Advances in Artificial Intelligence and Machine Learning; Short Communication 4 (1) 2103-2112 Received 8-01-2024; Accepted 15-03-2024; Published 22-03-2024

A Free From Local Minima Algorithm for Training Regressive MLP Neural Networks

Augusto Montisci augusto.montisci@unica.it
University of Cagliari
09123 via Marengo 2, Cagliari,
Italy

Corresponding Author: Augusto Montisci

Abstract

In this article an innovative method for training regressive MLP networks is presented, which is not subject to local minima. The Error-Back-Propagation algorithm, proposed by William-Hinton-Rummelhart, has had the merit of favoring the development of machine learning techniques, which has permeated every branch of research and technology since the mid-1980s. This extraordinary success is largely due to the black-box approach, but this same factor was also seen as a limitation, as soon more challenging problems were approached. One of the most critical aspects of the training algorithms was that of local minima of the loss function, typically the mean squared error of the output on the training set. In fact, as the most popular training algorithms are driven by the derivatives of the loss function, there is no possibility to evaluate if a reached minimum is local or global. The algorithm presented in this paper avoids the problem of local minima, as the training is based on the properties of the distribution of the training set, or better on its image internal to the neural network. The performance of the algorithm is shown for a well-known benchmark.

1. INTRODUCTION

Even if Machine Learning (ML) includes a multiplicity of paradigms, much different among them, most part can be considered as an evolution of Error Backpropagation algorithm (EBP) of Multi-Layer Perceptron (MLP) [1]. The credit of this algorithm consists in the fact that for the first time it was possible to train networks with an intermediate layer, and therefore reproduce non-linear input-output relationships. This was as true for classification problems as it was for regression problems. Concerning the latter ones, it has been demonstrated [2], that a Perceptron with a single hidden layer is a universal approximator, however leaving open the problem of determining both the number of neurons needed to solve a specific problem, and how to determine the connection weights. The EBP algorithm offered a tool to determine the value of the parameters, while the determination of the optimal number of neurons is still an open problem. Successively, methods have been presented to address both questions [3–8], but the EBP paradigm, with an important series of variations, still represents the standard of machine learning. This paradigm consists in finding the minimum of a loss function, which is typically given by the output mean squared error with respect to the target value. All the minimization techniques also developed in contexts other than that of ML have been proposed to solve this problem, but the standard is represented by the use of first and second order descent methods [9], in whose category the EBP itself falls. First order algorithms, such as EBP, have made a comeback with the advent of Deep Learning [10], as the huge number of parameters makes second order methods impractical, even in cases in which approximate expressions of the Hessian are used. The methods based on the derivatives of the loss function have the advantage of being simple to implement but lack a criterion that allows to establish whether a stationarity point of the loss function represents a global minimum or not. However, there is another limitation that derives from this type of black-box approach. In fact, the fact of considering the loss function as the only indicator of the network performance leads to not distinguishing the different roles of the single parts of the network, as well as the relevance of single examples of the training set.

As a premise to the description of the algorithm that will be presented in the next sections, it is worth rethinking the structure of the MLP network, going beyond the black-box interpretation. First, the need for the hidden layer derives from the fact that the input-output relationship is nonlinear. If this were not the case, a linear hidden layer would be completely useless, because the cascade of the two connecting layers is equivalent to their product. Furthermore, since the algebraic relationship between the output of the hidden layer, called *feature space* [11], and the output of the neural network is linear, we can think that the task of the hidden layer is to make linear the input-output relationship. In other words, for each output neuron, the points of the training set in the product space $\{feature\}\times\{output\}$ must be coplanar. In fact, any residual nonlinearity downstream of the feature space could not be corrected by the last layer of connections. Hence there is no reason to include in the loss function the weights of the last connections layer, since given the points in the *feature space*, the optimal set of weights is the one that corresponds to the linear regression plane.

Another important consideration concerns the different role of the hidden layer and the output layer. Indeed, if the former represents the degrees of freedom of the network, the latter constitutes a constraint, because the same distribution in the feature space must satisfy the coplanarity constraint with respect to different outputs.

This paper presents a training algorithm based on this interpretation of the algebraic structure of the MLP. The objective of the training is no longer to minimize the mean squared error of the output, but rather the coplanarity of the training points in the product space {feature} × {output}. If in this space some points lay outside the regression hyperplane, this displacement is put in evidence, and it can be expressed in terms of the weights of the first layer of connections. The proposed paradigm avoids local minima, because there isn't a factor which masks the error, like the derivatives of the activation function do in MSE derivative-based algorithms. The focus of training is the first layer connections, as it moves the training points in the {feature} × {output} space.

For sake of simplicity and without prejudice to the general validity of the results, in this article the case of regressive MLP networks with only one output (MISO) will be treated. The paper is structured as follows: in the first section, the analytical basis of the procedure is presented. The second section briefly describes the choices adopted for implementation in the Matlab environment. The third shows the results obtained with a well-known benchmark. At the end, conclusions are provided.

2. DESCRIPTION OF THE ALGORITHM

In this section the training algorithm will be described, referring to the scheme shown in FIGURE. 1.

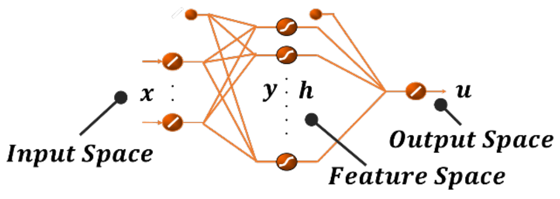

Figure 1: Multilayer Perceptron scheme

The choice of having only one output serves to simplify the discussion but it does not affect the generality of the conclusions. The MLP network implements an algebraic structure represented by the system of equations (1)

$$\begin{cases} \mathbf{W} \cdot \mathbf{x} + \mathbf{d} = \mathbf{y} \\ \mathbf{h} = \sigma(\mathbf{y}) \\ \mathbf{V} \cdot \mathbf{h} + b = u \end{cases} \tag{1}$$

where **x** is the input to the network, **W** is the weights matrix of the first layer of connections, **d** is the bias of the first layer, **y** is the input to the hidden layer, $\sigma(\,)$ is the nonlinear activation function of the hidden layer, **h** is the image of the input in the *feature space*, V is the weights vector of the second layer of connections, b is the bias of the second layer of connections, u is the output of the network. The first layer of connections linearly transforms the points from the input space of the network into the input space of the hidden layer. The hidden layer activation function is the only nonlinearity of the network. Through the first two transforms of (1), the distribution of the training set points in the input space is transformed into a distribution in the *feature space*. The first layer of connections must ensure that the points of the training set in the product space $\{feature\ space\} \times \{output\ space\}$ are all coplanar. If this does not happen, the second layer of connections will not be able to compensate for these displacements. The solution that minimizes the least squares is the regression plan. For this reason, even in conventional training algorithms, it is not convenient to include the weights of the second layer of connections within the loss function. It is convenient to define the loss as a function of the weights of the first layer and, by freezing this, to define the second layer by calculating the regression plane.

The algorithm presented in this paper, instead of finalizing the search to minimize the mean squared error of the output, aims to construct the feature space in such a way as to ensure the coplanarity of the points, or equivalently that the level curves of the function in the input space are transformed into straight lines in the feature space (FIGURE 2).

The condition for linearization can be expressed analytically as follows:

$$\mathbf{s} \cdot \sigma(\mathbf{W} \cdot \mathbf{x}_i + \mathbf{d}) = a \cdot u_i \qquad \forall\, i = 1, \ldots, N \tag{2}$$

where s is a vector of linear combination coefficients of the points in the *feature space*, while a is a scalar of arbitrary value, which determines the distance between the straight lines of level, and consequently the slope of the regression plan. Finally, N is the number of points in the training set. Dividing the equation (2) by a eliminates the uncertainty of the solution:

$$\hat{\mathbf{s}} \cdot \sigma(\mathbf{W} \cdot \mathbf{x}_i + \mathbf{d}) = u_i \qquad \forall\, i = 1, \ldots, N \tag{3}$$

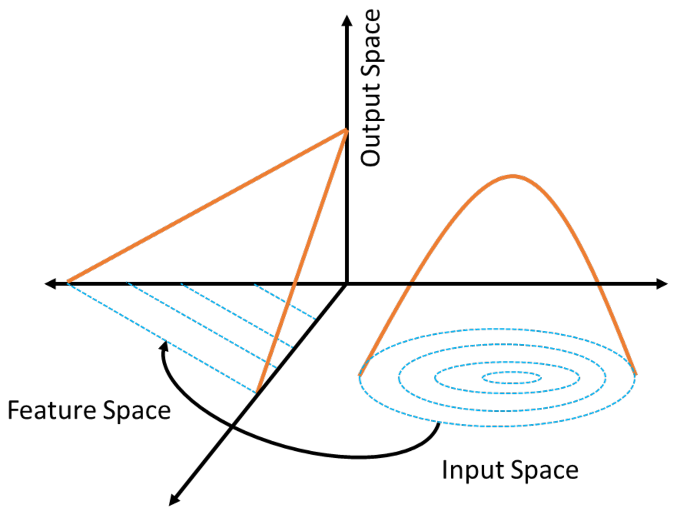

Figure 2: Linearization performed by the hidden layer

where $\hat{s}$ is the normalized variable. Globally the equation (3) is a system of N nonlinear equations in the variables $\mathbf{W}$, $\mathbf{d}$, $\hat{s}$. Fixing the values of $\mathbf{W}$ and $\mathbf{d}$, the same system becomes a system of linear equations overdetermined in the single variable $\hat{s}$. The proposed algorithm starts from a first order approximation of the left-side hand of the equation (3). Let $\mathbf{W}_0$, $\mathbf{d}_0$, $\hat{s}_0$ be a starting solution of the system (3) and $\delta\mathbf{W}$, $\delta\mathbf{d}$, $\delta\hat{s}$ the respective increments of the three variables which provide the solution:

$$(\hat{s}_0 + \delta\hat{s}) \cdot \sigma \left[(\mathbf{W}_0 + \delta\mathbf{W}) \cdot \mathbf{x}_i + (\mathbf{d}_0 + \delta\mathbf{d})\right] = u_i \tag{4}$$

Let replace the left-side hand of (4) with a first-order approximation of the incremented function:

$$(\hat{s}_0 + \delta\hat{s}) \cdot \left[\boldsymbol{\sigma}_0 + \nabla\sigma\left(\mathbf{W},\ \mathbf{d}\right) \odot \delta_{\mathbf{W},\ \mathbf{d}} \cdot \mathbf{1}\right] \approx u_i \tag{5}$$

where $\boldsymbol{\sigma}_0$ is the vector in the feature space corresponding to a zero increment, $\nabla\sigma\left(\mathbf{W},\ \mathbf{d}\right)$ is the gradient calculated with respect to the weights of the first layer connections, $\delta_{\mathbf{W},\ \mathbf{d}}$ is the increment of the variables $\mathbf{W}$ and $\mathbf{d}$, $\odot$ is the element-wise multiplication, $\mathbf{1}$ is a column vector of only 1s. Reordering the equation (5) and neglecting the infinitesimals of higher order we get:

$$\boldsymbol{\sigma}_0 \cdot \delta\hat{s} + \hat{s}_0 \cdot \nabla\sigma\left(\mathbf{W},\ \mathbf{d}\right) \odot \delta_{\mathbf{W},\ \mathbf{d}} \cdot \mathbf{1} \approx u_i - \hat{s}_0 \cdot \boldsymbol{\sigma}_0 \tag{6}$$

The equation (6) defines a linear, overdetermined system of equations, whose unknowns are the increments of the parameters. By combining (3) with (6) it is possible to implement an iterative procedure which provides the weights of the first layer of connections. Considering the fact, as previously indicated, that the second layer of connections is uniquely determined once the weights of the first layer have been assigned, the above-described iterative procedure provides the solution of the training problem.

The procedure takes place as follows. Given the initial value of the weights of the first layer of connections, the system (3) is solved into the variable $\hat{s}$. Since the system is overdetermined, it can only be solved by minimizing the root mean square error [12]. Using the obtained solution, together with the same weights of the first layer used previously, the coefficients and the known term of the system (6) are obtained. This system is also overdetermined, and therefore it too will have to be solved according to least squares. Only the increments of the weights of the first layer $\delta_{\mathbf{W},\ \mathbf{d}}$ are calculated, while $\hat{\mathbf{s}}$ is updated by solving the system (3) after updating the coefficients based on the new values of $\mathbf{W}$ and $\mathbf{d}$. The iterative process ends when the error in the system (3) is less than a pre-set threshold value. Alternatively, since the second layer of weights is uniquely determined by the first layer of weights, the error value of the current solution can be computed directly on the neural network output.

Since a first-order approximation is being used, the question remains open as to what the appropriate step length is to take at each iteration. For this reason, the *Line Search* method is adopted [13], sampling the segment from the current solution to the one obtained with the first-order approximation, and evaluating for each point the value of the output error. At each iteration, the assigned increment will be the one that corresponds to the minimum error.

As can be deduced, the number of operations to be performed for each iteration is much higher than those of a conventional method, but the proposed method has the advantage of not being subject to local minima, since the objective is no longer minimizing the mean squared error of the output, but rather solving a system of equations, and therefore the search for the solution is driven by the goal of the training, rather than, as usually happens, being constituted by an aggregate target function.

It is worth noting that the proposed method, as it is not driven by an aggregate loss function, allows the use of different criteria in the application of the line search, such as for example the minimization of the maximum error in the training set. The complete analysis of the potential of the method goes beyond the objectives of this paper and will be the subject of future developments.

3. RESULTS

A well-known benchmark for testing optimization methods [14], is used as an application example. In cases where the search for the optimal solution is carried out with the aid of a neural network [15], the accuracy of the network and its generalization capability assume fundamental importance. The general expression of the function is:

$$f\left(\mathbf{x}\right) = 418.9829 \cdot n - \sum_{i=1}^{n} x_i \ sin\left(\sqrt{|x_i|}\right) \tag{7}$$

where n is the size of the space where the function is defined. In this paper a dimension $n = 3$ was used, and a range [-500, 500] for the input variables. FIGURE 3 shows the Schwefel's function used as a test. The value of the function is represented by the color shades. It is apparent that the function has many maxima and minima, which implies that many hidden neurons will be required to adequately fit it. The Training Set consists of 5152 samples equispaced in the Input Space.

An MLP network with 200 hidden nodes has been trained for 4000 epochs. After 4000 epochs the MSE reached the value of $4.92 \cdot 10^{-5}$. FIGURE 4 shows the diagram of the mean squared error during the training. As can be seen, the error value after 4000 epochs had not yet stabilized, although the decrease had become much slower. Note that the error in the diagram refers to the output data previously normalized between -1 and 1. As can be seen, in the first iterations there is a considerable reduction of the error, while subsequently the descent speed is considerably reduced both in absolute

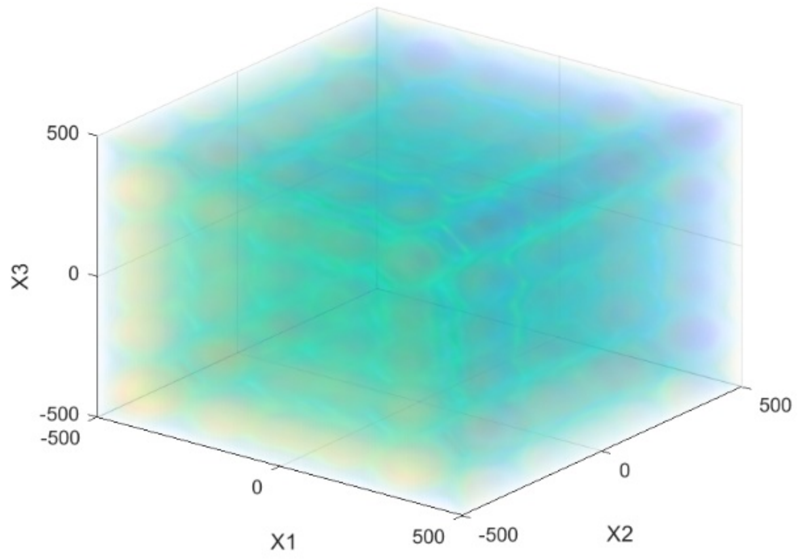

Figure 3: Schwefel's function defined in $n = 3$

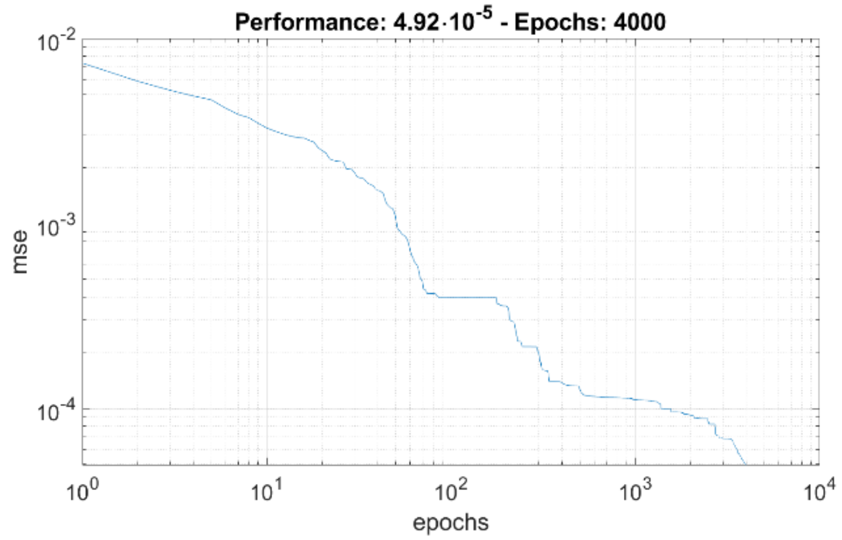

Figure 4: Performance diagram during the training

and relative terms. In some segments it can be noted that the average error appears constant, and in some cases, it slightly increases, but this does not prevent, after a suitable number of iterations, from resuming the decreasing trend. This means that although the mean (squared) error is constant, the iterative process continues to modify the weights, which allows the network to identify regions in the parameter space where it is possible to obtain performance improvements.

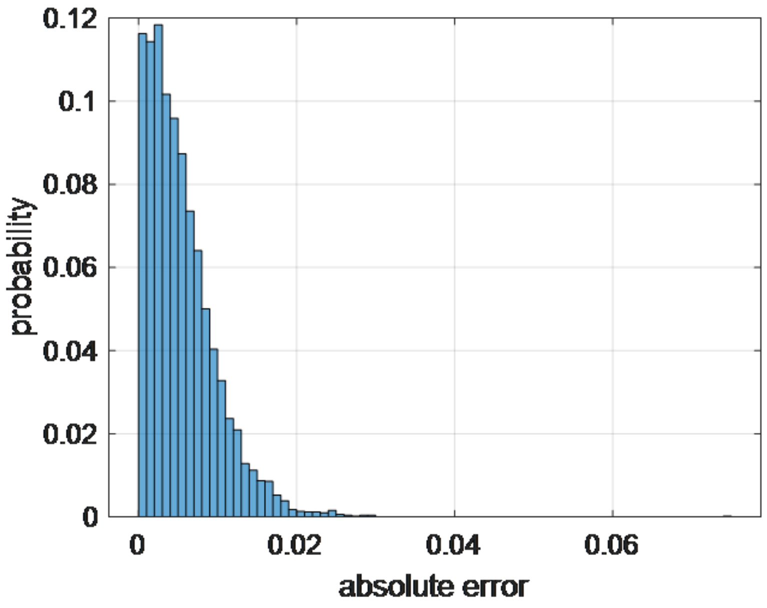

Figure 5: Probability density of the absolute output error

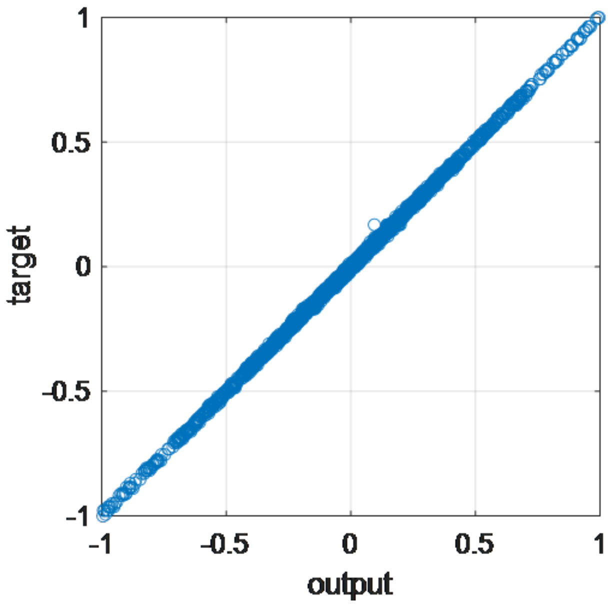

Figure 6: Regression diagram of the training set

FIGURE 5 and FIGURE 6 show the performance of the network. FIGURE 5 shows the probability distribution of the absolute error in the output. It turns out that the majority of the training set has an absolute value below 0.02, namely 1% of the total output range, and only in one case the error is greater than 0.03. The outlier of the distribution can be seen in the regression diagram in FIGURE 6, where it can be noted that in any case it is very close to the regression line. As the aim of the training is the coplanarity of the training set, it is also important to give a measure of such coplanarity by means of the coefficient of multiple correlation:

$$R^2 = 1 - \frac{\sum_i (u_i - NN_i)^2}{\sum_i (u_i - \bar{u})^2} \tag{8}$$

where NN_i is the output of the network corresponding to the i-th example, and $\bar{u}$ is the average of the target. The coefficient in (8) makes sense for a frozen set of weights of the first layer of connections, and it is related to the distribution of points in the product space $\{features\}\times\{output\}$. Furthermore, it is assumed that the second layer of the network corresponds to the regression hyperplane in the same product space. The obtained value of R^2 was 0.9995. As can be deduced from the diagram in FIGURE 4, the performance of the network could be improved by extending the training for a proper number of epochs.

As mentioned earlier, it is possible to redefine the search criterion based on the maximum error in the training set rather than the mean value, but this, like other issues affecting convergence acceleration, is beyond the scope of this paper and will be the subject of future works.

4. SOME IMPLEMENTATION NOTES

The algorithm was developed in Matlab environment. This has led, as far as possible, to exploit matrix calculation, rather than resorting to *for* or *while* loops. This applies especially to the construction, at each iteration, of the matrix of the system coefficients, which has made it possible to enormously contain the time required for a single iteration.

It is also worth making some considerations on the application of the *Line Search*, with which the length of the step to be taken at each iteration is established. The number of samples to explore at each iteration is a macro-parameter that must be set by the designer, and significantly affects the calculation time of the single iteration. In the case under study, the increment obtained as a solution of the linear system is divided into 1000 points distributed with a logarithmic law, to sample finely the function near the current solution. If the minimum of the error corresponds to the first sample, the minimum corresponds to a smaller increment than the minimum foreseen. In this case the search for the minimum is further refined using the bisection method applied to the interval between 0 and the smallest increment. If the minimum falls in correspondence with a null step size, the training procedure stops.

The procedure requires to solve many linear systems. The use of the Matlab library functions in the case where the matrix of the coefficients is ill-conditioned causes difficulties, both in terms of calculation times and instability of the iterative process. On the other hand, the equations system is in turn an approximation, therefore an approximated solution is suitable to continue the iteration. For this reason, the linear systems are solved by means of an approximated iterative method [16], rather than inverting the coefficient matrix. The obtained solution is anyway affected by the ill-conditioning of the coefficient matrix, and this generally results in a less precise solution, but the algorithm is much more stable, and it is able to quickly provide a reasonable solution.

Again, regarding the problem of the ill-conditioned coefficient matrices, it has been observed that many examples of the training set help to mitigate the problem. Increasing the training set does not significantly affect the calculation time, if not indirectly due to memory occupation. It has also been observed that the calculation time of the single iteration does not significantly depend on the number of hidden neurons, therefore it is observed that as the number of hidden neurons increases, the total training time decreases, because the number of epochs necessary to reach the target error is smaller.

As a stopping criterion, a double check was adopted on the number of epochs and on the minimum variation of the error between two successive epochs. In all the tested cases, irrespective of the number of examples, the algorithm stopped when the maximum number of epochs was reached. This means that there was room to further improve the precision of the neural network. In future developments it is planned to use a stopping criterion based on the stabilization of the weights of the connections, rather than on the outputs. This will prevent the procedure from stopping due to a momentary stabilization of the error trend, like the one found in the example in the previous section. If, in fact, it is observed that the weights of the connections continue to change, even if this does not produce any reduction in the average error of the network, the training must still proceed.

Analyzing weight trends during training offers the possibility of accelerating learning. If, in fact, this trend is regular, it is generally possible to extrapolate the values to predict which value it is tending to. This possibility will be the subject of future developments.

5. CONCLUSION

This paper describes a training algorithm for Multi Layer Perceptron neural networks whose main feature is not being subject to local minima, unlike the conventional algorithms, in which the search for the minimum of the loss function is driven by derivatives of the first and second order. Even at a preliminary implementation, the algorithm shows good performance and limited computational complexity as both the training set and the network size increase, which makes it a good candidate for the treatment of big data problems using neural networks. The performance of the algorithm has been tested on a benchmark that is difficult to treat with the most common training algorithms, obtaining very encouraging results. Finally, the paper introduces several lines of development that will be the subject of future publications.

References

[1] Rumelhart DE, Hinton GE, Williams RJ. Learning Representations by Back-Propagating Errors. Nature. 1986;323:533-536.

[2] Cybenko G. Approximation by Superpositions of a Sigmoidal Function. Math Control Signals Syst. 1989;2:303-314.

[3] Carcangiu S, Fanni A, Montisci A. A Constructive Algorithm of Neural Approximation Models for Optimization Problems. COMPEL Int J Comput Math Electr Electron Eng. 2009;28:1276-1289.

[4] Curteanu S, Cartwright H. Neural Networks Applied in Chemistry. I. Determination of the optimal topology of multilayer Perceptron neural networks. J Chemom. 2011;25:527-549.

[5] Delogu R, Fanni A, Montisci A. Geometrical Synthesis of MLP Neural Networks. Neurocomputing. 2008;71:919-930.

[6] Fernandez-Delgado M, Ribeiro J, Cernadas E, Ameneiro SB. Direct Parallel Perceptrons (Dpps): Fast Analytical Calculation of the Parallel Perceptrons Weights With Margin Control for Classification Tasks. IEEE Trans Neural Netw. 2011;22:1837-1848.

[7] Ploj B, Harb R, Zorman M. Border Pairs Method—Constructive MLP Learning Classification Algorithm. Neurocomputing. 2014;126:180-187.

[8] Ploj DB. Optimization for Multi-Layer Perceptron: Without the Gradient. In: Advances in machine learning research. Nova Science Publishers; 2014:65-112.

[9] Marquardt DW. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. J Soc Ind Appl Math. 1963;11:431-441.

[10] LeCun Y, Bengio Y, Hinton G. Deep Learning. Nature. 2015;521:436-444.

[11] Cortes C, Vapnik V. Support-Vector Networks. Mach Learn. 1995;20:273-297.

[12] Jódar L, Law AG, Rezazadeh A, Weston JH, Wu G, et al. Computations for the Moore-Penrose and Other Generalized Inverses. Congr Numerantium. 1991;80:57.

[13] Bazaraa MS, Sherali HD, Shetty CM. Nonlinear Programming: Theory and Algorithms. John Wiley & Sons. 2013.Bazaraa MS, Sherali HD, Shetty CM. Nonlinear Programming: Theory and Algorithms. John Wiley & Sons. 2013.

[14] Schwefel H-P. Numerical Optimization of Computer Models. John Wiley & Sons. 1981.

[15] Carcangiu S, Fanni A, Montisci A. Multi Objective Optimization Algorithm Based on Neural Networks Inversion. In: Bio-inspired systems: computational and ambient intelligence. 2009:744-751.

[16] Cannas B, Carcangiu S, Fanni A, Forcinetti R, Montisci A, et al. Hops: A New Tomographic Reconstruction Algorithm for Non-destructive Acoustic Testing of Concrete Structures. Adv Civ Eng Build Mater. 2012;831:259-263.