The Parallel Modification to the Levenberg-Marquardt Algorithm

Jarosław Bilski[1](✉), Bartosz Kowalczyk[1], and Konrad Grzanek[2,3]

[1] Institute of Computational Intelligence, Częstochowa University of Technology, Częstochowa, Poland
{Jaroslaw.Bilski,Bartosz.Kowalczyk}@iisi.pcz.pl

[2] Information Technology Institute, University of Social Sciences, Łódź, Poland

[3] Clark University, Worcester, MA 01610, USA
kongra@gmail.com

Abstract. The paper presents a parallel approach to the Levenberg-Marquardt algorithm (also called LM or LMA). The first section contains the mathematical basics of the classic LMA. Then the parallel modification to LMA is introduced. The classic Levenberg-Marquardt algorithm is sufficient for a training of small neural networks. For bigger networks the algorithm complexity becomes too big for the effective teaching. The main scope of this paper is to propose more complexity efficient approach to LMA by parallel computation. The proposed modification to LMA has been tested on a few function approximation problems and has been compared to the classic LMA. The paper concludes with the resolution that the parallel modification to LMA could significantly improve algorithm performance for bigger networks. Summary also contains a several proposals for the possible future work directions in the considered area.

Keywords: Feed-forward neural network
Parallel neural network training algorithm · Optimization problem
Levenberg-Marquardt algorithm · QR decomposition · Givens rotation

1 Introduction

The methods of Artificial Intelligence are broadly used in the modern world. Genetic algorithms, neural networks and their applications are the subject of many researches [11,12,14,15]. The most common use of the modern AI can be found in the areas of optimization, classification and recognition of the given patterns [1,2,13,16,17,21]. The Levenberg-Marquardt algorithm (also called LM or LMA) is a highly efficient and a popular teaching method for feedforward neural networks [9]. It is classified as a quasi-Newton method which finds a wide usage in areas of function minimization problems such as neural networks teaching [8]. One of the biggest downfalls of the LMA is a huge complexity contrary to

L. Rutkowski et al. (Eds.): ICAISC 2018, LNAI 10841, pp. 15–24, 2018.
https://doi.org/10.1007/978-3-319-91253-0_2

classical teaching methods as e.g. the well known Back Propagation algorithm with all its variants [3,7,10]. That limitation makes LMA impractical as a teaching algorithm for bigger networks. The following paper describes the parallel modification for LMA which can be applied to any feedforward multi-layered neural network.

2 The Classic Levenbert-Marquardt Algorithm

The Levenberg-Marquard method is an iterational algorithm mostly used for non-linear function optimization. It combines the advantages of a steepest descent and the Gauss-Newton methods. In order to teach a neural network, LMA is used to minimize the following error function

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

where $\varepsilon_i^{(L)}$ stands for non-linear neuron error and can be depicted as

$$\varepsilon_r^{(L)}(t) = \varepsilon_r^{(Lr)}(t) = y_r^{(L)}(t) - d_r^{(L)}(t) \tag{2}$$

while $d_r^{(L)}(t)$ is the *r-th* expected network response to the *t-th* teaching sample. In the Levenberg-Marquardt algorithm weights delta is given by

$$\Delta\left(\mathbf{w}(n)\right) = -\left[\nabla^{\mathbf{2}}\mathbf{E}\left(\mathbf{w}(n)\right)\right]^{-1}\nabla\mathbf{E}\left(\mathbf{w}(n)\right) \tag{3}$$

where $\nabla\mathbf{E}\left(\mathbf{w}(n)\right)$ stands for the gradient vector

$$\nabla\mathbf{E}\left(\mathbf{w}(n)\right) = \mathbf{J}^T\left(\mathbf{w}(n)\right)\varepsilon\left(\mathbf{w}(n)\right) \tag{4}$$

and $\nabla^{\mathbf{2}}\mathbf{E}\left(\mathbf{w}(n)\right)$ stands for the Hessian matrix

$$\nabla^{\mathbf{2}}\mathbf{E}\left(\mathbf{w}(n)\right) = \mathbf{J}^T\left(\mathbf{w}(n)\right)\mathbf{J}\left(\mathbf{w}(n)\right) + \mathbf{S}\left(\mathbf{w}(n)\right) \tag{5}$$

while $\mathbf{J}\left(\mathbf{w}(n)\right)$ is the Jacobian matrix

$$\mathbf{J}(\mathbf{w}\left(n\right)) = \begin{bmatrix} \frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{10}^{(1)}} & \frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial\varepsilon_1^{(L)}(1)}{\partial w_{N^L N^{L-1}}^{(L)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{10}^{(1)}} & \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{11}^{(1)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(1)}{\partial w_{N_L N_{L-1}}^{(L)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{10}^{(1)}} & \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{10}^{(1)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{ij}^{(k)}} & \cdots & \frac{\partial\varepsilon_{N_L}^{(L)}(Q)}{\partial w_{N_L N_{L-1}}^{(L)}} \end{bmatrix}. \tag{6}$$

The non-linear errors $\varepsilon_i^{(lr)}$ of the neurons in hidden layers are calculated as follows

$$\varepsilon_i^{(lr)}\left(t\right) \triangleq \sum_{m=1}^{N_{l+1}} \delta_i^{(l+1,r)}\left(t\right) w_{mi}^{(l+1)}, \tag{7}$$

$$\delta_i^{(lr)}\left(t\right) = \varepsilon_i^{(lr)}\left(t\right) f'\left(s_i^{(lr)}\left(t\right)\right). \tag{8}$$

That creates a possibility to calculate contents of the Jacobian matrix across all weights in the network

$$\frac{\partial \varepsilon_r^{(L)}\left(t\right)}{w_{ij}^{(l)}} = \delta_i^{(lr)}\left(t\right) x_j^{(l)}\left(t\right). \tag{9}$$

In the classic LMA all weights of a neural network are stored as a single vector. The $\mathbf{S}\left(\mathbf{w}(n)\right)$ factor from Eq. (5) is depicted as

$$\mathbf{S}\left(\mathbf{w}(n)\right) = \sum_{t=1}^{Q} \sum_{r=1}^{N_L} \varepsilon_r^{(L)}\left(t\right) \nabla^2 \varepsilon_r^{(L)}\left(t\right). \tag{10}$$

In the Gauss-Newton method it is assumed that $\mathbf{S}\left(\mathbf{w}(n)\right) \approx 0$, so the Eq. (3) can be simplified

$$\varDelta\left(\mathbf{w}(n)\right) = -\left[\mathbf{J}^T\left(\mathbf{w}(n)\right)\mathbf{J}\left(\mathbf{w}(n)\right)\right]^{-1}\mathbf{J}^T\left(\mathbf{w}(n)\right)\varepsilon\left(\mathbf{w}(n)\right). \tag{11}$$

For the Levenberg-Marquardt algorithm needs it is assumed that $\mathbf{S}\left(\mathbf{w}(n)\right) = \mu\mathbf{I}$ so the Eq. (3) takes the form

$$\varDelta\left(\mathbf{w}(n)\right) = -\left[\mathbf{J}^T\left(\mathbf{w}(n)\right)\mathbf{J}\left(\mathbf{w}(n)\right) + \mu\mathbf{I}\right]^{-1}\mathbf{J}^T\left(\mathbf{w}(n)\right)\varepsilon\left(\mathbf{w}(n)\right). \tag{12}$$

Let

$$\begin{gathered}\mathbf{A}\left(n\right) = -\left[\mathbf{J}^T\left(\mathbf{w}(n)\right)\mathbf{J}\left(\mathbf{w}(n)\right) + \mu\mathbf{I}\right] \\ \mathbf{h}\left(n\right) = \mathbf{J}^T\left(\mathbf{w}(n)\right)\varepsilon\left(\mathbf{w}(n)\right)\end{gathered} \tag{13}$$

so the Eq. (12) can be depicted as

$$\varDelta\left(\mathbf{w}(n)\right) = \mathbf{A}(n)^{-1}\mathbf{h}\left(n\right). \tag{14}$$

At this stage the QR decomposition can be used to solve Eq. (14). This results with obtaining a desired weight update vector $\varDelta\left(\mathbf{w}(n)\right)$.

$$\mathbf{Q}^T\left(n\right)\mathbf{A}\left(n\right)\varDelta\left(\mathbf{w}(n)\right) = \mathbf{Q}^T\left(n\right)\mathbf{h}\left(n\right), \tag{15}$$

$$\mathbf{R}\left(n\right)\varDelta\left(\mathbf{w}(n)\right) = \mathbf{Q}^T\left(n\right)\mathbf{h}\left(n\right). \tag{16}$$

For the results presented in Sect. 4 the QR decomposition based on Givens rotations has been used. The summary of the Levenberg-Marquardt teaching algorithm can be presented in 5 steps:

1. Calculate network outputs and errors across all teaching samples and solve the goal criterion.
2. Run backpropagation method and calculate the jacobian matrix.
3. Perform QR decomposition in order to obtain $\varDelta(\mathbf{w}(n))$.
4. Apply $\varDelta(\mathbf{w}(n))$ for a neural network and calculate the goal criterion once again. If the new error is smaller than the original one commit the weights, divide μ by β and proceed to step 1. If the new error is bigger than the original one, multiply μ by β and go back to step 3.
5. The algorithm is deemed to be finished once the gradient is reduced below the accepted threshold or the network error satisfies the predefined error goal.

3 Parallel Modification

As shown in the previous section (especially in Eq. (6)), the classic LMA requires computing a jacobian, which is a $[no \cdot np] \times [nw]$ matrix of error derivatives. Where no stands for a number of network outputs, np is a number of teaching samples and nw is a total number of weights in considered network. Then the Eq. (14) needs to be solved what involves the inversion of $[nw]$ x $[nw]$ size matrix. The clue of a presented modification is to create a set of jacobians, unique for each neuron of the network. Then the computation can be done in parallel for all neurons. In such case the algorithm complexity is significantly reduced to the biggest weight vector for the respective neuron.

The neural network shown in Fig. 1 is taken into further considerations. The network contains 2 inputs and 3 layers. The two hidden layers consist of 100 neurons while the output layer contains a single neuron. To simplify the calculations teaching vector is assumed to contain 100 samples. In the classic Levenberg-Marquardt algorithm (from now on also called LMC) the jacobian size is the following

$$[no \cdot np] \times \left[\sum_{l=1}^{L} N_l \cdot (N_{l-1} + 1) \right]. \tag{17}$$

By filling the above equation with considered network's values the jacobian size calculates as follows

$$[1 \cdot 100] \times [100 \cdot 3 + 100 \cdot 101 + 1 \cdot 101] = [100] \times [10501]. \tag{18}$$

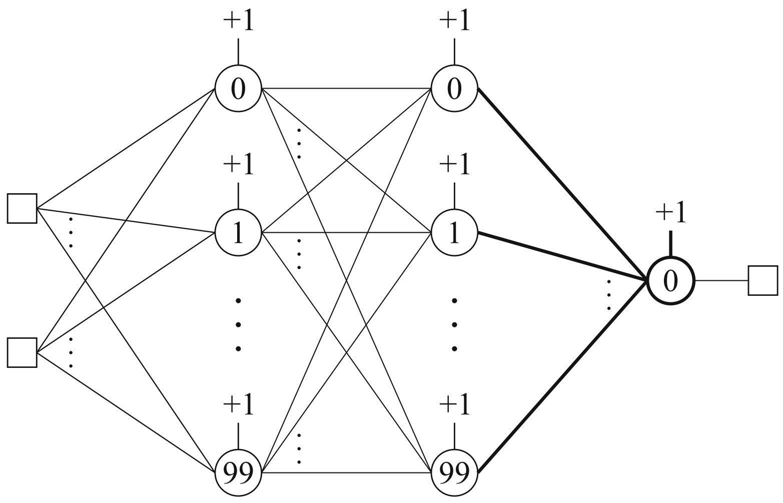

Fig. 1. The considered neural network. One of the neurons with the biggest weight vector and its connections are highlighted.

In a parallel approach to the Levenberg-Marquardt algorithm (from now on also called LMP) the separate jacobian matrix is created for each neuron of the network so it's size can be depicted as

$$[np] \times [\max(N_{l-1} + 1)], \tag{19}$$

which in considered network translates into the biggest jacobian of size

$$[100] \times [101]. \tag{20}$$

It is easy to see that the smaller jacobian matrix is, the faster QR decomposition is completed. Table 1 shows the comparison of jacobians sizes across all layers in the considered network for both LMC and LMP. Table 2 presents the average times of solving Eq. 14 for matrices of size $[10501] \times [10502]$ and $[101] \times [102]$. Intel core i7 CPU with the frequency set to 4.40 GHz has been used for computation.

Table 1. Sizes of jacobians in LMC and LMP comparison (LMP is divided into layers).

LMC	LMP		
—	1	2	3
$[100] \times [10501]$	$[100] \times [3]$	$[100] \times [101]$	$[100] \times [101]$

Table 2. Average times for solving Eq. (14) for matrices of size $[10501] \times [10502]$ and $[101] \times [102]$ in milliseconds.

LMC	LMP
628791,2	0,7942649

4 Results

4.1 Approximating $y = 4x(1 - x)$ Function

The first teaching problem is a logistic curve approximation given by the formula $y = 4x(1 - x)$. A fully connected 1-5-1 network has been used. Teaching set consists of 11 samples to cover the argument range in $x \in [0, 1]$. As a teaching goal the maximum error of value 0.001 has been set. The best teaching results LMP algorithm achieved with the narrow initial weight values $w_{init} \in [-0.5, 0.5]$ and the small value of $\beta \in [1.2, 4]$ factor. Figure 2 shows the networks outputs trained by LMC, LMP and the expected curve. Figure 3 shows the error for all teaching samples after a neural network training with LMP algorithm.

4.2 Approximating $y = \sin x \cdot \log x$ Function

The second teaching problem is a curve approximation given by the formula $y = \sin x \cdot \log x$. A fully connected 1-15-1 network has been used. Teaching set consists of 40 samples to cover the argument range in $x \in [0.1, 4]$. As a teaching

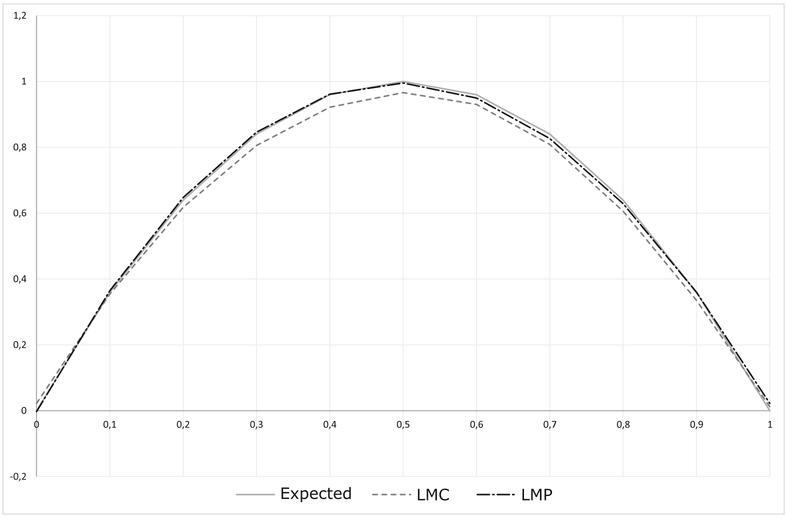

Fig. 2. Comparison of network outputs for $y = 4x(1 - x)$ function approximation problem. Teaching target is set as max epoch error < 0.001.

goal the maximum error of value 0.001 has been set. The best teaching results LMP algorithm achieved with the narrow initial weight values $w_{init} \in [-0.5, 0.5]$ and the small value of $\beta \in [1.2, 4]$ factor. Figure 4 shows the networks outputs trained by LMC, LMP and the expected curve. Figure 5 shows the error for all teaching samples after a neural network training with LMP algorithm.

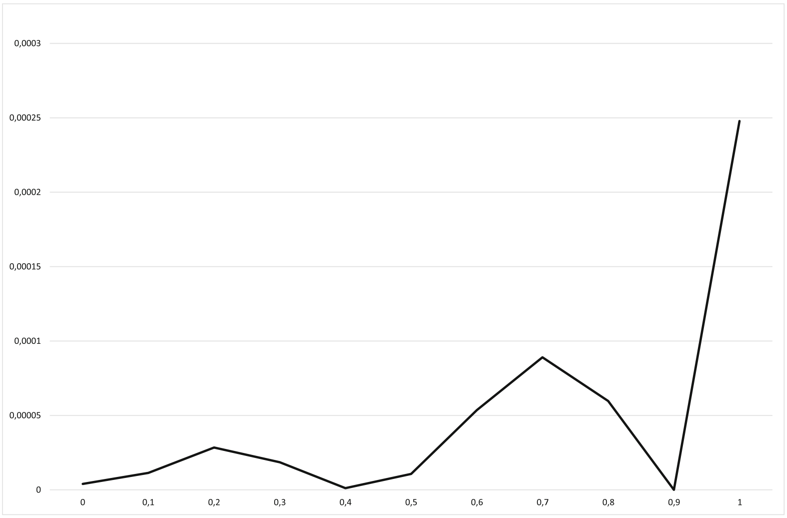

Fig. 3. Network error for $y = 4x(1 - x)$ function approximation problem trained by LMP. Teaching target is set as max epoch error < 0.001.

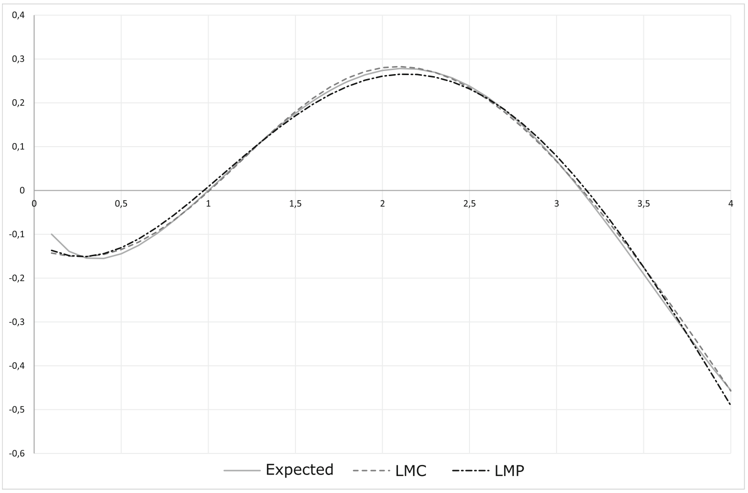

Fig. 4. Comparison of network outputs for $f(x) = \sin x \cdot \log x$ function approximation problem. Teaching target is set as max epoch error < 0.001.

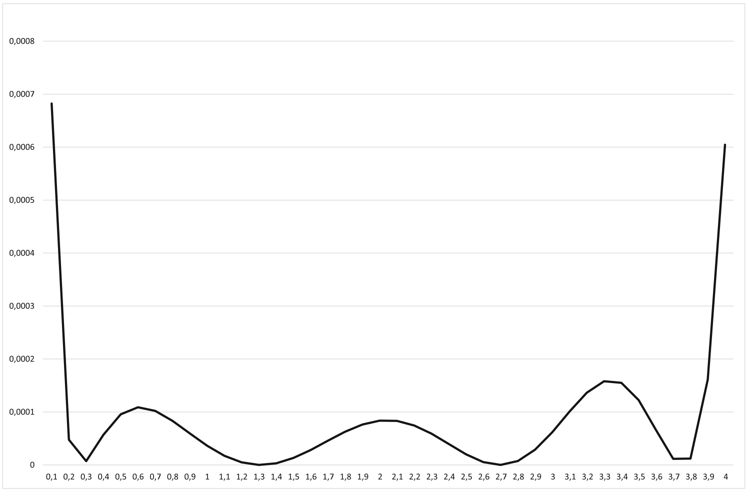

Fig. 5. Network error for $f(x) = \sin x \cdot \log x$ function approximation problem trained by LMP. Teaching target is set as max epoch error < 0.001.

5 Conclusion

Parallelization of the teaching algorithms seems to be a good direction for neural networks training optimization. This topic has been approached multiple times by many researchers e.g. [5,18–20]. The parallel approach to the popular Levenberg-Marquardt neural network teaching algorithm has been presented in this paper. In the classic form the LMA is characterized by a very high computational complexity which goes to the square of a network size. The presented parallel modification to the Levenberg-Marquardt algorithm seems to overcome this limitation maintaining the reasonable accuracy and performance of the archetype. As shown in Sect. 4 in both approximation problems, networks trained by LMP achieved the assumed requirements. The LMP method seems to be a good direction for Levenberg-Marquardt algorithm optimization.

In the near future additional consideration in this matter can be taken. First, the complete parallel implementation of the LMP algorithm can be performed e.g. using CUDA platform and GPU capabilities. Then, more complex approximation problems can be trained e.g. higher dimension functions. Also in deep analysis of influence of parameter β and initial network weights can be performed. Finally LMP implementation can be compared to similar neural network training algorithm parallel optimizations e.g. [4,6].

References

1. Starczewski, A.: A new validity index for crisp clusters. Pattern Anal. Appl. **20**(3), 687–700 (2015)
2. Starczewski, A., Krzyżak, A.: Improvement of the validity index for determination of an appropriate data partitioning. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) ICAISC 2017. LNCS (LNAI), vol. 10246, pp. 159–170. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59060-8_16
3. Bilski, J., Wilamowski, B.M.: Parallel Levenberg-Marquardt algorithm without error backpropagation. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) ICAISC 2017. LNCS (LNAI), vol. 10245, pp. 25–39. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59063-9_3
4. Bilski, J., Kowalczyk, B., Żurada, J.M.: Parallel implementation of the givens rotations in the neural network learning algorithm. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) ICAISC 2017. LNCS (LNAI), vol. 10245, pp. 14–24. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59063-9_2
5. Bilski, J., Smoląg, J.: Parallel realisation of the recurrent RTRN neural network learning. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 11–16. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69731-2_2
6. Bilski, J., Smoląg, J.: Parallel architectures for learning the rtrn and elman dynamic neural network. IEEE Trans. Parallel Distrib. Syst. **26**(9), 2561–2570 (2015)
7. Bilski, J., Smoląg, J., Żurada, J.M.: Parallel approach to the Levenberg-Marquardt learning algorithm for feedforward neural networks. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) ICAISC 2015. LNCS (LNAI), vol. 9119, pp. 3–14. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19324-3_1
8. Marqardt, D.: An algorithm for last-sqares estimation of nonlinear paeameters. J. Soc. Ind. Appl. Math. **11**(2), 431–441 (1963)
9. Hagan, M.T., Menhaj, M.B.: Training feedforward networks with the Marquardt algorithm. IEEE Trans. Neural Netw. **5**(6), 989–993 (1994)
10. Werbos, J.: Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University, Cambridge (1974)
11. Cpałka, K., Łapa, K., Przybył, A.: A new approach to design of control systems using genetic programming. Inf. Technol. Control **44**(4), 433–442 (2015)
12. Łapa, K., Cpałka, K.: On the application of a hybrid genetic-firework algorithm for controllers structure and parameters selection. In: Borzemski, L., Grzech, A., Świątek, J., Wilimowska, Z. (eds.) Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part I. AISC, vol. 429, pp. 111–123. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28555-9_10
13. Łapa, K., Cpałka, K., Galushkin, A.I.: A new interpretability criteria for neuro-fuzzy systems for nonlinear classification. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) ICAISC 2015. LNCS (LNAI), vol. 9119, pp. 448–468. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19324-3_41

14. Khan, N.A., Shaikh, A.: A smart amalgamation of spectral neural algorithm for nonlinear Lane-Emden equations with simulated annealing. J. Artif. Intell. Soft Comput. Res. **7**(3), 215–224 (2017)
15. Liu, H., Gegov, A., Cocea, M.: Rule based networks: an efficient and interpretable representation of computational models. J. Artif. Intell. Soft Comput. Res. **7**(2), 111–123 (2017)
16. Notomista, G., Botsch, M.: A machine learning approach for the segmentation of driving Maneuvers and its application in autonomous parking. J. Artif. Intell. Soft Comput. Res. **7**(4), 243–255 (2017)
17. Rotar, C., Lantovics, L.B.: Directed evolution - a new Metaheuristc for optimization. J. Artif. Intell. Soft Comput. Res. **7**(3), 183–200 (2017)
18. Rutkowska, D., Nowicki, R., Hayashi, Y.: Parallel processing by implication-based neuro-fuzzy systems. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 599–607. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-48086-2_66
19. Smoląg, J., Bilski, J.: A systolic array for fast learning of neural networks. In: V NNSC, pp. 754–758 (2000)
20. Smoląg, J., Bilski, J., Rutkowski, L.: Systolic array for neural networks. In: IV KSNiIZ, pp. 487–497 (1999)
21. Villmann, T., Bohnsack, A., Kaden, M.: Can learning vector quantization be an alternative to SVM and deep learning? Recent trends and advanced variants of learning vector quantization for classification learning. J. Artif. Intell. Soft Comput. Res. **7**(1), 65–81 (2017)