ANN ee543

CHAPTER VI

Learning in Feedforward Neural Networks

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

Introduction

- The method of storing and recalling information and experiences in the brain is not fully understood. However, experimental research has enabled some understanding of how neurons appear to gradually modify their characteristics because of exposure to particular stimuli.
- The most obvious changes have been observed to occur in the electrical and chemical properties of the synaptic junctions. For example the quantity of chemical transmitter released into the synaptic cleft is increased or reduced, or the response of the post-synaptic neuron to receive transmitter molecules is altered.
- The overall effect is to modify the significance of nerve impulses reaching that synaptic junction on determining whether the accumulated inputs to post-synaptic neuron will exceed the threshold value and cause it to fire.
- Thus learning appears to effectively modify the weighting that a particular input has with respect to other inputs to a neuron.
- In this chapter, learning in feedforward networks will be considered.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

- Perceptron was introduced by Frank Rosenblatt in the late 1950's (Rosenblatt, 1958) with a learning algorithm on it.
- Perceptron may have continuous valued inputs.
- It works in the same way as the formal artificial neuron defined previously.
- Its activation is determined by equation:

$$a=\mathbf{w}^T\mathbf{u} + \theta \qquad (6.1.1)$$

- Moreover, its output function is:

$$f(a)=\begin{cases}+1 & \text{for } a \geq 0\\ -1 & \text{for } 0 < a\end{cases} \qquad (6.1.2)$$

having value either +1 or -1.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

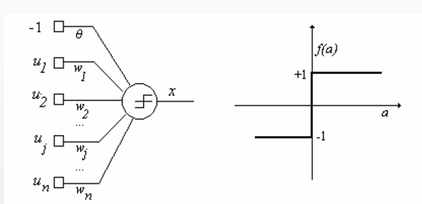


Figure 6.1. Perceptron

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

- Now, consider such a perceptron in N dimensional space (Figure 6.1), the equation

  $$\mathbf{w}^T\mathbf{u} + \theta = 0 \qquad (6.1.3)$$

  that is

  $$w_1u_1+w_2u_2+...+w_N u_N + \theta = 0 \qquad (6.1.4)$$

  defines a hyperplane.
- This hyperplane divides the input space into two parts such that at one side, the perceptron has output value +1, and in the other side, it is -1.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

- A perceptron can be used to decide whether an input vector belongs to one of the two classes, say classes A and B.
- The decision rule may be set as to respond as class A if the output is +1 and as class B if the output is -1.
- The perceptron forms two decision regions separated by the hyperplane.
- The equation of the boundary hyperplane depends on the connection weights and threshold.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

Example 6.1: When the input space is two-dimensional then the equation

$$w_1u_1+w_2u_2 + \theta= 0 \qquad (6.1.5)$$

defines a line as shown in the Figure 6.2.

This line divides the space of input variables u_1 and u_2, which is a plane, into to two separate parts.

In the given figure the elements of the classes A and B lies on the different sides of the line.

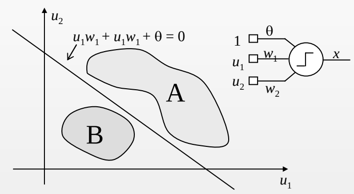


Figure 6.2. Perceptron output defines a hyperplane that divides input space into two separate subspaces

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

- Connection weights and the threshold in a perceptron can be fixed or adapted by using a number of different algorithms.
- The original perceptron convergence procedure developed by [Rosenblatt, 1959] for adjusting weights is provided in the following.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

Step 1: Initialize weights and threshold

Set each $w_j(0)$, for $j=0,1,2,..,N$, in $\mathbf{w}(0)$ to small random values. Here $\mathbf{w}=\mathbf{w}(t)$ is the weight vector at iteration time t and the component $w_0=\theta$ corresponds to the bias.

Step 2. Present New Input and Desired output:

Present new continuous valued input vector $\mathbf{u}^k$ along with the desired output y^k, such that:

$$y^k = \begin{cases} +1 & \text{if input is from class } A \\ -1 & \text{if input is from class } B \end{cases}$$

Step 3. Calculate actual output

$$x^k = \mathrm{f}(\mathbf{w}^T \mathbf{u}^k)$$

Step 4. Adapt weights

$$\mathbf{w}(t+1)=\mathbf{w}(t)+\eta\,(y^k-x^k(t))\,\mathbf{u}^k$$

where η is a positive gain fraction less than 1

Step 5. Repeat steps 2-4 until no error occurs

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

Example 6.2:

Figure 6.3 demonstrates how the line defined by the perceptrons parameters is shifted in time as the weights are updated.

Although it is not able to separate the classes A and B with the initial weights assigned at time $t=0$, it manages to separate them at the end.

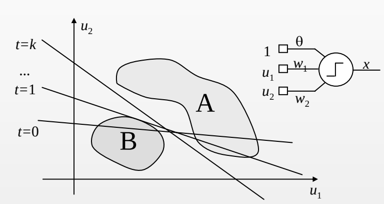


Figure 6.3. Perceptron convergence

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.1. Perceptron Convergence Procedure

- In [Rosenblatt, 1959] it is proved that if the inputs presented from the two classes are separable, that is they fall on opposite sides of some hyperplane, then the perceptron convergence procedure always converges in time. Furthermore, it positions the final decision hyperplane such that it separates the samples of class A from those of class B.
- One problem with the perceptron convergence procedure is that decision boundary may oscillate continuously when the distributions overlap or the classes are not linearly separable (Figure 6.4).

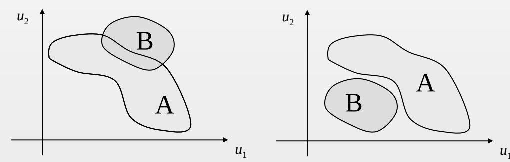


Figure 6.4. (a) Overlapping distributions
(b) non linearly separable distribution

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

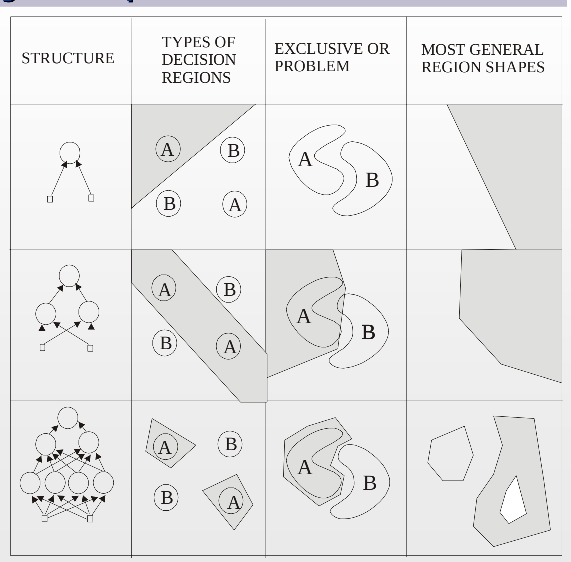


Figure 6.5. Types of regions that can be formed by single and multi-layer perceptrons (Adapted from Lippmann 89)

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.2 LMS Learning Rule

- A modification to the perceptron convergence procedure forms the Least Mean Square (LMS) solution for the case that the classes are not separable.
- This solution minimizes the mean square error between the desired output and the actual output of the processing element.
- The LMS algorithm was first proposed for Adaline (Adaptive Linear Element) in [Widrow and Hoff 60].

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.2 LMS Learning Rule

- The structure of Adaline is shown in the Figure 6.6. The part of the Adaline that executes the summation is called Adaptive Linear Combiner

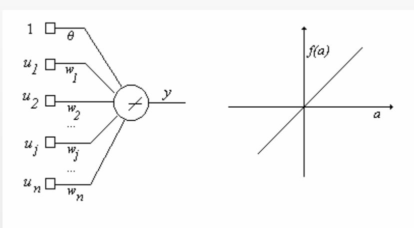


Figure 6.6 Adaline

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.2 LMS Learning Rule

- The output function of the Adaline can be represented by the identity function as:

$$f(a)=a \tag{6.2.1}$$

- So the output can be written in terms of input and weights as:

$$x = f(a) = \sum_{j=0}^{N} w_j u_j \tag{6.2.2}$$

where the bias is implemented via a connection to a constant input u_0, which means the input vector and the weight vector are of space $R^{(N+1)}$ instead of R^N.

- The output equation of Adaline can be written as:

$$x=\mathbf{w}^T\mathbf{u} \tag{6.2.3}$$

where $\mathbf{w}$ and $\mathbf{u}$ are weight and input vectors respectively having dimension $N+1$.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.2 LMS Learning Rule

- Suppose that we have a set of input vectors $\mathbf{u}^k$, $k=1..K$, each having its own desired output value y^k.
- The performance of the Adaline for a given input value $\mathbf{u}^k$ can be defined by considering the difference between the desired output y^k and the actual output x^k, which is called error and denoted as ε.
- Therefore, the error for the input $\mathbf{u}^k$ is as follows:

$$\varepsilon^k=y^k-x^k=y^k-\mathbf{w}^T\mathbf{u}^k \tag{6.2.4}$$

- The aim of the LMS learning is to adjust the weights through a training set $\{(\mathbf{u}^k,y^k)\}$, $k=1..K$, such that the mean of the square of the errors is minimum.
- The mean square error is defined as:

$$<(\varepsilon^k)^2> = \lim_{k\to\infty} \tfrac{1}{K}\sum_{k=1}^{K}(\varepsilon^k)^2 \tag{6.2.5}$$

where the notation <.> denotes the mean value.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.2 LMS Learning Rule

- The mean square error can be rewritten as:

$$<(\varepsilon^k)^2> = <(y^k - \mathbf{w}^{\mathsf{T}}\mathbf{u}^k)^2>$$
$$= <(y^k)^2> + \mathbf{w}^{\mathsf{T}} <\mathbf{u}^k \times \mathbf{u}^k> \mathbf{w} - 2 <y^k \mathbf{u}^{k\mathsf{T}}> \mathbf{w} \qquad (6.2.6)$$

where T denotes transpose and x is the outer vector product.

- Defining input correlation matrix **R** and a vector **P** as

$$\mathbf{R} = <\mathbf{u}^k \times \mathbf{u}^k> = <\mathbf{u}^k \mathbf{u}^{k\mathsf{T}}> \qquad (6.2.7)$$
$$\mathbf{P} = <y^k \mathbf{u}^k> \qquad (6.2.8)$$

results in:

$$e(\mathbf{w}) = <(\varepsilon^k)^2> = <(y^k)^2> + \mathbf{w}^{\mathsf{T}}\mathbf{R}\,\mathbf{w} - 2\mathbf{P}^{\mathsf{T}}\mathbf{w} \qquad (6.2.9)$$

- The optimum value **w*** for the weight vector corresponding to the minimum of the mean squared error can be obtained by evaluating the gradient of $e(\mathbf{w})$.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.2 LMS Learning Rule

- The point which makes the gradient zero gives us the value of **w***. That is:

$$\nabla e(\mathbf{w})\Big|_{\mathbf{w}=\mathbf{w}^*} = \frac{\partial e(\mathbf{w})}{\partial \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}^*} = 2\mathbf{R}\mathbf{w}^* - 2\mathbf{P} = 0 \qquad (6.2.10)$$

- Here, the gradient is

$$\nabla e(\mathbf{w}) = \left[\partial e/\partial w_1 \quad \partial e/\partial w_2 \quad \dots \quad \partial e/\partial w_n \right]^{\mathsf{T}} \qquad (6.2.11)$$

and it is a vector extending in the direction of the greatest rate of change.

- The gradient of a function evaluated at some point is zero if the function has a maximum or minimum at that point.

- The error function is of the second degree. So it is a paraboloid and it has a single minimum at point **w***.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.2 LMS Learning Rule

- When we set the gradient of the mean square error to zero, this implies that

$$\mathbf{R}\mathbf{w}^*=\mathbf{P} \qquad (6.2.12)$$

and then

$$\mathbf{w}^*=\mathbf{R}^{-1}\mathbf{P} \qquad (6.2.13)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.3 Steepest Descent Algorithm.

- The analytical calculation of the optimum weight vector for a problem is rather difficult in general.
- Not only does the matrix manipulation get cumbersome for the large dimensions, but also each component of **R** and **P** itself is an expectation value.
- Thus, explicit calculations of **R** and **P** require knowledge of the statistics of the input signal [Freeman 91].
- A better approach would be to let the Adaline Linear Combiner to find the optimum weights by itself through a search over the error surface.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.3 Steepest Descent Algorithm.

- Instead of having a purely random search, some intelligence is added to the procedure such that the weight vector is changed by considering the gradient of $e(\mathbf{w})$ iteratively [Widrow 60], according to formula known as ***delta rule*:**

$$\mathbf{w}(t+1)=\mathbf{w}(t)+\Delta\mathbf{w}(t) \quad (6.3.1)$$

where

$$\Delta\mathbf{w}(t)=-\eta\nabla e(\mathbf{w}(t)) \quad (6.3.2)$$

In the above formula η is a small positive constant.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.3 Steepest Descent Algorithm.

- For the real valued scalar function $e(\mathbf{w})$ on a vector space $\mathbf{w}\in R^N$, the gradient $e(\mathbf{w})$ gives the direction of the steepest upward slope, so the negative of the gradient is the direction of the steepest descent. This fact is demonstrated in Figure 6.7 for a parabolic error surface on two dimensions.

- In Section 6.2 we have considered the linear output function in the derivation of the optimum weight $\mathbf{w}^*$ for the minimum error. However in the general case, we should consider any nonlinearity $f(.)$ at the output of the neuron. It should be noted that in such a case the error surface is no more a paraboloid, so it may have several local minima.

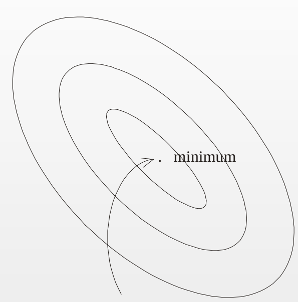


Figure 6.7 Direction of the steepest gradient descent on the paraboloid error surface on two-dimensional weight space. Only the equpotential curves of the error surface is shown instead of the 3D-error surface.

ANN ee543

CHAPTER VI : ***Learning in Feedforward Neural Networks***

6.3 Steepest Descent Algorithm.

- For an input $\mathbf{u}^k$ applied at time t, $(\varepsilon^k(t))^2$ can be used as an approximation to $<\varepsilon(k)^2>$, where

$$\varepsilon^k(t)=y^k-f(a^k)=y^k-f(\mathbf{w}(t)^T\mathbf{u}^k) \quad (6.3.3)$$

- Therefore, we obtain:

$$\nabla<(\varepsilon^k)^2> \simeq \nabla(\varepsilon^k(t))^2 = \nabla(y^k-f(a^k))^2 \quad (6.3.4)$$

- With a differentiable function $f(.)$ having derivative $f'(.)$, it becomes

$$\nabla(y^k-f(a))^2= -2\,\varepsilon^k(t)\, f'(a)\, \nabla a \quad (6.3.5)$$

ANN ee543

CHAPTER VI : ***Learning in Feedforward Neural Networks***

6.3 Steepest Descent Algorithm.

- Since

$$\nabla a^k= \nabla\, \mathbf{w}(t)^T\mathbf{u}^k=\mathbf{u}^k \quad (6.3.6)$$

the weight update formula becomes:

$$\mathbf{w}(t+1)=\mathbf{w}(t)+2\eta\varepsilon^k(t)\, f'(a)\mathbf{u}^k. \quad (6.3.7)$$

- Notice that for Adaline's linear output function:

$$f'(a)=1 \quad (6.3.8)$$

- For sigmoid function it is:

$$f'(a)=\frac{\partial}{\partial a}\left(\frac{1}{1+e^{-a/T}}\right)=\frac{1}{T}f(a)(1-f(a)) \quad (6.3.9)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.3 Steepest Descent Algorithm.

Step 1: Apply an input vector $\mathbf{u}^k$ with an desired output value y^k to the neuron's input
Step 2: By considering $\mathbf{u}^k$ and using the current value of the weight vector determine the value of the activation vector a^k:

$$a^k=\mathbf{w}(t)^{\mathbf{T}}\mathbf{u}^k$$

Step 3: Determine the value of the derivative of the output function using the current value of activation a^k, that is:

$$f'(a^k)=\left.\frac{\partial f(a)}{\partial a}\right|_{a=a^k}$$

Step 4: Determine the value of error $\varepsilon^k(t)$ as:

$$\varepsilon^k(t)=y^k-f(a^k)$$

Step 5: Update the weight vector with respect to following update formula

$$\mathbf{w}(t+1)=\mathbf{w}(t)+2\eta f'(a^k)\,\varepsilon^k(t)\mathbf{u}^k$$

Step 6: Repeat steps 1-5 until $<\varepsilon^k(t)>$ reduces to an acceptable level.

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.3 Steepest Descent Algorithm.

- The parameter η in the algorithm determines the stability and the speed of convergence of the weight vector towards the minimum error value.
- The value of η should be tuned well. If it is chosen too small this effects considerably the convergence time.
- On the other hand, if changes are too large, the weight vector may wander around the minimum as shown in the Figure 6.8, without being able to reach it.

Figure 6.8. Inappropriate value of learning rate may cause oscillations in the weight values without convergence

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.3 Steepest Descent Algorithm.

- Notice that, the iterative weight update by the delta rule is derived by assuming constant $\mathbf{u}^k$.
- Therefore, it tends to minimize the error with respect to applied $\mathbf{u}^k$.
- In fact, we require the average error, that is:

$$e = <(\varepsilon^k)^2> = \tfrac{1}{K}\sum_{k=1}^{K}(\varepsilon^k)^2 \qquad (6.3.10)$$

to be minimum

- This implies that

$$\frac{\partial e}{\partial w_j} = \tfrac{1}{K}\sum_{k=1}^{K}\frac{\partial(\varepsilon^k)^2}{\partial w_j} = \tfrac{1}{K}\sum_{k=1}^{K}\frac{2\varepsilon^k\partial\varepsilon^k}{\partial w_j} \qquad (6.3.11)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.3 Steepest Descent Algorithm.

- Therefore, the net change in w_j after one complete cycle of pattern presentation is expected to be:

$$w_j(t+K) = w_j(t) - \eta\frac{1}{K}\sum_{k=1}^{K}\frac{2\varepsilon^k\partial\varepsilon^k}{\partial wj} \qquad (6.3.12)$$

- However, this would be true that if the weights are not updated along a cycle, but only at the end.
- By changing the weights as each pattern is presented, we depart to some extend from the gradient descent in *e*.
- Nevertheless, provided the learning rate is sufficiently small, this departure will be negligible and the delta rule will implement a very close approximation to gradient descent in mean squared error [Freeman 91].

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Single Layer Network

- Consider a single layer multiple output network as shown in the Figure 6.9.
- Here, we still have N inputs denoted u_j, $j=1..N$, but M processing elements whose activations and outputs are denoted as a_i and x_i , $i=1..M$ respectively.
- w_{ji} is used to denote the strength of the connection from the j^{th} input to the i^{th} processing element.
- In vector notation w_{ji} is the j^{th} component of weight vector $\mathbf{w}_i$, while u_j is the j^{th} component of the input vector $\mathbf{u}$.

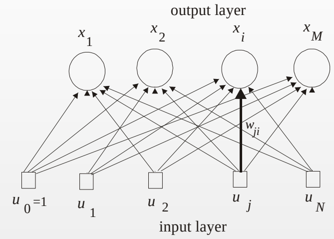


Figure 6.9. Single Layer Multiple output network

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

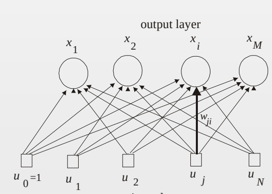
6.4. The Backpropagation Algorithm : Single Layer Network

- Let $\mathbf{u}^k$ and $\mathbf{y}^k$ to represent the k^{th} input sample and the corresponding desired output vector respectively
- Let the error observed at the output i, when $\mathbf{u}^k$ is applied at the input, be

$$\varepsilon_i^k = y_i^k - x_i^k \qquad (6.4.1)$$

- If the error is to be written in terms of the input vector $\mathbf{u}^k$ and the weights $\mathbf{w}_i$, we obtain

$$\varepsilon_i^k = y_i^k - f(\mathbf{w}_i^{\mathsf{T}}\mathbf{u}^k) \qquad (6.4.2)$$



ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Single Layer Network

- If we take partial derivative with respect to w_{ji} by applying the chain rule

$$\frac{\partial \varepsilon_i^k}{\partial w_{ji}} = \frac{\partial \varepsilon_i^k}{\partial x_i^k}\frac{\partial x_i^k}{\partial w_{ji}} \qquad (6.4.3)$$

where

$$\frac{\partial \varepsilon_i^k}{\partial x_i^k} = -1 \qquad (6.4.4)$$

and

$$\frac{\partial x_i^k}{\partial w_{ji}} = f'(a_i^k)u_j^k \qquad (6.4.5)$$

we obtain

$$\frac{\partial \varepsilon^k}{\partial w_{ij}} = -f'(a^k)u_j^k \qquad (6.4.6)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Single Layer Network

- If we define the total output error for input $\mathbf{u}^k$ as the sum of the square of the errors at each neuron output, that is:

$$e^k = \frac{1}{2}\sum_{i=1}^{m}(\varepsilon_i^k)^2 \qquad (6.4.7)$$

then partial derivative of the total error with respect to w_{ji} when $\mathbf{u}^k$ is applied at the input can be written as:

$$\frac{\partial e^k}{\partial w_{ji}} = \frac{\partial e^k}{\partial \varepsilon_i^k}\frac{\partial \varepsilon_i^k}{\partial w_{ji}} \qquad (6.4.8)$$

which is

$$\frac{\partial e^k}{\partial w_{ji}} = -\varepsilon_i^k f'(a^k)u_j \qquad (6.4.9)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Single Layer Network

- By defining

$$\delta_i^k = \varepsilon_i^k f'(a^k) \qquad (6.4.10)$$

it can be reformulated as

$$\frac{\partial e^k}{\partial w_{ji}} = -\delta_i^k u_j^k \qquad (6.4.11)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Single Layer Network

- For the error to be minimum, the gradient of the total error with respect to the weights should be

$$\nabla e^k = \mathbf{0} \qquad (6.4.12)$$

where $\mathbf{0}$ is the vector having $N.M$ entries each having value zero.

- In other words, it should be satisfied:

$$\frac{\partial e^k}{\partial w_{ji}} = 0 \quad for\ \ j = 1..N,\ i = 1..M \qquad (6.4.13)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Single Layer Network

- In order to reach the minimum of the total error, without solving the above equation, we apply the delta rule in the same way explained for the steepest descent algorithm:

$$\mathbf{w}(t+1)=\mathbf{w}(t)-\nabla \mathrm{e}^k \qquad (6.4.14)$$

in which

$$w_{ji}(t+1)=w_{ji}(t)-\eta\frac{\partial \mathrm{e}^k}{\partial w_{ji}} \quad \textit{for } j=1..N,\ i=1..M \qquad (6.4.15)$$

that is

$$w_{ji}(t+1)=w_{ji}(t)+\eta\delta_i^k u_j^k \quad \textit{for } j=1..N,\ i=1..M \qquad (6.4.16)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

- Now assume that another layer of neurons is connected to the input side of the output layer.
- Therefore we have the input, hidden and the output layers as shown in Figure 6.10. In order to discriminate between the elements of the hidden and output layers we will use the subscripts L and o respectively.
- Furthermore, we will use h as the index on the hidden layer elements, while still using index j and i for the input and output layers respectively.

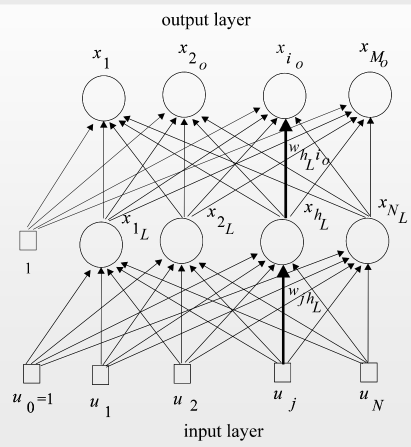


Figure 6.10 Multilayer network

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

- In such a network, the output value of i^{th} neuron of output layer can be written as:

$$x_{i_o}^k = f_o(\mathbf{w}_{i_o}{}^{\mathrm{T}}\mathbf{x}_L^k) \qquad (6.4.17)$$

where $\mathbf{x}_L^k$ being the vector of output values at hidden layer that is connected as input to the output layer. The value of the h^{th} element in $\mathbf{x}_L{}^k$ is determined by the equation:

$$x_{h_L}^k = f_L(\mathbf{w}_{h_L}{}^{\mathrm{T}}\mathbf{u}^k) \qquad (6.4.18)$$

- Notice that

$$\mathbf{w}_{h_L}{}^{\mathrm{T}}\mathbf{u}^k = \sum_{j=1}^{N} w_{jh_L} u_j^k \qquad (6.4.19)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

- The partial derivative of the output of a neuron i_o of output layer with respect to hidden layer weight w_{jhL} can be determined by applying the chain rule

$$\frac{\partial x_{i_o}^k}{\partial w_{jh_L}} = \frac{\partial x_{i_o}^k}{\partial x_{h_L}^k}\frac{\partial\, x_{h_L}^k}{\partial\, w_{ij_L}} \qquad (6.4.20)$$

- By using Eq. (6.4.17) and (6.4.19) this can be written as

$$\frac{\partial x_{i_o}^k}{\partial w_{jh_L}} = (f_o'(a_{i_o}^k) w_{h_L i_o})(f_L'(a_{h_L}^k) u_j^k) \qquad (6.4.21)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

- Then the partial derivative of the total error

$$e^k = \tfrac{1}{2}\sum_{i_o=1}^{M}(\varepsilon^k_{i_o})^2 = \tfrac{1}{2}\sum_{i_o=1}^{M}(y^k_{i_o} - x^k_{i_o})^2 \qquad (6.4.22)$$

with respect to the hidden layer weight w_{jh_L} can be written as

$$\frac{\partial e^k}{\partial w_{jh_L}} = -\sum_{i_o=1}^{M}\varepsilon^k_{i_o} f'_o(a^k_{i_o}) w_{h_L i_o} f'_L(a^k_{h_L}) u^k_j \qquad (6.4.23)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

- It can be reformulated as

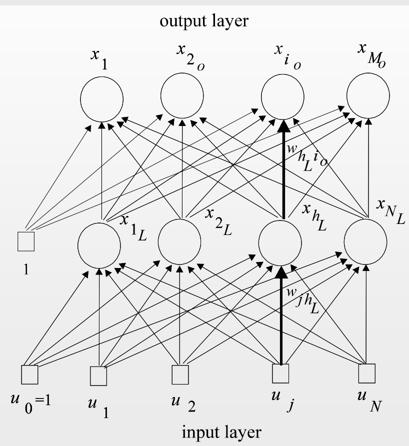
$$\frac{\partial e^k}{\partial w_{jh_L}} = -\sum_{i_o=1}^{M}\delta^k_{i_o} w_{h_L i_o} f'_L(a^k_{h_L}) u^k_j \qquad (6.4.24)$$

- When defined

$$\delta^k_{h_L} = f'_L(a^k_{h_L}) \sum_{i_o=1}^{M}\delta^k_{i_o} w_{h_L i_o} \qquad (6.4.25)$$

it becomes

$$\frac{\partial e^k}{\partial w_{jh_L}} = -\delta^k_{h_L} u_j \qquad (6.4.26).$$



ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

- Therefore, the weight update rule for the hidden layer

$$w_{jh_L}(t+1) = w_{jh_L}(t) - \eta \frac{\partial e^k}{\partial w_{jh_L}} \qquad (6.4.27)$$

can be reformulated in analogy with the weight update rule of the output layer, as

$$w_{jh_L}(t+1) = w_{jh_L}(t) + \eta \delta^k_{h_L} u_j \qquad (6.4.28)$$

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

- This weight update rule may be generalized for the networks having several hidden layers as:

$$w_{j_{(L-1)}h_L}(t+1) = w_{j_{(L-1)}h_L}(t) + \eta \delta^k_{h_L} x_{j_{(L-1)}} \qquad (6.4.29)$$

where *L* and (*L*-1) are used to denote any hidden layer and its previous layer respectively.

- Furthermore,

$$\delta^k_{j_{(L-1)}} = f'_{L-1}(a^k_{j_{(L-1)}}) \sum_{h_L=1}^{N_L} \delta^k_{h_L} w_{j_{(L-1)}h_L} \qquad (6.4.30)$$

where N_L is the number of neurons at layer *L.*

ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

Step 0. Initialize weights: to small random values;

Step 1. Apply a sample: apply to the input a sample vector $\mathbf{u}^k$ having desired output vector $\mathbf{y}^k$;

Step 2. Forward Phase:

Starting from the first hidden layer and propagating towards the output layer:

2.1. Calculate the activation values for the units at layer *L* as:

2.1.1. If *L*-1 is the input layer

$$a^k_{h_L} = \sum_{j=0}^{N} w_{jh_L} u^k_j$$

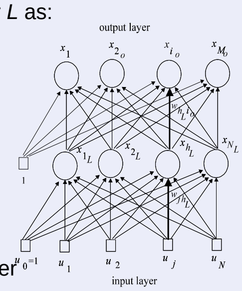
2.1.2. If *L*-1 is a hidden layer

$$a^k_{h_L} = \sum_{j_{L-1}=0}^{N_{L-1}} w_{j_{(L-1)}h_L} x^k_{j_{(L-1)}}$$

2.2. Calculate the output values for the units at layer *L* as:

$$x^k_{h_L} = f_L(a^k_{h_L})$$

in which use index i_o instead of h_L if it is an output layer



ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

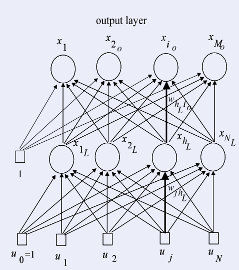
Step 4. Output errors: Calculate the error terms at the output layer as:

$$\delta^k_{i_o} = (y^k_{i_o} - x^k_{i_o}) f'_o(a^k_{i_o})$$

Step 5. Backward Phase Propagate error backward to the input layer through each layer *L* using the error term

$$\delta^k_{h_L} = f'_L(a^k_{h_L}) \sum_{i_{L+1}=1}^{N_{L+1}} \delta^k_{i_{(L+1)}} w^k_{h_L i_{(L+1)}}$$

in which, use i_o instead of $i_{(L+1)}$ if $L+1$ is an output layer;



ANN ee543

CHAPTER VI : *Learning in Feedforward Neural Networks*

6.4. The Backpropagation Algorithm : Multi Layer Network

Step 6. Weight update: Update weights according to the formula

$$w_{j_{(L-1)}h_L}(t+1) = w_{j_{(L-1)}h_L}(t) + \eta\delta^k_{h_L} x^k_{j_{(L-1)}}$$

Step7. Repeat steps 1-6 until the stop criterion is satisfied, which may be chosen as the mean of the total error

$$< e^k > = < \tfrac{1}{2}\sum_{i_o=1}^{M}(y^k_{i_o} - x^k_{i_o})^2 >$$

is sufficiently small.