[15] S. Adwankar and R. N. Banavar, "A recurrent network for dynamic system identification," *Int. J. Syst. Sci.*, vol. 28, no. 12, pp. 1239–1250, 1997.
[16] F. Garces, V. Becerra, C. Kambhampati, and K. Warwick, *Strategies for Feedback Linearization: A Dynamic Neural Network Approach*. New York: Springer-Verlag, 2003.
[17] A. Poznyak, E. Sanchez, and W. Yu, *Differential Neural Networks for Robust Nonlinear Control*, Singapore: World Scientific, 2001.
[18] K. Funahashi and Y. Nakamura, "Approximation of dynamical-systems by continuous-time recurrent neural networks," *Neural Netw.*, vol. 6, no. 6, pp. 801–806, 1993.
[19] M. Kimura and R. Nakano, "Learning dynamical systems by recurrent neural networks from orbits," *Neural Netw.*, vol. 11, no. 9, pp. 1589–1599, 1998.
[20] T. W. S. Chow and X. D. Li, "Modeling of continuous time dynamical systems with input by recurrent neural networks," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 47, pp. 575–578, 2000.
[21] L. Ljung, *System Identification: Theory for the User*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.
[22] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. New York: Wiley, 1987.
[23] O. Nelles, *Nonlinear System Identification*. Berlin, Germany: Springer-Verlag, 2001.
[24] S. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.
[25] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Crarendon Press, 1995.
[26] *The Handbook of Brain Theory and Neural Networks*, M. Arbib, Ed., MIT Press, Cambridge, MA, 1999.
[27] V. Becerra, J. Calado, P. Silva, and F. Garces, "System identification using dynamic neural networks: Training and initialization aspects," in *Proc. IFAC World Congr.*, Barcelona, Spain, 2002, pp. 235–240.
[28] H. Khalil, *Nonlinear Systems*. New York: Macmillan, 1992.

Improvement of the Neighborhood Based Levenberg–Marquardt Algorithm by Local Adaptation of the Learning Coefficient

A. Toledo, M. Pinzolas, J. J. Ibarrola, and G. Lera

***Abstract*—In this letter, an improvement of the recently developed neighborhood-based Levenberg–Marquardt (NBLM) algorithm is proposed and tested for neural network (NN) training. The algorithm is modified by allowing local adaptation of a different learning coefficient for each neighborhood. This simple add-in to the NBLM training method significantly increases the efficiency of the training episodes carried out with small neighborhood sizes, thus, allowing important savings in memory occupation and computational time while obtaining better performance than the original Levenberg–Marquardt (LM) and NBLM methods.**

***Index Terms*—Learning algorithms, neural networks (NNs).**

Manuscript received February 18, 2004; revised July 19, 2004. This work was supported in part by the Spanish Comisión Interministerial de Ciencia y Tecnología under Grant DPI2000-04150-P403.

A. Toledo is with the Department Tecnología Electrónica, Universidad Politécnica de Cartagena, Cartagena 30202, Spain (e-mail: Ana.Toledo@upct.es).
M. Pinzolas and J. J. Ibarrola are with the Department Ingeniería de Sistemas y Automática, Universidad Politécnica de Cartagena, Cartagena 30202, Spain (e-mail: Miguel.Pinzolas@upct.es; Juliojose.Ibarrola@upct.es).
G. Lera is with the Department Automática y Computación, Universidad Pública de Navarra, Pamplona 31006, Spain (e-mail: Galera@unavarra.es).

Digital Object Identifier 10.1109/TNN.2005.849849

I. Introduction

The Levenberg–Marquardt (LM) algorithm [1]–[3] has been one of the learning methods most successfully imported from the optimization field to the neural network (NN) training task. However, a large number of tuning parameters (weights) makes this algorithm inefficient even when the network to be trained is of regular size.

The core of the LM algorithm is the calculus and inversion, at each training cycle, of an approximation to the Hessian. This requires storing and then inverting a matrix with n^2 elements, n being the number of adaptable parameters. In the neighborhood-based Levenberg–Marquardt (NBLM) algorithm proposed in [4] and [5], a modification of LM method, consisting in taking only a subset (neighborhood) $n_1 < n$ of the adjustable weights at each iteration, was developed and tested. In [5], the experimental evidence showed that, by performing a LM step on a single neighborhood at each training iteration, not only significant savings in memory occupation and computing effort were obtained, but also the overall performance of the LM method was improved upon. The NBLM has been extensively tested in real-world applications, both to train offline [6], [7] and online [8], NNs and neurofuzzy systems, with excellent results.

In this letter, the local nature of NBLM is exploited to further improve the learning results. It is shown that, by locally adapting a learning coefficient of the LM method for each neighborhood, statistically better results are obtained. To test the performance of this modification, several experiments have been carried out by using both the testing function employed in [5] and randomly generated functions. The experimental results show that this local adjustment makes NBLM behave better than LM even for very small neighborhoods, thus, allowing important savings in memory and computational time.

II. Learning Algorithm

In the original LM algorithm, the Jacobian $\vec{j}_X$ of the error with respect to the weight and bias variables $\vec{X}$ is calculated. Then, an approximation of the Hessian $(\mathbf{jj})$ is created by the outer product approximation

$$\mathbf{jj} = \bar{j}_X^T \cdot \bar{j}_X. \tag{1}$$

In the LM method, the change $d\bar{X}$ in vector $\bar{X}$ is obtained by solving

$$\begin{aligned}\bar{j}_e &= \bar{j}_X \cdot E\\ \alpha &= \mathbf{jj} + \mathbf{I}\cdot\lambda\\ \alpha \cdot d\bar{X} &= -\bar{j}_e\end{aligned} \tag{2}$$

where E are all errors and $\mathbf{I}$ is the identity matrix.

The adaptive value λ is increased by λ_{INC} (typically $\lambda_{\mathrm{INC}} = 10$) until the previous change results in a reduced error value. The change is then made to the network and λ is decreased by λ_{DEC} (typically $\lambda_{\mathrm{DEC}} = 0.1$). This learning coefficient λ balances the behavior of the method between a second- and first-order one. If it is very small, the alpha matrix becomes an approximation to the Hessian, and the method becomes the inverse-Hessian method. If its value is big enough, the method becomes analogous to steepest descent.

It can be easily seen from (2) that, if m is the number of weights, it is necessary to calculate and store the m^2 elements of the alpha matrix at each LM cycle, and find its inverse, which needs about m^3 operations to be carried out. This makes the LM method very expensive both in terms of memory and number of operations when the network to be trained has a significant number of adaptive weights.

NBLM is a modification of the LM algorithm, based upon a reduction in the number of weights adapted at each iteration [4], [5]. Inspired by the way the biological brains seem to act (different groups of neurons perform different tasks), the network is divided into several neighborhoods that act as independent learning units. This allows applying the LM method to one neighborhood in each step of the algorithm, thus, decreasing the number of operations and the amount of memory required.

In spite of the local character of the original NBLM method, the learning coefficient λ is globally adapted; that is, only one λ coefficient is considered to make the balance between the first- or second-order shape of the error surface at each learning cycle. However, as applying the LM method to one neighborhood in each step is equivalent to performing an optimization in a reduced subspace of the weight space at each learning cycle, it seems clear that a better performance can be expected if different lambda coefficients are used for each neighborhood considered.

With this modification, the improved NBLM algorithm can be summarized as follows.

1) Define the network structure, assign initial weights and define neighborhoods [4], [5] (or as in Section III).
2) Assign one initial value λ_i of the learning coefficient for each neighborhood considered.
3) Randomly select the neighborhood to be trained.
4) Perform an LM cycle on the selected neighborhood [see (2)], and evaluate the network error with the new weights. Stop the learning process if one of the termination conditions (elapsed time, maximum number of epochs, ...) is met.
5) If the error has decreased, decrease the value of the corresponding λ_i, and go back to step 3).
6) Otherwise, increase the value of λ_i, and go back to step 4).

In what follows, we refer to the previous algorithm as the modified NBLM.

III. Experimental Results

All the software used in the experiments has been developed in Matlab. While comparing against the original LM training method, the LM implementation chosen is that of the Matlab's Neural Network toolbox.

As in [5], our goal was to study the evolution of the training error with different neighborhood sizes. As the error evolution strongly depends on the initial weight selection, in each experiment we carried out a set of 100 training episodes, initializing all the weights from a uniformly distributed, symmetric random number generator giving values in the $[-1, 1]$ interval. Then the mean of the error evolutions is considered against time. Comparisons were performed with both LM and standard NBLM methods, taking several neighborhood sizes. The initial values for all the λ_i were chosen to be 10^{-3}, $\lambda_{\rm INC}$, and $\lambda_{\rm DEC}$ were 10 and 10^{-1}, respectively. As our consideration is that only a subspace in the weight space is trained in each training cycle, in the modified algorithm neighborhoods will be selected among the total vector of weights, instead of being composed of neuron groups. In this letter, the neighborhoods have been formed by cutting the total weight vector into equally sized chunks. This maintains the biological sense of neighborhood, as weights belonging to the same chunk are likely from adjacent neurons. Experiments made by assigning weights randomly to a neighborhood have not shown significant differences, except in the case of Test 3. This will be commented on with more detail ahead in the text.

Another critical question is the selection of the training set of examples and of the network structure. In this letter, results coming from three different tests are shown. The first set of examples and network structure are the ones used in [4] and [5]. This trial provides a baseline for comparison with the NBLM method proposed in [5]. On the second test, the examples have been extracted from randomly generated functions, in order to discard, as far as possible, structural influence of the training set on the results. In this second test, the size of the network used is greatly increased, and that will allow probing how the method scales with the number of weights. Lastly, a particular case is presented, to show how when dealing with fine tuning, the performance is even better.

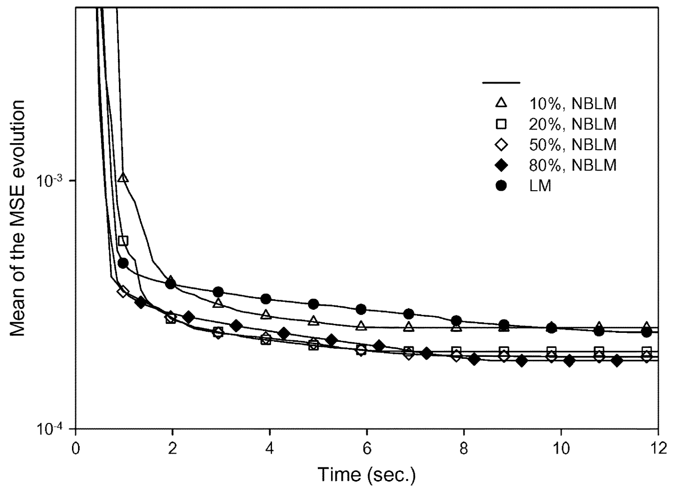

Fig. 1. Mean of the MSE evolution against time for different neighborhood sizes (NBLM Test 1).

A. Test 1

As in [4] and [5], the first function is given by

$$f(x) = \frac{\sin(0.3 \cdot x)}{x}, \quad \text{in } x \in [1, 80]. \tag{3}$$

The NN is composed of a hidden layer with 30 neurons (with hyperbolic tangent activation function $a(x) = \tanh(x)$) and one linear neuron in the output layer.

The examples needed to train the networks are obtained by uniformly sampling the function in the integer values of the independent variable on the specified domain. One hundred training episodes were run for each of four different neighborhood sizes, both with NBLM and the proposed method, and for standard LM. Then, the mean of the 100 evolutions is computed and displayed. In the figures, the proposed method is named modified NBLM.

The results obtained for the function given by (3) are shown in Figs. 1–3. In all the figures, neighborhoods are labeled by the percentage of weights considered for training at each iteration.

In Fig. 1, it can be seen that the best results [lowest mean squared error (MSE)] for the original NBLM are obtained for neighborhoods containing 80% of the weights. Only for the smallest neighborhood the performance of the standard LM is better.

That changes with the proposed algorithm. As Fig. 2 shows, the method behaves better than standard LM for every one of the considered neighborhood sizes. It can be observed that the performance of the 20% neighborhood is significantly increased with the proposed modification, reaching lower error values faster than LM and NBLM. Note that, when compared against LM, the use of a 20% neighborhood implies utilizing only a 4% of the memory storage, as the memory required grows as a square of the number of weights. This allows efficient training of very large networks that will be otherwise physically impossible to train with second-order methods due to memory requirements. The modified NBLM algorithm, even with a 10% neighborhood

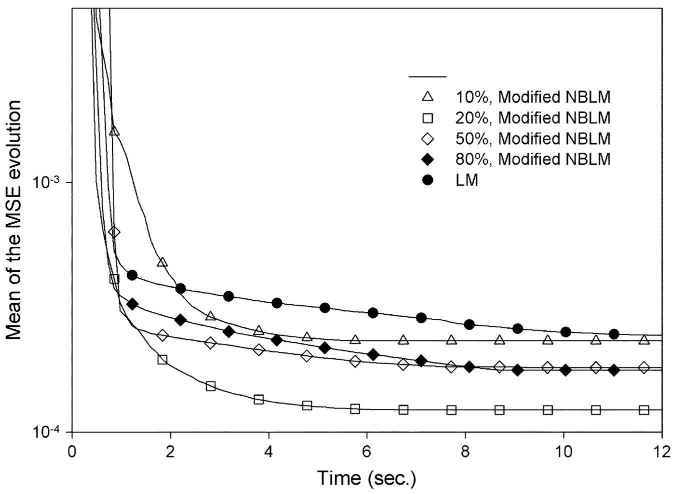

Fig. 2. Mean of the MSE evolution against time for different neighborhood sizes (modified NBLM Test 1).

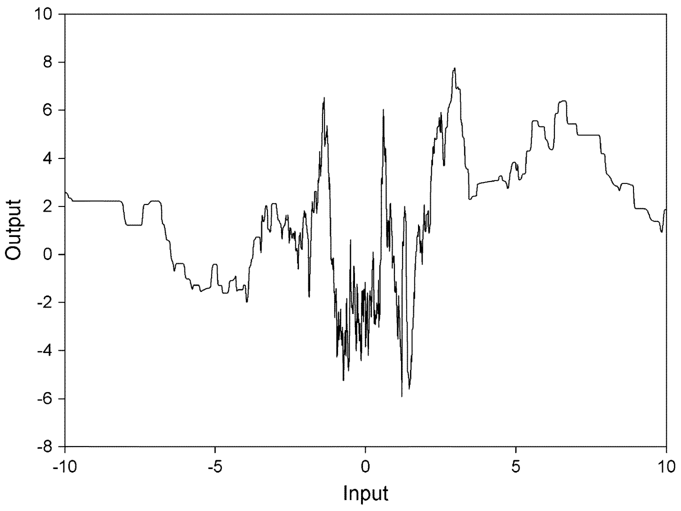

Fig. 4. Randomly generated function for Test 2.

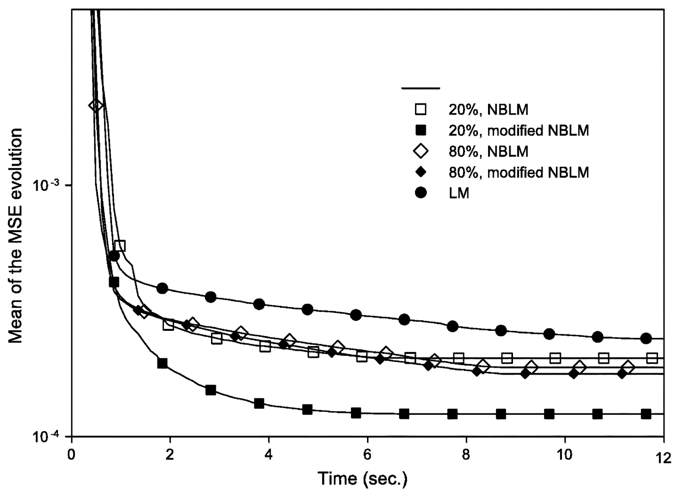

Fig. 3. Comparison of the MSE evolution against time for different neighborhood sizes (NBLM and modified NBLM Test 1).

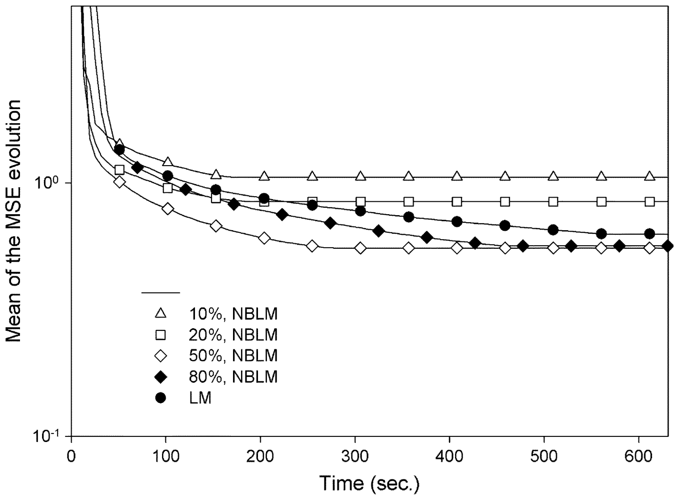

Fig. 5. Mean of the MSE evolution against time for different neighborhood sizes (NBLM Test 2).

achieves better performance than LM, besides enabling a 99% savings on the memory requirements.

In Fig. 3, the advantages of the proposed modification became manifest. For the sake of simplicity, only two neighborhood sizes have been compared, maintaining the LM error evolution as a reference. As expected, it can be seen that the smaller the neighborhood size, the greater the improvements (up to a 40% lower final error for the 20% neighborhood, only 5, 6% for the 80% one), due to their more local character.

B. Test 2

In order to test if the characteristics of the function from which the examples are extracted has an influence on the behavior of the algorithm, one hundred random functions have been generated in the following ways:

1) a single-input–single-output (SISO) NN is generated, with 900 hyperbolic tangent neurons in the hidden layer and one linear neuron in the output;
2) the weights of the hidden layer are initialized randomly in the interval $[-1000, 1000]$ to assure that the generated function is spiky enough;
3) weights of the output neuron are randomly initialized inside the interval $[-1, 1]$.

For each experiment, a different network is created and 2000 examples are generated by giving to that network uniformly distributed inputs in the $[-10, 10]$ interval. The form of the functions generated by this method is similar to that of Fig. 4.

The training NN is composed of a hidden layer with 300 neurons (with hyperbolic tangent activation function) and one linear neuron in the output layer. Again, one hundred training episodes were run for each of four different neighborhood sizes, both with NBLM and the proposed modified NBLM method and for standard LM.

The means of the error evolutions are shown in Figs. 5–7. As in the previous subsection, neighborhoods are labeled by the percentage of weights considered for training at each iteration. In Fig. 5, the means of the error evolution against time for NBLM method are shown. This time, only the 50% and 80% lines stay below the LM one in the long run. In spite of this, memory savings are still very significant (75% and 36%, respectively).

The corresponding error evolutions for the proposed method are represented in Fig. 6. It can be seen that all neighborhood sizes but 10% perform better than LM. The improvements with respect to unmodified NBLM can be better observed in Fig. 7. Again, performance is improved for all neighborhood sizes, especially for the smallest ones. The most significant reduction on the error corresponds to the 20% neighborhood (up to a 34% decrease). The 50% neighborhood gives

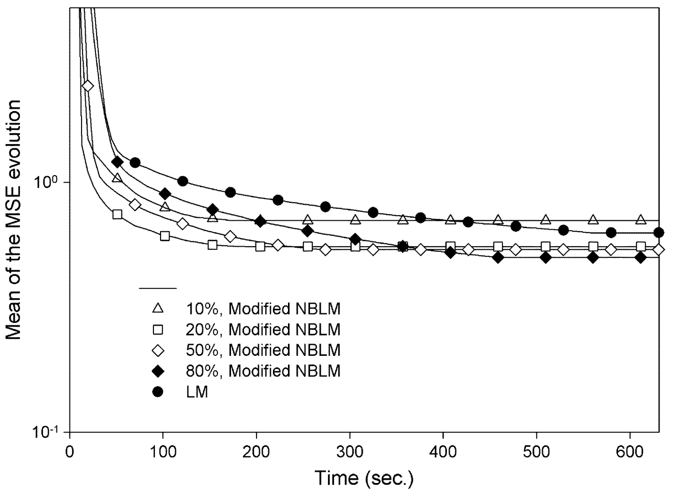

Fig. 6. Mean of the MSE evolution against time for different neighborhood sizes (modified NBLM Test 2).

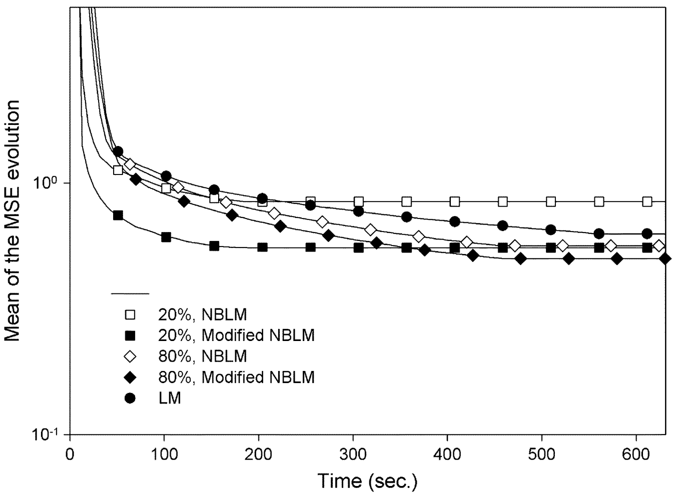

Fig. 7. Comparison of the MSE evolution against time for different neighborhood sizes (NBLM and modified NBLM Test 2).

the least improvement in the performance (only a 3% decrease in the error).

C. Test 3

To further illustrate the advantages of the local adaptation strategy, experiments have been performed involving 8000 examples taken from the function in (3). The NN employed has a hidden layer of 300 hyperbolic tangent neurons. Such a large number of neurons allows for a very accurate reproduction of the original function. As the function is not contaminated by noise, overfitting is not a problem in this case, in which only the precision of the training algorithm is being examined.

The means of the one hundred error evolutions for each neighborhood size can be observed in Figs. 8–10. It can be seen from the figures that the proposed method greatly improves the training performance. A final error nearly two magnitude orders lower than both the LM and NBLM algorithms is achieved for the 20% neighborhood size. This behavior can be explained considering that, when errors are very small, the local behavior of the proposed algorithm allows better adaptation through the specialization of some parts of the network in reproducing zones of the objective function. This is supported by the fact that, as mentioned previously, experiments made by assigning weights randomly to a neighborhood (and therefore losing the concept of biological neighborhood) could not reproduce the results of Test 3, while for Tests 1 and 2 they offered similar results.

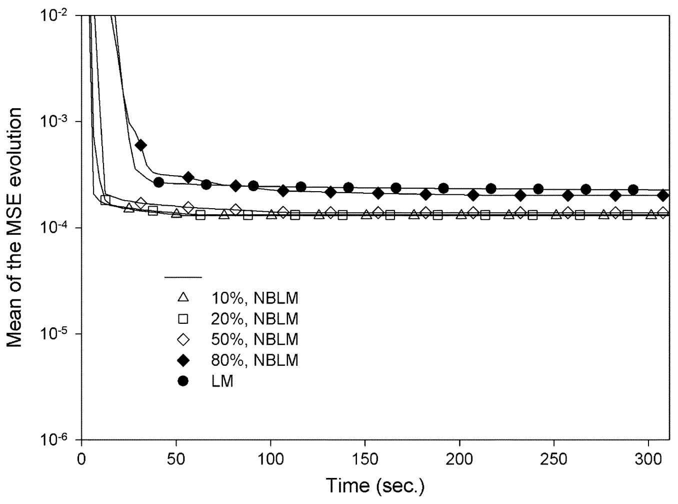

Fig. 8. Mean of the MSE evolution against time for different neighborhood sizes (NBLM Test 3).

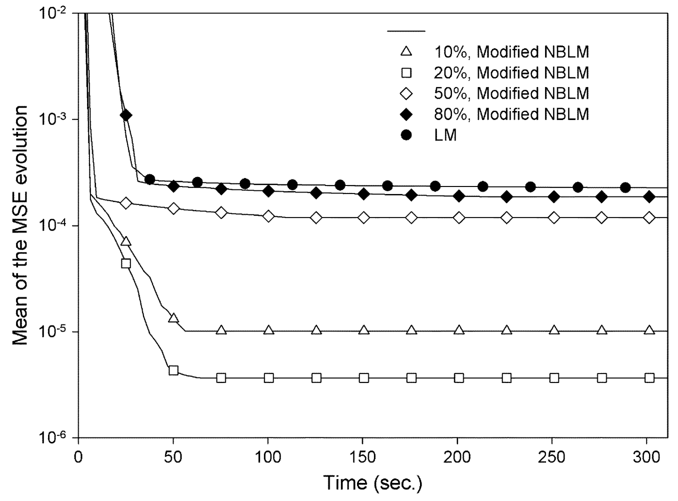

Fig. 9. Mean of the MSE evolution against time for different neighborhood sizes (modified NBLM Test 3).

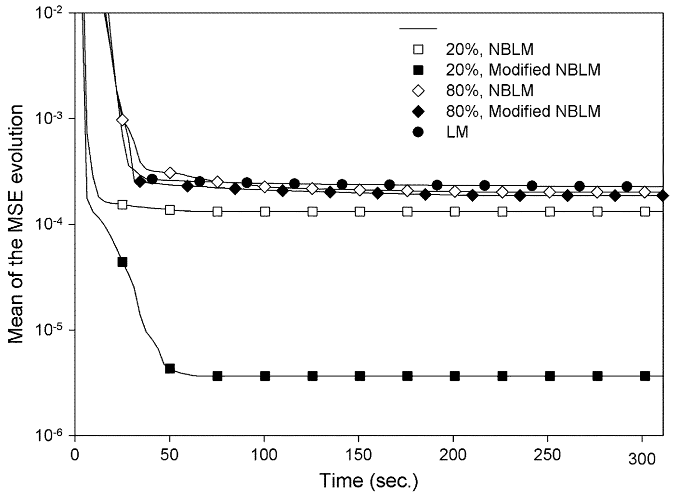

Fig. 10. Comparison of the MSE evolution against time for different neighborhood sizes (NBLM and modified NBLM Test 3).

IV. Conclusion

In this letter, an enhancement of the NBLM algorithm is proposed. It is shown that, by locally adapting one learning coefficient of the NBLM for each neighborhood, significant improvements on the performance of the method are achieved. The suggested modification requires only minor changes in the original algorithm, and reinforces the local character of the NBLM.

With the proposed local adaptation, the modified NBLM achieves better performance than the LM method even for very small neighborhood sizes. This allows very large NNs to be efficiently trained in a fraction of the time LM would require, still reaching lower error rates. Moreover, it makes possible to retain the efficiency of that method in those situations where the application of the original algorithm was impractical.

Acknowledgment

The authors would like to thank the reviewers for their useful hints and constructive comments throughout the reviewing process.

References

[1] K. Levenberg, "A method for the solution of certain problems in least squares," *Q. Appl. Math.*, vol. 2, pp. 164–168, 1944.
[2] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *J. Soc. Ind. Appl. Math.*, vol. 11, pp. 431–441, 1963.
[3] M. T. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.
[4] G. Lera and M. Pinzolas, "A quasilocal Levenberg-Marquardt algorithm for neural network training," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'98)*, vol. 3, Anchorage, AK, May, pp. 2242–2246.
[5] ——, "Neighborhood-based Levenberg-Marquardt algorithm for neural network training," *IEEE Trans. Neural Netw.*, vol. 13, no. 5, pp. 1200–1203, Sep. 2002.
[6] M. Pinzolas, J. J. Astrain, J. R. González, and J. Villadangos, "Isolated hand-written digit recognition using a neurofuzzy scheme and multiple classification," *J. Intell. Fuzzy Syst.*, vol. 12, no. 2, pp. 97–105, Dec. 2002.
[7] J. M. Cano, M. Pinzolas, J. J. Ibarrola, and J. López-Coronado, "Identificación de funciones utilizando sistemas lógicos difusos," in *Proc. XXI Jornadas Automática*, Sevilla, Spain, 2000.
[8] M. Pinzolas, J. J. Ibarrola, and J. López-Coronado, "A neurofuzzy scheme for on-line identification of nonlinear dynamical systems with variable transfer function," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'00)*, Como, Italy, pp. 215–221.

Sparse Component Analysis and Blind Source Separation of Underdetermined Mixtures

Pando Georgiev, Fabian Theis, and Andrzej Cichocki

***Abstract*—In this letter, we solve the problem of identifying matrices $\mathrm{S} \in \mathbb{R}^{n\times N}$ and $\mathrm{A} \in \mathbb{R}^{m\times n}$ knowing only their multiplication $\mathrm{X} = \mathrm{AS}$, under some conditions, expressed either in terms of A and sparsity of S (*identifiability* conditions), or in terms of X (sparse component analysis (SCA) conditions). We present algorithms for such identification and illustrate them by examples.**

***Index Terms*—Blind source separation (BSS), sparse component analysis (SCA), underdetermined mixtures.**

I. Introduction

One of the fundamental questions in data analysis, signal processing, data mining, neuroscience, etc. is how to represent a large data set $\mathbf{X}$ (given in form of a $(m \times N)$-matrix) in different ways. A simple approach is a linear matrix factorization

$$\mathbf{X} = \mathbf{AS} \quad \mathbf{A} \in \mathbb{R}^{m\times n}, \quad \mathbf{S} \in \mathbb{R}^{n\times N} \tag{1}$$

where the unknown matrices $\mathbf{A}$ (dictionary) and $\mathbf{S}$ (source signals) have some specific properties, for instance:

1) the rows of $\mathbf{S}$ are (discrete) random variables, which are statistically independent as much as possible—this is independent component analysis (ICA) problem; 2) $\mathbf{S}$ contains as many zeros as possible—this is the sparse representation or sparse component analysis (SCA) problem; 3) the elements of $\mathbf{X}$, $\mathbf{A}$, and $\mathbf{S}$ are nonnegative—this is nonnegative matrix factorization (NMF) [8].

There is a large amount of papers devoted to ICA problems [2], [5] but mostly for the case $m \geq n$. We refer to [1], [6], [7], and [9]–[11] for some recent papers on SCA and underdetermined ICA $(m < n)$.

A related problem is the so called blind source separation (BSS) problem, in which we know *a priori* that a representation such as in (1) exists and the task is to recover the sources (and the mixing matrix) as accurately as possible. A fundamental property of the complete BSS problem is that such a recovery (under assumptions in 1) and non-Gaussianity of the sources) is possible up to permutation and scaling of the sources, which makes the BSS problem so attractive.

In this letter, we consider SCA and BSS problems in the underdetermined case ($m < n$, i.e., more sources than sensors, which is more challenging problem), where the additional information compensating the limited number of sensors is the *sparseness* of the sources. It should be noted that this problem is quite general and fundamental, since the sources could be not necessarily sparse in time domain. It would be sufficient to find a linear transformation (e.g., wavelet packets), in which the sources are sufficiently sparse.

In the sequel, we present new algorithms for solving the BSS problem: matrix identification algorithm and source recovery algorithm under conditions that the source matrix $\mathbf{S}$ has at most $m - 1$ nonzero elements in each column and if the identifiability conditions

Manuscript received November 20, 2003; revised July 25, 2004.

P. Georgiev is with ECECS Department, University of Cincinnati, Cincinnati, OH 45221 USA (e-mail: pgeorgie@ececs.uc.edu).
F. Theis is with the Institute of Biophysics, University of Regensburg, D-93040 Regensburg, Germany (e-mail: fabian@theis.name).
A. Cichocki is with the Laboratory for Advanced Brain Signal Processing, Brain Science Institute, The Institute of Physical and Chemical Research (RIKEN), Saitama 351-0198, Japan (e-mail: cia@bsp.brain.riken.jp).

Digital Object Identifier 10.1109/TNN.2005.849840