LETTER Communicated by Nicol Schraudolph

A Global Optimum Approach for One-Layer Neural Networks

Enrique Castillo
castie@unican.es
Department of Applied Mathematics and Computational Sciences, University of Cantabria and University of Castilla-La Mancha, 39005 Santander, Spain

Oscar Fontenla-Romero
oscarfon@mail2.udc.es
Bertha Guijarro-Berdiñas
cibertha@udc.es
Amparo Alonso-Betanzos
ciamparo@udc.es
Department of Computer Science, Faculty of Informatics, University of A Coruña, 15071 A Coruña, Spain

The article presents a method for learning the weights in one-layer feed-forward neural networks minimizing either the sum of squared errors or the maximum absolute error, measured in the input scale. This leads to the existence of a global optimum that can be easily obtained solving linear systems of equations or linear programming problems, using much less computational power than the one associated with the standard methods. Another version of the method allows computing a large set of estimates for the weights, providing robust, mean or median, estimates for them, and the associated standard errors, which give a good measure for the quality of the fit. Later, the standard one-layer neural network algorithms are improved by learning the neural functions instead of assuming them known. A set of examples of applications is used to illustrate the methods. Finally, a comparison with other high-performance learning algorithms shows that the proposed methods are at least 10 times faster than the fastest standard algorithm used in the comparison.

1 Introduction

Single-layer networks were widely studied in the 1960s, and their history has been reviewed in several places (Widrow & Lehr, 1990). For a one-layer linear network, the weight values minimizing the sum-of-squares er-

 © 2002 Massachusetts Institute of Technology

ror function can be found in terms of the pseudo-inverse of a matrix (Bishop, 1995). Nevertheless, if a nonlinear activation function, such as a sigmoid or hyperbolic tangent, is used or if a different error function is considered, a closed-form solution is no longer possible. However, if the activation function is differentiable, as is the case for the mentioned functions, the derivatives of the error function with respect to the weight parameters can be easily evaluated. These derivatives can then be used in a variety of gradient-based optimization algorithms for efficiently finding the minimum of the error function. The performance of these algorithms depends on several design choices, such as the selection of the step size and momentum terms, most of which are currently made by rules of thumb and trial and error. In addition to this problem, it is possible for the one-layer neural network to get stuck in a local minima, as shown by Brady, Raghavan, and Slawny (1989), Sontag and Sussmann (1989), and Budinich and Milotti (1992).

The existence of this problem was mathematically demonstrated by Sontag and Sussmann (1989), who provided a neural network example without hidden layers and with a sigmoid transfer function, for which the sum of squared errors has a local minimum that is not a global minimum. They observed that the existence of local minima is due to the fact that the error function is the superposition of functions whose minima are at different points. In the case of linear neurons, all of these functions are convex, so no difficulties appear because a sum of convex functions is also convex. However, sigmoidal units lead to nonconvex functions, so it is not guaranteed that their sums will have a unique minimum. Moreover, it was shown (Auer, Hebster, & Warmuth, 1996) that the number of such minima can grow exponentially with the input dimension (d). In particular, they proved that for the squared error and the logistic neural function, the error function of a single neuron for n training examples may have $\lfloor n/d \rfloor^d$ local minima. In fact, this holds for any error and transfer functions for which their composition is continuous and has a bounded range.

The absence of local minima under certain separability or linear independence conditions (Budinich & Milotti, 1992; Gori & Tesi, 1992) or modifications of the least-squares error norm (Sontag & Sussmann, 1991) has also been proven. However, the problem of characterizing these extrema for the standard least-squares error, using only a finite number of arbitrary inputs, has not been previously solved, nor there is a clear understanding of the conditions that cause the appearance of local minima. Coetzee and Stonick (1996) proposed a method for the a posteriori evaluation, in single-layer perceptrons, of whether a weight solution is unique or globally optimal and for a priori scaling of desired vector values to ensure uniqueness through analysis of the input data. Although these results are potentially useful for evaluating optimality and uniqueness, the minima can be characterized only after training is complete.

In this article, a new training algorithm is proposed in order to avoid the problem of local minima in a supervised one-layer neural network with

nonlinear activation functions. In section 2, we describe the rationale of the problem. In section 3, the method for learning the weights of the network, leading to a global optimal solution, is presented. This approach is later enhanced by learning the neural functions. Section 4 deals with an alternative learning method that allows studying the variability of the different weights and indirectly testing the quality of the network. Section 5 gives some examples of applications to illustrate how the proposed methods can be used in practice. In section 6, the proposed methods are compared with other high-performance standard methods in terms of learning speed. Finally, section 7 gives some conclusions and recommendations.

2 Motivation

Consider the neural network in Figure 1a, where it is assumed that the nonlinear neural functions $f_1, f_2, \ldots, f_J$ are invertible. The set of equations relating inputs and outputs is given by

$$y_{js} = f_j\left(w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is}\right); \quad j = 1, 2, \ldots, J; \quad s = 1, 2, \ldots, S, \tag{2.1}$$

where w_{j0} and w_{ji}; $i = 1, 2, \ldots, I$, are the threshold values and the weights associated with neuron j (for $j = 1, 2, \ldots, J$) and S is the number of data points.

System 2.1 has $J \times S$ equations in $J \times (I + 1)$ unknowns. However, since the number of data is large ($S >> I + 1$), in practice, this set of equations in w_{ji} is not compatible and consequently has no solution. Thus, the usual approach is to consider some errors, δ_{js}; equation 2.1 is transformed into

$$\delta_{js} = y_{js} - f_j\left(w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is}\right); \quad j = 1, 2, \ldots, J; \quad s = 1, 2, \ldots, S, \tag{2.2}$$

and to estimate (learn) the weights, the sum of squared errors,

$$Q_1 = \sum_{s=1}^{S}\sum_{j=1}^{J}\delta_{js}^2 = \sum_{s=1}^{S}\sum_{j=1}^{J}\left(y_{js} - f_j\left(w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is}\right)\right)^2, \tag{2.3}$$

is minimized.

It is important to note that due to the presence of the neural functions f_j, the function appearing in equation 2.3 is nonlinear in the w_{ji} weights. Thus, Q_1 is not guaranteed to have a global optimum.

Actual methods for learning the weights in neural networks have two shortcomings. First, the function Q_1 to be optimized has several (usually many) local optima. This implies that the users cannot know whether they

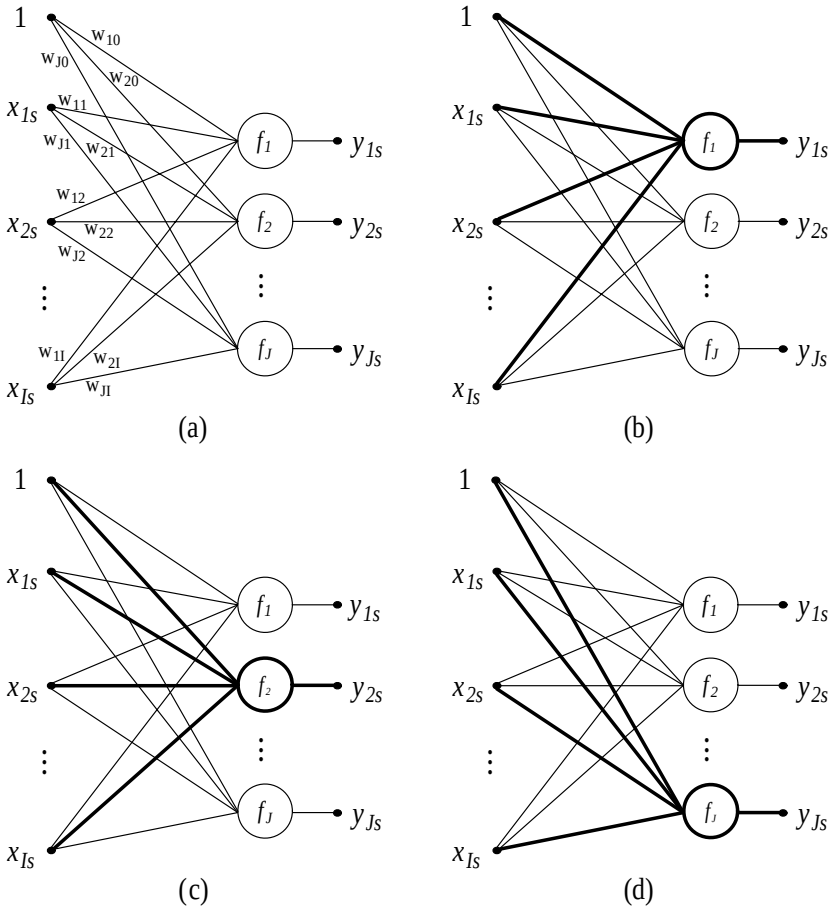

Figure 1: (a) Original one-layer feedforward neural network. (b–d) J independent neural networks leading to the J independent systems of linear equations in equation 3.4.

are in the presence of a global optimum and how far from it the local optimum resulting from the learning process is. In fact, several users of the same computer program can get, for the same data and model, different solutions if they use a different starting set of weights in the learning procedures. Second, the learning process requires a huge amount of computational power compared with other models.

Since for each j the weights w_{j0} and w_{ji}; $i = 1, 2, \ldots, I$ are related only to y_{js}, and appear only in S of the $J \times S$ equations in equation 2.2, it is evident that the problem of learning the weights can be separated into J independent

problems (one for each j). This means that the neural network in Figure 1a can be split in J independent simpler neural networks (see Figures 1b, 1c, and 1d). Thus, the problem becomes much simpler. In what follows we deal with only one of these problems (for a fixed j).

3 Proposed Method for Learning the Weights Based on Linear Systems of Equations

Motivated by the problems indicated in the previous section, an alternative approach for learning the weights is proposed.

The system of equations, 2.2, that measures the errors in the output scale (the units of the y_{js}) can be written as

$$\epsilon_{js} = w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is} - f_j^{-1}(y_{js}); \quad s = 1, 2, \ldots, S; \quad j = 1, 2, \ldots, J, \tag{3.1}$$

which measures the errors in terms of the input scale (units of the x_{is}).

We have two possibilities for learning the weights:

1. This option is to minimize the sum of squared errors,

$$Q_{2j} = \sum_{s=1}^{S} \epsilon_{js}^2 = \sum_{s=1}^{S} \left(w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is} - f_j^{-1}(y_{js}) \right)^2, \tag{3.2}$$

which leads to the system of linear equations,

$$\frac{\partial Q_{2j}}{\partial w_{jp}} = \begin{cases} 2\sum_{s=1}^{S} \left(w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is} - f_j^{-1}(y_{js}) \right) x_{ps} = 0 & \text{if} \quad p > 0 \\ 2\sum_{s=1}^{S} \left(w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is} - f_j^{-1}(y_{js}) \right) = 0 & \text{if} \quad p = 0, \end{cases} \tag{3.3}$$

which can be written as

$$\left.\begin{array}{rcl} \sum_{i=1}^{I} \left(\sum_{s=1}^{S} x_{is}x_{ps} \right) w_{ji} & = & \sum_{s=1}^{S} (f_j^{-1}(y_{js}) - w_{j0})x_{ps}; \; p = 1, 2, \ldots, I, \\ \sum_{i=1}^{I} \left(\sum_{s=1}^{S} x_{is} \right) w_{ji} & = & \sum_{s=1}^{S} (f_j^{-1}(y_{js}) - w_{j0}). \end{array}\right\}. \tag{3.4}$$

2. Alternatively, we can minimize the maximum absolute error,

$$\epsilon = \min_{\mathbf{w}} \left\{ \max_{s} \left| w_{j0} + \sum_{i=1}^{I} w_{ji}x_{is} - f^{-1}(y_{js}) \right| \right\}, \tag{3.5}$$

which can be stated as the following linear programming problem: Minimize ϵ subject to

$$\left.\begin{array}{rcl} w_{j0} + \sum_{i=1}^{I} w_{ji} x_{is} - \epsilon & \leq & f_j^{-1}(y_{js}) \\ -w_{j0} - \sum_{i=1}^{I} w_{ji} x_{is} - \epsilon & \leq & -f_j^{-1}(y_{js}) \end{array}\right\}; \quad s = 1, 2, \ldots, S; \tag{3.6}$$

the global optimum can be easily obtained by well-known linear programming techniques. In addition, it is easy to find the set of all possible solutions leading to the global optimum. This can be achieved by testing which constraints in equation 3.6 are active and solving the corresponding systems of linear equations to find the extreme points of the solution polytope.

Consequently, we obtain the global optimum by solving a linear system of equations or a linear programming problem, methods that require much less computational power than that involved in minimizing Q_1 in equation 2.3.

3.1 Learning the Neural Functions. An important improvement is obtained if we learn the f_j^{-1} functions instead of assuming that they are known. More precisely, they can be assumed to be linear convex combinations of a set

$$\{\phi_1(x), \phi_2(x), \ldots, \phi_R(x)\}$$

of invertible basic functions,

$$f_j^{-1}(x) = \sum_{r=1}^{R} \alpha_{jr} \phi_r(x); \; j = 1, 2, \ldots, J, \tag{3.7}$$

where $\{\alpha_{jr}; \; r = 1, 2, \ldots, R\}$ is the set of coefficients associated with function f_j^{-1}, which has to be chosen for the resulting function f_j^{-1} to be invertible. Without loss of generality, it can be assumed to be increasing.

Then, as before, we have two options:

1. Minimize, with respect to w_{ji}; $i = 0, 1, \ldots, I$ and α_{jr}; $r = 1, 2, \ldots, R$, the function

$$Q_{2j} = \sum_{s=1}^{S} \epsilon_{js}^2 = \sum_{s=1}^{S} \left(w_{j0} + \sum_{i=1}^{I} w_{ji} x_{is} - \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js}) \right)^2, \tag{3.8}$$

subject to

$$\sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js_1}) \leq \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js_2}); \quad \forall y_{js_1} < y_{js_2},$$

which forces the candidate functions f_j^{-1} to be increasing, at least in the corresponding intervals.

2. Alternatively, for each $j = 1, 2, \ldots, J$, we can minimize the maximum absolute error,

$$\epsilon = \min_{w,\alpha} \left\{ \max_s \left| w_{j0} + \sum_{i=1}^{I} w_{ji} x_{is} - \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js}) \right| \right\}, \tag{3.9}$$

which can be stated as the following linear programming problem: Minimize ϵ subject to

$$\begin{aligned} w_{j0} + \sum_{i=1}^{I} w_{ji} x_{is} - \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js}) - \epsilon &\le 0 \\ -w_{j0} - \sum_{i=1}^{I} w_{ji} x_{is} + \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js}) - \epsilon &\le 0 \\ \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js_1}) &\le \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_{js_2}); \quad \forall y_{js_1} < y_{js_2} \\ \sum_{r=1}^{R} \alpha_{jr} \phi_r(y_0) &= 1, \end{aligned} \tag{3.10}$$

which has a global optimum easily obtainable.

In order to avoid transformations with a large derivative, it is convenient to limit the first derivative of the $f_j^{-1}(y_{js})$ transformation. This can be done by adding the extra constraints,

$$\frac{df_j^{-1}(y)}{dy} = \sum_{r=1}^{R} \alpha_{jr} \phi_r'(y) \ge \beta; \; j = 1, 2, \ldots, J, \tag{3.11}$$

where β is the upper bound of the derivative of the inverse function that is equal to the inverse of the derivative of the original function.

Once the f_j^{-1} have been obtained, to use the network, it is necessary to work with their inverses, which can be dealt with using, for example, the bisection method. To avoid problems with the dimensions of the data, it is strongly suggested that the data be transformed to the unit hypercube.

4 Alternative Learning Method: Variability Study

If we write equation 2.1 as

$$w_{j0} + \sum_{i=1}^{I} w_{ji} x_{is} = f_j^{-1}(y_{js}); \quad s = 1, 2, \ldots, S, \tag{4.1}$$

since the number of unknowns is $(I + 1)$, we realize that a given set with $(I + 1)$ data points is enough (apart from a degenerated linear system) for learning the weights $\{w_{ij}; i = 0, 1, \ldots, I\}$.

The proposed alternative consists of learning the weights with a selected (deterministically or randomly) class of subsets, each containing $v \geq (I+1)$ data points, and determining the variability of the different weights based on the resulting values. To proceed, repeat steps 1 and 2 below, a large number of times, that is, from $r = 1$ to $r = m$ with m large, and then proceed to steps 3 and 4:

Step 1: Select, at random, a subset D_r of $v \geq (I + 1)$ data points from the initial set of S data points.

Step 2: Use the system of equations 3.4 or 3.6, or solve the linear programming problems in equations 3.8 and 3.10 to obtain the corresponding set of weights $\{w_{ji}^r; i = 0, 1, \ldots, I\}$.

Step 3: Calculate the means μ_{ji}, medians M_{ji}, and standard deviations σ_{ji} of the sets $\{w_{ji}^r\}; \forall i$.

Step 4: Return $\hat{w}_{ji} = \mu_{ji}$ or $\hat{w}_{ji} = M_{ji}$ as the estimates of w_{ji}, and σ_{ji} as a measure of the precision in the estimate of w_{ji}.

5 Example Applications

In this section, we illustrate the proposed methods by their application to well-known sets of data examples. The first two cases, in sections 5.1 and 5.2,[1] correspond to classification problems and are used to verify the soundness of the proposed learning methods in sections 3 and 4. The last two examples, in sections 5.3 and 5.4[2], are regression problems employed to compare the approach proposed in section 3, the improvement method presented in section 3.1 that allows the learning of the neural functions and the Levenberg-Marquardt backpropagation method (Hagan & Menhaj, 1994). In order to estimate the true error of the neural networks, a leave-one-out cross-validation was used, except in the case of example 5.3.

5.1 The Fisher Iris Data Example. Our first example uses the Iris data (Fisher, 1936), perhaps one of the best-known databases in the pattern recognition literature. Fisher's article is a classic in the field and continues to be referenced frequently. The data set contains three classes of 50 instances

[1] Data obtained from the UCI Machine Learning Repository (http://www.ics.uci.edu/∼mlearn/MLRepository.html).

[2] Data obtained from the Working Group on Data Modeling Benchmarks of the IEEE Neural Networks Council Standards Committee (http://neural.cs.nthu.edu.tw/jang/benchmark).

Table 1: Estimated Weights for Iris Data and Associated Standard Deviations.

Weight	Value	Mean	Median	SD
w_{10}	0.9704812	0.9642	0.9567	0.0614
w_{11}	−0.0273781	−0.0263	−0.0262	0.0149
w_{12}	−0.0498872	−0.0495	−0.0500	0.0170
w_{13}	0.0722752	0.0713	0.0717	0.0128
w_{14}	0.0979362	0.0996	0.0985	0.0199

Note: Equation 3.6 and the method in section 4 are used.

each, where each class refers to a type of iris plant (Setosa, Versicolour, and Virginica). One class is linearly separable from the other two, which are not linearly separable from each other.

The attribute information used to determine the type of plant (1 for Setosa, 2 for Versicolour, and 3 for Virginica) consists of sepal length in centimeters, sepal width in centimeters, petal length in centimeters, and petal width in centimeters.

We have used the neural network described in section 2 and Figure 1 and the learning method described in section 3. The neural function has been assumed to be

$$f^{-1}(x) = \arctan(x).$$

This neural function was used also in all the following examples. The resulting value of the mean squared error was 0.0337, and the obtained weights are shown in the second column of Table 1.

The dot plot in Figure 2B shows the value of the output for each of the 150 test examples employed in the leave-one-out cross-validation and the cutoff discriminating values (dashed lines) for the three groups. The resulting classification rule is as follows:

1. If $\tan(w_{10} + \sum_{i=1}^{4} w_{ji}x_{is}) \leq 1.4$, the plant s is in class Setosa.
2. If $1.4 < \tan(w_{10} + \sum_{i=1}^{4} w_{ji}x_{is}) \leq 2.365$, the plant s is in class Versicolour.
3. If $2.365 < \tan(w_{10} + \sum_{i=1}^{4} w_{ji}x_{is})$, the plant s is in class Virginica.

With these criteria, 98.67% of the plants are classified correctly. The confusion matrix in Table 2 shows the number of cases misclassified. Next, the alternative learning method described in section 4 was used. One hundred subsets of 50 randomly selected data points were used to learn the set of weights $\{w_{i1}; i = 0, 1, 2, 3, 4\}$. Their means, medians, and standard deviations and their associated empirical cumulative distributions functions are shown in Table 1 and Figure 2A. A comparison of the different weight estimates shows that they are very similar. On the other hand, the low values

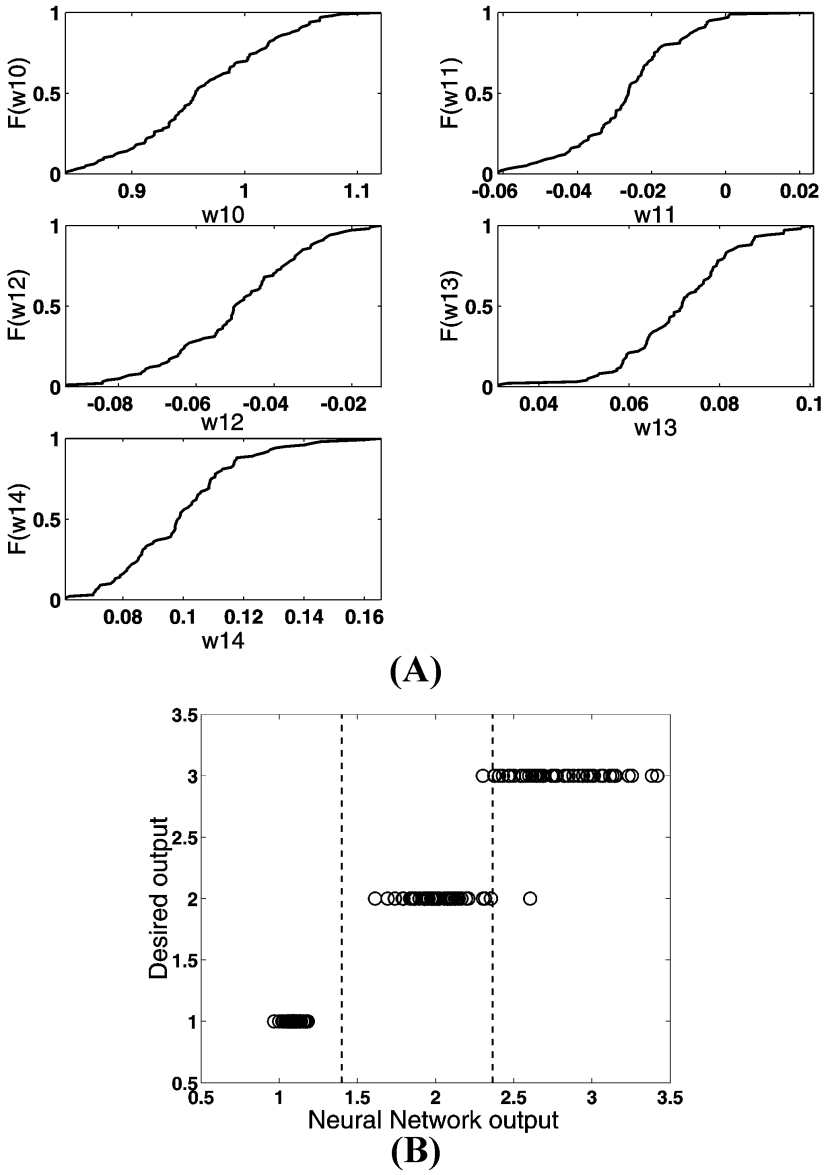

Figure 2: (A) Empirical cumulative distribution functions of the weights. (B) Plot for the iris data example.

Table 2: Confusion Matrix for the Iris Data.

System Classification	True Classification		
	Setosa	Versicolour	Virginica
Setosa	50	0	0
Versicolour	0	49	1
Virginica	0	1	49

Table 3: Attributes Used in Breast Cancer Data.

Attribute	Range
Clump thickness	1–10
Uniformity of cell size	1–10
Uniformity of cell shape	1–10
Marginal adhesion	1–10
Single epithelial cell size	1–10
Bare nuclei	1–10
Bland chromatin	1–10
Normal nucleoli	1–10
Mitoses	1–10

of the standard deviations suggest that the model is adequate for the Iris data set.

5.2 Breast Cancer Data. The breast cancer database was obtained from the University of Wisconsin Hospitals, Madison, from William H. Wolberg (Bennett & Mangasarian, 1992). It consists of 699 instances—458 (65.5%) benign and 241 (34.5%) malignant—with 16 having some missing data. We have used only the data with complete information: 683 data instances. The meanings of the nine attributes used to determine the class (0 for benign, 1 for malignant) are shown in Table 3. The dot plot in Figure 3B shows the value of the output for each of the 683 test examples employed in the leave-one-out cross-validation and the cutoff discriminating values (dashed lines) for the two classes. The resulting classification criterion is as follows:

1. If $\tan(w_{10} + \sum_{i=1}^{9} w_{ji}x_{is}) \leq 0.28$, the cancer s is classified as malignant.
2. If $\tan(w_{10} + \sum_{i=1}^{9} w_{ji}x_{is}) > 0.28$, the cancer s is classified as benign.

The number of cases classified correctly is 96.92%. The confusion matrix in Table 4 shows the number of misclassified data for each of the classes. Using the methods described in sections 3 and 4, the weight estimates with their standard deviations, shown in Table 5, have been obtained for 100 sub-

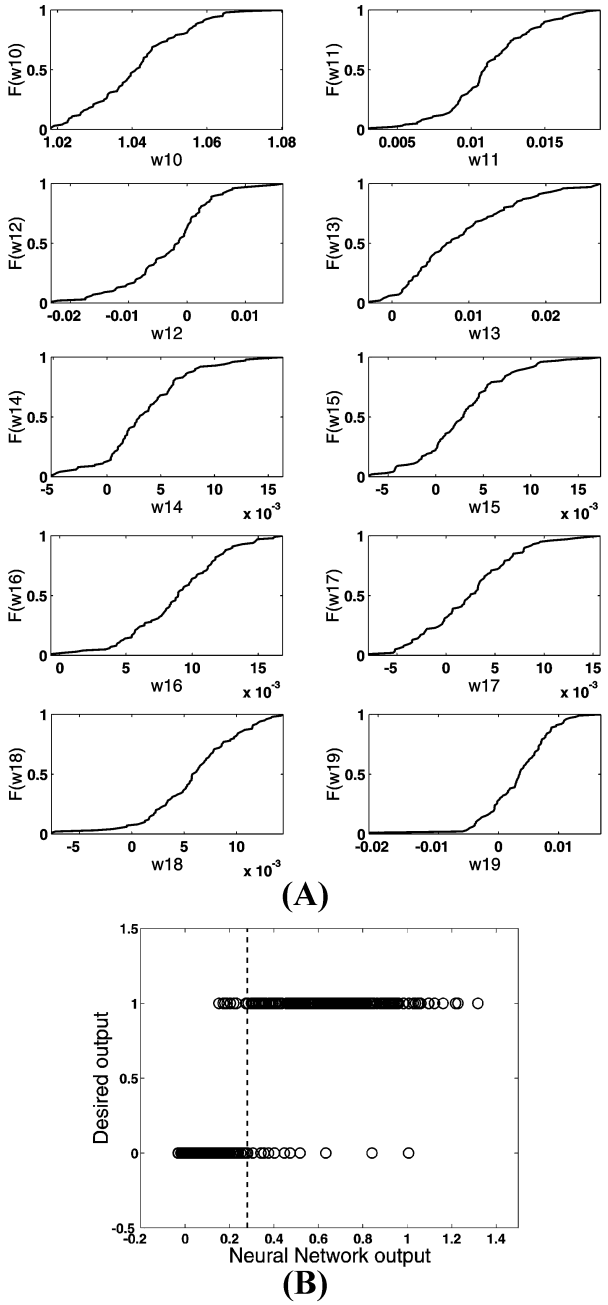

Figure 3: (A) Empirical cumulative distribution functions of the weights. (B) Plot for the breast cancer data.

Table 4: Confusion Matrix for Breast Cancer Data.

System Classification	True Classification	
	Benign	Malignant
Benign	432	9
Malignant	12	230

Table 5: Estimated Weights for Breast Cancer Data and Associated Standard Deviations.

Weight	Value	Mean	Median	SD
w_{10}	1.0530	1.0415	1.0411	0.0128
w_{11}	0.0069	0.0112	0.0109	0.0030
w_{12}	0.0048	−0.0028	−0.0015	0.0074
w_{13}	0.0034	0.0089	0.0074	0.0070
w_{14}	0.0018	0.0036	0.0030	0.0041
w_{15}	0.0022	0.0031	0.0029	0.0047
w_{16}	0.0099	0.0089	0.0089	0.0035
w_{17}	0.0042	0.0025	0.0025	0.0047
w_{18}	0.0041	0.0060	0.0059	0.0043
w_{19}	0.0002	0.0033	0.0035	0.0054

Note: Equation 3.6 and section 4 were used.

sets of 300 randomly selected data points. Figure 3A contains the associated empirical cumulative distributions of the weights. The estimates are very similar.

5.3 Modeling a Three-Input Nonlinear Function. In this example, we consider using the learning algorithm proposed in section 3 (see equation 3.3) and the improved method proposed in section 3.1 to model the nonlinear function

$$y = z^2 + z + \sin(z),$$

where

$$z = 3x_1 + 2x_2 - x_3.$$

We carried out 100 simulations employing 800 training data and 800 test data uniformly sampled from the input ranges $[0, 1] \times [0, 1] \times [0, 1]$. The values of the nonlinear function y were normalized in the interval [0.05, 0.95].

In the method with learnable neural functions, we employ the following polynomial family,

$$\{\phi_1(y), \phi_2(y), \ldots, \phi_R(y)\} = \{x, x^2, \ldots, x^R\}, \tag{4.2}$$

Table 6: Mean Error and Variability for the Three-Input Nonlinear Function.

	Mean Error	Variability
NN with known neural functions	1.155e-02	1.729e-04
NN with learnable neural functions	5.438e-03	6.509e-06
NN trained with Levenberg-Marquardt	1.189e-02	1.746e-04

as basic functions in equation 3.7. Several values of R and β (see equation 3.11) were tried. The results presented in this example were achieved with $R = 7$ and $\beta = 0.4$. Table 6 shows the mean error and the variability obtained for the test data over these 100 simulations. Also, the results obtained by the Levenberg-Marquardt method are shown in this table in order to compare our methods with a standard one. We tested the statistical significance of the differences between the mean error of the systems in Table 6 using a t-test. As can be seen, the neural network with known neural functions and that trained using the Levenberg-Marquardt obtained similar performance, whereas for a 99% significance level, the neural network with learnable functions outperformed the other two approaches.

Figure 4 shows a graph of the real output against the desired output (for one of the simulations) and the first 100 samples of the curve predicted by the networks in Table 6.

5.4 Box-Jenkins Furnace Data. This example deals with the problem of modeling a gas furnace that was first presented by Box and Jenkins (1970). The modeled system consists of a gas furnace in which air and methane are combined to form a mixture of gases containing carbon dioxide (CO_2). The furnace output, the CO_2 concentration, is measured in the exhaust gases at the outlet of the furnace. The data set corresponds to a time series consisting of 296 pairs of observations of the form $((u(t), y(t))$, where $u(t)$ represents the methane gas feed rate at the time step t and $y(t)$ is the concentration of CO_2 in the gas outlets. The sampling time interval is 9 seconds. The goal is to predict $y(t)$ based on $\{y(t-1), y(t-2), y(t-3), y(t-4), u(t-1), u(t-2), u(t-3), u(t-4), u(t-5), u(t-6)\}$. This reduces the number of effective data points to 290. Also, before the training began, the output, $y(t)$, was normalized in the interval [0.05,0.95].

The methods proposed in sections 3 (neural network with known neural functions using equation 3.1) and 3.1 (neural network with learnable neural functions) were employed to predict the CO_2 concentration. In the latter, several values of R and β were tried. The results obtained, for test data, using the polynomial family in equation 4.2, $R = 15$ and $\beta = 0.4$, are shown in Table 7. Again, results obtained by the Levenberg-Marquardt method are included in this table. To test the significance of the difference between the mean errors of the systems in Table 7, a t-test was used. As in the previous

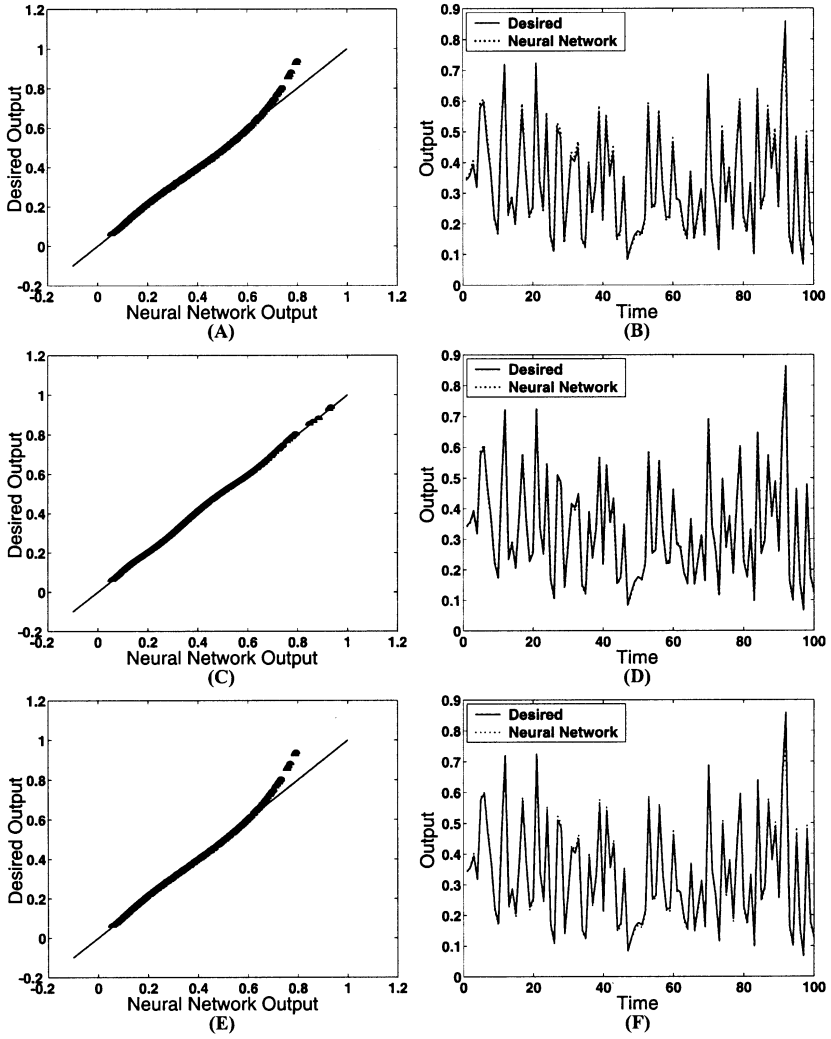

Figure 4: Comparison of the three-input nonlinear function. Neural network with given neural functions: (A) real against desired output and (B) observed (solid line) and predicted (dashed line) curves. Neural network with learnable neural functions: (C) real against desired output and (D) observed (solid line) and predicted (dashed line) curves. Levenberg-Marquardt method: (E) real against desired output and (F) observed (solid line) and predicted (dashed line) curves.

Table 7: Mean Error and Variability, Box-Jenkins Furnace Data.

	Mean Error	Variability
NN with known neural functions	2.084e-02	2.617e-04
NN with learnable neural functions	1.667e-02	1.794e-04
NN trained with Levenberg-Marquardt	2.090e-02	2.626e-04

example, for a 99% significance level, significant performance differences between the neural network with learnable functions and the other two approaches were found. Therefore, we can affirm that the improved method proposed in section 3.1 outperforms the first one introduced (in section 3), and so the learning of the neural functions enhances the performance of the system.

Figure 5 shows a plot of the real output against the desired output and the source data versus the time series predicted by our two neural systems and the Levenberg-Marquardt method.

6 A Learning Speed Comparison

This section contains a comparative study of the learning speed between the method proposed in section 3 (see equation 3.4) and several high performance algorithms. To this end, the examples described in sections 5.1, 5.2, 5.3, and 5.4 are used. The following algorithms were used in the experiments:

- Levenberg-Marquardt backpropagation (LM) (Hagan & Menhaj, 1994)
- Fletcher-Powell conjugate gradient (CGF) (Fletcher & Reeves, 1964; Hagan, Demuth, & Beale, 1996)
- Polak-Ribiére conjugate gradient (CGP) (Fletcher & Reeves, 1964; Hagan et al., 1996)
- Conjugate gradient with Powell-Beale restarts (CGB) (Powell, 1977; Beale, 1972)
- Scaled conjugate gradient (SCG) (Moller, 1993)
- Quasi-Newton with Broyden, Fletcher, Goldfarb, and Shanno updates (BFGS) (Dennis & Schnabel, 1983)
- One-Step secant backpropagation (OSS) (Battiti, 1992)
- Resilient backpropagation (RP) (Riedmiller & Braun, 1993)

For each of the data sets, 100 trials, with initial randomly weights, were carried out using a Silicon Graphics Origin 200 computer. The networks were trained using the mean squared error function in all cases. The training was stopped when the error goal (4e-4, 3e-4, 7.7e-3, and 1.7e-2 for examples

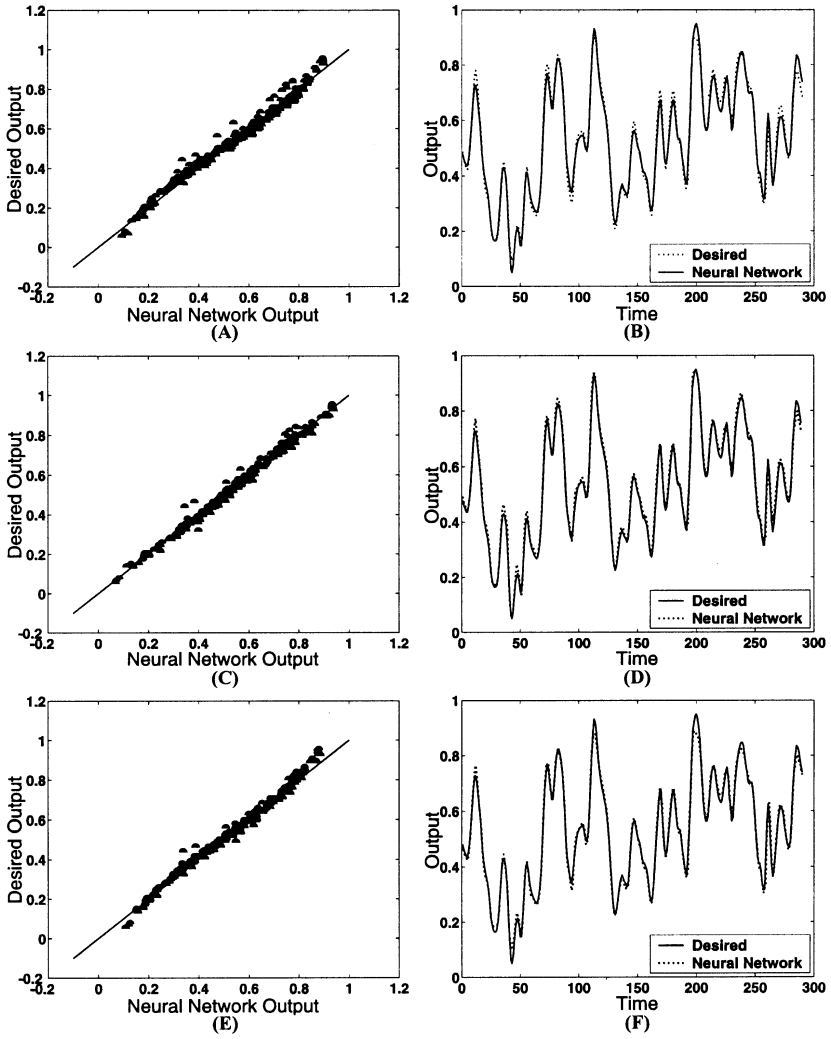

Figure 5: Comparison of results for Box-Jenkins data. Neural network with known neural functions: (A) real against desired output and (B) observed (solid line) and predicted (dashed line) time series. Neural network with learnable neural functions: (C) real against desired output and (D) observed (solid line) and predicted (dashed line) time series. Levenberg-Marquardt method: (E) real against desired output, and (F) observed (solid line) and predicted (dashed line) time series.

Table 8: Training Time for Iris Data.

Algorithm	Mean Time(s)	Ratio	Minimum Time(s)	Maximum Time(s)	SD
SLE	0.0054	1.00000	0.0053	0.0059	0.000074
LM	0.1538	28.4800	0.1255	0.1837	0.012203
CGF	0.7929	146.830	0.1133	1.4683	0.269170
CGP	0.8885	164.540	0.1260	1.6335	0.320160
CGB	0.7166	132.700	0.1131	1.2714	0.211600
SCG	0.6719	124.430	0.1190	1.6152	0.200100
BFGS	1.3441	248.910	0.7065	1.7931	0.266610
OSS	7.5704	1401.93	1.4436	22.398	4.333000
RP	7.2496	1342.52	1.1705	19.511	4.351700

Table 9: Training Time for Breast Cancer Data.

Algorithm	Mean Time(s)	Ratio	Minimum Time(s)	Maximum Time(s)	SD
SLE	0.0245	1.0000	0.0238	0.0265	0.00038
LM	0.2351	9.6000	0.1989	0.3119	0.02264
CGF	1.3737	56.070	0.1219	12.592	1.62990
CGP	1.6820	68.650	0.1338	20.211	2.86940
CGB	1.1796	48.150	0.1226	7.3879	0.86438
SCG	0.7870	32.120	0.1296	1.9263	0.19669
BFGS	2.7466	112.11	0.6902	8.3371	0.84858
OSS	39.230	1601.2	2.7709	173.39	56.0295
RP	1.6540	67.510	0.6553	2.1685	0.31739

5.1, 5.2, 5.3, and 5.4, respectively) was attained, the maximum number of epochs (2.000) was reached, or the gradient fell below a minimum threshold gradient (1e-10).

Tables 8 through 11 summarize the training speed results of the networks associated with all of the algorithms used. The tables contain the mean, the minimum and maximum time and the standard deviation obtained in 100 simulations. Also, the fastest algorithm is used as the reference for the time ratio parameter, so that it reflects how many times each algorithm is slower, on average, than the fastest algorithm. The method proposed in this article was identified with the SLE (system of linear equations) acronym. As can be seen in the tables, even in the worst case, the SLE method is about 10 times faster than the next faster algorithm. Finally, it is also important to note that the high speed differences obtained between our proposed method and the others are due to the iterative nature of the latter, while the former requires only solving a linear system of equations or a programming problem.

Table 10: Training Time for Nonlinear Function Data.

Algorithm	Mean Time(s)	Ratio	Minimum Time(s)	Maximum Time(s)	SD
SLE	0.0057	1.0000	0.0055	0.0066	0.00015
LM	0.1768	31.020	0.1642	0.1901	0.01122
CGF	0.8346	146.42	0.5026	1.3558	0.19879
CGP	0.8814	154.63	0.4800	1.8231	0.31403
CGB	0.7844	137.61	0.4426	1.3517	0.22015
SCG	0.5832	102.32	0.3701	0.8441	0.10821
BFGS	1.1666	204.67	0.7801	1.4787	0.13563
OSS	3.3554	588.67	1.6546	7.6841	1.03380
RP	3.1387	550.65	1.0870	4.5230	0.98790

Table 11: Training Time for Box-Jenkins Data.

Algorithm	Mean Time(s)	Ratio	Minimum Time(s)	Maximum Time(s)	SD
SLE	0.0243	1.0000	0.0224	0.0963	0.0073
LM	0.3178	13.080	0.2306	1.0212	0.0892
CGF	6.7133	276.27	2.5028	10.770	1.4806
CGP	13.658	562.05	4.2101	98.329	23.169
CGB	6.0728	249.91	4.2211	8.4621	0.9492
SCG	6.2298	256.37	3.8638	11.047	1.2715
BFGS	3.5819	147.40	2.4899	4.2659	0.3173
OSS	106.25	4372.4	69.439	113.09	9.6851
RP	31.572	1299.3	31.464	32.032	0.0785

7 Conclusion

In this article a new learning method for single-layer neural networks has been presented. This approach, which measures the errors in terms of the input scale instead of the output scale, allows reaching the global optimum of the error surface avoiding local minima. Two alternative training procedures were proposed for learning the weights. The first one, which employs the sum of squared errors as a cost function, is based on a system of linear equations, and the second one, which minimizes the maximum absolute error function, is based on a linear programming problem. It is worthwhile mentioning that due to the techniques employed to minimize the objective function, these approaches exhibit a very fast convergence. The proposed methods were later extended so that learning the neural functions is also possible. To this end, these functions were assumed to be a linear combination of invertible basic functions. In all the models presented in this article, a polynomial combination has been used, but many others can be considered (e.g., Fourier expansion). Finally, the examples show that the proposed

methods present good performance on well-known data sets. Moreover, the approach that allows learning the neural functions seems to be superior to the one with fixed neurons. The comparison with other learning algorithms shows that the proposed method clearly outperforms all existing methods. Finally, an important issue to address is the correspondence between minima of the standard error (see equation 2.3) and the novel formulation of the error (see equation 3.2). This correspondence is exact when the error is zero, that is, when the estimated model coincides with the real model underlying the data. Under other conditions, this correspondence does not exist and needs further analysis, which will be addressed in future work.

Acknowledgments

We thank the Universities of Cantabria and Castilla-La Mancha, the Dirección General de Investigación Científica y Técnica (DGICYT) (project PB98-0421), Iberdrola, and the Xunta de Galicia (project PGIDT99COM10501 and the predoctoral grants programme 2000/01) for partial support of this research.

References

Auer, P., Hebster, M., & Warmuth, M. K. (1996). Exponentially many local minima for single neurons. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Advances in neural processing systems*, *8* (pp. 316–322). Cambridge, MA: MIT Press.

Battiti, R. (1992). First and second order methods for learning: Between steepest descent and Newton's method. *Neural Computation*, *4*(2), 141–166.

Beale, E. M. L. (1972). A derivation of conjugate gradients. In F. A. Lootsma (Ed.), *Numerical methods for nonlinear optimization*. London: Academic Press.

Bennett, K. P., & Mangasarian, O. L. (1992). Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, *1*, 23–34.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. New York: Oxford University Press.

Box, G. E. P., & Jenkins, G. M. (1970). *Time series analysis, forecasting and control*. San Francisco: Holden Day.

Brady, M., Raghavan, R., & Slawny J. (1989). Back propagation fails to separate where perceptrons succeed. *IEEE Transactions on Circuits and Systems*, *36*, 665–674.

Budinich, M., & Milotti, E. (1992). Geometrical interpretation of the backpropagation algorithm for the perceptron. *Physica A*, *185*, 369–377.

Coetzee, F. M., & Stonick V. L. (1996). On uniqueness of weights in single layer perceptrons. *IEEE Transactions on Neural Networks*, *7*(2), 318–325.

Dennis, J. E., & Schnabel, R. B. (1983). *Numerical methods for unconstrained optimization and nonlinear equations*. Englewood Cliffs, NJ: Prentice Hall.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics*, *7*, 179–188.

Fletcher, R., & Reeves, C. M. (1964). Function minimization by conjugate gradients. *Computer Journal*, *7*, 149–154.

Gori, M., & Tesi, A. (1992). On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *14*(1), 76–85.

Hagan, M. T., Demuth, H. B., & Beale, M. H. (1996). *Neural network design*. Boston: PWS Publishing.

Hagan, M. T., & Menhaj, M. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, *5*(6), 989–993.

Moller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, *6*, 525–533.

Powell, M. J. D. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming*, *12*, 241–254.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*. San Francisco.

Sontag, E., & Sussmann, H. J. (1989). Backpropagation can give rise to spurious local minima even for networks without hidden layers. *Complex Systems*, *3*, 91–106.

Sontag, E., & Sussmann, H. J. (1991). Back propagation separates where perceptrons do. *Neural Networks*, *4*, 243–249.

Widrow, B., & Lehr, M. A. (1990). Thirty years of adaptive neural networks: perceptron, madeline and backpropagation. *Proceedings of the IEEE*, *78*(9), 1415–1442.

Received May 11, 2001; accepted August 22, 2001.