sciendo

JAISCR, 2020, Vol. 10, No. 4, pp. 299 – 316
10.2478/jaiscr-2020-0020

LOCAL LEVENBERG-MARQUARDT ALGORITHM FOR LEARNING FEEDFORWAD NEURAL NETWORKS

Jarosław Bilski[1,*], Bartosz Kowalczyk[1], Alina Marchlewska[2], Jacek M. Zurada[3]

[1]*Department of Computer Engineering, Czestochowa University of Technology, al. Armii Krajowej 36, 42-200 Częstochowa, Poland*

[2]*University of Social Science, Łódź, Poland and Clark University Worcester, MA, USA*

[3]*Department Electrical and Computer Engineering, University of Louisville, Louisville, KY 40292, USA*

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

Submitted: 21st October 2019; Accepted: 19th May 2020

Abstract

This paper presents a local modification of the Levenberg-Marquardt algorithm (LM). First, the mathematical basics of the classic LM method are shown. The classic LM algorithm is very efficient for learning small neural networks. For bigger neural networks, whose computational complexity grows significantly, it makes this method practically inefficient. In order to overcome this limitation, local modification of the LM is introduced in this paper. The main goal of this paper is to develop a more complexity efficient modification of the LM method by using a local computation. The introduced modification has been tested on the following benchmarks: the function approximation and classification problems. The obtained results have been compared to the classic LM method performance. The paper shows that the local modification of the LM method significantly improves the algorithm's performance for bigger networks. Several possible proposals for future works are suggested.

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

1 Introduction

Nowadays, artificial intelligence exists not only in the world of science but also in industry. One of the most interesting areas of AI are neural networks. Each year researchers across the world produce an incredible number of scientific papers whose main theme originates from AI, especially from neural networks as in [1, 2, 3, 4, 5, 6, 7, 8, 9]. Industry strongly benefits from that by applying more and more advanced AI solutions in their products. The biggest industrial beneficiaries of the neural networks are medicine and health care [10, 11, 12, 13, 14, 15], banking and finances [16, 17, 18], but also safety [19, 20, 21, 22, 23] and entertainment [24, 25, 26].

All applications of neural networks share the same common feature – a network needs to be trained in order to solve a specific problem. There are many training algorithms which are derived directly from the original backpropagation method [27] such as [28, 29, 30]. There are also more complex algorithms which involve Newton's

method, such as the Levenberg-Marquardt (LM) algorithm, which was initially proposed in [31].

The LM algorithm is a supervised training method that can be applied to any feedforward neural network, which from now on will be also referred as "FF". A neural network is built from layers. Each layer is built from a finite number of neurons. The last layer of the network has a special function acting as a network's output, hence it is called the output layer. In most practical applications there are networks with more than a single layer. All layers preceding the network's output are called hidden layers. Feedforward neural networks can have various topologies. The most common is the multilayer perceptron also called an MLP. In such networks each layer is connected only to the previous one. An exemplary MLP network is shown in Figure 1. The next common FF topology is the fully connected multilayer perceptron, simply referred to as an FCMLP. This type of network is similar to the classic MLP with additional layer connections. As shown in Figure 2 each layer of the FCMLP network maintains connections to all preceding layers. Due to that, an FCMLP network with the same neuron count will contain many more weights than a standard MLP of the same size. Additional interesting variations are fully connected cascade networks (FCCs). A network of such type is similar to the FCMLP network whose layers contain only a single neuron. Each layer is connected to all preceding layers and network inputs as shown in Figure 3. Originally, the FCC network was used by P. Werbos in his research on the backpropagation method [27]. It is worth noting that the special case of the FCC network is the FCMLP network, while the MLP network is a special case of the FCMLP network.

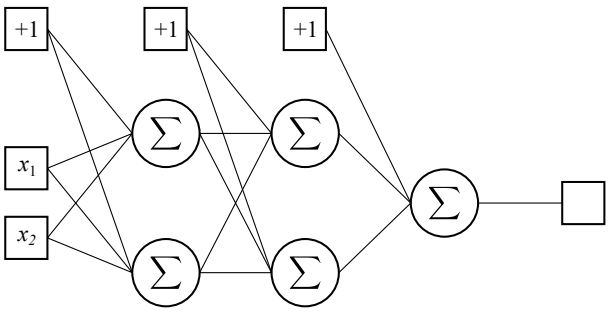

Figure 1. MLP network with 5 neurons and two hidden layers.

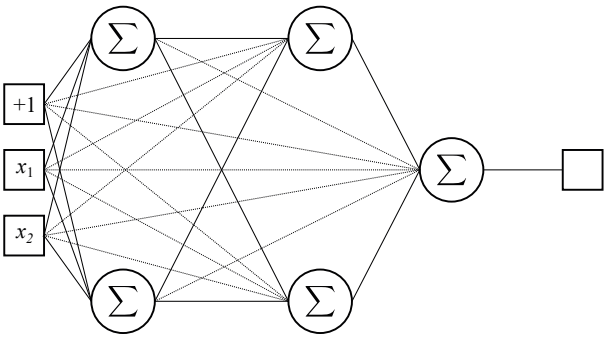

Figure 2. FCMLP network with 5 neurons and two hidden layers (excessive connections are marked with the dotted line).

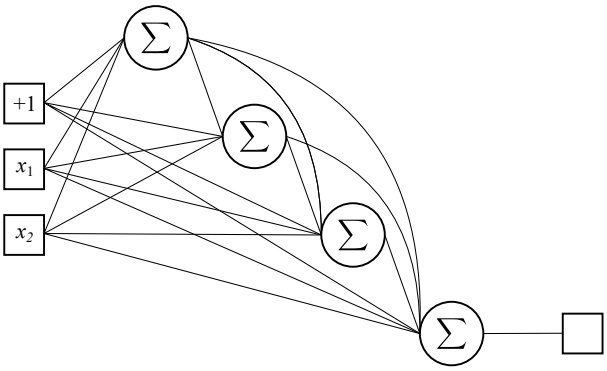

Figure 3. FCC network with 4 neurons.

While the LM algorithm is a very popular and robust method of finding the function minimum in most applications, it is still burdened with several disadvantages. Some of them are serious enough to the extent that makes the LM algorithm completely impractical. Through the years many researchers have been making attempts at devising the LM algorithm optimization techniques.

The LM algorithm is a second-order method which combines the advantages of the Gauss-Newton and the gradient descend methods. As most of the neural networks training algorithms, the classic LM can also become stuck in the local minimum. In classical first-order methods this problem can be solved by applying the momentum factor. Such modification helps to overshoot the local minima and find the right direction towards the optimal solution. The momentum factor can be selected arbitrarily and remain fixed through the training or be dynamically adjusted based on the convergence process. Such approach has been presented in [32], where the main idea was to combine the advantages of the LM and CG methods in order to

increase the robustness of the training. The authors made an effort to develop two variants of the momentum Levenberg-Marquardt algorithm with both, fixed and adaptable momentum size. While the presented algorithms were proven to be more efficient in the scope of the training time than the classic LM, they both were still burdened with the biggest of the LM disadvantages – a great computational complexity due to the size of a single Jacobian matrix.

While approaching complex experiments, the classical methods of neural networks training can report a very poor convergence rate in the flat spot of the error function. Typically, the flat spot problem causes a significant training slowdown due to low gradient values of the hidden neurons. In the first-order methods such problem can be addressed by fine tuning of the training parameters. This is not the case in the more complex second-order methods. The LM algorithm's base complexity is very high, so application of additional training parameters should be avoided. An interesting approach to this problem has been presented in [33]. In order to overcome a stalemate in the convergence process caused by the flat-spot problem, the authors proposed the weights compression modification to the Levenberg-Marquardt algorithm. This technique is used to push the neurons' gradients out of the linear area of the activation function in order to boost the training. The authors have reported a significant improvement in terms of the success ratio achieved by their idea over the classic LM variant. Moreover, the authors proved that their idea does not increase computational complexity significantly enough for various topologies of the feed forward networks.

The authors of [34] focused their effort on a slightly different aspect of the LM algorithm optimizations. In most implementations sensitivity coefficients of the Jacobian matrix are calculated by using the numerical differentiation methods. This results in obtaining approximated values of the gradient derivatives. The authors noticed that when tackling complex problems in which transient states and severe non-linearities occur, it is the classic approach to the Jacobian computation that might bring significant instabilities to the training process. Being the core of the LM algorithm, the Jacobian matrix needs to be computed as precisely as possible. In order to achieve that, the authors proposed a complex variable differentiation method to be utilized during the LM training. The authors have proved this technique is able to increase the stability of the LM training process, but it is not able to decrease the computational complexity.

Some researchers also noticed that the Jacobian matrix size might be the root cause of the computational complexity problem in the classic Levenberg-Marquardt algorithm. In [35] the authors propose a dedicated optimization for the LM algorithm while registering the nonrigid images. They made an attempt to reuse a once constructed Jacobian matrix in two iterations of the algorithm instead of one. The main idea behind it was to perform an additional step after the classic LM calculations in order to establish the optimal correction vector. Additionally, in order to increase the performance a linear search was used. The authors reported their method to be more efficient than the classic LM variant due to calculating the Jacobian matrix only once per two algorithm iterations.

As shown in the discussion above, many approaches have been made in order to optimize the classic Levenberg-Marquardt algorithm, but none of them touches directly the core of the problem, i.e., the size of the Jacobian matrix. This structure is consecutively constructed throughout the whole epoch. As shown in Section 3, it can grow significantly for bigger networks especially during the training that utilizes very long training sets. In such cases a practical implementation of the classic LM algorithm makes it of no use due to a great computational complexity and memory usage.

In this paper the novel method for optimizing the LM algorithm is presented. It is achieved by splitting the single Jacobian matrix and transferring it locally to all neurons respectively. Such approach significantly reduces computational complexity of the classic variant and opens the possibility for parallel computations. During the research the following original and novel contributions have been made:

1. The mathematical derivation of the local modification to the classic Levenberg-Marquardt algorithm has been presented.
2. The consecutive steps of the local LM algorithm have been precisely described.
3. The QR decomposition along with the Givens

rotations have been used to solve equations with an inverse matrix in both LM algorithms.

4. The performance of the local LM variant has been compared with that of the classic Levenberg-Marquardt algorithm.

5. The original benchmark procedure has been developed in order to achieve the most valuable and reliable results.

6. Both, local and classic variants of the LM have been tested on various topologies of feedforward networks utilizing multiple benchmarks.

7. The proposed modification solves the computational complexity problem caused by a huge single Jacobian matrix of the classic LM variant.

The paper has been divided into several sections. Section 2 is devoted to a detailed description of the classic Levenberg-Marquardt algorithm. Both, the mathematical and the practical implementation approaches are presented. Section 3 contains the core of this paper, i.e. the discussion of the local modification of the LM algorithm. The crucial differences to the classic variant are highlighted. As the foreword in Section 4, the utilized original benchmarking procedure and all important nomenclature are explained. Then, the selected results are presented and described. Section 5 concludes the considerations on the local Levenberg-Marquardt and presents the possible directions for future research.

2 The classic Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm (LM) is a second-order training method for feedforward neural networks. Based on the shape of the error function, the LM algorithm can adjust the speed of training. It is possible by using the advantages of both the steepest descent and the quasi-Newton methods. In the LM training the objective function is given by the following equation

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

where Q is the number of samples and $\varepsilon_r^{(L)}$ is a nonlinear neuron error which is 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 $r-th$ expected vector of the $t-th$ teaching sample. The weight update is calculated as follows

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

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

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

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

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

and $\mathbf{J}(\mathbf{w}(n))$ is the Jacobian matrix

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

The neurons of the hidden layers calculate nonlinear errors $\varepsilon_i^{(lr)}$ according to the following formula

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

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

Based on that all elements of the Jacobian matrix can be calculated for each weight of the network

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

All weights of the network are stored in a single vector, which is typical of the Levenberg-Marquardt algorithm. The $\mathbf{S}(\mathbf{w}(n))$ factor introduced in equation (5) is defined as

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

Conveniently, in the Gauss-Newton method it can be assumed that $\mathbf{S}(\mathbf{w}(n)) \approx 0$, which leads to the simplifying of equation (3) as follows

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

The Levenberg-Marquardt algorithm performs a weights update only once at the end of each epoch. At this moment, all elements of the Jacobian matrix are already calculated by using equations (6), (7), (8) and (9). For optimization of further calculations in the LM algorithm, it is assumed that $\mathbf{S}(\mathbf{w}(n)) = \mu\mathbf{I}$. Based on that, equation (3) is transformed accordingly

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

At this stage, the algorithm attempts to find the proper weight update vector for the network by solving equation (12). This can be achieved by the QR decomposition, but before that, the equation needs to be transformed further. Let

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

$$\mathbf{h}(\mathbf{n}) = \mathbf{J}^{\mathbf{T}}(\mathbf{w}(\mathbf{n}))\varepsilon(\mathbf{w}(\mathbf{n})). \tag{14}$$

Then equation (12) takes the following form

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

which is an entry point to the QR decomposition. This process is an iterative method of transforming any non-singular matrix to the product of the upper triangle $\mathbf{R}$ and the orthogonal $\mathbf{Q}$ matrices. The transformation happens by way of using the following equations

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

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

Since $\mathbf{R}$ is an upper triangle matrix, solving equation (17) is not too complex any more and results in obtaining the weight update vector $\Delta(\mathbf{w}(n))$. In this paper the QR decomposition is accomplished by the Givens rotations as shown in [36]. The presented Levenberg-Marquardt algorithm can be summarized in the following steps:

1. Present the next sample and calculate the network's output.
2. Perform the backpropagation and update the respective rows of the Jacobian using (9).
3. Continue to step 1 until all samples of the teaching sequence have been presented. Then, calculate the error criterion and proceed to 4.
4. Solve equation (12) to obtain the weight update vector $\Delta(\mathbf{w}(n))$.
5. Update all weights of the network with respect to the obtained $\Delta(\mathbf{w}(n))$ vector and calculate the error criterion again.
6. Compare the new error with the previous one. If the new error is smaller, the update is assumed to have been successful. Divide μ by β and proceed to the next epoch in step 1. If the new error is greater, the update is assumed to have failed. Multiply μ by β and calculate a new update in step 4.
7. The training concludes once the target error criterion value is met or the training limit is reached.

3 The local modification

In practical applications the biggest concern of the LM algorithm is the size of the Jacobian matrix given by equation (6). This structure is created consecutively throughout the epoch for all weights of the network. Based on the fact that the Jacobian matrix is the entry point to the LM algorithm, it is not so easy to optimize further computations for the weight update as shown in equations (9 – 17). This Section brings the idea and explanation how to approach the aforementioned obstacles in the local variant of the Levenberg-Marquardt algorithm, from now on also referred to as the LLM.

Let *no* stand for the number of network outputs, *np* for the total number of samples, *nw* for the total number of weights of all neurons, and *nn* for the total number of neurons in the network. The main idea behind the proposed modification is to split a single Jacobian matrix into smaller matrices, unique for each neuron. In this case, instead of having a single Jacobian $\mathbf{J}(\mathbf{w}(n))$ of size $[no \cdot np] \times [nw]$, its

structure is split into *nn* Jacobians $\mathbf{J}_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right)$ of size $[np]\times[N_{l-1}+1]$. The Jacobian matrix of i-th neuron in l-th layer is given by the following equation

$$\mathbf{J}_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right) = \begin{bmatrix} \frac{\partial\varepsilon_i^{(l)}(1)}{\partial w_{i0}^{(l)}} & \cdots & \frac{\partial\varepsilon_i^{(l)}(1)}{\partial w_{ij}^{(l)}} & \cdots & \frac{\partial\varepsilon_i^{(l)}(1)}{\partial w_{iN^{l-1}}^{(l)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial\varepsilon_i^{(l)}(t)}{\partial w_{i0}^{(l)}} & \cdots & \frac{\partial\varepsilon_i^{(l)}(t)}{\partial w_{ij}^{(l)}} & \cdots & \frac{\partial\varepsilon_i^{(l)}(t)}{\partial w_{iN^{l-1}}^{(l)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial\varepsilon_i^{(l)}(Q)}{\partial w_{i0}^{(l)}} & \cdots & \frac{\partial\varepsilon_i^{(l)}(Q)}{\partial w_{ij}^{(l)}} & \cdots & \frac{\partial\varepsilon_i^{(l)}(Q)}{\partial w_{iN^{l-1}}^{(l)}} \end{bmatrix}, \tag{18}$$

where each element is computed as follows

$$\frac{\partial\varepsilon_i^{(l)}(t)}{w_{ij}^{(l)}} = \delta_i^{(l)}(t)\, x_j^{(l)}(t). \tag{19}$$

It is worth noticing that a single neuron has only one output. This automatically eliminates the *no* factor from the equation. Since the entry point of the calculations is moved down from the network level to the respective neurons, many operations become independent of each other. To express that in the formal language, equations (9 – 17) need to be improved in order to respect the layer (l) and neuron (i) indices. The vector of weight deltas becomes local for any given neuron which is given by the following formula

$$\Delta_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right) = -\left[\mathbf{J}_i^{(l)T}\left(\mathbf{w}_i^{(l)}(n)\right)\mathbf{J}_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right)+\mu_i^{(l)}\mathbf{I}\right]^{-1}\cdot \cdot\mathbf{J}_i^{(l)T}\left(\mathbf{w}_i^{(l)}(n)\right)\varepsilon_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right). \tag{20}$$

Next, it is translated into the matrix forms in order to formulate the QR decomposition entry point as follows

$$\mathbf{A}_i^{(l)}(n) = -\left[\mathbf{J}_i^{(l)T}\left(\mathbf{w}_i^{(l)}(n)\right)\mathbf{J}_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right)+\mu_i^{(l)}\mathbf{I}\right] \tag{21}$$

$$\mathbf{h}_i^{(l)}(n) = \mathbf{J}_i^{(l)T}\left(\mathbf{w}_i^{(l)}(n)\right)\varepsilon_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right). \tag{22}$$

At this stage the local weight update vector of a respective neuron is calculated according to the following equation

$$\Delta_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right) = \mathbf{A}_i^{(l)}(n)^{-1}\mathbf{h}_i^{(l)}(n). \tag{23}$$

The QR decomposition is performed locally by each neuron as shown in the following formulas

$$\mathbf{Q}_i^{(l)T}(n)\,\mathbf{A}_i^{(l)}(n)\,\Delta_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right) = \mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n), \tag{24}$$

$$\mathbf{R}_i^{(l)}(n)\,\Delta_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right) = \mathbf{Q}_i^{(l)T}(n)\,\mathbf{h}_i^{(l)}(n). \tag{25}$$

The Levenberg-Marquardt algorithm with the local modification can be summarized by the following procedure:

1. Present the next sample to the network and calculate the network's output.
2. Perform the backpropagation and update the respective rows of all Jacobians using (19).
3. Continue to step 1 until all samples of the teaching sequence have been presented. Then, calculate the error criterion and proceed further to 4. Additionally, calculate the error criterion for each neuron using equation (2).
4. Solve equations (20) to achieve the weight update vectors $\Delta_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right)$ for all neurons.
5. Update the weights of the neurons in the network with respect to the obtained $\Delta_i^{(l)}\left(\mathbf{w}_i^{(l)}(n)\right)$ vectors and calculate the error criterion again. Additionally, calculate the error criterion for each neuron using equation (2).
6. Compare the new neuron's error $\varepsilon_i^{(l)}(\hat{n})$ with the previous one. If the new error is smaller, the update for a respective neuron is assumed to have been successful. Divide $\mu_i^{(l)}$ by β. If the new error is greater, the update for a respective neuron is assumed to have failed. Multiply $\mu_i^{(l)}$ by β.
7. Verify the overall error of the network. If the weights corrections of particular neurons have resulted in a reduced error, proceed to the next epoch in 1. If the error is not reduced, continue to 4.
8. The training concludes once the target error criterion value is met, the training limit is reached or all neurons of the network have approached $\mu_i^{(l)}$ max value.

4 Results

In order to examine the local variant of the Levenberg-Marquard algorithm (LLM) the benchmarking procedure has been established. Its detailed description can be found in subsection 4.1. The main idea which stands behind the presented benchmarks is to compare the LLM with the classic LM implementation. The selected benchmarks cover various areas of typical problems that are solved by the neural networks. In the first place, the approximations of non-linear functions are presented. The prepared teaching sequences simulate a single (logistic curve) and two variable (Hang, Sinc) functions. The second group of tests are the classifications, where a 4-2-4 "tight" encoder problem and a parity detection circuit simulation are presented.

During the benchmarking also a wide range of network topologies have been used. This includes the classic feedforward, fully connected and cascade networks. In order to improve the readability of the presented results, a consistent naming of the networks is used. The multilayered perceptron containing L layers with n_l ($l \in [1,\ldots,L]$) neurons in each are referred to as an "MLP-$[n_l]L$". The same network with excessive connections between all its layers (not only to the previous one) is additionally prefixed with an "FC", which stands for "Fully Connected". The fully connected cascade network with n neurons is referred to as an FCC-n.

4.1 Benchmark methodology

The practical implementations of neural networks and training algorithms contain a great number of parameters which are used to modulate the training process. Some of them are adjusted before the run and then act as runtime constants (training goal, error criterion, epoch limit, etc.). There are also parameters whose values are set before the run and then, they are modified by the training algorithms. In the LM and LLM such variables are μ and μ_i, respectively initiated by β value. Based on the initial set of all those parameters, the training might be successful or might fail. In order to prepare stable and reproducible results, the consistent methodology has been applied for each performed benchmark.

Across all the tests, several common constants have been used. They are presented in Table 1. In order to gather valuable statistical data, each test has been retried 100 times. The result of each test can be either a success or failure. The success is when the network's error criterion reaches the predefined error threshold (set individually for each benchmark). The test is marked as failed if the epoch limit is reached before converging the given criterion. During each run, the samples of the training sets are presented randomly. In all cases, the weights are selected randomly from the range $[-0.5, 0.5]$.

Table 1. Common experiment setup

Epoch limit	1000
Experiment retry count	100
Sequence type	Random
Init weights range	$[-0.5, 0.5]$

Each benchmark also assumed a range of runtime variables (β), which produced a vast amount of data. To support the process of gathering only the most valuable one, the selection criterion ξ given by the equation (26) has been used.

$$\xi_{\text{algorithm}} = \frac{\text{SuccessRatio}}{\text{EpochAverage}}. \tag{26}$$

The tables and charts presented in the following subsections have been gathered with respect to the greatest selection criterion value. Also, a common naming and definitions for the columns in the tables have been used. The β factor stands for the size of the "step" in the LM and LLM algorithms. The success ratio "SR" expressed in percentage $[\%]$ gives the number of successful trials. The average epoch count "Ep." shows the number of epochs that were required on average to reach the predefined error goal. The average time duration "T" expressed in milliseconds $[ms]$ gives the average duration of a single successful trial. The description of each benchmark is summarized in the table, which compares the LLM against the classic LM algorithm. In order to highlight the crucial factor of the LLM optimization, the Jacobian matrix sizes are shown in the column named "Jacobian". This information has a slightly different interpretation in the LM and LLM rows. For the classic LM variant, the column "Jacobian" shows the size of the single Jacobian matrix that is created during computations, while

for the LLM, it shows the biggest Jacobian matrix in the network.

4.2 Approximation

The approximation benchmarks are meant to simulate the relation f between the identities of sets X and Y, which in the formal language is formulated as $f\colon \mathbf{X} \rightarrow \mathbf{Y}$. In classic mathematics such relation is given by a function f, which can be directly implemented as a computer routine. In some complex cases it might be very hard to formulate an explicit relation between identities. Assuming set X contains the inputs of the system and Y contains the known outputs, it is easy to create a teaching sequence for a neural network which is capable of mapping the identities of X to the identities of Y. In the following Sections, the LM and LLM algorithms are used to simulate the response of several non-linear functions.

4.2.1 The single argument logistic function

The logistic function used in this benchmark performs the following mapping

$$f(x) = 4x(1-x) \qquad x \in [0,1]. \tag{27}$$

The training set utilizes 11 samples which map arguments in the range of $x \in [0,1]$. The most important common parameters for the logistic function benchmark are shown in Table 2.

Table 2. Initial setup for the logistic function training

Target error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training sequence size	11

The mapping performed by the logistic function is not very complex. Moreover, the training set is not too big. Due to that, small neural networks (with fewer than 8 neurons) were used. During the training various FCCs and a single FCMLP networks have been used. The results are shown in Table 3.

Table 3. Results of the logistic function training by the LLM algorithm

Network	α	SR	Ep.	T
FCC-2	8	75	10.37	0.32
FCC-3	10	97	9.47	0.28
FCC-4	9	100	8.61	0.4
FCC-5	9.5	99	8.02	0.47
FCC-6	10	98	7.7	0.34
FCC-7 1	0	94	7.53	0.4
FCMLP-5-1	9.5	100	8.14	0.27

The logistic function benchmark shows an overall high success ratio with more than 94% in most cases. It proves that even a two neuron FCC network is able to handle the logistic curve approximation with a 75% success rate at the cost of some additional epochs comparing to the bigger networks. Also, the relation between the number of neurons and the convergence time can be seen. Increasing the number of neurons in FCC networks seems to reduce the average epoch count with a slightly increased training time. In Tables 4 and 5 the comparison results of the LLM and LM trainings for FCC-7 and FCMLP-5-1 networks are shown.

Table 4. Summary of the logistic function using the FCC-7 network

Alg.	β	SR	Ep.	T	Jacobian
LM	2	93	15.25	1.85	11×35
LLM	10	94	7.53	0.4	11×8

For the biggest of the tested cascade networks in the logistic function approximation benchmark, the LM and LLM algorithms achieved a similar success ratio. The LLM turns out to be twice as fast than the classic LM algorithm in the scope of the average epoch convergence.

Table 5. Summary of the logistic function using the FCMLP-5-1 network

Alg.	β	SR	Ep.	T	Jacobian
LM	2	100	14.94	0.88	11×17
LLM	19.5	100	8.14	0.27	11×7

A similar observation regarding the performance of both algorithms can be made in the fully connected multilayered perceptron benchmark. Both algorithms achieved a 100% success ra-

tio where the LLM converged generally faster than the LM. Figures 4 and 5 show an exemplary convergence process that is closest to the average one of the LLM and LM during benchmarks on the FCC-7 and FCMLP-5-1 networks.

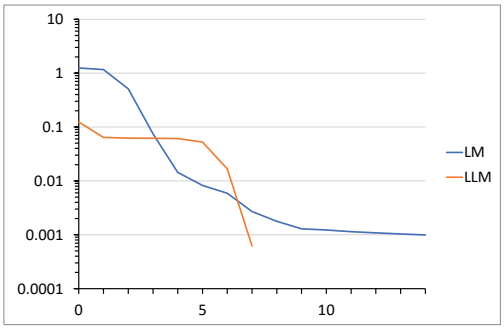

Figure 4. Exemplary training process of the logistic function using the FCC-7 network

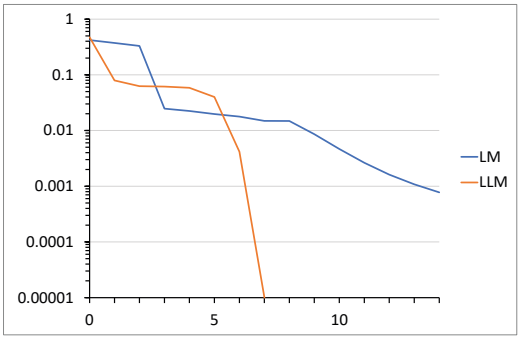

Figure 5. Exemplary training process of the logistic function using the FCMLP-5-1 network

4.2.2 The single argument composite function

In this benchmark the following single argument function is trained

$$f(x) = \sin x \cdot \log x \quad x \in [0.1, 4]. \tag{28}$$

The training set contains 40 samples starting from 0.1 up to 4. The target error for this benchmark is set to 0.001 as an epoch average. The most important parameters of the training setup are presented in Table 6.

Table 6. Initial setup for the composite function training

Target error	0.001
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training sequence size	40

While the complexity of the tested composite function is similar to the logistic function used in the previous benchmark, its training set is several times bigger. The LLM algorithm has been used to train various networks which contain up to 7 neurons. The results and obtained statistics are shown in Table 7.

Table 7. Results of the composite function training by the LLM algorithm

Network	β	SR	Ep.	T
FCC-3	10	85	7.38	0.38
FCC-4	9.5	92	6.89	0.48
FCC-5	9.5	93	6.43	0.5
FCC-6	8.5	88	6.24	0.73
FCC-7	6.5	88	6.55	0.89
MLP-5-1	9.5	100	7.08	0.44

The composite function training concluded with an overall high success ratio for the FCC networks and a perfect 100% SR for the MLP network. In this scenario, an FCC with 3 neurons was able to conclude 85% of the training runs with the success. In terms of the average epoch count required in order to establish the given criterion, the value oscillates around 7. The most successful average training time does not exceed 0.44 ms. The comparison of the LLM and LM performance for FCC-5 and MLP-5-1 networks is shown in Tables 8 and 9, respectively.

Table 8. Summary of the composite function training utilizing the FCC-5 network

Alg.	β	SR	Ep.	T	Jacobian
LM	2	95	13.77	1.75	40×20
LLM	9.5	93	6.43	0.5	40×6

The best success ratio with the value of 93% for the cascade networks was established by the FCC-5 network. In comparison to the classic LM, it is only 2% lower while utilizing the same network and training criteria. In terms of convergence epoch average, the LLM algorithm requires half the number that the classic LM does. Also in the scope of the average training time, the LLM algorithm is 3.5 times faster than its classic counterpart.

Table 9. Summary of the composite function training utilizing the MLP-5-1 network

Alg.	β	SR	Ep.	T	Jacobian
LM	1.5	76	22.12	2.28	40×16
LLM	9.5	100	7.08	0.44	40×6

A significant improvement in terms of all analysed parameters is shown in the training that utilizes the MLP-5-1 network. Every run concluded with the success while the average training time was not longer than 0.44 ms. Figures 6 and 7 show the closest to the average convergence process for the FCC-5 and MLP-5-1 networks.

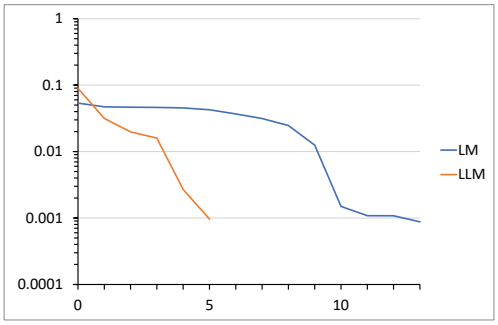

Figure 6. Exemplary convergence process for the composite function training using the FCC-5 network.

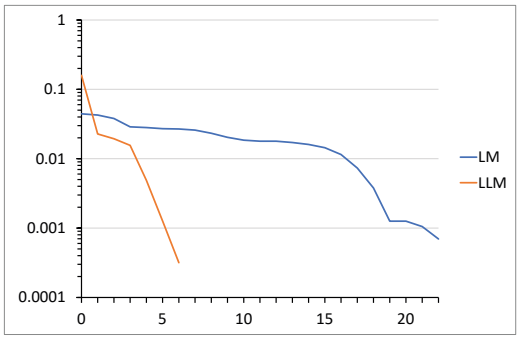

Figure 7. Exemplary convergence process for the composite function training using the MLP-5-1 network

4.2.3 The two-argument Hang function

The Hang routine is a two-argument non-linear function which performs the following mapping

$$f(x_1,x_2) = \left(1 + x_1^{-2} + \sqrt{x_2^{-3}}\right)^2 \qquad x_1,x_2 \in [1,5]. \tag{29}$$

In this benchmark the training set contains 50 samples that cover arguments in the range of $x_1,x_2 \in [1,5]$. The most important common parameters for the Hang benchmark are presented in Table 10.

Table 10. Initial setup for the Hang training

Target error	0.009
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training sequence size	50

The Hang function performs quite complex non-linear mapping of two arguments. In order to handle this case correctly, the networks need to be extended. The smallest network contains 8, while the biggest one 18 neurons. During benchmarking several FCC and FCMLP networks have been used. The training results are shown in Table 11.

Table 11. Results of the Hang training by the LLM algorithm.

Network	β	SR	Ep.	T
FCC-8	9.5	15	8.93	2.08
FCC-10	3.5	41	13.2	4.4
FCC-12	5	47	10.81	5.44
FCC-14	6	49	9.35	6.02
FCC-16	5.5	40	9.12	6.97
FCC-18	9.5	27	7	7.77
MLP-15-1	4	56	11.09	2.9
FCMLP-15-1	3.5	59	11.31	2.45

The Hang benchmark turns out to be much more demanding than the single argument functions. The highest success ratio for the LLM algorithm has been achieved by the MLP networks with SR $>$ 56%. However, in those scenarios the average training time does not exceed 3 ms. The highest SR value for the FCC networks has been achieved for the FCC-14 network with a 49% success rate. In Tables 12 and 13 the comparison results of the LLM and LM trainings for the FCC-14 and FCMLP-15-1 networks are shown.

For the FCC-14 network, the LLM algorithm was able to converge almost 9 times faster than the classic LM at the cost of the success ratio. Also, the LLM requires several fewer epochs on average in order to meet the training requirements.

Table 12. Summary of the Hang function using the FCC-14 network

Alg.	β	SR	Ep.	T	Jacobian
LM	2.5	86	13.71	52.94	50×133
LLM	6	49	9.35	6.02	50×16

Table 13. Summary of the Hang function using the FCMLP-15-1 network

Alg.	β	SR	Ep.	T	Jacobian
LM	3	92	12.99	8.96	50×63
LLM	3.5	59	11.31	2.45	50×18

An average training of the FCMLP network turns out to be significantly faster than for a fully connected cascade. Also in this case, the LLM algorithm manifests a better average convergence time than the classic LM. The exemplary graphs which are the closest to the average convergence processes of the LLM and LM during the benchmarks on the FCC-14 and FCMLP-15-1 networks are shown in Figures 8 and 9.

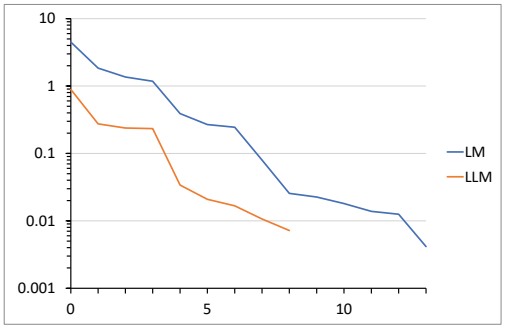

Figure 8. Exemplary training process of the Hang function using the FCC-14 network

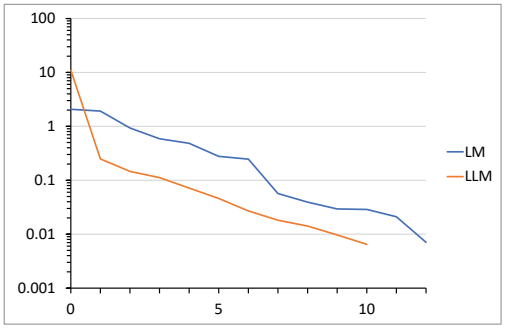

Figure 9. Exemplary training process of the Hang function using the FCMLP-15-1 network

4.2.4 The two-argument Sinc function

The Sinc function is a composition of two sine functions. In the Sinc benchmark this function accepts two arguments and performs the following mapping

$$y = f(x_1, x_2) = \begin{cases} 1 & x_1 = x_2 = 0 \\ \frac{\sin x_2}{x_2} & x_1 = 0 \wedge x_2 \neq 0 \\ \frac{\sin x_1}{x_1} & x_2 = 0 \wedge x_1 \neq 0 \\ \frac{\sin x_1}{x_1} \frac{\sin x_2}{x_2} & \text{in other cases.} \end{cases} \quad (30)$$

The applied Sinc training set contains 121 samples which correspond to the arguments in the range of $x_1, x_2 \in [-10, 10]$. The critical constant parameters for the Sinc benchmark are shown in Table 14.

Table 14. Initial setup for the Sinc training

Target error	0.005
Criterion	Epoch average
Activation in hidden layers	Hyperbolic tangent
Training sequence size	121

Similar to Hang, the Sinc function also performs a rather complex mapping of its arguments. In this benchmark the same set of networks was used as in the Hang scenarios, but the training set is more than twice as long. The benchmark results for the LLM algorithm for various networks are shown in Table 15.

Table 15. The Sinc training result by the LLM algorithm

Network	β	SR	Ep.	T
FCC-14	6	25	5.68	9.68
FCC-16	10	47	5.34	10.52
FCC-18	10	75	5.37	14.21
MLP-25-1	7.5	38	6.5	6.29
FCMLP-25-1	7.5	44	6.66	6.43
MLP-15-15-1	9	59	5.73	22.79
FCMLP-15-15-1	4	73	6.19	32.78

The non-linear two-argument Sinc function proves to be a more reliable benchmark for the LLM algorithm than the Hang approximation. In terms of the success ratio, the best performance has been achieved by the 18-neuron cascade network. A fully connected multilayer perceptron turns out to be im-

mune to the LLM training. In Tables 16 and 17, the comparison results of the LLM and LM trainings for the FCC-18 and FCMLP-15-15-1 networks are shown.

Table 16. Summary of the Sinc function using the FCC-18 network

Alg.	β	SR	Ep.	T	Jacobian
LM	1.5	97	14.39	248.23	121×207
LLM	10	75	5.37	14.21	121×20

The LLM algorithm again outperforms the classic LM in the terms of time performance. Again, the success ratio of the LLM is lower than the LM, but the difference is not as significant as manifested by the Hang benchmark.

Table 17. Summary of the Sinc function using the FCMLP-15-15-1 network

Alg.	β	SR	Ep.	T	Jacobian
LM	2	97	18.35	1704.65	121×348
LLM	4	73	6.19	32.78	121×33

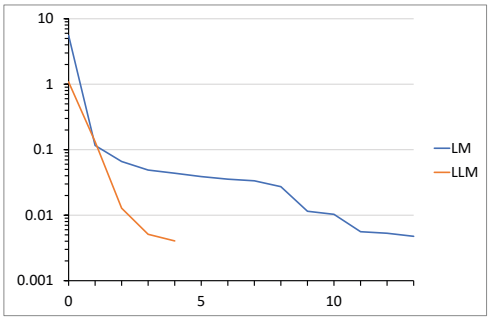

Figure 10. Exemplary training process of the Sinc function using the FCC-18 network

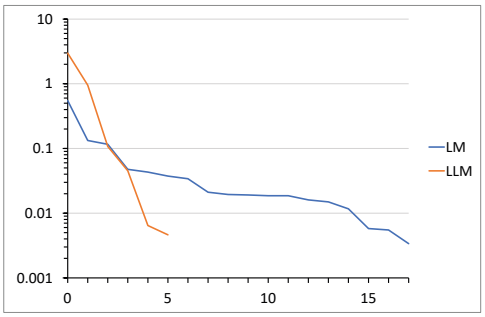

Figure 11. Exemplary training process of the Sinc function using the FCMLP-15-15-1 network

The training of the selected multilayer perceptron network only once resulted in a success. In Figures 10 and 11 the exemplary, closest to the average convergence process of the LLM and LM for the FCC-18 and FCMLP-15-15-1 networks are shown.

4.3 Classification

The goal of the classification benchmark is to find classifier h that is able to apply the class $\mathbf{y} \in \mathbf{Y}$ to the identity $\mathbf{x} \in \mathbf{X}$ for the given set of data $\{(\mathbf{x}_1, y), \ldots, (\mathbf{x}_n, y)\}$. In the formal language such relation can be depicted as $h\colon \mathbf{X} \to \mathbf{Y}$. Using training algorithms and a proper training set, it is possible to train a neural network to categorize the data based on their similarities and common patterns, so called features. In the next subsections the encoder and parity detection benchmarks are presented.

4.3.1 The "tight" encoder

The encoder problems are special cases of the classification benchmarks. In such tasks the network needs to find a way to replicate the inputs in the outputs while having too few neurons in the hidden layer to handle this task easily. Usually encoders are simulated by the MLP-M-N networks with N inputs and a training set with N samples. In each sample only one bit corresponds to the high state while the rest of inputs are of the low state. While standard encoders loosely define their sizes as $M < N$, the "tight" encoders strictly define their structure as $M = \log_2 N$. In order to test the LLM and LM performance, an attempt to simulate the "tight" encoder by the MLP-2-4 network has been made. The training set utilizes 4 samples. The crucial parameters of the benchmark are shown in Table 18.

Table 18. Initial setup for the encoder training

Target error	0.1
Criterion	Epoch average
Activation in hidden layers	Sigmoid
Training sequence size	4

In Table 19 the comparison of the results for the LLM and LM trainings are shown. Both tested algorithms manifest a high success ratio. The classic LM training resulted in 98% SR while the local modification ended with 97%. The average convergence time is slightly better for the LLM algorithm.

In Figure 12, the exemplary graphs which are the closest to the average convergence process for the LLM and LM trainings are shown.

Table 19. Summary of the encoder using the MLP-2-4 network.

Alg.	β	SR	Ep.	T	Jacobian
LM	9.5	98	3.64	0.32	4×22
LLM	9	97	6.07	0.24	4×5

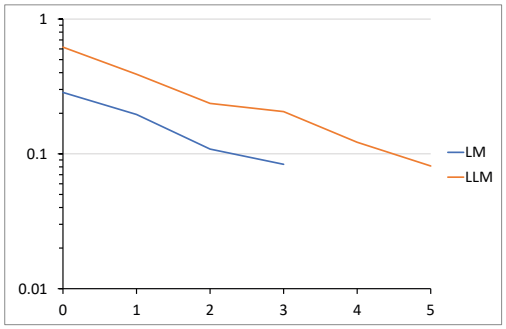

Figure 12. Exemplary training process of the encoder using the MLP-2-4 network

4.3.2 The parity detection

The parity detection is a well known feature of dedicated circuits in the electronics domain. Its primary purpose is to detect any data flow or transmission errors. The basic 2-bit parity detection can be performed by a single XNOR gate. In the parity detection benchmark, this feature is extended to an n-bit problem. To cover all possible scenarios, the training set needs to contain n^2 samples.

In order to test the performance of the LLM against that of the LM, an attempt to train the MLP-10-1 network in order to simulate a 4-bit parity detection circuit has been made. In this case, the training set consists of 16 samples that cover all possible variants of high and low states of the inputs. The network at the output should give a high state if the number of high inputs was even or low state otherwise. The most important parameters of the parity benchmark are shown in Table 20.

The parity detection benchmark results for both algorithms are shown in Table 21. Similar to the encoder problem, the success ratio of both algorithms is high. Again, the local modification of the Levenberg-Marquardt algorithm turns out to be much faster than its classic counterpart. In Figure 13 the exemplary and closest to the average convergence process for the LLM and LM trainings are shown.

Table 20. Initial setup for the parity detection training

Target error	0.1
Criterion	Epoch max
Activation in hidden layers	Sigmoid
Training sequence size	16

Table 21. Summary of the parity detection using the MLP-10-1 network

Alg.	β	SR	Ep.	T	Jacobian
LM	1.5	98	19.35	12.81	16×61
LLM	10	100	3.8	0.62	16×11

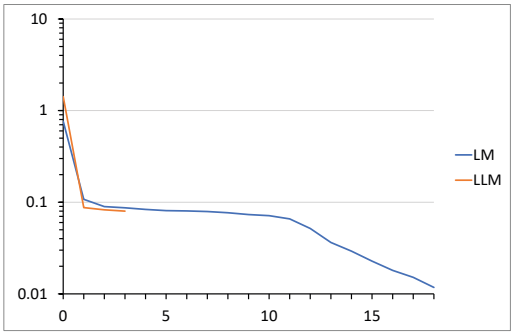

Figure 13. Exemplary training process of the parity detection using the MLP-10-1 network

4.3.3 The points classification

The points classification problem used in this benchmark is designed to classify network inputs into two groups. The training set contains 100 points in the two-dimensional space and the information whether this point belongs to a circle. Based on that information a neural network should find out whether a given point is inside a circle or not. The epoch average error has been used as the training criterion with the accepted error threshold set to 0.02. In Table 22 relevant setup parameters have been shown.

In the points classification benchmark several FCC and FCMLP networks have been used. The resulting statistics of the LLM training are shown

in Table 23. Per presented data the points classification training done by the LLM algorithm proves to be very stable. In most cases the benchmarks have reported a 100% success rate with an average epoch count between 9.8 and 11.2. Regardless of the network topology, the best 100% success rate has been achieved by the networks with at least 12 neurons. In terms of an average epoch count, the fastest convergence has been achieved by the FCC-18 and FCMLP-7-7-1 networks. Also, the relation between the neuron count and the average training time can be seen. The bigger the network is, the longer the training takes while the average epoch count decreases.

Table 22. Initial setup for the points classification training

Target error	0.02
Criterion	Epoch average
Activation in hidden layers	Sigmoid
Training sequence size	100

Table 23. Results of the points classification training by the LLM algorithm

Alg.	β	SR	Ep.	T
FCC-8	9.5	97	10.87	6.1
FCC-10	10	99	10.29	9.6
FCC-12	9.5	100	10.21	13.17
FCC-14	10	100	9.89	14.01
FCC-16	10	100	9.8	15.08
FCC-18	10	100	9.62	15.6
FCMLP-3-3-1	9.5	96	11.2	2.85
FCMLP-5-5-1	10	99	10.3	4.82
FCMLP-7-7-1	10	100	10.04	8.22

The biggest of the tested networks in the LLM training have been compared to the same scenarios trained by the classic LM algorithm. The results are shown in Tables 24 and 25.

Table 24. Summary of the points classification training utilizing the FCC-18 network

Alg.	β	SR	Ep.	T	Jacobian
LM	1.5	91	65.74	751.77	100×207
LLM	10	100	9.62	15.6	100×20

Table 25. Summary of the points classification training utilizing the FCMLP-7-7-1 network

Alg.	β	SR	Ep.	T	Jacobian
LM	1.5	89	66.97	237.07	100×108
LLM	10	100	10.04	8.22	100×17

In both cases the training by the LLM algorithm has shown a better performance across all considered areas. The classic LM has not been able to establish a 100% success ratio while the LLM modification has. The average epoch count has required to establish the convergence on the same level was more than 6 times smaller for the LLM than for the classic LM. This directly translates into a far shorter average training time. The exemplary convergence processes for the FCC-18 and FCMLP-7-7-1 networks are shown in Figures 14 and 15.

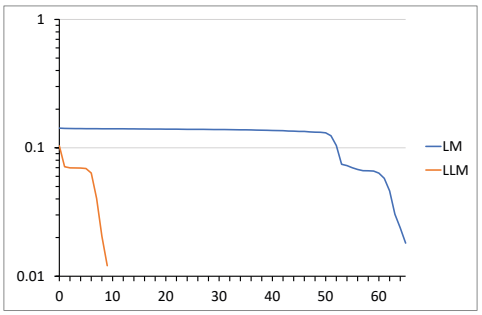

Figure 14. Exemplary convergence process for the points classification training using the FCC-18 network

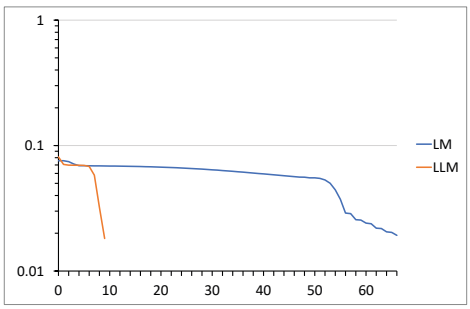

Figure 15. Exemplary convergence process for the points classification training using the FCMLP-7-7-1 network

5 Conclusion

The local Levenberg-Marquardt algorithm has been developed in order to mitigate the greatest disadvantages of its precursor. The proposed optimiza-

tion eliminates a huge computational overhead that occurs while utilizing the classic LM algorithm for big networks and training sets. This is possible by assembling the Jacobian matrices locally for each neuron. These operations are completely independent of each other, hence can be performed in parallel. Due to growing capabilities of multiprocessing devices, parallelization of training algorithms should be a natural direction of their evolution as initially proposed in [37, 38, 39, 40, 41, 42, 43]. As the next step of our research in the scope of the LLM algorithm, a parallel implementation will be attempted.

The presented experiment contains a total of 7 various benchmarks that cover 4 approximation and 3 classification cases. Single variable function benchmarks are characterized by rather small networks and a small number of samples. The overall success ratio in this scope is high establishing more than 94% and 85% for the logistic and the composite functions benchmarks, respectively. The LLM training time is shorter and the required epoch count is much smaller than for the classic LM variant.

However, the two variable function approximation benchmarks were more challenging. These scenarios utilize more complex networks and longer training sets. In both, the Hang and Sinc problems, a significant time improvement can be seen. It is worth noticing that also the average epoch count is reduced in trainings held by the LLM algorithm. This is caused by reducing the Jacobian matrix size several times in the LLM comparing to the classic LM algorithm.

The attempted classification benchmarks contain the encoder simulation, parity detection and points in the circle classification problems. While the first two of them utilize dedicated network topologies, the points classification benchmark involves a wide range of tested networks. In all classification scenarios the LLM algorithm has achieved a very high success ratio, which in most cases equals 100%. The classification benchmarks also confirm a significant average training time reduction due to far smaller Jacobian matrices.

The greatest benefit of using the LLM algorithm is visible in benchmarks that utilize big networks accompanied by long training sets. The training time is significantly reduced due to a smaller Jacobian matrix size in the LLM algorithm.

Based on the obtained results, the presented LLM variant reveals new optimization perspectives for the well known classic Levenberg-Marquardt algorithm. The top highlights discussed in this paper in the scope of the LLM algorithm can be summarized as follows:

1. The implementation effort for the LLM is not significantly bigger than for the classic Levenberg-Marquardt algorithm.

2. The obtained results show that the proposed solution performs relatively better than the classic LM algorithm in terms of the training time.

3. Obtaining a much shorter training time for the LLM algorithm results from a significant reduction of the Jacobian matrix size.

4. The overall success ratio of attempted runs is satisfactory.

5. The LLM algorithm is prone to the parallel implementation, which requires only basic synchronisation techniques.

In our future work we plan to extend our algorithm to deal with temporal dynamics and to implement our algorithm to solve several industrial problems, see e.g. [44, 45, 46].

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 - 2022 project number 020/RID/2018/19, the amount of financing PLN 12,000,000.00.

References

[1] Ryotaro Kamimura. Supposed maximum mutual information for improving generalization and interpretation of multi-layered neural networks. Journal of Artificial Intelligence and Soft Computing Research, 9(2):123–147, 2019.

[2] M. Abbas M. Javaid, Jia-Bao Liu, W. C. Teh, and Jinde Cao. Topological properties of four-layered neural networks. Journal of Artificial Intelligence and Soft Computing Research, 9(2):111–122, 2019.

[3] Oded Koren, Carina Antonia Hallin, Nir Perel, and Dror Bendet. Decision-making enhancement in a big data environment: Application of the k-means algorithm to mixed data. Journal of Artificial Intelligence and Soft Computing Research, 9(4):293–302, 2019.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Sou Nobukawa, Haruhiko Nishimura, and Teruya Yamanishi. Pattern classification by spiking neural networks combining self-organized and reward-related spike-timing-dependent plasticity. Journal of Artificial Intelligence and Soft Computing Research, 9(4):283–291, 2019.

[11] Miguel Costa, Daniel Oliveira, Sandro Pinto, and Adriano Tavares. Detecting driver's fatigue, distraction and activity using a non-intrusive ai-based monitoring system. Journal of Artificial Intelligence and Soft Computing Research, 9(4):247–266, 2019.

[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] Gustavo Botelho de Souza, Daniel Felipe da Silva Santos, Rafael Gonçalves Pires, Aparecido Nilceu Marananil, and Jo ao Paulo Papa. Deep features extraction for robust fingerprint spoofing attack detection. Journal of Artificial Intelligence and Soft Computing Research, 9(1):41–49, 2019.

[20] Apeksha Shewalkar, Deepika Nyavanandi, and Simone A. Ludwig. Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru. Journal of Artificial Intelligence and Soft Computing Research, 9(4):235–245, 2019.

[21] 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.

[22] 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.

[23] 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.

[24] Max W. Y. Lam. One-match-ahead forecasting in two-team sports with stacked bayesian regressions. Journal of Artificial Intelligence and Soft Computing Research, 8(3):159–171, 2018.

[25] 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.

[26] 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.

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

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

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

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

[39] 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.

[40] Jarosław Bilski and Jacek Smol. 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.

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

[42] Jarosław Bilski, Jacek Smolag, 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.

[43] 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.

[44] Ewaryst Rafajłowicz and Wojciech Rafajłowicz. Iterative learning in optimal control of linear dynamic processes. International Journal of Control, 91(7):1522–1540, 2018.

[45] Ewaryst Rafajłowicz and Wojciech Rafajłowicz. Iterative learning in repetitive optimal control of linear dynamic processes. LNCS, 9692:705–717, 06 2016.

[46] Piotr Jurewicz, Wojciech Rafajłowicz, Jacek Reiner, and Ewaryst Rafajłowicz. Simulations for tuning a laser power control system of the cladding process. In Khalid Saeed and Władysław Homenda, editors, Computer Information Systems and Industrial Management, pages 218–229, Cham, 2016. Springer International Publishing.

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 about 60 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.

Bartosz Kowalczyk received the M.Sc. degree in computer science from Częstochowa University of Technology in 2015 and now is a Ph.D. student in computer science in the Department of Computational Intelligence at 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 conference proceedings. M.Sc. Kowalczyk attended several Conferences on Artificial Intelligence and Soft Computing.

Alina Marchlewska received the M.Sc. degree in mathematics from the Department of Mathematics and Computer Science, University of Łódź, Poland. She obtained the Ph.D. degree in mathematics from the same University. Currently, she is an assistant professor at the Social Academy of Sciences in Lodz. Her research interests include social computing and applications of various artificial intelligence technologies and computing methods in selected IT problems.

Jacek M. Zurada, Ph.D., (Life Fellow IEEE'14, INNS Fellow) has received his degrees from Gdansk University of Technology, Poland. He now serves as a Professor of Electrical and Computer Engineering at the University of Louisville, Kentucky. He authored or co-authored several books and over 420 papers in computational intelligence, neural networks, machine learning, and rule extraction, and delivered over 100 invited talks in Mexico, Chile, The Netherlands, China, India, Singapore, Turkey, Hong Kong, Hungary, Germany, Malaysia, Poland, and Italy. His work has been cited over 14,000 times (Google Scholar).

In 2014 he served as IEEE V-President, Technical Activities (TAB Chair). He also chaired the IEEE TAB Strategic Planning Committee (2016), IEEE TAB Periodicals Committee (2010-11), and TAB Periodicals Review and Advisory Committee (2012-13), and was the Editor-in-Chief of the IEEE Transactions on Neural Networks (1997-03), Associate Editor of the IEEE Transactions on Circuits and Systems, Neural Networks and was member of the Editorial Board of The Proceedings of the IEEE. In 2004-05, he served as President of the IEEE Computational Intelligence Society. He is a Distinguished Lecturer for IEEE Systems, Man and Cybernetics Society.

Professor Jacek Zurada is an Associate Editor of Neurocomputing, and of several international journals. He is a member of the Polish Academy of Sciences. He has been awarded numerous distinctions, including the 2013 Joe Desch Innovation Award, 2015 UofL Distinguished Service Award, and five honorary professorships. He has been a Board Member of IEEE, IEEE CIS and IJCNN.