Parallel Approach to the Levenberg-Marquardt Learning Algorithm for Feedforward Neural Networks

Jarosław Bilski[1(✉)], Jacek Smoląg[1], and Jacek M. Żurada[2,3]

[1] Institute of Computational Intelligence, Częstochowa University of Technology, Częstochowa, Poland
{Jaroslaw.Bilski,Jacek.Smolag}@iisi.pcz.pl
[2] Information Technology Institute, University of Sociel Sciences, Łódź, Poland
[3] Department Electrical and Computer Engineering, University of Louisville, Louisville, KY 40292, US
jacek.zurada@louisville.edu

Abstract. A parallel architecture of the Levenberg-Marquardt algorithm for training a feedforward neural network is presented. The proposed solution is based on completely new parallel structures to effectively reduce high computational load of this algorithm. Detailed parallel neural network structures are explicitely discussed.

1 Introduction

Feedforward neural networks have been investigated by many scientists e.g. [1], [10], [19], [23], [26], [37], [39]. Gradient methods have been often used to train feedforward networks, see e.g. [14], [24], [38]. In the traditional approach neural networks learning algorithms, like other learning algorihms [25], [28], [29], [31], [34], are implemented on a serial computer. Due to the computational complexity of the learning algorithm, the serial implementation is very time consuming and slow. The Levenberg Marquart algorithm [15], [22] is one of the most effective learning methods, but requires particularly complex calculations. Unfortunately, for very large networks the computational load of the Levenberg-Marquardt algorithm makes it impractical. A suitable solution to this problem is the use of high performance dedicated parallel structures, see eg. [2] - [9], [32], [33]. This paper presents a new concept of parallel realisation of the Levenberg-Marquardt learning algorithm. A single iteration of the parallel architecture requires much fewer computation cycles than a serial implementation. The efficiency of this new architecture is very satisfying and is explained in the last part of the paper.

A sample structure of the feedforward network is shown in Fig. 1. The network has L layers, N_l neurons in each $l-th$ layer and N_L outputs. The input vector contains N_0 input signals. The equation (1) describes the recall phase of the network

$$\begin{aligned} s_i^{(l)}(t) &= \sum_{j=0}^{N_{l-1}} w_{ij}^{(l)}(t)\, x_i^{(l)}(t), \\ y_i^{(l)}(t) &= f(s_i^{(l)}(t)). \end{aligned} \tag{1}$$

L. Rutkowski et al. (Eds.): ICAISC 2015, Part I, LNAI 9119, pp. 3–14, 2015.
DOI: 10.1007/978-3-319-19324-3_1

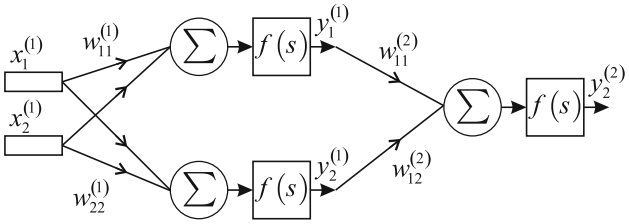

Fig. 1. Feedforward neural network sample structure

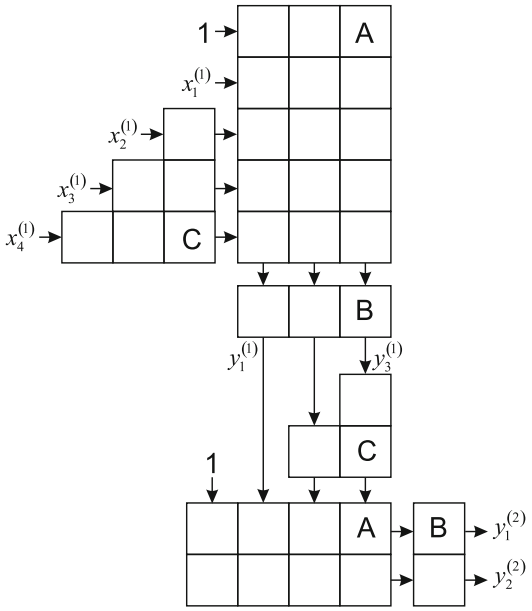

Fig. 2. Pipelined version of recall phase of the feedforward network

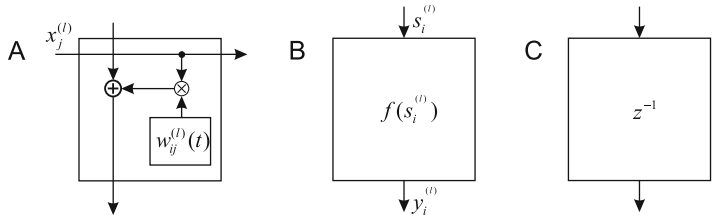

Fig. 3. The structures of recall phase processing elements

The parallel realisation of the recall phase algorithm uses the architecture which requires many simple processing elements. The pipelined version of the parallel realisation of the feedforward network in recall phase (1) is depicted in Fig. 2 and its processing elements (PE) in Fig. 3. Two main kinds of functional

processing elements are used in the proposed solution. The aim of the processing elements A is to create matrices which contain values of weights in all layers. The input signals are entered parallelly into the rows of the elements, multiplied by weights and finally the received results are summed in the columns. The activation function for each neuron in the $l-th$ layer is calculated after determination of product $\mathbf{w}_i^{(l)}\mathbf{x}^{(l)}$ in the processing element of type B. Additional processing element C is used to delay transferred data. Thus the structure operates in a pipeline flow and next results can be obtained after only one step. The outputs of neurons in the previous layer act the same time as inputs to the next layer. The output $\mathbf{y}^{(L)}$ for the last layer is the output of the whole network.

The Levenberg-Marquard method [15], [22] is used to train the feedforward network. The following goal criterion is minimized

$$E\left(\mathbf{w}\left(n\right)\right)=\frac{1}{2}\sum\nolimits_{t=1}^{Q}\sum\nolimits_{r=1}^{N_L}\varepsilon_r^{(L)^2}\left(t\right)=\frac{1}{2}\sum\nolimits_{t=1}^{Q}\sum\nolimits_{r=1}^{N_L}\left(y_r^{(L)}\left(t\right)-d_r^{(L)}\left(t\right)\right)^2 \tag{2}$$

where $\varepsilon_i^{(L)}$ is defined as

$$\varepsilon_r^{(L)}(t)=\varepsilon_r^{(Lr)}(t)=y_r^{(L)}(t)-d_r^{(L)}(t) \tag{3}$$

and $d_r^{(L)}(t)$ is the $r-th$ desired output in the $t-th$ probe.

The Levenberg-Marquardt algorithm is a modification of the Newton method and is based on the first three elements of the Taylor series expansion of the goal function. In the classical case a change of weights is given by

$$\Delta\left(\mathbf{w}(n)\right)=-\left[\nabla^{\mathbf{2}}\mathbf{E}\left(\mathbf{w}(n)\right)\right]^{-1}\nabla\mathbf{E}\left(\mathbf{w}(n)\right) \tag{4}$$

this requires knowledge of the gradient vector

$$\nabla\mathbf{E}\left(\mathbf{w}(n)\right)=\mathbf{J}^T\left(\mathbf{w}(n)\right)\varepsilon\left(\mathbf{w}(n)\right) \tag{5}$$

and the Hessian matrix

$$\nabla^{\mathbf{2}}\mathbf{E}\left(\mathbf{w}(n)\right)=\mathbf{J}^T\left(\mathbf{w}(n)\right)\mathbf{J}\left(\mathbf{w}(n)\right)+\mathbf{S}\left(\mathbf{w}(n)\right) \tag{6}$$

where $\mathbf{J}\left(\mathbf{w}(\mathbf{n})\right)$ in (5) and (6) is the Jacobian matrix

$$\mathbf{J}(\mathbf{w}\left(n\right))=\begin{bmatrix}\frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{10}^{(1)}} & \frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{N_LN_{L-1}}^{(L)}}\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{10}^{(1)}} & \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{N_LN_{L-1}}^{(L)}}\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{10}^{(1)}} & \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{10}^{(1)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{N_LN_{L-1}}^{(L)}}\end{bmatrix}. \tag{7}$$

The errors $\varepsilon_i^{(lr)}$ in the hidden layers are calculated as follows

$$\varepsilon_i^{(lr)}(t) \stackrel{\wedge}{=} \sum_{m=1}^{N_{l+1}} \delta_i^{(l+1,r)}(t)\, w_{mi}^{(l+1)}, \tag{8}$$

$$\delta_i^{(lr)}(t) = \varepsilon_i^{(lr)}(t)\, f'\left(s_i^{(lr)}(t)\right). \tag{9}$$

On this basis, the components of the Jacobian matrix for each weight can be determined

$$\frac{\partial \varepsilon_r^{(L)}(t)}{w_{ij}^{(l)}} = \delta_i^{(lr)}(t)\, x_j^{(l)}(t). \tag{10}$$

It should be noted that derivatives (10) are computed in a similar way it is done in the classical backpropagation method, except that each time there is only one error given to the output. In this algorithm, the weights of the entire network are treated as a single vector and their derivatives form the Jacobian matrix $\mathbf{J}$.

The $\mathbf{S}(\mathbf{w}(n))$ component (6) is given by the formula

$$\mathbf{S}(\mathbf{w}(n)) = \sum\nolimits_{t=1}^{Q} \sum\nolimits_{r=1}^{N_L} \varepsilon_r^{(L)}(t) \nabla^2 \varepsilon_r^{(L)}(t). \tag{11}$$

In the Gauss-Newton method it is assumed that $\mathbf{S}(\mathbf{w}(n)) \approx 0$ and that equation (4) takes the form

$$\Delta(\mathbf{w}(n)) = -\left[\mathbf{J}^T(\mathbf{w}(n))\,\mathbf{J}(\mathbf{w}(n))\right]^{-1} \mathbf{J}^T(\mathbf{w}(n))\,\varepsilon(\mathbf{w}(n)). \tag{12}$$

In the Levenberg-Marquardt method is is assumed that $\mathbf{S}(\mathbf{w}(n)) = \mu\mathbf{I}$ and that equation (4) takes the form

$$\Delta(\mathbf{w}(n)) = -\left[\mathbf{J}^T(\mathbf{w}(n))\,\mathbf{J}(\mathbf{w}(n)) + \mu\mathbf{I}\right]^{-1} \mathbf{J}^T(\mathbf{w}(n))\,\varepsilon(\mathbf{w}(n)). \tag{13}$$

By defining

$$\begin{aligned} \mathbf{A}(n) &= -\left[\mathbf{J}^T(\mathbf{w}(n))\,\mathbf{J}(\mathbf{w}(n)) + \mu\mathbf{I}\right] \\ \mathbf{h}(n) &= \mathbf{J}^T(\mathbf{w}(n))\,\varepsilon(\mathbf{w}(n)) \end{aligned} \tag{14}$$

the equation (13) is as follows

$$\Delta(\mathbf{w}(n)) = \mathbf{A}(n)^{-1}\mathbf{h}(n). \tag{15}$$

The equation (15) can be solved using the QR factorization

$$\mathbf{Q}^T(n)\,\mathbf{A}(n)\,\Delta(\mathbf{w}(n)) = \mathbf{Q}^T(n)\,\mathbf{h}(n), \tag{16}$$

$$\mathbf{R}(n)\,\Delta(\mathbf{w}(n)) = \mathbf{Q}^T(n)\,\mathbf{h}(n). \tag{17}$$

This paper used the Householder reflection method for the QR factorization. Operation of the Levenberg-Marquardt algorithm is described below. In practice, the algorithm is implemented in 5 steps:

1. The calculation of the network outputs for all input data, errors and the goal criterion.
2. The calculation the Jacobian matrix, by applying the backpropagation method for each error individually.
3. The calculation of $\Delta(\mathbf{w}(n))$ by using the QR factorization.
4. The recalculation of the goal criterion (2) for $\mathbf{w}(n) + \Delta(\mathbf{w}(n))$. If the goal criterion is less than the one calculated in step 1, then μ should be reduced β times, the new weight vector remains and the algorithm returns to Step 1. Otherwise, the μ value should be increased β times and the algorithm goes back to step 3.
5. The algorithm terminates when the gradient falls below a preset value or the goal function falls below a preset value.

2 Parallel Realisation

First, the errors in all neurons using backpropagation are calculated assuming that each time only one error is given to the output and than the Jacobian matrix is determined. This is accomplished by the structure shown in Fig. 4. Its processing elements are shown in Fig. 5. The A processing elements are used to calculate the error $\varepsilon_r^{(L)}$ (3) in the output layer. The B elements transfer the errors to the linear part of neurons (9), and the D processing elements compute errors

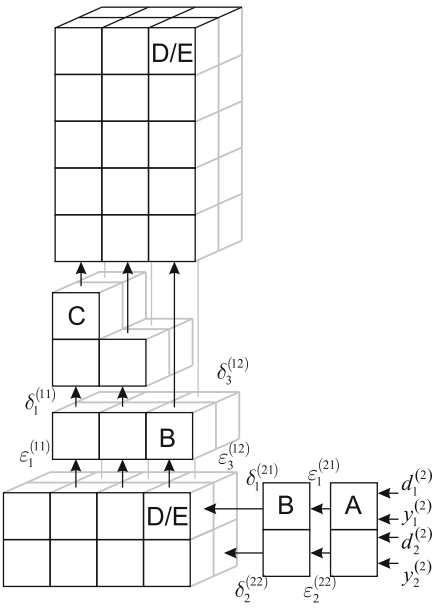

Fig. 4. The structure showing how to propagate error back and compute the Jacobian matrix elements

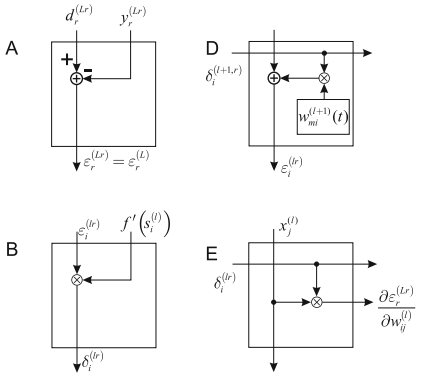

Fig. 5. The processing elements for propagating error back and computing the Jacobian matrix elements

$\varepsilon_i^{(lr)}$ in the hidden layers (8). The E processing elements determine the elements of the Jacobian matrix (10). It should be noted that at the same time all rows of the Jacobian matrix for all output errors of a single sample are determined. This is achieved by the use of L parallel layers. The structure shown in Fig. 4 starts operation immediately after the calculation of the outputs perfomed by the structure in Fig 2 for the data of the first sample. The $\mathbf{A}(n)$ matrix (14) is calculated based on the Jacobian matrix. These calculations are performed by the structure shown in Fig. 6. In the same figure the internal structure of the individual processing elements in this structure is also shown. At the same time, the vector $\mathbf{h}(n)$ (14) is determined by the structure shown in Figure 7.

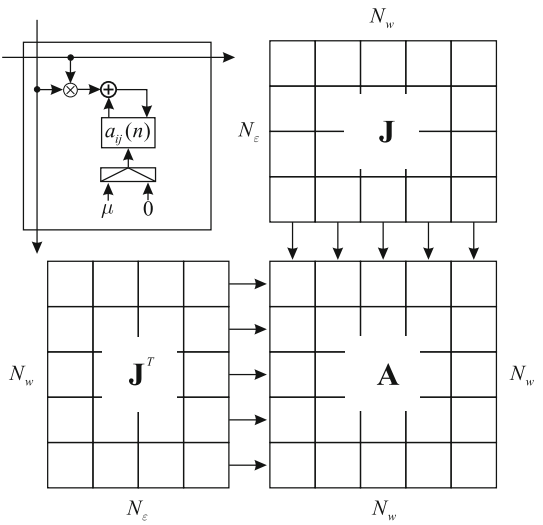

Fig. 6. The structure for computing the **A** matrix and its processing element

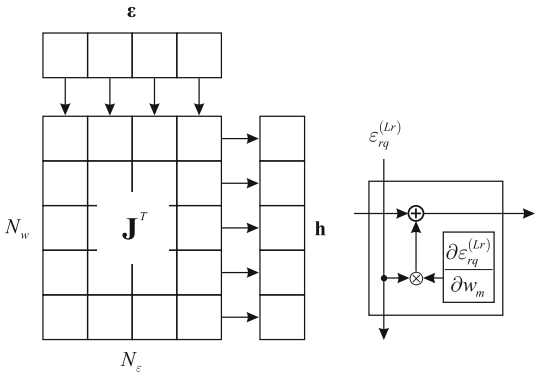

Fig. 7. The structure for computing the **h** vector and its processing element

After calculating the $\mathbf{A}(n)$ matrix and the $\mathbf{h}(n)$ vector, the equation (15) is solved. The equation (15) can be solved using the QR factorization . This will be achieved by the use of the Hauseholder reflections. The parallel structure calculating matrices $\mathbf{R}$ and $\mathbf{Q}^T\mathbf{h}$ is shown in Fig. 8. Elements A2 transform

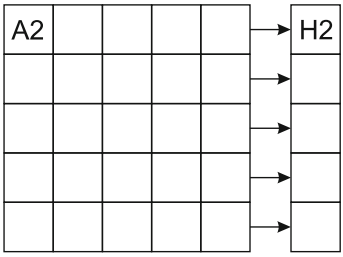

Fig. 8. The general structure for parallelization of the QR decomposition

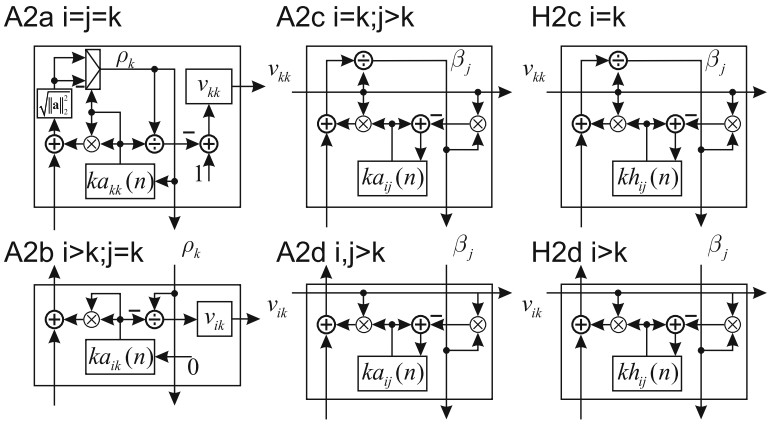

Fig. 9. The processing elements of the QR decomposition

the elements of the **A** matrix and elements H2 transform the elements of the **h** vector. This step performs a sequence of the Householder reflection so as to reset the elements below the main diagonal of the **A** matrix. First, the elements in the first column are reset, then the second and so on, until the last but one. The **A** matrix and the **h** vector are transformed. The vectors used to perform reflections are based on the columns of the **A** matrix, except that it includes the elements from the main diagonal to the end of the column. It should be noted that the QR decomposition process requires the $N_w - 1$ matrix reflections. The A2 and H2 processing elements will operate differently depending on the phase (k) of the process (Fig. 9). The A2a and A2b elements determine the module of the **a** subvector and, on this basis, calculate the value

$$\rho_k = \begin{cases} \|\mathbf{a}_k\|_2 & for\ a_{kk} \leq 0 \\ -\|\mathbf{a}_k\|_2 & for\ a_{kk} > 0, \end{cases} \tag{18}$$

and the reflection vector

$$\mathbf{v}_k = \begin{bmatrix} \mathbf{0} \\ \bar{\mathbf{v}}_k \end{bmatrix}. \tag{19}$$

The **v** vector is transmitted to the elements A2c and A2d which in the next columns calculate the values of the reflected vectors $\bar{\mathbf{a}}$

$$\bar{\mathbf{a}} = \mathbf{a} - \mathbf{v}\beta \tag{20}$$

where

$$\beta = \frac{\mathbf{v}^T \mathbf{a}}{\gamma} \tag{21}$$

$$\gamma = v_1. \tag{22}$$

The H2c and H2d elements operate in the same manner on the **h** vector. The construction of all the processing elements is shown in Fig. 9. After determination of the **R** matrix and the $\mathbf{Q^T h}$ vector the equation (17) is solved. This is realized by the structure shown in Fig. 10. Its elements are also shown in this figure. The A3a and A3b elements determine the value of $\Delta(\mathbf{w}(n))$, and the W elements update the weights.

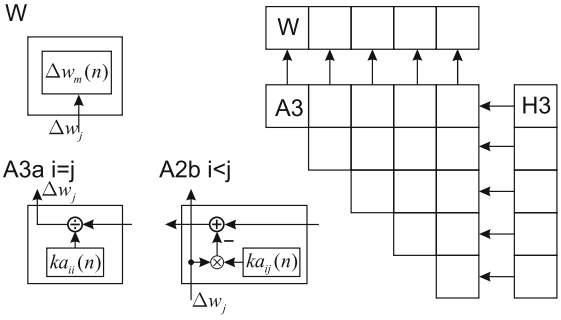

Fig. 10. The structure for computing the weight vector **w** and its processing element

3 Conclusion

In this paper the parallel realisation of the Leveberg-Marquardt learning algorithm for a feedforward neural network is proposed. It is assumed that all multiplications and additions are performed within the same time unit. To make the presentation of the results simple, graphs only for the neural network shown in Fig. 1 with variable neuron number in the hidden layer are presented. We can compare computational performance of the parallel implementation of the Levenberg-Marquardt learning algorithm with a sequential solution for a two-layer network with two inputs, one output, up to $N = 100$ neurons in the hidden layer and up to $Q = 100$ samples of the learning data of a neural network. Computational complexity of the serial Levenberg-Marquardt learning algorithm is of order $\mathcal{O}(N^3)$ and equals $TS = 21\frac{1}{3}N^3 + 72N^2 + 48\frac{2}{3}N + 32N^2Q + 38NQ + 10Q$. In the presented parallel architecture each epoch requires only $TP = 16N^2 + 53N + 2Q + 15$ time units (see Fig. 11). Performance factor ($PF = TS/TP$) of parallel realisation of the Levenberg-Marquardt learning algorithm achieves nearly 330 for $N = 100$ neurons in the hidden layer, $Q = 100$ samples of the learning data and it grows fast when these numbers grow, see Fig. 11. It has been observed that the performance of the proposed solution is promising. An analogous parallel approach can be used for other advanced learning algorithms of feedforward neural networks, see eg. [1], [7]. In the future research we plan to design parallel realisation of learning of other structures including probabilistic neural networks [27] and various fuzzy [11], [17], [20], [35], [36], [40], and neuro-fuzzy structures [12], [13], [16], [18], [21], [30].

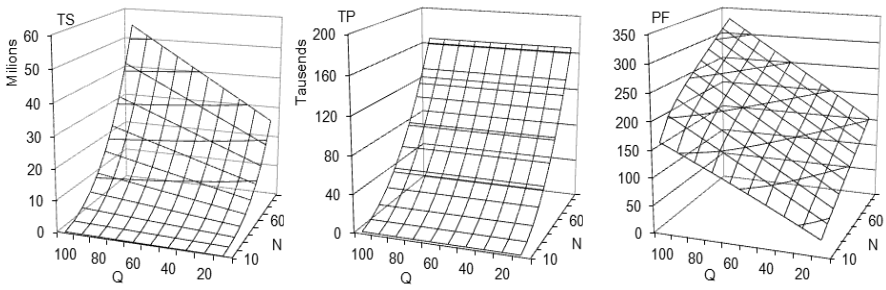

Fig. 11. Number of times cycles in a) classical (serial), b) parallel implementation and c) performance factor

References

1. Bilski, J.: The UD RLS algorithm for training the feedforward neural networks. International Journal of Applied Mathematics and Computer Science 15(1), 101–109 (2005)

2. Bilski, J., Litwiński, S., Smoląg, J.: Parallel realisation of QR algorithm for neural networks learning. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 158–165. Springer, Heidelberg (2004)
3. Bilski, J., Smoląg, J.: Parallel realisation of the recurrent RTRN neural network learning. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 11–16. Springer, Heidelberg (2008)
4. Bilski, J., Smoląg, J.: Parallel realisation of the recurrent Elman neural network learning. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part II. LNCS (LNAI), vol. 6114, pp. 19–25. Springer, Heidelberg (2010)
5. Bilski, J., Smoląg, J.: Parallel realisation of the recurrent multi layer perceptron learning. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2012, Part I. LNCS (LNAI), vol. 7267, pp. 12–20. Springer, Heidelberg (2012)
6. Bilski, J., Smoląg, J.: Parallel approach to learning of the recurrent Jordan neural network. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part I. LNCS (LNAI), vol. 7894, pp. 32–40. Springer, Heidelberg (2013)
7. Bilski, J.: Parallel Structures for Feedforward and Dynamical Neural Networks (in Polish). AOW EXIT (2013)
8. Bilski, J., Smoląg, J., Galushkin, A.I.: The parallel approach to the conjugate gradient learning algorithm for the feedforward neural networks. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2014, Part I. LNCS (LNAI), vol. 8467, pp. 12–21. Springer, Heidelberg (2014)
9. Bilski, J., Smoląg, J.: Parallel Architectures for Learning the RTRN and Elman Dynamic Neural Networks. IEEE Transactions on Parallel and Distributed Systems PP(99) (2014), doi:10.1109/TPDS.2014.2357019
10. Chu, J.L., Krzyźak, A.: The recognition of partially occluded objects with support vector machines, convolutional neural networks and deep belief networks. Journal of Artificial Intelligence and Soft Computing Research 4(1), 5–19 (2014)
11. Cpałka, K., Rutkowski, L.: Flexible Takagi-Sugeno Fuzzy Systems. In: Proceedings of the Int. Joint Conference on Neural Networks, Montreal, pp. 1764–1769 (2005)
12. Cpałka, K., Łapa, K., Przybył, A., Zalasiński, M.: A new method for designing neuro-fuzzy systems for nonlinear modelling with interpretability aspects. Neurocomputing 135, 203–217 (2014)
13. Cpalka, K., Rebrova, O., Nowicki, R., et al.: On design of flexible neuro-fuzzy systems for nonlinear modelling. International Journal of General Systems 42(6), Special Issue: SI, 706–720 (2013)
14. Fahlman, S.: Faster learning variations on backpropagation: An empirical study. In: Proceedings of Connectionist Models Summer School, Los Atos (1988)
15. Hagan, M.T., Menhaj, M.B.: Training feedforward networks with the Marquardt algorithm. IEEE Transactions on Neural Networks 5(6), 989–993 (1994)
16. Korytkowski, M., Nowicki, R., Rutkowski, L., Scherer, R.: AdaBoost Ensemble of DCOG Rough–Neuro–Fuzzy Systems. In: Jędrzejowicz, P., Nguyen, N.T., Hoang, K. (eds.) ICCCI 2011, Part I. LNCS, vol. 6922, pp. 62–71. Springer, Heidelberg (2011)

17. Korytkowski, M., Rutkowski, L., Scherer, R.: From ensemble of fuzzy classifiers to single fuzzy rule base classifier. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 265–272. Springer, Heidelberg (2008)
18. Korytkowski, M., Scherer, R.: Negative Correlation Learning of Neuro-fuzzy System Ensembles. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part I. LNCS (LNAI), vol. 6113, pp. 114–119. Springer, Heidelberg (2010)
19. Laskowski, L., Jelonkiewicz, J.: Self-Correcting Neural Network for stereo-matching problem solving. Fundamenta Informaticae 138, 1–26 (2015)
20. Łapa, K., Przybył, A., Cpałka, K.: A new approach to designing interpretable models of dynamic systems. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part II. LNCS (LNAI), vol. 7895, pp. 523–534. Springer, Heidelberg (2013)
21. Łapa, K., Zalasiński, M., Cpałka, K.: A new method for designing and complexity reduction of neuro-fuzzy systems for nonlinear modelling. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part I. LNCS (LNAI), vol. 7894, pp. 329–344. Springer, Heidelberg (2013)
22. Marqardt, D.: An algorithm for last-sqares estimation of nonlinear paeameters. J. Soc. Ind. Appl. Math., 431–441 (1963)
23. Patan, K., Patan, M.: Optimal training strategies for locally recurrent neural networks. Journal of Artificial Intelligence and Soft Computing Research 1(2), 103–114 (2011)
24. Riedmiller, M., Braun, H.: A direct method for faster backpropagation learning: The RPROP Algorithm. In: IEEE International Conference on Neural Networks, San Francisco (1993)
25. Romaszewski, M., Gawron, P., Opozda, S.: Dimensionality reduction of dynamic msh animations using HO-SVD. Journal of Artificial Intelligence and Soft Computing Research 3(3), 277–289 (2013)
26. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McCelland, J. (red.) Parallel Distributed Processing, ch. 8, vol. 1. The MIT Press, Cambridge (1986)
27. Rutkowski, L.: Multiple Fourier series procedures for extraction of nonlinear regressions from noisy data. IEEE Transactions on Signal Processing 41(10), 3062–3065 (1993)
28. Rutkowski, L.: Identification of MISO nonlinear regressions in the presence of a wide class of disturbances. IEEE Transactions on Information Theory 37(1), 214–216 (1991)
29. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: Decision trees for mining data streams based on the gaussian approximation. IEEE Transactions on Knowledge and Data Engineering 26(1), 108–119 (2014)
30. Rutkowski, L., Przybył, A., Cpałka, K., Er, M.J.: Online speed profile generation for industrial machine tool based on neuro-fuzzy approach. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part II. LNCS (LNAI), vol. 6114, pp. 645–650. Springer, Heidelberg (2010)
31. Rutkowski, L., Rafajlowicz, E.: On optimal global rate of convergence of some nonparametric identification procedures. IEEE Transactions on Automatic Control 34(10), 1089–1091 (1989)
32. Smoląg, J., Bilski, J.: A systolic array for fast learning of neural networks. In: Proc. of V Conf. Neural Networks and Soft Computing, Zakopane, pp. 754–758 (2000)

33. Smoląg, J., Rutkowski, L., Bilski, J.: Systolic array for neural networks. In: Proc. of IV Conf. Neural Networks and Their Applications, Zakopane, pp. 487–497 (1999)
34. Starczewski, A.: A clustering method based on the modified RS validity index. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part II. LNCS (LNAI), vol. 7895, pp. 242–250. Springer, Heidelberg (2013)
35. Starczewski, J., Rutkowski, L.: Connectionist structures of type 2 Fuzzy Inference Systems. In: 4th International Conference on Parallel Processing and Applied Mathematics, Nalenczow, Poland (2001)
36. Starczewski, J., Rutkowski, L.: Interval type 2 neuro-fuzzy systems based on interval consequents. In: Neural Networks and Soft Computing. Advances In Soft Computing, pp. 570–577 (2003)
37. Tadeusiewicz, R.: Neural Networks (in Polish). AOW RM (1993)
38. Werbos, J.: Backpropagation through time: What it does and how to do it. Proceedings of the IEEE 78(10) (1990)
39. Wilamowski, B.M., Yo, H.: Neural network learning without backpropagation. IEEE Transactions on Neural Networks 21(11), 1793–1803 (2010)
40. Zalasiński, M., Cpałka, K.: New approach for the on-line signature verification based on method of horizontal partitioning. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2013, Part II. LNCS (LNAI), vol. 7895, pp. 342–350. Springer, Heidelberg (2013)