Fast Evaluation of Neural Networks via Confidence Rating ⋆

Jerónimo Arenas-García *, Vanessa Gómez-Verdejo

Aníbal R. Figueiras-Vidal

Department of Signal Theory and Communications

Universidad Carlos III de Madrid

28911 Leganés-Madrid, Spain

Abstract

Neural Networks have become very useful tools for input-output knowledge discovery. However, some of the most powerful schemes require very complex machines, and thus a large amount of calculation. This paper presents a general technique to reduce the computational burden associated to the operational phase of most neural networks that calculate their output as a weighted sum of terms, which comprises a wide variety of schemes, such as Multi-Net or Radial Basis Function networks. Basically, the idea consists on sequentially evaluating the sum terms, using a series of thresholds which are associated to the confidence that a partial output will coincide with the overall network classification criterion. Furthermore, we design some procedures for conveniently sorting out the network units, so that the most important ones are evaluated first. The possibilities of this strategy are illustrated with some experiments on a benchmark of binary classification problems, using RealAdaboost and RBF networks, that show that important computational savings can be achieved without significant degradation in terms of recognition accuracy.

Preprint submitted to Neurocomputing *28 February 2006*

1 Introduction

Artificial Neural Networks (ANNs) are nowadays widely used in classification and regression problems because of their extraordinary abilities for input-output knowledge discovery [1,2]. During the training phase, ANNs automatically adjust their parameters by means of a set of labeled patterns. Later, in the operational phase, the network is used to predict the output corresponding to new unseen patterns, or the class they belong to. Some of the most classical techniques, such as Multi-Layer Perceptrons (MLPs) and Radial Basis Functions Networks (RBFNs) [1] are known to have universal approximation capabilities.

The performance of traditional neural networks can be improved even further by means of Multi-Net (M-N) systems [3] that combine the output of several base learners. There are many approaches to construct ANN ensembles, adopting the name that was introduced by Hasen and Salomon [4]. Classifications of these methods from different points of view can be found in [5]. Among all these procedures, boosting schemes [6,7], and in particular Real Adaboost (RA) [8], have demonstrated excellent performance.

Based on statistical learning theory [9], Support Vector Machines (SVMs) [10] have recently gained popularity, achieving excellent results, mainly in binary classification problems. Although the recognition capabilities of M-N systems and SVMs is out of discussion, the computational complexity of the resulting machines can be very high if the number of base learners or support vectors,

⋆ This work has been partly supported by CICYT grant TIC2002-03713.

* Corresponding author.
Email address: jarenas@tsc.uc3m.es (Jerónimo Arenas-García).

respectively, is very large.

In this paper, we present a method for fast evaluation of neural networks that can lead to a significant saving during the operational phase of ANNs. One of the most important advantages of the method is that it can be used with any kind of network that calculates its output as a linear weighted sum of terms, what comprises a wide variety of neural systems.

The organization of the paper is as follows: next section is dedicated to the introduction of the fast classification method, and to the discussion of several implementation details. In Section 3 we include an extensive number of experiments to illustrate the benefits and applicability of the strategy when using both RA and RBFNs. Finally, we devote the last section of the paper to conclude our work and enlighten some lines for further research.

2 Fast Evaluation of Neural Networks

In this paper we consider the binary classification problem in which we are given a set of labeled data $\{\mathbf{x}_i, t_i\}_{i=1}^N$, with $\mathbf{x}_i \in \Re^d$ being the i-th input pattern and $t_i \in \{-1, 1\}$ its associated target. We will only pay attention to neural networks that predict the output using a weighted linear sum of M different terms:

$$y(\mathbf{x}) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m o_m(\mathbf{x}) + b\right], \tag{1}$$

where α_m are the weights assigned to the different network units, $o_m(\mathbf{x})$, and b is a bias term. Note that, to keep our formulation as general as possible, we are not assuming any predetermined implementation for the network units. In this way, $o_m(\mathbf{x})$ could be the output of a *kernel* function [11], as well as the

output of a base learner of a M-N system.

In this paper, we propose a very simple procedure to speed up the evaluation of (1) and so the operational phase of the network, what can be specially advantageous when the number of units that are combined is large. Concretely, we propose to sequentially evaluate the different terms in (1), achieving the following partial sums:

$$y_k(\mathbf{x}) = \sum_{m=1}^{k} \alpha_m o_m(\mathbf{x}) + b. \tag{2}$$

Hopefully, if the absolute value of $y_k(\mathbf{x})$ is high enough, the current sign of $y_k(\mathbf{x})$ will coincide with $y(\mathbf{x})$, making it possible to avoid the evaluation of the restating $M - k$ network units. To be more precise, if

$$y_k(\mathbf{x}) > \eta_k^+ \qquad \text{or} \qquad y_k(\mathbf{x}) < \eta_k^-,$$

where $\{\eta_k^+, \eta_k^-\}_{k=1}^{M-1}$ are some thresholds to be fixed during the training of the network, we will classify $\mathbf{x}$ according to the present sign of $y_k(\mathbf{x})$. In the following, we will refer to this process as fast or partial classification, and to $y_k(\mathbf{x})$ as a partial classifier. On the contrary, if

$$\eta_k^- < y_k(\mathbf{x}) < \eta_k^+$$

we assume that the confidence about $y(\mathbf{x})$ being equal to the sign of $y_k(\mathbf{x})$ is insufficient, making it necessary to proceed with the evaluation of the $(k+1)$-th unit. Since a fast classification with the k-th partial classifier makes unnecessary to compute the outputs of units $o_m(\mathbf{x}), m = k+1, \ldots, M$, this procedure can lead to important computational savings during the operational phase of the network.

In most classification tasks we find both difficult and easy patterns, i.e., pat-

terns which are close to the border of classification and others which lie far away from it. With the previous procedure we expect to classify the easiest patterns in an early stage, while difficult patterns will still require the evaluation of most of the network units. So, in the average, we expect to reduce the number of units that need to be evaluated without affecting the performance of the network.

Obviously, for the correct application of the method it is necessary to conveniently select the thresholds $\{\eta_k^+, \eta_k^-\}_{k=1}^{M-1}$ during the training phase. Furthermore, to get the maximum reduction in the average number of units evaluated it is important to sort out the units in such a way that those with a higher influence in the overall classification are evaluated first. Finally, bias b is appropriate only when classifying according to (1), but we could perfectly find better biases for the partial classifiers $y_k(\mathbf{x})$. The next three subsections address each of these problems, and explain how they can be solved during the training phase of the network with little extra computational effort.

2.1 Setting the Classification Thresholds

In first term, we propose to set up the positive and negative thresholds, $\{\eta_k^+\}_{k=1}^{M-1}$ and $\{\eta_k^-\}_{k=1}^{M-1}$ respectively, to guarantee that the original network criterion is preserved and no additional errors are provoked by the fast classification method. It is useful to distinguish two different cases depending on the range of $o_m(\mathbf{x})$:

i) $o_m(\mathbf{x}) : \Re^d \rightarrow [-1, 1]$: In this case the difference between the overall network

and its k-th partial output is given by

$$\Delta y_k(\mathbf{x}) = y_M(\mathbf{x}) - y_k(\mathbf{x}) = \sum_{m=k+1}^{M} \alpha_m o_m(\mathbf{x}) \tag{3}$$

that can be bounded by

$$-\sum_{m=k+1}^{M} |\alpha_m| < \Delta y_k(\mathbf{x}) < \sum_{m=k+1}^{M} |\alpha_m|$$

Thus, choosing

$$\eta_k^+ = \sum_{m=k+1}^{M} |\alpha_m|; \qquad \eta_k^- = -\eta_k^+ \tag{4}$$

satisfies our requirement that the overall network classification is preserved.

ii) $o_m(\mathbf{x}) : \Re^d \rightarrow [0,1]$: Using similar arguments to those in the previous case we get

$$\sum_{\substack{m=k+1 \\ \alpha_m<0}}^{M} \alpha_m < \Delta y_k(\mathbf{x}) < \sum_{\substack{m=k+1 \\ \alpha_m>0}}^{M} \alpha_m$$

so that we should set

$$\eta_k^+ = -\sum_{\substack{m=k+1 \\ \alpha_m<0}}^{M} \alpha_m; \qquad \eta_k^- = -\sum_{\substack{m=k+1 \\ \alpha_m>0}}^{M} \alpha_m \tag{5}$$

to guarantee that the classification criterion is not affected.

Obviously, (4) and (5) are worst-case choices, consequently providing limited computational savings. It is immediate to relax these selections by introducing an additional parameter $\beta \in [0,1]$ that allows to redefine the thresholds as

$$\eta_k^{+\prime} = \beta\eta_k^+; \qquad \eta_k^{-\prime} = \beta\eta_k^-$$

Parameter β is related to the confidence that a fast decision on $\mathbf{x}$ will respect the criterion of the overall classifier. As we have explained, if $\beta = 1$ we are 100% sure that no additional errors (or hits) are introduced by the fast classification process. On the contrary, when $\beta = 0$ only the first unit is considered. Values

of β between 0 and 1 correspond to situations where an intermediate degree of certainty is required.

2.2 Sorting out the Network Units

As explained before, it is not only the selection of the thresholds, but also the order of the network units which plays an essential role in the performance of the fast classification method. As a general rule, to achieve a maximum computational saving it is convenient to place the most relevant units in the first positions.

By construction, certain networks directly give the units in decreasing order of importance. This is true, for instance, for incremental schemes that grow with the objective of refining the border of classification at each step, such as RealAdaboost (RA) [8], the Cascade-Correlation (CC) algorithm [12], or the Growing Support Vector Classifier of [13], where this fact was already used to speed up the classification phase.

In general, however, it will become necessary to use some measure of relevance to sort out the units that compose the network. Although it is possible to think of many other alternatives, an easy way to measure the relevance of the units would be to use their *average activation* in the training set that we define as

$$\gamma_m = \sum_{i=1}^{N} |\alpha_m o_m(\mathbf{x}_i)| \tag{6}$$

In this way, the units would be sorted out according to decreasing γ_m.

The above measure of relevance has the advantage that it can be used with any network that implements a function of the form (1). On the other hand, one

could expect that a better performance could be obtained if designing *ad hoc* criteria for each particular network. For instance, in the experiments section we will apply our fast classification method to the RBFN implementation of [14]. These networks are trained to minimize the cost function

$$C = \frac{1}{2}\sum_{i=1}^{N}\left[t_i - y_M(\mathbf{x}_i)\right]^2 \tag{7}$$

where $y_M(\mathbf{x}_i)$ is the output of the network before the sign activation, namely

$$y_M(\mathbf{x}_i) = \sum_{m=1}^{M}\alpha_m o_m(\mathbf{x}_i) \tag{8}$$

To estimate the influence of each unit $o_m(\mathbf{x})$ in the overall quadratic error incurred by the network, we will first derive (7) with respect to the outputs of that unit for each training pattern, i.e.,

$$\nabla_{o_m} C = \begin{bmatrix} \dfrac{\partial C}{\partial o_m(\mathbf{x}_1)} \\ \vdots \\ \dfrac{\partial C}{\partial o_m(\mathbf{x}_N)} \end{bmatrix} = -\alpha_m \begin{bmatrix} t_1 - y_M(\mathbf{x}_1) \\ \vdots \\ t_N - y_M(\mathbf{x}_N) \end{bmatrix} \tag{9}$$

Note that this gradient can be considered as a measure of the efectiveness of unit $o_m(\mathbf{x})$ to decrease (7). Now, taking the square norm of $\nabla_{o_m} C$ gives

$$\|\nabla_{o_m} C\|_2^2 = \alpha_m^2 \sum_{i=1}^{N}\left[t_i - y_M(\mathbf{x}_i)\right]^2 \tag{10}$$

Finally, considering that the sum in the above expression is the same for all units, we can conclude that the influence of each unit $o_m(\mathbf{x})$ in the cost function is directly related to the square of its associated weight, α_m, thus justifying to sort out the units of the RBFN according to a decreasing value of $|\alpha_m|$.

For illustrating the benefits of using an appropriate order of the network units, we have considered two examples in which the fast evaluation method was ap-

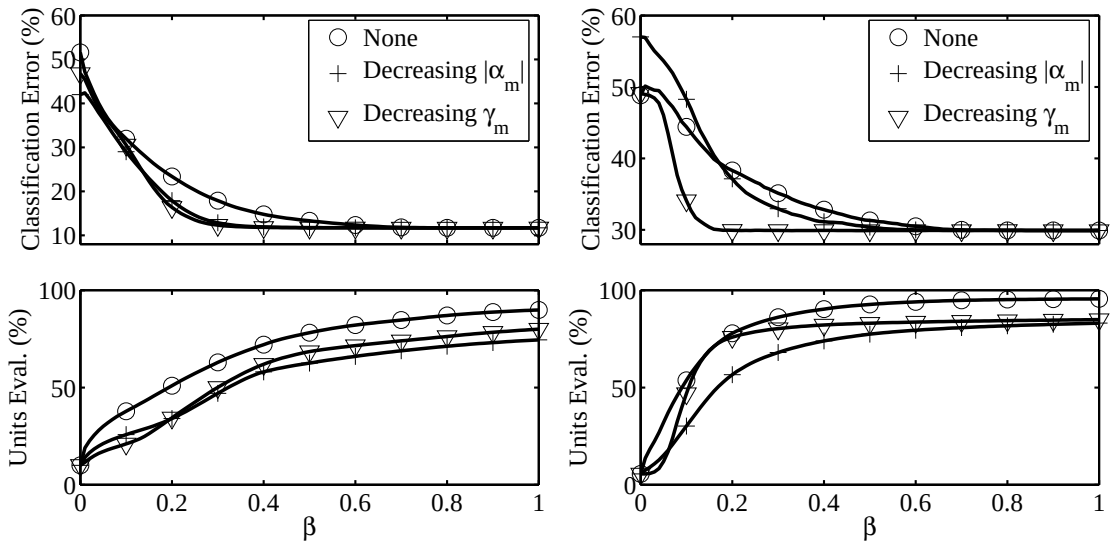

Fig. 1. Classification Error and average percentage of units evaluated during the test phase for a RBFN classifier using different strategies for sorting out the network units. (a) *Kwok* dataset. (b) *Contraceptive* dataset.

plied to RBFNs trained on the *kwok* and the *contraceptive* datasets whose main characteristics will be described in the next section. In Fig. 1 we have represented the classification error and the average number of units that are evaluated for different values of β when using different orders for the network units. We can see that using an appropriate order is very important if one wants to achieve significant computational savings without affecting the recognition performance of the original classifier (which is obtained with $\beta = 1$).

2.3 Using different biases for early classification

According to our fast classification strategy, when a pattern is classified using only the k first units of the network, we take the sign of $y_k(\mathbf{x})$ as given by

(2). However, it seems likely that the bias b calculated for the overall classifier does not work optimally for all partial classifiers. Consequently, to get a better performance we will use a different bias for each partial classifier, i.e., we propose to classify pattern $\mathbf{x}$ according to the sign of

$$\tilde{y}_k(\mathbf{x}) = \sum_{m=1}^{k} \alpha_m o_m(\mathbf{x}) + b_k \tag{11}$$

where the new biases $\{b_k\}_{k=1}^{M-1}$ are obtained to minimize the overall square error incurred by each of the partial classifiers over the whole training dataset, i.e,

$$b_k = \frac{1}{N} \sum_{i=1}^{N} t_i - y_k(\mathbf{x}_i) \tag{12}$$

Although there exist other possibilities to calculate the above biases (for instance, we could minimize the number of errors), (12) is expected to work well for most distributions of $y_k(\mathbf{x}_i)$.

To see why using different thresholds for the successive partial outputs is important, Fig. 2 depicts the classification error rate that is achieved by a RBFN network trained on the *contraceptive* dataset, both when the original bias b is kept unmodified, and when different biases b_k are used for the partial classifications. For this example, the units were sorted out according to decreasing $|\alpha_m|$; however, similar curves are also obtained for other orders and problems. It is important to remark that using different biases has no effect on the number of units evaluated for fixed β, but it can lead to significant improvements in terms of classification error.

In Table 1 we summarize our method for fast evaluation of neural networks.

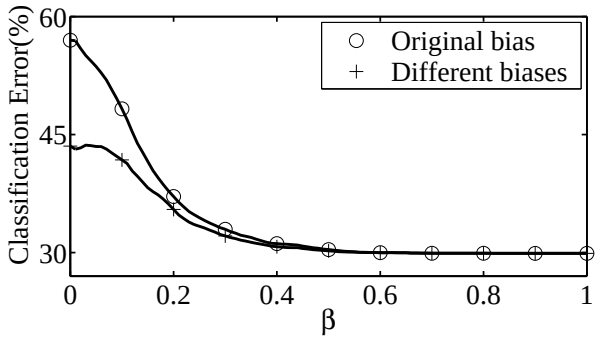

Fig. 2. Advantages of using different biases for the partial networks. The classification error is represented for a RBFN classifier trained on the *contraceptive* dataset, both when using the original bias and modified biases for the intermediate outputs.

Although steps 3 to 5 require some additional computational effort during the training phase, it can be seen that this overload is not very important given the fact that the outputs of the units, $o_m(\mathbf{x}_i)$, are already calculated during the training phase.

3 Experiments

To illustrate the validity of the fast classification strategy, we have carried out a number of experiments on a benchmark of seven binary classification tasks. A summary of the main characteristics of each problem, including the number of training and test patterns in each class and the dimension of the input space, is given in Table 2. *Kwok* is a synthetic problem that was introduced in [15]; the rest are datasets of real data taken from the Machine Learning Repository of the University of California Irvine [16]. When no predefined train/test partition was provided, we used a 10 fold cross-validation strategy employing 60% data for the training stage and the remaining 40% data for

1.- Inputs: $\{\mathbf{x}_i, t_i\}_{i=1}^{N}$, β

2.- Train network $\rightarrow$ α_m, b, $o_m(\cdot)$

3.- Sort out the units in order of decreasing relevance

4.- Calculate thresholds according to (4) or (5) $\rightarrow$ $\{\eta_k^{+'}, \eta_k^{-'}\}_{k=1}^{M-1}$

5.- Calculate biases according to (12) $\rightarrow$ $\{b_k\}_{k=1}^{M-1}$

6.- Operational phase:

$y_0(\mathbf{x}) = b$

Loop $k = 1 : M-1$

Calculate $o_k(\mathbf{x})$

$y_k(\mathbf{x}) = y_{k-1}(\mathbf{x}) + \alpha_k o_k(\mathbf{x})$

If ($y_k(\mathbf{x}) > \eta_k^{+'}$ or $y_k(\mathbf{x}) < \eta_k^{-'}$)

$\tilde{y}_k(\mathbf{x}) = y_k - b + b_k$

Classify $\mathbf{x} \rightarrow y(\mathbf{x}) = \text{sign}\,[\tilde{y}_k(\mathbf{x})]$

Go to 6 and proceed with next pattern

End

End

$y_M(\mathbf{x}) = y_{M-1}(\mathbf{x}) + o_M(\mathbf{x})$

Classify $\mathbf{x} \rightarrow y(\mathbf{x}) = \text{sign}\,[y_M(\mathbf{x})]$

Go to 6 and proceed with next pattern

Table 1
Summary of the method for Fast Evaluation of neural networks.

testing the classifiers performance. In the next two subsections we present results when using RealAdaboost ensembles of MLP classifiers, and for RBF networks. In both cases, all results were averaged over 50 independent designs (5 experiments for each fold when using 10 fold cross-validation).

	# train	# test	dim
kwok	300/200	6120/4080	2
tictactoe	199/376	133/250	9
contraceptive	506/377	338/252	9
image	821/1027	169/293	18
spam	1088/1673	725/1115	57
phoneme	952/2291	634/1527	5
waveform	2694/1306	659/341	21

Table 2
Benchmark dataset description.

3.1 Fast Classification with RealAdaboost schemes

RealAdaboost (RA) [8] is a boosting algorithm that works by sequentially adding learners to an ensemble, progressively paying attention to the hardest to classify patterns. For this set of experiments we have used MLP networks with one single hidden layer as the elements of RA, using the basic backpropagation algorithm for their training. The number of learners that build up the ensemble (T) is selected by stopping its growing when the mean value of the output weights becomes lower than 0.01. The results that we show correspond to the number of MLP hidden neurons (NH) that achieves the best ensemble performance.

It is easy to identify the RA classification function with Eq. (1) by simply setting $M = T$, and letting $o_m(\mathbf{x})$ and α_m be the outputs of each base learner and its associated weight. As explained in the beginning of Subsection 2.2, for RA it is not necessary to modify the order of the units. Furthermore, when building the RA ensemble all partial classifiers are optimized under the

constraint that no bias is added at the output. Consequently, in this case we will simply set $b_k = b = 0$.

In Fig. 3 we have depicted the influence of β on the test classification error (CE) and on the average number of learners evaluated (T_{av}) for all the problems in our benchmark. To make more clear the computational saving, T_{av} is represented in the figure as a percentage of the total number of learners in each network. As expected, β imposes a compromise between these two quantities, but there exists a wide range for which no performance degradation is observed and significant complexity reductions can be achieved.

Results for the complete benchmark are reported in Table 3. The original CE of RA is given under the '$\beta = 1$' column. We can see that, even for this very conservative case, important reductions in the computational load can be achieved. It is important to remark that this gain is obtained at a negligible cost during the training phase. Finally, computational savings of about 90% are not rare for lower values of β, keeping in all cases a very similar CE to that of the overall RA scheme.

3.2 Fast Classification with RBF Networks

RBFNs are well-known classifiers that obtain their output as a weighted linear combination of *radial basis functions*. We will only consider here the case of Gaussian kernels:

$$o_m(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_m\|^2}{2\sigma_m^2}\right)$$

We used 2%, 5%, and 10% of the training data as the number of kernels in the RBFN, keeping the setting which offered a best result on a validation

	NH	T	CE (%) $\beta = 1$	T_{av} $\beta = 1$	CE (%) $\beta = .5$	T_{av} $\beta = .5$	CE (%) $\beta = .3$	T_{av} $\beta = .3$
kwok	9	10	11.63	3.56 (−64.4%)	11.63	2.05 (−79.5%)	11.63	1.55 (−84.5%)
tictactoe	8	100	2.51	71.1 (−28.9%)	2.53	55.3 (−44.7%)	2.58	42.27 (−57.7%)
contraceptive	4	10	28.75	5.95 (−40.5%)	28.75	4.3 (−57%)	28.77	3.26 (−67.4%)
image	9	40	2.24	4.88 (−87.8%)	2.26	2.93 (−92.7%)	2.37	2.18 (−94.6%)
spam	5	50	5.68	9.48 (−81%)	5.69	5.13 (−89.7%)	5.73	3.09 (−93.8%)
phoneme	28	10	13.72	3.68 (−63.2%)	13.72	2.22 (−77.8%)	13.74	1.67 (−83.3%)
waveform	2	20	7.89	6.73 (−66.3%)	7.9	3.69 (−81.6%)	7.89	2.23 (−88.8%)

Table 3
RealAdaboost test classification error (CE) and average number of learners evaluation (T_{av}) for the seven problems in the benchmark and three values of β. Computational savings with respect to the complete evaluation of the RA machine are given inside parentheses.

set consisting of 20 % of the training data. We have used the excellent implementation of RBFNs by G. Rätsch (see, for instance, [14]), which has the advantage of automatically adjusting the position and width of the kernels by means of a gradient descent scheme.

Results are reported in Table 4 in terms of test CE and average number of Kernel Evaluations per Pattern (KEPP) during the test phase [13]. Again, the CE is kept under control even for $\beta = 0.3$, but now the computational saving is not as impressive as in the RA case.

	KEPP	CE (%) $\beta = 1$	KEPP $\beta = 1$	CE (%) $\beta = .5$	KEPP $\beta = .5$	CE (%) $\beta = .3$	KEPP $\beta = .3$
kwok	10	11.68	7.45 (−25.5%)	11.75	6.26 (−37.4%)	12.85	4.7 (−53%)
tictactoe	29	0.39	24.45 (−15.7%)	0.39	22.55 (−22.2%)	0.39	20.57 (−29.1%)
contraceptive	18	29.9	14.96 (−16.9%)	30.21	13.94 (−22.5%)	32.11	12.24 (−32%)
image	185	3.16	151.7 (−18%)	3.16	139.98 (−24.3%)	3.16	129.83 (−29.8%)
spam	277	6.05	246.67 (−10.9%)	6.05	235.24 (−15.1%)	6.05	224.17 (−19.1%)
phoneme	325	12.06	284.92 (−12.3%)	12.06	268.67 (−17.3%)	12.06	253.17 (−22.1%)
waveform	80	8.75	61.76 (−22.8%)	8.75	56.96 (−28.8%)	8.87	52.06 (−34.9%)

Table 4
Test CE and Kernel Evaluations per Pattern (KEPP) when using RBFNs. The relative reduction on the KEPP with respect to the number of units in the network (second column) is also shown inside parentheses.

3.3 On the selection of parameter β

In the two previous subsections we showed that the fast evaluation method offers a straightforward way to achieve computational savings during the operation of neural networks. However, as it can be easily seen from Fig. 3, selection of parameter β imposes a tradeoff between the computational saving and the final CE rate of the classifier. So, to improve the applicability of the fast classification strategy it is important to provide some method for the automatic selection of β.

An obvious solution to the problem of β selection is using cross-validation, but this would require using a smaller data set for training the network. Alternatively, we propose a very simple procedure consisting on applying the fast

classification strategy to the training set, calculating the training classification error and average number of unit evaluations for different values of β. Then, we select the smallest β that satisfies some constrain, either on the maximum allowable performance degradation, or on the maximum computational load that can be accepted.

As an example, we applied this strategy to the RBF networks of Section 3.2. We explored different values for β in the range [0,1], with a step of 0.01, choosing the smallest value (β_0) that guaranteed that the error in the training dataset did not increase more than 1%. The results in Table 5 show that this very simple procedure offers a reasonably good performance. We can see that, in all problems but *kwok*, the CE increment is kept well under 1%. Regarding the number of kernel evaluations per pattern, we can see that savings around 30% are not rare. Furthermore, in two of the problems which require larger networks, *image* and *phoneme*, the average computational load is reduced to around half of the original.

4 Conclusions

In this paper we have presented a simple procedure to alleviate the computational burden required by powerful neural network classifiers. The scheme works by sequentially comparing the partial outputs of the net to a positive and a negative thresholds, taking decisions as soon as a desired degree of certainty is reached.

Procedures to set up thresholds during the training of the network have also been presented, as well as some methods to appropriately order the network

	KEPP	CE (%)	β_0	CE (%) β_0	KEPP β_0
kwok	10	11.68	0.38	11.97 (+2.5%)	5.65 (−43.5%)
tictactoe	29	0.39	0.36	0.39 (=)	21.36 (−26.34%)
contraceptive	18	29.9	0.52	30.11 (+0.7%)	12.96 (−28%)
image	185	3.16	0.06	3.16 (=)	88.91 (−51.9%)
spam	277	6.05	0.11	6.03 (−0.3%)	192 (−30.7%)
phoneme	325	12.06	0.05	12.11 (+0.4%)	179.9 (−44.7%)
waveform	80	8.75	0.37	8.77 (+0.2%)	54.2 (−32.2%)

Table 5
Test CE and Kernel Evaluations per Pattern (KEPP) for RBF networks with the fast classification strategy. Original CE and KEPP are displayed in the two first columns, while the two last columns stand for the CE and KEPP (and relative variations) when using β_0 for setting the thresholds. Parameters β_0 have been automatically selected as described in the text.

units in decreasing order of relevance, so that a maximum computational saving can be achieved. These procedures need to be carried out just once after the training phase, and they require little extra computational effort.

The proposed scheme has shown to be specially well suited for Adaboost type classifiers, in which the constituent units are by construction given in order of relevance. Other ANN schemes can also obtain significant reductions in computational load, as we have shown with our experiments with Radial Basis Function networks. Logical lines for future research consist on the application of the fast evaluation method to other kinds of classifiers, as well as the extension of these ideas to speed up the training phase of neural networks.

References

[1] C. M. Bishop: Neural Networks for Pattern Recognition. Oxford University Press, New York, NY (1995).

[2] O. D. Duda, P. E. Hart and D. G. Stork: Pattern Classification. Wiley, New York, NY (2001).

[3] A. J. C. Sharkey (ed.): Combining Artificial Neural Nets. Ensemble and Modular Multi-Net Systems. Perspectives in Neural Computing. Springer-Verlag, London, UK (1999).

[4] L. K. Hansen and P. Salamon: Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence, **12** (1990) 993–1001.

[5] S. Haykin: Neural Networks, A Comprehensive Foundation (2nd ed.). Upper Saddle River, NJ, Pretince-Hall, (1999).

[6] Y. Freud and R.E. Schapire: Experiments with a New Boosting Algorithm. Proc. 13th Intl. Conf. Machine Learning, Bari, Italy (1996) 148–156.

[7] Y. Freud and R.E. Schapire: Game Theory; On-line Prediction, and Boosting. Proc. 9th Annual Conference on Computational Learning Theory, Desenzano del Garda, Italy (1996) 325–332.

[8] R. E. Schapire and Y. Singer: Improved boosting algorithms using confidence-rated predictions. Machine Learning, **37**(3) (1999) 297–336.

[9] V. N. Vapnik: The Nature of Statistical Learning Theory. New York, NY, Springer-Verlag, (1995).

[10] C. J. C. Burges: A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, **2** (1998) 121–167.

[11] B. Schölkopf and A. J. Smola: Learning with Kernels. MIT Press, Cambridge, MA (1998).

[12] S. E. Falhman and C. Lebiere: The cascade-correlation learning architecture. In DS Touretzky, editor, Advances in Neural Information Processing Systems 2, San Mateo, DA (1990) 524–532.

[13] E. Parrado-Hernández, I. Mora-Jiménez, J. Arenas-García, A. R. Figueiras-Vidal and A. Navia-Vázquez: Growing Support Vectors with Controlled Complexity. Pattern Recognition **36**(7) (2003) 1479–1488.

[14] G. Raetsch: Robust Boosting vis Convex Optimization. PhD thesis (Appendix B.6), Postdam, Germany, October 2001. Available on-line at *http://www2.fml.tuebingen.mpg.de/raetsch/Members/raetsch/raetschpub/Rae01/bibliography_entry_view.*

[15] J. T. Kwok: Moderating the Output of Support Vector Machine Classifiers. IEEE Trans. on Neural Networks, **10**(5) (1999) 1018–1031.

[16] C. L. Blake and and C. J. Merz: UCI Repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html. University of California, Irvine, Dept. of Information and Computer Sciences (1998).

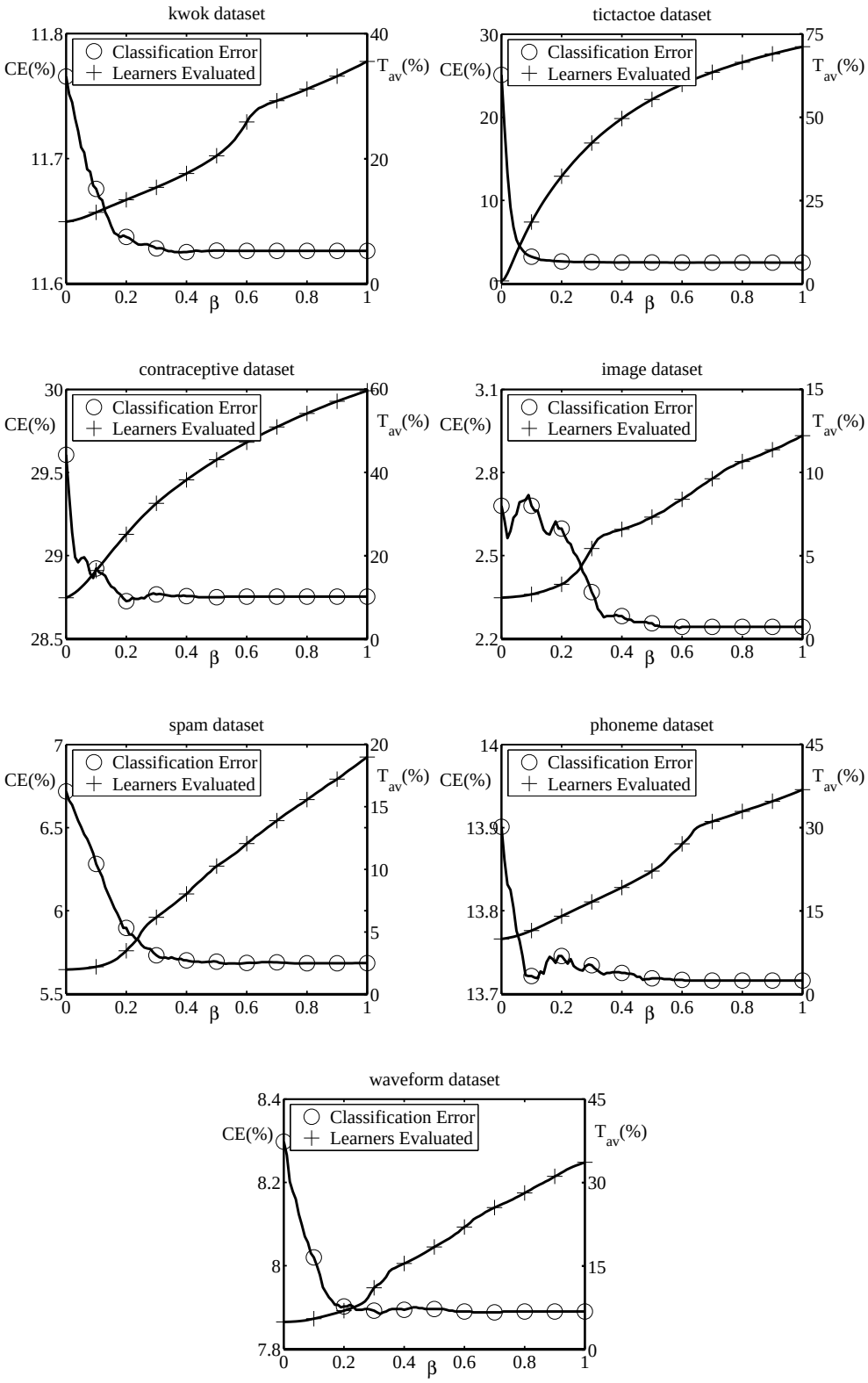

Fig. 3. Fast Classification with RealAdaboost ensembles for different values of the confidence rate parameter β. Each figure shows the evolution with β of the classification error (CE), as well as that of the average number of learners that need to be evaluated, T_{av}, expressed as a percentage of the total number of learners in the network (T).

List of Figures

List of Tables