International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:6, 2008

An Improved Conjugate Gradient Based Learning Algorithm for Back Propagation Neural Networks

N. M. Nawi[1], R. S. Ransing[*] and M. R. Ransing[*]

Abstract**—**The conjugate gradient optimization algorithm is combined with the modified back propagation algorithm to yield a computationally efficient algorithm for training multilayer perceptron (MLP) networks (CGFR/AG). The computational efficiency is enhanced by adaptively modifying initial search direction as described in the following steps: (1) Modification on standard back propagation algorithm by introducing a gain variation term in the activation function, (2) Calculation of the gradient descent of error with respect to the weights and gains values and (3) the determination of a new search direction by using information calculated in step (2). The performance of the proposed method is demonstrated by comparing accuracy and computation time with the conjugate gradient algorithm used in MATLAB neural network toolbox. The results show that the computational efficiency of the proposed method was better than the standard conjugate gradient algorithm.

Keywords**—**Adaptive gain variation, back-propagation, activation function, conjugate gradient, search direction.

I. INTRODUCTION

GRADIENT based methods are one of the most widely used error minimization methods used to train back propagation networks. The back-propagation (BP) training algorithm is a supervised learning method for multi-layered feed-forward neural networks [1]. It is essentially a gradient descent local optimization technique which involves backward error correction of network weights. Despite the general success of back-propagation method in the learning process, several major deficiencies are still needed to be solved. The convergence rate of back-propagation is very low and hence it becomes unsuitable for large problems. Furthermore, the convergence behavior of the back-propagation algorithm depends on the choice of initial values of connection weights and other parameters used in the algorithm such as the learning rate and the momentum term.

[1] Faculty of Information Technology and Multimedia, Kolej Universiti Teknologi Tun Hussein Onn (KUiTTHO), P. O. Box 101, 86400, Parit Raja, Batu Pahat, Johor Darul Takzim, Malaysia
E-mail:matyie.usm97@gmail.com, currently PhD student, University of Wales, Swansea, UK
[*] Civil and Computational Engineering Centre, University of Wales, Singleton Park, Swansea, SA2 8PP, United Kingdom.
E-mail:R.S.Ransing@swansea.ac.uk

Improving the training efficiency of neural network based algorithms is an active area of research and numerous papers have been proposed in the literature. Early days of back propagation algorithms saw improvements on: (i) selection of better activation function [2-4]; (ii) selection of dynamic learning rate and momentum [5-7].

Later, as summarized by Bishop [8] various optimization techniques were suggested for improving the efficiency of error minimization process or in other words the training efficiency. Among these are methods of Fletcher and Powel [9] and the Fletcher-Reeves [10] that improve the conjugate gradient method of Hestenes and Stiefel [11] and the family of Quasi-Newton algorithms proposed by Huang [12].

Recently, the influence of gain was studied by few researchers [13-15]. The gain parameter controls the steepness of the activation function. It has been shown that a larger gain value has an equivalent effect of increasing the learning rate.

This paper suggests that a simple modification to the initial search direction, i.e. the gradient of error with respect to weights, can substantially improve the training efficiency. It was discovered that if the gradient based search direction is locally modified by a gain value used in the activation function of the corresponding node, significant improvements in the convergence rates can be achieved. It has also been shown that the proposed method is robust, easy to compute, and easy to implement into the well known nonlinear conjugate gradient algorithms.

The remaining of the paper is organised as follows: Section II illustrates the proposed method. Sections III discuss the the implementation of the proposed method with Conjugate gradient method. Experiments and simulation results are discussed in section IV. The final section contains concluding remarks and short discussion for further research.

II. THE PROPOSED METHOD

In this section, a novel approach for improving the training efficiency of gradient descent method (back propagation algorithm) is presented. The proposed method modifies the gradient based search direction by changing the gain value adaptively for each node.

The following iterative algorithm is proposed by the authors for changing the gradient based search direction using a gain value. The gradient based search direction is a function of gradient of error with respect to weights.

Step 1 *Initialize the weight vector with random values and the vector of gain values with one.*

Step 2 *Calculate the gradient of error w.r.t. to weights using Equation (4), and gradient of error w.r.t. to gain using Equation (8).*

Step 3 *Use the gradient weight vector and gradient of gain calculated in step 2 to calculate the new weight vector using equation (6)and vector of new gain values using equation (10) for use in the next epoch.*

Step 4 *Repeat the following steps 2 and 3 on an epoch-by-epoch basis until the selected error minimization criteria is satisfied.*

With an optimisation perspective, the objective of a learning process in neural networks is to find a weight vector w that minimizes the difference between the actual output and the desired output on both training and testing data sets. Namely,

$$\min_{w\in\Re^n} E(w) \tag{1}$$

Consider a multilayer feed forward neural network(FNN) [1] with one output layer, one input layer and one or more hidden layers. Each layer has a set of units, nodes, or neurons. It is usually assumed that each layer is fully connected with a previous layer without direct connections between layers which are not consecutive. Each connection has a *weight.*

For a particular input pattern, define an error function on that pattern as,

$$E=\frac{1}{2}\sum_k (t_k - o_k^L)^2 \tag{2}$$

where o_k^L be the activation of the k^{th} node of layer L. Let w_{ij}^L be the weight on the connection from the i^{th} node in layer $L-1$ to the j^{th} node in layer L. The overall error on the training set is simply the sum, across patterns, of the pattern error E.

The net input to the j^{th} node of layer L is defined as $net_j^L = (w_j^L, o^{L-1}) = \sum_k w_{j,k}^L o_k^{L-1}$, The activation of a node o_j^L is given by a function of its net input,

$$o_j^L = f(c_j^L net_j^L) \tag{3}$$

where f is any function with bounded derivative, and c_j^L is a real value called the gain of the node. Note that at $c_j^L = 1$ this activation function becomes the usual logistic activation function.

The weight update expression in equation (6) with a non-unit gain value is derived by differentiating the error term as given in Equation (2) with respect to w_{ij}^L as follows:

$$\frac{\partial E}{\partial w_{ij}^L} = \frac{\partial E}{\partial net^{L+1}} \cdot \frac{\partial net^{L+1}}{\partial o_j^L} \cdot \frac{\partial o_j^L}{\partial net_j^L} \cdot \frac{\partial net_j^L}{\partial w_{ij}^L}$$

$$= [-\delta_1^{L+1} \ldots - \delta_n^{L+1}] . \begin{bmatrix} w_{1j}^{L+1} \\ \vdots \\ w_{nj}^{L+1} \end{bmatrix} . f'(c_j^L net_j^L) c_j^L . o_j^{L-1} \tag{4}$$

In particular, the first three factors of (4) indicate that

$$\delta_1^L = (\sum_k \delta_k^{L+1} w_{k,j}^{L+1}) f'(c_j^L net_j^L) \tag{5}$$

As we noted that the iterative formula (5) for δ_1^L is the same as standard back propagation [16] except for the appearance of the value gain. By combining (4) and (5) yields the learning rule for weights:

$$\Delta w_{ij}^L = \eta \delta_j^L c_j^L o_j^{L-1}$$

$$= \eta \frac{\partial E}{\partial w_{ij}^L}$$

$$w_{new} = w_{old} + \Delta w_{ij}^L \tag{6}$$

where η is a 'learning rate' and the search direction or gradient vector at point w_{ij}^L is $d = \frac{\partial E}{\partial w_{ij}^L} = g$.

In the proposed method the calculation of the gradient of error $g^{(n)}$ at step n is a function of gain $c_j^{L(n)}$.

$$d^{(n)} = -\frac{\partial E}{\partial w_{ij}^{L(n)}}(c_j^{L(n)}) = g^{(n)}(c_j^{L(n)}) \tag{7}$$

The gain value at step n is calculated using gradient of error w.r.t. to gain,

$$\frac{\partial E}{\partial c_j^L} = (\sum_k \delta_k^{L+1} w_{k,j}^{L+1}) f'(c_j^L net_j^L) net_j^L \tag{8}$$

Then the gradient descent rule for the gain value becomes,

$$\Delta c_j^L = \eta \delta_j^L \frac{net_j^L}{c_j^L} \tag{9}$$

At the end of each iteration the new gain value is updated using a simple gradient based method as given by the formula,

$$c_j^{new} = c_j^{old} + \Delta c_j^L \tag{10}$$

III. THE IMPLEMENTATION OF THE PROPOSED ALGORITHM WITH THE CONJUGATE GRADIENT

One of the remarkable properties of the conjugate gradient method is its ability to generate, in a very economical fashion, a set of vectors with a property known as conjugacy [8]. Most widely used conjugate gradient algorithms are given by Fletcher and Powel [9] and Fletcher-Reeves [10]. Both these procedures generate conjugate search directions and therefore aim to minimize a positive definite quadratic function of n variables in n steps.

The proposed algorithm referred to CGFR/AG begins the minimization process with an initial estimate w_0 and an initial search direction as:

$$d_0 = -\nabla E(w_0) = -g_0 \quad (11)$$

The search direction at $(n+1)^{th}$ iteration is calculated as:

$$d_{(n+1)} = -\frac{\partial E}{\partial w_{(n+1)}}(c_{i,n+1}) + \beta_{(n+1)} d_n(c_{i,n}) \quad (12)$$

where the scalar $\beta_{(n+1)}$ is to be determined by the requirement that d_n and d_{n+1} must fulfil the conjugacy property [8]. There are many formulae for the parameter $\beta_{(n+1)}$ and the choice of the formulae for selection of $\beta_{(n+1)}$ is problem dependent [17].

In this paper, common formula as referred by Fletcher and Reeves [10] and is used:

$$\beta_{n+1} = \frac{g_{n+1}^T g_{n+1}}{g_n^T g_n} \quad (13)$$

Like $\beta_{(n+1)}$, the computation of learning rate η also requires knowledge as that of $\beta_{(n+1)}$. The learning rate η can be optimally chosen so as to minimise the error $E(\eta)$ along the chosen search direction d_n.

$$E(\eta) = E(w_{(n+1)}(\eta)) = E(w_n + \eta_n d_n) \quad (14)$$

This gives us an automatic procedure for setting the learning rate, once the search direction is chosen. This procedure is also referred to as 'line search' method.

In this paper we used Golden section search technique to obtain optimized learning rate [18]. The golden section search technique starts by restricting η in $[\eta_1, \eta_h]$. In this paper we set $\eta_1 > 0$ and $\eta_h < 1$, then the following steps are performed.

- Compute $E(\eta_1)$, $E(\eta_h)$
- If $E(\eta_1) < E(\eta_h)$, then set $\eta_h = \eta_1 - 0.618(\eta_1 - \eta_h)$
- If $E(\eta_1) > E(\eta_h)$, then set $\eta_h = \eta_1 + 0.618(\eta_1 - \eta_h)$
- The process is repeated until $(\eta_1 - \eta_h) < \varepsilon$ and then set $\eta = \frac{\eta_1 + \eta_h}{2}$.

The complete CGFR/AG algorithm works as follows;

Step 1 *Initialize the weight vector randomly, the gradient vector* g_0 *to zero and gain vector to unit values. Let the first search direction* d_0 *be* g_0*. Set* $\beta_0 = 0$, $epoch = 1$ *and* $n = 1$*. Let Nt be the total number of weight values. Select a convergence tolerence* CT.

Step 2 *At step* n*, evaluate gradient vector* $g_n(c_n)$.

Step 3 *Evaluate* $E(w_n)$*. If* $E(w_n) < CT$ *then STOP training ELSE go to* ***step 4***.

Step 4 *Calculate a new gradient based search direction which is a function of gain parameter:* $d_n = -g_n(c_n) + \beta_n d_{n-1}$

Step 5 *IF* $n > 1$ *THEN,*

update $\beta_{n+1} = \frac{g_{n+1}^T(c_{n+1}) g_{n+1}(c_{n+1})}{g_n^T(c_n) g_n(c_n)}$ *ELSE go to* ***step 6***.

Step 6 *IF* $[(epoch+1)/Nt] = 0$ *THEN 'restart' the gradient vector with* $d_n = -g_{n-1}(c_{n-1})$ *ELSE go to* ***step 7***.

Step 7 *Calculate the optimal value for learning rate* η_n^* *by using line search technique such as described in Equation(14).*

Step 8 *Update* w_n*:* $w_{n+1} = w_n - \eta_n^* d_n$

Step 9 *Evaluate new gradient vector* $g_{n+1}(c_{n+1})$ *with respect to gain value* c_{n+1}.

Step10 *Calculate new search direction:* $d_{n+1} = -g_{n+1}(c_{n+1}) + \beta_{n+1}(c_n) d_n$

Step11 *Set* $n = n+1$ *and go to* ***step 2***.

IV. PRELIMINERIES

The performance criterion used in this research focuses on the speed of convergence, measured in number of iterations and CPU time. The benchmark problems used to verify our algorithm are available in the open literature [19]. To perform the experiments the data is divided into training and testing sets. After each system processes a training set, its performance is measured on the corresponding test set. To reduce statistical fluctuations, results are averaged over several training and testing sets. In this case two-thirds of the examples in each category are randomly placed in the training set, and the remaining once are placed in the corresponding test sets. Four classification problems have been tested including Iris classification problem, 7 bit parity problem, Wisconsin breast cancer classification problem and diabetes classification problem.

The simulations have been carried out on a Pentium IV with 3 GHz PC, 1 GB RAM and using MATLAB version 6.5.0 (R13).

On each problem, three algorithms have been simulated. The first algorithm is standard conjugate gradient with Fletcher-Reeves update('*traincgf*') from MATLAB Neural Network Toolbox version 4.0.1. The other two algorithms are standard conjugate gradient (CGFR) and our proposed conjugate gradient method with adaptive gain (CGFR/AG). To compare the performance of the proposed algorithm with respect to those other two algorithms, network parameters such as network size and architecture (number of nodes, hidden layers etc), and gain parameters were kept same. For all problems the neural network had one hidden layer with five hidden nodes and sigmoid activation function was used for all nodes.

Prior to training, the weights are initialized to small random values. The reason to initialize weights with small values is to prevent saturation (where one or more hidden nodes is highly active or inactive for all patterns and therefore insensitive to the training process) and random to break symmetry [20]. In order to reduce statistical fluctuation, all algorithms were tested using the same initial weights that were initialized randomly from range [0, 1] and received the input patterns for training in the same sequence. Toolbox default values were used for the heuristic parameters, of the above algorithms, unless stated otherwise. For the proposed of comparison and hence there were no particular reason for their choice, all tested algorithms were fixed with the values of learning rate was 0.3 and momentum term was 0.4. The initial value used for the gain parameter was one.

For each run, the numerical data is showed in two files; (1) the summary results file, and (2) the detail version of the results for that particular algorithm. The number of iterations until convergence is accumulated for each algorithm from which the mean, the standard deviation and the number of failures are calculated.

For each run also, the generalization accuracy of all algorithms is calculated adopted from Watkins [21] paper. Watkins determined generalization accuracy by the inverse of the distance of the simulations result from the real answer, expressed as a percentage of the limits of the range.

$$Accuracy(\%) = \frac{1 - \left| t_k - o_k^L \right|}{UB - LB} * 100 \qquad (15)$$

where UB and LB represent the upper bound and the lower bound. Both are defined based on the type of activation function that is used during the simulations, in this case all simulations is run using sigmoid activation function so UB is defined as one and LB is defined as zero. The final accuracy is reached by taking the mean of all the runs.

V. SIMULATION RESULTS ON BENCHMARK

For each problem, 100 different trials were run, each with different initial random set of weights. For each run, the number of iterations required for convergence is reported. For an experiment of 100 runs, the mean of the number of iterations, the standard deviation, and the number of failures are collected. A failure occurs when the network exceeds the maximum iteration limit; each experiment is run to one thousand iterations except for back propagation which is run to ten thousand iterations; otherwise, it is halted and the run is reported as a failure. Convergence is achieved when the outputs of the network conform to the error criterion as compared to the desired outputs. For each of the database used in the simulations, the web site is given on which they can be downloading with a brief description. Further details are available in the original web sites. Two of the four data sets come from UCI repository of machine learning databases.

A. IRIS CLASSIFICATION PROBLEM

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/iris/iris.data)

This is a classical classification dataset made famous by Fisher [22], who used it to illustrate principles of discriminant analysis. This is perhaps the best-known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The selected architecture of the FNN is 4-5-3 with target error was set as 0.05 and the maximum epochs to 1000.

TABLE I
THE SUMMARY OF CPU TIME NEEDED TO CONVERGE FOR IRIS CLASSIFICATION PROBLEM

	IRIS classification problem (target error=0.05)		
	Mean	CPU time(s)/Epoch	Total CPU time(s) to converge
traincgf	70	7.52×10^{-2}	3.83
CGFR	39	4.59×10^{-2}	1.93
CGFR/AG	29	4.85×10^{-2}	1.42

TABLE II
THE DETAIL VERSION OF THE RESULTS FROM TABLE I

traincgf			**CGFR**			**CGFR/AG**		
Epochs	**CPU/ epoch**	**Total CPU time(s)**	**Epochs**	**CPU/ epoch**	**Total CPU time(s)**	**Epochs**	**CPU/ epoch**	**Total CPU time(s)**
44	0.12	5.22	25	0.05	1.16	37	0.04	1.66
44	0.12	5.44	73	0.05	3.83	29	0.04	1.28
44	0.11	4.97	24	0.05	1.13	38	0.05	1.72
-	-	-	26	0.04	1.14	32	0.04	1.41
261	0.04	9.72	30	0.05	1.39	34	0.05	1.53
27	0.15	3.92	42	0.05	1.95	28	0.04	1.25
66	0.07	4.78	29	0.05	1.36	26	0.04	1.14
39	0.08	2.94	28	0.05	1.42	22	0.04	0.97
70	0.05	3.70	34	0.05	1.58	28	0.04	1.19
88	0.07	5.75	34	0.04	1.50	28	0.04	1.22
77	0.05	3.94	41	0.05	1.91	21	0.04	0.88
102	0.05	4.70	33	0.04	1.47	35	0.04	1.55
620	0.03	21.31	36	0.04	1.59	36	0.05	1.64

70	0.06	4.05	38	0.06	2.17	33	0.04	1.48
47	0.07	3.31	54	0.05	2.69	20	0.04	0.86
16	0.16	2.63	32	0.04	1.39	38	0.04	1.70
44	0.07	3.28	39	0.05	1.81	36	0.05	1.72
87	0.04	3.64	26	0.04	1.16	27	0.05	1.23
18	0.13	2.25	43	0.05	2.00	22	0.05	1.02
40	0.07	2.99	29	0.04	1.25	22	0.04	0.95
98	0.04	4.06	37	0.05	1.81	24	0.04	1.06
41	0.07	2.83	29	0.04	1.27	26	0.04	1.16
38	0.07	2.53	-	-	-	27	0.04	1.19
141	0.04	5.52	37	0.06	2.06	25	0.04	1.08
23	0.16	3.73	28	0.04	1.23	26	0.04	1.13
20	0.14	2.72	40	0.04	1.80	20	0.04	0.84
20	0.12	2.33	30	0.04	1.31	28	0.05	1.36
100	0.07	6.89	26	0.04	1.11	35	0.05	1.58
13	0.24	3.16	27	0.04	1.19	29	0.05	1.55
89	0.05	4.78	27	0.04	1.17	33	0.04	1.48
24	0.11	2.63	30	0.04	1.30	20	0.04	0.88
66	0.06	3.94	27	0.04	1.17	27	0.04	1.20
44	0.16	6.97	114	0.07	7.95	26	0.05	1.17
13	0.19	2.47	36	0.04	1.58	35	0.05	1.61
87	0.04	3.88	31	0.04	1.34	38	0.05	1.73
31	0.08	2.55	33	0.04	1.44	26	0.04	1.16
37	0.08	3.03	30	0.04	1.33	32	0.04	1.42
16	0.14	2.19	24	0.04	1.03	34	0.04	1.52
36	0.09	3.08	31	0.04	1.34	42	0.05	1.92
38	0.08	3.22	25	0.04	1.08	28	0.05	1.27
13	0.25	3.22	29	0.05	1.31	24	0.04	1.03
44	0.07	3.02	33	0.04	1.47	28	0.04	1.22
-	-	-	229	0.06	12.89	22	0.04	0.95
130	0.04	5.64	26	0.04	1.13	36	0.04	1.58
48	0.06	3.05	27	0.04	1.14	33	0.05	1.49
130	0.04	4.59	24	0.04	1.02	23	0.04	0.98
13	0.17	2.27	24	0.04	1.05	48	0.05	2.20
51	0.06	2.98	24	0.04	1.03	28	0.04	1.22
37	0.07	2.44	44	0.05	2.02	56	0.05	2.66
33	0.06	2.11	43	0.06	2.45	58	0.05	2.77
59	0.07	3.98	27	0.04	1.17	21	0.04	0.89
37	0.08	2.97	39	0.04	1.72	21	0.04	0.91
23	0.09	2.05	46	0.06	2.64	37	0.05	1.78
44	0.08	3.42	32	0.04	1.36	24	0.05	1.13
46	0.07	3.02	30	0.04	1.31	33	0.05	1.58
36	0.09	3.22	259	0.06	14.64	38	0.05	1.89
65	0.08	5.33	26	0.04	1.11	24	0.06	1.33
22	0.11	2.45	28	0.04	1.22	28	0.06	1.55
132	0.04	5.20	41	0.06	2.38	33	0.06	1.86
156	0.03	5.23	25	0.04	1.08	22	0.06	1.23
44	0.06	2.70	46	0.06	2.67	-	-	-
89	0.04	3.50	25	0.04	1.06	28	0.06	1.58
173	0.03	5.38	44	0.04	1.94	27	0.05	1.48
175	0.03	5.27	39	0.04	1.72	26	0.05	1.42
102	0.04	3.99	81	0.07	5.38	29	0.06	1.66
51	0.06	3.30	28	0.04	1.22	26	0.05	1.38
80	0.05	3.61	28	0.04	1.19	20	0.05	1.09
17	0.14	2.33	24	0.04	1.03	25	0.06	1.38
23	0.09	2.12	41	0.04	1.80	27	0.06	1.52
34	0.09	3.11	27	0.04	1.17	23	0.05	1.23
14	0.20	2.77	36	0.05	1.64	27	0.06	1.55
47	0.07	3.19	32	0.05	1.59	34	0.06	1.89
46	0.06	2.77	22	0.05	1.11	29	0.05	1.59
34	0.06	2.11	43	0.05	2.05	22	0.05	1.19
192	0.03	5.81	27	0.04	1.19	25	0.06	1.41
-	-	-	41	0.05	1.92	25	0.05	1.34
29	0.11	3.13	32	0.04	1.42	29	0.06	1.61
130	0.03	4.42	28	0.04	1.23	21	0.06	1.20
111	0.03	3.12	26	0.04	1.09	25	0.05	1.36
60	0.05	3.27	32	0.04	1.36	25	0.06	1.38
54	0.05	2.95	28	0.05	1.38	24	0.06	1.34
31	0.08	2.63	38	0.06	2.17	31	0.05	1.69
-	-	-	28	0.05	1.33	22	0.05	1.20
44	0.07	3.30	39	0.05	2.02	34	0.06	1.91
54	0.06	3.17	33	0.05	1.67	32	0.06	1.78
12	0.19	2.28	47	0.04	2.08	19	0.06	1.08
23	0.11	2.53	24	0.04	1.03	27	0.05	1.47
-	-	-	54	0.05	2.47	23	0.05	1.24
61	0.05	2.91	29	0.05	1.34	30	0.06	1.69
68	0.05	3.28	32	0.05	1.45	20	0.06	1.11
15	0.15	2.30	45	0.06	2.61	31	0.06	1.75
87	0.04	3.88	-	-	-	28	0.06	1.61
88	0.04	3.72	25	0.05	1.13	35	0.06	1.97
89	0.04	3.61	28	0.04	1.19	35	0.06	1.97
101	0.04	3.88	55	0.06	3.33	-	-	-
132	0.03	3.46	24	0.04	1.03	32	0.06	1.81
-	-	-	64	0.05	3.08	29	0.06	1.64
131	0.03	4.41	46	0.04	2.05	20	0.05	1.08
134	0.03	4.56	44	0.04	1.95	26	0.06	1.44
130	0.03	4.16	34	0.04	1.50	24	0.06	1.36
70	**0.075**	**3.83**	**39**	**0.045**	**1.93**	**29**	**0.049**	**1.42**

Table (I) illustrates the benefit of using the proposed method in comparison the conjugate gradient method The proposed algorithm reached the target error after only about 29 epochs as opposed to the standard CGFR at about 39 epochs and clearly we see that there is an improvement ratio, nearly 2.4, for the number of epochs compare to neural network toolbox, and almost 2.6749 for the convergence time. Table (II) clearly shows that even though the proposed method required more CPU time per epoch to calculate the function of gain value but it converged more faster as compared to other methods. This is because the proposed method improved the search direction for each iteration which reduced the total number of epochs.

Figure 1 shows that all algorithms performed almost the same accuracy in generalizing the testing data. The neural network toolbox '*traincgf*' performed worst as the standard deviation for epochs is the highest as compared to others. The number of failures for the proposed method is also small as compared to '*traincgf*'. It can also be seen that '*traincgf*' failed to converge at certain trials and performed a wide range of epoch (higher standard deviation) to reach the target error.

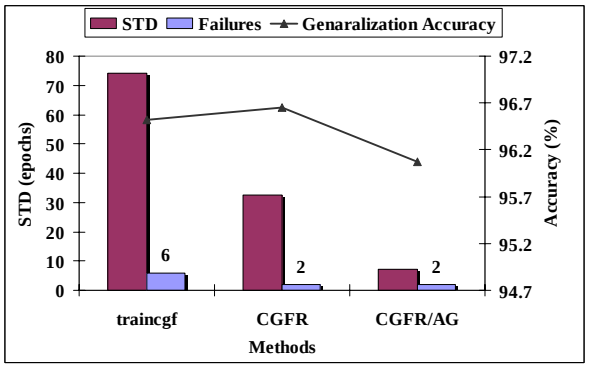

Fig. 1 Summary of algorithms performance on IRIS problem

B. 7 BIT PARITY PROBLEM

(http://homepages.cae.wisc.edu/~ece539/data/parity7r-training data)
(http://homepages.cae.wisc.edu/~ece539/data/parity7t-testing data)

The parity problem is also one of the most popular initial testing tasks and very demanding classification for neural network to solve, because the target-output changes whenever a single bit in the input vector changes and this makes generalization difficult and learning does not always converge easily [23]. The selected architecture of the FNN is 7-5-1. The target error has been set to 0.1 and the maximum epochs to 2000.

TABLE III
THE CPU TIME NEEDED TO CONVERGE FOR 7 BIT PARITY PROBLEM

	7 bit parity (target error=0.1)		
	Number of Epochs	CPU time(s)/Epoch	Total time of converge
traincgf	283	6.06×10^{-2}	10.9697
CGFR	148	7.00×10^{-2}	10.7365
CGFR/AG	114	7.56×10^{-2}	8.7778

Table (III) shows that the proposed algorithm exhibit very good average performance in order to reach target error with only 114 epochs as opposed to the standard CGFR at about 148 epochs and '*traincgf*' with 283 epochs. The result clearly shows that the new algorithm outperforms other two algorithms with an improvement ratio, nearly 1.3, with respect to total convergence time.

TABLE IV
THE DETAIL VERSION OF THE RESULTS FROM TABLE III

traincgf			CGFR			CGFR/AG		
Epochs	CPU/ epoch	Total CPU time(s)	Epochs	CPU/ epoch	Total CPU time(s)	Epochs	CPU/ epoch	Total CPU time(s)
831	0.03	25.24	179	0.08	13.55	59	0.08	4.56
197	0.06	10.98	70	0.07	4.92	64	0.07	4.20
281	0.04	10.84	102	0.08	8.53	78	0.07	5.28
369	0.04	14.14	346	0.07	24.39	154	0.07	10.47
57	1.58	90.01	208	0.07	14.80	65	0.07	4.70
192	0.47	90.67	248	0.08	18.98	90	0.10	9.33
186	0.05	8.75	87	0.08	6.53	107	0.10	10.80
1928	0.03	51.72	109	0.07	7.52	78	0.08	6.13
49	0.11	5.23	237	0.07	16.12	86	0.07	6.24
47	0.06	2.72	83	0.06	5.23	65	0.08	5.09
185	0.03	6.05	444	0.07	32.06	63	0.07	4.25
35	0.10	3.67	62	0.07	4.48	165	0.07	11.72
110	0.04	4.34	220	0.07	14.61	107	0.11	11.61
326	0.03	10.42	125	0.08	10.24	142	0.08	11.75
1153	0.03	32.53	61	0.07	4.05	268	0.07	18.55
-	-	-	143	0.08	11.11	90	0.11	9.91
141	0.05	6.74	100	0.06	6.44	113	0.11	11.97
47	0.08	3.61	230	0.07	16.53	79	0.08	6.09
463	0.03	13.83	108	0.07	8.09	74	0.08	5.69
553	0.02	13.81	74	0.07	5.55	81	0.07	6.06
94	0.04	4.05	80	0.07	5.88	61	0.07	4.48
185	0.04	6.80	207	0.07	13.46	99	0.09	9.31
64	0.09	5.92	66	0.07	4.58	121	0.11	12.77
333	0.03	9.52	239	0.07	16.56	171	0.08	13.89
1189	0.03	29.88	103	0.08	8.11	59	0.08	4.44
158	0.05	8.00	96	0.07	7.11	185	0.07	13.25
67	0.06	3.99	78	0.07	5.56	50	0.07	3.58
67	0.05	3.45	92	0.07	6.45	81	0.07	5.77
60	0.05	2.97	49	0.07	3.63	59	0.07	4.06
103	0.05	5.58	80	0.07	5.73	147	0.07	10.59
39	0.08	2.95	68	0.07	4.92	97	0.10	9.67
694	0.03	19.92	146	0.07	10.67	115	0.11	12.23
60	0.07	4.00	143	0.07	10.55	72	0.08	5.73
188	0.03	6.11	367	0.07	25.69	158	0.07	11.41
960	0.03	29.05	71	0.08	5.56	66	0.07	4.53
121	0.04	4.94	135	0.08	10.38	156	0.07	10.89
405	0.03	13.22	305	0.08	23.44	79	0.06	5.09
500	0.03	16.13	105	0.07	7.14	98	0.10	9.98
47	0.05	2.50	-	-	-	96	0.10	9.19
188	0.03	5.88	74	0.07	4.84	63	0.07	4.50
46	0.08	3.56	413	0.07	29.55	129	0.08	10.56
102	0.03	3.52	72	0.07	5.11	88	0.07	6.31
571	0.03	17.09	369	0.08	30.58	112	0.07	8.34
141	0.04	4.95	82	0.08	6.16	68	0.07	4.84
94	0.05	4.55	76	0.07	5.22	163	0.07	11.38
43	0.08	3.55	156	0.08	11.97	160	0.07	11.00
475	0.03	12.53	447	0.07	33.44	193	0.07	13.70
-	-	-	75	0.07	4.99	76	0.07	5.36
526	0.03	15.63	111	0.06	6.94	320	0.07	23.53
1381	0.03	35.00	116	0.07	8.36	117	0.08	9.58
104	0.04	4.02	102	0.07	7.05	97	0.09	9.03
340	0.03	10.25	109	0.08	8.23	87	0.08	6.81
650	0.03	19.34	119	0.06	7.72	161	0.07	11.11
-	-	-	92	0.08	7.00	200	0.07	13.41
152	0.04	5.47	126	0.08	9.50	126	0.10	12.33
97	0.04	3.61	93	0.07	6.45	65	0.07	4.45
120	0.04	4.55	98	0.07	6.63	115	0.08	9.03
105	0.04	3.91	118	0.08	9.19	229	0.08	17.95
54	0.05	2.95	119	0.07	7.94	156	0.07	11.00
748	0.03	23.08	55	0.07	3.80	207	0.07	14.44
33	0.07	2.36	484	0.07	35.97	88	0.07	6.14
38	0.06	2.45	-	-	-	107	0.07	8.00
42	0.06	2.55	85	0.07	6.03	142	0.09	12.36
278	0.03	8.70	87	0.07	6.17	77	0.05	3.83
372	0.03	10.94	93	0.07	6.36	111	0.09	10.45
263	0.04	9.99	91	0.07	6.30	60	0.05	3.03
48	0.06	2.80	282	0.07	20.16	70	0.05	3.74
125	0.05	6.03	67	0.09	5.75	70	0.07	4.67
60	0.09	5.34	80	0.08	6.34	121	0.09	10.66
105	0.04	4.17	88	0.08	7.08	87	0.09	7.98
495	0.03	16.06	640	0.08	50.70	41	0.07	2.83
322	0.04	13.47	82	0.07	5.59	104	0.09	9.06
98	0.07	6.50	261	0.07	18.05	83	0.08	6.31
140	0.04	5.66	86	0.07	5.98	80	0.07	5.45
325	0.03	9.83	71	0.06	4.60	93	0.09	8.06
45	0.06	2.72	96	0.08	7.25	86	0.07	5.73
-	-	-	90	0.07	6.16	79	0.07	5.67
508	0.03	13.02	206	0.07	13.92	-	-	-
266	0.03	7.20	96	0.07	6.44	57	0.10	5.55
178	0.04	7.70	121	0.07	9.06	72	0.09	6.55
106	0.03	3.50	109	0.07	7.32	105	0.09	9.34
1241	0.03	31.77	96	0.08	7.22	83	0.07	5.91
155	0.03	5.00	108	0.07	7.83	242	0.07	18.11
356	0.02	8.75	265	0.07	19.17	74	0.07	5.20
185	0.04	6.78	92	0.07	6.44	81	0.07	5.58
69	0.05	3.38	84	0.08	6.48	232	0.07	15.09
42	0.06	2.41	144	0.07	10.58	96	0.09	8.97
101	0.03	3.42	137	0.08	10.30	75	0.07	5.13
281	0.02	6.91	121	0.07	8.36	74	0.07	4.86
99	0.03	3.44	238	0.07	16.74	49	0.06	3.17
126	0.03	4.19	70	0.06	4.44	65	0.07	4.28
139	0.03	4.30	-	-	-	428	0.07	31.27
324	0.03	8.59	114	0.07	7.92	-	-	-
463	0.03	13.03	115	0.08	9.23	61	0.07	4.13
231	0.03	6.52	114	0.08	8.81	46	0.07	3.23
186	0.03	5.44	161	0.07	10.88	146	0.07	10.27
509	0.03	13.22	75	0.08	5.64	66	0.08	5.00
-	-	-	119	0.08	9.11	305	0.07	21.36
49	0.07	3.22	109	0.06	7.03	241	0.07	17.80
91	0.03	3.11	259	0.08	20.27	151	0.08	11.52
283	**0.061**	**10.97**	**148**	**0.070**	**10.74**	**114**	**0.070**	**8.78**

From Figure 2, we find that the proposed method can provide essentially the same generalization results as those of the standard conjugate gradient and neural network toolbox. With lower standard deviation and lower number of failures as we can see from Figure 2, the convergence of the proposed method is not only faster than other two but also much more stable. It should also be pointed out that although all algorithms particularly CGFR and CGFR/AG demonstrate similar generalization percentage during testing, the correctly and incorrectly classified vectors in CGFR can be correctly classified by the CGFR/AG and vice versa. This is due to the improved strategies involved in CGFR/AG in obtaining the optimal search direction for each iteration.

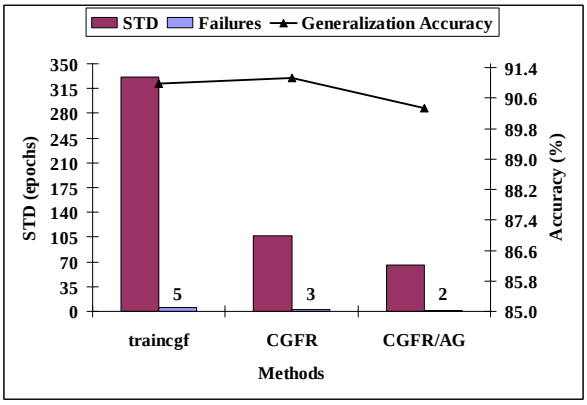

Fig. 2 Summary of algorithms performance on 7-bit parity problem

C. WISCONSIN BREAST CANCER PROBLEM

This dataset was created based on the 'breast cancer Wisconsin' problem dataset from UCI repository of machine learning databases from Dr. William H. Wolberg [24]. This problem tries to diagnosis of breast cancer by trying to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. The selected architecture of the FNN is 9-5-2. The target error is set as to 0.015 and the maximum epochs to 2000.

TABLE V
THE CPU TIME NEEDED TO CONVERGE FOR BREAST CANCER PROBLEM [23]

	Breast cancer problem (target error=0.015)		
	Number of Epochs	CPU time(s)/Epoch	Total time of converge
traincgf	71	$7.15x10^{-2}$	3.7843
CGFR	65	$5.02x10^{-2}$	3.3184
CGFR/AG	39	$4.28x10^{-2}$	1.5501

TABLE VI
THE DETAIL VERSION OF THE RESULTS FROM TABLE V

traincgf			**CGFR**			**CGFR/AG**		
Epochs	**CPU/ epoch**	**Total CPU time(s)**	**Epochs**	**CPU/ epoch**	**Total CPU time(s)**	**Epochs**	**CPU/ epoch**	**Total CPU time(s)**
68	0.09	6.34	33	0.05	1.81	62	0.02	1.11
65	0.06	3.89	62	0.06	3.72	24	0.05	1.09
-	-	-	34	0.05	1.86	12	0.04	0.52
46	0.06	2.77	35	0.06	2.11	19	0.05	0.97
69	0.04	2.94	62	0.06	3.84	19	0.05	0.98
122	0.06	7.45	-	-	-	18	0.05	0.83
12	0.17	2.08	44	0.05	2.11	53	0.05	2.53
141	0.04	5.05	46	0.12	5.33	34	0.06	2.12
75	0.05	3.41	38	0.06	2.11	66	0.03	2.28
23	0.10	2.33	19	0.05	1.00	17	0.04	0.75
47	0.06	2.86	62	0.06	3.55	70	0.04	2.55
74	0.05	3.55	15	0.05	0.78	45	0.02	1.12
-	-	-	53	0.06	2.97	47	0.05	2.16
75	0.04	3.11	9	0.05	0.45	45	0.05	2.05
65	0.09	5.91	62	0.06	3.59	14	0.05	0.66
129	0.04	4.61	70	0.06	4.11	8	0.05	0.42
123	0.04	4.78	50	0.06	2.80	23	0.05	1.12
110	0.04	4.11	39	0.05	2.14	21	0.04	0.94
17	0.15	2.63	67	0.06	4.23	34	0.03	1.12
45	0.06	2.75	64	0.05	3.52	36	0.05	1.88
30	0.08	2.49	55	0.04	2.23	22	0.04	0.98
34	0.08	2.78	74	0.06	4.09	16	0.05	0.73
45	0.07	3.11	33	0.05	1.74	7	0.04	0.30
56	0.07	4.10	78	0.07	5.34	14	0.05	0.69
-	-	-	121	0.05	6.45	16	0.06	1.00
41	0.06	2.63	-	-	-	21	0.04	0.94
315	0.03	9.50	63	0.05	3.44	-	-	-
39	0.07	2.63	67	0.06	4.16	36	0.05	1.64
55	0.09	5.22	62	0.05	3.34	20	0.04	0.89
10	0.24	2.44	41	0.05	2.25	23	0.04	1.03
89	0.04	3.58	65	0.06	3.75	19	0.04	0.84
66	0.05	3.05	26	0.05	1.41	18	0.05	0.83
28	0.10	2.92	68	0.06	3.92	28	0.05	1.42
19	0.12	2.19	45	0.08	3.77	64	0.03	2.16
69	0.05	3.78	128	0.06	7.78	33	0.05	1.50
132	0.04	4.66	21	0.05	1.11	22	0.05	1.00
156	0.03	5.23	62	0.06	3.53	23	0.05	1.11
41	0.07	3.02	124	0.06	7.53	111	0.02	2.12
63	0.05	3.24	63	0.06	3.61	25	0.08	2.11
40	0.11	4.30	17	0.05	0.89	18	0.05	0.83
19	0.11	2.08	125	0.06	7.64	54	0.05	2.61
78	0.06	4.41	22	0.05	1.17	41	0.05	1.92
65	0.07	4.52	38	0.05	2.08	63	0.05	3.09
36	0.10	3.74	36	0.06	2.32	30	0.05	1.38
187	0.04	7.41	66	0.06	3.81	18	0.05	0.84
66	0.05	3.47	17	0.05	0.88	71	0.05	3.56
67	0.05	3.39	-	-	-	66	0.03	2.22
45	0.06	2.88	63	0.06	3.63	56	0.02	1.12
133	0.04	5.55	63	0.06	3.63	13	0.05	0.59
20	0.12	2.41	82	0.06	4.81	24	0.04	1.08
-	-	-	66	0.05	3.56	71	0.04	2.78
33	0.08	2.53	19	0.04	0.84	44	0.05	2.12
126	0.04	4.92	74	0.05	3.67	56	0.05	2.73
34	0.08	2.72	126	0.05	6.55	39	0.05	1.83
67	0.05	3.06	54	0.05	2.56	33	0.06	2.00
139	0.03	4.72	67	0.05	3.34	31	0.05	1.44
123	0.06	7.45	124	0.05	6.47	65	0.03	2.11
11	0.24	2.61	42	0.05	1.95	63	0.04	2.22
14	0.15	2.13	45	0.05	2.09	33	0.05	1.55
43	0.06	2.69	19	0.04	0.84	63	0.03	1.99
44	0.06	2.83	195	0.05	10.45	67	0.03	2.12
28	0.13	3.59	121	0.04	5.23	37	0.05	1.91
132	0.04	5.13	73	0.05	3.52	74	0.03	2.21
20	0.12	2.41	171	0.04	6.73	18	0.06	1.08
22	0.10	2.30	233	0.03	6.45	18	0.05	0.81
20	0.16	3.27	45	0.08	3.43	64	0.03	2.09
66	0.05	3.58	72	0.05	3.36	60	0.04	2.22
128	0.03	4.45	63	0.05	2.92	13	0.04	0.58
32	0.07	2.25	34	0.04	1.52	22	0.04	0.97
189	0.04	7.17	66	0.03	2.01	7	0.04	0.28
16	0.13	2.09	77	0.03	2.12	14	0.04	0.63
72	0.05	3.39	17	0.04	0.74	28	0.05	1.27
13	0.16	2.12	87	0.05	4.23	23	0.12	2.77
67	0.05	3.09	62	0.05	2.88	36	0.05	1.70
213	0.04	8.68	30	0.04	1.33	63	0.03	2.09
129	0.04	4.78	19	0.04	0.83	16	0.04	0.70
121	0.03	4.12	65	0.05	2.98	13	0.04	0.56
32	0.09	2.86	36	0.04	1.61	18	0.05	0.83
32	0.09	2.99	18	0.04	0.77	23	0.04	1.00
65	0.05	3.28	66	0.05	3.56	71	0.04	3.15
65	0.05	3.41	65	0.03	2.22	69	0.03	2.11
67	0.06	3.75	125	0.05	5.95	12	0.04	0.53
17	0.19	3.23	74	0.05	3.42	67	0.05	3.36
76	0.05	3.75	63	0.05	2.92	70	0.04	2.77
70	0.04	3.14	88	0.05	4.66	56	0.02	1.12
32	0.10	3.34	62	0.05	2.86	63	0.03	1.99
77	0.05	3.49	63	0.05	2.95	69	0.03	2.01
71	0.07	5.31	19	0.04	0.83	62	0.05	3.05
141	0.03	4.58	130	0.05	6.22	14	0.04	0.63
42	0.06	2.53	65	0.03	2.12	15	0.04	0.67
30	0.09	2.75	121	0.05	6.11	35	0.05	1.63
32	0.10	3.23	68	0.05	3.66	17	0.04	0.75
135	0.04	4.81	124	0.05	5.89	130	0.02	3.19
66	0.05	3.25	87	0.03	2.90	78	0.03	2.21
26	0.11	2.78	124	0.05	5.97	64	0.03	1.98
25	0.10	2.45	62	0.05	2.84	22	0.05	1.00

223	0.04	8.57	21	0.04	0.92	-	-	-
44	0.07	2.89	135	0.05	6.45	10	0.04	0.44
70	0.05	3.52	34	0.04	1.52	75	0.04	2.88
87	0.05	4.09	13	0.04	0.55	67	0.03	2.12
71	**0.06**	**3.78**	**65**	**0.05**	**3.32**	**39**	**0.04**	**1.55**

Table (V) and (VI) shows average number of iterations as well as the average CPU time taken by all three algorithms in reaching the target error. Observe that the proposed method (CGFR/AG) clearly outperforms the classical conjugate gradient method as well as neural network toolbox. Both methods exhibits a linear convergence to reach the target error, but CGFR/AG algorithm takes significant smaller number of iterations. The CGFR/AG training algorithm takes only 39 epochs to reach the target error as compared to CGFR at about 65 epochs and worst for '*traincgf*' that need about 71 epochs to converge. Still the proposed algorithm outperforms other two algorithms with an improvement ratio, nearly 2.5, for the total time of convergence.

As for generalization performance, all algorithms achieved the similar results, but '*traincgf*' had the highest standard deviation of epochs and the highest number of failures which indicate that it is more unstable as compare to the proposed method. This make CGFR/AG method a better choice since it has two failures for 100 different trials.

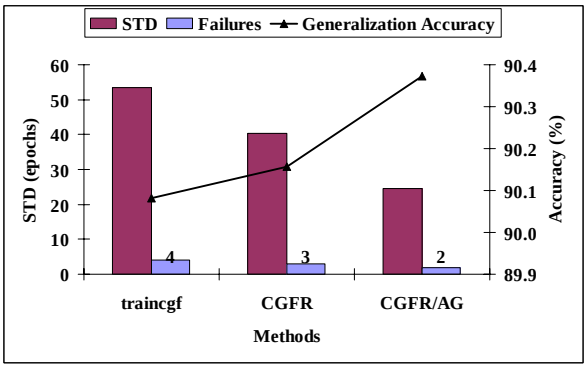

Fig. 3 Summary of algorithms performance on cancer problem

D. DIABETES CLASSIFICATION PROBLEM

(ftp://ftp.ics.uci.edu/pub/machine-learning-databases/diabetes)

This dataset was created based on the 'Pima Indians diabetes' problem dataset from the UCI repository of machine learning database. From the dataset doctors try to diagnose diabetes of Pima Indians based on personal data (age, number of times pregnant) and the results of medical examinations (e.g. blood pressure, body mass index, result of glucose tolerance test, etc.) before decide whether a Pima Indian individual is diabetes positive or not. The selected architecture of the Feed-forward Neural Network is 8-5-2. The target error is set to 0.01 and the maximum epochs to 1000.

TABLE VII
THE CPU TIME NEEDED TO CONVERGE FOR DIABETES PROBLEM [23]

	Diabetes classification problem (target error=0.01)		
	Number of Epochs	CPU time(s)/Epoch	Total time of converge
traincgf	98	$4.77x10^{-2}$	4.0280
CGFR	51	$4.93x10^{-2}$	2.6095
CGFR/AG	40	$4.85x10^{-2}$	2.0121

Table (VII) shows that the CGFR/AG reached the target error after only about 40 epochs as opposed to the standard CGFR at about 98 epochs and clearly we see that there is an improvement ratio, nearly 2.5, for the number of epochs compare to neural network toolbox, and almost 2 for the convergence time.

TABLE VIII
THE DETAIL VERSION OF THE RESULTS FROM TABLE VII

traincgf			**CGFR**			**CGFR/AG**		
Epochs	**CPU/ epoch**	**Total CPU time(s)**	**Epochs**	**CPU/ epoch**	**Total CPU time(s)**	**Epochs**	**CPU/ epoch**	**Total CPU time(s)**
64	0.10	6.28	81	0.05	3.94	29	0.04	1.30
66	0.08	5.30	47	0.05	2.16	38	0.04	1.70
-	-	-	48	0.05	2.19	25	0.05	1.16
104	0.03	3.39	46	0.05	2.09	29	0.04	1.28
77	0.05	3.59	50	0.05	2.28	58	0.05	2.70
76	0.06	4.81	59	0.05	2.78	27	0.05	1.33
106	0.03	3.48	114	0.06	6.92	-	-	-
229	0.03	5.88	58	0.05	2.72	17	0.04	0.75
32	0.07	2.13	35	0.05	1.58	48	0.05	2.17
25	0.09	2.20	67	0.05	3.19	48	0.05	2.20
359	0.02	8.72	65	0.05	3.13	46	0.05	2.09
60	0.05	2.78	57	0.05	2.67	37	0.04	1.66
63	0.04	2.81	24	0.04	1.06	22	0.04	0.97
72	0.04	2.92	39	0.04	1.75	28	0.04	1.25
72	0.04	2.58	26	0.04	1.16	43	0.04	1.92
75	0.04	2.69	54	0.05	2.52	61	0.05	2.89
61	0.04	2.36	66	0.05	3.17	71	0.05	3.39
82	0.04	2.95	46	0.05	2.13	22	0.04	0.97
60	0.04	2.39	92	0.06	5.36	46	0.05	2.13
27	0.08	2.05	54	0.05	2.47	46	0.05	2.08
31	0.07	2.31	47	0.05	2.39	32	0.05	1.61
251	0.03	8.41	46	0.05	2.33	44	0.05	2.17
64	0.05	3.05	34	0.05	1.69	33	0.05	1.63
67	0.04	2.84	47	0.05	2.39	28	0.05	1.38
105	0.04	3.73	27	0.05	1.34	55	0.05	2.89
47	0.06	2.99	39	0.05	1.91	26	0.05	1.31
27	0.08	2.16	77	0.05	4.02	44	0.05	2.31
75	0.04	3.14	40	0.05	2.02	37	0.05	1.88
56	0.05	2.80	58	0.05	3.00	46	0.05	2.41
73	0.04	3.13	38	0.05	1.94	28	0.05	1.39
524	0.03	16.91	54	0.05	2.88	54	0.05	2.80
182	0.04	7.81	43	0.05	2.19	38	0.05	1.94
63	0.08	5.11	46	0.05	2.33	25	0.05	1.23
70	0.05	3.84	45	0.05	2.41	37	0.05	1.77
73	0.05	3.33	36	0.05	1.80	21	0.05	1.06
63	0.05	2.97	32	0.05	1.63	42	0.05	2.08
115	0.04	4.72	55	0.05	2.94	53	0.05	2.77
66	0.05	3.33	49	0.05	2.55	43	0.05	2.22
83	0.04	3.39	66	0.05	3.58	28	0.05	1.49
39	0.07	2.88	37	0.05	1.91	28	0.05	1.39
60	0.06	3.49	-	-	-	53	0.05	2.77
28	0.12	3.41	51	0.05	2.67	41	0.05	2.09
82	0.04	3.34	45	0.05	2.25	54	0.05	2.86
43	0.06	2.70	64	0.05	3.50	-	-	-
144	0.03	4.98	79	0.06	4.63	34	0.07	2.55
-	-	-	66	0.05	3.39	20	0.05	0.99
92	0.04	3.59	45	0.05	2.23	19	0.05	0.92

120	0.04	4.30	54	0.05	2.72	20	0.05	0.95
121	0.04	4.56	56	0.05	2.78	64	0.05	3.50
74	0.05	3.38	76	0.06	4.25	24	0.05	1.20
61	0.04	2.73	51	0.05	2.66	73	0.06	4.13
67	0.05	3.09	54	0.05	2.84	33	0.05	1.66
208	0.03	6.34	43	0.05	2.19	20	0.05	0.99
60	0.05	2.83	61	0.05	3.30	46	0.05	2.09
57	0.05	3.00	85	0.06	4.81	35	0.05	1.77
236	0.03	7.88	37	0.05	1.86	19	0.10	1.91
340	0.03	10.58	40	0.05	2.02	39	0.05	1.91
132	0.04	4.73	71	0.06	3.98	51	0.05	2.56
215	0.03	6.20	82	0.06	4.67	48	0.05	2.45
60	0.05	2.70	23	0.05	1.13	21	0.04	0.91
345	0.03	10.86	51	0.05	2.66	34	0.05	1.74
25	0.14	3.61	49	0.05	2.53	-	-	-
97	0.04	3.73	51	0.05	2.67	33	0.05	1.69
61	0.04	2.61	-	-	-	43	0.05	2.17
61	0.05	2.77	43	0.05	2.23	56	0.05	2.83
-	-	-	64	0.05	3.41	42	0.05	2.14
55	0.05	2.86	57	0.05	3.09	48	0.05	2.53
70	0.04	3.06	57	0.05	3.06	55	0.05	2.84
28	0.08	2.19	20	0.05	0.95	52	0.05	2.61
64	0.05	3.02	64	0.05	3.31	23	0.05	1.14
62	0.04	2.70	51	0.05	2.55	22	0.05	1.08
69	0.05	3.16	42	0.05	2.11	39	0.05	1.99
86	0.04	3.50	55	0.05	2.75	26	0.05	1.27
59	0.05	2.77	40	0.05	1.97	47	0.05	2.41
25	0.09	2.28	43	0.05	2.13	47	0.05	2.42
32	0.07	2.34	29	0.05	1.39	19	0.05	0.92
90	0.04	3.69	102	0.06	5.99	44	0.05	2.19
121	0.04	4.56	37	0.05	1.84	37	0.05	1.86
59	0.05	2.77	47	0.05	2.34	42	0.05	2.13
117	0.04	4.81	42	0.05	2.08	60	0.05	3.27
25	0.08	2.05	-	-	-	28	0.05	1.42
59	0.05	2.97	25	0.05	1.23	36	0.05	1.89
62	0.09	5.38	66	0.05	3.61	-	-	-
-	-	-	52	0.05	2.69	65	0.05	3.55
232	0.03	7.56	26	0.05	1.30	27	0.05	1.31
49	0.06	2.97	27	0.05	1.33	18	0.05	0.89
60	0.04	2.55	51	0.05	2.67	23	0.05	1.14
102	0.04	3.86	36	0.05	1.81	29	0.05	1.42
115	0.03	3.89	55	0.05	2.94	63	0.05	3.39
76	0.04	3.39	45	0.05	2.27	22	0.05	1.09
60	0.05	2.98	21	0.05	1.03	15	0.05	0.73
66	0.05	3.02	41	0.05	2.08	46	0.05	2.27
65	0.04	2.91	61	0.06	3.38	70	0.06	3.91
223	0.03	6.86	30	0.05	1.50	55	0.05	2.92
70	0.04	3.14	39	0.05	1.99	50	0.05	2.59
46	0.06	2.69	50	0.05	2.59	52	0.05	2.59
32	0.07	2.30	54	0.05	2.88	52	0.05	2.72
-	-	-	70	0.06	3.88	114	0.06	6.70
390	0.03	11.89	50	0.05	2.59	40	0.05	2.02
67	0.04	3.00	39	0.05	1.98	31	0.05	1.56
98	**0.05**	**4.03**	**51**	**0.05**	**2.61**	**40**	**0.05**	**2.01**

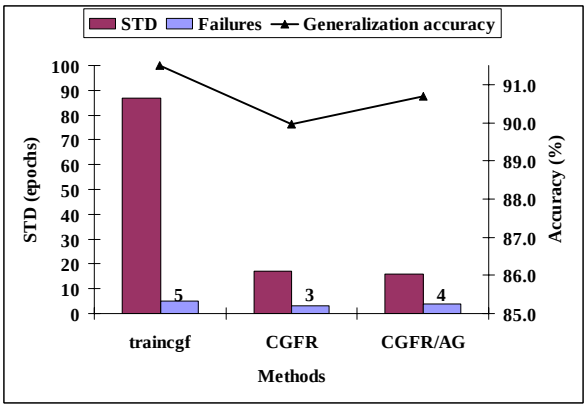

Fig. 4 Summary of algorithms performance on diabetes problem

Figure 4 shows that all algorithms performs similar results in classified the testing data correctly. The CGFR even though has only three failures and with a standard deviation of 17.03 as compared to '*traincgf*', it produced acceptable results. Yet the CGFR/AG outperformed the CGFR with a standard deviation of 15.94 epochs, and with four failures. The small value of the standard deviation says that the 100 different trials were very consistent.

VI. CONCLUSION

In this paper, a new fast learning algorithm for neural networks based on Fletcher-Reeves update with adaptive gain (CGFR/AG) training algorithms is introduced. The proposed method improved the training efficiency of back propagation neural network algorithms by adaptively modifying the gradient search direction. The gradient search direction is modified by introducing the gain value. The proposed algorithm is generic and easy to implement in all commonly used gradient based optimization processes. The simulation results showed that the proposed algorithm is robust and has a potential to significantly enhance the computational efficiency of the training process.

ACKNOWLEDGMENT

The financial support offered by University of Tun Hussein Onn Malaysia (UTHM), towards the research studies of first author is gratefully acknowledged.

REFERENCES

[1] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning internal representations by error propagation.* in D.E. Rumelhart and J.L. McClelland (eds), Parallel Distributed Processing, 1986. 1: p. 318-362.
[2] A. van Ooyen and B. Nienhuis, *Improving the convergence of the back-propagation algorithm.* Neural Networks, 1992. 5: p. 465-471.
[3] M. Ahmad and F.M.A. Salam, *Supervised learning using the cauchy energy function.* International Conference on Fuzzy Logic and Neural Networks, 1992.
[4] Pravin Chandra and Yogesh Singh, *An activation function adapting training algorithm for sigmoidal feedforward networks.* Neurocomputing, 2004. 61: p. 429-437.
[5] R.A. Jacobs, *Increased rates of convergence through learning rate adaptation.* Neural Networks, 1988. 1: p. 295-307.
[6] M.K. Weir, *A method for self-determination of adaptive learning rates in back propagation.* Neural Networks, 1991. 4: p. 371-379.
[7] X.H. Yu, G.A. Chen, and S.X. Cheng, *Acceleration of backpropagation learning using optimized learning rate and momentum.* Electronics Letters, 1993. 29(14): p. 1288-1289.
[8] Bishop C. M., *Neural Networks for Pattern Recognition*. 1995: Oxford University Press.
[9] R. Fletcher and M. J. D. Powell, *A rapidly convergent descent method for nlinimization.* British Computer J., 1963: p. 163-168.
[10] Fletcher R. and Reeves R. M., *Function minimization by conjugate gradients.* Comput. J., 1964. 7(2): p. 149-160.
[11] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systerns.* J. Research NBS, 1952. 49: p. 409.
[12] HUANG H.Y., *A unified approach to quadratically convergent algorithms for function minimization.* J. Optim. Theory Appl., 1970. 5: p. 405-423.
[13] Thimm G., Moerland F., and Emile Fiesler, *The Interchangeability of Learning Rate an Gain in Back propagation Neural Networks.* Neural Computation, 1996. 8(2): p. 451-460.
[14] Holger R. M. and Graeme C. D., *The Effect of Internal Parameters and Geometry on the Performance of Back-Propagation Neural Networks.* Environmental Modeling and Software, 1998. 13(1): p. 193-209.

[15] Eom K. and Jung K., *Performance Improvement of Back propagation algorithm by automatic activation function gain tuning using fuzzy logic.* Neurocomputing, 2003. 50: p. 439-460.
[16] Rumelhart D. E., Hinton G. E., and Williams R. J., *Learning internal representations by back-propagation errors.* Parallel Distributed Processing, 1986. 1 (Rumelhart D.E. et al. Eds.): p. 318-362.
[17] C.H. Chen and Hongtao Lai, *An empirical study of the Gradient Descent and the Conjugate Gradient backpropagation neural networks.* IEEE, 1992: p. 132-135.
[18] Curtis F. Gerald and Patrick O. Wheatley, *Applied Numerical Analysis. Seventh Edition*. 2004: Addison-Wesley.
[19] L.Prechelt, *Proben1 - A set of Neural Network Bencmark Problems and Benchmarking Rules.* Technical Report 21/94, 1994: p. 1-38.
[20] Adrian J. Sheperd, *Second Order Methods for Neural Networks-Fast and Reliable Training Methods for Multi-layer Perceptrons*, ed. J.G. Taylor. 1997: Springer. 143.
[21] Dave Watkins, *Clementine's Neural Networks Technical Overview*. Technical Report, 1997.
[22] Fisher R.A., *The use of multiple measurements in taxonomic problems.* Annals of Eugenics, 1936. 7: p. 179 -188.
[23] Erik Hjelmas and P.W. Munro, *A comment on parity problem.* Technical Report, 1999: p. 1-7.
[24] Mangasarian O. L. and W.W. H., *Cancer diagnosis via linear programming.* SIAM News, 1990. 23(5): p. 1-18.

Nazri Mohd Nawi received his B.S. degree in Computer Science from University of Science Malaysia (USM), Penang, Malaysia. His M.Sc. degree in computer science was received from University of Technology Malaysia (UTM), Skudai, Johor, Malaysia. He has been working toward his Ph.D. degree in Mechanical Engineering department, University of Wales Swansea. At present, his research interests are in optimisation, data mining and neural networks

Rajesh S. Ransing is a Senior Lecturer at the University of Wales Swansea. He received his B.E in Mechanical Engineering from University of Poona, Pune, India, he received his M.E. from Indian Institute of Science, Bangalore, India and his Ph.D. from University of Wales Swansea in 1989, 1992 and 1996. He has published over 40 papers in refereed journals, one patent and has organised many symposiums, workshop, and conferences on this topic. He is also on the executive committee of Natural Computing Applications Forum. His research interests are in the fields of data analysis, optimisation methods, natural computing and nano-meso scale computation.

Meghana R. Ransing received the B.E. in Computer Engineering from University of Poona, Pune, India and the Ph.D. degree in engineering from the University of Wales Swansea in 1995 and 2003. She is currently a senior research officer in school of engineering at the University of Wales Swansea. She has published over 10 papers in refereed journals. Her research interests are in data analysis and natural computing.