Neural Process Lett (2011) 34:293–303
DOI 10.1007/s11063-011-9199-4

A method to improve the performance of multilayer perceptron by utilizing various activation functions in the last hidden layer and the least squares method

Krzysztof Halawa

Published online: 12 October 2011

Abstract This article presents a fast and uncomplicated method to modify multilayer perceptrons allowing for a considerable single-step reduction of the cost function which in this case is the mean of squared errors. The method consists in, but is not limited to the change of neuron activation functions in the last hidden layer and in the single application of the least squares method. No changes are made to neuron weights in any hidden layer. Some essential strong points of the method lie in the fact that it can be used to improve operation of networks trained earlier and the learning process need not be started from the very beginning.

Keywords Multilayer perceptron · Activation functions · Least squares method

1 Introduction

Many problems are encountered with learning a multilayer perceptron (MLP). Local minima of the cost function cause serious difficulty. The results are greatly influenced by initial values of weights. Often, the learning process is run repeatedly with various initial weights. Usually, gradient algorithms, such as the Levenberg–Marquardt, the conjugate gradients and the variable metric ones are used to learn MLP. The papers [1,2] among others show the learning way with the recurrent least squares method (LSM). There are algorithms which successively use non-linear optimization techniques together with the LSM. The staggered training of MLP [3] is one of them. Initially, LSM is used to optimize the weights of the output layer. The weights of the remaining layers are then subjected to nonlinear optimization. These two steps are repeated alternately. When determination of the weights is completed, they are subjected to pruning in order to reduce the number of connections and to improve the network generalization capability. The MLP learning problems convinced numerous researchers to search for other network structures where all parameters are subjected to changes during learning and could be optimized in a single step using LSM. For instance, the networks with orthogonal

K. Halawa (✉)
Wroclaw University of Technology, 27 Wybrzeze Wyspianskiego St., 50-370 Wroclaw, Poland
e-mail: krzysztof.halawa@pwr.wroc.pl

activation functions where changes are made only for linear output neurons were proposed in [4–7]. A significant drawback of these networks lies in the fact that as the number of inputs increases, the number of weights grows exponentially. In [8] it was proven that for MLP with one hidden layer and with neurons having sigmoidal activation functions, the integrated squared error is of the order $O(\frac{1}{n})$, where n is the number of neurons in the hidden layer. It was assumed that the function approximated by the network was bound on the first moment of the magnitude distribution of the Fourier transform. Barron [8] also demonstrated that for networks with radial basis functions (RBFs), the integrated squared approximation error cannot be less than $O(\frac{1}{n^{2/q}})$, where q is the input-dimension.

Despite of a difficult learning process, MLP is one of the most popular network structures since it has essential advantages.

A predisposition for operation with multi-dimensional data being one of them. There are numerous applications where MLPs have hundreds of inputs and large amounts of neurons. From time to time, an operating network needs to be improved without the necessity to re-start the whole learning and pruning processes from the beginning. This article proposes such a method.

Neurons in the hidden layers of MLP have activation functions of the sigmoidal shape. An example of such a function is $f(x) = \tanh(x)$. In order to reduce the number of numerical calculations, the hyperbolic tangent can be replaced by the bipolar function $f(x) = \frac{2}{1+\exp(-2x)} - 1$ or the binary function $f(x) = \frac{1}{1+\exp(-2x)}$. Many microcontrollers have a low processing capacity. Look-up tables with values necessary for interpolation of the activation function or piecewise polynomial models [9] may be used in devices under the control of such microcontrollers. Numerous papers have been published where various shapes of activation functions were examined. The iteration process with non-linear optimization algorithms was used to select the parameters of these functions. In [10] the activation function $f(x) = \frac{a(1-\exp(-bx))}{1+\exp(-bx)}$ was used, where a and b were the parameters selected with gradient algorithms. In [11], the Catmull–Rom spline curves were applied.

In [12], it was proven that the perceptron with at least one hidden layer is a universal approximator provided the activation function of the neurons are squashing functions, i.e. a function $f : \mathcal{R} \to [0, 1]$ is a squashing function if it is non-decreasing, $\lim_{x\to\infty} f(x) = 1$ and $\lim_{x\to-\infty} f(x) = 0$ If $\lim_{x\to-\infty} f(x) = -1$, then this MLP is also the universal approximator.

The further structure of this article is as follows: Section 2 outlines the proposed method. Section 3 provides the results of numerical experiments. A comparison was made between the results gained by the proposed method as compared to those reached by MLP with sigmoid activation functions in all hidden layers. A summary is placed at the end of this article.

2 Proposed method

If the proposed method is used, the activation functions of the neurons in the last hidden layer may take various shapes. To select the shapes of these all activation functions, it is merely necessary to solve the set of the normal equations once. In this article, it was assumed that MLP, whose performance is to be improved, has neurons with sigmoid activation functions in hidden layers, whereas neurons in the output layer have the linear activation function $f_{out}(x) = x$. Many times this layer possesses neurons of the linear activation function.

For the sake of a concise notation, formulae are given for MISO MLP (Multiple Input, Single Output). A similar procedure is used for networks with several outputs. It was assumed that the minimized cost function is

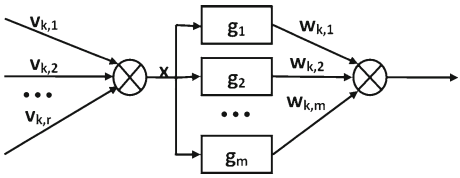

Fig. 1 The structure of the k-th neuron in the last hidden layer, $\otimes$ denotes the adder, r is the number of neuron inputs, $v_{k,1}, v_{k,2}, \ldots, v_{k,r}$ are the weights assigned to the connections with the previous layer

$$E = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{F}(\mathbf{u}_i) - d_i \right)^2, \tag{1}$$

where $\hat{F}(\mathbf{u}_i)$ is the network output value when the network inputs are equal to the elements of the vector $\mathbf{u}_i = \left[u_{1,i}, u_{2,i}, \ldots, u_{q,i}, \right]^T$, q is the number of the network inputs, d_i is the desired value of the network output assigned with $\mathbf{u}_i$, N is the number of the pairs $\{\mathbf{u}_i, d_i\}$ in the learning set. The formula (1) is commonly used as a cost function.

Let $f(x)$ denote the activation function in the last hidden layer prior application of the proposed method. Let $g_1(x), g_2(x), \ldots, g_m(x)$ be the consecutive functions of the series

$$f\left(\frac{x}{2^h}\right), f\left(\frac{x}{2^{h-1}}\right), \ldots, f(x), \ldots, f\left(2^{h-1}x\right), f\left(2^h x\right),$$

where h is a positive integer, $m = 2h + 1$. In applications where a small number of arithmetical operation is significant, the author of this article recommends assumption $h < 2$. Increasing of h may result in significant decrease of the cost function. After selection of h, the activation functions of all neurons in the last hidden layer are changed into the functions

$$f_k(x) = w_{k,1} g_1(x) + w_{k,2} g_2(x) + \cdots + w_{k,m} g_m(x), \tag{2}$$

where $f_k(x)$ denotes the activation function of the k-th neuron in the last hidden layer, $k = 1, \ldots, s$, s is the number of neurons in the last hidden layer, $w_{k,1}, \ldots, w_{k,m} \in \mathcal{R}$. The way of calculating $w_{k,1}, \ldots, w_{k,m}$ was described hereinafter. For various k, the $w_{k,1}, \ldots, w_{k,m}$ values are, mostly, not the same, therefore each neuron in the final hidden layer may have another activation function. Figure 1 shows the proposed structure of the neurons in this layer.

The next step consists of a change of values for all weights in the output layer into 1. When all weights in the output layer have the value 1, then the problem of optimizing the vector $\mathbf{w}$ reduces to solving the set of normal equations

$$\mathbf{Z}\mathbf{w} = \mathbf{d}, \tag{3}$$

where $\mathbf{w} = \left[w_{1,1}, w_{1,2}, \ldots, w_{1,m}, w_{2,1}, w_{2,2}, \ldots, w_{2,m}, \ldots, w_{s,1}, w_{s,2}, \ldots, w_{s,m}, b \right]^T$, $\mathbf{d} = [d_1, d_2, \ldots, d_N]^T$,

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1(\mathbf{u}_1) & \mathbf{z}_2(\mathbf{u}_1) & \cdots & \mathbf{z}_s(\mathbf{u}_1) & 1 \\ \mathbf{z}_1(\mathbf{u}_2) & \mathbf{z}_2(\mathbf{u}_2) & \cdots & \mathbf{z}_s(\mathbf{u}_2) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{z}_1(\mathbf{u}_N) & \mathbf{z}_2(\mathbf{u}_N) & \cdots & \mathbf{z}_s(\mathbf{u}_N) & 1 \end{bmatrix},$$

$\mathbf{z}_k(\mathbf{u}_i) = [g_1(x_k(\mathbf{u}_i)), g_2(x_k(\mathbf{u}_i)), \ldots, g_m(x_k(\mathbf{u}_i))], i = 1, \ldots, N, k = 1, \ldots, s, x_k(\mathbf{u}_i)$ is the weighted sum of neuron inputs for the k-th neuron in the last hidden layer, i.e. $x_k(\mathbf{u}_i) = \sum_{a=1}^{r} v_{k,a} \psi_{k,a}(\mathbf{u}_i)$, $\psi_{k,a}$ is the a-th input of the k-th neuron, b is the bias in the output layer.

The values of the new activation functions' parameters which minimize the cost function (1) are determined by the formula

$$\mathbf{w} = \left(\mathbf{Z}^T\mathbf{Z}\right)^{-1}\mathbf{Z}^T\mathbf{y}. \tag{4}$$

For the sake of numerical problems, the parameters of the activation functions should not be calculated directly using (4). Instead, they should be determined using the appropriate numerical methods for the set of the normal equations (3). As an example, the singular value decomposition may be applied. If the matrix $\mathbf{Z}$ is ill-conditioned, the truncated singular value decomposition or ride regression can be applied to calculate the vector $\mathbf{w}$ [13].

When some elements of vector $\mathbf{w}$ are much smaller than the remaining ones, the number of the terms of the sum in (2) may be reduced. Many algorithms exist to solve the modified least squares problem (MLSP), which is encountered when columns representing small-value vector elements are removed from the matrix $\mathbf{Z}$. Such a problem may be solved very simply by using the QR decomposition [13].

The proposed method can be outlined as follows:

(1) Changing the activation functions of all neurons in the last hidden layer into the functions given by (2)
(2) Changing the weights of all neurons in the output layer to 1.
(3) Solving the set of normal equations (3) using proper numerical methods.
(4) Calculating the value of the cost function (1)
(5) The possible removal from the matrix $\mathbf{Z}$ of the columns corresponding to the lowest values of $\mathbf{w}$, and solving the MLSP, that was created by this removal. Recalculating the value of the cost function. If the cost function changes significantly, item 5 is repeated. Otherwise, the changes introduced in this item are withdrawn.

The set of the equations (3) can be solved much faster than training MLP using non-linear optimization methods. One of the functions from the series $g_1(x), g_2(x), \ldots, g_m(x)$ is equal to $f(x)$. Therefore, it is obvious that the modified network must have at least the same approximation capabilities as MLP, prior to application of the method under consideration. The following section shows how the outlined method improved the performance of MLPs which were trained using the Levenberg–Marquardt algorithm.

3 Numerical experiments

In Sect. 3.1 the results obtained during approximation of a two-argument differentiable function are shown. In Sect. 3.2, the results obtained with use of benchmarks of the well-known UC Irvine Machine Learning Repository [14]. Two data sets were chosen, that featured with high numbers of instances and attributes. All experiments were carried out using a Pentium T4400 2.2 GHz, 3GB RAM computer. Computational tasks were made with Matlab software utilizing the learning function for the Levenberg–Marquardt algorithm included in the Neural Networks Toolbox.

3.1 Two-argument function approximation

Ten identical networks with one hidden layer were trained to map the function $F(u_1, u_2) = u_1^2 - u_2^2$, which is shown in Fig. 2. The output layer included one neuron with a linear activation function. Initial values of the weights were selected using the Nguyen–Widrow algorithm. The activation functions in the hidden layer were determined by the formula $f(x) =$

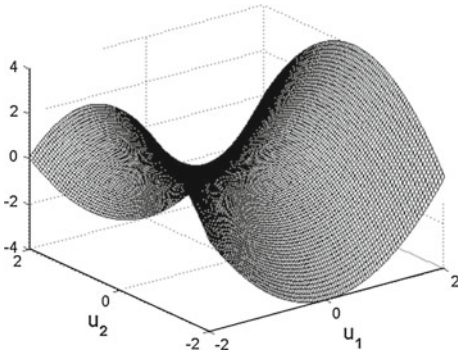

Fig. 2 The graph of the function $F(u_1, u_2) = u_1^2 - u_2^2$

$\frac{2}{1+exp(-2x)} - 1$. It was assumed that $g_1(x) = f(-0.25x)$, $g_2(x) = f(-0.5x)$, $g_3(x) = f(-x)$, $g_4(x) = f(-2x)$, $g_5(x) = f(-4x)$.

A set of 10,201 data was used for network learning. The values of network inputs in the learning set were evenly distributed over the area [-2,2] X [-2,2]. Initially, the networks were learnt using the Levenberg–Marquardt algorithm. Learning was terminated after 100 epochs as it was found that after 70 epochs the changes of the cost function (1) were insignificant. Following completion of learning with the Levenberg–Marquardt algorithm, the proposed method was used. Item 5 was not performed. The results are shown in Tables 1, 2 and in Fig. 3. Times in Tables 1, 2, 3, and 4 are in seconds. $\overline{F}_{MLP}(u_1, u_2)$ denotes the mean value of the network output following learning with the Levenberg–Marquardt algorithm. $\overline{F}_{MET}(u_1, u_2)$ represents the mean value of the network output after application of the proposed method.

By using the proposed method, the value of the cost function (1) was reduced by several orders in less than 1% of the time used for learning larger MLP with the Levenberg–Marquardt algorithm.

The next stage consisted of increasing the number of neurons in the hidden layer to 50. The results for this enlarged number of neurons are shown in Tables 3, 4 and in Fig. 4. Learning the network with 50 neurons was terminated after 200 epochs because it was found that, after 120 epochs the changes in the cost function (1) were very insignificant. Figure 5 shows the value of the cost function in consecutive epochs.

The proposed method allowed for the reduction in the value of the cost function (1) by several orders of magnitude both for networks with 10 and those with 50 neurons in the hidden layer. Significantly better results were reached using smaller networks trained with the described method as opposed to larger networks having five times more neurons in the hidden layer, that were learnt with the Levenberg–Marquardt algorithm. The application of the proposed method for the network with 10 neurons in the hidden layer was over 2,000 times faster than training MLP with 50 neurons in the hidden layers from the beginning with the Levenberg–Marquardt algorithm.

3.2 Experiments with UCI benchmarks

In this Section, the results obtained with use of benchmarks from the well-known UC Irvine Machine Learning Repository [14]. The data set Wine Quality and the data set Communities and Crimes were used. These databases were selected for they have a high number of instances and attributes.

The Wine Quality Data Set may be downloaded from the page http://archive.ics.uvi.edu/ml/datasets/Wine+Quality. A relevant article concerning this benchmark is [15]. For the

Table 1 Results for the networks with 10 neurons in the hidden layer

Network number	1	2	3	4	5
Value (1) after learning with the Levenberg–Marquard alg.	0.689×10^{-6}	0.782×10^{-6}	1.394×10^{-6}	2.660×10^{-6}	1.193×10^{-6}
Value (1) after using the proposed method	0.324×10^{-13}	0.199×10^{-13}	1.500×10^{-13}	28.95×10^{-13}	29.05×10^{-13}
Learning time for the Levenberg–Marquard alg.	22.04	21.73	22.41	21.82	21.79
Resolving time for set of equations (3)	0.156	0.156	0.141	0.140	0.156
Network number	6	7	8	9	10
Value (1) after learning with the Levenberg–Marquard alg.	1.407×10^{-6}	1.007×10^{-6}	1.733×10^{-6}	2.854×10^{-6}	6.526×10^{-6}
Value (1) after using the proposed method	5.277×10^{-13}	4.863×10^{-13}	3.833×10^{-13}	0.005×10^{-13}	5.532×10^{-13}
Learning time for the Levenberg–Marquard alg.	21.96	21.79	21.79	21.77	21.82
Resolving time for set of equations (3)	0.171	0.156	0.156	0.156	0.156

Table 2 Mean values of the results from Table 1

Value (1) after learning with the Levenberg–Marquard alg.	1.768×10^{-6}
Value (1) after using the proposed method	7.947×10^{-13}
Learning time for the Levenberg–Marquard alg. $\bar{t}_{LevMar}$	21.89
Resolving time for set of equations (3) $\bar{t}_{LSM}$	0.154
$\bar{t}_{LevMar}/\bar{t}_{LSM}$	142.2

experiments, the data set of the Portuguese red wines "Vinho Verde" was used, that includes 1600 instances. In this data set, due to privacy and logistic issues only physicochemical and sensory attributes are given. There is no information concerning prices, grape sort, etc. The input variables are: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol. The output variable is the wine quality that is contained within the interval from 0 up to 10.

The other data set used was Communities and Crime Data Set that may be downloaded from http://archive.ics.uci.edu/ml/datasets/Communities+and+Crime. It includes 1994 instances. In this set, there are 128 attributes: 127 input variables such as, e.g., the number of unemployed people, the median of wages, percentage of people living in areas classified as urban, the number of police officers, etc. The attribute intended for predicting is the number of crimes (per capita violent crime). For the experiments, 21 input variables were used (the

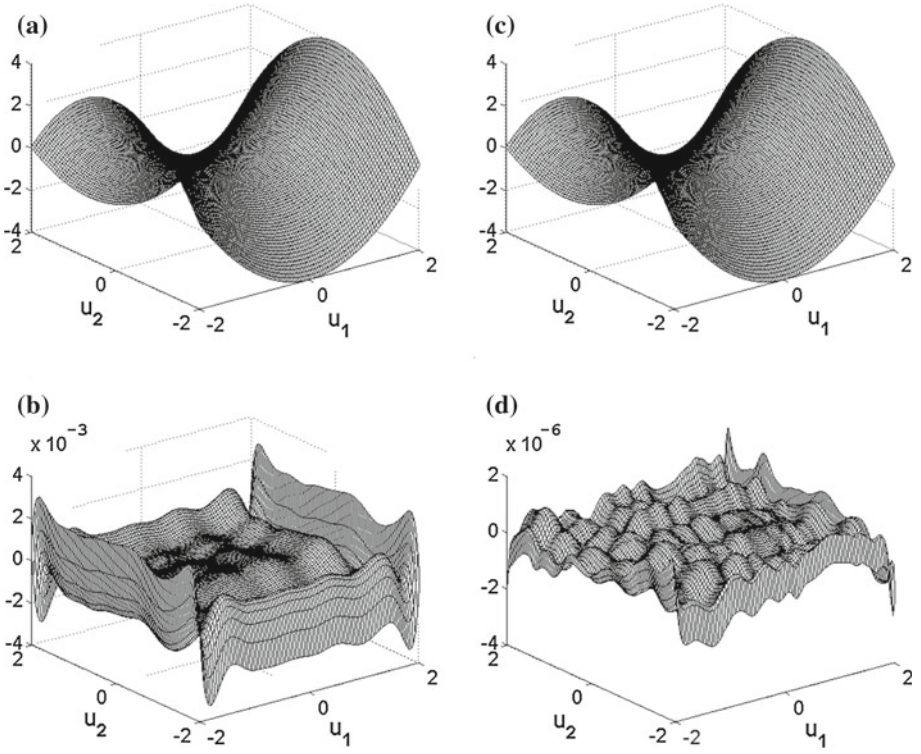

Fig. 3 Graphs for the network with 10 neurons in the hidden layer: **a** $\overline{F}_{MLP}(u_1, u_2)$, **b** $F(u_1, u_2) - \overline{F}_{MLP}(u_1, u_2)$, **c** $\overline{F}_{MET}(u_1, u_2)$ and **d** $F(u_1, u_2) - \overline{F}_{MET}(u_1, u_2)$

Table 3 Results for the network with 50 neurons in the hidden layer

Network number	1	2	3	4	5
Value (1) after learning with the Levenberg–Marquard alg.	0.447×10^{-7}	0.031×10^{-7}	0.125×10^{-7}	0.646×10^{-7}	0.974×10^{-7}
Value (1) after using the proposed method	3.665×10^{-17}	1.070×10^{-17}	2.284×10^{-17}	3.475×10^{-17}	2.233×10^{-17}
Learning time for the Levenberg–Marquard alg.	325.9	325.4	324.3	323.8	324.9
Resolving time for set of equations (3)	2.949	2.918	2.932	2.902	2.948
Network number	6	7	8	9	10
Value (1) after learning with the Levenberg–Marquard alg.	1.829×10^{-7}	0.994×10^{-7}	1.937×10^{-7}	5.331×10^{-7}	0.624×10^{-7}
Value (1) after using the proposed method	4.759×10^{-12}	4.637×10^{-12}	2.089×10^{-12}	1.122×10^{-12}	7.487×10^{-12}
Learning time for the Levenberg–Marquard alg.	324.4	326.9	328.8	324.7	330.9
Resolving time for set of equations (3)	2.902	2.808	2.823	2.964	3.198

Table 4 Mean values of the results from Table 3

Value (1) after learning with the Levenberg–Marquard alg.	1.288×10^{-7}
Value (1) after using the proposed method	7.179×10^{-13}
Learning time for the Levenberg–Marquard alg. $\bar{t}_{LevMar}$	326.05
Resolving time for set of equations (3) $\bar{t}_{LSM}$	2.93
$\bar{t}_{LevMar}/\bar{t}_{LSM}$	111.2

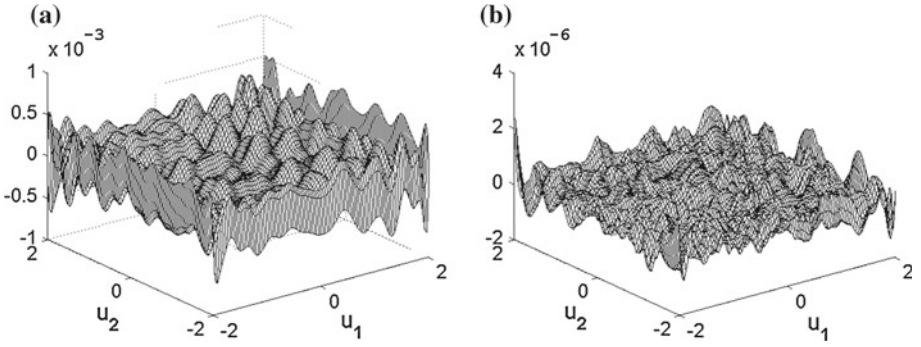

Fig. 4 Graphs for the network with 50 neurons in the hidden layer: **a** $F(u_1, u_2) - \overline{F}_{MLP}(u_1, u_2)$ and **b** $F(u_1, u_2) - \overline{F}_{MET}(u_1, u_2)$

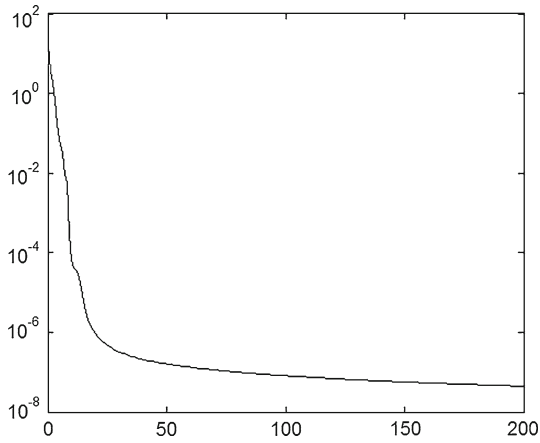

Fig. 5 Changes of the cost function (1) in successive epochs for MLP learnt, with the Levenberg–Marquardt algorithm. This MLP had 50 neurons in the hidden layer

fields from 18 up to 38). The related data set used in [16]. That article includes a description of the integration of the three files [17].

20 identical networks were learnt to predict the wine quality. These networks had 11 inputs, each. In the hidden layer, there were 40 neurons. In the output layer, there was one neuron with a linear activation function. The initial values of the weights were chosen with the Nguyen–Widrow algorithm. The learning was terminated after 100 epochs since the cost function changes were very insignificant. Following completion of the learning with the Levenberg–Marquardt algorithm, the proposed method was used. Item 5 was not performed. $h = 2$ was assumed. In the Table 5, the mean values of the results are presented.

To predict the number of crimes, 10 networks were used, that had 21 inputs and 35 neurons in the hidden layer, each. The learning with the Levenberg–Marquardt algorithm was

Table 5 Mean values of the results for the Wine Quality data set

Value (1) after learning with the Levenberg–Marquard alg.	0.2614
Value (1) after using the proposed method	0.2287
Learning time for the Levenberg–Marquard alg. $\bar{t}_{LevMar}$	71.1542
Resolving time for set of equations (3) $\bar{t}_{LSM}$	0.2046
$\bar{t}_{LevMar}/\bar{t}_{LSM}$	347.8

Table 6 Mean values of the results for the Communities and Crime data set

Value (1) after learning with the Levenberg–Marquard alg.	4.1619×10^{-4}
Value (1) after using the proposed method	3.6721×10^{-4}
Learning time for the Levenberg–Marquard alg. $\bar{t}_{LevMar}$	1000.1
Resolving time for set of equations (3) $\bar{t}_{LSM}$	0.2776
$\bar{t}_{LevMar}/\bar{t}_{LSM}$	3602.9

terminated after 50 epochs. In the proposed method, $h = 2$ was assumed. The mean values of the results are presented in the Table 6.

In a time shorter than the duration of one epoch of learning for the Levenberg–Marquardt algorithm, the proposed method enabled to decrease the cost function value of more than 10%. Application of the described method is recommended, in particular, when further teaching with use of gradient-type algorithms results in only very insignificant changes of the cost function. Probably, when higher h was chosen, the cost function value would be noticeably lower, but the equation set solving time would be higher.

4 Conclusions

This article presents the method which provides for an uncomplicated and efficient way to improve MLP approximation capabilities for many functions. It allows for optimization in individual shapes of the activation function for each neuron in the last hidden layer. To this purpose, it is sufficient to solve the set of the normal equations (3) only once. The proposed method can also be used to improve the performance of the networks trained at an earlier time. It is perfectly suited to be applied to further improve the operation of the application wherein MLP successfully works for some time. In many cases, it is an attractive alternative to the usual approach which consists of enlarging the amount of neurons as well as the re-starting of the learning process from the beginning. It requires neither the iterative learning process nor large operational memory. As shown in Sect. 3.1, in less than 1% of time used to train MLP with the Levenberg–Marquardt algorithm, it may well reduce the value of the cost function (1) by several orders of magnitude. The author of this article supposes that the described method fits especially well to approximation of functions, the partial derivatives of which change slowly on a big part of the independent variable domain.

The experiments conducted with the data sets of the UCI Machine Learning Repository confirmed that, for networks of a higher number of neurons in the hidden layer and inputs, the application of the proposed method is reasonable and makes it possible, within a short time, to decrease the cost function value though further teaching with the gradient-type methods does not result in a noticeable improvement.

An essential advantage of the described method is its easy implementation with special commands for solving sets of linear equations that are available in most mathematical

computing environments. Free open-source libraries with proper numerical methods exist for each popular programming language. It is also worth mentioning that the proposed method is applicable for use with look-up tables or piecewise polynomial models. This method requires no change in weights of the neurons in any hidden layer.

References

1. Al-Batah MS, Isa NAM, Zamli KZ, Azizli KA (2010) Modified recursive least squares algorithm to train the hybrid multilayered perceptron (HMLP) network. Appl Soft Comput 10:236–244
2. Azimi-Sadjadi MR, Liou RJ (1992) Fast learning process of multilayer neural networks using recursive least squares method. IEEE Trans Signal Process 40(2):446–450
3. Nelles O (2001) Nonlinear system identification: from classical approaches to neural networks and fuzzy models. Springer, Berlin, pp 253–255
4. Tseng CS, Chen CS (2004) Performance comparison between the training method and the numerical method of the orthogonal neural network in function approximation. Int J Intell Syst 19(12):1257–1275
5. Zhu C, Shukla D, Paul FW (2002) Orthogonal functions for system identification and control. In: Leondes CT (ed) Neural network systems techniques and applications. Control and dynamic systems, vol 7. Academic Press, San Diego, pp 1–73
6. Rafajowicz E, Pawlak M (1997) On function recovery by neural networks based on orthogonal expansions. In: Proc. 2nd world congress of nonlinear analysis. Nonlinear analysis, theory and applications, vol 30, no 3. Pergamon Press, Oxford, pp 1343–1354
7. Halawa K (2008) Fast and robust way of learning the Fourier series neural networks on the basis of multidimensional discrete Fourier transform. In: Lecture notes in computer science. Lecture notes in artificial intelligence, vol 5097. Springer, New York, pp 62–70
8. Barron AR (1993) Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans Inform Theory 39(3):930–945
9. Ebert T, Bänfer O, Nelles O (2010) Multilayer perceptron network with modified sigmoid activation functions. In: Artificial intelligence and computational intelligence. Lecture notes in computer science, vol 6319. Springer, New York, pp 414–421
10. Chen C, Chang W (1996) A feedforward neural network with function shape autotuning. Neural Netw 9(4):627–641
11. Guarnieri S, Piazza F (1999) Multilayer feedforward networks with adaptive spline activation function. IEEE Trans Neural Netw 10(3):672–683
12. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
13. Björck A (1996) Numerical methods for least squares problems. SIAM: Society for Industrial and Applied Mathematics, Philadelphia
14. Frank A, Asuncion A (2011) UCI machine learning repository. University of California, School of Information and Computer Science, Irvine. http://archive.ics.uci.edu/ml
15. Cortez P, Cerdeira A, Almeida F, Matos T, Reis J (2009) Modeling wine preferences by data mining from physicochemical properties. Decis Support syst 47(4):547–553
16. Redmond MA, Baveja A (2002) A data-driven software tool for enabling cooperative information sharing among police departments. Eur J Oper Res 141:660–678
17. Computer Files: Department of Commerce, Bureau of the Census, Census of Population and Housing 1990. United States: Summary Tape File 1a and 3a, U.S. Department of Justice, Federal Bureau of Investigation (1995) Crime in the United States, U.S. Department of Justice, Bureau of Justice Statistics, Law Enforcement Management and Administrative Statistics. U.S. Department Of Commerce, Bureau Of The Census Producer, Washington and Inter-university Consortium for Political and Social Research Ann Arbor