A New Computational Approach to the Levenberg-Marquardt Learning Algorithm

Jarosław Bilski(✉), Barosz Kowalczyk, and Jacek Smoląg

Institute of Computational Intelligence, Częstochowa University of Technology, Częstochowa, Poland
{Jaroslaw.Bilski,Barosz.Kowalczyk,Jacek.Smolag}@pcz.pl

Abstract. A new parallel computational approach to the Levenberg-Marquardt learning algorithm is presented. The proposed solution is based on the AVX instructions to effectively reduce the high computational load of this algorithm. Detailed parallel neural network computations are explicitly discussed. Additionally obtained acceleration is shown based on a few test problems.

Keywords: Neural network learning algorithm · Levenberg-marquardt learning algorithm · Vector computations · Approximation · Classification

1 Introduction

Artificial feedforward neural networks have been studied by many scientists e.g. [2,12,14,27,28,31,43,45]. One of the most frequently used methods for training feedforward neural networks are gradient methods, see e.g. [18,29,44]. Most of the simulations of neural networks learning algorithms, like other learning algorithms [19,20,30,33,34,36,40,41], work on a serial computer. The computational complexity of many learning algorithms is very high. This makes serial implementation very time consuming and slow. The Levenberg Marquart (LM) algorithm [21,26] is one of the most effective learning algorithms, unfortunately, it requires a lot of calculations. But, for very large networks the computational load of the LM algorithm makes it impractical. A suitable solution to this problem is the use of high performance dedicated parallel structures, see eg. [3,5–13,38,39,48]. This paper shows a new parallel computational approach to the LM algorithm based on vector instruction. The results of the study of a new parallel approach to the LM algorithm is shown in the last part of the paper.

A sample structure of the feedforward neural network is shown in Fig. 1. This sample network has L layers, N_l neurons in each $l-th$ layer, and N_L outputs.

This work has been supported by the Polish National Science Center under Grant 2017/27/B/ST6/02852.

L. Rutkowski et al. (Eds.): ICAISC 2022, LNAI 13588, pp. 16–26, 2023.
https://doi.org/10.1007/978-3-031-23492-7_2

The input vector contains N_0 input values. The Eq. (1) describes the recall phase of the network

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

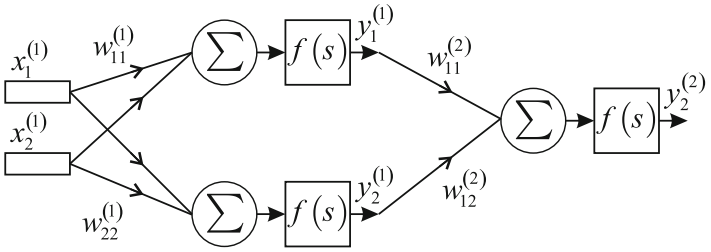

Fig. 1. Sample feedforward neural network.

The Levenberg-Marquard method [21,26] is used to train the feedforward neural network. The following loss function is minimized

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

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 LM algorithm is a modification of the Newton method and is based on the first three elements of the Taylor series expansion of the loss function. A change of weights is given by

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

this requires knowledge of the gradient vector

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

and the Hessian matrix

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

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

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

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

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

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

Based on this, the elements of the Jacobian matrix for each weight can be computed

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

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 Eq. (13) is as follows

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

The Eq. (15) can be solved using the QR factorization

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

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

This paper used the Givens rotations for the QR factorization. The operation, in 5 steps, of the LM algorithm is described below:

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

2 Vector Solution for Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm needs high computing power. Each epoch starts with steps 1 and 2, and next steps 3 and 4 can be repeated a few times. Figure 2 shows a single epoch of the LM algorithm, showing the first two steps and repeating steps 3 and 4. It is worth noting that the next pairs of steps 3 and 4 are independent of each other and can be performed at the same time. They only differ in the μ parameter value and have the same starting point. Thus, they can be run parallel on separate processor cores. However, the solution proposed in this article uses processor vector instructions. Vector instructions allow 4, 8, and even 16 operations to be performed in parallel. This approach enables simultaneous determination of new 4, 8, or 16 points in the weight space using only one processor core, see Fig. 3. Figure 3a shows the epoch of the LM algorithm with the use of four-element vectors. After completing the first two

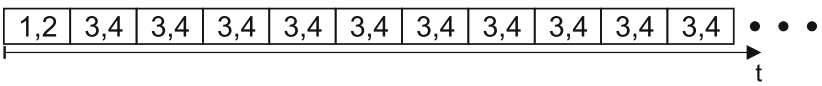

Fig. 2. Sample illustration for computational steps in LM algorithm.

steps, the algorithm calculates steps 3 and 4 for the next 4 parameters μ at one time. Thus, the three consecutive computations of steps 3 and 4 are performed earlier and therefore do not take computational time. The rectangles with the line in the middle symbolize steps 3 and 4, which are used in the standard calculation method and are omitted in calculations using vector instructions. Figure 3b shows the version with eight-element vectors.

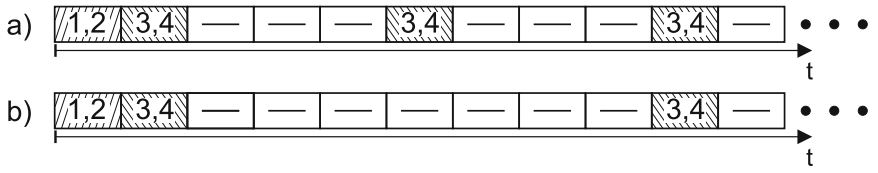

Fig. 3. Sample illustration for calculating method with vector instructions. a) the 4-elements vector, b) the 8-elements vector

Figure 4 shows an example of the learning process using the LM algorithm. In the following epochs, you can see a different number of steps 3 and 4 repetitions. There are epochs where the repetition does not occur and there are those with a

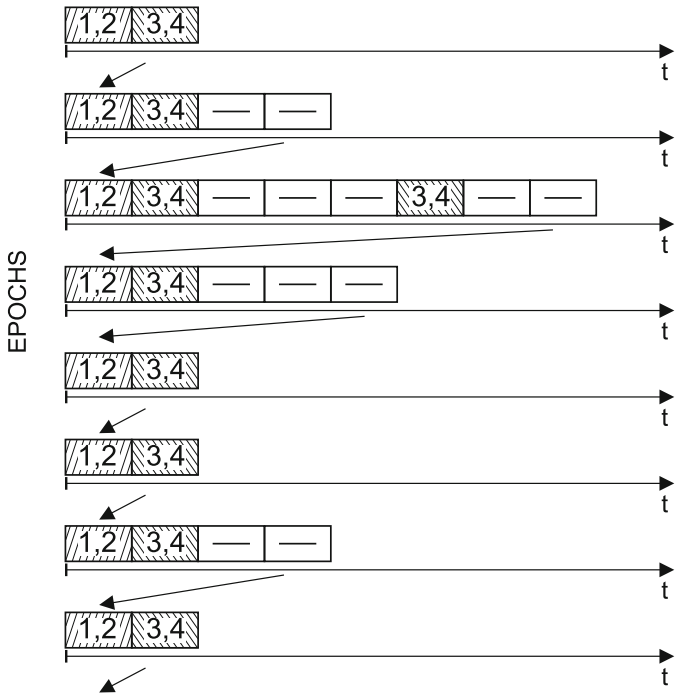

Fig. 4. Sample illustration for training process with vector instructions.

large number of repetitions, in this case, vector instructions can be used, which makes it possible to calculate up to four pairs of steps 3 and 4 at the same time and consequently shortening the learning time. Of course, eight- or sixteen-element vectors can be used instead of using four-element vectors. This increases the parallelism and speed of the proposed calculation method.

3 Experimental Results

The proposed solution was tested against the classical variant of the Levenberg-Marquardt learning algorithm on several test problems. Two types of forward-coupled artificial neural networks were tested in the experiment: MLP — Multilayer Perceptron, FCMLP — Fully Connected Multilayer Perceptron. The performance of the presented calculation method was measured in average training time in milliseconds. The presented results are compiled according to the best combination of training parameters. In all cases, the initial weights were randomly selected from the range [–0.5,0.5]. The number of epochs has been limited to 1,000. Each training session was repeated 100 times.

3.1 Logistic Function Approximation

The logistic function is a unary function given by the formula

$$y = f(x) = 4x(1 - x) \tag{18}$$

The teaching sequence contains 11 samples where $x \in [0, 1]$. The average accepted error threshold has been set to 0.001. Table 1 shows the simulation results for two kinds of neural networks MLP and FCMLP. Both networks have five neurons in the hidden layer. The symbols LM, LMP 4, LMP 8, and LMP 16 represent the average network training time using the LM algorithm and its vector versions for 4, 8, and 16 element vectors, respectively. The speed factor means how many percent the vector version is faster then the classical one and is given by the formula

$$SF = \left(1 - \frac{LMPx}{LM}\right) * 100\% \tag{19}$$

Table 1. Training results for the LOG function.

Network	LM [ms]	LMP4 [ms]	SF [%]	LMP8 [ms]	SF [%]	LMP16 [ms]	SF [%]
MLP-1-5-1	0.880	0.440	50	0.434	50	0.433	50
FCMLP-1-5-1	0.588	0.311	47	0.306	48	0.305	48

3.2 Hang Function Approximation

The Hang function is a nonlinear two-argument x_1 and x_2 function with the following formula

$$y = f(x_1, x_2) = \left(1 + x_1^{-2} + \sqrt{x_2^{-3}}\right)^2 \tag{20}$$

The Hang teaching sequence contains 50 samples that cover arguments in the range of $x_1, x_2 \in [1, 5]$. The target error threshold was set to 0.001 as the epoch average. The results of simulations for the Hang function are shown in Table 2. Both tested networks have 15 neurons in the hidden layer.

Table 2. Training results for the HANG function.

Network	LM [ms]	LMP4 [ms]	SF [%]	LMP8 [ms]	SF [%]	LMP16 [ms]	SF [%]
MLP-2-15-1	27.235	13.191	51	12.553	53	12.462	54
FCCMLP-2-15-1	34.237	16.691	51	16.165	52	16.111	52

3.3 IRIS Function Classification

The iris dataset contains 150 instances describing three species of iris flowers. The flowers are identified with 4 numerical attributes describing the lengths and widths of the petals of the flower. The target error has been set to 0.05. Table 3 shows the simulation results.

Table 3. Training results for the IRIS function.

Network	LM [ms]	LMP4 [ms]	SF [%]	LMP8 [ms]	SF [%]	LMP16 [ms]	SF [%]
MLP-4-6-6-3	528.183	242.789	54	229.337	56	223.374	57
FCCMLP-4-6-6-3	1851.720	870.468	52	842.894	54	831.464	55

3.4 The Two Spirals Classification

Two spirals is a well-known classification problem where a neural network has to identify one of the two helices based on two-dimensional coordinates. The training set for this problem contains 96 samples. The target error has been set to 0.05. Table 4 shows the simulation results.

Table 4. Training results for the TS function.

Network	LM [ms]	LMP4 [ms]	SF [%]	LMP8 [ms]	SF [%]	LMP16 [ms]	SF [%]
MLP-2-5-5-5-1	166.819	77.954	53	76.139	54	75.555	54
FCMLP-2-5-5-5-1	349.704	165.037	52	161.613	53	161.192	53

4 Conclusion

In this paper, the new computational approach to the Levenberg-Marquardt learning algorithm for a feedforward neural network is proposed. Two types of feedforward neural networks were used in the experiments: multilayer perceptron and fully interconnected multilayer perceptron. The networks were trained with different training sets: Logistic function, Hang, Iris, and Two Spirals. We can compare the computational performance of the proposed solution, based on vector instructions of the Levenberg-Marquardt learning algorithm, with a classical solution. The conducted experiments showed a significant reduction of the real learning time. For all training sets, calculation times have been reduced by an average of 50%. It has been observed that the performance of the proposed solution is promising.

A vector approach can be used for other advanced learning algorithms of feedforward neural networks, see eg. [2,8]. In the future research, we plan to design parallel realization of learning of other structures including probabilistic neural networks [32] and various fuzzy [1,15,20,22,24,37,42,46,47], and neuro-fuzzy structures [16,17,23,25,35].

References

1. Bartczuk, Ł, Przybył, A., Cpałka, K.: A new approach to nonlinear modelling of dynamic systems based on fuzzy rules. Int. J. Appl. Math. Comput. Sci. (AMCS) **26**(3), 603–621 (2016)
2. Bilski, J.: The UD RLS algorithm for training the feedforward neural networks. Int. J. Appl. Math. Comput. Sci. **15**(1), 101–109 (2005)
3. 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). https://doi.org/10.1007/978-3-540-24844-6_19
4. 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). https://doi.org/10.1007/978-3-540-69731-2_2
5. 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. LNCS (LNAI), vol. 6114, pp. 19–25. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13232-2_3

6. 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. LNCS (LNAI), vol. 7267, pp. 12–20. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29347-4_2
7. 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. LNCS (LNAI), vol. 7894, pp. 32–40. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38658-9_3
8. Bilski, J.: Parallel structures for feedforward and dynamical neural networks (in Polish). AOW EXIT (2013)
9. 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. LNCS (LNAI), vol. 8467, pp. 12–21. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07173-2_2
10. Bilski, J., Smoląg, J.: Parallel architectures for learning the RTRN and elman dynamic neural networks. IEEE Trans. Parallel Distrib. Syst. PP(99), (2014). https://doi.org/10.1109/TPDS.2014.2357019
11. Bilski, J., Kowalczyk, B., Marchlewska A., Żurada J.M.: Local levenberg-marquardt algorithm for learning feedforwad neural networks. J. Artif. Intell. Soft Comput. Res. **10**(4), 299–316 (2020). https://doi.org/10.2478/jaiscr-2020-0020
12. Bilski, J., Kowalczyk, B., Marjański, A., Gandor, M., Żurada, J.M.: A novel fast feedforward neural networks training algorithm. J. Artif. Intell. Soft Comput. Res. **11**(4), 287–306 (2021). https://doi.org/10.2478/jaiscr-2021-0017
13. Bilski J., Rutkowski L., Smoląg J., Tao D., A novel method for speed training acceleration of recurrent neural networks. Inf. Sci. **553**, 266–279 (2021). https://doi.org/10.1016/j.ins.2020.10.025
14. Chu, J.L., Krzyzak, A.: The recognition of partially occluded objects with support vector machines, convolutional neural networks and deep belief networks. J. Artif. Intell. Soft Comput. Res. **4**(1), 5–19 (2014)
15. Cpałka, K., Rutkowski, L.: Flexible takagi-sugeno fuzzy systems. In: Proceedings of the International Joint Conference on Neural Networks, Montreal, pp. 1764–1769 (2005)
16. 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)
17. Cpalka, K., Rebrova, O., Nowicki, R. et al.: On design of flexible neuro-fuzzy systems for nonlinear modelling. Int. J. Gener. Syst. **42**(6), Special Issue: SI, 706–720 (2013)
18. Fahlman, S.: Faster learning variations on backpropagation: an empirical study. In: Proceedings of Connectionist Models Summer School, Los Atos (1988)
19. Gabryel, M., Przybyszewski, K.: Methods of searching for similar device fingerprints using changes in unstable parameters. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds.) ICAISC 2020. LNCS (LNAI), vol. 12416, pp. 325–335. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61534-5_29
20. Gabryel, M., Scherer, M.M., Sułkowski, Ł, Damaševičius, R.: Decision making support system for managing advertisers by ad fraud detection. J. Artif. Intell. Soft Comput. Res. **11**, 331–339 (2021)
21. Hagan, M.T., Menhaj, M.B.: Training feedforward networks with the Marquardt algorithm. IEEE Trans. Neuralnetworks **5**(6), 989–993 (1994)

22. 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). https://doi.org/10.1007/978-3-540-69731-2_26
23. Korytkowski, M., Scherer, R.: Negative correlation learning of neuro-fuzzy system. LNAI **6113**, 114–119 (2010)
24. 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. LNCS (LNAI), vol. 7895, pp. 523–534. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38610-7_48
25. 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. LNCS (LNAI), vol. 7894, pp. 329–344. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38658-9_30
26. Marqardt, D.: An algorithm for last-sqares estimation of nonlinear paeameters. J. Soc. Ind. Appl. Math. 431–441 (1963)
27. Niksa-Rynkiewicz, T., Szewczuk-Krypa, N., Witkowska, A., Cpałka, K., Zalasiński, M., Cader, A.: Monitoring regenerative heat exchanger in steam power plant by making use of the recurrent neural network. J. Artif. Intell. Soft Comput. Res. **11**(2), 143–155 (2021). https://doi.org/10.2478/jaiscr-2021-0009
28. Patan, K., Patan, M.: Optimal training strategies for locally recurrent neural networks. J. Artif. Intell. Soft Comput. Res. **1**(2), 103–114 (2011)
29. Riedmiller, M., Braun, H.: A direct method for faster backpropagation learning: the RPROP Algorithm. In: IEEE International Conference on Neural Networks, San Francisco (1993)
30. Romaszewski, M., Gawron, P., Opozda, S.: Dimensionality reduction of dynamic msh animations using HO-SVD. J. Artif. Intell. Soft Comput. Res. **3**(3), 277–289 (2013)
31. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Parallel Distributed Processing, vol. 1, ch. 8, Rumelhart, D.E., McCelland, J. (red.). The MIT Press, Cambridge, Massachusetts (1986)
32. Rutkowski, L.: Multiple Fourier series procedures for extraction of nonlinear regressions from noisy data. IEEE Trans. Sig. Process. **41**(10), 3062–3065 (1993)
33. Rutkowski, L.: Identification of MISO nonlinear regressions in the presence of a wide class of disturbances. IEEE Trans. Inf. Theor. **37**(1), 214–216 (1991)
34. Rutkowski, L., Jaworski, M., Pietruczuk, L., Duda, P.: Decision trees for mining data streams based on the gaussian approximation. IEEE Trans. Knowl. Data Eng. **26**(1), 108–119 (2014)
35. 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. LNCS (LNAI), vol. 6114, pp. 645–650. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13232-2_79
36. Rutkowski, L., Rafajlowicz, E.: On optimal global rate of convergence of some nonparametric identification procedures. IEEE Trans. Autom. Control **34**(10), 1089–1091 (1989)
37. Rutkowski, T., Łapa, K., Jaworski, M., Nielek, R., Rutkowska, D.: On explainable flexible fuzzy recommender and its performance evaluation using the akaike

information criterion. In: Gedeon, T., Wong, K.W., Lee, M. (eds.) ICONIP 2019. CCIS, vol. 1142, pp. 717–724. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36808-1_78
38. Smoląg, J., Bilski, J.: A systolic array for fast learning of neural networks. In: Proceedings of V Conference Neural Networks and Soft Computing, Zakopane, pp. 754–758 (2000)
39. Smoląg, J., Rutkowski, L., Bilski, J.: Systolic array for neural networks. In: Proceedings of IV Conference Neural Networks and Their Applications, Zakopane, pp. 487–497 (1999)
40. 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. LNCS (LNAI), vol. 7895, pp. 242–250. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38610-7_23
41. Starczewski J.T. Advanced Concepts in Fuzzy Logic and Systems with Membership Uncertainty, volume 284 of Studies in Fuzziness and Soft Computing. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-29520-1
42. Starczewski, J.T., Goetzen, P., Napoli, Ch.: Triangular fuzzy-rough set based fuzzification of fuzzy rule-based systems. J. Artif. Intell. Soft Comput. Res. **10**, 271–285 (2020)
43. Tadeusiewicz, R.: Neural Networks (in Polish). AOW RM (1993)
44. Werbos, J.: Backpropagation through time: what it does and how to do it. In: Proceedings of the IEEE, vol. 78, no. 10 (1990)
45. Wilamowski, B.M., Yo, H.: Neural network learning without backpropagation. IEEE Trans. Neural Network. **21**(11), 1793–1803 (2010)
46. 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. LNCS (LNAI), vol. 7895, pp. 342–350. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38610-7_32
47. Zalasiński, M., Łapa, K., Cpałka, K.: Prediction of values of the dynamic signature features. Expert Syst. Appl. **104**, 86–96 (2018)
48. El Zini J., Rizk Y., Awad M.: An optimized parallel implementation of non-iteratively trained recurrent neural networks. Journal of Artif. Intell. Soft Comput. Res. **11**(1), 33–50 (2021). https://doi.org/10.2478/jaiscr-2021-0003