IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN:2319-765X. Volume 10, Issue 1 Ver. II. (Feb. 2014), PP 53-61
www.iosrjournals.org

The Method of Steepest Descent for Feedforward Artificial Neural Networks

Muhammad Hanif[1], Md. Jashim Uddin[2] and Md Abdul Alim[3]
[1]*Associate Professor , Department of Applied Mathematics, Noakhali Science and Technology University, Noakhali 3814, Bangladesh.*
[2]*Lecturer , Department of Applied Mathematics, Noakhali Science and Technology University, Noakhali 3814, Bangladesh.*
[3]*Assistant Professor, Chittagong University, Chittagong 4331, Bangladesh.*

Abstract: *In this paper, we implement the method of Steepest Descent in single and multilayer feedforward artificial neural networks. In all previous works, all the update weight equations for single or multilayer feedforward artificial neural networks has been calculated by choosing a single activation function for various processing unit in the network. We, at first, calculate the total error function separately for single and multilayer feedforward artificial neural networks and then calculate the three new update weight equations for taking different activation function in different processing unit separately single and multilayer feedforward artificial neural networks. An example is given to show usefulness of this implementation.*

Keywords: *Feedforward Artificial Neural Networks, Back propagation Learning, Activation Functions, Training.*

I. Introduction

Feed-forward artificial neural networks (FNN) [1-3] have been widely used for various tasks, such as pattern recognition, function approximation, dynamical modeling, data mining, and time series forecasting. The training of FNN is mainly undertaken using the back-propagation (BP) [4-5] based learning. The Backpropagation training algorithm for training feed-forward networks was developed by Paul Werbos[Paul Werbos84], and later by Parker[Parker85] and Rummelhart [Rummelhart 94] . This type of network configuration is the most common in use, due to its ease of training. It is estimated that over 80% of all neural network projects in development use backpropagation. The reason for the name "backprogagation" is that the output errors are "propagated back" from the output layer to the hidden layer, and are used in the update equation for the hidden layer weights. There are two phases in its learning cycle, one to propagate the input pattern through the network and the other to adapt the output, by changing the weights in the network. It is the error signals that are back propagated in the network operation to the hidden layer(s). The portion of the error signal that a hidden-layer neuron receives in this process is an estimate of the contribution of a particular neuron to the output error. Adjusting on this basis the weights of the connections, the squared error, or some other metric, is reduced in each cycle and finally minimized, if possible. A Back-Propagation network consists of at least three layers of units: an input layer, at least one intermediate hidden layer, and an output layer. Typically, units are connected in a feed-forward fashion with input units fully connected to units in the hidden layer and hidden units fully connected to units in the output layer. When a Back- Propagation network is cycled, an input pattern is propagated forward to the output units through the intervening input-to-hidden and hidden-to-output weights.

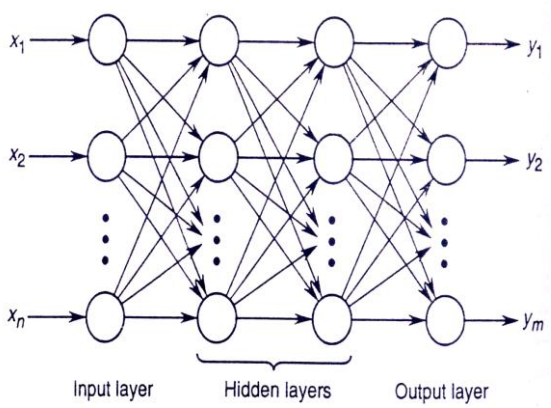

Figure 1 Structure of a feedforwarded artificial neural network.

The method of steepest descent [7-8] is so popular among many mathematicians: it is very simple, easy to use, and each repetition is fast. But the biggest advantage of this method lies in the fact that it is guaranteed to find the minimum through numerous times of iterations as long as it exists. However, this method also has some big flaws: If it is used on a badly scaled system, it will end up going through an infinite number of iterations before locating the minimum, and since each of steps taken during iterations are extremely small, thus the convergence speed is pretty slow, this process can literally take forever! Al- though a larger step size will increase the convergence speed, but it could also result in an estimate with large error. For example, if there is a long and narrow valley in the error surface, the component of the gradient in the direction that points along the base of the valley is very small while the component along the valley walls is quite large. This results in motion more in the direction of the walls even though we have to move a long distance along the base and a small distance along the walls. In this paper, we implement this method, separately, in single and multilayer feedforward artificial neural networks and have been more useful results.

Single-Layer Network

Consider a single layer feedforward neural networks of k neurons with k-th output as shown in the Figure 2.

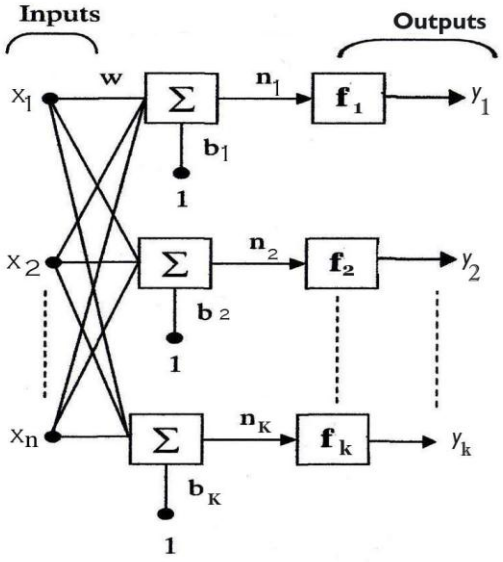

Figure 2. Single layer feedforward neural networks

There are n inputs x_i, where $i = 1.....,n.$, $\mathbf{W} = (w_{ij}) = \begin{pmatrix} w_{11}\ w_{12}\ .....w_{1n} \\ w_{21}\ w_{22}\ .....w_{2n} \\ \ldots\ \ldots\ \ldots \\ w_{k1}\ w_{k2}\ .....w_{kn} \end{pmatrix}$, is the weighted matrix with kxn size is used to denote the strength of the connection from the *i*th input to the *j*th processing element., b_i, where $i = 1.....,k$, are the biases for k-th processing unit, n_i, where $i = 1.....,k$, are the net inputs for k-th processing unit f_i, where $i = 1.....,k$, are the activation functions for k-th processing unit and y_i, where $i = 1.....,k$, are the outputs for k neurons. Note that the $w_{10}, w_{20} \ldots, w_{k0}$ are initial weights for k-th processing unit and $x_o = 1$ is the initial input signal. We also consider $t_{i,}$ e_i and E_i where $i = 1.....,k$ are the target(Desired), error signal and Mean-Square Error (Sum of Squared Error) function respectively.

We have the following calculations
Net inputs:

$$n_1 = w_{11}x_1 + w_{12}x_2 + ... + w_{1n}x_n + b_1 = \sum_{j=1}^{n} w_{1j}x_j + b_1$$

$$n_2 = w_{21}x_1 + w_{22}x_2 + ... + w_{2n}x_n + b_2 = \sum_{j=1}^{n} w_{2j}x_j + b_2$$

…. …. …. …. ….

$$n_k = w_{k1}x_1 + w_{k2}x_2 + ... + w_{kn}x_n + b_k = \sum_{j=1}^{n} w_{kj}x_j + b_k$$

(1)

Biases:

$$b_1 = w_{10}x_0 = w_{10}.1 = w_{10}$$

$$b_2 = w_{20}x_0 = w_{20}.1 = w_{20}$$

…. ….

$$b_k = w_{k0}x_0 = w_{k0}.1 = w_{k0}$$

$$[\because x_0 = 1] \quad (2)$$

From (1) and (2) we have;

$$n_1 = w_{11}x_1 + w_{12}x_2 + ... + w_{1n}x_n + w_{10} = \sum_{j=0}^{n} w_{1j}x_j$$

$$n_2 = w_{21}x_1 + w_{22}x_2 + ... + w_{2n}x_n + w_{20} = \sum_{j=0}^{n} w_{2j}x_j$$

…. …. …. ….

$$n_k = w_{k1}x_1 + w_{k2}x_2 + ... + w_{kn}x_n + w_{k0} = \sum_{j=0}^{n} w_{kj}x_j$$

(3)

Network Outputs:

$$y_1 = f_1(n_1) = f_1\left(\sum_{j=0}^{n} w_{1j}x_j\right)$$

$$y_2 = f_2(n_2) = f_2\left(\sum_{j=0}^{n} w_{2j}x_j\right)$$

…. …. ….

$$y_k = f_k(n_k) = f_k\left(\sum_{j=0}^{n} w_{kj}x_j\right)$$

(4)

Error Signals:

$$e_1 = t_1 - y_1$$

$$e_2 = t_2 - y_2$$

….

$$e_k = t_k - y_k$$

Where $t_1, t_, \cdots t_k$ are target outputs for given neurons in the output layer. (5)

Mean-Square Error functions:

$$E_1 = \frac{1}{2}e_1^{\,2} = \frac{1}{2}(t_1 - y_1)^2$$

$$E_2 = \frac{1}{2}e_2^{\,2} = \frac{1}{2}(t_2 - y_2)^2$$

…. …. ….

$$E_k = \frac{1}{2}e_k^{\,2} = \frac{1}{2}(t_k - y_k)^2$$

(6)

The total error(for n input patterns):

$$E_{total} = \sum_k E_k = E_1 + E_2 + \ldots + E_k = \frac{1}{2}\sum_k \left(t_k(n) - y_k(n)\right)^2 \qquad (7)$$

Multiple-Layer Network

Consider a three layer feedforward neural networks as shown in the Figure 3.

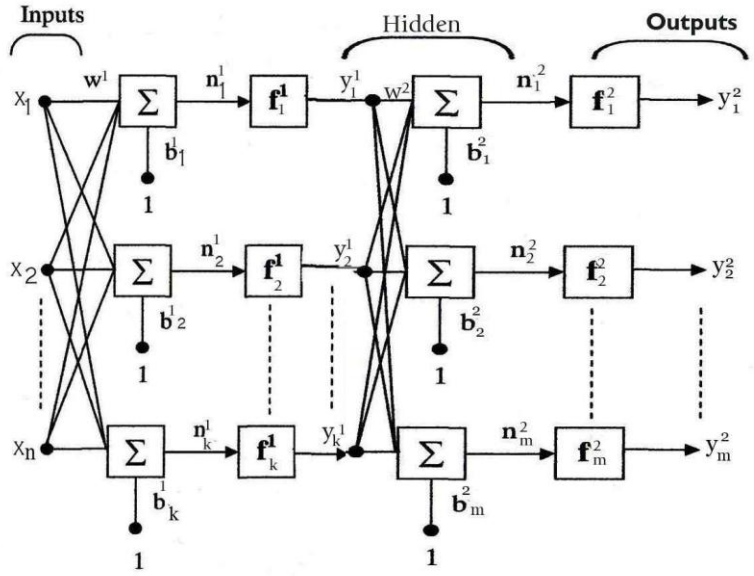

Figure 3. Three layer feedforward neural networks

From input to hidden layer:

There are n inputs x_i, where $i = 1.....,n.$, $\mathbf{w^1} = \left(w_{ij}\right) = \begin{pmatrix} w^1_{11}\ w^1_{12} \ldots\ldots\ldots w^1_{1n} \\ w^1_{21}\ w^1_{22} \ldots\ldots\ldots w^1_{2n} \\ \ldots\ldots\ \ldots\ldots\ \ldots\ldots \\ w^1_{k1}\ w^1_{k2} \ldots\ldots\ldots w^1_{kn} \end{pmatrix}$, is the weighted matrix with kxn size is used to denote the strength of the connection from the *i*th input to the *j*th processing element., There are k neurons in the hidden layer, b^1_j, where $j = 1.....,k$, are the biases for k-th processing unit , n^1_j, where $j = 1.....,k$, are the net inputs for k-th processing unit f^1_j, where $j = 1.....,k$, are the activation functions for k-th processing unit and y^1_j, where $j = 1.....,k$, are the outputs for k neurons. Note that the $w^1_{10}, w^1_{20} \ldots, w^1_{k0}$ are initial weights for k-th processing unit and $x^1{}_o = 1$ is the initial input signal.

We have the following calculations

Net inputs:

$$n^1_1 = w^1_{11}x_1 + w^1_{12}x_2 + \ldots. + w^1_{1n}x_n + b^1_1 = \sum_{j=1}^{n} w^1_{1j}x_j + b^1_1$$

$$n^1_2 = w^1_{21}x_1 + w^1_{22}x_2 + \ldots. + w^1_{2n}x_n + b^1_2 = \sum_{j=1}^{n} w^1_{2j}x_j + b^1_2$$

…. …. …. …. ….

$$n^1_k = w^1_{k1}x_1 + w^1_{k2}x_2 + \ldots. + w^1_{kn}x_n + b^1_k = \sum_{j=1}^{n} w^1_{kj}x_j + b^1_k$$

(8)

Biases:

$$b_1^1 = w_{10}^1 x_0 = w_{10}^1 .1 = w_{10}^1$$
$$b_2^1 = w_{20}^1 x_0 = w_{20}^1 .1 = w_{20}^1$$
....
$$b_k^1 = w_{k0}^1 x_0 = w_{k0}^1 .1 = w_{10}^1$$
$$[\because x_0 = 1] \qquad (9)$$

From (8) and (9) we have;

$$n_1^1 = w_{11}^1 x_1 + w_{12}^1 x_2 + ... + w_{1n}^1 x_n + w_{10}^1 = \sum_{j=0}^{n} w_{1j}^1 x_j$$
$$n_2^1 = w_{21}^1 x_1 + w_{22}^1 x_2 + ... + w_{2n}^1 x_n + w_{20}^1 = \sum_{j=0}^{n} w_{2j}^1 x_j$$
....
$$n_k^1 = w_{k1}^1 x_1 + w_{k2}^1 x_2 + ... + w_{kn}^1 x_n + w_{k0}^1 = \sum_{j=0}^{n} w_{kj}^1 x_j \qquad (10)$$

Network Outputs in the hidden layer:

$$y_1^1 = f_1^1(n_1^1) = f_1^1\left(\sum_{j=0}^{n} w_{1j}^1 x_j\right)$$
$$y_2^1 = f_2^1(n_2^1) = f_2^1\left(\sum_{j=0}^{n} w_{2j}^1 x_j\right)$$
....
$$y_k^1 = f_k^1(n_k^1) = f_k^1\left(\sum_{j=0}^{n} w_{kj}^1 x_j\right) \qquad (11)$$

From hidden layer to output layer:

There are k inputs y_j^1, where $j = 1....., k$, $\mathbf{w^2} = (w_{ij}) = \begin{pmatrix} w_{11}^2 & w_{12}^2 & .....w_{1k}^2 \\ w_{21}^2 & w_{22}^2 & .....w_{2k}^2 \\ ..... & ..... & ..... \\ w_{m1}^2 & w_{m2}^2 & .....w_{mk}^2 \end{pmatrix}$, is the weighted matrix with mxk size is used to denote the strength of the connection from the *j*th input to the *s*th processing element., There are m neurons in the output layer, b_s^2, where $s = 1....., m$, are the biases for m-th processing unit , n_s^2, where $s = 1....., m$, are the net inputs for m-th processing unit f_s^2, where $s = 1....., m$, are the activation functions for m-th processing unit and y_s^2, where $s = 1....., m$ are the outputs for m neurons. Note that the $w_{10}^2, w_{20}^2 \ldots, w_{m0}^2$ are initial weights for m-th processing unit and $y_0^1 = 1$ is the initial input signal.

We have the following calculations

Net inputs:

$$n_1^2 = w_{11}^2 y_1^1 + w_{12}^2 y_2^1 + ... + w_{1k}^2 y_k^1 + b_1^2 = \sum_{j=1}^{k} w_{1j}^2 y_j^1 + b_1^2$$
$$n_2^2 = w_{21}^2 y_1^1 + w_{22}^2 y_2^1 + ... + w_{2k}^2 y_k^1 + b_2^2 = \sum_{j=1}^{k} w_{2j}^2 y_j^1 + b_2^2$$
....

$$n_m^2 = w_{m1}^2 y_1^1 + w_{m2}^2 y_2^1 + ... + w_{mk}^2 y_k^1 + b_m^2 = \sum_{j=1}^{k} w_{mj}^2 y_j^1 + b_m^2$$

(12)

Biases:

$$b_1^2 = w_{10}^2 y_0^1 = w_{10}^2 .1 = w_{10}^2$$
$$b_2^2 = w_{20}^2 y_0^1 = w_{20}^2 .1 = w_{20}^2$$
....
$$b_m^2 = w_{m0}^2 y_0^1 = w_{m0}^2 .1 = w_{m0}^2$$

(13)

From (12) and (13) we have;

$$n_1^2 = w_{11}^2 y_1^1 + w_{12}^2 y_2^1 + ... + w_{1k}^2 y_k^1 + w_{10}^2 = \sum_{j=0}^{k} w_{1j}^2 y_j^1$$
$$n_2^2 = w_{21}^2 y_1^1 + w_{22}^2 y_2^1 + ... + w_{2k}^2 y_k^1 + w_{20}^2 = \sum_{j=0}^{k} w_{2j}^2 y_j^1$$
....
$$n_m^2 = w_{m1}^2 y_1^1 + w_{m2}^2 y_2^1 + ... + w_{mk}^2 y_k^1 + w_{m0}^2 = \sum_{j=0}^{k} w_{mj}^2 y_j^1$$

(14)

Network Outputs in the output layer:

$$y_1^2 = f_1^2\left(n_1^2\right) = f_1^2\left(\sum_{j=0}^{k} w_{1j}^2 y_j^1\right) = f_1^2\left(w_{10}^2 + \sum_{j=1}^{k} w_{1j}^2 f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s\right)\right)$$
$$y_2^2 = f_2^2\left(n_2^2\right) = f_2^2\left(\sum_{j=0}^{k} w_{2j}^2 y_j^1\right) = f_2^2\left(w_{20}^2 + \sum_{j=1}^{k} w_{2j}^2 f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s\right)\right)$$
....
$$y_m^2 = f_m^2\left(n_2^2\right) = f_m^2\left(\sum_{j=0}^{k} w_{mj}^2 y_j^1\right) = f_m^2\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj}^2 f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s\right)\right)$$

(15)

Error Signals:

$$e_1^2 = t_1^2 - y_1^2$$
$$e_2^2 = t_2^2 - y_2^2$$
....
$$e_m^2 = t_m^2 - y_m^2$$

Where $t_1^2, t_2^2 \cdots, t_m^2$ are target outputs for given neurons in the output layer.

(16)

Mean-Square Error functions:

$$E_1^2 = \frac{1}{2}\left(e_1^2\right)^2 = \frac{1}{2}\left(t_1^2 - y_1^2\right)^2$$
$$E_2^2 = \frac{1}{2}\left(e_2^2\right)^2 = \frac{1}{2}\left(t_2^2 - y_2^2\right)^2$$
....
$$E_m^2 = \frac{1}{2}\left(e_m^2\right)^2 = \frac{1}{2}\left(t_m^2 - y_m^2\right)^2$$

(17)

The total error in the output layer(for k input patterna):

$$E_{total} = \sum_m E_m^2 = E_1^2 + E_2^2 + \ldots + E_m^2 \quad = \frac{1}{2}\sum_m \left[t_m^2(k) - f_m^2\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 xs \right) \right) \right]^2 \tag{18}$$

Implementations

Single layer Network

From equation (7) we have

$$E_{total} = \frac{1}{2} \sum_k \left(t_k(n) - f_k\left(\sum_{j=0}^{n} w_{kj} x_j \right) \right)^2 \tag{19}$$

We have;

$$\frac{\partial E_{total}}{\partial w_{kj}} = -\sum_k \left(t_k(n) - f_k\left(\sum_{j=0}^{n} w_{kj} x_j \right) \right) \quad f_k'\left(\sum_{j=0}^{n} w_{kj} x_j \right).x_j \tag{20}$$

Using Steepest Descent Method we have the update weight equation is as

$$w_{kj}^{(p+1)} = w_{kj}^{\ p} + \eta \sum_k \left(t_k(n) - f_k\left(\sum_{j=0}^{n} w_{kj} x_j \right) \right) \quad f_k'\left(\sum_{j=0}^{n} w_{kj} x_j \right).x_j \tag{21}$$

Multiple layer networks

From (18) we have

$$E_{total} \quad = \frac{1}{2}\sum_m \left[t_m^2(k) - f_m^2\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 xs \right) \right) \right]^2 \tag{22}$$

$$\frac{\partial E_{total}}{\partial w_{mj}^2} = \quad -\sum_m \left[t_m^2(k) - f_m^2\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 xs \right) \right) \right]$$
$$\left(f_m^2 \right)'\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj}^2 f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s \right) \right) \quad \left(\sum_{j=1}^{k} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s \right) \right) \tag{23}$$

Using Steepest Descent Method we have the update weight equation is as

$$w_{mj}^{2(p+1)} = w_{mj}^{2(p)} + \eta \quad \sum_m \left[t_m^2(k) - f_m^2\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 xs \right) \right) \right]$$
$$\left(f_m^2 \right)'\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj}^2 f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s \right) \right) \quad \left(\sum_{j=1}^{k} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s \right) \right)$$

and

$$\frac{\partial E_{total}}{\partial w_{kj}^1} = \frac{\partial E_{total}}{\partial y_m^2(k)} \cdot \frac{\partial y_m^2(k)}{\partial y_k^1(n)} \cdot \frac{\partial y_k^1(n)}{\partial w_{kj}^1} \quad = -\sum_m \left[t_m^2(k) - f_m^2\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 xs \right) \right) \right]$$
$$\left(f_m^2 \right)'\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj}^2 f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s \right) \right) \sum_{j=1}^{k} w_{mj}^2 \quad \left(f_k^1 \right)'\left(\sum_{j=0}^{n} w_{kj}^1 x_j \right).x_j$$

Using Steepest Descent Method we have the update weight equation is as

$$w_{kj}^{1(p+1)} = w_{kj}^{1(p)} + \eta \sum_m \left[t_m^2(k) - f_m^2\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj} f_j^1\left(\sum_{s=0}^{n} w_{js}^1 xs \right) \right) \right]$$

$$\left(f_m^2\right)'\left(w_{m0}^2 + \sum_{j=1}^{k} w_{mj}^2 f_j^1\left(\sum_{s=0}^{n} w_{js}^1 x_s \right) \right) \sum_{j=1}^{k} w_{mj}^2 \ \left(f_k^1\right)'\left(\sum_{j=0}^{n} w_{kj} x_j \right).x_j \qquad (24)$$

Numerical Example

Consider the single layer feedforward neural networks as shown in figure 2. Suppose that there are 2 inputs $x_1 = 0.2, x_2 = 0.6$, $\mathbf{w^0} = = \begin{pmatrix} w_{11}^0 & w_{12}^0 \\ w_{21}^0 & w_{22}^0 \end{pmatrix} = \begin{pmatrix} 0.1 & 0.3 \\ 0.3 & 0.4 \end{pmatrix}$, is the 2x2 order weighted matrix is used to denote the strength of the connection from the input to processing unit., the biases for two processing units are b_1, b_2, the net inputs for two processing unit are n_1^0, n_2^0, $f_1 = \frac{1}{1+e^{-n}}$, $f_2 = \frac{e^n - e^{-n}}{e^n + e^{-n}}$ are the activation functions for the processing unit and y_1, y_2, are the outputs for 2 neurons. Note that the $w_{10} = 0.01, w_{20} = 0.02$ are initial weights for two processing unit and $x_o = 1$ is the initial input signal. Consider e_1, e_2 and E_1, E_2 are the error signals and Mean-Square Error (Sum of Squared Error) functions respectively and $t_{1,} = 0.7$, $t_2 = 0.8$ are the target(Desired). Use a step size of η = 10.

We have the following calculations

Net inputs:

$$n_1^0 = w_{11}^0 x_1 + w_{12}^0 x_2 + b_1 = 0.2 \times 0.1 + 0.6 \times 0.3 + 0.01 \times 1 = 0.21$$

$$n_2^0 = w_{21}^0 x_1 + w_{22}^0 x_2 + b_2 = 0.2 \times 0.3 + 0.6 \times 0.4 + 0.02 \times 1 = 0.32$$

Network Outputs:

$$y_1^0 = f_1\left(n_1^0\right) = \left(\frac{1}{1+e^{-0.21}} \right) = 0.5522$$

$$y_2^0 = f_2\left(n_2^0\right) = \left(\frac{e^{0.32} - e^{-0.32}}{e^{0.32} + e^{-0.32}} \right) = 0.3094$$

Error Signals:

$$e_1^0 = t_1 - y_1^0 = 0.7 - 0.5522 = 0.1478$$

$$e_2^0 = t_1 - y_2^0 = 0.8 - 0.3094 = 0.4906$$

Mean-Square Error functions:

$$E_1^0 = \frac{1}{2} e_1^{0\,2} = 0.0109$$

$$E_2^0 = \frac{1}{2} e_2^{0\,2} = 0.1203$$

The total error (for 2 input patterns):

$$E^{0}{}_{total} = E_1^0 + E_2^0 = .0109 + 0.1203 = 0.1312$$

To update the weights, we use equation (21)

$$w_{11}^1 = w_{11}^0 + \eta\left(t_1 - f_1\left(n_1^0\right)\right) f'\left(n_1^0\right) x_1 = 0.1 + 10 \times 0.1478 \times 0.2472 \times 0.2 = 0.1730$$

$$w_{12}^1 = w_{12}^0 + \eta\left(t_1 - f_1\left(n_1^0\right)\right) f'\left(n_1^0\right) x_2 = 0.3 + 10 \times 0.1478 \times 0.2472 \times 0.6 = 0.5192$$

$$w_{21}^1 = w_{21}^0 + \eta\left(t_2 - f_2\left(n_2^0\right)\right) f'\left(n_2^0\right) x_1 = 0.3 + 10 \times 0.4906 \times 0.2852 \times 0.2 = 0.5798$$

$$w_{22}^1 = w_{22}^0 + \eta\left(t_2 - f_2\left(n_2^0\right)\right) f'\left(n_2^0\right) x_2 = 0.4 + 10 \times 0.4906 \times 0.2852 \times 0.6 = 1.2395$$

The update weight matrix:

$$\mathbf{W^1} = \begin{pmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{pmatrix} = \begin{pmatrix} 0.1730 & 0.5192 \\ 0.5798 & 1.2395 \end{pmatrix}$$

$$n_1^1 = w_{11}^1\ x_1 + w_{12}^1\ x_2 + b_1 = 0.173 \times 0.2 + 0.5192 \times 0.6 + 0.01 \times 1 = 0.3561$$

$$n_2^1 = w_{21}^1\ x_1 + w_{22}^1\ x_2 + b_2 = 0.5798 \times 0.2 + 1.2395 \times 0.6 + 0.02 \times 1 = 0.8796$$

Network Outputs:

$$y_1^1 = f_1\left(n_1^1\right) = \left(\frac{1}{1+e^{-0.3561}}\right) = 0.5880$$

$$y_2^1 = f_2\left(n_2^1\right) = \left(\frac{e^{0.8796} - e^{-0.8796}}{e^{0.8796} + e^{-0.8796}}\right) = 0.7062$$

Error Signals:

$$e_1^1 = t_1 - y_1^1 = 0.7 - 0.5880 = 0.112$$

$$e_2^1 = t_1 - y_2^1 = 0.8 - 0.7062 = 0.0938$$

Mean-Square Error functions:

$$E_1^1 = \frac{1}{2} e_1^{1^2} = 0.0062$$

$$E_2^1 = \frac{1}{2} e_2^{1^2} = 0.0043$$

The total error (for 2 input patterns):

$$E^1{}_{total} = E_1^1 + E_2^1 = .0062 + 0.0043 = 0.0105$$

Obviously, $E^1{}_{total} < E^0{}_{total}$; that is, the actual output of the neural network has become closer to the target output as a result of updating the weights.

II. Conclusion

A simple but effective description on feedforward artificial neural networks has been made in here giving emphasize on the backpropagation algorithm, since it is widely used and many other algorithms are derived from it. We have implemented the steepest descent method in single and multiple-layer feedforward artificial neural networking problem and set-up a numerical test example. Firstly, We have calculated the total error function separately for single and multilayer feedforward artificial neural networks and then calculated the three new update weight equations for taking different activation function in different processing unit separately single and multilayer feedforward artificial neural networks. The convergence behavior of our example shows that the results of the actual network output is as "close" to our desired(target) output.

Acknowledgement

The Noakhali Science and Technology University and the University of Chittagong, for providing a stimulating environment for research in connection with this article.

References

[1] Adby, P.R. and Dempster, M.A.H. 1974. Introduction to Optimization Methods. *Haisted Press*, New York.

[2] Battiti, R. 1992. First and second-order methods for learning: Between Steepest Descent and Newton's methods. *Neural Computation* **4**: 141-166.

[3] Johansson, E.M., Dowla, F.U. and Goodman, D.M. 1992. Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method. *Intl. J. Neural Systems*, **2:** 291-301 Judd, J.S. 1990. Neural Network Design and the complexity of Learning. *MIT Press*, Cambridge, MA

[5] Muhammad Hanif. 2002. Optimality of Unconstrained Nonlinear Programming Problem. M.Phil Thesis, University of Chittagong, Chittagong, Bangladesh.

[6] Mehra] P., and Wah, B. W. 1992. Artificial neural networks: concepts and theory *IEEE Comput. Society Press*,.

[7]. X. Yu, M. O. Efee, and O. Kaynak.2002. A general backpropagation algorithm for feed-forward neural networks learning. *IEEE Trans. Neural Networks*. **13:**251–254.

[8] X. H. Hu and G. A. Chen.1997. Efficient backpropagation learning using optimal learning rate and momentum. *Neural Networks* **10**:517-527.

[9] Zurada, J. M. 2002. Introduction to artificial neural systems. *M. G. Road, Mumbai: Jaico.*