On Training Of Artificial Neural Networks

L.N.M.Tawfiq & R.S.Naoum

College of Education Ibn Al-Haitham, Baghdad University

المستخلص

يتضمن البحث مناقشة أنواع مختلفة من خوارزميات تدريب الشبكات العصبية ذات التغذية التقدمية

وفي كل تلك الخوارزميات استخدمنا مشتقة دالة الطاقة لتحديد كيفية ضبط الأوزان بحيث تصبح دالة الطاقة أصغر ما يمكن و لقد استخدمنا خوارزمية الانتشار المرتد لزيادة سرعة التدريب. تختلف الخوارزميات أعلاه في حساباتها و لذلك نحصل على صيغ متنوعة في اتجاه التفتيش و الخزن الذي تقتضيه فقد أثبتت النتائج العملية بأن أيا من الخوارزميات أعلاه لا تمتلك خواص رئيسية مثل الاستقرارية و التقارب و التي تجعلها مناسبة لكل المسائل .

Abstract

In this paper we describe several different training algorithms for feed forward neural networks. In all of these algorithms we use the gradient of the performance function, energy function, to determine how to adjust the weights such that the performance function is minimized, where the back propagation algorithm has been used to increase the speed of training. The above algorithms have a variety of different computation and thus different type of form of search direction and storage requirements, however non of the above algorithms has a global properties which suited to all problems.

INTRODUCTION

Back propagation(BP)process can train multilayer FFNN's. With differentiable transfer functions, to perform a function approximation to continuous function $f \in R^n$, pattern association and

pattern classification. The term of back propagation to the process by which derivatives of network error with respect to network weights and biases, can be computed. This process can be used with a number of different optimization strategies.

There are two different ways in which BP algorithms can be implemented; incremental mode and batch mode. All of algorithms, in this paper, operate in the batch mode and are invoked using certain type of training.

1.GRADIENT(STEEPEST) DESCENT(TRAINGD)

A standard back propagation algorithm is a gradient descent algorithm (as in the Widrow-Hoff learning rule) .For the basic steepest (gradient) descent algorithm, the weights and biases are moved in the direction of the negative gradient of the performance function . For the method of gradient descent, the weight update is given by: $w(k+1) = w(k) + \alpha_k(-g_k)$ ……..….(1)

where α_k regulates the learning rate and g_k is the gradient of the error surface at w(k).

If the learning rate is made too large the algorithm become unstable. If the learning rate is set too small, the algorithm will take a long time to converge. Then the convergence condition is satisfied by choosing: $0 < \alpha_k < \frac{1}{2\lambda_{max.}}$

where $\lambda_{max.}$ is the largest eigenvalue of weight matrix [1].

2. GRADIENT DESCENT WITH MOMENTUM(TRAINGDM)

There is another training algorithm for FFNN that often provides faster convergence. The weight update formulas for gradient descent with momentum is given by:

$$w(k+1) = w(k) + \alpha_k(-g_k) + \mu(w(k) - w(k-1))$$

that is: $w(k+1) = w(k) + \alpha_k(-g_k) + \mu\Delta w(k)$ ……(2)

where the momentum parameter μ is constrained to be in the range (0, 1). Momentum allows the ANN to make reasonably large weight adjustments, while using a smaller learning rate to prevent a large response to the error from any one of training pattern .

3. FASTER TRAINING

In this section, we will discuss several high performance algorithms fall into two main categories. The first category uses heuristic techniques, which were developed from an analysis of the performance of the standard gradient descent algorithm. Another heuristic modification is the momentum technique, variable learning rate and resilient back propagation.The second category of fast algorithms uses standard numerical optimization techniques such as: conjugate gradient, quasi-Newton and Levenberg-Marquardt.

3.1.Variable Learning Rate

With standard gradient descent, the learning rate is held constant through out training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. If the learning rate is set too high, the algorithm become unstable. If the learning rate is too small, the algorithm will take too long to converge. Our numerical results shows that it is not practical to determine the optimal setting for the learning rate before training and, in fact, the optimal learning rate changes during the training process, as the algorithm moves across the performance surface.

We now describe in some detail one-dimensional search procedure that is guaranteed to find a learning rate satisfying the strong Wolfe conditions (3). As before, we assume that ρ is a search direction and that f is bounded below along the direction ρ. The algorithm has two stages. The first stage begins with a trial estimate α_1, and keeps increasing it until it finds either an acceptable learning rate or an interval of desired learning rates. In the latter case, the second stage is invoked by calling a function called zoom (Zoom Algorithm), which successively decreases the size of the interval until an acceptable learning rates is identified. Now we introduce Strong Wolfe Conditions:

$$f(w_k + \alpha_k\rho_k) \le f(w_k) + 10^{-4}\alpha_k \nabla f_k^T \rho_k \text{.....(3a)}$$

$$|\nabla f(w_k + \alpha_k\rho_k)\, \rho_k| \le 0.1|\nabla f_k^T \rho_k| \text{..... (3b)}$$

Variable Learning Rate Algorithm

Set $\alpha_0 \leftarrow 0$, choose $\alpha_1 > 0$ and $\alpha_{max.}$;

$i \leftarrow 1$;

repeat

Evaluate $\phi(\alpha_i)$;

If $\phi(\alpha_i) > \phi(0) + 10^{-4}\alpha_i\phi'(0)$ or $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$ and $i > 1]$

$\alpha_* \leftarrow$ zoom (α_{i-1}, α_i) and stop;

Evaluate $\phi'(\alpha_i)$;

If $|\phi'(\alpha_i)| \leq -0.1\phi'(0)$

Set $\alpha_* \leftarrow \alpha_i$ and stop;

If $\phi'(\alpha_i) \geq 0$

Set $\alpha_* \leftarrow$ zoom(α_{i-1}, α_i) an stop;

Choose $\alpha_{i+1} \in (\alpha_i, \alpha_{max})$

$i \leftarrow i + 1$,

end (repeat).

Note that, the sequence of trial learning rates $\{\alpha_i\}$ is monotonically increasing, but that the order of the arguments supplied to the zoom function may vary. The procedure uses the knowledge that the interval (α_{i-1}, α_i) contains learning rate satisfying the strong Wolfe conditions if one of the following three conditions is satisfied:

(i) α_i violates the sufficient decrease condition;

(ii) $\phi(\alpha_i) \geq \phi(\alpha_{i-1})$;

(iii) $\phi'(\alpha_i) \geq 0$.

The last step of the algorithm performs extrapolation to find the next trial value α_{i+1}. To implement this step, we can use approaches like the interpolation procedures above, or we can simply set α_{i+1} to some constant multiple of α_i.

We now specify the function zoom, which will requires a little explanation. The order of its input arguments is such that each call has the form zoom (α_{Lo}, α_{hi}), where:

(a) The interval bounded by α_{Lo} and α_{hi} contains learning rates that satisfy the strong Wolfe conditions;

(b) α_{Lo} is among all learning rates generated so far and satisfying the sufficient decrease condition, the one giving the smallest function value; and

(c) α_{hi} is chosen so that $\phi'(\alpha_{lo})(\alpha_{hi} - \alpha_{10}) < 0$.

Each iteration of zoom generates an iterate α_j between α_{Lo} and α_{hi}, and then replaces one of these end points by α_j in such a way that the properties (a), (b) and (c) continue to hold.

Zoom Algorithm

Repeat

Interpolate (using quadratic, cubic, or bisection) to find a trial learning rate α_j between α_{lo} and α_{hi};

Evaluate $\phi(\alpha_j)$;

If $\phi(\alpha_j) > \phi(0) + 10^{-4}\phi'(0)$ or $\phi(\alpha_j) \geq \phi(\alpha_{lo})$

$\alpha_{hi} \leftarrow \alpha_j$;

else

evaluate $\phi'(\alpha_j)$;

if $|\phi'(\alpha_j)| \leq -0.1\phi'(0)$

set $\alpha_* \leftarrow \alpha_j$ and stop;

if $\phi'(\alpha_j)(\alpha_{hi} - \alpha_{lo}) \geq 0$

$\alpha_{hi} \leftarrow \alpha_{lo}$;

$\alpha_{lo} \leftarrow \alpha_j$;

end (repeat).

If the new estimate α_j happens to satisfy the strong Wolfe conditions, then Zoom has served its purpose of identifying such a point, so it terminates with $\alpha_* = \alpha_j$. Otherwise, if α_j satisfies the sufficient decrease condition and has a lower function value than α_{Lo}, then we set $\alpha_{Lo} \leftarrow \alpha_j$ to maintain condition (b). If this results in a violation of condition (c), we remedy the situation by setting α_{hi} to the old value of α_{Lo}.

3.2.Resilient Back Propagation (TRAINRP)

The resilient back propagation training algorithm eliminates the harmful effect of having a small slope at the extreme ends of sigmoid transfer functions in hidden layers. Only the sign of the derivative of the transfer function is used to determine the direction of the weight update: the magnitude value of the derivative has no effect on the weight update. Our results shows the resilient back propagation is generally much faster than the standard gradient descent algorithm.

Also it has a nice property that it requires only a modest increase in memory requirements, and thus we do need to store the update values for each weight and bias.

3.3.Quasi-Newton Algorithms

Quasi-Newton (or secant) methods are based on Newton's method but dont require calculation of second derivatives (at each step). They update an approximate Hessian matrix at each iteration of the algorithm.

The optimum weight value can be computed in an iterative manner by writing:

$$w(k+1) = w(k) - \alpha_k H^{-1} g \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4)$$

where α_k is the learning rate, g_k is the gradient of the error surface with respect to the w(k) and H is the Hessian matrix (second derivatives of the error surface with respect to the w(k)). We can show that the Quasi-Newton's method converges to the optimal weight w*. Now rewrite the equation of Newton's method as: $w^* = w(k) - \frac{1}{2} H^{-1} g_k$ .....(5)

Therefore, from eqs.(4) and (5), we get :$w(k+1) = w(k) - 2\alpha_k(w(k) - w^*) = w(k)(1 - 2\alpha_k) + 2\alpha_k w^*$

Starting with an initial weight of w(0), we get:

$$w(1) = w(0)(1 - 2\alpha_k) + 2\alpha_k w^* = w^* + (1 - 2\alpha_k)(w(0) - w^*)$$

$$w(2) = w(1)(1 - 2\alpha_k) + 2\alpha_k w^* = w(0)(1 - 2\alpha_k)^2 + 2\alpha_k w^*(1 - 2\alpha_k) + 2\alpha_k w^*$$

$$= w^* + (1 - 2\alpha_k)^2(w(0) - w^*)$$

$$w(k) = w^* + (1 - 2\alpha_k)^m(w(0) - w^*)$$

Since $w(0) - w^*$ is fixed, $w(k)$ converges to w^*, provided: $0 < 2\alpha_k \leq 1$, i.e., $0 < \alpha_k \leq ½$.

We see that in the quasi-Newton method the steps do not proceed along the direction of the gradient . Now we introduce two quasi-Newton algorithms :

3.3.1.BFGS Quasi-Newton Algorithm (TRAINBFG)

This algorithm requires more computation for each iteration and our results shows more storage require than the CG methods, although, generally, converges in fewer iterations. For a very large ANN it may be better to use resilient back propagation or one of the CG algorithms. For smaller ANN, however, BFGS quasi-Newton algorithm can be used as an efficient training function.

3.3.2.One Step Secant Algorithm (TRAINOSS)

Since the BFGS algorithm requires more storage and computation in each iteration than the CG algorithms, there is need for a secant approximation with smaller storage and computation

requirements. The one step secant (OSS) method is an attempt to bridge the gap between the CG algorithms and the quasi-Newton (secant) algorithms .

This algorithm does not store the complete Hessian matrix; it assumes that at each iteration the previous Hessian was the identity matrix. This has the additional advantage that the new search direction can be calculated without computing a matrix inverse .

3.4.Levenberg-Marquardt Algorithm (TRAINLM)

The Levenberg-Marquardt algorithm was designed to approach second order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares, then the Hessian matrix can be approximated as $H = J^T J$ and the gradient can be computed as $g = J^T e$, where J is the Jacobian matrix, which contains first derivatives of the network errors with respect to the weights and biases, and e is a vector of network errors. The Levenberg-Marquardt algorithm uses this approximation to the Hessian matrix in the following Newton update: $w(k+1) = w(k) - [J^T J + \mu I]^{-1} J^T e$

when the scalar $\mu = 0$, this is just Newton's method. When μ is large, this becomes gradient descent with a small step size.

3.5.Conjugate Gradient Algorithms (TRAINCG)

The conjugate gradient algorithms perform a search along conjugate directions, which produces generally faster convergence than gradient descent directions [Hagan and Beale, 1996]. The CG algorithms start out by searching in the gradient descent direction (negative of the gradient) on the first iteration ،$\rho_0 = -g_0$. Then the next search direction is determined so that it is conjugate to previous search directions, that is:

$w(k+1) = w(k) + \alpha_k \rho_k$. Where $\rho_k = -g_k + \beta_k \rho_{k-1}$.

The various versions of CG are distinguished by the manner in which the β_k is computed.

In this paper, we will present six different variations of CG algorithms with a comparison between them. In most of the training algorithms a learning rate is used to determine the length of the weight update (step size).

In most of the CG algorithms, the step size is adjusted at each iteration. A search is made along the CG direction to determine the step size, which will minimize the performance function along that line search. The CG algorithms that usually used in ANN as a training algorithm is much faster than variable learning rate back propagation, and are sometimes faster than Resilient back propagation, although the results will vary from one problem to another.

3.5.1.FLETCHER-REEVES UPDATE (TRAINCGF)

The general procedure for determining the new search direction is to combine the new gradient descent direction with the previous search direction: $\rho_k = -g_k + \beta_k \rho_{k-1}$

For Fletcher-Reeves update procedure [2] : $\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$

The training parameters for traincgf are: epochs, show, goal, time, min-grad, srchFcn.

The training status will be displayed every show iterations of the algorithm. The other parameters determine when the training is stopped. The training will stop when the number of iterations exceeds an epochs, if the performance function drops below goal, if the magnitude of the gradient is less than mingrad or if the training time is longer than time in seconds. The parameter srchfcn is the name of the line search function. traincgf generally converges in fewer iterations than Resilient back propagation (TRAINRP) (although there is more computation required in each iteration).

3.5.2.POLAK-RIBIERE UPDATE (TRAINCGP)

Another version of the conjugate gradient algorithm was proposed by Polak and Ribiere[3].

For the Polak-Ribiere update, the constant β_k is computed from: $\beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}}$

The traincgp routine has performance similar to traincgf. It is difficult to predict which algorithm will perform best on a given problem. The storage requirements for Polak-Ribiere (four vectors) are slightly larger than for Fletcher-Reeves (three vectors).

3.5.3.DXON UPDATE (TRAINCGD)

We propose another version of the conjugate gradient algorithm, which derive from classical method proposed by Dixon [4].

For the Dixon update, the constant β_k is computed by: $\beta_k = \frac{-g_k^T g_k}{\rho_{k-1}^T g_{k-1}}$

The training parameters for traincgd are: epochs ،show ،goal ،time ، min-grad, max-fail, srchFcn, scal-tol, alpha, beta, delta, gama, low-lim, up-lim, maxstep, minstep, bmax.

The training status will be displayed every show iterations of the algorithm. The other parameters determine when the training is stopped. The training will stop if the number of iterations exceeds epochs, if the performance function drops below goal, if the magnitude of the gradient is less than mingrad, or if the training time is longer than time seconds, max-fail which is associated with the early stopping technique.

The parameter srchFcn is the name of the line search function. The remaining parameters are associated with specific line search routines. The default line search routine srchcha is used.

The traincgd routine has performance, which is some what better than traincgp for some problems, although performance on any given problem is difficult to predict.

The storage requirements for the Dixon algorithm (three vectors).

3.5.4.AL-ASSADY AND AL-BAYATI UPDATE (TRAINCGA)

We use another version of the conjugate gradient algorithm, when the classical method proposed by Al-Assady and Al-Bayati [5].

For the Al-Assady and Al-Bayati update, the constant β_k is computed by: $\beta_k = \frac{-g_k^T \Delta g_{k-1}}{\rho_{k-1}^T g_k}$

The training parameters for traincga are: epochs, show, goal, time, min-grad, max-fail, srchFcn. The storage requirements for the Al-Assady and Al-Bayati algorithm (four vectors)

3.5.5.HESTENES-STIEFEL UPDATE (TRAINCGH)

We will consider another version of the CG algorithm, when the classical method proposed by Hestenes-Stiefel [6].

For the Hestenes-Stiefel update, the constant β_k is computed by: $\beta_k = \frac{g_k^T \Delta g_{k-1}}{\rho_{k-1}^T \Delta g_{k-1}}$

The traincgh routine has performance similar to traincgd.

The storage requirements for the Hestenes-Stiefel algorithm (four vectors)

3.5.6.REYADH-LUMA UPDATE (TRAINCGR)

We propose a new version of the CG algorithm when the search direction at each iteration is determined by: $\rho_k = - g_k + \beta_k \rho_{k-1}$

Where the constant β_k is computed by: $\beta_k = \frac{g_k^T \Delta g_{k-1}}{\rho_{k-1}^T g_{k-1}}$

The training parameters for traincgr are: epochs, show, goal, time, min-grad, max-fail, sigma, lambda.

We have previously discussed the first six parameters and the parameter sigma determines the change in the weight for the second derivative approximation .The parameter lambda regulates the indefiniteness of the derivative.

The storage requirement for Reyadh-Luma (four vectors)

Remark

1. For all CG algorithms, the search direction will be periodically reset to the negative of the gradient. The standard reset point

occurs when the number of iterations is equal to the number of ANN parameters (weights and biases).

2. For all CG algorithms, the parameters show and epoch set to 5 and 300, respectively.

3. Each of the CG algorithms, which we have discussed so far, requires a line search at each iteration. This line search is computationally expensive, since it requires that the ANN response to all training inputs which should be computed several times for each search. But the other hand one can designed an algorithm to avoid the time consuming for performing line search.

4.Newton's method has a quadratic convergence property , that is $|e_{n+1}| \leq \varepsilon |e_n|^2$ and thus often

converges faster than CG methods. Unfortunately, it is expensive because we need to compute the

Hessian matrix (second derivatives of the error surface with respect to the weight).

3.5.7.LINE SEARCH ROUTINES (SRCHCHA)

The method of srchcha was designed to be used in a combination with a CG algorithm for ANN training. We have used this routine as the default search for most of the CG algorithms, since it appears to produce excellent results for many different problems. It does require the computation of the derivatives (back propagation) in

addition to the computation of performance function, but it over comes this limitation by locating the minimum with fewer steps.

4.SPEED AND MEMORY COMPARISON

It is very difficult to know which training algorithm will be the fastest for a given problem. It will depend on many factors including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the ANN, the error goal, and whether the ANN is being used for pattern recognition (discriminant analysis) or function approximation (regression).

In general, on ANN's which contain up to a few hundred weights the Levenberg-Marquardt algorithm will have the fastest convergence. The trainrp function is the fastest algorithm on pattern recognition problems. However, it does not perform well on function approximation on problems. The CG algorithms, in particular traincgp, seem to perform well over a wide variety of problems, particularly for ANN's with a large number of weights. The traincgr algorithm is almost as fast as the Levenberg-Marquardt algorithm on function approximation problems (faster for large ANN's) and is almost as fast as trainrp on pattern recognition problems. The CG algorithms have relatively modest memory requirements.

The trainbfg performance is similar to that of trainlm. It does not require as much storage as trainlm, but the computation required does increase geometrically with the size of the ANN, since the

equivalent of a matrix inverse must be computed at each iteration. Of the CG algorithms, the traincgd requires the most storage, but usually has the fastest convergence. The traincgh and traincga have easily implemented for large problem.

The variable learning rate algorithm traingdx is usually much slower than the other methods and has about the same storage requirements as trainrp but it can still be useful for some problems. For most situations, we recommend that we try to use the Levenberg - Marquardt algorithm first, if this algorithm requires too much memory, then try traincgp or traincgr or trainbfg algorithm. The following table gives some example convergence times for the various algorithms on one particular regression problem. In this problem a 1-15-1 FFNN's was trained on a data set with 41 input/output pairs until a mean square error performance of 0.008 was obtained. Twenty different test runs were made for each training algorithm to obtain the average numbers shown in the table.

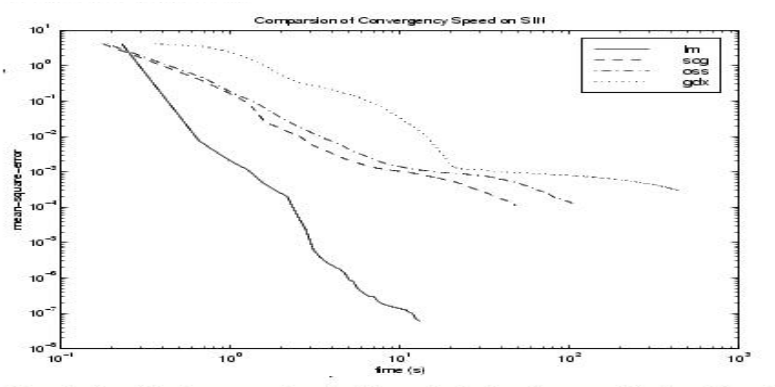
Function	*Technique*	*Time(sec)*	*Epochs*
Trainrp	Rprop.	12.95	185
Traincgh	Hestenes-stiefel CG	27.22	112
Traincgf	Fletcher-Powell CG	18.03	94
Traincgp	Polak-Ribiere CG	18.66	79
Traincgd	Dixon CG	24.52	101
Traincgr	Reyadh-Luma CG	14.98	58
Trainbfg	BFGS quasi-Newton	9.76	38
Trainlm	Levenberg-Marquardt	2.07	8

Traingdx	Variable learning rate	63.17	124
Traincga	Al-Assady and Al-Bayati CG	71.36	54

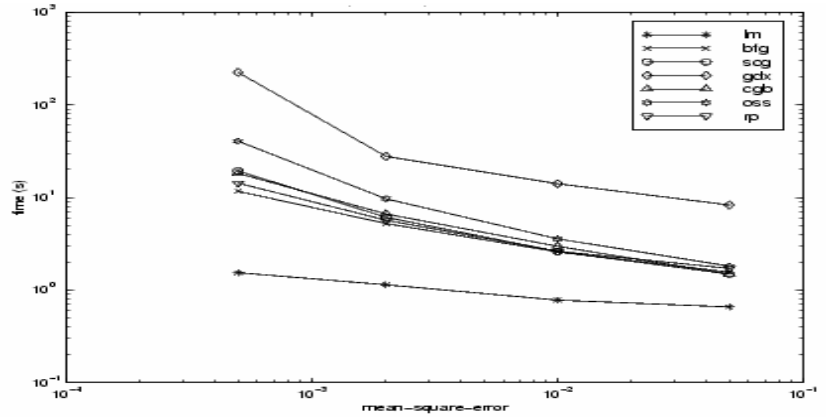
Now we introduce the following problem. 1-5-1 network, with tansig transfer functions in the hidden layer and a linear transfer function in the output layer, is used to approximate a single period of a sine wave. The following table summarizes the results of training the ANN using nine different training algorithms. Each entry in the table represents 30 differ- ent trials,where different random initial weights are used in each trial.In each case,theANN is trained until the squared error is less than 0.002. The fastest algorithm for this problem is the Levenberg-Marquardt algorithm.On the average,it is over four times faster than thenext fastest algorithm. This is the type of problem for which the LM algorithm is best suited -- a function approximation problem where the network has less than one hundred weights and the approximation must be very accurate.

Algorithm	*Mean.Time(s)*	*Min.Time(s)*	*Max.Time(s)*
LM	1.14	0.65	1.83
BFG	5.22	3.17	14.38
RP	5.67	2.66	17.24
CGF	7.86	3.57	31.23
CGP	8.24	4.07	32.32
OSS	9.64	3.97	59.63
CGR	5.92	2.31	16.47
CGA	27.69	17.21	258.15
CGD	6.09	3.18	23.64
CGH	6.61	2.99	23.65

The performance of the various algorithms can be affected by the accuracy required of the approximation. This is demonstrated in the following figure, which plots the mean square error versus execution time (averaged over 30 trials) for several representative algorithms. Here we can see that the error in the LM algorithm decreases much more rapidly with time than the other algorithms shown.



The relationship between the algorithms is further illustrated in the following figure, which plots the time required to converge versus the mean square error convergence goal. We can see that as the error goal is reduced the improvement provided by LM algorithm becomes more pronounced Some algorithms perform better as the error goal is reduced (LM and BFG), and other algorithms degrade as the error goal is reduced (OSS and CGA).



5.LIMITATIONS AND CAUTIONS

The gradient descent algorithm is generally very slow, because it requires small learning rates for stable learning. The momentum variation is usually faster than simple gradient descent, since it allows higher learning rates while maintaining stability, but it is still too slow

for many practical applications. These two methods would normally be used only when incremental training is desired. Multi-layered networks are capable of performing just about any linear or non-linear computation, and can approximate any reasonable smooth function arbitrarily well. Such networks overcome the problems associated with the feed forward and linear networks.

Picking the learning rate for a non-linear network is still an open problem. As with linear networks, a learning rate that is too large leads to unstable learning. Conversely, a learning rate that is too small results in incredibly long training times. Unlike linear networks, there is no easy way of picking a good learning rate for non-linear multilayer networks.

The error surface of a non-linear network is more complex than the error surface of a linear network. The problem is that non-linear transfer function in multilayer networks introduce many local minima in the error surface. Settling in a local minimum may affect the convergence and depending on how close the local minimum is to the global minimum and how low an error is required. In any case, be cautioned that although a multilayer back propagation network with enough neurons can implement just about any function, back propagation will not always find the correct weights for the optimum solution.

References

[1] B. Yegnanarayana, Artificial Neural Networks, Newdelhi, 2000.

[2] R. Fletcher and C.M. Reeves, Function Minimization by Conjugate Gradients, Computer Journal, Vol. 7, P. 149 – 154, 1964.

[3] E. Polak and G. Ribiere, Note sure La Convergence does methods Directions Conjugate, Rev. Fr. Infr, Rech open, 16-R1, 6, 1969.

[4] L.G. Dixon, Conjugate Gradient algorithms quadratic termination with out linear search, Jor. of Tnst. of Math. and its applications, Vol. 15, 1975.

[5] A. Al - Bayati and N. Al - Assady, Conjugate Gradient Methods, Technical Research Report, NO.1, School of Computer Studies, Leeds University, U. K., 1996.

[6] M. R. Hestenes and E. Stiefel, Methods of Conjugate Gradient for Solving linear System, J. Res. NBS, Vol. 49, 1952.