prespecified value, then convergence is achieved. The generalization performance of the MLP should be evaluated on a testing set, which contains examples not seen before by the network.

3 The MAXCORE principle

In Eq. 6, we have defined the *backpropagated* error signal for the ith hidden neuron as

$$e_i^{(h)}(t) = \sum_{k=1}^{M} m_{ki}(t)\delta_k^{(o)}(t) = \sum_{k=1}^{M} m_{ki}^*(t)e_k^{(o)}(t), \tag{10}$$

where $m_{ki}^*(t) = m_{ki}(t)\varphi_k'\left[u_k^{(o)}(t)\right]$ is the weight between the kth output neuron and the ith hidden neuron modulated by the derivative of the kth output activation function.

From Eq. 10, we can easily note that $e_i^{(h)}(t)$ is the result of a linear combination of the output error signals $e_k^{(o)}(t), k = 1, \ldots, M$, with weights $m_{ki}^*(t)$. Thus, the higher the value of $m_{ki}^*(t)$ and, hence, of $m_{ki}(t)$, the higher the correlations between $e_i^{(h)}(t)$ and $e_k^{(o)}(t)$. A similar linear relationship can be derived for the signals $e_i^{(h)}(t)$, associated with the hidden neurons, and the signals $e_j^{(i)}(t)$, associated with the input units (see Eq. 16).

From the exposed, we can state the following general principle, henceforth called the *Principle of Maximum Correlation of Errors* (MAXCORE):

The MAXCORE principle Relevant synaptic weights tend to generate higher correlations between error signals associated with the neurons of a given layer and the error signals propagated back to the previous layer. Nonrelevant (i.e. prunable) weights tend to generate smaller correlations.

The MAXCORE is the guiding principle behind the proposal of the weight pruning procedure to be described in the next sections.

4 The proposed methodology

The procedure to be described is a *pruning method*. The user should first train a MLP network with a relative large number of hidden neurons until achieves the lowest possible value for $\varepsilon_{\text{train}}$. Recall that large MLPs are prone to overfitting due to local optimization of their cost function, which can lead to a poor generalization performance. After that, the application of a pruning procedure can discard *nonrelevant* weights of an overparameterized network, thus providing a smaller model with equivalent or better generalization performance than the original one.

The main idea behind the pruning procedure to be described is to maintain those connections that produce higher correlations between the errors in a given layer and the errors that are propagated back to the preceding layer. Weights associated with smaller error correlations are candidates to be discarded.

4.1 Pruning hidden-to-output layer weights

The pruning procedure starts with the training data being submitted once again to the trained network. No weight updating is allowed from this stage on. Once all the N training examples are presented, determine the error matrices $\mathbf{E}_o$ and $\mathbf{E}_h$, defined as follows:

$$\mathbf{E}_o = \begin{bmatrix} e_1^{(o)}(1) & e_2^{(o)}(1) & \cdots & e_M^{(o)}(1) \\ e_1^{(o)}(2) & e_2^{(o)}(2) & \cdots & e_M^{(o)}(2) \\ \vdots & \vdots & \vdots & \vdots \\ e_1^{(o)}(N) & e_2^{(o)}(N) & \cdots & e_M^{(o)}(N) \end{bmatrix}_{N\times M} \tag{11}$$

and

$$\mathbf{E}_h = \begin{bmatrix} e_0^{(h)}(1) & e_1^{(h)}(1) & \cdots & e_Q^{(h)}(1) \\ e_0^{(h)}(2) & e_1^{(h)}(2) & \cdots & e_Q^{(h)}(2) \\ \vdots & \vdots & \vdots & \vdots \\ e_0^{(h)}(N) & e_1^{(h)}(N) & \cdots & e_Q^{(h)}(N) \end{bmatrix}_{N\times(Q+1)}. \tag{12}$$

The rows of the matrix $\mathbf{E}_o$ correspond to the errors generated by the output neurons for a given training example. Hence, this matrix is called from now on *the matrix of output errors*, in contrast to the matrix $\mathbf{E}_h$, whose rows correspond to the backpropagated errors associated with the hidden neurons. In particular, the first column of $\mathbf{E}_h$ corresponds to backpropagated errors associated with the thresholds $m_{k0} = \theta_k^{(o)}, k = 1, \ldots, M$.

The second step involves the computation of the following matrix product

$$\mathbf{C}_{oh} = \mathbf{E}_o^T \mathbf{E}_h, \tag{13}$$

where the superscript T denotes the transpose of a matrix. Note that the (k, i)th entry of $\mathbf{C}_{oh}$, denoted by $\mathbf{C}_{oh}[k, i]$, corresponds to the scalar product (i.e. the cross-correlation) of the kth column of $\mathbf{E}_o$ with the ith column of $\mathbf{E}_h$,

$$\mathbf{C}_{oh}[k, i] = \sum_{t=1}^{N} e_k^{(o)}(t) e_i^{(h)}(t), \tag{14}$$

for $k = 1, \ldots, M$, and $i = 0, \ldots, Q$.

The third step requires sorting the absolute values of entries $\mathbf{C}_{oh}[k, i]$ in ascending order

$$|\mathbf{C}_{oh}[\mathbf{r}_1]| < |\mathbf{C}_{oh}[\mathbf{r}_2]| < \cdots < |\mathbf{C}_{oh}[\mathbf{r}_L]|, \tag{15}$$

where the vector $\mathbf{r}_l = (k_l, i_l)$ contains the coordinates of the entry of position l in the ranking, and $L = \dim(\mathbf{C}_{oh}) = M \times (Q + 1)$ is the number of entries in $\mathbf{C}_{oh}$.

The fourth step involves the execution of the pruning procedure shown in Table 1. In this table, J_{train} denotes a performance index used to evaluate the network on the training data, such as the average squared error $\varepsilon_{\text{train}}$ or the classification rate CR_{train}. The constant J_{tol} is a user-defined value. If $J_{\text{train}} = \varepsilon_{\text{train}}$, then J_{tol} is the maximum error allowed for the training data. Thus, we eliminate a given connection $m_{\mathbf{r}_l}$ only if the value of J_{train}, computed after the elimination of that connection, remains lower than J_{tol}. If $J_{\text{train}} = \text{CR}_{\text{train}}$, then J_{tol} is the minimum recognition rate allowed for the training data. In this case, we eliminate a given connection $m_{\mathbf{r}_l}$ only if the value of J_{train}, computed after the elimination of that connection, still remains higher than J_{tol}.

4.2 Pruning input-to-hidden layer weights

For pruning the weights connecting the input units to the hidden neurons, $\{w_{ij}\}$, we now need to propagate the error signals $e_i^{(h)}(t)$ toward the input units in order to generate the error signals $e_j^{(i)}(t), j = 0, \ldots, P$:

$$e_j^{(i)}(t) = \sum_{i=1}^{Q} w_{ij}(t)\delta_i^{(h)}(t) = \sum_{i=1}^{Q} w_{ij}^*(t) e_i^{(h)}(t), \tag{16}$$

where $w_{ij}^*(t) = w_{ij}(t)\varphi_i'\left[u_i^{(h)}(t)\right]$ is the weight between the ith hidden neuron and the jth input unit modulated by the derivative of the ith hidden activation function.

After the presentation of N training vectors, the resulting errors can be organized into the error matrix $\mathbf{E}_i$, defined as

$$\mathbf{E}_i = \begin{bmatrix} e_0^{(i)}(1) & e_1^{(i)}(1) & \cdots & e_P^{(i)}(1) \\ e_0^{(i)}(2) & e_1^{(i)}(2) & \cdots & e_P^{(i)}(2) \\ \vdots & \vdots & \vdots & \vdots \\ e_0^{(i)}(N) & e_1^{(i)}(N) & \cdots & e_P^{(i)}(N) \end{bmatrix}_{N\times(P+1)}. \tag{17}$$

Similarly to the procedure described in the previous section, we need to compute the following matrix product

$$\mathbf{C}_{hi} = (\mathbf{E}_h^-)^T \mathbf{E}_i, \tag{18}$$

where the matrix $\mathbf{E}_h^-$ is obtained from matrix $\mathbf{E}_h$ by removing its first column. This does not influence the pruning process since there are no connections among the constant input of the hidden layer (i.e. $y_0(t) = -1$) and the input units.

The (i, j)th entry of $\mathbf{C}_{hi}$, denoted by $\mathbf{C}_{hi}[i, j]$, is given by

$$\mathbf{C}_{hi}[i, j] = \sum_{t=1}^{N} e_i^{(h)}(t) e_j^{(i)}(t), \tag{19}$$

for $i = 1, \ldots, Q$, and $j = 0, \ldots, P$. Then, we sort the absolute values of entries $\mathbf{C}_{hi}[i, j]$ in ascending order

$$|\mathbf{C}_{hi}[\mathbf{s}_1]| < |\mathbf{C}_{hi}[\mathbf{s}_2]| < \cdots < |\mathbf{C}_{hi}[\mathbf{s}_L]|, \tag{20}$$

where the vector $\mathbf{s}_l = (i_l, j_l)$ contains the coordinates of the entry of position l in the ranking, and $L = \dim(\mathbf{C}_{hi}) = Q \times (P + 1)$ is the number of entries in $\mathbf{C}_{hi}$.

The final step involves the execution of the pruning procedure shown in Table 2. For obvious reasons, from

Table 1 Pruning procedure for hidden-to-output layer weights

```
1.  Set l = 1;                          // set the ranking index to 1
2.  WHILE l ≤ L DO                      // begin the pruning loop
      2.1. Set a = m_rl;                // save current value
      2.2. Set m_rl = 0;                // set the weight to zero
      2.3. Compute J_train;             // Performance Index on training set
      2.4. IF J_train < J_tol,          // We set J_train = CR_train
           THEN
                Set m_rl = a;           // recover former value
           ENDIF
      2.5. Set l = l + 1;               // continue pruning
    ENDWHILE
```

now on we refer to the proposed method as the CAPE method (**C**ross-correlation **A**nalysis of back-**P**ropagated **E**rrors). This method must be applied reapeatedly until no more weights are to be pruned (see the flowchart in Fig. 1).

5 Model selection using the CAPE method

As a proof of concept, we work initially with a synthetic two-dimensional data set in order to visualize the hyperplanes (i.e. lines in this case) due to each hidden neuron and the resulting global decision surface. This data set is composed of 200 pattern vectors divided into two nonlinearly separable classes. For training, 140 of them are randomly chosen while the remaining ones are used for testing the performance of the pruned network.

The second set of simulations involves preliminary tests using two benchmarking data sets (Iris and Wine). The Iris data set is composed of 150 4-dimensional pattern vectors distributed into 3 classes. The Wine data set is composed of 178 13-dimensional vectors also distributed into 3 classes. For the Iris (Wine) data set, 35 (33) patterns per class are randomly selected for training purposes and the remaining 45 (79) ones are used for testing.

For a more demanding classification task, we have used a biomedical data set kindly made available for this research by the *Group of Applied Research in Orthopaedics* (GARO) of the *Centre Médico-Chirurgical de Réadaptation des Massues*, Lyon, France. The task consists in classifying patients as belonging to one out of three categories: Normal (100 patients), Disk Hernia (60 patients), or Spondylolisthesis (150 patients). Each patient is represented in the database by six biomechanical attributes

Table 2 Pruning procedure for input-to-hidden layer weights

```
1.  Set l = 1;                          // set the ranking index to 1
2.  WHILE l ≤ L DO                      // begin the pruning loop
      2.1. Set b = w_sl;                // save current value
      2.2. Set w_sl = 0;                // set the weight to zero
      2.3. Compute J_train;             // Performance Index on training set
      2.4. IF J_train < J_tol,          // We set J_train = CR_train
           THEN
                Set w_sl = b;           // recover former value
           ENDIF
      2.5. Set l = l + 1;               // continue pruning
    ENDWHILE
```

derived from the shape and orientation of the pelvis and cervical, thoracic, and lumbar spine: pelvic incidence, pelvic tilt, sacral slope, pelvic radius, lumbar lordosis angle, and grade of spondylolisthesis. The training set is formed by randomly selecting 42 pattern vectors per class, and the remaining 184 examples are used for testing purposes. This data set is available for the interested reader upon request. For the interested reader, we recommend the additional reading of the works of Berthonnaud et al. [3] and Rocha Neto and Barreto [26].

Weights and biases are randomly initialized within the range -0.5 to $+0.5$. All neurons use hyperbolic tangent activation functions, and the inputs are normalized to the range of network activations. Features are normalized to zero mean and unit variance. The output target vectors use the 1-out-of-M binary encoding. The learning rate was set to $\eta = 0.001$ for all simulations. All the networks and pruning methods were implemented in MATLAB 7.0.

5.1 Results for the synthetic data set

For the first data set, we initially trained a fully connected MLP network with $Q = 12$ hidden neurons. The input and output dimensions were set to $P = 2$ and $M = 1$, respectively. This network was trained until the MSE value for the training data, $(\varepsilon_{\text{train}})$, reached a steady-state, i.e. no significant changes are observed.

Numerical results are shown in Table 3. In this table, $N_c = (P + 1) \times Q + (Q + 1) \times M$ is the initial number of weights, CR_{train} and CR_{test} stand for the classification rate for the training and testing data, respectively, $\varepsilon_{\text{test}}$ is the mean-squared error in testing data set, and AIC stands for Akaike's Information criterion.[1]

The application of the CAPE and OBS methods involves several steps. Initially, they are applied to Architecture 1, with $Q = 12$ hidden neurons and, hence, $N_c = 49$ adjustable parameters as a whole. Note that this initial architecture classifies the data accurately. However, we are not only interested in good classification rates, but also in knowing whether there is redundancy, i.e. whether this architecture is oversized. The application of CAPE and OBS resulted in a pruned architecture with $Q = 9$ hidden neurons, but with less connections than a fully connected MLP architecture with $Q = 9$ neurons ($N_c = 37$). The classification rates of both architectures remained the same as before. The resulting AIC value is slightly higher for the architecture pruned by the CAPE method, mainly due to its larger number of parameters.

Once the performances of the pruned architectures for $Q = 9$ hidden neurons remained at acceptable levels, we started to wonder what would happen if we apply the

[1] The AIC has the follow structure $AIC = -2\ln(\varepsilon_{\text{train}}) + 2N_c$ [23].

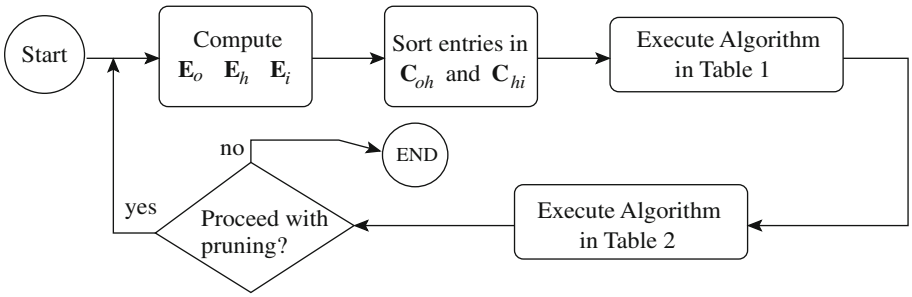

Fig. 1 Flowchart for the CAPE methodology

Table 3 Numerical results for the synthetic data set

	Q	N_c	CR_{train}	CR_{test}	ε_{train}	ε_{test}	AIC
Architecture 1	12	49	100	100	0.0090	0.0098	107.42
CAPE	9	29	100	100	0.0239	0.0231	65.47
OBS	9	28	100	100	0.0208	0.0166	63.75
Architecture 2	8	33	100	100	0.0088	0.0153	75.47
CAPE	7	22	100	100	0.0139	0.0119	52.55
OBS	7	22	100	100	0.0139	0.0119	52.55
Architecture 3	6	25	100	100	0.0075	0.0110	59.79
CAPE	4	17	100	100	0.0120	0.0087	42.85
OBS	4	17	100	100	0.0120	0.0087	42.85
Architecture 4	4	17	100	100	0.1385	0.1440	37.95
CAPE	3	13	100	100	0.1383	0.1458	29.96
OBS	3	13	100	100	0.1383	0.1458	29.96
WDE	4	17	100	98.33	0.0334	0.0442	40.80

pruning methods to another fully connected, one-hidden-layered MLP (Architecture 2) that has approximately the same number of connections $N_c = 33$ as the pruned Architecture 1. The rationale for doing that was to verify whether there was still room for further weight pruning. The application of CAPE and OBS to Architecture 2 led to the same pruned architecture with $Q = 7$ neurons and $N_c = 22$ connections. Again, note that the pruned Architecture 2 has less connections than a fully connected network with $Q = 7$ neurons ($N_c = 29$). We continued with the pruning procedure for more two fully connected networks, namely, Architectures 3 and 4. The results were identical for both the CAPE and OBS methods.

The results of the application of the WDE algorithm to the Architecture 1 are also shown in Table 3. Since this algorithm works during the training phase (see the "Appendix"), we cannot compare it directly with the OBS and CAPE methods, which are applied to already trained networks. Thus, only the final result is shown. The regularization parameter is set to $\lambda = 0.001$ after some experimentation. The application of the WDE algorithm resulted in an architecture with 4 neurons and 17 connections. However, the CR_{test} for the resulting architecture is lower than that achieved by the pruned Architecture 4.

In Fig. 2, it is shown the resulting global decision surface and the position of the hyperplanes (i.e. lines) associated with the unpruned Architecture 1. We can see clearly that there are *too many* lines (i.e. redundant neurons), much more than the number required to discriminate well between the two classes. For comparison purposes, the decision surface and the hyperplanes for Architecture 4, pruned by the CAPE method, are shown in Fig. 3, where only three lines remained. The successive application of the CAPE method, starting from Architecture 1, eventually led to the minimum number of hidden neurons required to solve this nonlinearly separable problem.

Note that the OBS algorithm also led to the same optimal three-hidden-neurons architecture, but at the expenses of much higher computational costs. Recall that the OBS algorithm requires the computation of the inverse of a Hessian matrix, one for each neuron in the network, every time the architecture is retrained. The CAPE method requires no matrix inversion at all. In addition, the CAPE method does not require the specification of the regularization parameter λ (WDE algorithm), which should be estimated by trial-and-error or by cross-validation.

5.2 Results for the Iris data set

Table 4 shows the results achieved by the pruned architectures on the Iris data set. The initial network topology

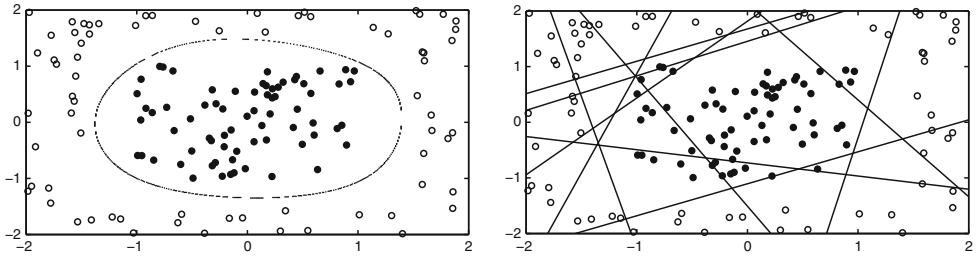

Fig. 2 Decision surface (*left*) and individual lines (*right*) for a trained MLP with $Q = 12$ hidden neurons

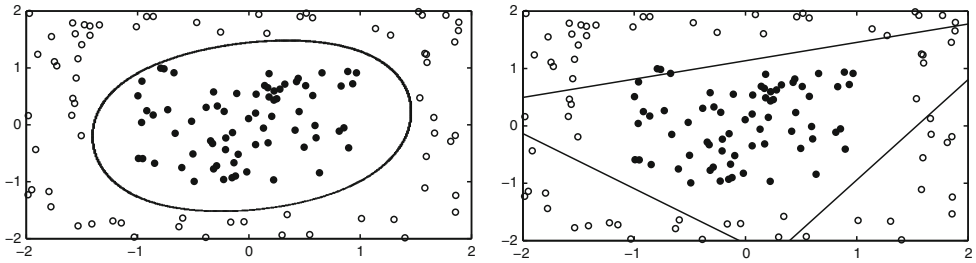

Fig. 3 Decision surface (*left*) and individual lines (*right*) for a pruned MLP with $Q = 3$ hidden neurons

($Q = 9$ hidden neurons) is independently trained and tested four times, using the same training and testing data sets. Only the weight initialization and the order of presentation of the training vectors per epoch were changed from one training/testing run to another. This was necessary to emphasize differences between the two pruning methods.

Weight pruning is carried out using the highest classification rate achieved for training data as reference (i.e. $J_{\text{train}} = \text{CR}_{\text{train}}$). In general, both methods presented equivalent performance on testing data if we consider only average classification rates and mean-squared errors in the evaluation. In terms of the final number of weights, the OBS method presented a slight advantage over the CAPE, since it ended with a smaller number of weights and hidden neurons. However, as we show later, if we also consider the resulting confusion matrix for the pruned networks, the networks pruned by CAPE performed better than the ones pruned by OBS.

An interesting behavior was observed in this simulation. Those architectures labeled with an asterisk (*) had an output neuron pruned. For all four initial architectures, the application of the CAPE method pruned an output neuron, while the OBS method pruned an output neuron in 3 out of 4 cases. As mentioned earlier, occasionally a neuron has to be pruned when all the connections converging to (or emanating from) that neuron have been pruned. In this simulation specifically, the fact that an output neuron has been pruned and may indicate that the output coding is redundant. As a matter of fact, the 1-out-of-M output encoding used in all simulation in this paper required three neurons to represent the three classes of the Iris data set.

Table 4 Results of the application of the pruning methods (Iris data)

Method	Q	N_c	CR_{train}	CR_{test}	$\varepsilon_{\text{train}}$	$\varepsilon_{\text{test}}$
Model 1	9	75	99.05	95.56	0.0199	0.0562
CAPE (*)	9	41	99.05	95.56	0.1822	0.2153
OBS (*)	9	41	99.05	95.56	0.1849	0.2162
Model 2	9	75	99.05	93.33	0.0194	0.0580
CAPE (*)	7	37	99.05	95.56	0.1932	0.2309
OBS (*)	7	37	99.05	93.33	0.1884	0.2243
Model 3	9	75	99.05	93.33	0.0203	0.0677
CAPE (*)	6	20	99.05	93.33	0.2969	0.3144
OBS (*)	5	18	99.05	93.33	0.3444	0.3586
Model 4	9	75	99.05	93.33	0.0127	0.0858
CAPE (*)	4	16	99.05	93.33	0.1765	0.2422
OBS	2	11	99.05	93.33	0.2868	0.3793
WDE	9	75	99.05	93.33	0.3277	0.3274

However, it is possible to represent the three classes using only two output neurons! This redundancy in output class representation was found, without any a priori information about the task, by both the CAPE and OBS methods, but it is particularly a strong feature of the CAPE method.

Figure 4a shows the classification results for Model 1 with $Q = 9$ hidden neurons and $M = 3$ output neurons. The points in the figure correspond to neurons' output values $y_k^{(o)}$, $k = 1, 2, 3$; thus, the output space is three-dimensional. The output coordinates $(y_1^{(o)}, y_2^{(o)}, y_3^{(o)})$ corresponding to input vectors classified as belonging to the setosa class (in blue) are concentrated around the

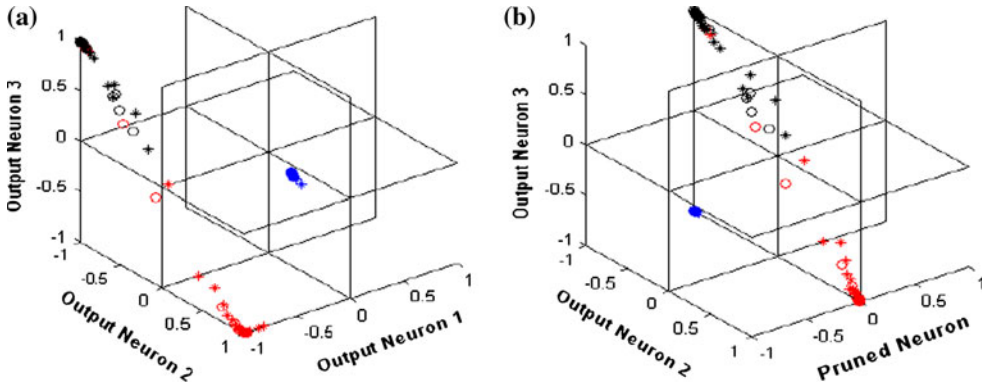

Fig. 4 Training (*asterisk*) and testing (*circle*) classification results for the Iris data set. **a** Unpruned MLP and **b** pruned MLP with pruned output set to zero

coordinates (+1, −1, −1). Points corresponding to input vectors classified as belonging to the versicolor class (in red) and to the virginica class (shown in black) are concentrated around the coordinates (−1, +1, −1) and (−1, −1, +1), respectively. There is some superposition among points belonging to the versicolor and virginica classes, resulting in a few classification errors.

Figure 4b shows the resulting classification achieved by the Model 1 after the application of the CAPE method. The pruned architecture presents $N_c = 41$ weights, $Q = 9$ hidden neurons, and still $M = 3$ output neurons. However, since all the weights from the hidden layer to the output neuron 1 has been pruned, the output of this neuron (i.e. $y_1^{(o)}$) plays no role now and has been set to zero. Hence, the output coordinates for the classes setosa, versicolor, and virginica now spread around the following points (0, −1, −1), (0, +1, −1), and (0, −1, +1), respectively.

Numerical results are shown in Table 4. Average classification rates and error values were computed for the pruned networks using the original 1-out-M encoding for desired outputs, maintaining three output neurons, however with the pruned output set to zero (i.e. $y_1^{(o)} = 0$). Despite that, classification rates for the testing data have been maintained at acceptable levels (i.e. >93%) for the pruned architectures.

If we retrain the pruned architectures now using only two output neurons, with desired outputs encoded as (−1, −1) (setosa class), (+1, −1) (versicolor class), and (−1, +1) (virginica class), the classification rates improve considerably. For example, the classification rates for the CAPE-pruned Model 1 changed from 95.5% (3rd row in Table 4) to 98.9% (not shown in Table 4). It was also observed a reduction in the mean-squared error values. The testing error changed from $\varepsilon_{\text{test}} = 0.2153$ to $\varepsilon_{\text{test}} = 0.0829$. Improvements in average classification rates and mean-squared errors were also observed for the OBS-pruned Model 1.

We also analyzed the effects of pruning on the confusion matrices of the classifiers. The confusion matrix of the unpruned Model 4 is shown in Table 5 using testing data

Table 5 Confusion matrix for unpruned Model 4 (Iris data)

	Predicted class		
Real class	33.3	0	0
	0	26.7	6.7
	0	0	33.3

Table 6 Confusion matrices for the pruned Model 4 (Iris data)

	CAPE			OBS		
	Predicted class			Predicted class		
Real Class	33.3	0	0	33.3	0	0
	0	26.7	6.7	2.3	26.7	4.4
	0	0	33.3	0	0	33.3

(45 pattern vectors, 15 for each class). The first row of this table indicates that all 15 pattern vectors of class setosa (i.e. 33.3% of the total amount of testing data) are correctly assigned to this class. The same occurs for the 15 pattern vectors of class virginica (third row). For the 15 data vectors of the class versicolor, 12 of them (26.7% of the total) are correctly classified as belonging to the class versicolor, while 3 pattern vectors of the class versicolor (6.7% of the total) are erroneously assigned to class virginica.

The confusion matrices obtained for Model 4, pruned by the CAPE and OBS methods, are shown in Table 6. It is clear that the confusion matrix obtained from the CAPE-pruned Model 4 is better than that produced by the OBS-pruned Model 4. As a conclusion, we recommend that the pruned architectures must be further evaluated by their confusion matrices, when presenting equivalent classification rates.

5.3 Results for the vertebral column data set

For this simulation, we set $J_{\text{tol}} = 85\%$ to serve as a reference for the minimum acceptable recognition rate for

Table 7 Numerical results of the successive application of the pruning methods to vertebral column data set

	Q	N_c	CR_{train}	CR_{test}	ε_{train}	ε_{test}	AIC
Architecture 1	24	243	89.68	87.50	0.1132	0.1274	490.36
CAPE	19	126	92.06	86.96	0.1662	0.1273	255.59
OBS	22	134	92.06	86.96	0.1777	0.1598	271.46
PWM	21	121	89.68	84.24	0.1733	0.1544	245.51
Architecture 2	18	183	96.03	86.41	0.0616	0.1891	371.57
CAPE	14	98	95.24	88.59	0.1089	0.1632	200.43
OBS	14	91	95.24	87.50	0.1529	0.1827	185.76
PWM	16	96	96.03	88.59	0.1453	0.1918	195.86
Architecture 3	13	133	93.65	81.52	0.0765	0.2311	271.14
CAPE	13	103	93.65	83.70	0.1211	0.2113	210.22
OBS	13	102	93.65	80.43	0.1267	0.2410	208.13
PWM	13	96	93.65	78.80	0.1356	0.2348	196.00

training and testing purposes. That is, even if a pruned network has produced a CR_{train} value higher than 85%, its CR_{test} value should also be higher than 85%. Network pruning is carried out progressively through successive applications of the CAPE/OBS/PWM methods. Numerical results are summarized in Table 7.

The CAPE method has produced a pruned Architecture 1 with $Q = 19$ hidden neurons and $N_c = 126$ connections, i.e. with less connections than a fully connected MLP architecture with the same number of hidden neurons ($N_c = 193$). The OBS method has ended with a pruned network with $Q = 22$ hidden neurons and $N_c = 134$. The final classification rates (CR_{train} and CR_{test}) of both methods were coincidentally the same.

Important facts are worth mentioning with respect to the PWM method [5], which is described in the "Appendix". Firstly, despite the fact that it has ended with the network with the smallest number of connections ($N_c = 121$) for $Q = 21$ hidden neurons, its CR_{test} is smaller than 85%. Secondly, due the small number of weights, it has achieved the lowest AIC value, wrongly indicating this architecture as the best one for this problem. These results indicate that the AIC index is possibly not the best model selection method for classification problems, since it is based on the average squared error only, not on classification rates.

Similarly to the procedure described in Sect. 5.1, we have trained another fully connected, one-hidden-layered MLP, denoted Architecture 2, with a number of hidden neurons $Q = 18$ close to the final value achieved by the CAPE-pruned Architecture 1. The training parameters for the Architecture 2 were the same as before. The application of the CAPE and the OBS methods to Architecture 2 led to the same number of hidden neurons ($Q = 14$), with the latter achieving less connections than the former. However, the classification rate CR_{test} for the CAPE method was higher. The AIC model, in this case, indicated the OBS method as the one providing the best pruned architecture, despite the fact that the CAPE method has produced the highest classification rate in testing data. This was mainly due to the smaller number of connections achieved by the OBS method.

The PWM method performed quite well in terms of classification rates, ending with a pruned architecture with $Q = 16$ number of neurons and $N_c = 96$ connections. It is worth mentioning, however, that the PWM is highly sensitive to weight initialization, and the numerical results shown in Table 7 for the PWM method corresponds to the best one obtained after 10 training/testing runs. The CAPE and the OBS are much less sensitive to weight initialization.

As the recognition rate CR_{test} for the pruned Architecture 2, irrespective of the pruning method, also remained above $J_{tol} = 85\%$, we decided to start pruning another fully connected, one-hidden-layered MLP, with $Q = 13$. However, no method was able to reduce further the number of hidden neurons, only the number of weights. Anyway, all pruned networks achieved recognition rates on testing data below the minimum allowed value. Thus, one can infer that the architectures best suited to the problem are the pruned ones obtained from Architecture 2.

For the sake of comparison, we evaluated two SVM classifiers, using linear and RBF kernels, for 50 runs on the vertebral column data set. The SVM classifiers were both trained by the standard SMO algorithm Platt [22]. These classifiers achieved average recognition rates of 85.1% (linear kernel) and 85.3% (RBF kernel), which are much inferior than the recognition rate reported in Table 7 for the CAPE-pruned Architecture 2 (88.59%).

6 Feature subset selection using CAPE

If all the weights emanating from one or more input units have been discarded, the features associated with those input nodes are to be considered nonrelevant ones, since they will play no role in the computations. Thus, the proposed weight pruning method can be also used for feature selection.[2] For this simulation, we used the Wine (13 features and 178 instances) and the Dermatology (34 features and 366 instances) benchmarking data sets.

For the *Wine* data set, we trained an MLP with $Q = 9$ hidden neurons using all $N_{feat} = 13$ features and the whole data set.[3] The total number of weights is $N_c = 156$. The

[2] Since the proposed approach is dependent on the classifier model, it belongs to the class of *wrappers* for feature subset selection ([16]).

[3] Recall that the task now is feature selection, not pattern classification. Thus, we can train the network with all the available pattern vectors.

Table 8 Progressive feature selection for the Wine data set

	Q	N_c	CR_{train}	ε_{train}	N_{feat}	*Out*
Architecture 1	9	156	100	0.0006	13	–
CAPE	7	37	100	0.2018	10	6, 7, 9
OBS	9	62	100	0.0257	12	6
Architecture 2	9	129	100	0.0023	10	–
CAPE	7	32	100	0.2599	9	5
OBS	8	43	100	0.0360	10	–
Architecture 3	9	120	100	0.0011	9	–
CAPE	5	26	100	0.0953	8	2
OBS	6	29	100	0.0717	9	–
Architecture 4	9	111	100	0.0011	8	–
CAPE	4	25	100	0.1100	8	–
OBS	5	28	100	0.0349	8	–

network was trained until it achieved $CR_{train} = 100\%$. Then, the CAPE method was applied with $J_{tol} = 99.50\%$.

The results shown in Table 8 suggest that there are at least 3 redundant input features, as we can see on the rightmost column. Hence, we created another training data set, now with only $N_{feat} = 10$ features by excluding the features 6, 7, and 9. Another MLP was then trained and pruned. This procedure was repeated until no more features were to be discarded. At the end of 4 runs of the CAPE method, only 8 features remained.

Figure 5a, b illustrate CAPE's pruning power. In Fig. 5a, we can see the original fully connected network, Architecture 1 ($P = 13$, $Q = 9$, $M = 3$, and $N_c = 156$), while in Fig. 5b, we see the resulting pruned architecture ($N_{feat} = 8$, $Q = 4$, $M = 3$, and $N_c = 25$) after the application of CAPE method. As final step, we evaluated this pruned architecture by training and testing it using the 10-fold cross-validation scheme. No weight pruning was allowed at this step. The resulting average classification rate was 98.60%.

To further evaluate CAPE in feature selection tasks, we trained a fully connected MLP with $Q = 10$ hidden neurons using the *Dermatology* data set, which contains 366 patterns distributed into six classes. Each vector had initially $N_{feat} = 34$ features that were used to categorize six skin diseases. The whole data set was used for the training process. A procedure similar to the one applied to the *Wine* data set was followed. All architectures were trained to achieve $CR_{train} > 99.50\%$ with $J_{tol} = 99.50\%$. The results are summarized in Table 9.

The CAPE-based feature subset selection approach has led to the elimination of 15 features at the end of the pruning process of the Architecture 7, while the OBS-based approach has led to the elimination of only 8 features. Using the remaining 19 features, the pruned Architecture 7 achieved a recognition rate of $CR_{train} = 99.73\%$.

An interesting situation happens when we further apply the CAPE algorithm to Architecture 8. Elimination of features 16 and 26 is suggested (1st trial). However, when we pruned out these two features, the resulting architecture did not achieve a recognition rate above the acceptable minimum (i.e. $CR_{train} < J_{tol} = 99.50\%$) for a new trained MLP using only 17 features. We then decided to train two other MLPs using 18 features. The first one is trained after the elimination of feature 16 only (2nd trial), while the second one is trained after the elimination of feature 26 only (3rd trial). The 2nd trial achieved the minimum acceptable recognition rate, while the 3rd trial did not. So, we maintained feature 26 and eliminated feature 16.

The final MLP classifier, pruned by CAPE on the Dermatology data set, presents the following numbers: $N_{feat} = 18$, $Q = 10$, $M = 6$, and $N_c = 109$. The features 1, 7, 11, 12, 13, 16, 17, 18, 20, 24, 25, 29, 30, 32, 33, and 34 can be discarded, since they are considered nonrelevant

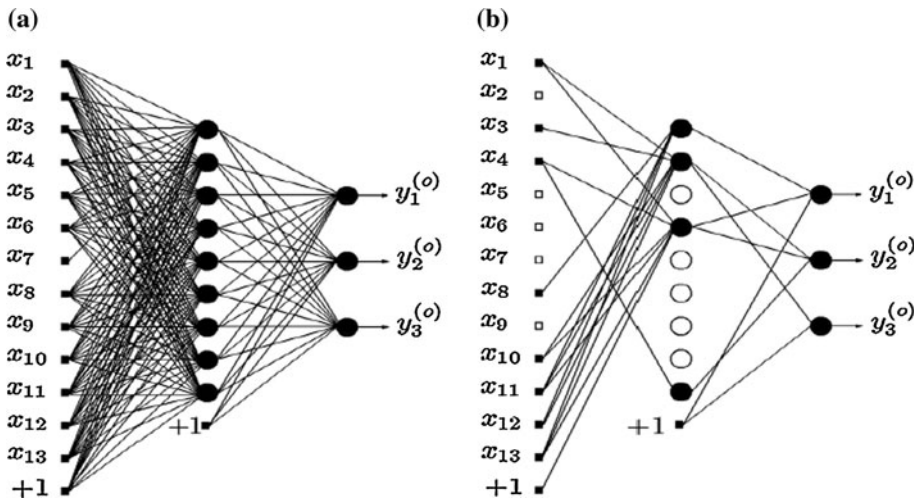

Fig. 5 **a** Fully connected MLP ($Q = 9$ and $N_c = 156$) and **b** pruned MLP ($Q = 4$ and $N_c = 25$)

Table 9 Progressive feature selection for the Dermatology data set

	Q	N_c	CR_{train}	ε_{train}	N_{feat}	*Out*
Architecture 1	10	416	99.73	0.0030	34	–
CAPE	7	85	99.73	0.0193	27	1, 11, 20, 29, 32, 33, 34
OBS	9	141	99.73	0.0298	31	1, 11
Architecture 2	10	346	99.73	0.0034	27	–
CAPE	8	106	99.73	0.0607	25	17, 30
OBS	10	110	99.73	0.0423	26	30
Architecture 3	10	326	100	0.0023	25	–
CAPE	10	121	100	0.0648	23	12, 25
OBS	10	115	100	0.0595	24	16
Architecture 4	10	306	99.73	0.0037	23	–
CAPE	7	78	99.73	0.0580	22	18
OBS	10	110	99.73	0.0374	23	–
Architecture 5	10	296	99.73	0.0036	22	–
CAPE	9	104	99.73	0.0900	21	24
OBS	10	110	99.73	0.0860	22	–
Architecture 6	10	286	99.73	0.0032	21	–
CAPE	9	98	99.73	0.0814	20	7
OBS	9	98	99.73	0.0454	19	7, 10
Architecture 7	10	276	99.73	0.0039	20	–
CAPE	9	97	99.73	0.0525	19	13
OBS	10	92	99.73	0.0525	19	13
Architecture 8	10	266	99.73	0.0042	19	–
CAPE (1st trial)	9	78	99.00	0.0485	17	16, 26
CAPE (2nd trial)	9	78	99.73	0.0485	17	16
CAPE (3rd trial)	9	78	99.00	0.0485	17	26
OBS	10	111	99.73	0.0366	18	16

for the classification task of interest. As final step, we evaluated the pruned architecture by training and testing it using the tenfold cross-validation scheme. No weight pruning is allowed at this step. The resulting average classification rate was 98.00%.

6.1 Comparison with related methods

In this section, we compare the performance of the CAPE-based feature subset selection method with those reported in some recent works available in the literature.

Luukka [18] introduced a wrapper approach for feature subset selection using fuzzy entropy measures. This approach removed only 5 features of the Dermatology data set, achieving an average classification rate of 98.28%. Rocha et al. [25] proposed two strategies for evolving MLP-like classifiers, named *Topology-optimization Evolutionary Neural Network* (TENN) and *Simultaneous Evolutionary Neural Network* (SENN) and evaluated them on several benchmarking data sets, including the Dermatology data set. It is reported average classification rates of 95.7% (TENN) and 95.3% (SENN). For the sake of comparison, on the Dermatology data set, the CAPE-based feature selection method resulted in an average classification rate of 98.00% using only 18 (out of 34) features.

Moustakidis and Theocharis [19] introduced the SVM-FuzCoC, a powerful SVM-based feature selection method that achieved an average classification rate of 97.12% for the Wine data set using 6 features in average. For the Dermatology data set, the SVM-FuzCoC achieved an average classification rate of 94.11% using 12 features in average. For the Dermatology (Wine) data set, the CAPE-based feature selection method resulted in an average recognition rate of 98.00% (98.6%) using 18 (8) features. As a conclusion, the performance of the SVM-FuzCoC method in terms of the trade-off between the smallest number of features and the highest classification rate is slightly better than that provided by the CAPE method, but this is achieved at the expenses of much higher computational efforts.

7 CAPE × OBS: estimating the computational costs

The computational cost is an important issue to be addressed during evaluation of algorithms. Sometimes, the algorithm effectiveness in providing a solution to a given problem requires the execution of complex computations and the use of excessive memory resources, which can create severe difficulties in real-time or low-cost applications.

In this paper, we have evaluated the CAPE and OBS methods in several classification and feature selection tasks. One may argue that the CAPE performed only slightly better than the OBS to justify its choice as a pruning method in the place of the classical OBS/OBD methods. As mentioned before, the OBS/OBD methods require the computation of the inverse Hessian matrix for a given neuron in order to determine the weight saliencies (see the "Appendix"). Computation of the inverse Hessian matrix for MLP networks is a tricky task itself. Exact computation of the Hessian matrix for MLP networks has been made possible [4], but its inversion is still subject to numerical ill-conditioning. In this regard, the CAPE method demands much less computational efforts that the OBS/OBD methods, since no matrix inversion is required.

In the results to be described, we take into account all the essential mathematical operations used to compute the weight saliencies (S_i), as required by the OBS algorithm, and the cross-correlation indexes ($\mathbf{C}_{oh}[k,i]$ and $\mathbf{C}_{hi}[i,j]$) in CAPE algorithm. Tables 10 and 11 show the number of mathematical operations required by the CAPE and OBS methods, respectively, as a function of the number of pattern vectors (N), the input dimension (P), the number of hidden neurons (Q), and the number of output neurons (M).

Figure 6 shows how the number of operations required by both pruning methods evolves as a function of the number of weights. For the sake of simplicity, we have considered that sums, multiplications, divisions, etc. have the same computational cost.

The number of operations required by the OBS (denoted by $N_{op\_obs}(N_c)$) and the CAPE (denoted by $N_{op\_cape}(N_c)$) was computed from the application of both algorithms over an MLP with $P = 2$ input units, $M = 1$ output neuron, and the number of hidden neurons Q ranging from 1 to 10. The number of instances in the data set is $N = 140$. One can easily infer that the OBS method has a quadratic dependence on N_c, while the CAPE method presents a linear dependence with very small slope. This result can help a user in deciding in favor of the application of the CAPE pruning method to pattern classification.

Table 10 Mathematical operations demanded by the CAPE method

	Number of operations
+	$N.[3PQ + 3Q.M - P + 2Q + 3M - 2] - PQ - QM - M - 1$
−	$N.[Q + 2M]$
×	$N.[4PQ + 4QM + 5Q + 5M]$
÷	–
Tanh	$N.[Q + M]$

Table 11 Mathematical operations demanded by the OBS method

	Number of operations
+	$N.[3P^2Q^2 + 3P^2Q^2.M + 3Q^2M^2 + 2QM + Q + M]$
−	$N.[2PQ^2M + P^2Q^2 + Q^2M^2 + PQ + 4QM + Q + 2M]$
×	$N.[8PQ^2M + 4P^2Q^2 + 4Q^2M^2 + 8PQ + 11QM + 3Q + 3M]$
÷	$N.[P^2Q^2 + 2PQ^2M + Q^2M^2 + PQ + QM]$
Tanh	$N.[Q + M]$

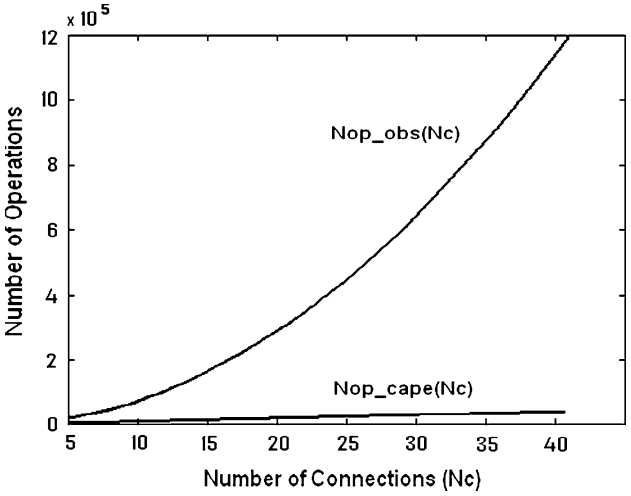

Fig. 6 Estimated costs for CAPE × OBS when pruning an MLP (2,Q, 1) classifier

8 Conclusions

In this paper, we introduced the CAPE method, an easy-to-apply and efficient procedure for pruning nonrelevant weights of a trained MLP classifier. The CAPE method was motivated by the MAXCORE principle, and it is based on the correlation analysis of the errors produced by the output neurons and the errors propagated back to previous layers. Comprehensive computer simulations using synthetic and real-world data indicate that, in terms of performance, the CAPE method compares favorably with standard pruning

techniques, such as the OBS, WDE, and PWM, with the advantage of requiring much lower computational efforts.

Acknowledgments The authors thank Prof. Ajalmar Rêgo da Rocha Neto (Federal Institute of Ceará—IFCE) for running the experiments with the SVM classifiers on the vertebral column data set. We also thank the anonymous reviewers for their valuable suggestions for improving this paper.

Appendix

The WDE algorithm originates from a regularization method that modifies the error function by adding a term that penalizes large weights. As a consequence, Eqs. 7, 8 are now written as [23]

$$m_{ki}(t+1) = m_{ki}(t)\left(1 - \frac{\lambda}{(1+m_{ki}^2(t))^2}\right) + \eta\delta_k^{(o)}(t)y_i^{(h)}(t),$$

$$w_{ij}(t+1) = w_{ij}(t)\left(1 - \frac{\lambda}{(1+w_{ij}^2(t))^2}\right) + \eta\delta_i^{(h)}(t)x_j(t),$$

where $0 < \lambda < 1$ is a user-defined parameter.

The OBS algorithm [15] requires that the weights are ranked based on the computation of *weight saliencies* defined as

$$S_i = \Delta E_i = \frac{1}{2}\frac{\omega_i^2}{[\mathbf{H}^{-1}]_{ii}} \tag{21}$$

where ω_i is the ith weight (or bias) of interest and $[\mathbf{H}^{-1}]_{ii}$ is the ith diagonal entry of the inverse of the Hessian matrix $\mathbf{H} = [H_{ij}] = \frac{\partial^2 E}{\partial\omega_i\partial\omega_j}$.

Pruning by weight magnitude (PWM) is a pruning method based on the elimination of small magnitude weights ([5]). Weights are sort in increasing order of magnitude. Starting from the smallest weight, a given weight is pruned as long as its elimination does not decrease the classification rate in training data set to a value below a predefined value.

References

1. Aran O, Yildiz OT, Alpaydin E (2009) An incremental framework based on cross-validation for estimating the architecture of a multilayer perceptron. Int J Pattern Recogn Artif Intell 23(2):159–190
2. Benardos PG, Vosniakos GC (2007) Optimizing feedforward artificial neural network architecture. Eng Appl Artif Intell 20(3):365–382
3. Berthonnaud E, Dimnet J, Roussouly P, Labelle H (2005) Analysis of the sagittal balance of the spine and pelvis using shape and orientation parameters. J Spinal Disorders Tech 18(1):40–47
4. Bishop CM (1992) Exact calculation of the hessian matrix for the multi-layer perceptron. Neural Comput 4(4):494–501
5. Bishop CM (1995) Neural networks for pattern recognition. Oxford University Press, Oxford
6. Castellano G, Fanelli AM, Pelillo M (1997) An iterative pruning algorithm for feedforward neural networks. IEEE Trans Neural Netw 8(3):519–531
7. Cataltepe Z, Abu-Mostafa YS, Magdon-Ismail M (1999) No free lunch for early stopping. Neural Comput 11(4):995–1009
8. Curry B, Morgan PH (2006) Model selection in neural networks: some dificulties. Eur J Oper Res 170(2):567–577
9. Dandurand F, Berthiaume V, Shultz TR (2007) A systematic comparison of flat and standard cascade-correlation using a student-teacher network approximation task. Connect Sci 19(3): 223–244
10. Delogu R, Fanni A, Montisci A (2008) Geometrical synthesis of MLP neural networks. Neurocomputing 71:919–930
11. Engelbrecht AP (2001) A new pruning heuristic based on variance analysis of sensitivity information. IEEE Trans Neural Netw 12(6):1386–1399
12. Fahlman SE, Lebiere C (1990) The cascade-correlation learning architecture. In: Touretzky DS (ed) Advances in neural information processing systems. Morgan Kaufmann, San Mateo, vol 2, pp 524–532
13. Gómez I, Franco L, Jerez JM (2009) Neural network architecture selection: can function complexity help? Neural Process Lett 30:71–87
14. Hammer B, Micheli A, Sperduti A (2006) Universal approximation capability of cascade correlation for structures. Neural Comput 17(5):1109–1159
15. Hassibi B, Stork DG (1993) Second order derivatives for network pruning: optimal brain surgeon. In: Hanson SJ, Cowan JD, Giles CL (eds) Advances in neural information processing systems. Morgan Kaufmann, San Mateo, vol 5, pp 164–171
16. Kohavi R, John GH (1997) Wrappers for feature subset selection. Artif Intell 97(1–2):273–324
17. Littmann E, Ritter H (1996) Learning and generalization in cascade network architectures. Neural Comput 8(7):1521–1539
18. Luukka P (2011) Feature selection using fuzzy entropy measures with similarity classifier. Exp Syst Appl 38(4):4600–4607
19. Moustakidis S, Theocharis J (2010) SVM-FuzCoC: a novel SVM-based feature selection method using a fuzzy complementary criterion. Pattern Recogn 43(11):3712–3729
20. Nakamura T, Judd K, Mees AI, Small M (2006) A comparative study of information criteria for model selection. Int J Bifur Chaos 16(8):2153–2175
21. Parekh R, Yang J, Honavar V (2000) Constructive neural-network learning algorithms for pattern classification. IEEE Trans Neural Netw 11(2):436–451
22. Platt JC (1998) Fast training of support vector machines using sequential minimal optimization. In: Advances in Kernel methods: support vector learning. MIT Press, Cambridge, pp 185–208
23. Principe JC, Euliano NR, Lefebvre WC (2000) Neural and adaptive systems. Wiley, London
24. Reed R (1993) Pruning algorithms—a survey. IEEE Trans Neural Netw 4(5):740–747
25. Rocha M, Cortez P, Neves J (2007) Evolution of neural networks for classification and regression. Neurocomputing 70(16–18):1054–1060
26. Rocha Neto AR, Barreto GA (2009) On the application of ensembles of classifiers to the diagnosis of pathologies of the vertebral column: a comparative analysis. IEEE Latin Am Trans 7(4):487–496
27. Saxena A, Saad A (2007) Evolving an artificial neural network classifier for condition monitoring of rotating mechanical systems. Appl Soft Comput 7(1):441–454
28. Seghouane AK, Amari SI (2007) The AIC criterion and symmetrizing the kullback-leibler divergence. IEEE Trans Neural Netw 18(1):97–106

29. Stathakis D, Kanellopoulos I (2008) Global optimization versus deterministic pruning for the classification of remotely sensed imagery. Photogrammetr Eng Remote Sens 74(10):1259–1265
30. Trenn S (2008) Multilayer perceptrons: approximation order and necessary number of hidden units. IEEE Trans Neural Netw 19(5):836–844
31. Wan W, Mabu S, Shimada K, Hirasawa K, Hu J (2009) Enhancing the generalization ability of neural networks through controlling the hidden layers. Appl Soft Comput 9(1):404–414
32. Weigend AS, Rumelhart DE, Huberman AB (1990) Generalization by weight-elimination with application to forecasting. In: Lippmann RP, Moody J, Touretzky DS (eds) Advances in neural information processing systems. Morgan Kauffman, San Mateo, vol 3, pp 875–882
33. Xiang C, Ding SQ, Lee TH (2005) Geometric interpretation and architecture selection of the MLP. IEEE Trans Neural Netw 16(1):84–96
34. Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447
35. Yu J, Wanga S, Xi L (2008) Evolving artificial neural networks using an improved PSO and DPSO. Neurocomputing 71(4–6): 1054–1060