Neurocomputing 72 (2009) 3020–3027

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Improving performance of neural classifiers via selective reduction of target levels ☆

I. Mora-Jiménez [a,*], A.R. Figueiras-Vidal [b]

[a] Department of Signal Theory and Communications, Universidad Rey Juan Carlos, 28943 Fuenlabrada, Madrid, Spain
[b] Department of Signal Theory and Communications, Universidad Carlos III de Madrid, 28911 Leganés, Madrid, Spain

ARTICLE INFO

Article history:
Received 23 July 2008
Received in revised form
6 April 2009
Accepted 12 April 2009
Communicated by D. Xu
Available online 7 May 2009

Keywords:
Artificial neural networks
Classification
Learning algorithm
Reduced target level
Sample selection

ABSTRACT

Reducing the level of the targets corresponding to training samples for a machine classifier using the outputs of an auxiliary classifier is interesting because it allows to save expressive power unnecessarily dedicated to increase the output level of well-classified samples. In this paper we propose an iterative form of this selective reduction of target levels with a simple linear reduction schedule. Extensive simulations show that the proposed method has not only a performance better than or equal to conventional training or using static versions of the reduction, but also with respect to support vector machines (SVM). This potential advantage is accompanied by a smaller size and a design effort not much higher than the corresponding SVM, thus making the proposed method very attractive for practical applications.

1. Introduction

Conventional search procedures for training neural classifiers equally use all the examples to minimize a sample estimate of the selected cost function; however, the real problem is to define appropriate classification borders [38], which is not exactly equivalent to any of these procedures, but that is the key to obtain good generalization [39]. Thus, the possibility of having some examples more relevant for a good training (for a good definition of borders) must be accepted, and these examples have to receive a higher weight in the cost function estimate to be minimized. The subsequent problems are to determine what are these samples and how to emphasize them, a process which is named sample selection or, better, sample editing. Consequently, to design effective sample editing methods is a key subject in order to construct high performance neural classifiers.

There are many methods of sample editing, but just two main families. The first comes from the pioneer proposal of Hart [16], who considered that erroneous samples are important (although assuming that they are near the border). In this way, his condensed nearest neighbor algorithm eliminates correctly classified samples in k-nearest neighbor (k-NN) classifiers. This idea has been followed by many other researchers: In [5], several selection and repetition schemes based on error measures are proposed and evaluated; the decision-based neural networks (DBNN) of Kung and Taur [19] only consider erroneous samples along training; in [40], the highest erroneous samples are included after a preliminary training to improve its results; Munro proposes to repeat training for erroneous samples until convergence [26]; and [37] constitutes an additional example of this class of approaches. Even the basic formulation of boosting schemes seems to be based on the margin error of all the samples [11,12,35].

On the other hand, [36] postulates that the proximity to the border (or to a reasonable approximation of it) is the essential aspect to select relevant samples, proposing a method to build a piecewise linear border by identifying the closest opposite pairs of each class sample clusters in the examples set and training local linear discriminants with them; Lyhyaoui et al. [21] presents more identification schemes of this type, but the result is used to define the centers of radial basis transformations whose output weights are conventionally trained, offering structures and performances similar to those of support vector machines (SVM). The authors of [39] adopt also this point of view, as well as in [6,7,29].

The maximum margin cost which SVMs use is a sort of predefined combination of both manners of weighting the samples [9,25], as the boosting emphasis really is, as demonstrated in [13]. There are even discussions on the relative importance of both kind of samples, such as [10] ([30] is also

☆ This work has been partially supported by Research Project TEC2007-68096-C02-01/TCM from Spanish Government.
* Corresponding author. Tel.: +34 91 488 82 07; fax: +34 91 488 75 00.
E-mail address: inmaculada.mora@urjc.es (I. Mora-Jiménez).

0925-2312/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2009.04.002

interesting); unfortunately, there are no clear conclusions about what problem characteristics indicate the importance of each class. Papers [13,14] present a form for boosting in which the consideration of both classes of possibly relevant samples is decided by a design selectable parameter. More sample editing methods are referenced in [32].

Our proposal, which we call *selective reduction of target levels* (SRTL), consists on applying a procedure which can be considered as a sample editing method: reducing the absolute value of the target for each sample according to some "indication" of its proximity to the border. By reducing the target value we mean that no training is done when the classifier output is further than this value; so, on the one hand, in the case of clearly well-classified samples we do not try to force the classifier parameters to give an unnecessary high output, allowing them more freedom to take care of the samples that are lying near the border. On the other hand, in the case of clearly wrong classified samples, that could hardly be recovered—and probably this will not be adequate—the corresponding absolute error value is reduced, with the same effect as above. Thus, we propose to pay attention mainly to samples that are not far from the decision borders, but without completely excluding the influence of the other samples when the decisions about them are not clear enough. This kind of editing seems to be a good election: on the one hand, it permits an easy implementation; on the other hand, it allows a fine determination of the classification borders, and, simultaneously, to use other information (that corresponding to clearly correctly or wrongly classified samples) in a smoother form.

With respect to the "indication" of the proximity of each sample to the border, its purpose is to have an approximate idea of how far is the sample from the border, because a calculation of the corresponding distance would be computationally demanding; even more, since the theoretical border for the problem is not known and we have only approximate versions of it, it would be an approximation itself. A simple possibility is to use just a classifier output, because making it equal to zero gives us an approximate border if the class targets are symmetrical. Even more, it is well known that for square error criteria [18,31,33] and other more general cost functions [1,8,22,28], the classifier output is an estimate of the "a posteriori" probabilities—of its difference in binary cases if we use targets $\{-1,+1\}$—better and better when the training sample is more and more representative and the expressive power of the classification machine increases. Obviously, such estimates are very reasonable indicators to reduce the target values.

The rest of the paper is organized as follows. In the next section, we introduce concrete formulations of our target reduction approach for binary problems and two kinds of SRTL strategies: static, or based on the output of an external guide for each sample, and iterative, or based on values that are initialized using statistics of all the sample outputs of an external guide and modified epoch by epoch according to an appropriate algorithm. We must remark that the algorithms we propose are computationally efficient and clearly increase the performance measures, but their basic ideas can be applied by means of many other formulations; as expected, the results are similar, and an improved performance is obtained if the particular heuristic scheme is reasonable from an intuitive point of view. In Section 3 we present and evaluate experimental results for several benchmark problems. Finally, we expose our conclusions in Section 4.

2. Proposed target reduction algorithms

Let us consider the case of a binary classification problem with classes C_0 and C_1 and assume we are given a set of N labelled data (training set) $\{\mathbf{x}^{(i)}, t(\mathbf{x}^{(i)})\}_{i=1}^{N}$, with $\mathbf{x}^{(i)} \in \Re^d$ being the i-th input pattern and $t(\mathbf{x}^{(i)})$ its associated hard target: $t(\mathbf{x}^{(i)}) = -1$ if $\mathbf{x}^{(i)} \in C_0$ and $t(\mathbf{x}^{(i)}) = 1$ otherwise, with the same target for all patterns in the same class.

In general, the higher the magnitude provided by the classifier output, the further the sample is expected to be from the boundary. However, it is not usual that all samples are far from the boundary. Therefore, to find a better definition of the decision boundary, it seems reasonable to demand a different target to each sample, in proportion with its distance to the boundary.

The target reduction algorithms we propose reduce the hard targets of the training samples according to an "indication" of the statistical proximity of every sample to the boundary. These algorithms start from an auxiliary classifier whose output provides an estimate of the proximity of each training sample to the boundary, and use this estimate to reduce the desired target in that sample. This way, statistical uncertainty in each training sample drives the learning process, which can be focused on the examples that are harder to learn at each training epoch.

In the following subsections, we propose two different target reduction strategies, according to the method used to determine the reduced targets: static, when the reduced targets are constant during training, and iterative, when they are continuously reduced as training proceeds.

2.1. Static strategy

For any training sample $\mathbf{x}$, its reduced target $t_s(\mathbf{x})$ has the form:

$$\mathrm{sgn}[t_s(\mathbf{x})] = \mathrm{sgn}[t(\mathbf{x})] \tag{1a}$$

$$|t_s(\mathbf{x})| = |o_g(\mathbf{x})| \tag{1b}$$

where sgn is the sign function, and $o_g(\mathbf{x})$ is the output of an auxiliary classifier, or guide, with $o_g(\mathbf{x}) : \Re^d \to [-1,1]$. Once the reduced target values are calculated for the training samples, the final classifier is trained with the hard targets, but considering a zero cost when the output is better than the soft target, i.e.,

$$C_s(t(\mathbf{x}), o(\mathbf{x})) = \begin{cases} C(t(\mathbf{x}), o(\mathbf{x})) & \text{if } |o(\mathbf{x})| < |t_s(\mathbf{x})| \text{and } \mathrm{sgn}[o(\mathbf{x})] = \mathrm{sgn}[t(\mathbf{x})] \\ 0 & \text{in other case} \end{cases} \tag{2}$$

where $o(\mathbf{x})$ is the output of the classifier which is being trained. In our experiments, we use square error costs. Table 1 illustrates the algorithm steps for the static strategy.

Eqs. (1a) and (1b) represent an immediate extension of a previous approach [24], in which the guide was a k-NN classifier

Table 1
Algorithm steps for the static target reduction strategy.

Step 1:	Obtain the outputs of the auxiliary classifier for the N samples of the training set
Step 2:	Initialize the weights of the neural classifier
Step 3:	Use the hard targets $\{t(\mathbf{x}^{(i)})\}_{i=1}^{N}$ and the outputs obtained in Step 1 to get the reduced targets $\{t_s(\mathbf{x}^{(i)})\}_{i=1}^{N}$ according to (1a) and (1b)
Step 4:	Use the current weights to obtain the outputs of the neural classifier for the N samples of the training set, $\{o(\mathbf{x}^{(i)})\}_{i=1}^{N}$
Step 5:	Consider the outputs in Steps 3–4 to create the edited training set, composed of: - the wrongly classified samples, i.e., samples such that $\mathrm{sgn}[o(\mathbf{x})] \neq \mathrm{sgn}[t(\mathbf{x})]$ - and the correctly classified samples satisfying $\lvert o(\mathbf{x})\rvert < \lvert t_s(\mathbf{x})\rvert$
Step 6:	Use the edited training set in Step 5 to update the weights of the neural classifier
Step 7:	Repeat Steps 4–6 until the desired accuracy is obtained

(k odd) and

$$|o_g(\mathbf{x})| = \frac{|k_1 - k_0|}{k_1 + k_0} \tag{3}$$

where $\{k_i\}, i = 0, 1$ are the number of training samples corresponding to class $C_i, i = 0, 1$, respectively, among the $k = k_1 + k_0$ nearest to $\mathbf{x}$. Obviously, this particular form does not need any auxiliary training, but it pays the well-known drawbacks of high memory and search requirements in operation; additionally, and besides needing to select k by cross-validation (CV), the discrete character of (3) can be a limitation to the effectiveness of the method; so, we consider here the option of using a neural network as the guide.

2.2. Iterative strategy

It is obvious that, when using the static strategy, high values of $|o_g(\mathbf{x})|$ mean high absolute values of the corresponding target in the (final) classifier being trained. This means that a part of the expressive power of the classifier is spent in trying to reach these targets, although it would be enough to get lower values with the appropriate sign for a correct classification. Thus, it could be adequate to have a maximum value for the target magnitude, $T_{\max}$. However, there is a difficulty: at the beginning of the training process, high (absolute) target values are important for a fast convergence and, even more, to drive the machine weights in a direction leading to a correct definition of the border. Therefore, low values of $T_{\max}$ can be risky, while high values are not efficient in the sense we have described. So, it is reasonable to consider the possibility of reducing the magnitude of the targets as training proceeds on the final classifier, starting at $T_{\max}(1) = T_{\max}$. We propose how to select $T_{\max}$ in the next paragraph.

On the other side, when $|o_g(\mathbf{x})|$ is near to zero, there is a clear risk of allowing an erroneous classification of the sample, because the importance of the corresponding error in the overall cost is reduced. Thus, a minimum absolute value for the targets, $T_{\min}$, is established. We have verified that it is adequate to establish $T_{\min}$ and initial $T_{\max}$ with the help of an auxiliary classifier, by means of

- Sorting in ascending order the absolute output values corresponding to wrongly classified training samples, and assigning to $T_{\min}$ the value of its 10th percentile.
- Sorting in descending order the absolute output values corresponding to correctly classified training samples, assigning to $T_{\max}$ the value of its 10th percentile, and setting $T_{\max}(1)$ to $T_{\max}$.

We have also checked that the final classification results are not very sensitive to the selected percentiles.

The algorithm consists on reducing linearly the maximum value of the magnitude of the reduced targets, $T_{\max}(n)$ in the n-th training epoch, along a (preselected) maximum number of training epochs, $n_{\max}$, from initial $T_{\max}(1) = T_{\max}$ to final $T_{\max}(n_{\max}) = T_{\min}$, and interpolating the reduced target absolute value when the absolute value of the guide output is between $T_{\max}(1)$ and $T_{\min}$ according to a linear scheduling

$$|t_s(\mathbf{x}, n)| = \frac{T_{\max}(n) - T_{\min}}{T_{\max}(1) - T_{\min}}(|o_g(\mathbf{x})| - T_{\min}) + T_{\min} \tag{4}$$

where $o_g(\mathbf{x})$ are the (given and fixed) outputs of the auxiliary classifier.

The sign of the reduced target is that of the original target, and we remark that

$$\text{-if } |o_g(\mathbf{x})| > T_{\max}(1), \quad |t_s(\mathbf{x}, n)| = T_{\max}(n) \tag{5a}$$

$$\text{-if } |o_g(\mathbf{x})| < T_{\min}, \quad |t_s(\mathbf{x}, n)| = T_{\min} \tag{5b}$$

The proposed scheduling to get the reduced targets is graphically illustrated in Fig. 1. Training of the final classifier is carried out in the same manner that in the static scheme, i.e., using the cost given by (2). The steps for the iterative target softening strategy are illustrated in Table 2.

We have also checked that the final classification performance is not sensitive to the scheduling given by (4); in fact, exponential scheduling gives very similar results.

Table 2
Algorithm steps for the iterative target reduction strategy.

Step 1:	Obtain the outputs of the auxiliary classifier for the N samples of the training set
Step 2:	Use the outputs in Step 1 to compute parameters $T_{\min}$ and $T_{\max}$ according to the procedure explained in Section 2.2
Step 3:	Initialize the following parameters: - the training epoch number, parameter n: $n \leftarrow 1$ - the maximum number of training epochs, parameter $n_{\max}$ - the weights of the neural classifier
Step 4:	Compute parameter $T_{\max}(n)$, corresponding to the maximum magnitude of the reduced targets in the n-th training epoch For the first training epoch, use $T_{\max}(1) \leftarrow T_{\max}$
Step 5:	Use the results of Step 1, Step 2, and Step 4 to compute the reduced targets in the n-th training epoch, $\{t_s(\mathbf{x}^{(i)})\}_{i=1}^{N}$, according to (4), (5a), and (5b)
Step 6:	Use the current weights to obtain the outputs of the neural classifier for the N samples of the training set, $\{o(\mathbf{x}^{(i)})\}_{i=1}^{N}$
Step 7:	Consider the outputs in Steps 5–6 to create the edited training set, composed of: - the wrongly classified samples, i.e., samples such that $\text{sgn}[o(\mathbf{x})] \neq \text{sgn}[t(\mathbf{x})]$, - and the correctly classified samples satisfying $\|o(\mathbf{x})\| < \|t_s(\mathbf{x})\|$
Step 8:	Use the edited training set in Step 7 to update the weights of the neural classifier
Step 9:	Increase the training epoch number, $n \leftarrow n + 1$
Step 10:	Repeat Steps 4–9 while $n < n_{\max}$ and the desired accuracy is not obtained

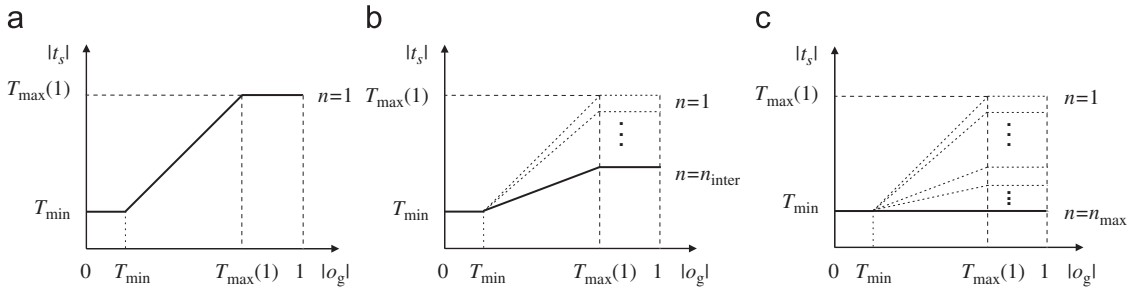


Fig. 1. Graphical representation of the schedule to determine the magnitude of the reduced target in the iterative SRTL strategy. (a) First training epoch ($n = 1$). (b) The curve used in an intermediate epoch ($n = n_{inter}$) in solid line, and the previous ones in dotted lines. (c) Situation in the last epoch ($n = n_{\max}$).

3. Experiments

In this section, we will evaluate classification accuracy, machine sizes, training effort, and operation computational load of the proposed schemes by comparing them with those of other machines when applied to a series of well-known benchmark problems. Before presenting the corresponding results, we provide the experiments details.

3.1. Experiment description

3.1.1. Designs under analysis

We will compare the proposed schemes:

- static guided by k-NN;
- static guided by neural network (NN);
- iterative.

among them and also versus the following machines: k-NN and conventionally trained NN (in order to get an idea of the improvements offered by using the target level reduction), DBNN (to have a similar training method as a reference), and SVM (to include a high performance machine which uses an implicit sample editing strategy).

We will adopt the radial basis function neural network (RBFNN) architecture with Gaussian elements for the conventional NN, DBNN, and proposed designs because the are well-known universal approximators [27] and they show good general characteristics. We use also Gaussian kernels for SVM. With respect to DBNN, we apply the hidden node DBNN structure [19] with both subnetworks being a linear combination of Gaussian RBF. And, finally, the Euclidean distance is used for the k-NN designs.

3.1.2. Datasets

We consider 11 binary classification problems. Nine of them are from the UCI Machine Learning Repository [3]: Abalone (*aba* in the paper), a multiclass problem converted into a binary problem according to [34], Wisconsin Breast Cancer (*bre*), SPECTF Heart images (*hea*), Image Segmentation (*ima*), Ionosphere (*ion*), Pima Indians Diabetes (*pim*), Sonar (*son*), Spambase (*spa*), and Waveform Data Generator (*wav*). The two other problems are synthetic: Ripley (*rpl*), presented in [32] and Twonorm (*2no*), introduced in [4].

All the datasets are preprocessed by normalizing each attribute to zero (sampled) mean and unit (sampled) standard deviation.

We use the predefined training and test sets when available. This is not the case for *bre*, *pim*, *spa*, and *2no*; for these datasets, we use 50 random partitions employing 60% of the data for the training stage and the remaining 40% for testing the classifiers performance. To select the design parameters, we will consider CV with 50 partitions of the training data, 80/20 for datasets with a predefined test partition and 66/33 for the remaining four cases.

Finally, for the RBF based schemes, performances are estimated by averaging 50 runs with different initializations.

3.1.3. Training

For both conventionally trained and guided NN, the Gaussian centers are initialized by means of an Euclidean distance k-means algorithm [20,23], and the output weights with random values coming from a zero mean Gaussian distribution with variance equal to 0.01.

The values of the variances of the Gaussian RBF are obtained from the corresponding k-means sample variances multiplied by a scale factor q which we select by CV, as we describe later.

When applicable, we train centers and weights of RBF by a square error based gradient algorithm with learning rates also selected by CV; as above said, variances are just selected, but not trained because centers move slightly. The number of training epochs is limited to 500, which is enough to converge according to the results of preliminary experiments; and we apply the early stopping method [2,15] to obtain the final design.

With respect to DBNN, each class is represented by a different subnet of Gaussian nodes. The procedure to initialize parameters of each subnet (centers, variances, and output weights) is the same than that used for conventional RBFNN, but considering samples from each class separately. We train centers and output weights of each subnet when classification is wrong, limiting the number of training epochs to 500 and applying the early stopping method to obtain the final design.

3.1.4. Selection of design parameters

For all the RBF based designs, including DBNN, we select by CV (checking that the best results are inside the corresponding intervals):

- The number of hidden neurons, H, as percentages $\{0.2, 0.5, 1, 1.5, 2, 2.5, 5, 10, 20, 30, 40, 50, 60, 70, 80\}$ of the number of samples for each dataset (in a progressive manner).
- The scaling factor of the elements variance, q, among $\{0.5, 1, 1.5, 2, 2.5, 3\}$.

We also use CV to select the training steps of the gradient algorithms among $\{1, 5, 10, 50\} \times 10^{-3}$.

Parameter k of the k-NN classifiers is selected by CV among initial value $k = 1$ and final value 10% of the corresponding number of training samples, with unity steps.

With respect to SVM schemes, we apply a five-fold CV and explore first the following combinations:

- Regularization parameter C: 11 equispaced values in $[10^{-2}, 10^{4}]$.
- Gaussian width σ: 11 equispaced values in $[2^{-5}\sqrt{d}, 2^{5}\sqrt{d}]$, where d is the input data dimension.

Choosing the pair of parameters (C, σ) with the maximum average accuracy and refining the exploration considering 11 values around every preselected value. The final best values of these parameters are used to construct the SVM with the whole training set.

The guide for each reduced target level scheme is the NN having the same value of H or the optimal k-NN (except for *ion*, in which we use 2-NN as the guide and not optimal 1-NN, in order to avoid identical conventional and static guided schemes), that are reasonable selections.

3.2. Simulation results

3.2.1. Classification accuracy

Table 3 presents the results of the experiments in terms of percentage of correct classifications for the test set, averaged over 50 runs for k-NN, the RBF schemes (DBNN, conventional, static, and iterative SRTL) and SVM (in the four problems with no predefined test partition). Standard deviation of the test classification rate is shown in brackets. Boldface numbers indicate the (qualitative) best result and those that are comparable.

These results show that:

(a) DBNN offers not more than episodic advantages (for *hea*, *ion*, and *son*) with respect to conventionally trained RBFNN. The

Table 3
Values of (averaged) test classification rates (standard deviation) for the 11 problems.

	DBNN	*k*-NN	SVM	Conventional RBFNN	*k*-NN guided static SRTL	RBFNN guided static SRTL	Iterative SRTL
aba	78.0 (1.0)	79.0 (0.4)	79.9	80.0 (0.3)	79.7 (0.3)	80.1 (0.4)	**80.7** (0.2)
bre	95.9 (1.1)	95.1 (1.0)	**96.6** (0.8)	96.5 (0.6)	**96.7** (0.7)	**96.7** (0.7)	**96.7** (0.6)
hea	81.3 (4.2)	57.5 (1.7)	83.3	76.8 (2.7)	75.2 (3.4)	75.4 (4.1)	**83.8** (1.4)
ima	96.2 (0.6)	96.4 (0.4)	**97.4**	97.1 (0.5)	96.9 (0.5)	96.9 (0.5)	**97.4** (0.3)
ion	94.8 (2.7)	93.0 (0.9)	**96.8**	90.0 (2.6)	92.3 (6.6)	88.8 (7.1)	**96.9** (1.2)
pim	76.1 (2.6)	74.4 (1.7)	77.3 (1.8)	77.5 (1.9)	77.4 (2.0)	77.1 (2.0)	**78.6** (1.5)
rpl	88.4 (1.8)	**90.8** (0.2)	88.0	90.6 (0.4)	90.6 (0.8)	**90.8** (0.6)	**90.7** (0.3)
son	83.3 (4.3)	**91.0** (3.6)	88.5	79.9 (2.3)	81.9 (3.5)	86.0 (3.2)	89.9 (1.6)
spa	92.5 (0.8)	87.0 (0.7)	92.8 (0.5)	**93.0** (0.6)	92.8 (0.7)	**92.9** (0.7)	**93.0** (0.4)
2no	97.6 (0.3)	**97.8** (0.2)	**97.8** (0.3)	**97.7** (0.2)	**97.7** (0.2)	**97.8** (0.2)	**97.8** (0.2)
wav	91.6 (0.6)	90.4 (0.4)	91.8	**91.9** (0.3)	91.7 (0.3)	**92.0** (0.2)	**92.0** (0.3)

Table 4
Sizes of the machines corresponding to the results in Table 3.

	DBNN	*k*-NN	SVM	Conventional RBFNN	*k*-NN guided static SRTL	RBFNN guided static SRTL	Iterative SRTL
aba	40	31	1189	100	40	40	40
bre	14	15	53	84	112	84	56
hea	32	3	40	20	32	12	32
ima	1036	3	1064	1036	1036	1036	888
ion	46	3	133	8	46	46	46
pim	8	21	252	92	8	92	8
rpl	10	17	221	100	40	100	10
son	40	1	76	58	40	40	48
spa	736	7	1092	1288	1104	1288	552
2no	14	233	2337	592	30	30	30
wav	960	111	994	960	960	960	960

Size is *k* for *k*-NN and number of nodes (RBF, kernels) for the other schemes.

same is true for static SRTL designs, that provide advantage for *ion* (*k*-NN guided form) and *son*, these advantages even being less relevant; but, on the other hand, a reduction of quality with respect to the conventional RBFNN is not as frequent as for DBNN.

(b) On the contrary, the performance improvement of the iterative SRTL machines is very clear for *aba*, *hea*, *ion*, *pim*, and *son*; and they are never worse than the other analyzed machines but for *son* (a problem in which *k*-NN provides the best result). We also remark that the performance of the iterative SRTL schemes with respect to all the rest is clearly better for problems *aba*, *hea*, and *pim*, and it is also better than the SVM performance for *rpl* and *son*.

So, we can conclude that using the iterative SRTL approach may lead to performance advantages even with respect to SVM.

3.2.2. Machines sizes

Table 4 presents the complexity (*k* for *k*-NN, number of nodes for the rest) of the machines corresponding to the results we have presented in Table 3. They can be considered as "implementation costs".

These numbers indicate that the iterative SRTL approach leads to designs with a lower (or equal) complexity than those of conventional RBFNN and static SRTL schemes, the only exceptions being problems *hea* and *son* (with respect to static SRTL schemes), two of the cases for which the iterative approach offers a clearly better performance. DBNN have also low sizes, but, as previously seen, their performance is poor.

As expected, the number of units of the RBF based designs are, in general, much lower than the number of kernels of SVM. With respect to *k*-NN sizes, we must remark that they do not indicate the full complexity of operation effort, because an additional search is needed.

3.2.3. Computational loads

In Table 5, we list the training and operation computational loads corresponding to the machines of Tables 3 and 4. These loads are measured in seconds that a workstation with a 1 GHz *UltraSPARC III* processor, 4 GB of RAM and two hard disks of 36 GB each, programming the algorithms in Matlab 6.5.0 and using the ISIS-SVM implementation (http://www.ece.umn.edu/groups/ece8591/software/svm.html) requires for designing or operating each machine. SRTL values include those of the corresponding guide schemes.

With respect to operation (classification) times, it can be seen, as expected, a direct proportion between computation time and machine sizes, except for *k*-NN methods, that present an increment which is very important, even considering that we have not used any efficient search algorithm to implement them. So, again, the iterative SRTL methods offer a general advantage, while the advantage of DBNN is compensated by their inferior performance.

Considering training times, we can conclude that, excluding *k*-NN, DBNN, and static SRTL machines guided by *k*-NN require the lower effort, followed by the iterative SRTL schemes, the static SRTL structures built by RBFNN, and SVM. This is reasonable, because DBNN only learn from wrongly classified samples and a *k*-NN is a fast guide, while SRTL algorithms do not learn in many steps, that increase along training; see Fig. 2.

SVM require, in general, more training effort than RBF based schemes, except for *hea*, *ion*, *pim*, and *son*, problems for which

Table 5
CPU times (in seconds) to train (first row) the different machines and to classify (second row) using them, for the 11 benchmark problems.

	DBNN	*k*-NN	SVM	Conventional RBFNN	*k*-NN guided static SRTL	RBFNN guided static SRTL	Iterative SRTL
aba	603	0	42 019.74	515.6	222.3	960	740
	0.85	4.92	298.63	1.15	0.85	0.85	0.85
bre	40	0	256.06	108.9	71.6	191.5	132.9
	0.004	0.166	3.018	0.012	0.015	0.012	0.01
hea	27.7	0	1.24	32.7	11.4	35.1	36.1
	0.002	0.03	0.66	0.002	0.002	0.001	0.002
ima	975.9	0	1203.83	1310	1081.3	2334.7	1671.2
	0.49	3.54	37.66	0.49	0.49	0.49	0.48
ion	33.2	0	7.95	31.3	87	54.5	51.3
	0.005	0.105	2.82	0.003	0.005	0.005	0.005
pim	94.3	0	76.71	99.6	27.9	142.3	134.6
	0.002	0.188	10.18	0.014	0.002	0.014	0.002
rpl	19.2	0	98.19	44.9	21.6	66.6	51.6
	0.002	0.101	1.81	0.004	0.003	0.004	0.002
son	20.3	0	2.29	28.6	12.8	74.7	55.4
	0.003	0.05	0.88	0.004	0.003	0.003	0.003
spa	3073.1	0	16 671.8	5698	3708.5	9684.6	7168
	0.77	22.73	205.25	0.79	0.78	0.79	0.76
2no	350.4	0	13 270.38	923.5	283.6	2351.4	1117.7
	0.30	21.2	300.16	1.27	0.32	0.32	0.32
wav	1062.1	0	11 946.16	4020.8	1435.2	6712.9	4990.2
	2.37	28.8	399.94	2.37	2.37	2.37	2.37

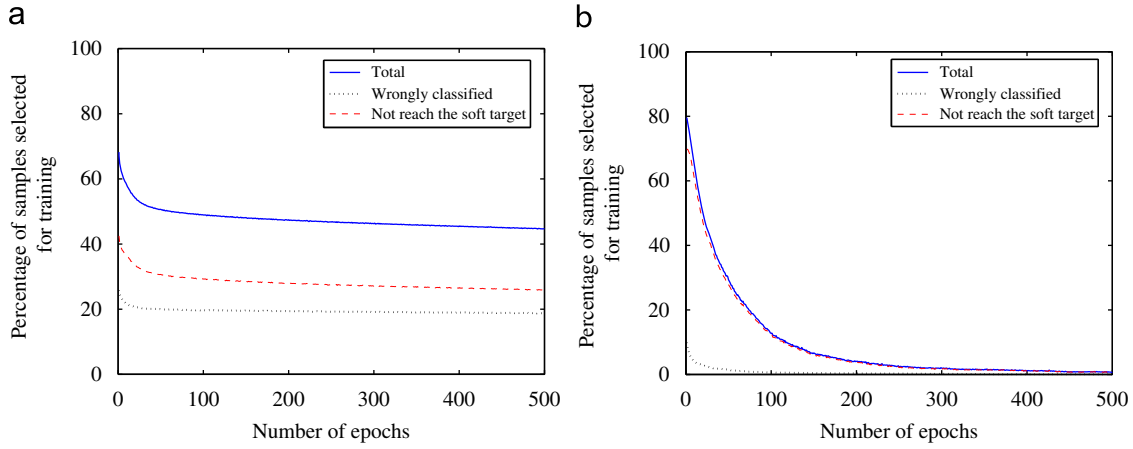


Fig. 2. Evolution of the percentage of selected training samples along training epochs in the iterative SRTL algorithms for (a) *aba* and (b) *son*. Solid lines show the total percentage of selected samples, sum of wrongly classified samples (dotted) and those of magnitude lower than the reduced target (dashed).

their size is relatively low and gradient algorithms seem to have a slow convergence. But we must remark the following facts:

(a) The ISIS-SVM package does not apply any procedure to reduce the design computational burden; we have checked that the training load can be significantly reduced by using the SVM^{light} software [17], an implementation of SVM in *C* language that includes more efficient optimization algorithms (however, a part of this advantage can be attributed to the use of *C* language); in any case, the corresponding training times remain higher, in general, than those of the iterative SRTL approach.

(b) A really fair comparison of design procedures will require to include the CV to determine the design parameters: k for k-NN; H, q, and learning rates for RBF based approaches; σ and C for SVM. For a rough estimation, this requires to multiply the above training times for the number of explored ensembles of parameters values: 15 (number of H values) $\times 6$ (number of q values) $\times 4$ (number of learning rates for the RBF centers) $\times 4$ (number of learning rates for the RBF weights) $= 1440$ for the iterative SRTL, 11 (number of C values) $\times 11$ (number of σ values) $\times 2$ (first and second explorations) $= 242$ for SVM. So, given the data in Table 5, it can be said that effective design efforts are roughly comparable. Thus, we can conclude that, given the potential advantages of the iterative SRTL methods in performance and size with respect to SVM, the effort which we need to design them is completely acceptable.

4. Conclusions

Selectively reduction of target levels consists on reducing the level of the target to be reached by the output of a classifier according to the proximity to the border of the corresponding sample estimated by the output of an auxiliary classifier. It is an interesting idea because it limits the application of the expressive power of the corresponding machine which is applied to lead outputs to unnecessarily high values.

In this paper, we have proposed an iterative version for carrying out this SRTL by means of a simple linear reduction schedule. Using RBFNN as implementation architectures, we have checked that the corresponding designs outperform (and never are worse) in several of 11 benchmark problems not only conventionally trained RBFNN, but also static versions of the same principle, and even SVM. Additionally, these iterative designs have smaller sizes and (subsequently) lower operation computational effort. Although selecting design parameters requires cross-validation and this increases the design computational effort, the whole design process is not much more intensive from the computational load point of view than that of an SVM; thus, the resulting iterative SRTL methods are attractive from a practical point of view.

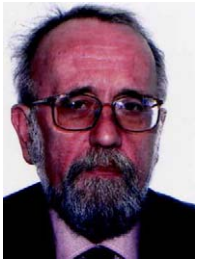
At the present time, we are extending our experiments to other machines and also to the construction of machine ensembles.

References

[1] J. Billa, A. El-Jaroudi, A method of generating objective functions for probability estimation, Eng. Appl. Artif. Intell. 9 (1996) 203–208.
[2] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, New York, NY, 1995.
[3] C.L. Blake, C.J. Merz, Repository of machine learning databases, Department of Information and Computer Science, University of California at Irvine, 1998, Available at: ⟨http://www.ics.uci.edu/ ~mlearn/MLRepository.html⟩.
[4] L. Breiman, Bias, variance and arcing classifiers, Technical Report 640, Statistics Department, University of California at Berkeley, 1996.
[5] C. Cachin, Pedagogical pattern selection strategies, Neural Networks 7 (1994) 175–181.
[6] R.K.M. Cheung, I. Lusting, A.L. Kornhauser, Relative effectiveness of training set patterns for back propagation, in: Proceedings of the IEEE International Conference on Neural Networks, San Diego, CA, vol. 1, 1992, pp. 673–678.
[7] S.H. Choi, P. Rockett, The training of neural classifiers with condensed datasets, IEEE Trans. Syst. Man Cybern. Part B 32 (2002) 202–206.
[8] J. Cid-Sueiro, J.I. Arribas, S. Urbán-Muñoz, A.R. Figueiras-Vidal, Cost functions to estimate *a posteriori* probabilities in multiclass problems, IEEE Trans. Neural Networks 10 (1999) 645–656.
[9] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (1995) 273–297.
[10] L. Franco, S.A. Cannas, Generalization and selection of examples in feed-forward neural networks, Neural Comput. 12 (2000) 2405–2426.
[11] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1997) 119–139.
[12] Y. Freund, R.E. Schapire, Discussion of paper "Arcing classifiers," by L. Breiman, Ann. Statist. 26 (1998) 824–832.
[13] V. Gómez-Verdejo, M. Ortega-Moral, J. Arenas-García, A.R. Figueiras-Vidal, Boosting by weighting critical and erroneous samples, Neurocomputing 69 (2006) 679–685.
[14] V. Gómez-Verdejo, J. Arenas-García, A.R. Figueiras-Vidal, A dynamically adjusted mixed emphasis method for building boosting ensembles, IEEE Trans. Neural Networks 19 (2008) 3–17.
[15] S. Haykin, Neural Networks: A Comprehensive Foundation, second ed., Prentice-Hall, Englewood Cliffs, NJ, 1999.
[16] P.E. Hart, The condensed nearest neighbor rule, IEEE Trans. Inf. Theory 14 (1968) 515–516.
[17] T. Joachims, Making large-scale SVM learning practical, in: B. Schölkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods—Support Vector Learning, The MIT Press, Cambridge, MA, 1999, pp. 169–184.
[18] F. Kanaya, S. Miyake, Bayes statistical behaviour and valid generalization of pattern classifying neural networks, IEEE Trans. Neural Networks 2 (1991) 471–475.
[19] S.Y. Kung, J.S. Taur, Decision-based neural networks with signal/image classification applications, IEEE Trans. Neural Networks 6 (1995) 170–181.
[20] D. Lowe, Adaptive Radial Basis Function nonlinearities, and the problem of generalisation, in: Proceedings of the First IEE International Conference on Artificial Neural Networks, London, UK, 1989, pp. 171–175.
[21] A. Lyhyaoui, M. Martínez-Ramón, I. Mora-Jiménez, M. Vázquez-Castro, J.L. Sancho-Gómez, A.R. Figueiras-Vidal, Sample selection via clustering to construct support-vector like classifiers, IEEE Trans. Neural Networks 10 (1999) 1474–1481.
[22] J.W. Miller, R. Goodman, P. Smyth, Objective functions for probability estimation, Proceedings of the 1991 International Joint Conference on Neural Networks, Seatle vol. 1 (1991) 881–886.
[23] J. Moody, C. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 1 (1989) 281–294.
[24] I. Mora-Jiménez, R. García-Marciel, A.R. Figueiras-Vidal, Improving kernel-based classifiers by guided dynamic sample selection, in: Proceedings of the 13th International Conference on Artificial Neural Networks In Engineering, St. Louis, Missouri, 2003, pp. 27–32.
[25] K.R. Müller, S. Mika, G. Rätsch, K. Tsuda, B. Schölkopf, An introduction to kernel-based learning algorithms, IEEE Trans. Neural Networks 18 (2001) 181–201.
[26] P.W. Munro, Repeat until bored: a pattern selection strategy, in: J.E. Moody, et al. (Eds.), Advances in Neural Information Processing Systems, 4, Morgan Kaufmann, San Mateo, CA, 1992, pp. 1001–1008.
[27] J. Park, I.W. Sandberg, Universal approximation using radial basis function networks, Neural Comput. 3 (1991) 246–257.
[28] B. Pearlmutter, J.B. Hampshire, Equivalence proof for multilayer perceptron classifiers and the Bayesian discriminant function, in: D.S. Touretzky, J.L. Elman, T.J. Sejnowski, G.E. Hinton (Eds.), Proceedings of the 1990 Connectionist Models Summer School, Morgan Kauffmann, San Mateo, CA, 1990, pp. 159–172.
[29] M. Plutowski, H. White, Selecting concise training sets from clean data, IEEE Trans. Neural Networks 4 (1993) 305–318.
[30] R. Reed, S. Oh, R.J. Marks, II, Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter, IEEE Trans. Neural Networks 6 (1995) 529–538.
[31] M.D. Richard, R.P. Lippmann, Neural Network classifier estimates Bayesian a posteriori probabilities, Neural Comput. 3 (1991) 461–483.
[32] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, UK, 1996.
[33] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley, B.W. Suter, The multilayer perceptron as an approximation to a Bayes optimal discriminant function, IEEE Trans. Neural Networks 1 (1990) 296–298.
[34] A. Ruiz, P.E. López-de-Teruel, Nonlinear kernel-based statistical pattern analysis, IEEE Trans. Neural Networks 12 (2001) 16–32.
[35] R.E. Schapire, Y. Singer, Improved boosting algorithms using confidence-rated predictions, Mach. Learn. 37 (1999) 297–336.
[36] J. Sklansky, L. Michelotti, Locally trained piecewise linear classifiers, IEEE Trans. Pattern Anal. Mach. Intell. 2 (1980) 101–111.
[37] E.M. Strand, W.T. Jones, An active pattern set strategy for enhancing generalization while improving backpropagation training efficiency, in: Proceedings of the International Joint Conference on Neural Networks, Baltimore, MD, vol. 1, 1992, pp. 830–834.

[38] V. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.
[39] M. Wann, T. Hediger, N.N. Greenbaun, The influence of training sets on generalization in feed-forward neural networks, in: Proceedings of the International Joint Conference on Neural Networks, San Diego, California, vol. 3, 1990, pp. 137–142.
[40] B.T. Zhang, Accelerated learning by active example selection, Int. J. Neural Syst. 5 (1994) 67–75.

Inmaculada Mora-Jiménez received the Telecommunication Engineering degree from Universidad Politécnica de Valencia, Spain, in 1998, and the Ph.D. degree from Universidad Carlos III de Madrid, Spain, in 2004. Currently, she is an Associate Professor in the Department of Signal Theory and Communications at Universidad Rey Juan Carlos, Spain. Her main research interests include statistical learning theory, neural networks, and their applications to image processing, bioengineering, and communications.



Aníbal R. Figueiras-Vidal received the Telecommunication Engineering degree (honors) from Universidad Politécnica de Madrid, Spain, in 1973, and the Doctor degree (honors) from Universidad Politécnica de Barcelona, Spain, in 1976. He is a Professor of Signal Theory and Communications at Universidad Carlos III, Madrid. His research interests are digital signal processing, digital communications, neural networks, and learning theory. He has (co)authored more than 300 journal and conference papers in these areas. Prof. Figueiras-Vidal received an ''Honoris Causa'' doctorate degree from Universidad de Vigo, Spain, in 1999. He is currently the President of the Royal Academy of Engineering of Spain.