

# A Comparison of First and Second Order Training Algorithms for Artificial Neural Networks

Syed Muhammad Aqil Burney, Tahseen Ahmed Jilani, Cemal Ardil

**Abstract**— Minimization methods for training feed-forward networks with Backpropagation are compared. Feedforward network training is a special case of functional minimization, where no explicit model of the data is assumed. Therefore due to the high dimensionality of the data, linearization of the training problem through use of orthogonal basis functions is not desirable. The focus is functional minimization on any basis. A number of methods based on local gradient and Hessian matrices are discussed. Modifications of many methods of first and second order training methods are considered. Using share rates data, experimentally it is proved that Conjugate gradient and Quasi Newton's methods outperformed the Gradient Descent methods. In case of the Levenberg-Marquardt algorithm is of special interest in financial forecasting.

**Keywords**— Backpropagation algorithm, conjugacy condition, line search, matrix perturbation

## I. INTRODUCTION

Feed forward neural networks are composed of neurons in which the input layer of neurons is connected to the output layer through one or more layers of intermediate neurons. The training process of neural networks involves adjusting the weights till a desired input/output relationship is obtained [12], [19], [34]. The majority of adaptation learning algorithms are based on the Widrow-Hoffback-algorithm [4]. The mathematical characterization of a multilayer feedforward network is that of a composite application of functions [36]. Each of these functions represents a particular layer and may be specific to individual units in the layer, e.g. all the units in the layer are required to have same activation function. The overall mapping is thus characterized by a composite function relating feedforward network inputs to output. That is

$$\mathbf{O} = \mathbf{f}_{\text{composite}}(\mathbf{x})$$

Using p-mapping layers in a p+1 layer feedforward net yield  $\mathbf{O} = \mathbf{f}^{L_p}(\mathbf{f}^{L_{p-1}}(\dots(\mathbf{f}^{L_1}(\mathbf{x}), \dots)))$ . Thus the interconnection

Manuscript received on December, 03, 2004.

Dr. S. M. Aqil Burney is Professor in the Department of Computer Science, University of Karachi, Pakistan, Phone: 0092-21-9243131 ext. 2447, fax: 0092-21-9243203, [Burney@computer.Org](mailto:Burney@computer.Org), [aqil\\_burney@yahoo.com](mailto:aqil_burney@yahoo.com).

Tahseen Ahmed Jilani is lecturer in the Department of Computer Science, university of Karachi and is Ph.D. research associate in the Department of Statistics, University of Karachi, Pakistan. [tahseenjilani@yahoo.com](mailto:tahseenjilani@yahoo.com)

Cemal Ardil is with the Azerbaijan National Academy of Aviation, Baku, Azerbaijan. [cemalardil@gmail.com](mailto:cemalardil@gmail.com)

weights from unit k in  $L_1$  to unit i in  $L_2$  are  $w^{L_1 \rightarrow L_2}$ . If hidden units have a sigmoidal activation function, denoted  $f^{sig}$

$$\mathbf{O}_i^{L_2} = \sum_{k=1}^{H_1} \mathbf{w}_{ik}^{L_1 \rightarrow L_2} \left\{ \mathbf{f}_k^{sig} \left( \sum_{j=1}^I \mathbf{w}_{kj}^{L_0 \rightarrow L_1} i_j \right) \right\}$$

Above equation illustrates neural network with supervision and composition of non-linear function.

## II. LEARNING IN NEURAL NETWORKS

Learning is a process by which the free Parameters of a neural network are adapted through a continuing process of stimulation by the environment in which the network is embedded [2]. The type of learning is determined by the manner in which the parameters changes take place. A prescribed set of well-defined rules for the solution of a learning problem is called learning algorithm [6]. The Learning algorithms differ from each other in the way in which the adjustment  $\Delta w_{kj}$  to the synaptic weight  $w_{kj}$  is formulated.

The basic approach in learning is to start with an untrained network. The network outcomes are compared with target values that provide some error. Suppose that  $\mathbf{t}_k(n)$  denote some desired outcome (response) for the  $k^{\text{th}}$  neuron at time n and let the actual response of the neuron is  $\mathbf{O}_k(n)$ . Suppose the response  $\mathbf{y}_k(n)$  was produced when  $\mathbf{x}(n)$  applied to the network. If the actual response  $\mathbf{y}_k(n)$  is not same as  $\mathbf{t}_k(n)$ , we may define an error signal as

$$\mathbf{e}_k(n) = \mathbf{t}_k(n) - \mathbf{y}_k(n)$$

The purpose of error-correction learning is to minimize a cost function based on the error signal  $\mathbf{e}_k(n)$ . Once a cost function is selected, error-correction learning is strictly an optimization problem. A cost function usually used in neural networks is mean-square-error criteria called L.M.S learning.

$$\mathbf{J} = E \left[ \frac{1}{2} \sum_k \mathbf{e}_k^2(n) \right] \quad (1)$$

Here summation runs over all neurons in the output layer of the network. This method has the task of continually search for the bottom of cost function in iterative manner Minimization of the cost function  $\mathbf{J}$  with respect to free

parameters of the network leads to so-called Method of Gradient Descent.

$$\mathbf{w}_{k,(n+1)} = \mathbf{w}_{k,(n)} - \eta \nabla \mathbf{J}_{(n)} \quad (2)$$

$\eta$  is called learning rate and it defines the proportion of error for weight updating (correction). The learning parameter has a profound impact on the performance of convergence of learning [21].

A plot of the cost function versus the synaptic weights characterizes the neural network consists of a multidimensional surface called error surface. The neural network consists of cross-correction learning algorithm to start from a n arbitrary point on the error surface (initial weights) and then move towards a global minima, in step-by-step fashion.

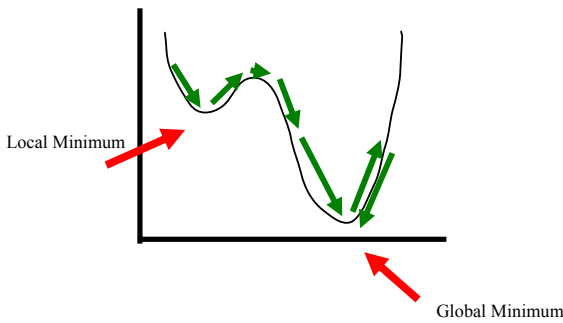


Fig: 1 Quadratic error surface with local and global minima.

We can gain understanding and intuition about the algorithm by studying error surfaces themselves—the function  $\mathbf{J}(\mathbf{w})$ . Such an error surface depends upon the task in hand, but even there are some general properties of error surfaces that seem to hold over a broad range of real-world pattern recognition problems. In case of non-linear neural network, the error surface may have troughs, valleys, canyons, and host of shapes, where as in low dimensional data, contains many minima and so many local minima plague the error landscapes, then it is unlikely that the network will find the global minimum. Another issue is the presence of plateaus—regions where the error varies only slightly as a function of weights see [9], [15] and. Thus in presence of many plateaus, training will get slow. To overcome this situation momentum is introduced that forces the iterative process to cross saddle points and small landscapes [16]. Neural network training begins with small weights; the error surface in the neighborhood of  $\mathbf{w} \approx 0$  will determine the general direction of descent. High dimensional space may afford more ways (dimensions) for the system to ‘get around’ a barrier or local maximum during training. For large networks, the error varies quite gradually as a single weight is changed.

### III. BACKPROPAGATION

For networks having differentiable activation functions, there exist a powerful and computationally efficient method, called error backpropagation for finding the derivatives of an error

function with respect to the weights and biases in the network. Gradient Descent algorithm is the most commonly used error backpropagation method [13], [14].

Standard backpropagation is a gradient descent algorithm, in which the network weights are moved along the negative of the gradient of the performance function. There are a number of variations on the basic algorithm that are based on other standard optimization techniques, such as conjugate gradient [22], [23].

Backpropagation was created by generalizing the Widrow-Hoff learning rule to multiple-layered networks with non-linear differentiable transfer functions.

For If the scalar product of the connections (weights) and the stimulus, we define

$$\mathbf{net} = \mathbf{w}^T \cdot \mathbf{x}$$

is large and have a differentiable activation function. Given a training set of input-target pairs of the form

$$H = \{(x^1, t^1), (x^2, t^2), \dots, (x^n, t^n)\}$$

The basic idea is to compute  $\frac{d(e^p)^2}{dw}$  we use the error correction method to adjust  $\mathbf{w}$  see [8]. Let for the  $k^{\text{th}}$  element of  $\mathbf{w}$ , i.e., we have

correction method to adjust  $\mathbf{w}$  see [8]. Let for the  $k^{\text{th}}$  element of  $\mathbf{w}$ , i.e., we have

$$\begin{aligned} \frac{d(e^p)^2}{dw_k} &= \frac{d(e^p)^2}{dO^p} \cdot \frac{dO^p}{d\mathbf{net}} \cdot \frac{d\mathbf{net}}{dw_k} \\ &= (O^p - t^p) \left[ \frac{df(\mathbf{net}^p)}{d\mathbf{net}^p} \right] \cdot \mathbf{x}_k^p \end{aligned}$$

which is the gradient of the pattern error with respect to weight  $W_k$  and forms the basis of the gradient descent training algorithm. Specifically, we assign the weight correction,  $\Delta w_k$ , such that

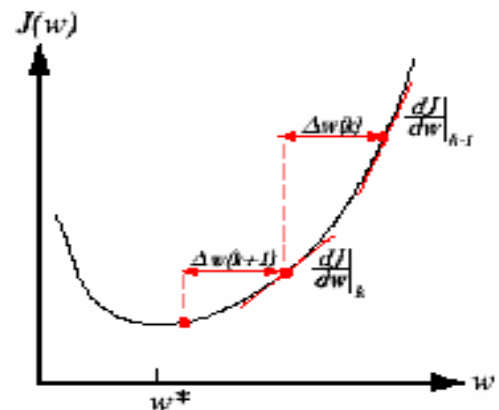


Fig 2 Gradient Descent algorithm showing minimization of cost function

$$\mathbf{w}_k^{j+1} = \mathbf{w}_k^j - \eta \left[ (\mathbf{O}^p - \mathbf{t}^p) \cdot \frac{d \mathbf{f}(\mathbf{net}^p)}{d \mathbf{net}^p} \cdot \mathbf{x}_k^p \right] \quad (3)$$

and for linear activation function

$$\mathbf{w}_k^{j+1} = \mathbf{w}_k^j - \eta \left[ \frac{(\mathbf{O}^p - \mathbf{t}^p)}{-2 \mathbf{e}^p} \cdot \mathbf{x}_k^p \right] \quad \mathbf{w}_k^{j+1} = \mathbf{w}_k^j + \eta \mathbf{e}^p \mathbf{x}_k^p$$

#### IV. LINE SEARCH

Mostly, the learning algorithms involve a sequence of steps through weight space. We can consider each of these steps in two parts. First decide the direction in which to move and secondly, how far to move in that direction. This direction of search provides minimum of the error function in that direction in weight space [18], [25], [27]. This procedure is referred to as line search and it forms the basis for several algorithms which are considerably more powerful than gradient descent. Suppose at step  $n$  in some algorithm the current weight vector is  $\mathbf{w}_n$  and we wish to obtain a particular search direction  $\mathbf{p}_n$  through weight space. The minimum along the search direction then gives the next values for the weight vector as:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \lambda \mathbf{p}_n \quad (4)$$

where the parameter  $\lambda$  is chosen to minimize

$$E(\lambda) = E(\mathbf{w}_n + \lambda \mathbf{p}_n) \quad (5)$$

This gives us an automatic procedure for setting the step length, once we have chosen the search direction. The line search represents a one dimensional minimization problem. This minimization can be performed in a number of ways. A simple approach would be to proceed long the search direction in small steps, evaluating the error function at each step (position), and stop when the error starts to increase. It is possible, however, to find much more efficient approaches. This includes the issue that whether local gradient information is preferable in line search. The search direction is towards the minimum is obtained (possibly) through proper weight adjustments, this involves search through negative direction of gradient information.

#### V. CONJUGATE GRADIENT

To apply line search to the problems of error function minimization, we need to choose a suitable search direction at each stage of the algorithm. Note that at minimum of the line search

$$\frac{\partial}{\partial \lambda} E(\mathbf{w}_n + \lambda \mathbf{p}_n) = 0 \quad (6)$$

$$\text{gives } \mathbf{g}(\mathbf{w}_{n+1})^T \cdot \mathbf{p}_n = 0 \quad \text{Where } \mathbf{g}_n = \frac{\partial \mathbf{E}}{\partial \mathbf{w}_{n+1}} \quad (7)$$

Thus the gradient of the new minimum is orthogonal to the search direction. Choosing successive search directions to the local (negative) gradient directions can lead to the problem of slow learning. The algorithm can then take many steps to

converge, even for a quadratic error function. The solution to this problem lies in choosing the successive search directions  $\mathbf{d}_n$  to minimize cost function such that, at each step of the algorithm, the component of the gradient parallel to the previous search direction which has just been made zero, is unaltered [17, ][24]. Suppose, we have already performed a line minimization along the direction  $\mathbf{d}_n$ , starting from the point  $\mathbf{w}_n$ , to give the new point  $\mathbf{w}_{n+1}$ . Then at the point  $\mathbf{w}_{n+1}$ , we have

$$\mathbf{g}_{n+1}(\mathbf{w}) \cdot \mathbf{p}_n = 0 \quad (8)$$

Now the new search  $\mathbf{d}_n$  such that

$$\mathbf{g}(\mathbf{w}_{n+1} + \lambda \mathbf{p}_{n+1})^T \cdot \mathbf{p}_n = 0$$

gives  $\mathbf{p}_{n+1} \cdot \mathbf{p}_n = 0$  (9)

that is along the new direction, the component of the gradient parallel to the previous search direction remains zero [31].

#### VI. CONJUGATE GRADIENT TECHNIQUES

The conjugate Descent algorithms have been derived on the assumption of a quadratic error function with a positive definite Hessian matrix. For a non-quadratic error function, the Hessian matrix will depend on the current weight vector, and so will need to be re-evaluated at each step of the algorithm. Since the evaluation of Hessian is computationally costly for non-linear neural networks, and since its evaluation would have to be done repeatedly, we would like to avoid having to use the Hessian. In fact, it turns out that the search direction and learning rate can be found with out explicit knowledge of  $\mathbf{H}$ . This leads to the CGA.

The conjugate method avoids this problem by incorporating an intricate relationship between direction and gradient vector. The conjugate gradient method is guaranteed to locate the minimum of any quadratic function of  $N$  variables in at most  $N$  steps. But for non-quadratic function, like as the cost function in M.L.P, the process is iterative rather than  $N$ -step, and a criterion for convergence is required. The weight vector of the network is updated in accordance with the rule in [4]

In conjugate gradient approach, successive steps in  $\Delta \mathbf{w}_n$  are chosen such that weight correction at previous step is preserved. A direction for  $y$ th weight correction is computed and line minimization in this direction is performed, generating  $\mathbf{w}_{n+1}$ .

Successive weight corrections are constrained to be conjugate to those used previously. Interestingly, this is achieved without inversion of Hessian matrix, when compared with Newton's method. There exists several techniques, most often gradient is involved. Suppose initial weight correction is  $\mathbf{p}(0) = -\mathbf{g}(0)$ . Line minimization in the direction  $\mathbf{p}(0)$  takes weight vector  $\mathbf{w}(0)$ . Beginning with  $K = 0$ , we require

subsequent gradient to be orthogonal to the previous weight correction direction; that is from [8]

$$\mathbf{p}_n^T \cdot \mathbf{g}_{n+i} = 0 \quad ; i = 1, 2, \dots, n$$

$$\mathbf{p}_n [\mathbf{g}_{n+2} - \mathbf{g}_{n+1}] = 0 \quad (10)$$

Where  $\mathbf{g}_{n+2} - \mathbf{g}_{n+1}$  is the change of gradient of  $\mathbf{E}$  from  $\mathbf{w}_{n+1}$  to  $\mathbf{w}_{n+2}$ .

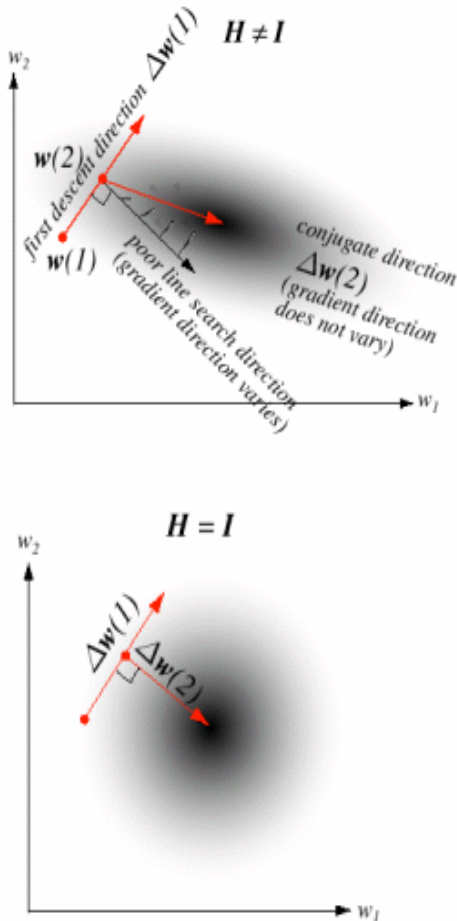


Fig. 3 Conjugal gradient descent in weight space employs a sequence of line searches. If  $\Delta \mathbf{w}_1$  is the first descent direction, the second direction obeys conjugacy condition; as such, the second descent does not "spoil" the contribution due to the previous line search. In the case where the Hessian is diagonal (right), the direction of the line searches are orthogonal [12].

Using Taylor's series, we have

$$\mathbf{E}(\mathbf{w}) \approx \mathbf{E}(\mathbf{w}_0) + \frac{\partial \mathbf{E}}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_0} (\mathbf{w} - \mathbf{w}_0) + \frac{1}{2!} (\mathbf{w} - \mathbf{w}_0)^T \frac{\partial^2 \mathbf{E}}{\partial^2 \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_0} (\mathbf{w} - \mathbf{w}_0)$$

$$\text{where } \mathbf{J} = \frac{\partial \mathbf{E}}{\partial \mathbf{w}} \text{ and } \mathbf{H} = \frac{\partial^2 \mathbf{E}}{\partial^2 \mathbf{w}} = \left[ \frac{\partial^2 \mathbf{E}}{\partial w_i \partial w_j} \right]$$

Differentiating with respect to  $\mathbf{w}$ , we have

$$\mathbf{g}(n) = \mathbf{F} + \mathbf{H}(\mathbf{w} - \mathbf{w}_0)$$

$$\text{Therefore } \mathbf{p}_n^T \mathbf{H} \mathbf{p}_{n+1} = 0 \quad (11)$$

which is called the conjugacy condition, see fig. 3. Weight correction direction  $\mathbf{p}_{n+2}$  and  $\mathbf{p}_{n+1}$  are said to be conjugate to one another in the context of  $\mathbf{H}$ , or  $\mathbf{H}$ -conjugate. It is important to note that there exist several methods of finding conjugate vector  $\mathbf{p}(n+1)$  see [29] and [31]. Each search direction vector is then computed as a linear combination of the current gradient vector and the previous direction vector.

$$\mathbf{p}_{n+1} = -\mathbf{g}_{n+1} + \beta_{n+1} \cdot \mathbf{p}_n \quad (12)$$

defines a path or sequence of search directions  $\mathbf{p}_n$ ;  $n = 1, 2, \dots, n$  in the weight space where  $\beta_n$  is a time varying parameter. There are various rules for determining  $\beta_n$  in terms of the gradient vector  $\mathbf{g}_n$  and  $\mathbf{g}_{n+1}$ . Each weight correction  $\mathbf{p}_n$  is computed sequentially with  $\mathbf{p}_0$  and is formed as the sum of the current negative gradient direction and a scaled version (*i.e.*,  $\beta_n$ ) of previous correction. Using (11) and (12), we have

$$\mathbf{p}_n^T \mathbf{H} \mathbf{p}_{n+1} = \mathbf{p}_n^T \mathbf{H} [-\mathbf{g}_{n+1} + \beta_{n+1} \mathbf{p}_n]$$

$$\beta_{n+1} = \frac{\mathbf{p}_n^T \mathbf{H} \mathbf{g}_{n+1}}{\mathbf{p}_n^T \mathbf{H} \mathbf{p}_n} \quad (13)$$

In most of the conjugate gradient algorithms, the step size is adjusted at each iteration, and a search is made along the conjugate gradient direction to determine the step size which minimizes the performance function along that line [10]. There are a number of different search functions. Some of these are,

$$\text{Fletcher-Reeves } \beta_{n+1} = \frac{\mathbf{g}_{n+1}^T \cdot \mathbf{g}_{n+1}}{\mathbf{g}_n^T \cdot \mathbf{g}_n} \quad (14)$$

$$\text{Polak-Ribiere } \beta_{n+1} = \frac{\mathbf{g}_{n+1}^T \cdot (\mathbf{g}_{n+1} - \mathbf{g}_n)}{\mathbf{g}_n^T \cdot \mathbf{g}_n} \quad (15)$$

$$\text{Powell-Beale Restarts } \beta_{n+1} = \frac{\mathbf{g}_{n+1}^T \cdot (\mathbf{g}_{n+1} - \mathbf{g}_n)}{\mathbf{p}_n^T \cdot (\mathbf{g}_{n+1} - \mathbf{g}_n)} \quad (16)$$

The conjugate gradient algorithms are usually much faster than variable learning rate backpropagation algorithm, although the results will vary from one problem to another. The conjugate Gradient algorithms required only a little more storage than the simple algorithm, so that are often a good choice for networks with a large number of weights. It is important to note that above three expressions are equivalent provided the error function is exactly quadratic. For non-quadratic error functions, the Polak-Ribiere form is generally found to give slightly better results than the other expressions, probably due to the fact that, if the algorithm is making little progress, so that successive gradient vectors are very similar, the Polak-Ribiere form gives a small value for  $\beta_j$  so that the

search direction  $\mathbf{p}_{n+1}$  tends to be reset to the negative gradient direction, which is equivalent to restarting the conjugate gradient procedure. For a quadratic error function the conjugate gradient algorithm finds the minimum after at most  $\mathbf{W}$  line minimizations, without calculating the Hessian matrix [39]. This clearly represents a significant improvement on the simple gradient descent approach which could take a very large number of steps to minimize even a quadratic error function.

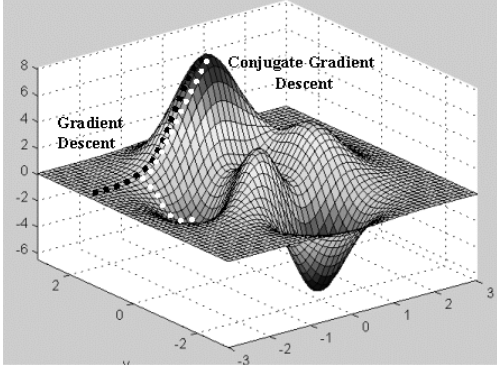


Fig. 4 A comparison between GDA and CGA minimization.

## VII. TRUST REGION MODELS

A large class of methods for unconstrained optimization is based on a slightly different model algorithm. In particular, the step size  $\alpha_j$  is nearly always taken as unity. In order to ensure that the descent condition holds, it may thus be necessary to compute several trial vectors. Methods of this type are often termed as trust-region methods.

For large values of  $\lambda_n$  the step size becomes small. Techniques such as this are well in standard optimization theory, where they are called model trust region methods [35], [37] and [38] because the model is effectively trusted only in a small region around the current search point. The size of the trust region is governed by the parameter  $\lambda_n$ , so that the large  $\lambda_n$  gives the small trust region see [33].

## VIII. QUASI-NEWTON METHODS

Using the local quadratic approximation, we can obtain directly an expression for the location of the minimum or more generally the stationary point of the error function

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

gives  $\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1} \mathbf{g}$

The vector  $-\mathbf{H}^{-1} \mathbf{g}$  is known as the Newton step or the Newton's direction [20],[29]. Unlike the local gradient vector,

the Newton location for a quadratic error surface, evaluated at any  $\mathbf{w}$ , points directly at the minimum of the error function provided  $\mathbf{H}$  is positive definite. Since the quadratic approximation in above algorithm is not exact, it would be necessary to apply  $\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1} \mathbf{g}$  iteratively with the Hessian being re-evaluated at each new search [40]. The Hessian for non-quadratic networks is computationally demanding since it required  $O(N\mathbf{w}^2)$  steps, where  $\mathbf{w}$  is the number of weights in the network and  $N$  is the number of patterns in the data and  $O(\mathbf{w}^3)$  operation for inversion in each iteration [5], [32]. The Newton's step may move towards a maximum or a saddle point rather than a minimum. This occurs if the Hessian is not positive definite. Thus the error is not guaranteed to be reduced at each iteration see [6], [11].

One of the drawbacks of Newton's method is that it requires the analytical derivative of Hessian matrix at each iteration. This is a problem if the derivative is very expensive or difficult to compute. In such cases it may be convenient to iterate according to

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \mathbf{G}^{-1} \mathbf{g}_n \quad (17)$$

where  $\mathbf{G}$  is an easily computed approximation to  $\mathbf{H}$ . For example, in one dimension, the secant method approximates the derivative with the difference quotient

$$a(n) = \frac{\mathbf{g}(\mathbf{w}_{n+1}) - \mathbf{g}(\mathbf{w}_n)}{\mathbf{w}_{n+1} - \mathbf{w}_n}$$

such an iteration is called a quasi-Newton method. If  $\mathbf{G}$  is positive definite, as it usually is, an alternative name is variable metric method. One positive advantage to using an approximation in place of  $\mathbf{H}$  is that  $\mathbf{G}$  can be chosen to be positive definite, ensuring that the step will not be attracted to a maximum of  $f$  when one wants a minimum. Another advantage is that  $\mathbf{G}(n)^{-1} \mathbf{g}(\mathbf{w}_n)$  is a descent direction from  $\mathbf{w}_n$  allowing the use of line searches.

Among general purpose quasi-Newton algorithms, the best is probably the Broydon-Fletcher-Goldfarb-Shanno (BFGS) algorithm. The BFGS algorithm builds upon the earlier and similar Davidon-Fletcher-Powell (DFP) algorithm. The BFGS algorithm starts with a positive definite matrix approximation to  $\mathbf{H}(\mathbf{w}_0)$  usually the identity matrix  $\mathbf{I}$ . At each iteration it makes a minimalist modification to gradually approximate  $\mathbf{H}^{-1}$ .

## IX. LEVENBERG-MARQUARDT ALGORITHM

Levenberg-Marquardt is a trust region based method with hyper-spherical trust region. This method works extremely well in practice, and is considered the most efficient algorithm for training median sized artificial neural networks. Like Quasi-Newton methods, the Levenberg-Marquardt algorithm was designed to approach second order training speed without having to compute Hessian matrix [1]. When the performance

function has the form of a sum of squares, then from (17) we have

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \mathbf{H}^{-1} \nabla F(\mathbf{w}) \quad (18)$$

here the Hessian matrix can be approximated as,  $\mathbf{H} \approx \mathbf{J}^T \mathbf{J}$

and the gradient can be computed as  $\mathbf{g} = \mathbf{J}^T \mathbf{e}$

where the Jacobian matrix  $\mathbf{J}$  contains the first derivatives of the network errors with respect to weights and biases, and  $\mathbf{e}$  is a vector of network errors. The Gauss-Newton update formula can be

$$\mathbf{W}_{n+1} = \mathbf{W}_n - (\mathbf{J}_n^T \mathbf{J}_n)^{-1} \mathbf{J}_n^T \mathbf{e}_n$$

Where  $(\mathbf{J}^T \mathbf{J})$  is positive definite, but if it is not, then, we make some perturbation into it that will control the probability of being non positive definite. Such that the recursion equation is

$$\mathbf{W}_{n+1} = \mathbf{W}_n - (\mathbf{J}_n^T \mathbf{J}_n + \lambda \mathbf{I})^{-1} \mathbf{J}_n^T \mathbf{e}_n \quad (19)$$

Where in neural computing context the quantity  $\lambda$  is called the learning parameter, it ensures that  $\mathbf{J}^T \mathbf{J}$  is positive definite. It is always possible to choose  $\lambda$  sufficiently large enough to ensure a descent step. The learning parameter is decreased as the iterative process approaches to a minimum. Thus, in Levenberg-Marquardt optimization, inversion of square matrix  $\mathbf{J}^T \mathbf{J} + \lambda \mathbf{I}$  is involved which requires large memory space to store the Jacobian matrix and the approximated

## X. NETWORK TRAINING RESULTS

Daily Shares rates data is considered and presented in concurrent form. For programming in C/C++, we used the approach of [3], [7], [26] and [30]. The average number of iterations to converge to the targets are indicated with each algorithm

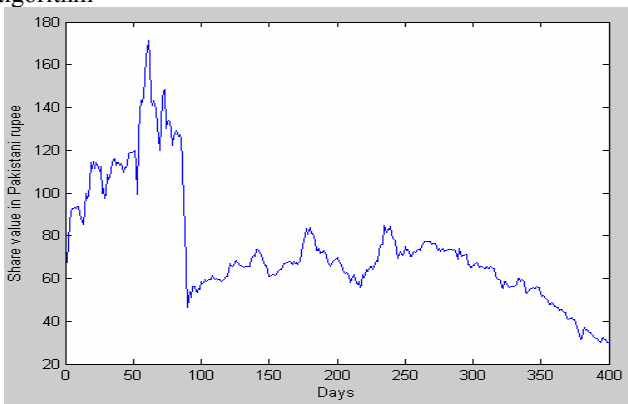


Fig. 5. Time series plot of share rates

We trained the network for each method and average of 50 successful results are taken and presented in the table.

Gradient descent method with momentum and learning rate is considered. Finally, the results of forecasting for 10 days are shown in the table. It is best to use the forecast for only one day due to possibility of error growth. The performance goal is  $10^{-5}$ .

Table: 1

A comparison of neural network training algorithms (in m iterations)						
Target Values	GD 2500 0	BGFS 45	L.M 21	CGFP 400	CGPR 240	SCG 1700
15.95	15.92	15.95	15.95	15.95	15.95	15.95
16.05	15.92	16.05	16.05	16.05	16.05	16.05
15.95	19.93	15.95	15.95	15.95	15.95	15.95
15.80	15.77	15.80	15.80	15.80	15.80	15.80
15.35	15.44	15.35	15.35	15.35	15.35	15.35
15.45	15.58	15.45	15.45	15.45	15.45	15.45
15.55	15.65	15.55	15.55	15.56	15.55	15.55
15.60	15.65	15.60	15.60	15.60	15.60	15.60
15.75	15.77	15.75	15.75	15.75	15.75	15.75
15.80	15.78	15.80	15.80	15.80	15.80	15.80

## REFERENCES

- [1] Aqil Burney S.M., Jilani A. Tahseen and Cemal Aril, 2004. Levenberg-Marquardt algorithm for Karachi Stock Exchange share rates forecasting. International Journal of Computational Intelligence, pp 168-173.
- [2] Aqil Burney S.M., Jilani A. Tahseen, 2002, Time Series forecasting using artificial neural network methods for Karachi Stock Exchange. A Project at Department of Computer Science, University of Karachi.
- [3] Aqil Burney S.M. Measures of central tendency in News-MCB stock exchange. Price index pp 12-31. Developed by Vital information services (pvt) limited, Karachi. C/o Department of Computer Science, University of Karachi.
- [4] B. Widrow and M. A. Lehr, 1990. 30 years of adaptive neural networks: Perceptron, Madaline, and Backpropagation. IEEE Proceedings-78, pp. 1415-1442.
- [5] Barak A. Pearlmutter, 1993. Fast Exact Multiplication by the Hessian. Siemens Corporate Research, Neural Computation.
- [6] Battiti, 1992. First and second-order methods for learning: between steepest descent and Newton's method. Neural Computation 4, pp. 141-166.
- [7] Blum, Adam, 1992. Neural Network in C++. John Wiley and Sons, New York.
- [8] Bortoletti, A., Di Fiore, C., Aselli, S. and Bellini, P. A new class of quasi-Newtonian methods for optimal learning in MLP-networks, IEEE Transactions on Neural Networks, vol: 14, Issue: 2.
- [9] Duda, R. O. Hart, P. E. and Stork, D. G. 2001. Pattern Classification, John Wiley & Sons.
- [10] C. M. Bishop, 1995. Neural networks for pattern recognition. Clarendon Press.
- [11] Chauvin, Y., & Rumelhart, D. 1995. Backpropagation: theory, architecture, and applications. Lawrence Erlbaum Association.
- [12] Chung-Ming Kuan and Tung Liu. 1995. Forecasting exchange rates using feedforward and recurrent neural networks. Journal of Applied Econometrics, pp. 347-64.
- [13] De Villiers and E. Barnard 1993, Backpropagation neural nets with one and two hidden layers, IEEE Transaction on Neural Networks 4, pp. 136-141
- [14] E. Rumelhart, G. E. Hinton, and R. J. Williams, 1986. Learning internal representations by error propagation, Parallel distributed processing. Vol. I, MIT Press, Cambridge, MA, pp. 318-362.
- [15] Ergezinger and E. Thomsen. 1995. An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer. IEEE Transaction on Neural Networks 6, pp. 31-42.
- [16] F. Molar, 1997. Efficient Training of feedforward Neural Networks. Ph.D thesis, Computer Science Department, Aarhus University
- [17] F. Møllar, 1993. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. Neural Networks, 6(4): pp. 525-533.

- [18] Gill, P. E., Murray, W. and Wright, M. H. 1993. *Practical Optimization*, Academic Press.
- [19] Ibeling Kaastra and Milton S. Boyd, 1995. Forecasting future trading volume using neural networks. *Journal of Future Markets*, pp. 953-70.
- [20] J. Shepherd, 1997. Second-order optimization methods. Second-order methods for neural networks. Springer-Verlag, Berlin, New York, pp. 43-72.
- [21] Jacobs, 1988. Increased rate of convergence through learning rate adaptation. *Neural Networks 1*, pp.295-307.
- [22] Hornik, M. Stinchcombe, and H. White, 1989. Multi-layer feedforward networks are universal approximators. *Neural Networks 2*, pp. 359-366.
- [23] Karayiannis and A. N. Venetsanopoulos, 1993. *Artificial neural networks: learning algorithms, performance evaluation, and applications*. Kluwer Academic, Boston, MA.
- [24] M. R. Hestenes and E. Stifle, 1952. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards-49*, pp. 409-436.
- [25] Michael Husken, Jens E. Gayko and Bernard Sendoff, 2000. Optimization for Problem Classes-Neural Networks that Learn to Learn. *IEEE Symposium of Evolutionary computation and Neural networks (ECNN-2000)*, pages 98-109, IEEE Press 2000.
- [26] Mir F. Atiya, Suzan M. El-Shoura, Samir I. Shaken, 1999. A comparison between neural network forecasting techniques- case study: river flow forecasting". *IEEE Transactions on Neural Networks*. Vol. 10, No. 2.
- [27] N. N. Schraudolph, Thore Grapple, 2003. Combining Conjugate Direction Methods with Stochastic Approximation of Gradients. *Proceedings of the Ninth International Workshop on Artificial Intelligence and Intelligence*.
- [28] N. N. Schraudolph, 2002. Fast curvature matrix-vector product for second-order gradient descent. *Neural Computation*, 14(7), pp. 1723-1728.
- [29] N. N. Schraudolph, 1999. Local gain adaptation in stochastic-gradient descent. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, pp. 569-574, Edinburgh, Scotland, 1999. IEE, London.
- [30] *Neural network toolbox*, 2002. : For use with Matlab, MathWorks, Natick, MA.
- [31] Patrick van der Smagt, 1994. Minimization methods for training feedforward neural networks. *Neural Networks 7*(1), pp. 1-11.
- [32] Pearlmutter, 1994. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1), pp.147-160.
- [33] Puha, P. K. H. Daohua Ming, 2003. Parallel nonlinear optimization techniques for training neural networks. *IEEE Transactions on Neural Networks*, vol: 14, Issue: 6, pages, pp. 1460-1468.
- [34] Scarselli and A. C. Tso, 1998. Universal approximation using feedforward neural networks: a survey of some existing methods, and some new results. *Neural Networks*, pp. 1537.
- [35] Ripley, B. D. 1994. Neural networks and related methods for classification. *Journal of the Royal Statistician Society*, B 56(3),409-456.
- [36] S. Haykin, *Neural networks*, 1994. A comprehensive foundation, IEEE Press, Piscataway, NJ.
- [37] S.D. Hunt, J. R. Deller, 1995. Selective training of feedforward artificial neural networks using matrix perturbation theory. *Neural networks*, 8(6), pp 931-944.
- [38] Welstead, Stephen T. 1994. *Neural network and fuzzy logic applications in C/C++*. John Wiley and Sons, Inc. N.Y.
- [39] Z. Strakoš and P. Tich' y., 2002. On error estimation in the conjugate gradient method and why it works in finite precision computations. *ETNA 13*, 56-80.
- [40] Zhou and J. Si, 1998. Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency. *IEEE Trans Neural Networks 9*, pp. 448-453.

degree in Mathematics from Strathclyde University, Glasgow with specialization in estimation, modeling and simulation of multivariate Time series models using algorithmic approach with software development.

He is currently professor and approved supervisor in Computer Science and Statistics by the High education Commission Government of Pakistan. He is also the project director of Umair Basha Institute of Information technology (UBIT). He is also member of various higher academic boards of different universities of Pakistan. His research interest includes AI, Soft computing neural networks, fuzzy logic, data mining, statistics, simulation and stochastic modeling of mobile communication system and networks and network security. He is author of three books, various technical reports and supervised more than 100 software/Information technology projects of Masters level degree programs and project director of various projects funded by Government of Pakistan.

He is member of IEEE (USA), ACM (USA) and fellow of Royal Statistical Society United Kingdom and also a member of Islamic society of Statistical Sciences. He is teaching since 1973 in various universities in the field of Econometric, Bio-Statistics, Statistics, Mathematic and Computer Science He has vast education management experience at the University level. Dr. Burney has received appreciations and awards for his research and as educationist.

**Tahseen A. Jilani** received the B.Sc.(Computer Science, Mathematics, Statistics) from Islamia Science College affiliated with the University of Karachi in 1998; First class second M.Sc. (Statistics,) from Karachi University in 2001. Since 2003, he is Ph.D. research fellow in the Department of Computer Science, University of Karachi.



His research interest includes AI, neural networks, soft computing, fuzzy logic, Statistical data mining and simulation. He is teaching since 2002 in the fields of Statistics, Mathematics and Computer Science.



**S. M. Aqil Burney** received the B.Sc.(Physics, Mathematics, Statistics) from D. J. College affiliated with the University of Karachi in 1970; First class first M.Sc. (Statistics,) from Karachi University in 1972 with M.Phil. in 1983 with specialization in Computing, Simulation and stochastic modeling in from risk management. Dr. Burney received Ph.D.