Qadir and Smith., Global J Technol Optim 2018, 9:1
DOI: 10.4172/2229-8711.1000223

Global Journal of
Technology & Optimization

Research Article Open Access

Direct Differentiation Based Hessian Formulation for Training Multilayer Feed forward Neural Networks using the LM Algorithm-Performance Comparison with Conventional Jacobian-Based Learning

Najam ul Qadir[1]* and Stephen Montgomery Smith[2]

[1]*Department of Mechanical Engineering, King Fahd University of Petroleum and Minerals, Dhahran, 31261, KSA, Saudi Arabia*
[2]*Department of Mathematics, University of Missouri, Columbia, MO 65211, USA*

Abstract

The Levenberg-Marquardt (LM) algorithm is the most commonly used training algorithm for moderate-sized feed forward artificial neural networks (ANNs) due to its high convergence rate and reasonably good accuracy. It conventionally employs a Jacobian-based approximation to the Hessian matrix, since exact evaluation of the Hessian matrix is generally considered computationally prohibitive. However, the storage of Jacobian matrix in computer memory is itself prone towards memory constraints, especially if the number of patterns in the training data exceeds a critical threshold. This paper presents a first attempt of evaluating the exact Hessian matrix using the direct differentiation approach for training a multilayer feed forward neural network using the LM algorithm. The weights employed for network training are initialized using a random number generator in MATLAB (R2010a). The efficiency of the proposed algorithm has been demonstrated using the well-known 2-spiral and the parity-*N* datasets, and the training performance has been compared with the Neural Network Toolbox in MATLAB (R2010a) which employs the conventional Jacobian-based learning methodology.

Keywords: Jacobian-based approximation; Feed forward; Hessian

Introduction

The LM algorithm is used to solve non-linear least squares optimization problems. It operates by interpolating between the Gauss-Newton (GN) algorithm and the gradient descent method, due to which it is capable of searching the desired optimum even if the initial guess is located far from it. The most well-known application of LM algorithm is the solution of a generalized least-squares curve-fitting problem, involving m empirical datum pairs of independent and dependent variables, to optimize the parameters of the model curve such that the sum of squares represented by equation (1) becomes minimal.

$$S(\beta)=\sum_{i=1}^{m}\left[y_i-f(x_i,\beta)\right]^2 \tag{1}$$

The training procedure of an ANN can also be viewed as a least-squares curve-fitting problem, since the objective function to be minimized is the mean squared error (MSE) between the network-computed matrix at the current training iteration and the target matrix, expressed as where **W** collectively represents the weights assigned to the connections between the network layers, **T** represents the target matrix assigned to the output layer, **P** represents the network-computed matrix at the output layer at the current training iteration, N_{pts} denotes the total number of patterns in the training dataset, and *M* represents the size of each pattern in **T**. A comparison of equations (1) and (2) suggests that the LM algorithm can also be utilized as an efficient training algorithm for ANNs.

$$\mathrm{MSE}(\mathrm{W})=\frac{1}{MN_{pts}}\sum_{m=1}^{N_{pts}}\sum_{n=1}^{M}\left[\mathrm{P}_{mn}(\mathrm{W})-\mathrm{T}_{mn}\right]^2 \tag{2}$$

In mathematics, the gradient is a generalization of the useful concept of the derivative to functions of several variables. If $f(x_1,\ldots,x_n)$ is a differentiable, scalar-valued function of several variables, its gradient is defined as a vector whose components are the *n* partial derivatives of *f*. The gradient of a scalar-valued function is thus a vector-valued function. In vector calculus, the Jacobian matrix or simply the Jacobian is the matrix of all first-order partial derivatives of a vector-valued function. For instance, if $f(x_1,\ldots,x_n)$ is a function which takes real *n*-tuples as input and produces real *m*-tuples as output, it can be considered to be a combination of *m* real-valued component functions, $F_1(x_1,\ldots,x_n),\ldots,F_m(x_1,\ldots,x_n)$. The partial derivatives of all these functions with respect to the variables $x_1,\ldots,x_n$ (if they exist) can be organized in an *m*-by-*n* matrix as of in equation (3).

$$\mathrm{J}(F)=\begin{bmatrix}\frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \frac{\partial F_m}{\partial x_1} & \cdots & \frac{\partial F_m}{\partial x_n}\end{bmatrix} \tag{3}$$

where, **J** represents the Jacobian of *F*. The Hessian matrix, or simply the Hessian, is a square matrix of all second-order partial derivatives of a function. For a given function, $f(x_1,x_2,\ldots,x_n)$, if all second-order partial derivatives of *f* exist and are observed to be continuous over its domain, then the Hessian of *f* can be expressed as of in equation(4).

***Corresponding author:** Najam ul Qadir, Department of Mechanical Engineering, King Fahd University of Petroleum and Minerals, Dhahran, 31261, KSA, Saudi Arabia, Tel: 966-13-860-2540; E-mail: najam_980110@yahoo.com

Received April 28, 2018; **Accepted** March 19, 2018; **Published** March 23, 2018

Citation: Qadir Nu, Smith SM (2018) Direct Differentiation Based Hessian Formulation for Training Multilayer Feed forward Neural Networks using the LM Algorithm-Performance Comparison with Conventional Jacobian-Based Learning. Global J Technol Optim 9: 223. doi: 10.4172/2229-8711.1000223

$$\mathrm{H}(f)=\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \quad (4)$$

The LM algorithm conventionally employs a Jacobian-based approximation to the Hessian matrix, since evaluation of its exact form has not only found to be intensely laborious but also computationally prohibitive [1-8]. However, in certain practical scenarios where training accuracy is prioritized over computational expense or the training data comprises an unusually large number of patterns, one might have to resort to the use of exact Hessian, since construction of the Jacobian itself incorporates the total number of patterns in the training data. It is due to this reason that the formulation of exact Hessian has still been the focus of a large number of research groups dedicated towards developing increasingly accurate training algorithms for considerably large-sized networks [9-14]. Bishop [9] reported an extended back-propagation algorithm, which allows all components of the Hessian matrix to be evaluated exactly for a feed forward network of an arbitrary topology. It was shown that the components of the Hessian matrix can be evaluated exactly using multiple forward propagation through the network, followed by multiple backward propagation. Piche [10] developed equations for the exact calculation of the second derivative of an error function with respect to (*w.r.t*) the weights of a back-propagation neural network. It was shown that for a recurrent network with a sufficiently small number of free parameters, it might be reasonable to explicitly compute the second derivative. Pearlmutter [11] proposed a numerically accurate, time-efficient, and flexible technique to directly compute the product of Hessian with a vector. However, practical usefulness of the technique for updating the weights during network training was observed to depend on reasonably unbiased estimates of the gradient vector. Rossi [12] proposed three different methods for evaluating the components of the exact Hessian matrix for an arbitrary feed forward neural network – the direct method, the mixed method, and the back-propagation method. However, none of the proposed methods is found to yield complete mathematical expressions of all the second-order components of the Hessian matrix for training arbitrary feed forward neural networks using a given training algorithm. In contrast, Toh et al. [13] presented explicit vector and matrix canonical forms for the Jacobian and Hessian and an iterative training algorithm for a feed forward neural network with a single hidden layer and a single output unit. Hence, the algorithm could not be utilized for solving problems involving more than one unit in the output layer. Guimarães and Ramírez [14] proposed the use of sub-matrices for exact computation and assemblage of the Hessian matrix for training a multilayer perceptron (MLP). However, the total number of function evaluations required for a single estimation of the Hessian matrix was expected to result in critically lower computational speed during the actual training procedure. Wilamowski and Yu [15] addressed the problem of Jacobian matrix storage and multiplication via direct computation of a symmetric quasi-Hessian matrix and a gradient vector. The training speed was observed to improve significantly; however, the level of achievable network precision was still compromised owing to the differences existing between the components of the quasi-Hessian and the corresponding exact Hessian matrix.

The research studies conducted so far as mentioned above are generally based on exact Hessian evaluation using the indirect differentiation approach for small and moderate-sized networks with reasonably good accuracy. To the best knowledge of the authors, this study presents the first attempt of utilizing the direct differentiation approach for evaluating the components of the exact Hessian matrix for training a multilayer feed forward neural network with a single hidden layer using the LM algorithm. The initial weights for network training are obtained using a random number generator in MATLAB (R2010a). The paper is organized as follows. The following section presents the formulation of the exact gradient vector and the exact Hessian matrix for network training using the LM algorithm. The subsequent section will then demonstrate the training efficiency of the proposed LM algorithm for the solution of the well-known 2-spiral problem and the parity-*N* problems. The training performance of the proposed algorithm has been compared with the Neural Network Toolbox in MATLAB (R2010a) which is based on the conventional Jacobian-based learning approach discussed above. The objective of the study is to present the proposed form of the LM algorithm as a novel means to train feed forward neural networks on unusually large-sized data sets with sufficiently high training speed and desirable level of accuracy for which the conventionally employed Jacobian-based approximations cannot be considered reliable owing to memory constraints and lack of network precision.

Mathematical Formulation

The Levenberg-marquardt algorithm

The weight update during the training process using the LM algorithm can be expressed as equation (5).

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha_i^{opt}\,\mathbf{S}_i \quad (5)$$

where i^{th} represents the vector containing all the weights and biases used for the network at the i^{th} training iteration, while $\mathbf{S}_i$ and $\mathbf{S}_i$ represent the optimal step-size and the search direction at the $\mathbf{S}_i$ training iteration respectively. For the exact form of LM algorithm presented in this paper, $\mathbf{S}_i$ can be expressed as equation (6).

$$\mathbf{S}_i = \left(\mathbf{H}_i + \mu_i \delta\right)^{-1} \mathbf{G}_i \quad (6)$$

where $\mathbf{H}_i$ and i^{th} represent the exact Hessian matrix and the exact gradient vector at the i^{th} training iteration, α_i^{opt} is the identity tensor, and α^{op} is the learning rate. For a general minimization scheme of a function $f(\mathbf{X})$, the optimal step-size α^{opt} can be expressed as equation (7).

$$\alpha^{opt} = \underset{\alpha}{\mathrm{Min}}\left[f(\mathbf{X}+\alpha\mathbf{S})\right] \quad (7)$$

In an actual training procedure of a MLP, the function f in equation (7) will represent the MSE at the $\mathbf{w}_i$ training iteration, while $\mathbf{X}$ will be replaced by $\mathbf{w}_i$. A typical entry of the network-computed matrix for a feed forward neural network with a single hidden layer can be expressed as equation (8)

$$\mathbf{P}_{mn} = f_2\left\{\sum_{j=1}^{N_H}\left[\mathbf{W}_{2,nj} f_1\left(\sum_{k=1}^{N}\mathbf{W}_{1,jk}\mathbf{I}_{mk} + \mathbf{b}_{1,j}\right)\right] + \mathbf{b}_{2,n}\right\} \quad (8)$$

$n = 1,\ldots,M,\ m = 1,\ldots,N_{pts}$

where f_1 is the activation function applied on input to each neuron in the hidden layer, f_2 is the activation function applied on input to each neuron in the output layer, $\mathbf{P}$ represents the $(N_{pts}\times M)$ network-computed matrix at each training iteration (M being the size of each pattern in $\mathbf{P}$), $\mathbf{I}$ represents the $(N_{pts}\times N)$ input matrix (N being the size of each pattern in $\mathbf{I}$), N_{pts} denotes the total number of patterns in each

of **P** and **I**, $\mathbf{W}_1$ is the number of neurons included in the hidden layer (hidden neurons), $\mathbf{W}_1$ represents the $(N_H \times N)$ matrix consisting of weights connecting the input layer to the hidden layer, $\mathbf{b}_1$ represents the $(N_H \times 1)$ vector consisting of biases applied to each of the hidden neurons, $(M \times N_H)$ represents the $(M \times N_H)$ matrix consisting of weights connecting the hidden layer to the output layer, and $(M \times 1)$ represents the $(M \times 1)$ vector consisting of biases applied to each of the neurons in the output layer. For this study, we have selected f_1 and f_2 to be the hyperbolic tangent and the pure linear functions respectively, *i.e.*, $f_1(x) = \tanh(x)$ and $f_2(x) = x$ in equation (9).

The r^{th} row of the Jacobian matrix, at a given training iteration, can be expressed as

$$\mathbf{J}_r = \left[\frac{\partial \mathbf{e}_r}{\partial \mathbf{W}_1} \quad \frac{\partial \mathbf{e}_r}{\partial \mathbf{W}_2} \quad \frac{\partial \mathbf{e}_r}{\partial \mathbf{b}_1} \quad \frac{\partial \mathbf{e}_r}{\partial \mathbf{b}_2} \right] \tag{9}$$

where $r = 1,\ldots, MN_{pts}$ and **J** is a $\left[MN_{pts} \times (MN_H + N_H N + M + N_H)\right]$ matrix. Each of the derivatives included in equation (9) can be expressed in indicial notation as equation (10) to equation (13).

$$\frac{\partial \mathbf{e}_r}{\partial \mathbf{W}_{1,jk}} = \frac{\partial (\mathbf{P}_{mn} - \mathbf{T}_{mn})_r}{\partial \mathbf{W}_{1,jk}} = \mathbf{W}_{2,nj} \mathbf{I}_{mk} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \tag{10}$$

$$\frac{\partial \mathbf{e}_r}{\partial \mathbf{W}_{2,nj}} = \frac{\partial (\mathbf{P}_{mn} - \mathbf{T}_{mn})_r}{\partial \mathbf{W}_{2,nj}} = \tanh\left(\sum_{k=1}^{N} \mathbf{W}_{1,jk} \mathbf{I}_{mk} + \mathbf{b}_{1,j} \right) \tag{11}$$

$$\frac{\partial \mathbf{e}_r}{\partial \mathbf{b}_{1,j}} = \frac{\partial (\mathbf{P}_{mn} - \mathbf{T}_{mn})_r}{\partial \mathbf{b}_{1,j}} = \mathbf{W}_{2,nj} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \tag{12}$$

$$\frac{\partial \mathbf{e}_r}{\partial \mathbf{b}_{2,n}} = \frac{\partial (\mathbf{P}_{mn} - \mathbf{T}_{mn})_r}{\partial \mathbf{b}_{2,n}} = 1 \tag{13}$$

where, **T** represents the $(N_{pts} \times M)$ target matrix. In equations (10)-(13), $j = 1,\ldots, N_H$, $k = 1,\ldots, N$, $n = 1,\ldots, M$, and $m = 1,\ldots, N_{pts}$.

The exact gradient vector, at a given training iteration, can be expressed in equation (14).

$$\mathbf{G}^T = \left[\frac{\partial \mathrm{MSE}}{\partial \mathbf{W}_1} \quad \frac{\partial \mathrm{MSE}}{\partial \mathbf{W}_2} \quad \frac{\partial \mathrm{MSE}}{\partial \mathbf{b}_1} \quad \frac{\partial \mathrm{MSE}}{\partial \mathbf{b}_2} \right] \tag{14}$$

where, **G** is a $\left[(MN_H + N_H N + M + N_H) \times 1\right]$ vector, and $\mathbf{G}^T$ represents the transpose of **G**. Following substitution of equation (8) in (2), each of the derivatives included in equation (14) can be expressed in indicial notation as

$$\frac{\partial \mathrm{MSE}}{\partial \mathbf{W}_{1,jk}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \mathbf{I}_{mk} (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \tag{15}$$

$$\frac{\partial \mathrm{MSE}}{\partial \mathbf{W}_{2,nj}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \tanh\left(\sum_{k=1}^{N} \mathbf{W}_{1,jk} \mathbf{I}_{mk} + \mathbf{b}_{1,j} \right) \tag{16}$$

$$\frac{\partial \mathrm{MSE}}{\partial \mathbf{b}_{1,j}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \tag{17}$$

and

$$\frac{\partial \mathrm{MSE}}{\partial \mathbf{b}_{2,n}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \tag{18}$$

In equations (15)–(18), $j = 1,\ldots, N_H$, $n = 1,\ldots, M$, and $n = 1,\ldots, M$.

The exact Hessian matrix, at a given training iteration, can be expressed as

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_1 \mathbf{W}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_1 \mathbf{W}_2} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_1 \mathbf{b}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_1 \mathbf{b}_2} \\ \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_2 \mathbf{W}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_2 \mathbf{W}_2} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_2 \mathbf{b}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_2 \mathbf{b}_2} \\ \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_1 \mathbf{W}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_1 \mathbf{W}_2} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_1 \mathbf{b}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_1 \mathbf{b}_2} \\ \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_2 \mathbf{W}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_2 \mathbf{W}_2} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_2 \mathbf{b}_1} & \frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_2 \mathbf{b}_2} \end{bmatrix} \tag{19}$$

where, **H** is a square matrix with the total number of rows/columns equal to $(MN_H + N_H N + M + N_H)$. Following substitution of equation (8) in (2), each of the derivatives included in equation (19) can be expressed in indicial notation as

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{1,ab} \mathbf{W}_{1,jk}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \mathbf{I}_{mk} \mathbf{W}_{2,na} \mathbf{I}_{mb} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\}, & \text{if } j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \mathbf{I}_{mk} \mathbf{I}_{mb} \left[\begin{array}{l} \mathbf{W}_{2,na} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\} \\ + 2(\mathbf{P}_{mn} - \mathbf{T}_{mn}) \left\{ \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^3 - \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right\} \end{array} \right], & \text{if } j = a \end{cases} \tag{20}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{1,ab} \mathbf{W}_{2,nj}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,na} \mathbf{I}_{mb} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\}, & \text{if } j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{I}_{mb} \left[\begin{array}{l} \mathbf{W}_{2,na} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\} \\ + (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \end{array} \right], & \text{if } j = a \end{cases} \tag{21}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{2,nj} \mathbf{W}_{1,ak}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,na} \mathbf{I}_{mk} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\}, & \text{if } j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{I}_{ma} \left[\mathbf{W}_{2,na} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) + (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \right] \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\}, & \text{if } j = a \end{cases} \tag{22}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{2,nj} \mathbf{W}_{2,ra}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right), & \text{if } r = n \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{1,ab} \mathbf{b}_{1,j}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \mathbf{W}_{2,na} \mathbf{I}_{mb} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\}, & \text{if } j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \mathbf{I}_{mb} \left[\begin{array}{l} \mathbf{W}_{2,na} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\} \\ + 2(\mathbf{P}_{mn} - \mathbf{T}_{mn}) \left\{ \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^3 - \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right\} \end{array} \right], & \text{if } j = a \end{cases} \tag{24}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{1,ab} \mathbf{b}_{2,n}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,na} \mathbf{I}_{mb} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\} \tag{25}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{2,na} \mathbf{b}_{1,j}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,na} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\}, & \text{if } j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \left[\mathbf{W}_{2,nj} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) + (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \right] \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\}, & \text{if } j = a \end{cases} \tag{26}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{W}_{2,na} \mathbf{b}_{2,r}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right), & \text{if } r = n \\ 0, & \text{otherwise} \end{cases} \tag{27}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{1,a} \mathbf{W}_{1,jk}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \mathbf{W}_{2,na} \mathbf{I}_{mk} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\}, & \text{if } j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \mathbf{I}_{mk} \left[\begin{array}{l} \mathbf{W}_{2,na} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\} \\ + 2(\mathbf{P}_{mn} - \mathbf{T}_{mn}) \left\{ \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^3 - \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right\} \end{array} \right], & \text{if } j = a \end{cases} \tag{28}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{1,a} \mathbf{W}_{2,nj}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,na} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\}, & \text{if } j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \left[\begin{array}{l} \mathbf{W}_{2,na} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al} \mathbf{I}_{ml} + \mathbf{b}_{1,a} \right) \right]^2 \right\} \\ + (\mathbf{P}_{mn} - \mathbf{T}_{mn}) \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \end{array} \right], & \text{if } j = a \end{cases} \tag{29}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{2,n} \mathbf{W}_{1,jk}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,nj} \mathbf{I}_{mk} \left\{ 1 - \left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl} \mathbf{I}_{ml} + \mathbf{b}_{1,j} \right) \right]^2 \right\} \tag{30}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{2,n}\mathbf{W}_{2,rj}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl}\mathbf{I}_{ml} + \mathbf{b}_{1,j}\right), & if \ r = n \\ 0, \ \text{otherwise} \end{cases} \tag{31}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{1,a}\mathbf{b}_{1,j}} = \begin{cases} \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj}\mathbf{W}_{2,na}\left\{1-\left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl}\mathbf{I}_{ml} + \mathbf{b}_{1,j}\right)\right]^2\right\}\left\{1-\left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al}\mathbf{I}_{ml} + \mathbf{b}_{1,a}\right)\right]^2\right\}, & if \ j \neq a \\ \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \sum_{n=1}^{M} \mathbf{W}_{2,nj} \begin{bmatrix} \mathbf{W}_{2,na}\left\{1-\left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl}\mathbf{I}_{ml} + \mathbf{b}_{1,j}\right)\right]^2\right\}\left\{1-\left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al}\mathbf{I}_{ml} + \mathbf{b}_{1,a}\right)\right]^2\right\} \\ +2\left(\mathbf{P}_{mn} - \mathbf{T}_{mn}\right)\left\{\left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl}\mathbf{I}_{ml} + \mathbf{b}_{1,j}\right)\right]^3 - \tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl}\mathbf{I}_{ml} + \mathbf{b}_{1,j}\right)\right\} \end{bmatrix}, & if \ j = a \end{cases} \tag{32}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{1,a}\mathbf{b}_{2,n}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,na}\left\{1-\left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,al}\mathbf{I}_{ml} + \mathbf{b}_{1,a}\right)\right]^2\right\} \tag{33}$$

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{2,n}\mathbf{b}_{1,j}} = \frac{2}{MN_{pts}} \sum_{m=1}^{N_{pts}} \mathbf{W}_{2,nj}\left\{1-\left[\tanh\left(\sum_{l=1}^{N} \mathbf{W}_{1,jl}\mathbf{I}_{ml} + \mathbf{b}_{1,j}\right)\right]^2\right\} \tag{34}$$

and

$$\frac{\partial^2 \mathrm{MSE}}{\partial \mathbf{b}_{2,n}\mathbf{b}_{2,r}} = \begin{cases} \frac{2}{M}, \text{if } r = n \\ 0, \text{otherwise} \end{cases} \tag{35}$$

In equations (20)-(35), $a, j = 1, \ldots, N_H$, $b, k = 1, \ldots, N$, and $n, r = 1, \ldots, M$. For ANN training using the Jacobian-based learning approach, the gradient and Hessian at the i^{th} training iteration can be expressed as

$$\mathbf{G}_i \approx \mathbf{J}_i^T \mathbf{e}_i \tag{36}$$

and

$$\mathbf{H}_i \approx \mathbf{J}_i^T \mathbf{J}_i \tag{37}$$

where, $\mathbf{e}_i$ represents the vector containing the network errors at the i^{th} training iteration.

If we assume that each component of the Jacobian or the exact Hessian matrix consumes R units of available computer memory, the total amount of memory consumed by the Jacobian matrix will be $MN_{pts}(MN_H + N_H N + M + N_H)R$ units, while the total memory consumed by the exact Hessian matrix will be $(MN_H + N_H N + M + N_H)^2 R$ units. However as $N_{pts} \to \infty$, a critical number of patterns will be reached at which the total number of memory units required for Jacobian matrix storage will exceed the total computer memory available, while the exact Hessian matrix will always be occupying the same $(MN_H + N_H N + M + N_H)^2 R$ units for all N_{pts}. Hence, we can conclude that the storage of Jacobian matrix in computer memory is constrained by the total number of patterns included in the training dataset, while the storage of the exact Hessian matrix is not bounded by any such constraint. However, a careful inspection of equations (9)-(13) and (14)-(37) clearly suggests that the proposed algorithm incorporates a significantly larger number of floating point operations required for the computation of the exact Hessian and is thus expected to result in substantially lower training speed than the conventional Jacobian-based approximations thus necessitating the use of high-speed computational resources. A flowchart illustrating the training process using the proposed LM algorithm is shown in Figure 1.

Training Results

The training process using the proposed LM algorithm was conducted on an **Intel** Core i5-2520M **2.5GHz** processor with 4 GB RAM. The initial learning rate was set to be equal to 0.1, and was updated according to equation (38).

$$\mu_{i+1} = \begin{cases} \mu_i - 0.1\mu_i, & \text{if } \mathrm{MSE}_{i+1} < \mathrm{MSE}_i \\ \mu_i + 10\mu_i, & \text{if } \mathrm{MSE}_{i+1} \geq \mathrm{MSE}_i \end{cases} \tag{38}$$

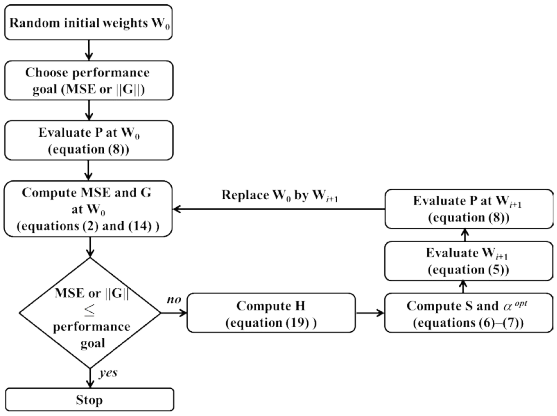

Figure 1: Flowchart of the training process using the proposed LM algorithm.

where MSE_i denotes the value of MSE observed at the i^{th} training iteration. The following sections present a performance comparison of the training results obtained using the proposed algorithm for the solution of the 2-spiral and the parity-*N* problems with the Neural Network Toolbox in MATLAB (R2010a) which is based on the Jacobian-based learning methodology.

2-Spiral problem

The governing equations used for generating the 2-spiral dataset can be expressed in equation (39) [16]:

$$\begin{cases} \mathbf{x}_1(n) = r_n \cdot \sin(\alpha_n) + 0.5 \\ \mathbf{x}_2(n) = r_n \cdot \cos(\alpha_n) + 0.5 \end{cases}$$

where

$$r_n = \begin{cases} \frac{0.4(105-n)}{104}, \text{for one spiral} \\ \frac{0.4(n-105)}{104}, \text{for the other spiral} \end{cases} \tag{39}$$

$$\alpha_n = \frac{\pi(n-1)}{16}$$

$$n = 1, 2, 3, \cdots$$

where $\mathbf{I}_s = [\mathbf{x}_1(n) \quad \mathbf{x}_2(n)]$ represents a typical pattern of the input matrix **P**. The targeted output $\mathbf{T}_s$ equals 1 if the two inputs in a given pattern correspond to a point on one spiral, and -1 if they correspond to a point on the other spiral (or vice versa). Hence the total number of patterns, s, contained in the dataset equals $2n$. As demonstrated by Wan et al., a standard MLP architecture with a single hidden layer requires at least 34 hidden neurons to solve the 2-spiral problem [17]. More recently, it has been shown experimentally by Hunter et al. that the Neuron-by-Neuron (NBN) training algorithm with Fully Connected Cascade (FCC) architecture is the best combination requiring only 7 hidden neurons to solve the 2-spiral problem [18]. For this study, the value of n is chosen to be equal to 500, which corresponds to $s = 1000$. Figure 2 shows a graphical illustration of the 2-spiral dataset based on the selected value of n (Figure 2).

Table 1 presents the training results obtained using the proposed algorithm obtained for the 2-spiral dataset for a range of 2-10 neurons in the hidden layer. For comparison, the results achieved using the NN

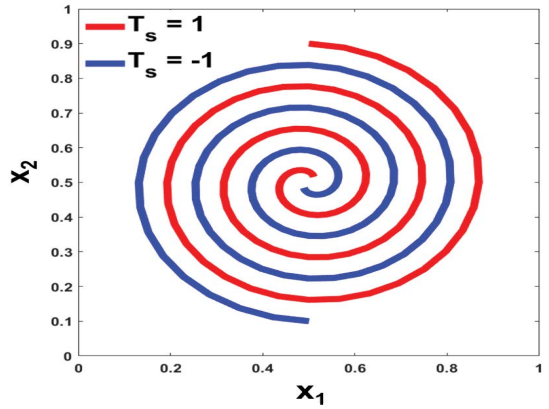

Figure 2: Graphical illustration of the 2-spiral problem dataset (n=500).

N_H	Performance Goal (MSE)	No. of iterations		Average convergence rate *w.r.t* no. of iterations	
		LM using exact G and H	Neural Network Toolbox	LM using exact G and H	Neural Network Toolbox
2	0.99400	196	66	0.0033	0.0190
3	0.99580	250	14	0.0127	0.2596
4	0.99402	239	17	0.0092	0.0074
5	0.99373	250	22	0.0193	0.0371
6	0.99409	247	39	0.0175	0.0714
7	0.98732	250	75	0.0230	0.0120
8	0.98834	250	32	0.0297	0.0448
9	0.98523	250	124	0.0348	0.0062
10	0.98949	249	30	0.0486	0.2570

Table 1: Training results obtained on the 2-spiral dataset for the proposed algorithm and the Neural Network Toolbox in MATLAB (R2010).

Toolbox to reach the same performance goal for a given value of N_H are also reported in Table 1. It can be clearly seen that the NN Toolbox shows a significantly superior performance than the proposed algorithm in solving the 2-spiral problem. More specifically, it was observed that the NN Toolbox required 136% lesser number of iterations on average to reach the same performance goal for 2–10 neurons in the hidden layer, and resulted in a 113% higher average convergence rate *w.r.t* the number of iterations as compared to the proposed algorithm.

The parity-mapping problem is recognized as one of the most popular benchmarks in assessing the capability of multilayer neural networks for solving complex real-world problems with an acceptable level of precision. The N-bit parity function is a mapping defined on 2^N distinct binary vectors that indicate whether the sum of the N components of a binary vector is odd or even. Let $\beta=[\beta_1,\beta_2,\ldots,\beta_N]^T$ be a vector in B^N where $B^N \triangleq \{\beta \in R^N : \beta_k = 0 \text{ or } 1 \text{ for } k=1,2,\ldots,N\}$. The mapping $\psi : B^N \rightarrow B^1$ is then defined as the N-bit parity function if denoted as equation (40).

$$\psi(\beta)=\begin{cases} 1 \text{ if } \sum_{k=1}^{N}\beta_k \text{ is odd} \\ 0 \text{ if } \sum_{k=1}^{N}\beta_k \text{ is even} \end{cases} \quad (40)$$

The parity mapping thus considered noticeably difficult for ANN learning since changes in a single bit was observed to result in significant changes in the output (Table 1).

Table 2 presents a comparison of the training results obtained for the 6-bit parity dataset using the proposed algorithm along with the corresponding results achieved using the NN Toolbox. It have been observed that for 2-10 neurons in the hidden layer, the proposed algorithm required 12% lesser number of iterations on average and a 173% higher average convergence rate *w.r.t* the number of iterations to reach the same performance goal as compared to the NN Toolbox in solving the parity-6 problem. Furthermore, it was observed that the proposed algorithm could reach a feasible solution to the parity-6 problem with a minimum of six neurons in the hidden layer, while the NN Toolbox was observed to require at least 7 hidden neurons.

Table 3 presents a comparison of the training results obtained for the 9-bit parity dataset using the proposed algorithm along with the corresponding results achieved using the NN Toolbox. In contrast to the parity-6 problem discussed above, the proposed algorithm was observed to require 122% greater number of iterations on average than the NN Toolbox to reach the same performance goal for 2-10 neurons in the hidden layer. However, it was observed to result in a 116% higher average convergence rate *w.r.t* the number of iterations than the NN Toolbox for solving the parity-9 problem for 2-10 neurons in the hidden layer. Furthermore, it was observed that the proposed

N_H	Performance Goal (MSE)	No. of iterations		Average convergence rate *w.r.t* no. of iterations	
		LM using exact G and H	Neural Network Toolbox	LM using exact G and H	Neural Network Toolbox
2	0.1547600	177	15	0.0047	0.0490
3	0.1103000	35	100	0.0664	0.0032
4	0.0539710	120	100	0.0042	0.0024
5	0.0626970	78	100	0.0401	0.0090
6	0.0530120	17	100	0.4838	0.0090
7	0.0935120	21	10	0.6251	0.0345
8	0.0682810	31	100	0.4775	0.0071
9	0.0078080	25	100	0.8986	0.0071
10	0.0098395	57	10	0.2085	0.0809

Table 2: Training results obtained on the parity-6 dataset for the proposed algorithm and the Neural Network Toolbox in MATLAB (R2010).

N_H	Performance Goal (MSE)	No. of iterations		Average convergence rate *w.r.t* no. of iterations	
		LM using exact G and H	Neural Network Toolbox	LM using exact G and H	Neural Network Toolbox
2	0.2267200	174	19	0.0244	0.0183
3	0.1825400	200	9	0.0055	0.0331
4	0.0518150	200	37	0.0193	0.0108
5	0.0514260	200	18	0.0231	0.0229
6	0.0050782	161	100	0.0325	0.0074
7	0.0156440	200	33	0.0797	0.0264
8	0.0144570	200	26	0.0604	0.0315
9	0.0215280	81	100	0.3452	0.0068
10	0.0023482	200	112	0.1549	0.0058

Table 3: Training results obtained on the parity-9 dataset for the proposed algorithm and the Neural Network Toolbox in MATLAB (R2010).

algorithm could reach a feasible solution to the parity-9 problem with a minimum of 9 neurons in the hidden layer, while the Neural Network Toolbox was observed to require at least 10 hidden neurons.

Since the proposed algorithm utilizes a completely random initial guess for the network weights, the simplest performance metric, which combines the training accuracy, the convergence rate and the required minimum number of iterations, could be expressed in equation (41) as:

$$\mathbf{M} = w_1 \left(1-G\right) + \frac{w_2}{I} + w_3 \, R_t \qquad (41)$$

where, G is the desired performance goal in terms of MSE with a user-defined weightage w_1, I is the least number of iterations required to reach G with a weightage w_2, and w_3 denotes the average convergence rate *w.r.t* the number of iterations with a weightage w_3.

Figure 3 displays a graphical comparison of the training performances observed for the proposed algorithm and the NN Toolbox for the 2-spiral dataset based on scaled values of **M** for equal weightages assigned to each of the three aforementioned performance parameters $\left(w_1 = w_2 = w_3 = 1/3\right)$. As already observed earlier in Table 1, the NN Toolbox shows a significantly better overall performance in solving the 2-spiral problem as the proposed algorithm for 2–10 neurons used in the hidden layer. The only exceptions lie in the case of 7 and 9 hidden neurons for which the proposed algorithm performs equally good and marginally better than the NN Toolbox, respectively. On average, the NN Toolbox was observed to result in a 111% better training performance than the proposed algorithm in solving the 2-spiral problem (Table 3).

Figure 4 displays a graphical comparison of the training performances observed for the proposed algorithm and the NN Toolbox for the parity-6 dataset based on the values of **M** obtained for equal weightages assigned to each of the three performance parameters. As observed in Figure 4, the NN Toolbox performs equally well as the proposed algorithm for 4 neurons in the hidden layer and shows better performance only for the case of 2 hidden neurons. For 6-9 neurons in the hidden layer, the proposed algorithm can be observed to perform substantially better than the NN Toolbox does. On average, the proposed algorithm was observed to show 25% better training performance then the NN Toolbox in solving the parity-6 problem.

Figure 5 displays a graphical comparison of the training performances observed for the proposed algorithm and the NN Toolbox for the parity-9 dataset based on the values of **M** obtained for equal weightages assigned to each of the three performance parameters. As illustrated in Figure 5, there is no significant difference between

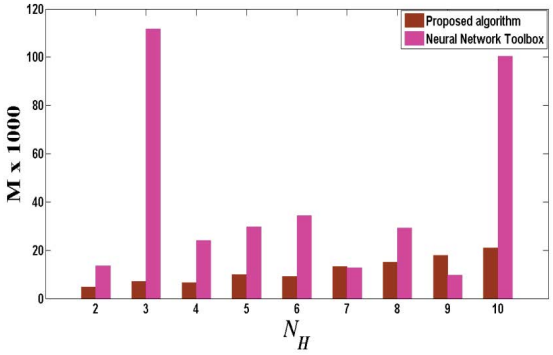

Figure 3: Graphical comparison of the proposed algorithm with the MATLAB Neural Network Toolbox (R2010a) based on scaled values of **M** for the 2-spiral dataset.

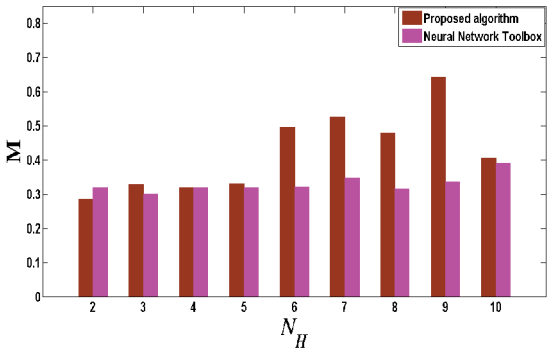

Figure 4: Graphical comparison of the proposed algorithm with the MATLAB Neural Network Toolbox (R2010a) based on values of **M** for the parity-6 dataset.

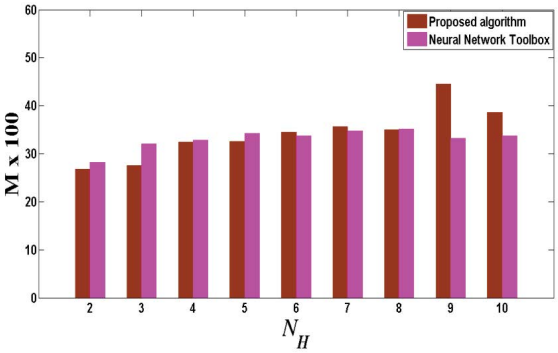

Figure 5: Graphical comparison of the proposed algorithm with the MATLAB Neural Network Toolbox (R2010a) based on scaled values of **M** for the parity-9 dataset.

the training performances observed for the proposed algorithm and the NN Toolbox for the case of 2-8 hidden neurons for solving the parity-9 problem. However, for the case of 9 and 10 hidden neurons, the proposed algorithm can be observed to perform substantially better than the NN Toolbox as shown in Figure 5. For the case of 9 neurons in the hidden layer, and considerably better for the case of 10 neurons. On average, the proposed algorithm was observed to show 3% better training performance then the NN Toolbox in solving the parity-9 problem. Figure 6 shows a graphical comparison of the training performances for the proposed algorithm and the NN Toolbox based on the evolution of MSE for the parity-6 and the parity-9 datasets using 6 and 9 hidden neurons, respectively. As clearly seen, both types of training algorithms are observed to suffer from premature trapping at local minima during the course, which accounts for a significant proportion of the total training duration. More specifically, the NN Toolbox was observed to overcome the saturation barrier only after 400 training iterations while the proposed algorithm was observed to be successful resume rapid convergence in less than 100 iterations for both types of datasets. It has been recently shown by Hunter et al. that the solution of a parity-N problem using a standard MLP architecture with one hidden layer typically requires a minimum of N+1 hidden neurons [18]. As clearly shown in Figure 6, the proposed form of LM algorithm is capable of solving a parity-N problem with a minimum of N hidden neurons in contrast to the NN Toolbox, which has been observed to achieve the solution with a minimum of N+1 hidden neurons in accordance with the findings reported by Hunter et al. [18].

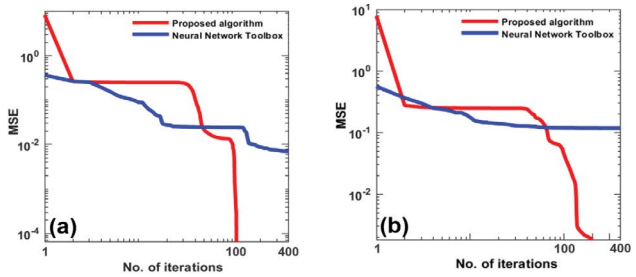

Figure 6: Evolution of MSE with training iterations for **(a)** parity-6 dataset with 6 hidden neurons and **(b)** parity-9 dataset with 9 hidden neurons, for the proposed algorithm and the Neural Network Toolbox (MATLAB R2010a).

Conclusion

The storage of Jacobian matrix in computer memory is prone towards memory constraints, especially if the number of patterns in the training data exceeds a critical threshold. The datasets employed in commercial practices occasionally contain millions of patterns, which the conventional training algorithms might fail to accommodate if the resulting Jacobian matrix is too large to be stored in the available computer memory. Until the advent of online-learning which treats each pattern independently during the course of network training, the only option currently available to train unusually large-sized networks is the usage of exact Hessian for achieving the weights update during the course of network training. This paper presents the first attempt of evaluating the exact Hessian matrix using direct differentiation approach for training a multilayer feed forward network using the LM algorithm. The weights are initialized using a random number generator in MATLAB (R2010a), and are updated using the standard LM algorithm with the Jacobian-approximated gradient and Hessian replaced by their exact counterparts. The training performance of the proposed algorithm has been demonstrated for the well-known 2-spiral and the parity-*N* problems, and the results have been compared with those obtained using the Neural Network Toolbox in MATLAB (R2010a) which has been developed using the Jacobian-based learning methodology. A simple performance metric has been formulated which combines the training accuracy, the average convergence rate and the required minimum number of iterations to reach a pre-defined performance goal. It has been observed that the Neural Network Toolbox performs remarkably better than the proposed algorithm for the solution of the 2-spiral dataset, while the proposed algorithm results in a relatively better training performance than the Neural Network Toolbox for both the parity-6 and the parity-9 datasets. However, the Neural Network Toolbox has been observed to suffer from premature trapping at local minima for both types of parity datasets, which continues indefinitely till the end of network training, while the proposed algorithm has been observed to overcome the trapping barrier in less than a 100 iterations followed by very fast convergence. In contrast to the previously proposed training algorithms, which require at least *N* hidden neurons to solve the parity-*N* problems, the proposed training algorithm has been observed to require a minimum of *N* hidden neurons to reach a feasible solution. The outcome of the study can serve as a basis to design robust algorithms for training multilayer network architectures on highly complex real-world problems with sufficiently high degree of precision and a practically feasible level of training efficiency.

Acknowledgement

The authors are highly thankful to Dr. David Jack (Baylor Univ., TX, U.S.A) and Dr Douglas Smith (Univ. of Missouri, Columbia, MO, U.S.A) in the conceptual phase, and Prof. Habib Abualhamayel (King Fahd Univ. of Petroleum and Minerals, Dhahran, K.S.A) for his computational support during the preparation of the manuscript.

Conflict of Interest

The authors declare that they do not have any mutual conflicts of interest regarding the publication of this paper.

References

1. Hagan MT (1994) Training feedforward networks with the Marquardt Algorithm. IEEE Transactions on Neural Networks 5: 989-993.
2. Suri NNRR, Deodhare D, Nagabhushan P (2002) Parallel Levenberg-Marquardt-based neural network training on Linux clusters-a case study. In: 3rd Indian Conference on Computer Vision, Graphics and Image Processing, Ahmadabad, India.
3. Kermani BG, Schiffman SS, Nagle HT (2005) Performance of the Levenberg-Marquardt neural network training method in electronic nose applications. Sensors and Actuators B Chemical 110: 13-22.
4. Mishra D, Yadav A, Ray S, Kalra PK (2005) Levenberg-Marquardt learning algorithm for integrate-and-fire neuron model. Neural Information Processing - Letters and Reviews 9: 41-51.
5. Suratgar AA, Tavakoli MB, Hoseinabadi A (2007) Modified Levenberg-Marquardt method for neural networks training. World Academy of Science, Engineering and Technology Int J Comp Inform Engineer6: 636-638.
6. Basterrech S, Mohammed S, Rubino G, Soliman M (2011) Levenberg-Marquardt training algorithms for random neural networks. The Computer Journal 54: 125-135.
7. Charles PK, Khan H, Kumar CR, Nikhita N, Roy S, et al. (2011) Artificial neural network based image compression using Levenberg-Marquardt algorithm. Int J Modern Engineer Res 1: 482-489.
8. Scanlan D, Mulvaney D (2013) Graphics processor unit hardware acceleration of Levenberg-Marquardt artificial neural network training, Int J Engineer Sci 2: 1-7.
9. Bishop CM (1992) Exact calculation of the Hessian matrix for the multi-layer perceptron. Neural Computation 4: 494-501.
10. Piche SW (1994) The second derivative of a recurrent network. In: IEEE International Conference on Computational Intelligence, Orlando, FL, USA 1: 250.
11. Pearlmutter BA (1994) Fast exact multiplication by the Hessian. Neural Computation 6: 147-160.
12. Rossi F (1996) Second differentials in arbitrary feed-forward neural networks. In: IEEE International Conference on Neural Networks, Washington DC, USA 1: 418-423.
13. Toh KA, Lu J, Yau WY (2001) Global feed-forward neural network learning for classification and regression. In: International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition, Sophia Antipolis, France 2134: 407-422.
14. Guimarães FG, Ramírez JA (2004) A pruning method for neural networks and Its application for optimization in electromagnetics. In: IEEE Transactions on Magnetics 40: 1160-1163.
15. Wilamowski BM, Yu H (2010) Improved computation for Levenberg–Marquardt training. In: IEEE Transactions on Neural Networks 21: 930-937.
16. Dhar VK, Tickoo AK, Koul R, Dubey BP (2010) Comparative performance of some popular artificial neural network algorithms on benchmark and function approximation problems. Pramana 74: 307-324.
17. Wan S, Banta LE (2006) Parameter incremental learning algorithm for neural networks. In: IEEE Transactions on Neural Networks 17: 1424-1438.
18. Hunter D, Yu H, Pukish MS, Kolbusz J, Wilamowski BM (2012) Selection of proper neural network sizes and architectures – a comparative study. In: IEEE Transactions on Industrial Informatics 8: 228-240.