JAISCR, 2023, Vol. 13, No. 2, pp. 45 – 61
10.2478/jaiscr-2023-0006

FAST COMPUTATIONAL APPROACH TO THE LEVENBERG-MARQUARDT ALGORITHM FOR TRAINING FEEDFORWARD NEURAL NETWORKS

Jarosław Bilski[1,*], Jacek Smoląg[1], Bartosz Kowalczyk[1],
Konrad Grzanek[2], Ivan Izonin[3]

[1]*Department of Computational Intelligence, Częstochowa University of Technology, al. Armii Krajowej 36, 42-200 Częstochowa, Poland*

[2]*Institute of Information Technologies, University of Social Sciences, ul. Sienkiewicza 9, 90-113 Łódź, Poland*

[3]*Department of Artificial Intelligence, Lviv Polytechnic National University Lviv, 79905, Ukraine*

[*]*E-mail: jaroslaw.bilski@pcz.pl*

Submitted: 19th October 2022; Accepted: 15th February 2023

Abstract

This paper presents a parallel approach to the Levenberg-Marquardt algorithm (LM). The use of the Levenberg-Marquardt algorithm to train neural networks is associated with significant computational complexity, and thus computation time. As a result, when the neural network has a big number of weights, the algorithm becomes practically ineffective. This article presents a new parallel approach to the computations in Levenberg-Marquardt neural network learning algorithm. The proposed solution is based on vector instructions to effectively reduce the high computational time of this algorithm. The new approach was tested on several examples involving the problems of classification and function approximation, and next it was compared with a classical computational method. The article presents in detail the idea of parallel neural network computations and shows the obtained acceleration for different problems.

Keywords: feed-forward neural network, neural network learning algorithm, Levenberg-Marquardt algorithm, QR decomposition, Givens rotation.

1 Introduction

Currently, artificial intelligence (AI) is widely used both in science research and in industry. Among the issues of artificial intelligence, neural networks (NN) deserve special attention. Continually many researchers publish a lot of scientific papers about artificial intelligence [1, 2], especially about neural networks e.g. [3, 4, 5, 6, 7, 8, 9, 10, 11]. There are more and more advanced AI solutions and methods that are willingly used in industry and various products. Important areas of neural networks applications are health care and medicine [12, 13, 14, 15], finances [16, 17, 18], but also safety [19, 20, 21] and entertainment [22, 23].

Each neural network can perform different tasks. In order for a given NN to carry out a specific task, it must be properly trained. This is done through the training algorithm. The most famous is the backpropagation method [24] Other methods have also been derived from it [25, 26, 27]. More complex algorithms based on Newton's method have also been developed, an example is proposed in [28] the Levenberg-Marquardt (LM) algorithm.

The LM algorithm is one of the most popular methods used for supervised training feedforward (FF) neural networks. These neural networks are made up of a number of layers and they contain neurons. The first layer of the network is the input layer and the last layer is the output layer. All layers except the output layer are hidden layers. Sometimes single-layer networks are used, but for most applications, networks consist of more than one layer. There are various topologies for feedforward neural networks. The most used is the multilayer perceptron (MLP). In such a network, the first layer connects to the network input, each successive layer connects to the previous layer only, and the output from the last layer is the network output. Figure 1 presents an example of the MLP network. The other FF topology is the fully connected multilayer perceptron (FCMLP). This type of network differs from the classic MLP in that the layers connect to the outputs of all previous layers. Figure 2 presents an example of the FCMLP network. It can be readily seen that the FCMLP network with the same number of neurons contains more weights than a standard MLP network, which can often reduce the size of the network, and yet the network can be trained effectively. It is worth noting that the MLP network is a special case of the FCMLP network.

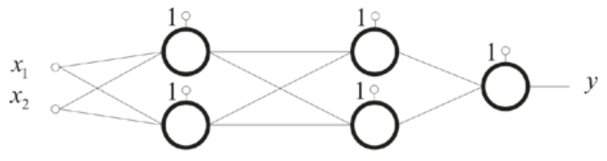

Figure 1. Sample MLP neural network.

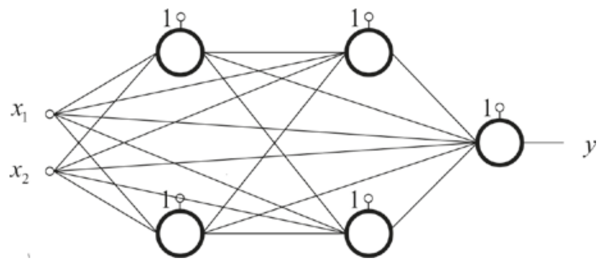

Figure 2. Sample FCMLP neural network.

The LM algorithm is a very popular and reliable method of finding the minimum of functions in most applications, unfortunately, it has several disadvantages. Sometimes they make the LM algorithm too computationally expensive and impractical. For many years, researchers have made various attempts to optimize this algorithm.

The LM algorithm belongs to the second-order methods and joins the advantages of two methods: the Gauss-Newton method and the steepest descent method. Unfortunately, like most neural network training algorithms, the LM algorithm can also get stuck at a local minimum. In first-order methods, one can try to solve this problem by using the momentum factor. This approach allows you to jump over local minima and find the right direction towards the optimal solution point. The momentum factor value can be set arbitrarily and does not change during network training, selected from several values depending on the current gradient, or change dynamically during network training. This attempt is made in the article [29]. The authors combined the advantages of the LM and CG methods and developed two variants of the Levenberg-Marquardt algorithm with a constant and adaptable momentum value. The developed algorithms are more effective in terms of learning time than the classic LM, but both have high computational complexity.

In the case of flat places of the error function, classical methods of teaching neural networks achieve a very low coefficient of convergence. This results in a significant slowing down of training due to low values of the gradient of hidden neurons. First-order methods, such as the steepest descent, use fine-tuning of training parameters. For more complex second-order methods such as the LM algorithm, the computational complexity is very high and the use of additional training parameters is avoided. In the paper [30], the authors, to overcome the impasse in the convergence process caused by the flat error function, proposed a modification involving the compression of weights in the Levenberg-Marquardt algorithm. This technique is used to draw neuronal gradients into a non-linear area of the activation function in order to accelerate training. The authors noted a significant improvement in the success rate compared to the classic variant of the LM algorithm. It should be noted that the proposed modification does not significantly increase the computational complexity in the feedforward network topology.

In the work [31] the authors noticed that the Jacobian matrix sensitivity coefficients calculated using numerical differentiation methods give approximate values of gradient derivatives. In some complex problems with transient states and severe non-linearities, this can lead to considerable instability in the learning process. Thus, the Jacobian matrix of the LM algorithm must be computed as accurately as possible. The authors proposed a complex method of variable differentiation to compute the Jacobian matrix. This solution increases the stability of the LM training process. Unfortunately, the computational complexity remains unchanged.

It is also easy to see that in the classical Levenberg-Marquardt algorithm, the size of the Jacobian matrix is the main cause of computational complexity. In the work [32], the authors present a modification of the LM algorithm for recording non-rigid images. They proposed to use the Jacobian matrix once determined in two successive iterations of the algorithm, instead of just one. After the classic calculation step of the LM algorithm, an extra step is performed to establish a new correction vector. Additionally, the linear search was used in the calculations to improve performance. The presented method is more efficient than the classic LM algorithm because the Jacobian matrix is computed only once every two iterations of the algorithm.

The original LM algorithm is extremely efficient in training small neural networks. In the case of larger neural networks, the computational complexity increases significantly due to the increase in the size of the Jacobian matrix. As a result, this method becomes ineffective and rarely used. In the work [9], the authors present a local modification of the Levenberg-Marquardt algorithm. They resign from computing a very large Jacobian matrix for the entire network. Instead, there are many small Jacobian matrices for individual neurons in the network. As a result, the training time was reduced by several to tens of times. Additionally, the number of epochs needed to train the network has been reduced.

It is clear from the above discussion that many attempts have been made to optimize the classical Levenberg-Marquardt algorithm. The main problem of the LM algorithm is the relatively long training time for larger networks. It is related to the size of the Jacobian matrix. This structure increases with larger networks, especially when training using very long training sets. In such cases, the LM algorithm becomes impractical due to the high computational complexity and too long a training time.

The article presents a new approach to calculations in the LM algorithm. It is based on the use of vector calculations to determine several successive steps in multi-step epochs of the training process. Thanks to this approach, the computation time is significantly reduced. The following original and innovative contributions have been made during the research:

1. The vector computational approach to the classic Levenberg-Marquardt algorithm has been presented.

2. The consecutive steps of the vector LM algorithm have been precisely described.

3. The performance of the vector computa-

tional approach to the LM has been compared with that of the classic Levenberg-Marquardt algorithm.

4. The original benchmarking procedure was developed to obtain results for vectors of various sizes.

5. Both, the vector and classic computational approach of the LM have been tested on various topologies of feedforward networks utilizing multiple benchmarks.

6. The proposed computational approach allows for a significant reduction in the computation time of the LM algorithm, and can also be applied to most of its modifications.

The article consists of several parts. Chapter 2 details the classical Levenberg-Marquardt algorithm. It includes both a mathematical and practical approach to implementation. Chapter 3 presents the idea behind this article, which is a discussion of the parallelism in the LM algorithm. Fundamental differences from the classical variant and the possibility of vector implementation are emphasized. In Chapter 4, the original vector approach was applied to some test problems, and then the test results were presented. Chapter 5 summarizes the proposed solution and the results obtained and presents possible directions for future research.

2 The classic Levenberg-Marquardt algorithm

The Levenberg-Marquardt (LM) second-order algorithm is used to train feed-forward neural networks. the LM algorithm can adjust the training speed according to the shape of the error function. Use the steepest descent methods as well as the quasi-Newton methods. The LM algorithm uses the loss function defined by the following equation:

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

where Q is the number of samples, N_L is number of network outputs and $\varepsilon_r^{(L)}$ is a non-linear neuron error defined as

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

and $d_r^{(L)}(t)$ is the $r-th$ desired vector of the $t-th$ training sample. Since the LM algorithm is a modification of Newton's method, it uses the first three elements of Taylor series expansion of the loss function. The weight change is calculated as follows

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

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

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

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

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

The $\mathbf{J}\left(\mathbf{w}(n)\right)$ in (4) and (5) is the Jacobian matrix

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

In the hidden layers, the neurons non-linear errors $\varepsilon_i^{(lr)}$ are calculated using the following formula:

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

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

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

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

Note that the derivatives (9) are computed in an analogous way as in the classical error backpropagation method, with the difference that only one error is given to the network output each time. All weights of the network are stored in a single vector, and their derivatives create the Jacobian $\mathbf{J}$ matrix.

The $\mathbf{S}(\mathbf{w}(n))$ element in equation (5) is defined as

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

In the Gauss-Newton method, we can assume the simplification that $\mathbf{S}(\mathbf{w}(n)) \approx 0$, which makes the equation (3) takes the following form

$$\begin{aligned}&\Delta(\mathbf{w}(n)) = \\ &= -\left[\mathbf{J}^T(\mathbf{w}(n))\mathbf{J}(\mathbf{w}(n))\right]^{-1}\mathbf{J}^T(\mathbf{w}(n))\boldsymbol{\varepsilon}(\mathbf{w}(n)).\end{aligned} \tag{11}$$

The Levenberg-Marquardt algorithm updates the weights once at the end of each epoch only. At this point, the entire Jacobian matrix is already computed by the equations (6), (7), (8), and (9). In the Levenberg-Marquardt algorithm, otherwise, the Gauss-Newton method assumes that $\mathbf{S}(\mathbf{w}(n)) = \mu\mathbf{I}$. Hence the equation (3) takes the form

$$\begin{aligned}&\Delta(\mathbf{w}(n)) = \\ &= -\left[\mathbf{J}^T(\mathbf{w}(n))\mathbf{J}(\mathbf{w}(n)) + \mu\mathbf{I}\right]^{-1}\cdot \\ &\cdot\mathbf{J}^T(\mathbf{w}(n))\boldsymbol{\varepsilon}(\mathbf{w}(n)).\end{aligned} \tag{12}$$

To determine the value of weights corrections, the (12) equation will be presented in the matrix form

$$\Delta(\mathbf{w}(n)) = \mathbf{A}(n)^{-1}\mathbf{h}(n), \tag{13}$$

where the matrices $\mathbf{A}$ and $\mathbf{h}$ are defined by

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

$$\mathbf{h}(n) = \mathbf{J}^T(\mathbf{w}(n))\boldsymbol{\varepsilon}(\mathbf{w}(n)). \tag{15}$$

The resulting equation (13) is solved using QR decomposition. It is an iterative method for converting any non-singular matrix to the product of the upper triangular matrix $\mathbf{R}$ and the orthogonal matrix $\mathbf{Q}$. The conversion is done using the following equations

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

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

Seeing that $\mathbf{R}$ is the matrix of the upper triangle, solving the equation (17) is relatively simple and yields the weight update vector $\Delta(\mathbf{w}(n))$. The QR decomposition is accomplished by using Givens rotation as shown in [33].

The Levenberg-Marquardt algorithm described above can be presented in the following steps:

1. The calculation of the network outputs, errors, and loss function for all input data from the training set.

2. The calculation of the whole Jacobian matrix, using the error backpropagation method for each output error individually.

3. The calculation of weight changes vector $\Delta(\mathbf{w}(n))$ using the QR decomposition.

4. The recalculation of the value of the loss function (1) for the newly obtained weights $\mathbf{w}(n) + \Delta(\mathbf{w}(n))$. If the loss function is less than that calculated previously in step 1, μ is divided by β, the weights vector is updated and the algorithm goes to the next epoch in step 1. Otherwise, μ is multiplied by β and the algorithm goes again to step 3 within the same epoch.

5. Stopping the LM algorithm when the loss function drops below the preset value or the gradient drops below the preset value.

3 Fast computational approach to the Levenberg-Marquardt algorithm

In practice, the biggest problem of the LM algorithm is the training time of larger neural networks resulting from the large size of the Jacobian matrix. This Section presents the idea and explanation of how to speed up the LM algorithm using vector instructions from modern processors. A similar result can also be achieved by using multi-core processors, but in this work, only one vector core of the processor was used, which allowed avoiding thread synchronization and freed up the remaining cores for other tasks.

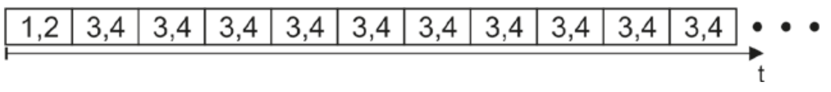

Figure 3. Sample illustration for computational steps in LM algorithm.

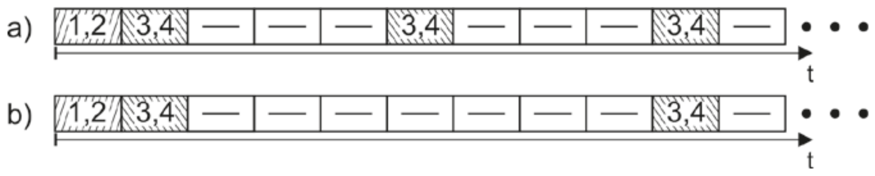

Figure 4. Sample illustration for calculating method with vector instructions. a) the 4-elements vector, b) the 8-elements vector

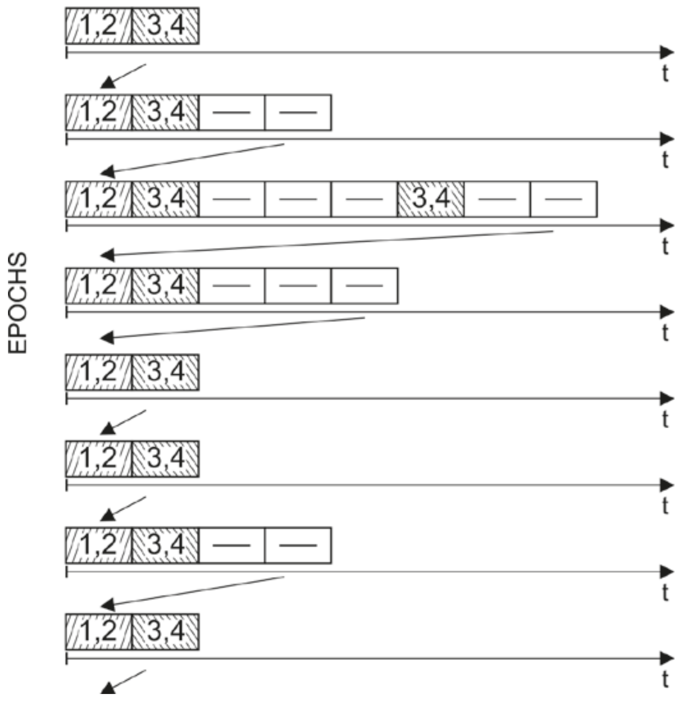

Figure 5. Sample illustration for training process with vector instructions.

Figure 3 shows the steps of one epoch of the LM algorithm, you can see the initial two steps and repeating steps 3 and 4.

The Levenberg-Marquardt algorithm is a method that requires relatively high computing power per epoch. Each epoch consists of several steps (see the previous Section), starting with two steps 1 and 2, and the next steps 3 and 4 are repeated as many times until the loss function value is reduced, finally, in the last step, the end of training criterion is checked.

It is easy to see that the successive pairs of steps 3 and 4 are independent of each other, which means that several such pairs of steps can be performed simultaneously. These pairs have the same starting point and differ only in the value of the μ parameter. This means that they can be run in parallel on different processors or separate cores of the same processor. However, this article proposes a solution that uses vector instructions from modern processors. The use of vector instructions makes it possible to execute 4, 8, and even 16 operations in parallel. The use of this approach allows us to determine in parallel the new 4, 8, or 16 points in the weighing space using only one processor core, see Figure 4. The LM algorithm using four-element vectors is shown in Figure 4a.

After completing the first two steps, the LM algorithm simultaneously carries out steps 3 and 4 for the next 4 (8, 16) μ parameters. Thus, the three consecutive computations of steps 3 and 4 are performed earlier in the computation time of the first pair of steps 3 and 4, therefore they do not consume CPU computing time. The rectangles with the line in the middle symbolize steps 3 and 4, which in the standard calculation method are normally performed sequentially, and in the presented approach they are calculated using vector instructions simultaneously with the first pair of steps 3 and 4, and therefore do not require additional time. The figure 4b shows the version of the LM algorithm that uses eight element vectors.

An exemplary training process using the LM algorithm is shown in Figure 5. You can clearly see that successive epochs have a different number of repetitions of steps 3 and 4. There are also epochs where the next repetition does not occur even once and there are epochs that have many repetitions, in this case, it is possible to use vector instructions, which allows you to calculate up to four pairs of steps 3 and 4 in parallel and consequently shorten the training time. Naturally, it is possible to use eight- or sixteen-element vectors instead of four-element vectors. This increases the parallelism of calculations and the speed of the proposed method. It should also be remembered that by increasing the size of the vector, memory consumption also increases.

4 Simulation results

In order to test the proposed fast method of computing the Levenberg-Marquard algorithm, a test procedure was developed. A detailed description of this procedure is provided in the 4.1 subsection. The main purpose of the presented tests is to compare the quick calculation of the Levenberg-Marquard algorithm with the classical one. The selected tests cover various common problems that can be solved by neural networks. A set of tests was prepared to cover the approximation of one- and two-dimensional functions and various examples of classification.

Various network topologies were also used during the tests. This includes classic multilayer perceptron networks and fully connected networks. In order to increase the transparency of the presented results, a consistent nomenclature of the network topology was used. A multi-layer perceptron containing L layers containing n_l ($l \in [1, \ldots, L]$) neurons in each of them is labeled " $\mathrm{MLP}[-n_l] - L$ ". The same network with additional connections to all previous layers (not only to the previous one) is additionally preceded by the tag "FC", which means "Fully Connected".

4.1 Test methodology

Practical implementations of neural networks and training algorithms contain a number of parameters that are used to control the learning process. The values of these parameters are set before starting the training process. Some of them work as constant, eg training target, error criterion, epoch limit, etc. Other parameters can be modified by the training algorithms during their operation. In the LM method, μ is such a variable parameter, it is initialized with β. Depending on the selected set of parameters, the training may be successful or not. A uniform methodology was used to prepare stable and reproducible results for each performed test.

Several common parameters were used in all tests. They are listed in the 1 table. In order for the obtained statistical data to be correct, each test was repeated 100 times. The test is successful when the network failure criterion reaches a predefined error threshold (set individually for each benchmark). The test fails if the epoch limit is reached before the criterion is converged. During each training process, sample sets are presented randomly. For all training processes, weights are randomly selected from $[-0.5, 0.5]$.

Table 1. Common experiment parameters.

Max number of epoch	1000
Number of experiment	100
Sequence of samples	Random
Starting weights range	$[-0.5, 0.5]$
The β factor	4

The tables presented in the following subsections contain the average training times for 100 repetitions expressed in milliseconds $[ms]$ for 100 repetitions for the classical method of calculating the LM algorithm and the proposed vector method for 4-, 8- and 16-element vectors, which was marked as LMP4, LMP8, and LMP16 respectively. The "AF" acceleration factor expressed as a percentage [%] shows how much the training time has been shortened for a given case. The acceleration factor is given by the formula

$$AF = \left(1 - \frac{LMPx}{LM}\right) * 100\% \qquad (18)$$

The tests were performed for two network topologies, the MLP and the FCMLP.

4.2 Approximation

Approximation tests are intended to simulate the f relation between the sets X and Y, which is formally formulated as $f : \mathbf{X} \to \mathbf{Y}$. Classically, such a relation is the f function, which can be written in a formula and implemented as a computer function. Unfortunately, in some complicated cases, it may be very difficult or impossible to give an unambiguous relationship between these sets. Given the set of X system inputs and Y of known and corresponding outputs, a training sequence for a neural network can be created that can map the set X to the set Y. In the following subsections, various methods of computing the LM algorithm are used to simulate the responses of selected nonlinear functions.

4.2.1 The logistic function

The unary logistic function is represented by the formula

$$f(x) = 4x(1-x) \quad x \in [0,1]. \qquad (19)$$

The training set contains 11 samples for function arguments in the range $x \in [0,1]$. The test initial parameters are listed in Table 2.

Table 2. Initial parameters for the logistic function training.

Expected error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	11

The simulation results for the two types of neural networks MLP and FCMLP are presented in the Table 3. Both networks have two layers with five neurons in the hidden layer. The designations LM, LMP4, LMP8, and LMP16 correspond to the average network training time using the LM algorithm and its three vector versions for 4, 8, and 16-element vectors, respectively.

Table 3. Training results for the LOG function.

Network		MLP 1-5-1	FCMLP 1-5-1
LM	[ms]	0.880	0.588
LMP4	[ms]	0.440	0.311
AF	[%]	50.0	47.1
LMP8	[ms]	0.434	0.306
AF	[%]	50.7	48.1
LMP16	[ms]	0.433	0.305
AF	[%]	50.8	48.1

It is easy to see that the FCMLP network in all cases takes less time to be properly trained than the MLP network. Moreover, the acceleration coefficients have similar values, improving slightly with increasing vector size.

4.2.2 The composite function

In this test the following unary composite function is trained

$$y = \begin{cases} -\cos(x) & for\ x \in \langle 0, \pi\rangle \\ -\cos(3x) & for\ x \in (\pi, 2\pi\rangle \end{cases} \tag{20}$$

The training set contains 23 samples for function arguments in the range $x \in [0, 2\pi]$. The test initial parameters are listed in Table 4.

Table 4. Initial parameters for the composite function training.

Expected error	0.01
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	23

The simulation results are presented in the Table 5. Both networks MLP and FCMLP have two layers with ten neurons in the hidden layer.

Table 5. Training results for the composite function.

Network		MLP 1-10-1	FCMLP 1-10-1
LM	[ms]	83.791	80.128
LMP4	[ms]	43.411	40.839
AF	[%]	48.2	49.0
LMP8	[ms]	41.627	39.813
AF	[%]	50.3	50.3
LMP16	[ms]	41.606	39.809
AF	[%]	50.3	50.3

The obtained acceleration factors are close to those obtained for the logistic function.

4.2.3 The two-argument Hang function

The Hang function is a nonlinear two-dimensional function with the following formula

$$f(x_1, x_2) = \left(1 + x_1^{-2} + \sqrt{x_2^{-3}}\right)^2 \qquad x_1, x_2 \in [1, 5]. \tag{21}$$

In this test, the training set contains 50 samples which are in the range $x_1, x_2 \in [1, 5]$. The Hang test initial parameters are shown in the Table 6.

Table 6. Initial parameters for the Hang function training.

Expected error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	50

The two-argument Hang function does a fairly complex nonlinear argument mapping. To properly handle this case, networks must be extended to 15 neurons in the hidden layer. The training results are presented in the table 7.

Table 7. Training results for the Hang function.

Network		MLP 2-15-1	FCMLP 2-15-1
LM	[ms]	27.235	34.237
LMP4	[ms]	13.191	16.691
AF	[%]	51.6	51.2
LMP8	[ms]	12.553	16.165
AF	[%]	53.9	51.2
LMP16	[ms]	12.462	16.111
AF	[%]	54.2	52.9

The two-argument Hang test turns out to be much more demanding than the unary functions. Nevertheless, the acceleration factors obtained are slightly higher.

4.2.4 The two-argument Sinc function

The two-argument Sinc function is two sine functions composition. The Sinc function takes the following form

$$y = f(x_1, x_2) = \\ = \begin{cases} 1 & x_1 = x_2 = 0 \\ \frac{\sin x_2}{x_2} & x_1 = 0 \wedge x_2 \neq 0 \\ \frac{\sin x_1}{x_1} & x_2 = 0 \wedge x_1 \neq 0 \\ \frac{\sin x_1}{x_1}\frac{\sin x_2}{x_2} & \text{in other cases.} \end{cases} \tag{22}$$

The Sinc training set has 121 samples for the arguments in the range of $x_1, x_2 \in [-10, 10]$. The Sinc test initial parameters are listed in the Table 8.

Table 8. Initial parameters for the Sinc function training.

Expected error	0.005
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	121

Table 9. Training results for the Sinc function.

Network		MLP 2-15-1	FCMLP 2-15-1
LM	[ms]	53.627	70.685
LMP4	[ms]	25.753	34.208
AF	[%]	52.0	51.6
LMP8	[ms]	24.872	33.424
AF	[%]	53.6	52.7
LMP16	[ms]	24.776	33.383
AF	[%]	53.8	52.8

Like the Hang function, the Sinc function also performs a fairly complex mapping of its arguments. The same networks are used in this test as for the Hang function, but the training set is more than twice as large. The training results are presented in table 9.

In the two-argument Sinc test, the obtained acceleration factors are in the range of 51.6 – 53.8%.

4.3 Classification

The purpose of the classification tests is to find the h classifier that will assign the $\mathbf{y} \in \mathbf{Y}$ class to the $\mathbf{x} \in \mathbf{X}$ input for a given dataset $\{(\mathbf{x}_1, y), \ldots, (\mathbf{x}_n, y)\}$. Formally, such a relationship is presented as $h\colon \mathbf{X} \to \mathbf{Y}$. A neural network can be trained to classify data based on its similarity and common patterns, the so-called features with the help of an appropriate training set. Some examples of classification are presented in the following Sections.

4.3.1 The IRIS classification

In this test, the training set contains 150 samples describing three varieties of iris flowers. The iris flowers are identified with four attributes describing the lengths and widths of the flower petals. The IRIS test initial parameters are shown in the Table 10.

Table 10. Initial parameters for the IRIS test training.

Expected error	0.05
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	150

In this case, neural networks with four inputs, three outputs, and two hidden layers, six neurons each, were used. The training results are presented in Table 11.

Table 11. Training results for the IRIS test.

Network		MLP 4-6-6-3	FCMLP 4-6-6-3
LM	[ms]	528.183	1851.720
LMP4	[ms]	242.789	870.468
AF	[%]	54.0	53.0
LMP8	[ms]	229.337	842.894
AF	[%]	56.6	54.5
LMP16	[ms]	223.374	831.464
AF	[%]	57.7	55.1

In this case, acceleration factors of 57.7% were obtained.

4.3.2 The Two Spirals classification

Two spirals is a well-known classification problem in which a neural network has to choose which of the two spirals a given point belongs to based on its two-dimensional coordinates. The training set contains 96 samples. The two spiral test initial parameters are shown in the Table 12.

Table 12. Initial parameters for the two spirals problem training.

Expected error	0.05
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	96

For two spiral problem, neural networks with two inputs, one output, and three hidden layers, five neurons each, were used. Table 13 shows the simulation results.

Table 13. Training results for the two spirals problem.

Network		MLP 2-5-5-5-1	FCMLP 2-5-5-5-1
LM	[ms]	166.819	349.704
LMP4	[ms]	77.954	165.037
AF	[%]	53.3	52.8
LMP8	[ms]	76.139	161.613
AF	[%]	54.4	53.8
LMP16	[ms]	75.555	161.192
AF	[%]	54.7	53.9

In the two spirals problem, the obtained acceleration factors are in the range of 52.8 – 54.7%.

4.3.3 The Heart disease classification

The heart disease database contains 75 input attributes, but only a subset of 13 is used in all published experiments. The goal of training is to find out if you have heart disease. The network output is an integer with the value 0 (no disease present) or 1,2,3,4 (disease presence). The training set contains 303 samples. The heart disease test initial parameters are presented in the Table 14.

Table 14. Initial parameters for the heart disease test.

Expected error	0.01
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	303

For the heart disease test, neural networks with thirteen inputs, one output, and two hidden layers, nine neurons each, were used. Table 15 shows the simulation results.

Table 15. Training results for the heart disease test.

Network		MLP 13-9-9-1	FCMLP 13-9-9-1
LM	[ms]	2661.37	13190.50
LMP4	[ms]	1244.88	6558.12
AF	[%]	53.2	50.3
LMP8	[ms]	1178.10	6343.68
AF	[%]	55.7	51.9
LMP16	[ms]	1160.57	6327.90
AF	[%]	56.4	52.0

In the heart disease test, the obtained acceleration factors are in the range of 50.3 – 56.4%.

4.4 Other Trained Problems

Three other additional tests from various fields will be presented here: determining the age of the abalone sea snail based on its physical characteristics, determining the strength of concrete from its physical parameters, and determining the crane power control.

4.4.1 The Abalone age

In this experiment, neural networks are trained to determine the age of a sea snail called *abalone* based on its eight physical properties. All samples have been normalized to the range $[-1, 1]$. The Abalone test contains 4177 samples, each with 8 inputs and one output. The Abalone test initial parameters are shown in the Table 16.

Table 16. Initial parameters for the abalone age training.

Expected error	0.012
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	4177

Table 17. Training results for the abalone age test.

Network		MLP 8-6-6-1	FCMLP 8-6-6-1
LM	[ms]	6187.58	8681.53
LMP4	[ms]	2895.08	4087.72
AF	[%]	53.2	52.9
LMP8	[ms]	2752.73	3881.05
AF	[%]	55.5	55.3
LMP16	[ms]	2725.98	3867.87
AF	[%]	55.9	55.4

For the Abalone age test, neural networks with eight inputs, one output, and two hidden layers, six neurons each, were used. Table 17 presents the simulation results.

In the abalone age test, the obtained acceleration factors are in the range of 52.9 – 55.9%.

4.4.2 The Concrete test

In this experiment, based on its eight physical properties, neural networks are trained to determine the concrete compressive strength based on its age and ingredients. All samples have been normalized to the range $[-1, 1]$. The concrete test contains 1030 samples, each with 8 inputs and one output. The Concrete test initial parameters are shown in the Table 18.

Table 18. Initial parameters for the concrete training.

Expected error	0.01
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	1030

Table 19. Training results for the concrete test.

Network		MLP 8-6-6-1	FCMLP 8-6-6-1
LM	[ms]	5017.19	1842.99
LMP4	[ms]	2453.76	869.342
AF	[%]	51.1	52.8
LMP8	[ms]	2421.89	828.089
AF	[%]	51.7	55.1
LMP16	[ms]	2418.00	825.561
AF	[%]	51.8	55.2

For the concrete test, neural networks with eight inputs, one output, and two hidden layers, six neurons each, were used. Table 19 presents the simulation results.

In the concrete test, the acceleration factors are in the range of 51.1 – 55.2%.

4.4.3 The Container Crane Controller test

The container crane controller data set has two input attributes (speed and angle) and one output attribute (power). It contains 15 samples. All samples have been normalized to the range $[-1, 1]$. The container crane controller test initial parameters are listed in the Table 20.

Table 20. Initial parameters for the container crane controller training.

Expected error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training set size	15

Table 21. Training results for the container crane controller test.

Network		MLP 8-6-6-1	FCMLP 8-6-6-1
LM	[ms]	6.785	6.557
LMP4	[ms]	3.368	3.284
AF	[%]	50.3	49.9
LMP8	[ms]	3.342	3.258
AF	[%]	50.7	50.3
LMP16	[ms]	3.336	3.257
AF	[%]	50.8	50.3

For this test, neural networks with two inputs, one output, and one hidden layer with ten neurons, were used. Table 21 presents the simulation results.

In the performed test, the acceleration factors were obtained in the range of 49.9 - 50.8

5 Conclusion

The vector approach to computation using the Levenberg-Marquardt algorithm was developed to increase its efficiency. The proposed optimization allows for parallelization of calculations for repeating steps in the classic LM algorithm. It is possible thanks to the use of vector calculations, which can be implemented in modern processors by SIMD (Single Instruction Multiple Data) instructions. The operations in several successive steps are completely independent of each other, so they can be performed in parallel. The growing possibilities of multiprocessor devices in the field of vector instructions are becoming a natural stimulus for the evolution of training algorithms towards their parallelization, as originally proposed in [34, 35, 36, 37, 38, 39, 40].

The presented experiment contains a total of 10 different test problems including 4 approximations of functions, 3 classification cases, and 3 other examples. The analyzed tests had data sets of various sizes. The size of the network and the number of inputs and outputs also differed. The overall success rate and the number of epochs needed to train the network did not depend on the calculation method, but only on the selected problem and network topology.

The training time for the vector approach is much shorter, and the acceleration factor is in the range of 47.1-57.7%.

In our research, three sizes of vectors were used: 4, 8, and 16. These vectors were implemented using the AVX (Advanced Vector eXtension) and AVX-512 instructions. As might be expected, the longer the vectors were, the greater the acceleration factor. However, for the tested problems, the differences in the acceleration coefficients were relatively small 0.4-3.7% only. Thus, it seems sufficient to use four-element vectors, the more so that as the number of elements in the vector increases, the memory occupancy also increases, but with the current memory size, this is usually not a problem. It is also worth noting that the proposed solution can be implemented without vector instructions, instead using multi-core processors. However, in this case, the thread synchronisation become necessary. The top highlights discussed in this article for the vector approach to computing the LM algorithm can be summarized as follows:

1. The implementation difficulty for the vector approach is similar to the classical implementation of the Levenberg-Marquardt algorithm.
2. The obtained results show that the proposed solution causes on average more than two times shorter training time compared to the classic LM algorithm.
3. Obtaining such a significant reduction in the training time of the LM algorithm results from the parallel execution of the next steps of the algorithm before they are required.
4. The success rate and the number of epochs needed to train the network are identical to the classical calculation method.
5. A vector approach to the LM algorithm can be replaced with a parallel implementation on multiple processor cores, but then requires the use of synchronization techniques.

In our future work, we plan to apply our vector approach to other neural network training algorithms, e.g. [9, 10, 11].

Disclaimer

This work has been supported by the Polish National Science Center under Grant 2017/27/B/ST6/02852 and the program of the Polish Minister of Science and Higher Education under the name ”Regional Initiative of Excellence” in the years 2019 - 2023 project number 020/RID/2018/19, the amount of financing PLN 12,000,000.00.

References

[1] Marcin Gabryel, Dawid Lada, Zbigniew Filutowicz, Zofia Patora-Wysocka, Marek Kisiel-Dorohinicki, and Guang Yi Chen. Detecting anomalies in advertising web traffic with the use of the variational autoencoder. Journal of Artificial Intelligence and Soft Computing Research, 12(4):255–256, 2022.

[2] Marcin Zalasiński, Łukasz Laskowski, Tacjana Niksa-Rynkiewicz, Krzysztof Cpałka, Aleksander Byrski, Krzysztof Przybyszewski, Paweł Trippner, and Shi Dong. Evolutionary algorithm for selecting dynamic signatures partitioning approach. Journal of Artificial Intelligence and Soft Computing Research, 12(4):267–279, 2022.

[3] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In 2017 International Conference on Engineering and Technology (ICET), pages 1–6, 2017.

[4] A. M. Taqi, A. Awad, F. Al-Azzo, and M. Milanova. The impact of multi-optimizers and data augmentation on tensorflow convolutional neural network performance. In 2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), pages 140–145, April 2018.

[5] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. Pattern Recognition, 77:354 – 377, 2018.

[6] Robert K. Nowicki and Janusz T. Starczewski. A new method for classification of imprecise data using fuzzy rough fuzzification. Inf. Sci., 414:33–52, 2017.

[7] Janusz T. Starczewski, Katarzyna Nieszporek, Michal Wróbel, and Konrad Grzanek. A fuzzy SOM for understanding incomplete 3d faces. In ICAISC (2), volume 10842 of Lecture Notes in Computer Science, pages 73–80. Springer, 2018.

[8] Michal Wróbel, Katarzyna Nieszporek, Janusz T. Starczewski, and Andrzej Cader. A fuzzy measure for recognition of handwritten letter strokes. In ICAISC (1), volume 10841 of Lecture Notes in Computer Science, pages 761–770. Springer, 2018.

[9] Jarosław Bilski, Bartosz Kowalczyk, Alina Marchlewska, and Jacek M. Zurada. Local levenberg-marquardt algorithm for learning feedforwad neural networks. Journal of Artificial Intelligence and Soft Computing Research, 10(4):299–316, 2020.

[10] Jarosław Bilski, Bartosz Kowalczyk, Andrzej Marjański, Michał Gandor, and Jacek Zurada. A novel fast feedforward neural networks training algorithm. Journal of Artificial Intelligence and Soft Computing Research, 11(4):287–306, 2021.

[11] Jarosław Bilski, Bartosz Kowalczyk, Marek Kisiel-Dorohinicki, Agnieszka Siwocha, and Jacek Żurada. Towards a very fast feedforward multilayer neural networks training algorithm. Journal of Artificial Intelligence and Soft Computing Research, 12(3):181–195, 2022.

[12] Xin Wang, Yi Guo, Yuanyuan Wang, and Jinhua Yu. Automatic breast tumor detection in abvs images based on convolutional neural network and superpixel patterns. Neural Computing and Applications, 31(4):1069–1081, 2019.

[13] Muhammad Irfan Sharif, Jian Ping Li, Muhammad Attique Khan, and Muhammad Asim Saleem. Active deep neural network features selection for segmentation and recognition of brain tumors using mri images. Pattern Recognition Letters, 129:181 – 189, 2020.

[14] P. Mohamed Shakeel, T. E. E. Tobely, H. Al-Feel, G. Manogaran, and S. Baskar. Neural network based brain tumor detection using wireless infrared imaging sensor. IEEE Access, 7:5577–5588, 2019.

[15] Alexander Rakhlin, Alexey Shvets, Vladimir Iglovikov, and Alexandr A. Kalinin. Deep convolutional neural networks for breast cancer histology image analysis. In Aurélio Campilho, Fakhri Karray, and Bart ter Haar Romeny, editors, Image Analysis and Recognition, pages 737–744, Cham, 2018. Springer International Publishing.

[16] Xin Cai, Yufeng Qian, Qingshan Bai, and Wei Liu. Exploration on the financing risks of enterprise supply chain using back propagation neural network. Journal of Computational and Applied Mathematics, 367:112457, 2020.

[17] Amin Hedayati Moghaddam, Moein Hedayati Moghaddam, and Morteza Esfandyari. Stock market index prediction using artificial neural network. Journal of Economics, Finance and Administrative Science, 21(41):89 – 93, 2016.

[18] Songqiao Qi, Kaijun Jin, Baisong Li, and Yufeng Qian. The exploration of internet finance by using neural network. Journal of Computational and Applied Mathematics, 369:112630, 2020.

[19] A. V. Kurbesov, D. V. Ryabkin, I. I. Miroshnichenko, N. A. Aruchidi, and K. Kh. Kalugyan. Automated voice recognition of emotions through the use of neural networks. In Rafik A. Aliev, Janusz Kacprzyk, Witold Pedrycz, Mo Jamshidi, Mustafa B. Babanli, and Fahreddin M. Sadikoglu, editors, 10th International Conference on Theory and Application of Soft Computing, Computing with Words and Perceptions - ICSCCW-2019, pages 675–682, Cham, 2020. Springer International Publishing.

[20] X. Changzhen, W. Cong, M. Weixin, and S. Yanmei. A traffic sign detection algorithm based on deep convolutional neural network. In 2016 IEEE International Conference on Signal and Image Processing (ICSIP), pages 676–679, Aug 2016.

[21] Katsuba Yurii and Grigorieva Liudmila. Application of artificial neural networks in vehicles' design self-diagnostic systems for safety reasons. Transportation Research Procedia, 20:283 – 287, 2017. 12th International Conference "Organization and Traffic Safety Management in large cities SPbOTSIC-2016, 28-30 September 2016, St. Petersburg, Russia.

[22] N. P. Patel and A. Kale. Optimize approach to voice recognition using iot. In 2018 International Conference On Advances in Communication and Computing Technology (ICACCT), pages 251–256, 2018.

[23] Yi Mou and Kun Xu. The media inequality: Comparing the initial human-human and human-ai social interactions. Computers in Human Behavior, 72:432 – 440, 2017.

[24] Werbos J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University, 1974.

[25] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, 1988.

[26] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In IEEE International Conference on Neural Networks, pages 586–591 vol.1, March 1993.

[27] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[28] M. T. Hagan and M.B. Menhaj. Training feedforward networks with the marquardt algorithm. IEEE Transactions on Neuralnetworks, 5:989–993, 1994.

[29] N. Ampazis and S. J. Perantonis. Two highly efficient second-order algorithms for training feedforward networks. IEEE Transactions on Neural Networks, 13(5):1064–1074, 2002.

[30] J. S. Smith, B. Wu, and B. M. Wilamowski. Neural network training with Levenberg–Marquardt and adaptable weight compression. IEEE Transactions on Neural Networks and Learning Systems, 30(2):580–587, 2019.

[31] Miao Cui, Kai Yang, Xiao liang Xu, Sheng dong Wang, and Xiao wei Gao. A modified Levenberg–Marquardt algorithm for simultaneous estimation of multi-parameters of boundary heat flux by solving transient nonlinear inverse heat conduction problems. International Journal of Heat and Mass Transfer, 97:908 – 916, 2016.

[32] Jiyang Dong, Ke Lu, Jian Xue, Shuangfeng Dai, Rui Zhai, and Weiguo Pan. Accelerated nonrigid image registration using improved Levenberg–Marquardt method. Information Sciences, 423:66 – 79, 2018.

[33] Jarosław Bilski, Bartosz Kowalczyk, and Jacek M. Żurada. Application of the givens rotations in the neural network learning algorithm. In Artificial Intelligence and Soft Computing, volume 9602 of Lecture Notes in Artificial Intelligence, pages 46–56. Springer-Verlag Berlin Heidelberg, 2016.

[34] Jacek Smoląg, Jarosław Bilski, and Leszek Rutkowski. Systolic array for neural networks. In IV KSNiIZ, pages 487–497, 1999.

[35] Jacek Smoląg and Jarosław Bilski. A systolic array for fast learning of neural networks. In V NNSC, pages 754–758, 2000.

[36] D. Rutkowska, R.K. Nowicki, and Y. Hayashi. Parallel processing by implication-based neuro–fuzzy systems. Lecture Notes in Computer Science, 2328:599–607, 2002.

[37] Jarosław Bilski and Jacek Smoląg. Parallel realisation of the recurrent RTRN neural network learning. In Artificial Intelligence and Soft Computing, volume 5097 of Lecture Notes in Computer Science, pages 11–16. Springer-Verlag Berlin Heidelberg, 2008.

[38] Jarosław Bilski and Jacek Smoląg. Parallel architectures for learning the RTRN and Elman dynamic neural network. IEEE Transactions on Parallel and Distributed Systems, 26(9):2561 – 2570, 2015.

[39] Jarosław Bilski, Jacek Smoląg, and Jacek M. Żurada. Parallel approach to the Levenberg-Marquardt learning algorithm for feedforward neural networks. In Artificial Intelligence and Soft Computing, volume 9119 of Lecture Notes in Computer Science, pages 3–14. Springer-Verlag Berlin Heidelberg, 2015.

[40] J. Bilski and B.M. Wilamowski. Parallel Levenberg-Marquardt algorithm without error backpropagation. Artificial Intelligence and Soft Computing, Springer-Verlag Berlin Heidelberg, LNAI 10245:25–39, 2017.

Jarosław Bilski – received the M.Sc. degree in electrical engineering from Częstochowa University of Technology in 1988 and Ph.D. degree (with honors) in computer science from AGH Academy of Science and Technology, Cracow, Poland in 1995. Now, he is an Associate Professor in the Department of Computational Intelligence at Częstochowa University of Technology, Częstochowa, Poland. His research interests include neural networks, learning algorithms, artificial intelligence and algorithm parallelization. He has published over 70 technical papers in journals and conference proceedings. Dr. Bilski is a member and founder of the Polish Neural Network Society. He has co-organized several Conferences on Artificial Intelligence and Soft Computing.
https://orcid.org/0000-0003-1769-3934

Jacek Smoląg received the M.Sc. degree in electrical engineering from Częstochowa University of Technology in 1991 and Ph.D. degree in electronics from Lodz University of Technology in 1998, Lodz, Poland. He is an Assistant Professor in the Institute of Computational Intelligence at Częstochowa University of Technology, Częstochowa, Poland. He is engaged in research in parallel processing focusing on mapping algorithms into parallel computers, parallel systolic processing and parallel architectures for artificial intelligence. His current research interests include design and implementation of neural network learning algorithms in hardware. He is a member of the Polish Neural Network Society.
https://orcid.org/0000-0002-1326-3374

Bartosz Kowalczyk - received the M.Sc. degree in computer science from Częstochowa University of Technology in 2015 and Ph.D. degree in computer science from Częstochowa University of Technology, Częstochowa, Poland. His research interests include linear algebra especially orthogonal transforms, learning algorithms and neural networks. He has published a few technical papers in journals and conference proceedings. M.Sc. Kowalczyk attended several Conferences on Artificial Intelligence and Soft Computing.
https://orcid.org/0000-0002-7683-9051

Konrad Grzanek, scientist, programmer and lecturer. Graduate of the Technical University of Łódź (FTIMS). Assistant professor at the Social Academy of Sciences. He holds a Ph.D. from Częstochowa University of Technology (CUT). His research interests focus on programming languages, software quality, software development processes, and artificial intelligence, in particular on combining machine learning methods with static software analysis. As a programmer, he is an advocate and promoter of the functional programming style. Author of over 30 publications related to various problems of computer science and software engineering.
https://orcid.org/0000-0003-2193-143X

Ivan Izonin is an Associate Professor at the Department of Artificial Intelligence of Lviv Polytechnic National University, Ukraine. He received his MSc degree in Computer science in 2011 and his MSc degree in Economic cybernetics in 2012. He received a Ph.D. in Artificial Intelligence in 2016. His main research interests are focused on small data approach, high-speed neural-like systems, non-iterative machine learning algorithms, and ensemble learning.
https://orcid.org/0000-0002-9761-0096