A WINDOWED CONJUGATE GRADIENT ALGORITHM FOR FAST NONLINEAR ADAPTIVE FILTERING

A. N. Birkett, R. A. Goubran
Department of Systems and Computer Engineering
Carleton University, 1125 Colonel By Drive
Ottawa, Canada, K1S 5B6
Tel: (613) 788-5747, Fax: (613) 788-5727
e-mail: birkett@sce.carleton.ca

ABSTRACT

A modified form of the partial conjugate gradient algorithm is presented which uses a gradient average window to provide a trade-off between convergence rate and complexity which is intermediate between the conventional backpropagation (BP) algorithm and Newton methods that require the storage and calculation of the Hessian of a matrix. Additional simplification is introduced by replacing the optimum step size α_k by a normalized step size $\overline{\alpha}$, in the same manner as the Normalized LMS algorithm. The new algorithm demonstrates improved convergence rates for on-line adaptive filtering with even small choices of window size.

KEYWORDS

Neural Networks, Conjugate Gradient, Adaptive Filtering, Nonlinear System Identification, Echo Cancellation.

1.0 INTRODUCTION

The limitations of the conventional backpropagation algorithm include sensitivity to parameter selection, the uncertainty of finding the global minimum of the error function, and excessively long training times required to obtain a small error output. The later shortcoming, i.e. the slow convergence to either a local or global minimum is the topic addressed in this study. Partial conjugate direction methods can be regarded as being somewhat intermediate between the method of steepest descent (i.e. backpropagation) and Newton's methods, in terms of complexity and convergence properties. Thus they give the designer the option of

improving the convergence rate at the expense of increased complexity. The specific application addressed here is nonlinear adaptive filtering and system identification, where there is a single input and output.

2.0 DESCRIPTION OF LEARNING ALGORITHMS

Consider a multilayer feedforward neural network filter as shown in Figure 3. The input delay line consists of p taps. The basic mechanism behind most supervised learning rules is to update the network weights and bias terms until the mean-squared error between the network output y and desired target signal d is minimized to below a predetermined level. The error signal at the output of a neuron i at time n is defined by;

$$e_i(n) = d_i(n) - y_i(n) \tag{1}$$

where $y_i(n)$ and $d_i(n)$ represent the output and desired signals respectively. The instantaneous cost function E_{inst} at time n is defined as;

$$E_{inst}(n) = e^2(n) = \sum_{i=1}^{N_L} e_i^2(n) \tag{2}$$

which is the instantaneous sum of squared errors of the network for N_L output nodes, in this case equal to one. The total cost function E_{tot} is defined as a sum of the instantaneous cost functions over the full training set;

$$E_{tot} = \sum_{n=1}^{K} E_{inst}(n) \tag{3}$$

where K is the number of training samples in a set. We can define alternate cost functions to be minimized, for example, a partial cost function can be calculated by taking a window n_w of past cost functions calculated using the current weight value $\mathbf{w}(n)$;

$$E_{partial}(n) = \sum_{i=0}^{n_w - 1} E_{inst}(n-i)\Big|_{\mathbf{w}(n)} \tag{4}$$

where $\mathbf{w}(n)$ is a weight supervector consisting all the weights in the network, including bias weights. Specifically we may write the m^{th} order supervector $\mathbf{w}(n)$ as;

$$[\mathbf{w}(n)]^T = [[\mathbf{w}^0(n)]^T, [\mathbf{w}^1(n)]^T, [\mathbf{w}^2(n)]^T, \ldots [\mathbf{w}^L(n)]^T] \quad (5)$$

where $\mathbf{w}^l(n)$ is the weight vector connecting layer l to layer $l+1$ at time (n), m equals the total number of weights in the network and L is the total number of layers in the network. The windowed conjugate gradient algorithm uses a cost function like (4) in the calculation of the gradient for updating the weight vector each iteration.

The supervised learning algorithms described here all attempt to minimize a particular cost function (referred to herein simply as E) by the delta rule [5];

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \frac{\partial E}{\partial \mathbf{w}} \quad (6)$$

The weight vector is incremented at each step towards the optimum weight vector using the negative gradient at that point

2.1 Backpropagation Algorithm

The backpropagation (BP) algorithm is based on propagating errors back to hidden nodes using an instantaneous gradient estimate. Referring to Figure 1, the activation level at the input to the sigmoidal nonlinearity of neuron j in layer l+1 is;

$$s_j^{l+1}(n) = \sum_{i=0}^{N_l} w_{ij}^l(n) u_i^l(n) \quad (7)$$

where $u_i^l(n)$ represents both the output from the previous layer and the input to weight matrix $\mathbf{w}^l$ at time n. The output of node j in layer l is;

$$u_j^l(n) = \varphi(s_j^l(n)) \quad (8)$$

where φ represents the nonlinear activation function, in this case a hyperbolic tangent. The output error $e(n)$ produced at the output layer (i.e. $l=L$) of the network is;

$$\begin{aligned} e(n) &= d(n) - u_1^L(n) \\ &= d(n) - N[\mathbf{w}(n), \mathbf{u}^0(n)] \end{aligned} \tag{9}$$

where $N[\mathbf{w}(n),\mathbf{u}^0(n)]$ is the nonlinear output sequence produced by the input vector $\mathbf{u}^0(n)$. The complete adaptation algorithm can be expressed as follows;

$$w_{ij}^l(n+1) = w_{ij}^l(n) - \eta \delta_j^{l+1}(n) \cdot u_i^l(n) \tag{10}$$

$$\delta_j^l(n) = \begin{cases} -2e(n)f'(s_j^L(n)) & \ldots l = L \\ f'(s_j^l(n)) \cdot \sum_{k=1}^{N_{L+1}} \delta_k^{l+1}(n) \cdot w_{jk}^l(n) & \ldots 1 \le l \le L-1 \end{cases} \tag{11}$$

where;

u^l represents the input vector to a layer l,

w^l_{ij} represents the weight vector connecting node i in layer l to node j in layer $l+1$

s_j^l represents the input activation to node j in layer l

$f'(\,)$ represents the derivative of the sigmoid function

δ_k^l represents the local gradient "delta" term of node k in layer l.

L is the total number of layers in the network

n is the time index

η is the step size parameter

The backpropagation terms are illustrated in Figure 2.

2.2 Full Conjugate Gradient Algorithm

The full conjugate gradient algorithm is based on updating the tap weights with new directions that are "non-interfering", in other words, conjugate to each other. The concept of "non-interfering" directions can be made mathematically explicit by considering a multidimensional function $f(\cdot)$. Let point $\mathbf{x}$ represent the origin of a particular multidimensional system with a set of linearly independent direction vectors $\mathbf{x}_0$,$\mathbf{x}_1$, $\mathbf{x}_2$,...$\mathbf{x}_{m-1}$ which represent the coordinate system. Let $\Delta\mathbf{x}$ be the m by 1 vector representing the distance

from the origin **x** along the direction vectors $\mathbf{x}_i$. Then any function value $f(\mathbf{x}+\Delta\mathbf{x})$can be approximated by its Taylor series as;

$$f(\mathbf{x}+\Delta\mathbf{x}) \approx f(\mathbf{x}) + \sum_i \frac{\partial f(\mathbf{x})}{\partial x_i} x_i + \frac{1}{2}\sum_{ij} \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} x_i x_j + \ldots$$
$$= c + \Delta\mathbf{x}^{\mathrm{T}} \cdot \mathbf{b} + \frac{1}{2}\Delta\mathbf{x}^{\mathrm{T}} \cdot \mathbf{Q} \cdot \Delta\mathbf{x} \qquad (12)$$

where

$$\mathbf{b} \equiv \frac{\partial f(\mathbf{x})}{\partial x_i} = \nabla f(\mathbf{x}) \qquad \mathbf{Q} \equiv \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} = \qquad (13)$$

The matrix **Q** is the $m \times m$ Hessian matrix of the function at **x,** and **b** is the $m \times 1$ gradient of the function at **x**. The approach in the conjugate direction method is to obtain a set of linearly independent direction vectors $\mathbf{x}_0,\mathbf{x}_1, \mathbf{x}_2,\ldots\mathbf{x}_{m-1}$ which are conjugate with respect to **Q** so that the optimum solution vector $\mathbf{x}^*$ minimizes (12). $\mathbf{x}^*$ can be expressed as;

$$\mathbf{x}^* = \alpha_0\mathbf{x}_0 + \alpha_1\mathbf{x}_1 + \ldots + \alpha_{m-1}\mathbf{x}_{m-1} \qquad (14)$$

and the constants are given by [1];

$$\alpha_i = \frac{\mathbf{x}_i^{\mathrm{T}}\mathbf{b}}{\mathbf{x}_i^{\mathrm{T}}\mathbf{Q}\mathbf{x}_i} \qquad (15)$$

The conjugate gradient algorithm determines the appropriate orthogonal set of direction vectors and constants α_i. If the direction vectors are mutually conjugate and linearly independent, then the initial guess **x** will converge to the optimum $\mathbf{x}^*$ after *m* steps, that is $\mathbf{x}_m=\mathbf{x}^*$. The conjugate gradient algorithm can be implemented in a block processing form as in [9], or as an on-line method described in this paper.

We now replace the general vector **x** with direction vector **d** to avoid confusion, and state the full conjugate gradient algorithm as follows:

1) Starting at some arbitrary weight vector w0 compute the gradient g0 at w0.
2) For k=0 to *M*,
   a) $\mathbf{w}_{k+1}=\mathbf{w}_k+\alpha_k\mathbf{d}_k$ where α_k minimizes the cost function E_{inst} according to (2).
   b) compute $\mathbf{g}_{k+1}$, the gradient at $\mathbf{w}_{k+1}$.

c)compute the new direction vector $\mathbf{d}_{k+1}= \mathbf{g}_{k+1}+\beta_k \mathbf{d}_k$, where β_k is given by;

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \quad (16)$$

3) Set $\mathbf{w}_0=\mathbf{w}_M$ and return to step 1.

This form of the algorithm requires a line search in step 2a) which can be computationally intensive. However, by modifying the CGA for a *nonquadratic* function as given in references [1][2] or [3], we can derive the following algorithm that does not require the knowledge of either the Hessian or a line minimization along a particular conjugate direction. This is done by replacing the calculation of the optimum step size with an approximation as follows;

$$\alpha_k = \frac{\mathbf{d}_k^T \mathbf{g}_k}{\mathbf{d}_k^T \mathbf{Q} \mathbf{d}_k} \approx \frac{-\mathbf{g}_k^T \mathbf{d}_k}{\mathbf{d}_k^T (\mathbf{g}_k - \mathbf{p}_k)} \quad (17)$$

where $\mathbf{p}_k$ is the gradient at $(\mathbf{w}_k - \mathbf{g}_k)$. A summary of this algorithm is given below;

Full Conjugate Gradient Algorithm:

For each time sample *n*, perform the following three steps:

Step 1. a) Starting with an initial weight vector $\mathbf{w}_0$*, compute the following;*

$$\mathbf{g}_0 = [\nabla f(\mathbf{w}_0)]^T \quad (18)$$

$$\mathbf{y}_0 = \mathbf{w}_0 - \mathbf{g}_0 \quad (19)$$

$$\mathbf{p}_0 = [\nabla f(\mathbf{y}_0)]^T \quad (20)$$

b) set $\mathbf{d}_0 = -\mathbf{g}_0$ (21)

Step 2. Repeat for k=0,1,...,M-1 where M is the size of the weight and bias vector $\mathbf{w}$;

a) set $\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{d}_k$, where α_k is the optimum step size, defined by; (22)

$$\alpha_k = \frac{-\mathbf{g}_k^T \mathbf{d}_k}{\mathbf{d}_k^T (\mathbf{g}_k - \mathbf{p}_k)} \quad (23)$$

b) Compute the gradients at the new weight vector $\mathbf{w}_{k+1}$

$$\mathbf{g}_{k+1} = [\nabla f(\mathbf{w}_{k+1})]^T \quad (24)$$

$$\mathbf{y}_{k+1} = \mathbf{w}_{k+1} - \mathbf{g}_{k+1} \quad (25)$$

$$\mathbf{p}_{k+1} = [\nabla f(\mathbf{y}_{k+1})]^T \quad (26)$$

c) Unless k=M-1, obtain the new direction vector $\mathbf{d}_{k+1}= -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$,where; (27)

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \qquad (28)$$

and repeat *Step 2 (a)*.

Step 3. Replace $\mathbf{w}_0$ by $\mathbf{w}_M$ and go back to *Step 1*.

The calculation of β_k is done according to the Fletcher-Reeves method rather than the Polak-Ribiere method since it has been shown that it gives smoother convergence towards the optimum weight vector [15].

The penalty for not having to calculate the Hessian matrix is that *two* gradient calculations must be performed for each iteration, one at $\mathbf{w}_{k+1}$ and one at $\mathbf{y}_{k+1}$, however, since the computation of the Hessian matrix is of order $O(M^2)$ and the calculation of a single gradient is of order $O(M)$ [17], the savings are substantial if the network order M is large and partial gradient methods are used.

2.3 Windowed Partial Conjugate Gradient Algorithm

In the CGA algorithm above, it is assumed that the gradient calculations are *true* gradients and that at least m conjugate directions can be calculated. However, this is expensive and sometimes impractical if real time processing is required. For example, in the batch mode of training in neural networks, the performance index is the sum of the squared errors over the *full* training set of input output pairs. In real time filtering and system identification, this is not always possible since we don't have the required memory storage or time to wait for all the training data to be presented before minimizing the performance index. In this case, an on-line method of approximating the gradient is required. If we use an *instantaneous* gradient estimate, as is done in the LMS and backpropagation algorithms, the CGA will terminate in one step [1]. This is because there will not be any more directions conjugate to the initial direction vector. However, a better approximation to the gradient can be obtained by calculating the estimate based on a window of past values of inputs. The next two sections expands on this concept for the linear perceptron (FIR) adaptive structure and then a multilayer perceptron.

2.3.1 Windowed Conjugate Gradient Algorithm for Linear Perceptrons

The simplest network consists of a single linear neuron trained by the LMS algorithm. The LMS algorithm uses an instantaneous value of gradient in place of it ensemble average so that we have the following update equation;

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \frac{1}{2}\mu[\nabla f(\mathbf{w}(n))]^T \\ &\approx \mathbf{w}(n) - \mu e(n)\mathbf{u}(n) \end{aligned} \tag{29}$$

where the true gradient $[\nabla f(\mathbf{w}(n))]^T$ has been replaced by the instantaneous estimate $2e(n)\mathbf{u}(n)$. Note that $\mathbf{u}(n)=[u(n), u(n\text{-}1) \ldots u(n\text{-}m\text{+}1)]^T$ is the tapped delay line consisting of delayed samples of the input, $\mathbf{w}(n)=[w_1(n), w_2(n) \ldots w_m(n)]^T$ is the tap weight vector, $e(n)$ is the error signal, $d(n)$ is the desired signal value at time n, and μ is the step size.

If we construct a gradient estimate by averaging the instantaneous gradient estimates over a specified number n_w of past values, there will be at least n_w linearly independent direction vectors in the gradient estimate. Specifically we replace the instantaneous gradient estimate by a windowed estimate as follows;

$$[\nabla f(\mathbf{w}(n))]^T = \mathbf{g}(n) \approx \left(\frac{2}{n_w}\right)\left[\sum_{i=0}^{n_w-1} \{[\mathbf{w}_0{}^T(n)\mathbf{x}(n-i) - d(n-i)]\mathbf{u}(n-i)\}\right] \tag{30}$$

With this gradient estimate, the CGA will terminate in n_w loops of *Step 2*. Note that the same type of windowed gradient calculation will be used in steps (18),(20),(24), and (26) except that in (20) and (26) the vector **y** is used in place of **w**. Notice that in this case, the gradient vector is obtained by using *past* values of the input vector **u** and desired scalar *d* with the *current* values of the weight vector **w**.

2.3.2 Simplification of the Algorithm

It can be argued that at low values of n_w, the gradient estimates are poor resulting in inappropriate values of the step size α_k. In addition, the computation of α_k in (23) requires $2mn_w$ multiplies and one division for a window of size n_w. This step size can be replaced by a constant value as done in reference [3] or with a nor-

malized step $\bar{\alpha}$ size as proposed here. By removing the calculation of α_k, the calculation of **p** and **y** are also no longer required, thus simplifying the algorithm further. The α_k are therefore replaced by a normalized step size as follows [4];

$$\bar{\alpha} = \frac{\gamma}{\varepsilon + \|\mathbf{u}(n)\|^2} = \frac{\gamma}{\varepsilon + \mathbf{u}^T(n)\mathbf{u}(n)} \quad (31)$$

where

γ is a number between 0 and 2
ε is a small positive number to prevent the denominator from going to zero
$\mathbf{u}(n)$ is the input vector to the transversal filter at time n.

The resulting algorithm obtained by combining the CGA with a windowed gradient calculation and normalized step size is called the Normalized Fast Conjugate Gradient Algorithm (NFCGA). The complete algorithm is summarized below:

Normalized Fast Conjugate Gradient Algorithm:(NFCGA)

For each iteration n, do *Steps 1 2* and *3.*

Step 1.

a) Starting with an initial weight vector $\mathbf{w}_0$, compute the following;

$$\mathbf{g}_0 = [\nabla f(\mathbf{w}_0)]^T = \left(\frac{2}{n_w}\right)\left[\sum_{i=0}^{n_w-1}\{[\mathbf{w}_0{}^T(n)\mathbf{x}(n-i) - d(n-i)]\mathbf{u}(n-i)\}\right] \quad (32)$$

b) set $\mathbf{d}_0 = -\mathbf{g}_0$ (33)

c) compute the normalized step size parameter α according to;

$$\bar{\alpha} = \frac{\gamma}{\varepsilon + \|\mathbf{u}(n)\|^2} = \frac{\gamma}{\varepsilon + \mathbf{u}^T(n)\mathbf{u}(n)} \quad (34)$$

Step 2. Repeat for k=0,1,. n_w-1 where $n_w \le m$

a) set $\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha\mathbf{d}_k$ (35)

b) Compute an estimate of the gradient at $\mathbf{w}_{k+1}$;

$$\mathbf{g}_{k+1} = [\nabla f(\mathbf{w}_{k+1})]^T = \left(\frac{2}{n_w}\right)\left[\sum_{i=1-n_w+1}^{n}\{[\mathbf{w}_{k+1}{}^T(n)\mathbf{x}(i) - d(i)]\mathbf{u}(i)\}\right] \quad (36)$$

c) Unless k=n_w-1, set $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k\mathbf{d}_k$,where; (37)

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \qquad (38)$$

Note that if $\beta_k > 1$, *go directly to Step three.*

Repeat *Step 2 a).*

Step 3. Replace $\mathbf{w}_0$ by $\mathbf{w}_k$ and go back to *Step 1*.

It should be pointed out that checking for $\beta_k > 1$ is a necessary step in the simplified version of this algorithm. As shown in [6], linearly filtered noisy gradient algorithms can provide faster convergence than the gradient estimate (29) however, when $\mathbf{g}_k$ are noisy, it is possible that $\mathbf{g}_{k+1} \approx \mathbf{g}_k$ and the new β_k will be close to or larger than 1. Successive iterations of Step 2 will only serve to move the weight vector away from the optimum value and possibly make the algorithm unstable. For example, Proakis [6] found it necessary to limit the value of $\beta_k < 1$ in a channel equalization experiment and obtained the conditions for stability which can be expressed as follows;

$$0 < \beta_k < 1$$
$$0 < \alpha < \frac{2(1+\beta_k)}{\lambda_{max}} \qquad (39)$$

where λ_{max} is the maximum eigenvalue of the input data. Specifically, the gradient averaging extends the upper limit of the region of stability of α from $2/\lambda_{max}$ to $2(1+\beta_k)/\lambda_{max}$ but β_k must be kept below 1.

2.3.3 Windowed Conjugate Gradient Backpropagation Algorithm

The windowed conjugate gradient algorithm for a neural network is similar to the algorithm presented in the previous section. The differences are (1) the network is nonlinear (2) the errors must be computed for hidden layers and not just the output layer (3) the previous values of the *hidden layer* outputs must be retained as well as the output layers in order to compute the windowed gradient.

We compute the gradient based on the average squared error of a *window* of training input/output pairs, rather than the *full* set of input/output pairs (as is done in the batch training mode) or a *single* input/output

pair (as is done in the individual update backpropagation mode). The errors however are backpropagated to *previous* layers in the same way as the conventional BP algorithm. The important point is that the window is moved for each new sample of the input that comes in i.e. it is a *sliding* window of past input/output pairs. The corresponding algorithm is termed the Windowed Fast Conjugate Gradient Algorithm (WFCGA) and is a nonlinear extension of the fast linear CGA presented in the previous section. The WFCGA is summarized below;

Windowed Fast Conjugate Gradient Algorithm: (WFCGA)

Initialization: Set weights and biases to random values.
For each iteration n, do *Steps 1 2* and *3.*
Step 1. a) Starting with an initial weight vector $\mathbf{w}_0$*, compute the following;*

$$\mathbf{g}_0 = [\nabla f(\mathbf{w}_0)]^T = \left(\frac{2}{n_w}\right)\left[\sum_{i=0}^{n_w-1} \mathbf{g}_{inst}(n-i)\Big|_{\mathbf{w}_0(n),\, \mathbf{u}^0(n-i),\, d(n-i)}\right] \quad (40)$$

where $\mathbf{g}_{inst}(n\text{-}i)$ is the instantaneous gradient calculated with the current network weight vector $\mathbf{w}_0(n)$ and past inputs $\mathbf{u}^0(n\text{-}i)$ and $d(n\text{-}i)$. Both $\mathbf{g}_{inst}(n\text{-}i)$ and $\mathbf{w}_0(n)$ are vectors of length M, where M is the total number of weights in the network.

b) set $\mathbf{d}_0 = -\mathbf{g}_0$ (41)

c) compute the normalized step size parameter α according to;

$$\bar{\alpha} = \frac{\gamma}{\varepsilon + \|\mathbf{u}(n)\|^2} = \frac{\gamma}{\varepsilon + \mathbf{u}^T(n)\mathbf{u}(n)} \quad (42)$$

Note that α could be replaced by a fixed step size here if desired;

Step 2. Repeat for k=0,1,. n_w-1 where $n_w \le m$

a) set $\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha\mathbf{d}_k$ (43)

b) Compute an estimate of the gradient at $\mathbf{w}_{k+1}$;

$$\mathbf{g}_{k+1} = [\nabla f(\mathbf{w}_{k+1})]^T = \left(\frac{2}{n_w}\right)\left[\sum_{i=0}^{n_w-1} \mathbf{g}_{inst}(n-i)\Big|_{\mathbf{w}_{k+1}(n),\, \mathbf{u}^0(n-i),\, d(n-i)}\right] \quad (44)$$

c) Unless k=n_w-1, set $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k\mathbf{d}_k$,where; (45)

$$\beta_k = \frac{\mathbf{g}_{k+1}^T\mathbf{g}_{k+1}}{\mathbf{g}_k^T\mathbf{g}_k} \quad (46)$$

Note that if $\beta_k > 1$, *go directly to Step three.*

Repeat *Step 2 a).*

Step 3. Replace $\mathbf{w}_0$ by $\mathbf{w}_k$ and go back to *Step 1*.

The calculation of the instantaneous gradient $\mathbf{g}_{inst}(n-i)\big|_{\mathbf{w}_{k+1}(n),\,\mathbf{u}^0(n-i),\,d(n-i)}$ is done by performing the following steps;

$$e(n-i) = d(n-i) - N[\mathbf{w}_{k+1}(n), \mathbf{u}^0(n-i)] \quad (47)$$

$$g_{ij}^{l}(n-i) = \delta_j^{l+1}(n-i) \cdot u_i^l(n-i) \quad (48)$$

where;

$$\delta_j^l(n-i) = \begin{cases} -2e(n-i)f'(s_j^L(n-i)) & \dots l = L \\ f'(s_j^l(n-i)) \cdot \sum_{k=1}^{N_{L+1}} \delta_k^{l+1}(n-i) \cdot w_{jk}^l(n) & \dots 1 \le l \le L-1 \end{cases} \quad (49)$$

Refer to Figure 2 for an illustration of terms. Note that the gradient vector $\mathbf{g}_{inst}(n\text{-}i)$ has the same size as the supervector $\mathbf{w}_k(n)$ and is formed by placing individual $g^l_{ij}(n\text{-}i)$ in much the same way that $\mathbf{w}(n)$ is formed in (5).

2.4 Complexity

The choice of n_w =1 implies no averaging in the gradient estimate and the WFCGA reverts to the BP algorithm. For higher values of n_w the complexity approaches that of algorithms that use the second derivative for obtaining the optimum step size and direction which have complexity $O(m^2)$[17]. The complexity of the WFCGA is $O(mn_w^2)$ since in *Step 2*, the weights are updated n_w times per iteration and the calculation of the averaged gradient is $O(mn_w)$.

3.0 SIMULATION

In this section, we apply the WFCGA to the identification of a nonlinear system constructed by generating a signal x which is then hard limited and then convolved with an exponentially decaying 50 tap impulse. The input signal x is obtained by a first order autoregressive (AR) process according to the equation $x(n)=0.9x(n-1)+0.2v(n)$ where $v(n)$ is a unit variance white noise sequence. The hard limiter has a linear region up to 0.5, beyond which the output is clipped with a limiting function which has a slope of 0.2. The system is illustrated in Figure 3. The results illustrated in Figure 4 show that for the AR input, the WFCGA converges at a rate much faster than the conventional BP algorithm, depending on the size of the gradient averaging window n_w. The larger the choice of n_w, the higher the convergence rate. The final misadjustment is approximately -18dB for all cases.

3.1 Application

In this section we attempt to identify a nonlinear loudspeaker in order to improve the performance of an acoustic echo canceller (AEC) as encountered in handsfree telephony. The AEC structure shown in Figure 5 must be capable of identifying and tracking not only the reflected signals from the room, i.e. its Acoustic Impulse Response (AIR), but also of modelling the nonlinear loudspeaker response Conventional AECs utilize a linear adaptive transversal filter to model the room impulse response and cancel the echo signal. The Normalized Least Mean Square (NLMS) algorithm [4] is the baseline by which performance of alternative models is measured but it is incapable of reducing nonlinear distortion. A measure of the AEC performance is the Echo Return Loss Enhancement (ERLE) which is defined as;

$$ERLE(dB) = \lim_{N\to\infty}\left[10\log\frac{E[p^2(n)]}{E[e^2(n)]}\right] \cong 10\log\left[\frac{\sigma^2_p}{\sigma^2_e}\right] \quad (50)$$

where σ^2_p and σ^2_e refer to the variances of the primary and error signals respectively and E is the statistical expectation operator.

The primary and reference data are collected by recording high volume signals in an anechoic chamber. The volume is 100 dB Sound Pressure Level (SPL) as measured at 0.5 meters from the loudspeaker. The microphone is placed 15 cm. from the loudspeaker output. The signals are sampled at 16 kHz and are later transferred to a computer for off-line analysis. Two adaptive filter structures were tested to identify the system (i) a 150 tap linear transversal filter trained using the NLMS algorithm (ii) a 3 layer TDNN with 150 input taps trained with both the CG and WFCGA. The experimental results shown in Figure 6 show the results for all cases. The NLMS has fast convergence but is incapable of obtaining an ERLE of greater than 19 dB due to the nonlinear loudspeaker. The TDNN trained with the BP algorithm is capable of identifying the system more effectively and achieves 25 dB ERLE but the initial convergence is much slower than the NLMS algorithm. Training the TDNN using the WFCGA with a window size n_w=5 results in convergence speed equivalent to the NLMS structure as well as obtaining 24 dB ERLE.

4.0 CONCLUSIONS

This paper has introduced a variant of the partial conjugate gradient algorithm based on using a gradient averaging window and normalized step size to replace the optimum step size. The WFCGA has reduced complexity as compared to the regular CGA but still has fast convergence for low values of gradient averaging window. Simulations illustrate that the WFCGA has faster convergence than the BP depending on the size of the gradient averaging window. Experimental results obtained for a nonlinear AEC illustrate the effectiveness of the WFCGA in improving the initial convergence rate.

REFERENCES

[1] D. G. Leunberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, 1973

[2] M. R. Hestenes, *Conjugate Direction Methods in Optimization*, Springer-Verlag, 1980.

[3] G. K. Boray, M. D. Srinath, “Conjugate Gradient Techniques for Adaptive Filtering”, *IEEE Trans. on Circ. and Sys*. Vol. CAS-1, pp. 1-10, Jan. 1992.

[4] S. Haykin, *Adaptive Filter Theory*, Englewood Cliffs, NJ: Prentice Hall, 1986

[5] S. Haykin, *Neural Networks: A comprehensive foundation*, Macmillan Publishing Co., Englewood Cliffs, NJ: , 1994

[6] J. Proakis, “Channel Identification for High Speed Digital Communications”, *IEEE Trans. Automat. Control*, Vol. AC-19, pp. 916-922, Dec. 1974

[7] C. Charlambous, “Conjugate Gradient Algorithms for Efficient Training of Artificial Neural Networks”, *Proc. IEEE*, Vol. 139, No. 3, pp. 301-310, 1992.

[8] L. E. Scales, *Introduction to Non-linear Optimization*, New York, Springer-Verlag, 1985.

[9]J. S. Lim, C. K. Un, “Conjugate Gradient Algorithm for Block FIR Adaptive Filtering”, *Electronic Letters*, , Vol. 29, No. 5, pp.428-429, March, 1993.

[10] R. A. Jacobs, “Increased rates of convergence through learning rate adaptation”, *Neural Networks*, Vol. 1, No. 4, pp. 295-308, 1988.

[11] M. S. Moller, “A scaled conjugate gradient algorithm for fast supervised learning” *Neural Networks*, Vol. 6, No. 4, pp. 525-534, 1993.

[12] X. Hu, G. Chen, S. Cheng, “Dynamic Learning Rate Optimization of the Backpropagation Algorithm”, *IEEE Trans. Neural Networks*, Vol. 6, No. 3, pp. 669-677, May 1995.

[13] A. G. Parlos, B. Fernandez, A. Atiya, J. Muthusami, W. Tsai, “An Accelerated Learning Algorithm for Multilayer Perceptron Networks” *IEEE Trans. Neural Networks*, Vol. 5, No. 3, pp. 493-497, May 1994.

[14] M. T. Hagan, M.B. Menhaj , “Training Feedforward Networks with the Marquardt Algorithm”, *IEEE Trans. Neural Networks*, Vol. 5, No. 6, pp. 989-997, Nov. 1994.

[15] S. Ergenzinger, E. Thomsen, “ An accelerated learning algorithm for multilayer perceptrons: optimization layer by layer”, *IEEE Trans. Neural Networks*, Vol. 6, No. 1, pp 31-42, Jan. 1995.

[16] R. Battiti, “First and second order methods for learning: Between steepest descent and Newtons method”, *Neural Computation*, Vol. 4, No. 2, pp 141-166, 1992.

[17] W. Buntine, A. A. Weigend,” Computing Second Derivative in Feed-Forward Networks: A review”; *IEEE Trans. Neural Networks*, Vol. 5, No. 3, pp. 481-488, May 1994.

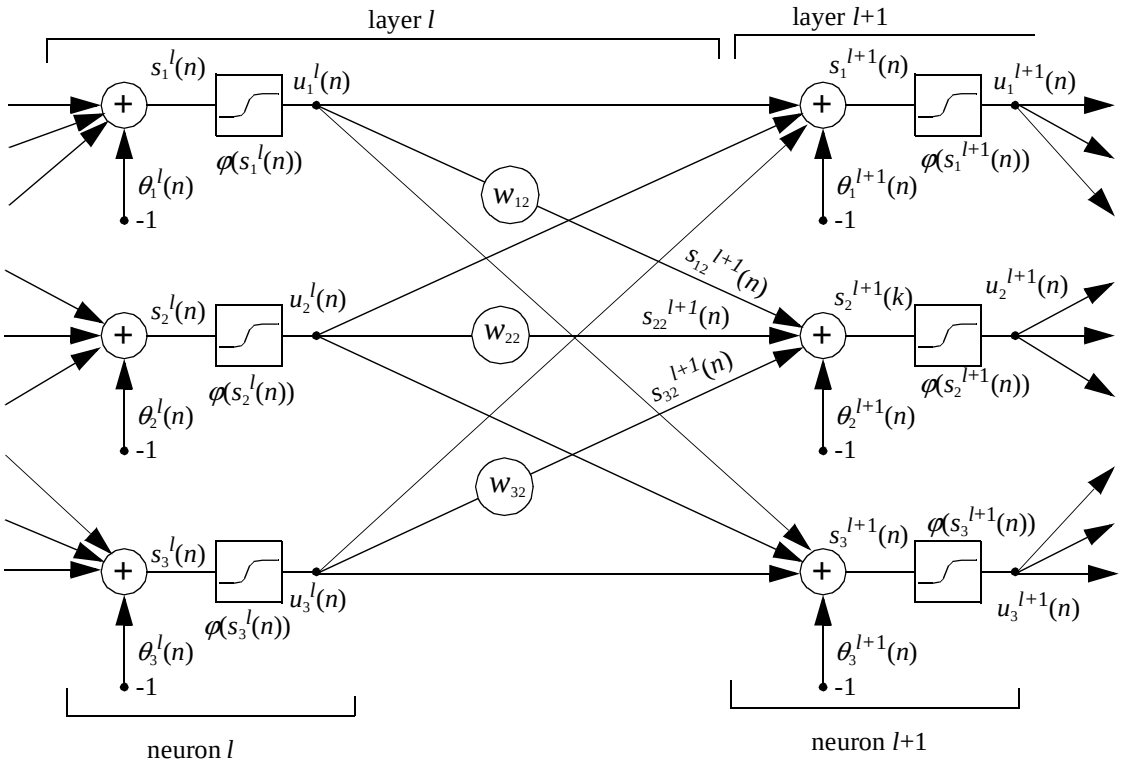

FIGURE 1. Forward signal propagation. The single weight values θ are the bias weights.

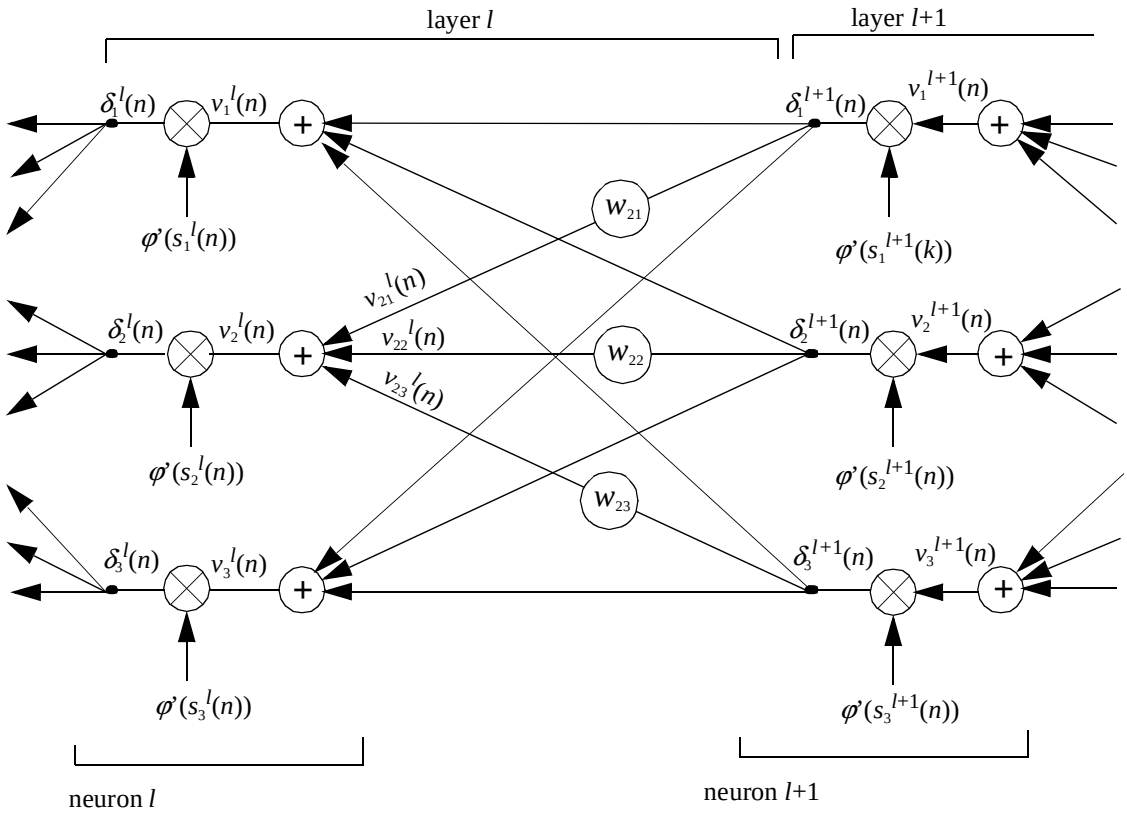

FIGURE 2. Backward error propagation.

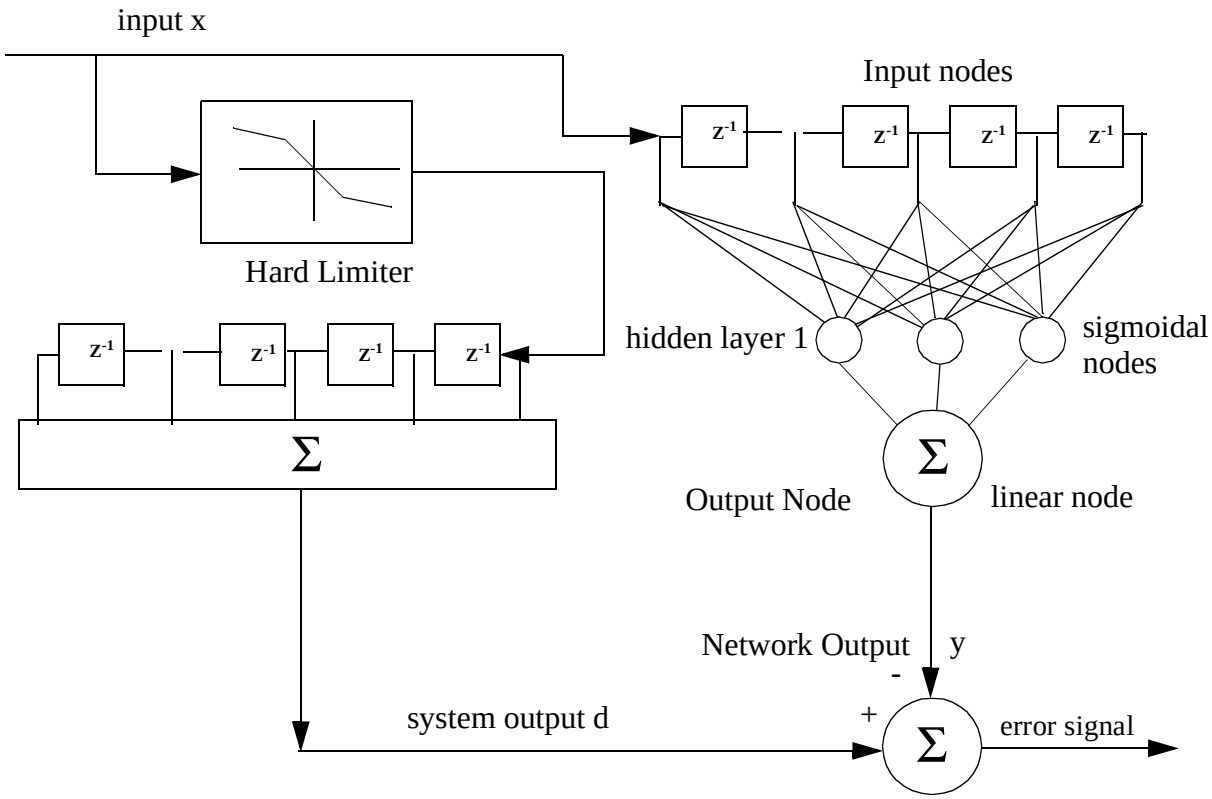

FIGURE 3. System identification model. The system to be identified is a fixed nonlinearity consisting of a linear portion up the value of 0.5 followed by a squashing function of slope =0.2. The output of this nonlinearity is then passed through a dispersive channel consisting of an exponentially decaying random noise impulse of length 50 taps. The neural network consists of a 50 tap input delay line followed by one or two hidden layers. The input x is a first order autoregressive sequence $x(n)=0.9x(n-1) + 0.2v(n)$ where $v(n)$ is a white noise sequence with variance =1.0.

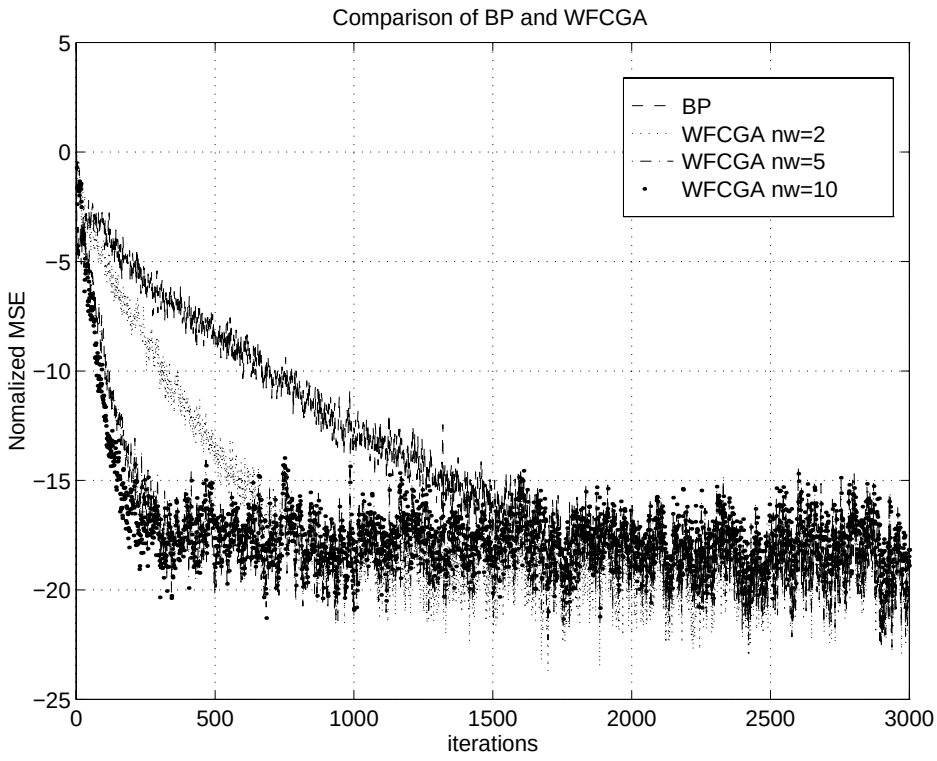

FIGURE 4. Comparison of the normalized MSE using the BP and WFCGA algorithms with n_w=2, 5 and 10 for the system identification model of Figure 3. A first order autoregressive signal is used to model the input signal x. Two hundred independent trials are used in the averaging process.

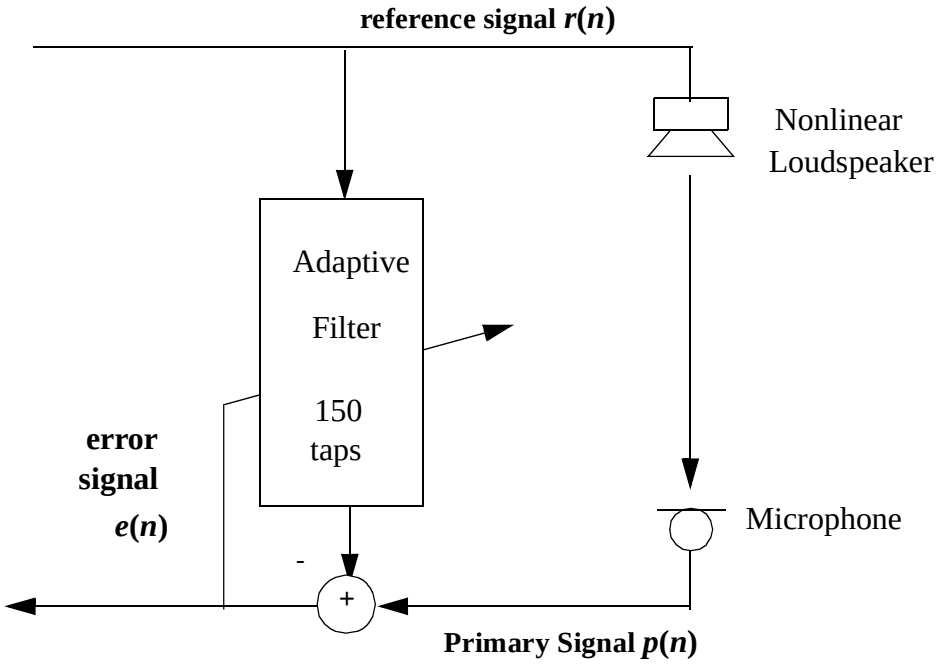

FIGURE 5. Echo cancellation system utilizing a linear transversal filter or a tapped delay line neural network for identification of the nonlinear loudspeaker.

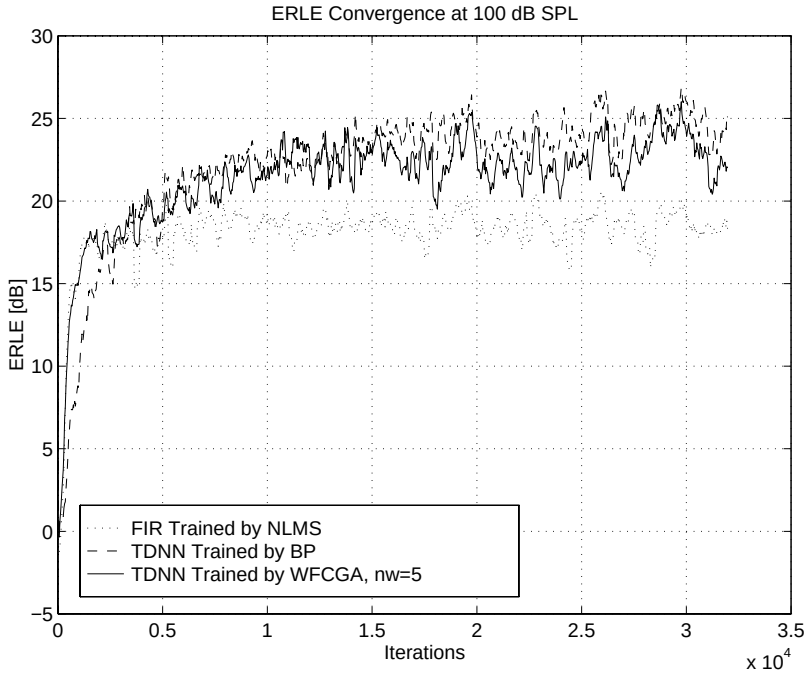

FIGURE 6. Experimental results comparing the converged ERLE curves of a 150 tap FIR structure trained using the NLMS algorithm with that of a TDNN trained using both the conventional BP algorithm and the WFCGA.