

NEURAL NETWORKS and COMPUTATION OF NEURAL NETWORK WEIGHTS AND BIASES by the GENERALIZED DELTA RULE and BACK-PROPAGATION of ERRORS

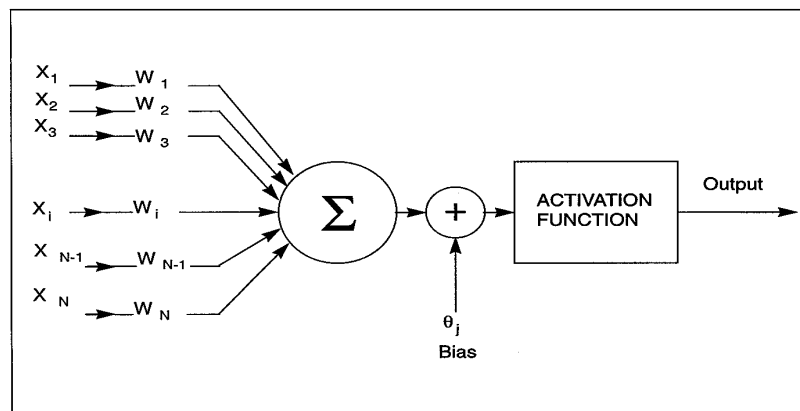
Dr. M. Turhan (Tury) Taner, Rock Solid Images
1995

INTRODUCTION:

In this note I will develop equations to determine the optimum weights and biases that minimize the errors between expected and actual outputs of a general Neural Network to a given training data set. The computational procedure is the Generalized Delta Rule with *Back Propagation* of errors, sometimes called the Rumelhart method.

Readers interested in a more general class of neural network optimizations should refer to "**Adaptive Pattern Recognition and Neural Networks**" by Yoh-Han Pao, published by Addison-Wesley, 1989 and references found in this book. There is also an article in the Leading Edge on Neural Networks, written by McCormack. Also a book published by SEG edited by Fred Aminzadeh entitled "**Expert Systems in Geophysics**". This article should help those who wish to understand the basic structure and computational background of the Neural Networks. I have also included two special issues of Proceeding of IEEE on Neural Networks at the end of this paper, which contain many invited papers by the originators of new methodologies and a considerable number of references.

It should be understood that Neural Networks are still being developed and an infinite variety of networks are being experimented. There is a general lack of understanding as to how the Neural Network works and what significance the weights of the hidden layers have. Also, there is no computational procedure to guarantee direct computation of the weights. The method described here, **the Generalized Delta Rule**, is an adaptation of the methodology used in solving non-linear equations by the steepest-gradient method. The most important part of the method is to recognize the recursive determination of the Delta functions first. After all of the Delta functions are determined the corrections to all of the weights are computed.



Perceptron or "NEURON"
Fig : 1

I will follow the direction presented in chapter 5 of Pao's book referred to above.

NEURONS:

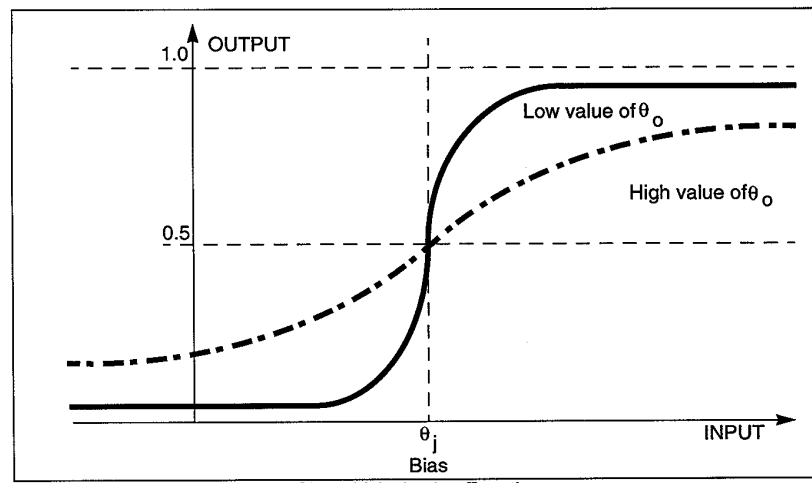
Figure 1 shows a schematic picture of a Neuron. The Neuron consists of a set of inputs $\mathbf{X}=[x(1),x(2),\dots,x(N)]$ and one output. As shown in the figure all that is in the rectangle represents components of the Neuron. This is also called the '**Perceptron**'. Functionally each input $x(i)$ is weighted or multiplied by a weight function $w(i)$ and summed. This is called **net input** of the Neuron,

$$net = \sum_{i=1}^N [x(i).w(i)] \quad (1)$$

Then, an activation function is computed. This simulates the firing of the nerve cells by inputs from other nerve cells. This activation function can be a step function, arc-tangent or a sigmoidal function. I will use the sigmoidal function for ease of computation. Before the activation function is computed, we add a bias value q , which serves as a threshold for the activation function. The output of the neuron is;

$$b(j) = 1./\{1. + \exp[-\frac{(net(j) + q(j))}{q(0)}]\} \quad (2)$$

Where $q(0)$ is a sigmoid shaping factor. If $q(0)$ is a high number, then the sigmoid will be a gently varying function. For lower values of $q(0)$ the sigmoid will take steeper form. This is illustrated in figure 2.



Sigmoid Activation Function
Fig : 2

Training a neuron means computing the values of the $w(i)$ and $q(j)$ weights and the bias so that it will correctly classify a given training set,

$$X.w = \begin{cases} +1 & \text{if } X \text{ belongs to class A} \\ 0 & \text{if } X \text{ belongs to class B} \end{cases} \quad (3)$$

If the input data is noisy, which is generally the case, we will never achieve the 0 and 1 results, they will be somewhere in-between. This is why some subjective threshold value has to be included to decide where the classification of **A** ends and classification of **B** begins. A single neuron, when trained, can perform as a linear discriminator. This is similar to the well known single or multi-channel **Wiener** operator.

LINEAR DISCRIMINATION BY A NEURON

The neuron shown in figure 1 was first developed by Widrow (1962) and was called the linear Perceptron. Let $\mathbf{X} = \{x(1), x(2), \dots, x(N)\}$ represent the input training pattern in the form of column vectors. We wish to solve for a column vector \mathbf{w} of M elements such that;

$$X \mathbf{w} = \mathbf{b} \quad (4)$$

where the elements of \mathbf{b} are the output values specified by the training set. In binary classification $\mathbf{b}=\mathbf{1}$ specifies the corresponding \mathbf{X} set belongs to the class \mathbf{A} and $\mathbf{b}=\mathbf{0}$ specifies the corresponding set \mathbf{X} belongs to the class \mathbf{B} . Since \mathbf{X} matrix has N number of columns and M number of rows and $M < N$, then we can solve this set of equations by the classical *least mean error squares* method. We can obtain the normal equations square matrix by pre-multiplying both sides by the transpose of the \mathbf{X} matrix;

$$\mathbf{X}^T . \mathbf{X} . \mathbf{w} = \mathbf{X}^T \mathbf{b} \quad (5)$$

and solve for \mathbf{w} ;

$$\mathbf{w} = (\mathbf{X}^T . \mathbf{X})^{-1} . \mathbf{X}^T \mathbf{b} \quad (6)$$

This expression can be simplified to ;

$$\mathbf{w} = \hat{\mathbf{X}} . \mathbf{b} \quad (7)$$

where $\hat{\mathbf{X}}$ is the pseudo-inverse of the original rectangular $(\mathbf{X}^T . \mathbf{X})^{-1} . \mathbf{X}^T$ matrix,

$$\hat{\mathbf{X}} = (\mathbf{X}^T . \mathbf{X})^{-1} . \mathbf{X}^T . \quad (8)$$

In theory, this inverse can be computed directly. However we may get some nonsensical values which do not represent the real situation. The inverse is computed by an iterative procedure called the linear perceptron algorithm which leads us to the Delta rule.

We wish to compute a single set of \mathbf{w} weights to yield correct set of outputs \mathbf{b} for all input patterns $\mathbf{x}(\mathbf{p})$. We start with an arbitrary set of values of $w^{(1)}(i)$ then update it by the following rule;

$$w^{(k+1)}(j) = w^k(j) + \mathbf{r}[b(j) - w^k(j)x(j)].x(j) \quad (9)$$

This updating continues until all the patterns are classified correctly, at which time $[b(j) - w^k(j)x(j)]$ becomes zero or very small. In most practical cases this cannot be reached, hence the iteration should be stopped when the sum of the squares of errors reaches some prescribed threshold value.

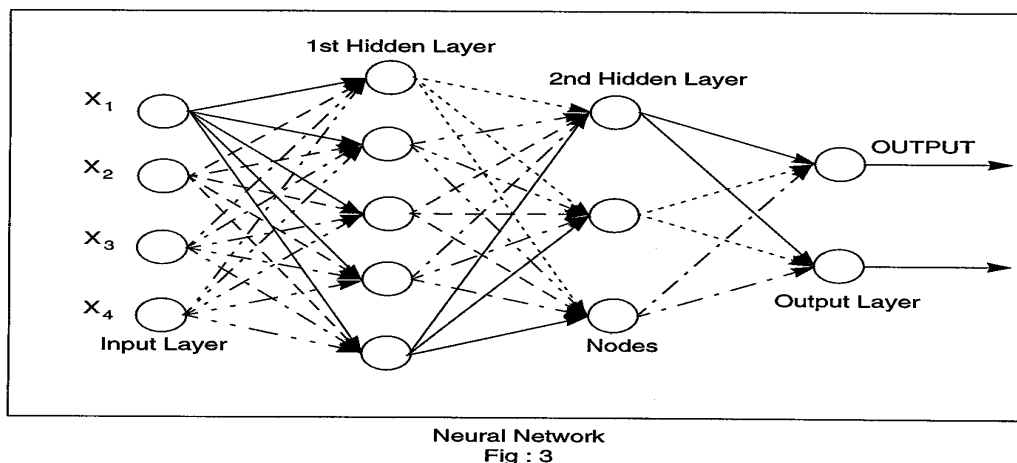
We can write the expression 9 in the form;

$$\Delta(\mathbf{W}) = \mathbf{h} . \mathbf{d} . \mathbf{X} \quad (10)$$

where \mathbf{d} is the difference between the desired output and the computed actual output produced by the perceptron. This is the delta-rule expression which states that the change in the weight vector should be proportional to the delta (the error) and to the input pattern.

NEURAL NETWORKS

A neural network consists of an inter-connection of a number of neurons. There are many varieties of connections under study, however here I will discuss only one type of network proposed by Rumelhart and others, which is called the "Semi-linear Feed-Forward" "*net*". In this network the data flows forward to the output continuously without any feed-back. It seems that this type of network is very suitable for many of the pattern recognition problems in Geophysics. A schematic picture of semi-linear feed-forward network is depicted in figure 3.



As seen in figure 3, each node represents a neuron which are organized in layers. The first is the input layer containing the receptacles of the neurons from the first hidden layer. The input is weighted and summed and transferred to the nodes as ``net'' of that node. A bias is added and the activation function is computed to become the output from that neuron. This output is transmitted to all the neurons in the next layer and so on until the output is generated. The layers between the input and output layers are called `Hidden' layers. A network with no hidden layers is capable of performing linear discrimination problems properly. Networks with hidden layers are capable of handling non-linear discrimination problems. The rule of thumb is ; **`more hidden layers can handle more complicated cases'**. However, the latest tests indicate that one hidden layer is generally sufficient for many of the problems we face today. In the case of binary classification only one output node can be used. However for more complicated multi-faceted classification any number of output nodes can be used. The only warning, the general rule of mathematics for linear or non-linear equations, is **``keep the number of unknowns equal to, or less than, the number of knowns''**. Therefore, the number of hidden layers, number of nodes on each layer and number of output nodes must be kept within a reasonable range. A large number of inputs in the training set may help the network overcome the effects of the noisy sets.

Neural networks take an extremely large number of iterations to train. Hence the larger the number of hidden layers and nodes, the longer it will take to train. There is no guarantee at this time the results will converge to the global minimum. The procedure of obtaining a global minimum is a good subject for young researchers. One of the more recently developed techniques is the so called **`Genetic'** Algorithm, which is based on a random trial and error correction scheme. This technique was used in the residual static correction, originated by Stanford University students under Prof. Claerbout.

HOW TO USE A NEURAL NETWORK

Neural networks are a relatively new development, hence their use is not as well developed as some of the other artificial intelligence tools. However, it is clear that problems that can be solved by conventional linear discrimination do not need neural network solutions. As we pointed out earlier, neural nets take an extremely long time to train, principally due to the lack of fast optimizing algorithms. In the near future, if such optimization develops, we may be able to use the neural nets with more efficiency. Here I would like to give a simple pattern recognition example using the neural network.

It is necessary to identify objects by the characteristics which separate them from others. These characteristics are called `features'. In pattern recognition we have to define each object by its features in the form of a series of numbers listed as feature vectors. For example, if we wish to teach the computer to pick the first breaks automatically, we have to first define some characteristics of the first breaks which makes them different from all other seismic events or noise patterns. Generally first breaks arrive as the first large event after a lower amplitude noise zone. Therefore, the difference between the mean amplitude level of the noise and the mean amplitude level of

the first break is an interesting feature to use. We see that after the arrival of the first breaks the amplitude remains at about the same level for a little while. Therefore, we cannot use the same criteria for differentiating between the first break and later arriving events. In this case we may use the derivative of the mean amplitude level. At the first break time the mean amplitude level increases from a mean noise level to the mean reflected energy level. Hence the rate of change is much larger than the change expected within the reflected energy zone. This is usually measured as the power ratio which has a maximum around the first break time. We usually pick a peak or a trough of the seismic trace. This way we are choosing a location on the trace where the first derivative is zero. If we pick a positive peak, then we are further reducing the possible points to only the positions with a positive value and with their first derivatives equal to zero. Since we are assuming that, in general, the reflected or refracted waves have higher energy than the noise zone, we can define a threshold level below which traces will be classified as noise zone and above which part of the trace will be classified as the reflected or refracted zone. Therefore, the amplitude of the trace envelope becomes one of the features. If we consider the envelope slope as another feature we can see that in the front side of reflection this slope will be positive and on the hind side the slope will be negative. At the peak of the envelope it will be zero. This means we can also define where to pick by giving a range of values of the envelope slope. If we consider positive values, we are picking a point corresponding to the front-side of the first-break wavelet. We can see that various seismic attributes, as well as the arrival times and velocities, can be used to describe the first break arrivals.

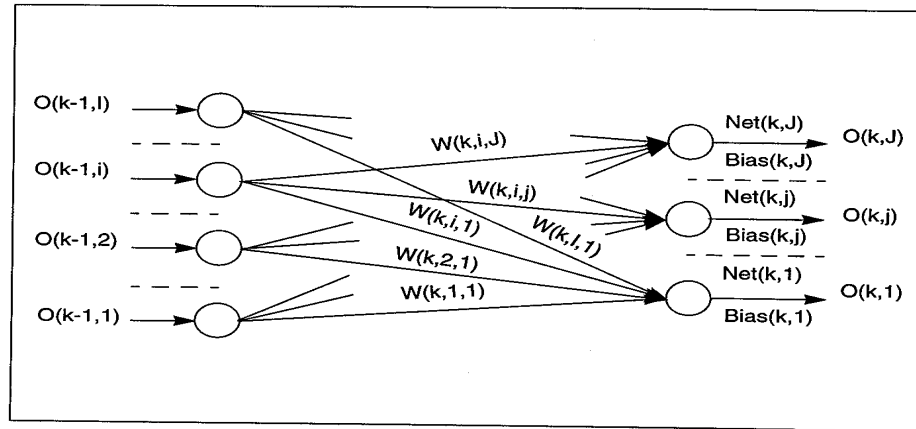
The general procedure is to select several shots along a given line representing the characteristics of the first-break patterns in their respective areas. We will select a zone several hundred milliseconds long around the first-break area. The user will indicate the first-break pick times for a range of offsets. The program first determines the times of all the peaks with the same polarity as the users request, then computes all of the attributes and times of all the peaks. It will mark the user given pick as the correct pick and all others as the wrong pick or pre-first-break pick or post-first-break pick. In the latter case we have three different classifications. If we have 8 attributes describing a pick, then our neural network will have 8 input nodes and 3 output nodes. We will have as many training sets of input data as the number of traces picked on the input data set. We can define our picks by giving an expected value for each training set. For example we can assign the first output node to represent the classification of the pre-first-break picks, the second to the actual picks and the third to the post-first-break picks. If the input set belongs to the pre-first-break events, then we expect the output will be [1.0, 0.0, 0.0], if it is the actual pick it will be [0.0, 1.0, 0.0] and the output for post-first-break pick will be [0., 0., 1.0]. Input data and their corresponding expected output values are input to the network for training.

The network will determine all of the coefficients by **back-propagation** of errors which will try to minimize the sum of the squares of the difference (errors) between the expected and the actual computed output. This usually takes hundreds or thousands of iterations. The rate of convergence is faster in earlier iterations and becomes slower as the iteration number increases. In some instances it will come to a plateau, which may be a local minimum and in that case any further iteration will not improve the results significantly. Here the computation will have to be stopped and the results may be used if a sufficient degree of convergence has been reached. We can check this by looking at the computed results and compare them to the expected values. If we reach a reasonable degree of convergence, we consider that the network is trained and it can now be used to pick first-breaks on the nearby shots. If the training is not sufficient, we can use a second shot as an additional training set, as a larger number of training sets should result in a better degree of convergence.

GENERALIZED "DELTA RULE" FOR SEMI-LINEAR FEED-FORWARD NET WITH "BACK-PROPAGATION" OF ERROR

As I have outlined above, I will discuss only the feed-forward semi-linear Neural networks for the purpose of uncomplicated discussion of the computation of the optimized weights and biases.

In order to simplify the continuity I will use similar notation to Pao. $X = [x(1), x(2), x(3), \dots, x(N)]$ is the input data set. Pao uses p as an index for the input data set sequence. Since we will use only one set at a time I will dispense with using the p index. Let the input X belong to the class b as defined by the expected target values of $T[t(1), t(2), t(3)]$. Which means that when we input X to the network we expect to get T as the output (See figure 4 for index configuration);



Output Layer
Fig : 4

$$X w = t \tag{11}$$

The actual output however is $O[O(1),O(2),O(3)]$, then the error vector is;

$$e = (T - O) \tag{12}$$

In the least mean error square computation we compute the w operators that minimizes the sum of squares of the error function E .

$$E = \frac{1}{2} \sum_{k=1}^K [t(k) - O(k)]^2 \tag{13}$$

The gradient of (n) dimensional error surface (E) is given by the vector defined by the partial derivatives of the error function with respect to the unknown weight and bias values;

$$\frac{\partial E}{\partial w(k, i, j)} \tag{14}$$

If we are at a minimum this function will be zero for all values of $w(i,j)$. In fact we set the derivatives of the E function equal to zero for direct solution of quadratic Error functions. The reason is simply that the derivatives of quadratic functions are linear equations and we have many algorithms to solve linear system of equations. Unfortunately the Error function for the Neural network is not a quadratic function and the resulting derivatives are not linear equations. We will use the method of the Generalized Delta function and the steepest descent algorithm. Since the gradient is given by partial derivatives, then if we use an incremental change of weights $\Delta w(i, j)$ proportional to the derivatives, we will improve the corresponding error;

$$\Delta w(k, i, j) = -\eta \frac{\partial E}{\partial w(k, i, j)} \tag{15}$$

where (k) represents the output layer index, (j) represents the j'th node of the output layer and index (i) represents the i'th node of the layer before the output layer. According to the network definition, the error (E) is expressed in terms of expected and actual output. And the actual output is the non-linear output of node (k);

$$O(k, j) = f[net(k, j)] \tag{16}$$

where $net(k)$ is the weighted sum of all outputs from the previous layer,

$$net(k) = \sum_{i=1}^I [w(k, i, j) \cdot O(k-1, i)] \quad (17)$$

We cannot compute the partial derivative of E with respect to any of the weights directly, but we can evaluate them by the use of the chain rule;

$$\frac{\partial E}{\partial w(k, i, j)} = \left[\frac{\partial E}{\partial net(k, j)} \right] \cdot \left[\frac{\partial net(k, j)}{\partial w(k, i, j)} \right] \quad (18)$$

We know from definition that at the j 'th node of the output layer the $net(k, j)$ is the weighted sum of actual outputs of the previous layer. Then the required derivative is ;

$$\frac{\partial net(k, j)}{\partial w(k, i, j)} = \frac{\partial \left\{ \sum_{i=1}^I [w(k, i, j) \cdot O(k-1, i)] + bias(k, j) \right\}}{\partial w(k, i, j)} = O(k-1, i) \quad (19)$$

which is the computed output of i 'th node of the last hidden ($k-1^{st}$) layer. At this time we introduce the \mathbf{d} function as;

$$\mathbf{d}(k, j) = - \frac{\partial E}{\partial net(k, j)} \quad (20)$$

Therefore we can write;

$$\Delta w(k, i, j) = \mathbf{h} \cdot \mathbf{d}(k, j) \cdot O(k-1, i) \quad (21)$$

as an expression in the form of the '*Generalized Delta*' rule.

To compute the second component, we again use the chain rule to express the partial derivative in terms of more easily computable components;

$$- \frac{\partial E}{\partial net(k, j)} = \left[- \frac{\partial E}{\partial O(k, j)} \right] \cdot \left[\frac{\partial O(k, j)}{\partial net(k, j)} \right] \quad (22)$$

(from expression 13) Where;

$$\frac{\partial E}{\partial O(k, j)} = -[t(j) - O(k, j)] \quad (23)$$

and ;

$$\frac{\partial O(k, j)}{\partial net(k, j)} = O(k, j) \cdot [1.0 - O(k, j)] \quad (24)$$

if we use the sigmoid $[1.0 + \exp-(O(k, j) + bias)]^{-1}$ as the excitation function. In this case the \mathbf{d} function will be ;

$$d(k, j) = [t(j) - O(k, j)] \cdot O(k, j) \cdot [1.0 - O(k, j)] \tag{25}$$

Therefore the update value for the weight for the output layer will be;

$$\Delta w(k, i, j) = h \cdot O(k, j) [1 - O(k, j)] \cdot [t(j) - O(k, j)] \cdot O(k - 1, i) \tag{26}$$

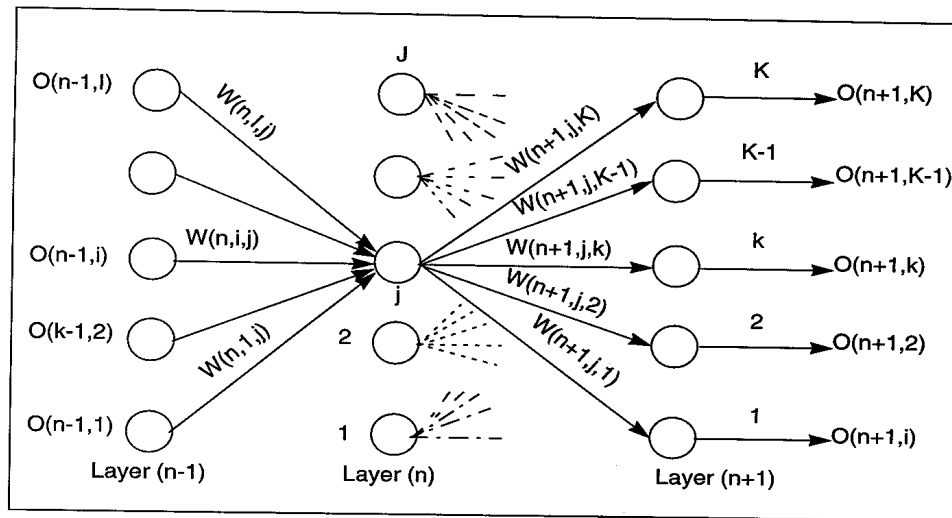
To update the bias value we will use a similar approach except instead of an arbitrary output from the layer before we will have unity. The rest of the developments will be the same. The only change is that;

$$\frac{\partial net(k, j)}{\partial bias(k, j)} = \frac{\partial \left\{ \sum_{i=1}^I [w(k, i, j) \cdot O(k - 1, i)] + bias(k, j) \right\}}{\partial bias(k, j)} = 1 \tag{27}$$

Therefore the update value for the bias at the output layer j'th node is;

$$\Delta bias(k, j) = h \cdot O(k, j) [1 - O(k, j)] \cdot [t(j) - O(k, j)] \tag{28}$$

Update of the hidden layer's weights and biases follows a similar line of development. Figure 5 shows the indices and input output relations between hidden, input or output layers.



Hidden Layer
Fig : 5

The main concept is that we start at the output layer and back-propagate the influence of error one layer at a time towards the input layer, hence the name 'Back-Propagation'. We compute the corrections for the weights and biases connected with the output layer as the first step. Next we will compute the last hidden layers weights and biases, and then the next and so on.

Let us assume that we have computed the n+1'st layer and we wish to compute the corresponding weights for the n'th layer. We will use *j* representing the node index on the n'th layer, index *i* for the n-1'st layer and index *k* for the n+1'st layer. Note that there may be a different number of nodes in different layers.

As before, the Gradient is given by the partial derivatives;

$$-\frac{\partial E}{\partial w(n, i, j)} \quad (29)$$

We will again use the chain rule to evaluate the derivatives;

$$-\frac{\partial E}{\partial w(n, i, j)} = \left[-\frac{\partial E}{\partial net(n, j)} \right] \cdot \left[\frac{\partial net(n, j)}{\partial w(n, i, j)} \right] \quad (30)$$

We can show that, as we have done above, the second partial derivative is;

$$\frac{\partial net(n, j)}{\partial w(n, i, j)} = \frac{\partial}{\partial w(n, i, j)} \left\{ \sum_{i=1}^I [w(n, i, j) \cdot O(n-1, i) + bias(n, i)] \right\} = O(n-1, i) \quad (31)$$

Note that $O(n-1, i)$ is the actual output of the i 'th node of layer $n-1$. If this is the input layer then we will use the actual input $x(i)$ instead of $O(n-1, i)$.

This derivative will also be equal to 1 for the case of the partial derivative with respect to the biases. The first partial derivative can be expanded by the chain rule,

$$\frac{\partial E}{\partial net(n, j)} = \left[\frac{\partial E}{\partial O(n, j)} \right] \cdot \left[\frac{\partial O(n, j)}{\partial net(n, j)} \right] \quad (32)$$

Where the first partial derivative will be equal to the sum of partial derivatives at each node of the $n+1$ 'st layer, which output from j 'th node of n 'th layer contributes;

$$-\frac{\partial E}{\partial O(n, j)} = -\sum_{k=1}^K \left[\frac{\partial E}{\partial net(n+1, k)} \right] \cdot \left[\frac{\partial net(n+1, k)}{\partial O(n, j)} \right] \quad (33)$$

The second part of the above expression is the partial derivative of the Net's of the $n+1$ 'st layer with respect to the outputs from the n 'th layer. Since Net's are the weighted sums of the outputs from the n 'th layer, then;

$$net(n+1, k) = \sum_{j=1}^J \{w(n+1, j, k) \cdot O(n, j)\} \quad (34)$$

Then the partial derivative will be equal to ;

$$-\frac{\partial net(n+1, k)}{\partial O(n, j)} = w(n+1, j, k) \quad (35)$$

We have computed the first part of the expression earlier (equations 22 through 25), so the final expression will becomes ;

$$-\frac{\partial E}{\partial O(n, j)} = \sum_{k=1}^K \{d(n+1, k) \cdot w(n+1, j, k)\} \quad (36)$$

Then according to the generalized delta rule , we will have the new d function as;

$$\mathbf{d}(n, j) = O(n, j)[1 - O(n, j)] \sum_{k=1}^K \{\mathbf{d}(n+1, k).w(n+1, j, k)\} \quad (37)$$

Since the update of the weights according to the delta rule is;

$$\Delta w(n, i, j) = \mathbf{h}.\mathbf{d}(n, j).O(n-1, i) \quad (38)$$

Then the update rule for the biases in the hidden layers are;

$$\Delta bias(n, j) = \mathbf{h}.\mathbf{d}(n, j) \quad (39)$$

This is repeated all the way back to the input layer, at which time $O(n-1, i)$ output values from the layer before are replaced by the actual input values $x(i)$.

As can be seen from the computation sequence, $\mathbf{d}(n, j)$ values at each layer, hidden or otherwise, and at each node can be computed independently of updated weights, starting from the output layer computed recursively backward towards the input layer. This is the most significant part of the computation which made the network optimization possible. Once the $\mathbf{d}(n, j)$ values are computed the rest of the update values for the weights are determined easily. There are other optimization algorithms such as the genetic algorithms and the conjugate gradient method. I have confined myself to the steepest descent method with the Delta rule for its elegance. Readers interested in other algorithms should refer to the references given below.

UPDATING STRATEGIES;

We compute $\Delta w(n, i, j)$ for each training input pattern. These are accumulated and an update is made after all of the training patterns are utilized. Then the update value will be;

$$\Delta w(n, i, j) = \sum_{p=1}^P \Delta w_p(n, i, j) \quad (40)$$

In the beginning we start the net with a random set of weights and biases. Initially errors will be quite large, and computed Δw values will only indicate the direction for proper correction, not the actual amount. We, therefore, approach the minimum cautiously. Let $w^m(n, i, j)$ be the weight set at the m 'th iteration, $\Delta w^m(n, i, j)$ are the set of updates. We can restrain the updates from rapid oscillation by adding a momentum term utilizing a portion of the previous update to be included in the latest update;

$$\Delta w^{m+1}(n, i, j) = \mathbf{h}.\mathbf{d}(k, j).O(k-1, i) + \mathbf{a}\Delta w^m(n, i, j) \quad (41)$$

\mathbf{a} is the momentum factor. If \mathbf{a} is large we will be strongly influenced by the last set of updates. Hence we will have a certain inertia to follow a particular direction with a smaller amount of rate of change. The value of \mathbf{h} should be small, which will give us a smaller incremental approach to the minimum. If \mathbf{h} is large, we may experience large variations in the weight space. On the other hand smaller \mathbf{h} values will require a larger number of iterations to reach the optimum solution. It may also get stuck in a local minima.

We can see that the updating strategy of back-propagation is a subject open to personal preference and experience. We may have to use an adaptive scheme, where we could use layer steps with a smaller momentum at the beginning and as we approach the minimum we can reduce the step sizes. If we get stuck in a local minimum we may use the genetic algorithm, adding random values to each weight, to see if we can get out of the local minimum.

REFERENCES:

I am giving here a partial reference list. The first two references give examples directly applicable in Geophysics. The book by Pao seems to be very informative and full of references. The last references are the special issues of the IEEE (the Institute of Electronics and Electrical Engineers) proceedings on the Neural network. The two issues referenced here contain a number of important articles written specially by the creators of the technology and their immediate students. Each issue also contains many references. I think anyone who wishes to investigate further into Neural Networks should find these references more than enough for a good start.

Veezhinathan, J., Wagner, D. and Ehlers, J ;1991, *First Break Picking Using a Neural Network, From 'Expert Systems in Geophysics'* edited by F.Aminzadeh and Chatterjee, published by **S.E.G.**

McCormack, Micheal D., 1991, *Neural Computing In Geophysics* , The Leading Edge, January Vol 10 No, 1 , pp 15 ,SEG publication.

Pao Y. H. , 1989, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley Publishing Company.

Haykin, Simon,1994, *Neural Networks, A comprehensive foundation*, Macmillan College Publishing Company.
(This is a vry comprehensive and well written book, easy to read and follow the subject)

Hertz, J., Krogh, A., Palmer, R.G., 1991, *Introduction to the Theory of Neural Computation*, Lecture Notes Vol. 1, Addison-Wesley Publishing Company.

Proceedings of the IEEE , 1990, Special issue on Neural Networks;

1. *Neural Networks ,theory and modeling* (September issue)
 2. *Neural networks ,analysis, techniques and applications* (October issue)
- (These issues have extensive literature and background articles)