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

Syed Muhammad Aqil Burney, Tahseen Ahmed Jilani, Cemal Ardił

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 outputs. That is

\[ \mathbf{O} = f_{\text{composite}} \left( \mathbf{x} \right) \]

Using p-mapping layers in a \( p+1 \) layer feedforward net yield

\[ \mathbf{O} = f L_{p} \left( f L_{p-1} \left( \ldots f L_{2} \left( f L_{1} \left( \ldots \right) \right) \right) \right) \]. Thus the interconnection weights from unit \( k \) in \( L_{i} \) to unit \( i \) in \( L_{j} \) are \( w_{jk}^{i} \). If hidden units have a sigmoidal activation function, denoted \( f_{\text{sig}} \)

\[ \mathbf{O}_{L_{j}}^{t} = \sum_{i=1}^{H} w_{ji}^{t} \left( \mathbf{f}_{\text{sig}} \left( \sum_{j=1}^{I} w_{ij}^{t-1} \mathbf{I}_{j}^{t-1} \right) \right) \]

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 \( \Delta w_{ij} \) to the synaptic weight \( w_{ij} \) 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 \( k^{th} \) neuron at time \( n \) and let the actual response of the neuron is \( \mathbf{O}_{j}(n) \). 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.

\[ J = E \left[ \frac{1}{2} \sum_{k} \mathbf{e}_{k}^{2}(n) \right] \]  

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 \( J \) with respect to free...
parameters of the network leads to so-called Method of Gradient Descent.

\[ w_{k,(n+1)} = w_{k,(n)} - \eta \nabla J(n) \]  

\( \eta \) 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.

![Image of error surface with local and global minima](Figure 1)

We can gain understanding and intuition about the algorithm by studying error surfaces themselves—the function \( J(w) \). 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].

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

\[ \text{net} = w^T \cdot x \]

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

\[ H = \left\{ (x^1, t^1), (x^2, t^2), \ldots, (x^n, t^n) \right\} \]

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

\[ \frac{d(e^p)^2}{dw_k} = \frac{d(e^p)^2}{dO_p} \frac{dO_p}{d\text{net}} \frac{d\text{net}}{d\text{net}^k} \frac{d\text{net}^k}{dw_k} \]

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

![Image of gradient descent algorithm](Figure 2)

Fig 2 Gradient Descent algorithm showing minimization of cost function

International Scholarly and Scientific Research & Innovation 1(1) 2007 146 scholar.waset.org/1999.4/9681
\[ w_{k+1} = w_k - \eta \left[ (O' - t') \frac{d E}{d n} \cdot x_k \right] \]  
and for linear activation function
\[ w_{k+1} = w_k - \eta \left[ (O' - t') \cdot x_k \right] \quad w_{k+1} = w_k + \eta e^p x_k^p \]

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 \( w_n \) and we wish to obtain a particular search direction \( p_n \) through weight space. The minimum along the search direction then gives the next values for the weight vector as:

\[ w_{n+1} = w_n + \lambda_n p_n \]  
where the parameter \( \lambda \) is chosen to minimize
\[ E(\lambda) = E(w_n + \lambda p_n) \]

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
\[ \frac{\partial}{\partial \lambda} E(w_n + \lambda p_n) = 0 \]  
gives \[ g(w_{n+1})^T \cdot p_n = 0 \]  
where \[ g_n = \frac{\partial E}{\partial w_{n+1}} \]
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 \( p_n \) 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 \( d_n \), starting from the point \( w_n \), to give the new point \( w_{n+1} \). Then at the point \( w_{n+1} \), we have
\[ g_{n+1}(w) \cdot p_n = 0 \]  
Now the new search \( d_n \) such that
\[ g(w_{n+1} + \lambda p_{n+1})^T \cdot p_n = 0 \]  
gives \[ p_{n+1} \cdot p_n = 0 \]
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 \( H \). 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 \( \Delta w_n \) 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 \( w_{n+1} \).

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 \( p(0) = -g(0) \). Line minimization in the direction \( p(0) \) takes weight vector \( w(0) \). Beginning with \( K = 0 \), we require
subsequent gradient to be orthogonal to the previous weight correction direction; that is from [8]
\[ p_n^T g_{n+1} = 0 \quad ; \quad i = 1, 2, \ldots, n \]
\[ p_n [g_{n+2} - g_{n+1}] = 0 \]  (10)
Where \( g_{n+2} - g_{n+1} \) is the change of gradient of \( E \) from \( w_{n+1} \) to \( w_{n+2} \).

Using Taylor’s series, we have
\[ E(w) \approx E(w_0) + \frac{\partial E}{\partial w} \big|_{w=w_0} \ (w-w_0) + \frac{1}{2!} (w-w_0)^2 \frac{\partial^2 E}{\partial^2 w} \big|_{w=w_0} (w-w_0) \]
where \( J = \frac{\partial E}{\partial w} \) and \( H = \frac{\partial^2 E}{\partial^2 w} = \left[ \frac{\partial^2 E}{\partial w_i \partial w_j} \right] \)

Differentiating with respect to \( w \), we have \( g(n) = F + H(w - w_0) \)

Therefore \( p_n^T H p_{n+1} = 0 \)  (11)
which is called the conjugacy condition, see fig. 3. Weight correction direction \( p_{n+1} \) and \( p_n \) are said to be conjugate to one another in the context of \( H \), or \( H \)-conjugate. It is important to note that there exist several methods of finding conjugate vector \( p(n + 1) \) 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.
\[ p_{n+1} = -g_{n+1} + \beta_{n+1} \cdot p_n \]  (12)
defines a path or sequence of search directions \( p_n \); \( n = 1, 2, \ldots, n \) in the weight space where \( p_n \) is a time varying parameter. There are various rules for determining \( \beta_n \) in terms of the gradient vector \( g_n \) and \( g_{n+1} \). Each weight correction \( p_n \) is computed sequentially with \( p_0 \) 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
\[ p_n^T H p_{n+1} = p_n^T H [-g_{n+1} + \beta_{n+1} \cdot p_n] \]
\[ \beta_{n+1} = \frac{p_n^T H g_{n+1}}{p_n^T H p_n} \]  (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,

Fletcher-Reeves \( \beta_{n+1} = \frac{g_{n+1}^T \cdot g_{n+1}}{g_n^T \cdot g_n} \)  (14)

Polak-Ribiere \( \beta_{n+1} = \frac{g_{n+1}^T (g_{n+1} - g_n)}{g_n^T g_n} \)  (15)

Powell-Beale Restarts \( \beta_{n+1} = \frac{g_{n+1}^T (g_{n+1} - g_n)}{p_n^T (g_{n+1} - g_n)} \)  (16)

The conjugate gradient algorithms are usually much faster than variable leaning 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 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 \( \beta_n \) so that the
the step size \( \beta_n \) 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 \( W \) line minimization, 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.

\[
\frac{g}{f} = \frac{\lambda_n}{g'}
\]

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 \( \beta_n \) is nearly always taken as unity. In order to
ensure that he 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

\[
E(w) = E(w^*) + \frac{1}{2} (w - w^*)^T H (w - w^*)
\]
gives \( w^* = w - H^{-1} g \)

The vector \( -H^{-1} g \) 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 \( w \), points directly at the minimum of the error function
provided \( H \) is positive definite. Since the quadratic
approximation in above algorithm is not exact, it would be
necessary to apply \( w = w - H^{-1} g \) iteratively with the
Hessian being re-evaluated at each new search [40]. The
Hessian for non-quadratic networks is computationally
demanding since it required \( O(Nw^2) \) steps, where \( W \) is the
number of weights in the network and \( N \) is the number of
patterns in the data and \( O(w^3) \) 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
occur 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

\[
w_{n+1} = w_n - G^{-1} g_n
\]

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

\[
a(n) = \frac{g(w_{n+1}) - g(w_n)}{w_{n+1} - w_n}
\]
such an iteration is called a quasi-Newton method. If \( G \) is
positive definite, as it usually is, an alternative name is
variable metric method. One positive advantage to using an
approximation in place of \( H \) is that \( G \) 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 \( G(n)^{-1} g(w_n) \) is a descent direction
from \( w_n \) 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 in place of \( H \) that is \( G \) 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 \( G(n)^{-1} g(w_n) \) is a descent direction
from \( w_n \) allowing the use of line searches.

IX. LEVENBERG-MARQUARDT ALGORITHM

Levenberg-Marquardt is a trust region based method with
hyper-spherical trust region. This method work extremely well
in practice, and is considered the most efficient algorithm for
training medium sized artificial neural networks. Like Quasi-
Newton methods, the Levenberg-Marquardt algorithm was
designed to approach second order training speed with out
having to compute Hessian matrix [1]. When the performance
function has the form of a sum of squares, then from (17) we have

$$w_{n+1} = w_n - H^{-1} \nabla F(w)$$  \hspace{1cm} \text{(18)}$$

where the Hessian matrix can be approximated as, $H \approx J^T J$

and the gradient can be computed as $g = J^T e$

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

$$W_{n+1} = W_n - (J_n^T J_n)^{-1} J_n^T e_n$$

Where $(J^T J)$ 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

$$W_{n+1} = W_n - (J_n^T J_n + \lambda I)^{-1} J_n^T e_n$$  \hspace{1cm} \text{(19)}$$

Where in neural computing context the quantity $\lambda$ is called the learning parameter, it ensures that $J^T J$ 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 $J^T J + \lambda I$ is involved which requires large memory space to store the Jacobian matrix and the approximated Hessian.

**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](image)

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 $10^{-5}$.**

**REFERENCES**


