LETTER Communicated by Chris Williams

Efficient Block Training of Multilayer Perceptrons

A. Navia-Vázquez
A. R. Figueiras-Vidal
DTC, Universidad Carlos III de Madrid, 28911-Leganés (Madrid), Spain

The attractive possibility of applying layerwise block training algorithms to multilayer perceptrons MLP, which offers initial advantages in computational effort, is refined in this article by means of introducing a sensitivity correction factor in the formulation. This results in a clear performance advantage, which we verify in several applications. The reasons for this advantage are discussed and related to implicit relations with second-order techniques, natural gradient formulations through Fisher's information matrix, and sample selection. Extensions to recurrent networks and other research lines are suggested at the close of the article.

1 Introduction

Multilayer perceptrons (MLP) offer a powerful approach for solving estimation and classification problems, where gradient descent with error backpropagation (BP) is the classical algorithm for training them. The BP algorithm not only has problems with local minima (also present in other search methods) but also with the speed of convergence. Similar drawbacks exist to some extent in many BP variants.

The so-called block learning methods represent an interesting alternative because they proceed blockwise, decomposing the global problem into simpler layer-by-layer minimizations, such that training consists of iteratively solving those minimizations by means of efficient methods. This new approach makes gradient descent unnecessary (hence, avoiding its problems) and has foreseeable additional benefits, such as lower computational cost along with easier analysis and control of the training process.

We will disregard as block training techniques those hybrid (between block and gradient) algorithms, such as the moving targets (MT) algorithm (Rohwer, 1991), the LS-based algorithm proposed by Di Claudio, Parisi and Orlandi (1993), and the dynamic hidden targets (DHT) algorithm (Song & Hassoun, 1990) because they use gradient descent to some extent. We will therefore consider as pure block training methods either those that rely on linearization of the network (Douglas & Meng, 1991; Deller & Hunt, 1992) or those that use the inverse of the activation function to propagate target values to the previous layer (Pethel & Bowden, 1992; Biegler-König & Bärmann, 1993). Furthermore, we will focus here on the latter because of

Neural Computation **12, 1429–1447** (2000)

their proved superior convergence. Basically, these methods decompose the global problem into minimization problems at every layer—mainly, least squares (LS) minimizations—with a twofold purpose: obtaining optimal weight values for every layer and backpropagating target values to previous layers.

In the following section, we present the block training framework in more detail and qualitatively analyze a direct implementation known as least squares backpropagation (LSB) (Biegler-König & Bärmann, 1993). In section 3, we propose a modification to LSB by solving reduced-sensitivity minimizations, thereby obtaining the reduced-sensitivity, least-squares block (RS-LSB) algorithm. Section 4 compares the performance of the proposed algorithm with some other well-known training methods, using some standard simulation examples. In section 5 we compare the computational cost per epoch of these algorithms. In section 6 we give some theoretical insights about the relationship between RS-LSB and some other efficient minimization techniques, such as natural gradient or second-order methods; additionally, we comment on its relationship with sample selection and regularization strategies. Finally, we present our conclusions and propose some further work.

2 Block Training

2.1 General Formulation. To preserve generality, a feedforward neural network (FFNN) scheme with K layers has been adopted, layer k containing N_k neurons. The goal of the network is to learn the associations $(\mathbf{x}_p, \mathbf{d}_p)$ for a total of P patterns, where $\mathbf{x}_p$ is the input and $\mathbf{d}_p$ the corresponding desired output. From now on, matrix notation is adopted by storing the input and output patterns as the rows of matrices $\mathbf{X}$ ($P \times N_0$) and $\mathbf{D}$ ($P \times N_L$), respectively. Using this notation and taking into account that every layer contains linear (weighted sum) and nonlinear (sigmoid-like activation function) components, the update equations for layer k are:

$$\mathbf{Z}^{(k)} = \mathbf{O}^{(k-1)}\mathbf{W}^{(k)} \tag{2.1}$$

$$\mathbf{O}^{(k)} = [\mathbf{1}|tanh\mathbf{Z}^{(k)}], \tag{2.2}$$

where | denotes augmentation, $\mathbf{O}^{(k-1)}$ is the output of the preceding layer, $\mathbf{W}^{(k)}$ is the weight matrix interconnecting layers $k-1$ and k, and $\mathbf{Z}^{(k)}$ is the matrix formed with the state values. In equation 2.2, a hyperbolic tangent function, which maps state values to the range $(-1, +1)$, has been used without loss of generality, and an unity vector ($\mathbf{1}$) has been added to $\mathbf{O}^{(k)}$ such that offset values for layer $k+1$ can be stored as the first row of $\mathbf{W}^{(k+1)}$. Additionally, matrices $\mathbf{T}^{(k)}$ and $\mathbf{Q}^{(k)}$ store output and state target values, respectively. Thus, $\mathbf{O}^{(0)} = \mathbf{X}$ (at layer 0) and $\mathbf{T}^{(K)} = \mathbf{D}$ (at layer K).

In block training, the global minimization problem is suboptimally decomposed by solving at every layer (and iteratively until convergence)

"double" least-squares problems—that is, with respect to the weights (to solve the linear part, equation 2.1, optimally) and with respect to the previous layer outputs (to propagate errors into the preceding layer). These adjustments are usually carried out by solving LS minimizations because, for linear problems, they lead to well-known robust and efficient algorithms. A more detailed description of a block training framework is given in the next section, where we describe a basic implementation.

2.2 A Qualitative Analysis of Direct Application. One of the simplest block training methods is the LSB algorithm (Biegler-König & Bärmann, 1993). We propose to apply at every layer the following three-step procedure. First, the inverse of the activity function is applied to target outputs in order to compute target states,

$$\mathbf{Q}^{(k)} = tanh^{-1}\left(\mathbf{T}^{(k)}\right). \tag{2.3}$$

Then the optimal weight matrix $\widehat{\mathbf{W}}^{(k)}$ for that layer can be obtained by solving

$$\min_{\widehat{\mathbf{W}}^{(k)}} \left\| \mathbf{O}^{(k-1)}\widehat{\mathbf{W}}^{(k)} - \mathbf{Q}^{(k)} \right\|_2. \tag{2.4}$$

Once the new weights are computed, a second minimization problem is solved to obtain output target values for the previous layer ($\mathbf{T}^{(k-1)}$),

$$\min_{\mathbf{T}^{(k-1)}} \left\| \mathbf{T}^{(k-1)}\widehat{\mathbf{W}}^{(k)} - \mathbf{Q}^{(k)} \right\|_2. \tag{2.5}$$

Thus, a training epoch is completed after applying this three-step procedure at layers $K, K-1, \ldots, 1$. This procedure is repeated until an appropriate convergence criterion is reached.

A final consideration has to be taken into account when implementing LSB. Specifically, the target values obtained for the previous layer usually lie outside the range $(-1, +1)$, requiring a normalization procedure to map them into the appropriate interval, before applying the inverse activation function, equation 2.3. A scaling matrix $\mathbf{C}^{(k)}$ has to be computed at every layer if normalization is necessary (a detailed description of the computation of $\mathbf{C}^{(k)}$, as well as its inverse, can be found in (Biegler-König & Bärmann, 1993). The new matrices are

$$\widetilde{\mathbf{T}}^{(k-1)} = \mathbf{T}^{(k-1)}\left(\mathbf{C}^{(k)}\right)^{-1}$$

$$\widetilde{\mathbf{W}}^{(k)} = \mathbf{C}^{(k)}\mathbf{W}^{(k)},$$

which ensures that $\widetilde{\mathbf{T}}^{(k-1)}\widetilde{\mathbf{W}}^{(k)} = \mathbf{T}^{(k-1)}\mathbf{W}^{(k)}$, and values in $\widetilde{\mathbf{T}}^{(k-1)}$ lie in $(-1, +1)$, such that equation 2.3 can be applied.

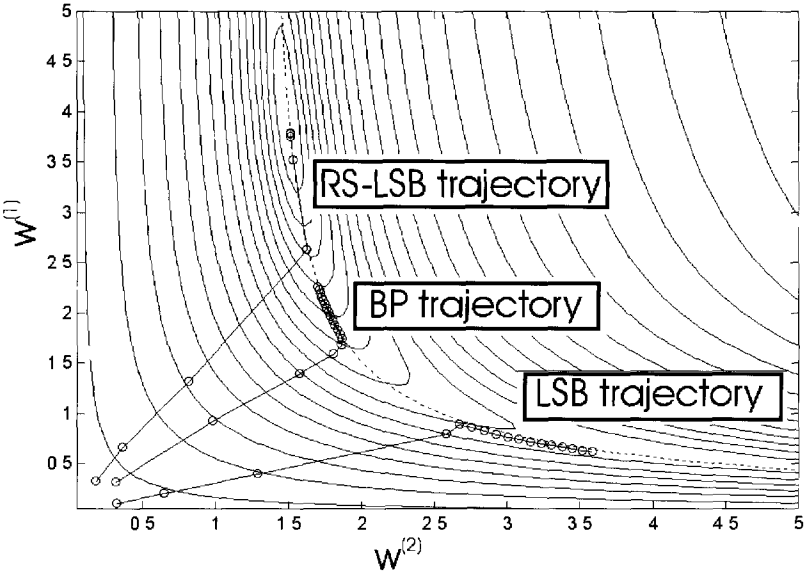

Figure 1: Contour plot and comparison of convergency trajectories of LSB, BP, and RS-LSB for the simple test case.

In some practical situations, we have found that LSB has problems related to the final error and weight values. To analyze the behavior of LSB, a very simple model has been selected, which, in spite of its simplicity, allows us to highlight the weak points of LSB and propose an improved block training algorithm.

The test problem is the following: Given a network with a single hidden neuron, linear activation in the second layer,[1] and specific target outputs, find the optimal weights using LSB with a random weight initialization. A test case can be created with random input values chosen in the interval $(-1, 1)$ and desired outputs computed using a model with $w^{(1)}$=4 and $w^{(2)}$=1.5.[2] The associated error surface can be easily computed for this case and is depicted in Figure 1 as a contour plot with respect to $w^{(1)}$ and $w^{(2)}$.

As it can be observed, there exists a minimum at (1.5, 4) and the surface resembles a valley with a curved bottom (marked with a dashed line); in this area partial derivatives become very small, and the convergence of

[1] In this case, no offset values have been used, and the weight matrices are reduced to real numbers $w^{(1)}$ and $w^{(2)}$.

[2] The choice of these values is not critical whenever $w^{(1)}$ is not very small; otherwise, the model would be nearly linear and the experiment irrelevant.

gradient-descent algorithms slows dramatically. This effect is corroborated by observing the gradient-descent trajectory in Figure 1, where the first 20 weight values have been represented by circles.

When LSB is used, it converges to the bottom of the valley in a few iterations (mainly adjusting $w^{(2)}$). Once there, it moves along the bottom, not toward the minimum but in the opposite direction (see Figure 1). We observe that LSB is not able to find a reasonable solution and, furthermore, it exhibits the undesirable behavior of moving away from the minimum.[3] The final situation achieved with LSB is a model where the hidden neuron works in its linear region (small weight values in the first layer and, hence, small state values for all the patterns).

Similar effects can be observed in networks with many neurons, where the LSB algorithm fails to obtain a good solution, as will be shown in section 4 by means of several real-world applications.

3 Reduced Sensitivity Block Training

In an MLP architecture, an error in state $z_p^{(k)}$ is propagated to the output of the neuron with a different gain $s_p^{(k)}$ (referred to as sensitivity) for every pattern. This "error gain" can be approximately computed as the derivative of the activation function at $z_p^{(k)}$; that is, sensitivity is high at the linear region of the activation function and low at the saturation regions. In a block training framework, the inverse of the activation function is used to compute target states from target outputs (see equation 2.3), and some problems appear when $s_p^{(k)}$ is very small, because target state values ($\mathbf{Q}^{(k)}$) show high variability even for small perturbations in $\mathbf{T}^{(k)}$ (the "inverse error gain" is $1/s_p^{(k)}$).

We propose to take advantage of a weighted LS (WLS) cost function (the weighting values being the sensitivities s_p) to obtain an improved solution:[4]

$$E(\mathbf{w}) = \sum_{p=1}^{P} \left[s_p e_p(\mathbf{w})\right]^2 = \sum_{p=1}^{P} \left[s_p \left(\mathbf{o}_p^T \mathbf{w} - q_p\right)\right]^2. \tag{3.1}$$

Minimizing $E(\mathbf{w})$ with respect to $\mathbf{w}$ can be carried out using classical LS techniques (such as Moore-Penrose pseudoinverse, QR or SVD decomposition, and gaussian elimination) by redefining $\mathbf{q}$ and $\mathbf{O}$ as follows,

$$\mathbf{q}_s = diag(s_p)\mathbf{q} \tag{3.2}$$

$$\mathbf{O}_s = diag(s_p)\mathbf{O}. \tag{3.3}$$

[3] This happens even when weights are initialized very close to the optimal values.

[4] Indexes referring to neuron and layer have been dropped for convenience. This error is local, and the WLS minimization has to be repeated for every neuron in the network.

and solving

$$\min_{\mathbf{w}} \left\| \mathbf{O}_s \mathbf{w} - \mathbf{q}_s \right\|_2 . \tag{3.4}$$

If we apply this WLS formulation to the LSB algorithm, we obtain the RS-LSB algorithm. In RS-LSB, a generalized expression for sensitivity can be used:

$$s_{p,j}^{(k)} = (tanh'(z_{p,j}^{(k)}))^{\beta}, \tag{3.5}$$

where *tanh'* is the derivative of the activation function. Expression 3.5 is qualitatively equivalent to simply evaluating the derivative, but it provides an additional adjustable parameter β.

Some interesting characteristics arise with this sensitivity definition because every neuron will selectively focus on patterns with maximal sensitivity.[5] At the first layer, the maximal sensitivity locus for neuron j is a strip in the N_0-dimensional input space, centered around the hyperplane $\mathbf{o}^{(0)}\mathbf{w}_j^{(1)} = 0$, and vanishing as the distance to the hyperplane increases. In following layers, this locus can take different shapes, because states are computed using nonlinear combinations of previous layer outputs. For illustration, we present a simple surface modeling problem. We constructed a training set using 400 samples from the surface depicted in Figure 2a and trained a 2-4-1 MLP using RS-LSB. The resulting surface is depicted in Figure 2b; in Figure 2c we represent the four sensitivity strips at the input space using gray shades (one strip for every neuron in the hidden layer, and drawn superimposed), where every hyperplane (center of every strip) has been marked using dashed lines. Several trials yielded different arrangements of strips at the first layer, but in every case, a circular sensitivity strip at the output layer (see Figure 2d) was obtained, and the problem was solved.

If we regard the problem represented in Figure 2a as a binary classification task (with a circular boundary between classes as solution), we observe that the network concentrates on patterns closer to such a boundary (see Figure 2d), thereby implementing a pattern selection strategy (this aspect will be discussed further in section 6.4).

We have observed through extensive simulation that the final performance is not very dependent on the exact location of the strips and that the block training method manages to generate a strip disposition, which, combined in following layers, leads to a satisfactory solution (the low variance of the experiments described in section 4 also suggests this fact). Note that

[5] Standard BP behaves in an analogous manner (neurons are more strongly sensitive to being trained by patterns that fall in their "active" region), while LSB does not; hence, one of the benefits of RS-LSB is to incorporate this pattern discrimination behavior into a block training framework.

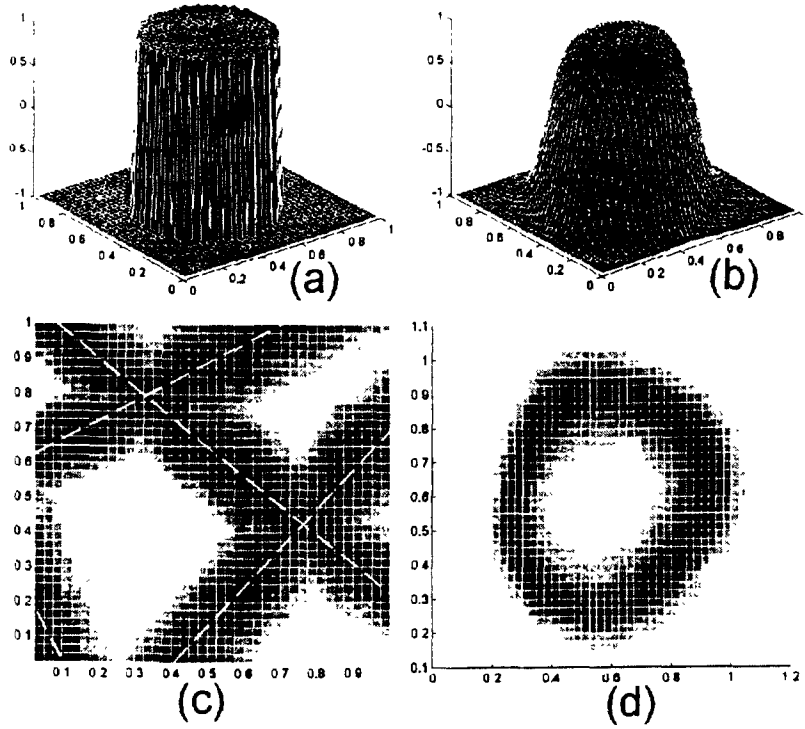

Figure 2: Visualization of the sensitivity strips in a simple case: (a) original surface and (b) NN approximation using a 2-4-1 MLP trained with RS-LSB. The sensitivity strips for the input layer can be observed in (c) (four superimposed, one for every neuron); the maximal sensitivity locus at the output layer is depicted in (d).

if $\beta = 0$, the algorithm reduces to LSB (strips having infinity width) and the network loses approximation capabilities, as will be shown in section 4 by means of several examples.

If small, random values are used to initialize the weights, the neuron states are usually close to zero ($z_p \approx 0$) at initial stages, yielding sensitivity values very close to the unity. In this case, there is very little discrimination (as in the LSB algorithm), and ill conditioning in the next layer may arise (especially when $N_k > N_{k-1}$). To avoid this, sensitivity is computed using modified state values in matrix $\mathbf{Z}^{(k)}$, scaled and shifted to the range $(-a, a)$:

$$s_{p,j}^{(k)} = \left(tanh' \left(a \frac{\left(2z_{p,j}^{(k)} - (m_1 + m_2) \right)}{(m_2 - m_1)} \right) \right)^{\beta} \tag{3.6}$$

where

$$m_1 = \min_{p,j} \left\{ z_{p,j}^{(k)} \right\}; \quad p = 1, \ldots, P; \; j = 1, \ldots, N_K$$

$$m_2 = \max_{p,j} \left\{ z_{p,j}^{(k)} \right\}; \quad p = 1, \ldots, P; \; j = 1, \ldots, N_K.$$

Sensitivity computation depends not only on β but also on the weight values of every neuron ($s = (tanh'(\mathbf{o}^T\mathbf{w}))^\beta$), and therefore the algorithm is not severely constrained by a particular choice of β (whenever it is greater than zero; otherwise LSB algorithm is obtained). In fact, we have found that similar sensitivity values can be obtained with different values of β with appropriate weight values. Because the method is not highly sensitive to the selection of β, we did not consider it necessary to explore this further analytically (nontrivial task). We suggest, instead, trying several values and retaining the best results (usually it is enough to try out $\beta = 1, 2, 3$), a procedure that we have followed in the experiments.

It is possible to simplify the RS method using incomplete data set analysis (Rao & Toutenburg, 1995), where some patterns (those with sensitivity below a threshold) can be purposely discarded or truncated. Analogous simplified procedures have been applied in (Azimi-Sadjadi & Liou, 1992; Wang & Chen, 1996); unfortunately, patterns are discarded if they lie outside the inverse activation function domain, which clearly is a suboptimal criterion.

4 Application Examples

Before presenting the experimental results in detail, we will describe some features (common in all of the experiments) such as the NN architecture, error measures, experimental set-up, convergence criterion, and particular implementations of the classical training algorithms used for comparison. Most of the specifications conform with those defined in Proben, an NN benchmark specification document and database (Prechelt, 1994), although we will explicitly state them for further clarity.

With respect to the NN architecture, we use throughout an MLP with a single hidden layer and feedforward connections only (no intralayer connections, or shortcuts); the number of inputs and hidden neurons are different in every experiment. A hyperbolic tangent is used in the hidden layer, and the output layer uses linear activation functions for estimation problems and sigmoidal ones for classification problems. Initialization is random, and weights are chosen in $[-0.1, 0.1]$. We restrict ourselves to single hidden-layer MLPs due to their universal approximation capabilities, with the number of hidden neurons chosen without much care. This may lead to suboptimal solutions but poses no problem for efficiency comparisons

of the training algorithms given a fixed architecture, the matter of main concern here.

Performance is evaluated using the normalized mean square error (NMSE = MSE/σ^2, σ^2 being the variance of the output training values) for approximation problems, and classification error (CE) for classification problems.

The convergence criterion used is early stopping by cross-validation. In particular, we interrupt the training process when generalization loss ($GL(n) = 100(E_{valid}(n)/E_{opt}(n) - 1)$, $E_{opt}(n) = \min\{E_{valid}(i), i = 1, \ldots, n\}$, as defined in Prechelt (1994) exceeds 1%. Data sets are split into training, validation, and test sets using a specified number of patterns, and the data are normalized to the interval $(-1, 1)$. Ten runs are conducted in every experiment, and resulting data are collected using mean values and standard deviations.

We have chosen Matlab to set up experiments because it is a widely used tool and many NN training algorithms are readily available. Although faster implementations of these algorithms are usually available in C or C++, the use of Matlab is reasonable for comparison purposes if we measure the number of floating-point operations (FLOPS) consumed by every algorithm until convergence. The gradient descent with error BP, fast backpropagation (FBP), and Levenberg-Mardquardt (LM) algorithms are available in the neural networks toolbox (trainbp, trainbpx, and trainlm routines, respectively), while conjugate gradient (CG) and quasi-Newton (with Broyden-Fletcher-Goldfarb-Shanno update, BFGS) methods were not available there and had to be extracted from Bishop's NETLAB library (routines conjgrad and quasinew, respectively, also coded in Matlab).[6] Block learning methods are also implemented in Matlab, using gauss elimination to solve every LS minimization. Unless otherwise specified, all of these algorithms have been run with the default parameter settings included in the original libraries.

We now describe the data sets used, as well as the results obtained with each of the algorithms.

4.1 The "Cancer" Data Set. This data set has been extracted from the Proben benchmark data set collection (Prechelt, 1994) and is also part of the UCI machine learning repository under the description "breast cancer Wisconsin."[7] The purpose here is to use some cell characteristics obtained by microscopic examination (nine in total) in order to classify a tumor as benign or malignant. The total number of patterns is 699, and we have chosen a data partition that uses the first 50% of the patterns for train-

[6] NETLAB can be obtained online at http://www ncrg.aston.ac.uk/netlab.

[7] Proben is available for anonymous FTP from the Neural Bench archive at Carnegie Mellon University (ftp.cs.cmu.edu,/afs/cs/project/connect/bench/contrib/prechelt).

Table 1: Performance Comparison for the "Cancer" Data Set.

	BP	FBP	CG	BFGS	LM	LSB	RS-LSB
Train CE %	3.7 (0.12)	4 (0.3)	3.7 (0.2)	4.6 (0.7)	3.9 (0.5)	4.9 (0.4)	3.8 (0.3)
Test CE %	1.7 (0.1)	2.2 (0.5)	1.8 (0.2)	2.8 (0 9)	1.8 (0.4)	2 8 (0.4)	1.7 (0.2)
No Iters.	2517 (887)	79 (36)	12.8 (0.5)	8 8 (4.2)	5.7 (2.9)	8 (1.2)	9.2 (0.9)
Load % BP	100%	3.5%	5.5%	3.5%	11%	1.1%	1.7%

Notes: We show Classification error (mean values and standard deviations, in parentheses), corresponding to 10 runs. The average number of epochs to converge and (measured) computational cost have also been included, the latter expressed in percentage with respect to BP load.

ing, 25% for validation, and 25% for testing (cancer1 partition in Proben). The MLP architecture used is 9-8-1 for all cases, and the parameters used are the default ones in the implementation, except for the learning rate in BP, which was set to 10^{-3} (otherwise training was exceedingly slow). In RS-LSB we used $a = 3$ and $\beta = 1$. The number of runs is 10, and the results of the experiment are shown in Table 1, where we present mean and standard values of training and test error classification. We observe that the best-performing methods are BP, CG, LM, and RS-LSB, LSB yielding a suboptimal solution (CE almost twice than in the RS-LSB case). Although further explanation will be given in next section, we can see that block learning methods are advantageous in computational load with respect to the other algorithms, RS-LSB being about three times faster than CG or six times faster than LM, two of the fastest algorithms yielding similar performance.

4.2 The "Boston Housing" Data Set. This data set has been extracted from the Delve[8] database and is also part of the Statlib database at the Carnegie Mellon University.[9] It contains information about housing in the Boston area. The purpose here (prototask described as "price") is to use some variables (for example, per capita crime rate by town, nitric oxides concentration, average number of rooms per dwelling, and pupil-teacher ratio by town—up to 13 mixed categorical and continuous variables) to pre-

[8] Delve—Data for Evaluating Learning in Valid Experiments—is a standardized environment designed to evaluate the performance of methods that learn relationships based primarily on empirical data; it can be accessed online at the Computer Science Department of the University of Toronto (http://www.cs.toronto.edu/~delve/).

[9] The location is http://lib.stat.cmu.edu/datasets/boston.

Table 2: Results for the "Boston Housing" Data Set.

	BP	FBP	CG	BFGS	LM	LSB	RS-LSB
Train NMSE	0.09 (0.05)	0.41 (0.12)	0.17 (0.05)	0.26 (0.06)	0.12 (0.2)	0 21 (0.02)	0.12 (0.03)
Test NMSE	0.18 (0.01)	0.45 (0.1)	0.27 (0.05)	0.3 (0.07)	0.21 (0.11)	0.31 (0.01)	0.19 (0.05)
No Iters.	688 (68)	51 (35)	15 (4.7)	7.6 (3.9)	7.4 (2 9)	10 (2 7)	31 (6.5)
Load % BP	100%	7%	24%	15%	363%	7%	19%

Notes: We illustrate performance by means of classification error (mean values and standard deviations, in parentheses), corresponding to 10 runs. The average number of epochs to converge and the (measured) computational cost have also been included, the latter expressed in percentage with respect to BP load.

dict house prices. The total number of patterns is 506 in this case, and we also used a data partition with the first half of the patterns for training, 25% for validation, and 25% for testing. The MLP architecture used is 13-20-1, and the training parameters are the default ones, except for the learning rate in BP, which was set to 10^{-3}. In RS-LSB we have used $a = 3$ and $\beta = 1$. The number of runs is 10, and the results are shown in Table 2, where we present mean and standard values of training and test NMSE. We observe that the best-performing methods are BP, LM, and RS-LSB, LSB yielding a suboptimal solution (NMSE almost double than with RS-LSB). In this case, RS-LSB is about 5 times faster than BP and 20 times faster than LM, the two algorithms yielding similar performance.

4.3 The "Computer Activity" Data Set. This data set is also included in the Delve data set and contains information about computer system measures (e.g., number of system calls per second, number of "write" calls) in order to predict CPU consumption. The number of available patterns is 8192, and we have used the first half of them for training, 25% for validation, and the rest for testing (with the original ordering). The MLP architecture used is 12-8-1, and the parameters are the default ones, except for the learning rate in BP, which was set to 5×10^{-5} (the default value was too high for this application and caused instabilities). In RS-LSB we have used $a = 3$ and $\beta = 3$. The number of runs is 10, and the results are shown in Table 3, where we present mean and standard values of training and test error classification. We observe that the best-performing methods are BP, FBP, CG, and RS-LSB, LSB yielding a suboptimal solution (NMSE about four times bigger than in RS-LSB case). This time, RS-LSB is about 13 times faster than BP, 6 times faster than FBP, and 11 times faster than CG.

Table 3: Experiment Results for the "Computer Activity" Data Set.

	BP	FBP	CG	BFGS	LM	LSB	RS-LSB
Train NMSE	0.04 (0.004)	0.07 (0.01)	0.04 (0.002)	0.15 (0.14)	0.3 (0.2)	0.26 (0.05)	0.08 (0.02)
Test NMSE	0.05 (0.005)	0.07 (0.01)	0.05 (0.002)	0.15 (0.13)	0.26 (0.19)	0.25 (0.03)	0.06 (0.01)
No Iters.	395 (120)	153 (57)	30 (5.4)	13.6 (7.9)	16 (5.6)	6.2 (1.7)	10.9 (4)
Load % BP	100%	39%	85%	37%	187%	4%	8%

Notes: We show classification error (mean values and standard deviations, in parentheses) corresponding to 10 runs. The average number of epochs to converge and the (measured) computational cost have also been included, the latter expressed in percentage with respect to BP load.

5 Computational Considerations

In order to evaluate the speed gain of block learning methods with respect to more classical approaches (BP, FBP, CG, BFGS, and LM), we have registered the number of floating operations per epoch consumed by every algorithm and the number of epochs needed to converge. In Tables 1 through 3 we show the computational load with respect to BP. It can be observed that from this point of view, block methods always appear to be advantageous, RS-LSB being from 3 to 13 times faster than the fastest competitor (among those with comparable final performance)—the moderate computational excess needed by RS-LSB with respect to LSB being justified by the improved quality of the solutions.

Although the numbers collected in Tables 1 through 3 reflect actual computational cost, it is convenient to give approximate expressions for computational load per epoch for every one of the algorithms under test. It is rather complicated to derive exact analytical expressions because of the numerous operations, and therefore we have obtained empirical expressions assuming computational cost linear in the number of weights for the BP, FBP, CG, and BFGS cases, quadratic in the number of weights for the LM case and quadratic in the number of neurons in LSB and RS-LSB; the free (multiplying) parameters are adjusted afterward by LS fitting in a logarithmic scale (i.e., computational loads are first converted to a log scale in order to obtain a similar fitting error for problems with different complexity). This estimation gives, as shown in Figure 3, estimates (-o-) close to measured numbers (-*-) (the analytical expressions are also included in Figure 3). The accuracy of these expressions serves to confirm that, as we had guessed, BP, FBP, CG, and BFGS behave linearly in the number of weights (N_W), LM is quadratic in N_W, and block training methods are quadratic in the number of neurons (N_N).

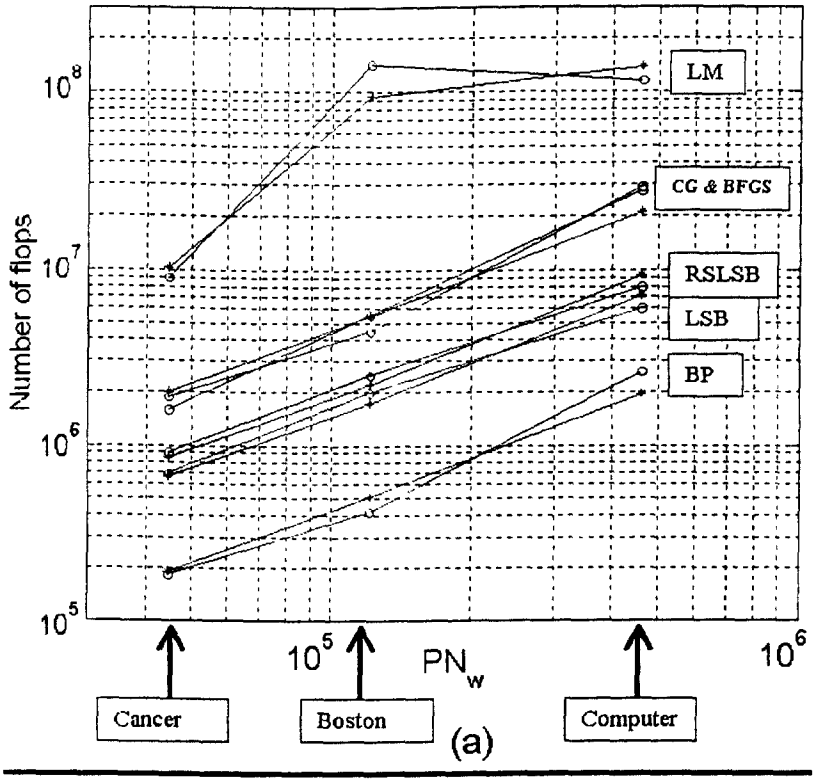

Figure 3: (a) Computational load (number of floating operations per epoch) as a function of PN_w for the three data sets under study (o) and empirical estimation (*). The estimates are accurate enough. (b) Estimated load expressions for every algorithm where N_W is the number of weights in the network (including biases) and N_N the number of neurons (including the input layer).

6 Some Theoretical Insights

6.1 Points of Tangency with Second-Order Searching. The basic local technique to find the minimum of a nonquadratic error function using

second-order information is the well-known Newton's method. It accepts that the error function $E(\mathbf{w})$ can be locally approximated by a second-order polynomial and computes Newton step $\Delta\mathbf{w}_N$ (such that $E(\mathbf{w} + \Delta\mathbf{w}_N)$ is minimal), solving

$$\mathbf{H}(\mathbf{w})\Delta\mathbf{w}_N = -\nabla E(\mathbf{w}) \tag{6.1}$$

where $\nabla E(\mathbf{w})$ is the gradient vector and $\mathbf{H}(\mathbf{w})$ the Hessian matrix. In our case, vector $\mathbf{w}$ represents one of the columns of $\mathbf{W}^{(k)}$; we are solving a local minimization problem at every neuron.

If we consider the error term $\mathbf{r}(\mathbf{w}) = tanh(\mathbf{Ow}) - \mathbf{t}$, then $E(\mathbf{w}) = \frac{1}{2}\mathbf{r}(\mathbf{w})^T\mathbf{r}(\mathbf{w})$, the gradient vector is

$$\nabla E(\mathbf{w}) = J(\mathbf{w})^T\mathbf{r}(\mathbf{w}) \tag{6.2}$$

and the Hessian matrix can be reasonably approximated (Battiti, 1992) using the Jacobian matrix $\mathbf{J}(\mathbf{w}) = \partial\mathbf{r}(\mathbf{w})/\partial\mathbf{w}$,

$$\mathbf{H}(\mathbf{w}) \simeq \mathbf{J}(\mathbf{w})^T\mathbf{J}(\mathbf{w}). \tag{6.3}$$

We obtain the Gauss-Newton step by solving equation 6.1 using these values,

$$\Delta\mathbf{w}_{GN} = -(\mathbf{J}(\mathbf{w})^T\mathbf{J}(\mathbf{w}))^{-1}\mathbf{J}^T(\mathbf{w})\mathbf{r}(\mathbf{w}). \tag{6.4}$$

such that strong similarities (and also qualitative differences) appear between the Gauss-Newton method and the RS-LSB algorithm (with $\beta = 1$ in equation 3.5. To facilitate their comparison, we obtain the analogous RS-LSB step by conveniently rewriting equation 3.4 and minimizing with respect to the weight update[10] $\Delta\mathbf{w}_{RS-LSB}$,

$$\min_{\Delta\mathbf{w}_{RS-LSB}} \left\|\mathbf{O}_s\mathbf{w} + \mathbf{O}_s\Delta\mathbf{w}_{RS-LSB} - \mathbf{q}_s\right\|_2, \tag{6.5}$$

and, if we use the Moore-Penrose generalized inverse to solve equation 6.5, we obtain

$$\begin{aligned}\Delta\mathbf{w}_{RS-LSB} &= -(\mathbf{O}_s^T\mathbf{O}_s)^{-1}\mathbf{O}_s^T(\mathbf{O}_s\mathbf{w} - \mathbf{q}_s) \\ &= -(\mathbf{O}_s^T\mathbf{O}_s)^{-1}\mathbf{O}_s^T\mathbf{r}_{RS-LSB}(\mathbf{w}).\end{aligned} \tag{6.6}$$

[10] Note that block learning methods can be formulated in direct or incremental form, the former yielding a new set of weights in every step (solving equation 3.4) and the latter obtaining weight increments (solving equation 6.5) to be added to present weight values, but both formulations leading to identical solutions.

It is straightforward to verify that $\mathbf{O}_s = \mathbf{J}(\mathbf{w})$ (using equation 3.5, with $\beta = 1$) such that, to compare RS-LSB and Gauss-Newton methods, only the following residual terms have to be considered:

$$\mathbf{r}_{GN}(\mathbf{w}) = \mathbf{r}(\mathbf{w}) = tanh(\mathbf{Ow}) - \mathbf{t} \tag{6.7}$$

$$\mathbf{r}_{RS-LSB}(\mathbf{w}) = (\mathbf{O}_s\mathbf{w} - \mathbf{q}_s) = diag(\mathbf{s})(\mathbf{Ow} - \mathbf{q}). \tag{6.8}$$

The multiplication by sensitivity in equation 6.8 emphasizes those error terms $\mathbf{o}_p^T\mathbf{w} - \mathbf{q}_p$ with low value for $|\mathbf{o}_p^T\mathbf{w}|$ because s_p becomes negligible when $|\mathbf{o}_p^T\mathbf{w}| \to \infty$, thereby yielding the localization properties mentioned in section 3. Thus, the RS-LSB algorithm, when considered as a whole, combines an accurate local method to adjust the weights of every neuron with the layerwise training approach that decomposes the global optimization problem into smaller LS problems.

When we use general expression 3.6 for the sensitivity, which means using $\beta \neq 1$ and the scaling procedure, RS-LSB and Gauss-Newton methods are no longer directly comparable, but the qualitative behavior of RS-LSB is maintained.

6.2 Relationship with Regularization Techniques. Taking advantage of the LS formulation, it would be easy to include an additional regularization term $\alpha \|\mathbf{w}\|_2$ in equation 3.4, such that the minimization becomes

$$\min_{\mathbf{w}} \left\{ \left\| \mathbf{O}_s\mathbf{w} - \mathbf{q}_s \right\|_2 + \alpha \|\mathbf{w}\|_2 \right\}, \tag{6.9}$$

which leads to well-known regularization techniques that either add a small value (α) to the diagonal of $\mathbf{O}_s^T\mathbf{O}_s$ or discard those singular values smaller than α. We have observed through experimentation that this extra regularization mechanism sometimes excessively limits the approximation capabilities of the network (being also highly sensitive to the choice of α). As the RS-LSB algorithm already relies on minimal norm solutions, it generally obtains solutions with good generalization, and we found no reason to include this additional mechanism. (Further comments on regularization are included in section 6.4.)

6.3 Relationship with Natural Gradient. It has been recently shown that the parameter space of perceptrons is not Euclidean but has a Riemannian structure (Amari, 1998). Thus, the ordinary gradient does not provide the steepest direction, which is given by the natural (contravariant) gradient. Preliminary results (Amari, 1998) show that the performance of this method is good, releasing BP from many of its convergency problems.

The Riemannian steepest descent direction of $E(\mathbf{w})$ is

$$-\widetilde{\nabla}E(\mathbf{w}) = \mathbf{G}^{-1}(\mathbf{w})(-\nabla E(\mathbf{w})), \tag{6.10}$$

where $\mathbf{G}(\mathbf{w})$ is the Riemannian metric tensor. If we formulate every neuron using a statistical model $t = tanh(\mathbf{o}^T\mathbf{w}) + n$, n being an $N(0, \sigma_n^2)$ random variable and $p_{\mathbf{o}}(\mathbf{o}^T)$ being the distribution of inputs, the input-output joint probability is

$$p(\mathbf{o}^T, t; \mathbf{w}) = p_{\mathbf{o}}(\mathbf{o}^T)\frac{1}{\sqrt{2\pi}\sigma_n}\exp\left\{-(tanh(\mathbf{o}^T\mathbf{w})-t)^2/\sigma_n^2\right\}, \tag{6.11}$$

and the geometry of the space is given by the Fisher information matrix

$$\mathbf{G}(\mathbf{w}) = E\left\{\left(\frac{\partial \log p(\mathbf{o}^T, t; \mathbf{w})}{\partial \mathbf{w}}\right)\left(\frac{\partial \log p(\mathbf{o}^T, t; \mathbf{w})}{\partial \mathbf{w}}\right)^T\right\}, \tag{6.12}$$

which becomes (Amari, 1998)

$$\mathbf{G}(\mathbf{w}) = \frac{1}{\sigma_n^2}E\{(tanh'(\mathbf{o}^T\mathbf{w}))^2\mathbf{o}\mathbf{o}^T\}. \tag{6.13}$$

The expectation in equation 6.13 has to be estimated using an average over the finite training set of P patterns:

$$\begin{aligned}\widehat{\mathbf{G}}(\mathbf{w}) &= \frac{1}{\sigma_n^2 P}\sum_{p=1}^{P}(tanh'(\mathbf{o}_p^T\mathbf{w})\mathbf{o}_p)(tanh'(\mathbf{o}_p^T\mathbf{w})\mathbf{o}_p)^T \\ &= \frac{1}{\sigma_n^2 P}\mathbf{J}^T(\mathbf{w})\mathbf{J}(\mathbf{w}).\end{aligned} \tag{6.14}$$

Block methods assume normally distributed errors in state-space and, in particular, RS-LSB uses the weighted error terms $e_p = s_p(\mathbf{o}_p^T\mathbf{w} - q_p)$ (see equation 3.1) such that the new joint probability function is

$$p(\mathbf{o}^T, t; \mathbf{w}) = p_{\mathbf{o}}(\mathbf{o}^T)\frac{1}{\sqrt{2\pi}\sigma_n}\exp\left\{-(diag(\mathbf{s})(\mathbf{o}^T\mathbf{w} - q))^2/\sigma_n^2\right\}. \tag{6.15}$$

It is also easy to find the Fisher information matrix in this case:

$$\widehat{\mathbf{G}}_{RS-LSB}(\mathbf{w}) = \frac{1}{\sigma_n^2 P}\sum_{p=1}^{P}(s_p\mathbf{o}_p)(s_p\mathbf{o}_p)^T = \frac{1}{\sigma_n^2 P}\mathbf{O}_s^T\mathbf{O}_s, \tag{6.16}$$

which is completely equivalent to equation 6.12, given the relationship $\mathbf{J}(\mathbf{w}) = \mathbf{O}_s$. We observe that the RS-LSB update, equation 6.6, contains a factor proportional to $\widehat{\mathbf{G}}_{RS-LSB}^{-1}(\mathbf{w})$. Therefore, it truly takes into account the nonorthonormal geometry of the space, benefiting from the good convergence properties attributed to natural gradient. On the other hand, note that in LSB, the estimated Fisher matrix is $\widehat{\mathbf{G}}_{LSB}(\mathbf{w}) = \frac{1}{\sigma_n^2 P}\mathbf{O}^T\mathbf{O}$ (it lacks the $tanh'(\mathbf{o}_p^T\mathbf{w})$ terms), which is clearly different from equation 6.12.

6.4 On Implicit Sample Selection and Regularization. Sample selection is a well-known strategy for improving the efficiency of neural networks. The first proposal for introducing sample selection (Munro, 1992) demonstrated this, as have many other modifications (Cachin, 1994), some using other cost objectives (Telfer & Szu, 1994), according to the discussion in Cid-Sueiro, Arribas-Sánchez, Urbán-Muñoz, and Figueiras-Vidal (1999).

Recently, powerful learning architectures, the support vector machines (SVM), have appeared as a result of formulations including generalization objectives (Vapnik, 1995; Vapnik, Golowich, & Smola, 1997; Schölkopf et al., 1997). These architectures also include (implicit) sample selection.

Our approach has some differences from these. Specifically, the selection is carried out independently at every neuron, and none of the patterns is (fully) discarded at any point. These characteristics can be advantageous. For example, the second approach seems to be a softer way of doing things, more adequate when training data may include misclassified patterns or outliers.

Finally, note that in the above references for SVM (and also in Bartlett, 1997), good performance is attributed to controlling the weight sizes (as in the first proposal of regularization for pruning purposes; (Hinton, 1987). In this sense, since LS minimizations yield minimum-norm solutions, LSB, and in particular RS-LSB, will also follow this principle.

7 Conclusions

We have seen that block training methods constitute a powerful approach to MLP training, promising a great reduction in computational effort with respect to other techniques. Some problems arise in direct implementations such as LSB, mainly concerning the final error and generalization capability. We have proposed and developed a reduced sensitivity approach, which has proved to be advantageous with respect to other well-known training algorithms, as shown by means of several real-world examples.

Our approach is also related to some other efficient training techniques, such as second-order approaches or natural gradient descent. Additionally, reducing the sensitivity of the model to some of the training patterns can be considered an implicit sample selection strategy, which also yields benefits for the global learning process.

Extending this approach to other scenarios, such as recurrent neural networks, could yield simpler and practically applicable designs, one of the limitations of this type of neural networks. To apply other cost functions and additional sample selection strategies are also interesting tasks, as are analyzing sensitivities and variable selection under these new training procedures.

Acknowledgments

This work has been partially supported by the III National R & D Plan of the Spanish Government (CICYT), under grant TIC96-0500-C10-03.

References

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation, 10,* 251–276.

Azimi-Sadjadi, M., & Liou, R. (1992). Fast learning process of multilayer neural networks using recursive least squares method. *IEEE Trans. on Signal Processing, 40,* 446–450.

Bartlett, P. (1997). For valid generalization, the size of the weights is more important than the size of the network. In M. M. Mozer, M. Jordan, T. Petsche (Eds.), *Advances in neural information processing systems, 9* (pp. 134–140). San Mateo, CA: Morgan Kaufmann.

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

Biegler-König, F., & Bärmann, F. (1993). A learning algorithm for multilayered neural networks based on linear least squares problems. *Neural Networks, 6,* 127–131.

Cachin, C. (1994). Pedagogical pattern selection strategies. *Neural Networks, 7,* 175–181.

Cid-Sueiro, J., Arribas-Sánchez, I., Urbán-Muñoz, S., & Figueiras-Vidal, A. R. (1999). Cost functions to estimate "a posteriori" probabilities in multi-class problems. *IEEE Trans. on Neural Networks, 10*(3), 645–656.

DiClaudio, E. D., Parisi, R., & Orlandi, G. (1993). LS-based training algorithms for neural networks. In C. Kamm & G. Kuhn (Eds.), *Neural networks for signal processing III* (pp. 22–29). New York: IEEE Press.

Deller, J., & Hunt, S. (1992). A simple "linearized" algorithm which outperforms back-propagation. In *Proc. Intl. Joint Conf. on Neural Networks* (Vol. 3, pp. 133–138).

Douglas, S., & Meng, T. (1991). Linearized least-squares training of multilayer feedforward neural networks. In *Proc. Intl. Joint Conference on Neural Networks* (Vol. 1, pp. 307–312).

Hinton, G. (1987). Learning traslation invariant recognition in massively parallel networks. In J. Bakker, A. Nijman, & P. Treleaven (Eds.), *Proc. PARLE Conference on Parallel Architectures and Languages Europe* (pp. 1–13). Berlin: Springer-Verlag.

Munro, P. (1992). Repeat until bored: A pattern selection strategy. In J. Moody, S. Hanson, & R. Lippmann (Eds.), *Advances in neural information processing systems, 4* (pp. 1001–1008). San Mateo, CA: Morgan Kaufmann.

Pethel, S., & Bowden, C. (1992). Neural network applications to nonlinear time series analysis. In *Proc. Intl. Conf. on Industrial Electronics, Control, Instrumentation and Automation, Power Electronics and Motion Control* (Vol. 2, pp. 1094–1098).

Prechelt, L. (1994). *Proben1—A set of neural network benchmark problems and benchmarking rules* (Tech. Rep. No. 21/94). Fakultat fur Informatik, Universitat Karlsruhe, Karlsruhe, Germany. Available online via anonymous FTP: ftp.ira.uka.de, /pub/papers/techreports /1994/1994-21.ps.Z.

Rao, C., & Toutenburg, H. (1995). *Linear models: Least squares and alternatives.* New York: Springer-Verlag.

Rohwer, R. (1991). The moving targets training algorithm. In D. Touretzky (Ed.), *Advances in neural information processing systems, 1* (pp. 558–565). San Mateo, CA: Morgan Kaufmann.

Schölkopf, B., Sung, K., Burges, C., Girosi, F., Nigoyi, P., Poggio, T., & Vapnik, V. (1997). Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Trans. on Signal Processing, 45*, 2758–2765.

Song, J., & Hassoun, M. (1990). Learning with hidden targets. In *Proc. Intl. Joint Conf. on Neural Networks* (Vol. 3, pp. 93–98).

Telfer, B., & Szu, H. (1994). Energy functions for minimizing misclassification error with minimum-complexity networks. *Neural Networks, 7*, 809–818.

Vapnik, V. (1995). *The nature of statistical learning theory.* New York: Springer-Verlag.

Vapnik, V., Golowich, S., & Smola, A. (1997). Support vector method for function approximation, regression, estimation and signal processing. In M. Mozer, M. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems, 9* (pp. 281–287). Cambridge, CA: MIT Press.

Wang, G.-J., & Chen, C.-C. (1996). A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures. *IEEE Trans. Neural Networks, 7*, 768–775.

Received June 26, 1998; accepted May 15, 1999.