World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:8, 2008

An Improved Learning Algorithm based on the Conjugate Gradient Method for Back Propagation Neural Networks

N. M. Nawi, M. R. Ransing, and R. S. Ransing

Open Science Index, Computer and Information Engineering Vol:2, No:8, 2008 publications.waset.org/446.pdf

Abstract**—**The conjugate gradient optimization algorithm usually used for nonlinear least squares is presented and is combined with the modified back propagation algorithm yielding a new fast training multilayer perceptron (MLP) algorithm (CGFR/AG). The approaches presented in the paper consist of three steps: (1) Modification on standard back propagation algorithm by introducing gain variation term of the activation function, (2) Calculating the gradient descent on error with respect to the weights and gains values and (3) the determination of the new search direction by exploiting the information calculated by gradient descent in step (2) as well as the previous search direction. The proposed method improved the training efficiency of back propagation algorithm by adaptively modifying the initial search direction. Performance of the proposed method is demonstrated by comparing to the conjugate gradient algorithm from neural network toolbox for the chosen benchmark. The results show that the number of iterations required by the proposed method to converge is less than 20% of what is required by the standard conjugate gradient and neural network toolbox algorithm.

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

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 the network weights. Despite the general success of back-propagation in learning the neural networks, several major deficiencies are still needed to be solved. First, the back-propagation algorithm will get trapped in local minima especially for non-linearly separable problems [2] such as the XOR problem [3]. Having trapped into local minima, back-propagation may lead to failure in finding a global optimal solution. Second, the convergence rate of back-propagation is still too slow even if learning can be achieved.

Furthermore, the convergence behavior of the back-propagation algorithm depends very much on the choices of initial values of connection weights and the parameters in the algorithm such as the learning rate and the momentum.

Improving the training efficiency of neural network based algorithm 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 energy function [4-6]; (ii) selection of dynamic learning rate and momentum [7-9].

Later, as summarized by Bishop[10] 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[11] and the Fletcher-Reeves[12] that improve the conjugate gradient method of Hestenes and Stiefel[13] and the family of Quasi-Newton algorithms proposed by Huang[14].

Among BP learning speed-up algorithms, those using the “gain variation” term are among the easiest to implement. The gain variation term controls the steepness of the activation function. It has been recently shown that a BP algorithm using gain variation term in an activation function converges faster than the standard BP algorithm [15-17]. However, it was not noticed that gain variation term can modify the local gradient to give an improved gradient search direction for each training iteration.

This paper suggests that a simple modification to the initial search direction can also substantially improve the training efficiency of almost all major optimization methods. It was discovered that if the initial 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 irrespective of the optimization algorithm used. Furthermore the proposed method is robust, easy to compute, and easy to implement into well known nonlinear conjugate gradient algorithms as will be shown later in next section.

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 initial search direction by changing the gain value adaptively for each node.

N. M. Nawi is with 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).

M. R. Ransing and R. S. Ransing are with Civil and Computational Engineering Centre, University of Wales, Singleton Park, Swansea, SA2 8PP, United Kingdom (phone:+440-1792 295902; fax: :+440-1792 295903; e-mail: R.S.Ransing@swansea.ac.uk).

The following iterative algorithm is proposed by the authors for changing the initial search direction using a gain value.

Initialize the weight vector with random values and the vector of gain values with one. Repeat the following steps 1,2 and 3 on an epoch-by-epoch basis until the given error minimization criteria are satisfied.

Step 1 *By introducing gain value into activation function, calculate the gradient for weight vector by using Equation (6), and gradient for gain value by using Equation (9).*

Step 2 *Calculate the gradient descent on error with respect to the weights and gains values.*

Step 3 *Use the gradient weight vector and gradient of gain calculated in step 1 to calculate the new weight vector and vector of new gain values for use in the next epoch.*

In general, the objective of a learning process in neural network is to find a weight vector w which minimizes the difference between the actual output and the desired output. Namely,

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

Suppose for a particular input pattern o^0 and let the input layer is layer 0. The desired output is the teacher pattern $t = [t_1 ... t_n]^T$, and the actual output is o_k^L, where L denotes the output layer. Define an error function on that pattern as,

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

The overall error on the training set is simply the sum, across patterns, of the pattern error E.

Consider a multilayer feed forward neural network(FNN) [1] has one output layer and one input layer with 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.* Let o_k^s be the activation of the k^{th} node of layer s, and let $o^s = [o_1^s ... o_n^s]^T$ be the column vector of the activation values in the layer s and the input layer as layer 0. Let w_{ij}^s be the weight on the connection from the i^{th} node in layer $s-1$ to the j^{th} node in layer s, and let $w_j^s = [w_{1j}^s ... w_{nj}^s]^T$ be the column vector of weights from layer $s-1$ to the j^{th} node of layer s. The net input to the j^{th} node of layer s is defined as $net_j^s = (w_j^s, o^{s-1}) = \sum_k w_{j,k}^s o_k^{s-1}$, and let $net^s = [net_1^s ... net_n^s]^T$ be the column vector of the net input values in layer s. The activation of a node is given by a function of its net input,

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

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

By introducing "gain variation" term in this activation function, then only updating formulas for δ_1^s are changed while others are the same as the standard back propagation.

To simplify the calculation, taken from the equation (2) we then can perform gradient descent on E with respect to w_{ij}^s. The chain rule yields

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

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

where $\delta_j^s = -\frac{\partial E}{\partial net_j^s}$. In particular, the first three factors of (4) indicate that

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

As we noted that the iterative formula (5) for δ_1^s is the same as standard back propagation [18] except for the appearance of the value gain. By combining (4) and (5) yields the learning rule for weights:

$$\Delta w_{ij}^s = \eta \frac{\partial E}{\partial w_{ij}^s} = \eta \delta_j^s o_j^{s-1} \tag{6}$$

where η is a small positive constant called 'step length' or 'learning rate' and the search direction or gradient vector is $d = \Delta w_{ij}^s = g$.

In this approach, at step n is the calculation for gradient of error $g^{(n)}$ is modified by including the variation of gain value to yield

$$d^{(n)} = \Delta w_{ij}^{s(n)} (c_j^{s(n)}) = g^{(n)} (c_j^{s(n)}) \tag{7}$$

The gradient descent on error with respect to the gain can also be calculated by using the chain rule as previously described; it is easy to compute as

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

Then the gradient descent rule for the gain value becomes,

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

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

Open Science Index, Computer and Information Engineering Vol:2, No:8, 2008 publications.waset.org/446.pdf

Open Science Index, Computer and Information Engineering Vol:2, No:8, 2008 publications.waset.org/446.pdf

$$c_j^{new} = c_j^{old} + \Delta c_j^s \qquad (10)$$

III. The Implementation of Proposed Method with Conjugate Gradient Method

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 [10]. Most widely used conjugate gradient algorithms are given by Fletcher and Powel [11] and the Fletcher-Reeves [12]. Both these procedures generate conjugate directions of search and therefore aim to minimize a positive definite quadratic function of n variables in n steps.

Our proposed algorithm known as CGFR/AG begins the minimization process with an initial estimate w_0 and an initial search direction as:

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

Then, for every epoch by using our proposed method in Equation (7) the search direction at (n+1)th iteration is calculated as:

$$d_{n+1} = -\frac{\delta E}{\delta w_{n+1}}(c_{i,n+1}) + \beta_n(c_{i,n})d_n(c_{i,n}) \qquad (12)$$

where the scalar β_n is to be determined by the requirement that d_n and d_{n+1} must fulfil the conjugacy property[10]. There are many formulae for the parameter β_n. In this paper we used the formula introduced by Fletcher and Reeves [12] and is given as:

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

The complete CGFR/AG algorithm works as follows;

Step 1 *Initializing the weight vector randomly, the gradient vector* $g_0 = 0$ *and gain value as one. Let the first search direction* $d_0 = g_0$. *Set* $\beta_0 = 0$, $epoch = 1$ *and* $n = 1$. *Let Nt is the number of weight parameters. Select a convergence tolerence* CT.

Step 2 *At step* n, *evaluate gradient vector* $g_n(c_n)$ *with respect to gain vector* c_n *and calculate gain vector.*

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

Step 4 *Calculate a new search direction:* $d_n = -g_n(c_n) + \beta_{n-1}d_{n-1}$

Step 5 *For the first iteration, check if* $n > 1$ *THEN with the function of gain, update* $\beta_{n+1} = \dfrac{g_{n+1}^T(c_{n+1})g_{n+1}(c_{n+1})}{g_n^T(c_n)g_n(c_n)}$ *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:* $E(w_n + \eta_n^* d_n) = \min_{\lambda \geq 0} E(w_n + \eta_n d_n)$

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:* $d_{n+1} = -g_{n+1}(c_{n+1}) + \beta_n(c_n)d_n$

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

IV. Simulation Results

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 taken from the open literature by Prechelt [19]. 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 Dell, 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 other standard optimization algorithms from the MATLAB neural network toolbox, network parameters such as network size and architecture (number of nodes, hidden layers etc), values for the initial weights 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. 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, all tested algorithms were fixed with the values of learning rate = 0.3 and momentum term = 0.4. The initial value used for the gain parameter was one.

The results of all three algorithms will be presented as table which summarize the performance of the algorithms for simulations that have reached solution. All algorithms were trained with 100 trials, if an algorithm fails to converge, it is considered that it fails to train the FNN, but its epochs, CPU time and generalization accuracy are not included in the statistical analysis of the algorithms.

A. Iris Classification Problem

This is a classical classification dataset made famous by Fisher[20], 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

Open Science Index, Computer and Information Engineering Vol:2, No:8, 2008 publications.waset.org/446.pdf

frequently to this day. The selected architecture of the FNN is 4-5-3 with target error was set as 0.01 and the maximum epochs to 2000.

TABLE I
THE CPU TIME NEEDED TO CONVERGE FOR IRIS CLASSIFICATION PROBLEM [24]

	IRIS classification problem (target error=0.01)		
	Number of Epochs	CPU time(s)/Epoch	Total time of converge
traincgf	69	5.54×10^{-2}	3.8071
CGFR	39	4.90×10^{-2}	1.9146
CGFR/AG	29	4.94×10^{-2}	1.4232

Table I shows that 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.

B. 7 Bit Parity Problem

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 [21]. 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 II
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	273	3.88×10^{-2}	10.6129
CGFR	147	7.26×10^{-2}	10.6496
CGFR/AG	114	7.12×10^{-2}	8.1057

Table II 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 147 epochs and *traincgf* with 273 epochs. The result clearly shows that the new algorithm outperform others two algorithm with an improvement ratio, nearly 1.3, for the total time of converge.

C. Winconsin 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 [22]. 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 III
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	5.34×10^{-2}	3.7883
CGFR	65	5.11×10^{-2}	3.3060
CGFR/AG	39	4.00×10^{-2}	1.5503

In Table III, it is worth noticing that the performance of the CGFR/AG training algorithm since it take only 39 epochs to reach the target error compare to CGFR at about 65 epochs and worst for *traincgf* that need about 71epochs to converge. Still the proposed algorithm outperforms others two algorithms with an improvement ratio, nearly 2.5, for the total time of converge.

D. Diabetes Classification Problem

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 IV
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	97	4.12×10^{-2}	4.0060
CGFR	50	5.16×10^{-2}	2.6000
CGFR/AG	40	5.05×10^{-2}	2.0030

Table IV shows that the new algorithm reached the target error after only about 40 epochs as opposed to the standard CGFR at about 97 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.

V. CONCLUSION

In this paper, 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 initial search direction. The initial 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.

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] Marco Gori and Alberto Tesi, On the problem of local minima in back-propagation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1992. 14(1): p. 76-86.
[3] E.K. Blum, Approximation of Boolean functions by sigmoidal networks: Part I: XOR and other two-variable functions. Neural Computation, 1989. 1(4): p. 532-540.
[4] A. van Ooyen and B. Nienhuis, Improving the convergence of the back-propagation algorithm. Neural Networks, 1992. 5: p. 465-471.
[5] M. Ahmad and F.M.A. Salam, Supervised learning using the cauchy energy function. International Conference on Fuzzy Logic and Neural Networks, 1992.
[6] Pravin Chandra and Yogesh Singh, An activation function adapting training algorithm for sigmoidal feedforward networks. Neurocomputing, 2004. 61: p. 429-437.
[7] R.A. Jacobs, Increased rates of convergence through learning rate adaptation. Neural Networks, 1988. 1: p. 295-307.
[8] M. K. Weir, A method for self-determination of adaptive learning rates in back propagation. Neural Networks, 1991. 4: p. 371-379.
[9] 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.
[10] Bishop C. M., Neural Networks for Pattern Recognition. 1995: Oxford University Press.
[11] R. Fletcher and M. J. D. Powell, A rapidly convergent descent method for nlinimization. British Computer J., 1963: p. 163-168.
[12] Fletcher R. and Reeves R. M., Function minimization by conjugate gradients. Comput. J., 1964. 7(2): p. 149-160.
[13] M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systerns. J. Research NBS, 1952. 49: p. 409.
[14] Huang H.Y., A unified approach to quadratically convergent algorithms for function minimization. J. Optim. Theory Appl., 1970. 5: p. 405-423.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] L. Prechelt, Proben1 - A set of Neural Network Bencmark Problems and Benchmarking Rules. Technical Report 21/94, 1994: p. 1-38.
[20] Fisher R.A., The use of multiple measurements in taxonomic problems. Annals of Eugenics, 1936. 7: p. 179 -188.
[21] Erik Hjelmas and P.W. Munro, A comment on parity problem. Technical Report, 1999: p. 1-7.
[22] Mangasarian O. L. and W.W. H., Cancer diagnosis via linear programming. SIAM News, 1990. 23(5): p. 1-18.
[23] Lutz Prechelt, ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz. 1994.
[24] R. A. Fisher, ftp://ftp.ics.uci.edu/pub/machine-learningdatabases/iris/iris.data. 1988.

Open Science Index, Computer and Information Engineering Vol:2, No:8, 2008 publications.waset.org/446.pdf

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.

Meghana R. Ransing received the B.Sc. 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.

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 30 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.