MODULE - 2

SUPERVISED LEARNING

Perceptron learning and Non Separable sets, a-Least Mean Square Learning, MSE Error surface, Steepest Descent Search, μ-LMS approximate to gradient descent, Application of LMS to Noise Cancelling, Multi-layered Network Architecture, Backpropagation Learning Algorithm, Practical consideration of BP algorithm.

Perceptron learning and Non Separable sets

The convergence of the perceptron learning algorithm has been proved for pattern sets that are known to be linearly separable. No such guarantee exists for linearly non separable cases because in weight space, no solution core exists.

When a set of training patterns is not linearly separable, then for any set of weights w_k there will exist some training vector x_k such that w_k misclassifies x_k .

Consequently, the perceptron learning algorithm will continue to make weight changes indefinitely (weight perturbation never stops) .What may happen if perceptron learning is linearly non separable and what happens to the weights.

Can they grow arbitralily large? No. There will be misclassification for some samples.

For a finite set of training pattern X, if the individual patterns x_k have integer or rational components x_i^k . , the perceptron learning,

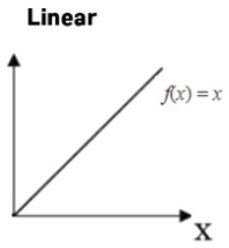
1. Produce weight vector that correctly classifies all training patterns if and only if X is linearly separable.
2. Leave and revisit a specific weight vector if X is linearly non separable.

1. Explain α LMS algorithm

In the perceptron learning algorithm, the objective was to achieve a linear separation of input patterns by attempting to correct the classification error on a misclassified pattern in each iteration. The restriction was put on the activation to be a binary threshold so that the desired outputs are $d_k = \{0,1\}$ or $\{-1,1\}$ and the patterns are linearly separable.

In LMS, we remove these restrictions and consider a training set of the form $T = \{X_k , d_k \}$ where $X_k = x_0 , x_1 , x_2 , \ldots x_n$ are the training input data and d_k is the desired output for each kth iteration.

To allow desired output to vary continuously and smoothly over some interval, we simply change the activation function, from binary threshold to **linear**. (a linear activation function f(x) = x where the input is the same as the output) .



The signal output or the actual output or the net activation of the neuron, s_k , then equals to the desired output d_k which is

$S_k = d_k = X_k^T W_k$

Now, we define a linear error , e_k , related to this training pair $\{X_k , d_k\}$ as the difference between the desired output and the actual output

$$e_k = d_k - s_k = d_k - x_k^T w_k$$

In α LMS algorithm, the objective is to minimize this linear error measure over patterns of the training set T.

Operation details of the α LMS :

1. Incorporating a linear error measure into the weight space update procedure yields α LMS learning algorithm.
2. α LMS applied to a single adaptive linear neuron is depicted by

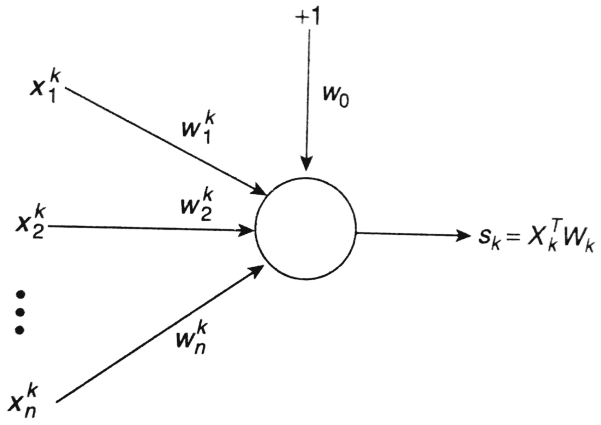


Fig. 5.20 *An adaptive linear neuron.*

3. The recursive weight updating equation for α lms is :

$$w_{k+1} = w_k + \eta\, e_k \frac{x_k}{\|x_k\|^2}$$

(w_{k+1} = weight vector to be calculated, w_k = weight vector, η = learning rate, e_k = linear error, x_k = input vector in normalised form (mod square))

Where the weight vector w_k is modified by the product of scaled error e_k and normalised input vector x_k.

Any weight update is generally represented by :

$$w_{k+1} = w_k + \Delta w_k$$

where $\Delta w_k = \eta\, e_k \frac{x_k}{\|x_k\|^2}$

$$\Rightarrow \Delta w_k = \eta \frac{e_k}{\|x_k\|} \frac{x_k}{\|x_k\|}$$

$$\Rightarrow \Delta w_k = \hat{\eta}_k\, e_k\, \hat{x}_k$$

Here $\hat{x}_k$ = unit vector in the direction X_k, $\hat{\eta}_k$ is a pattern- normalised learning rate.

NOTE: 1. weights are changed in the direction of x_k as in perceptron learning algorithm except that a unit vector $\hat{x}_k$ is used in place. 2. The learning rate is scaled from iteration to iteration (unit vector) which makes this algorithm self-normalising. The advantage of normalising x_k is that the larger magnitude vectors do not dominate the weight space update process.

Working of the α LMS :

We calculate the error for changing weights and static x_k . The change in error for x_k depends on the extent of change in w_k.

$$e_{k+1} = d_k - x_k^T w_{k+1} \quad \text{—(1)}$$

$$e_k = d_k - x_k^T w_k \quad \text{—(2)}$$

$$\Delta e_k = e_{k+1} - e_k \qquad \text{(substitute (1) and (2))}$$

$$\Delta e_k = d_k - x_k^T w_{k+1} - d_k - x_k^T w_k$$

$$\Delta e_k = - x_k \Delta w_k^T$$

$$\left(\Delta w_k = \eta\, e_k \frac{x_k}{\|x_k\|^2}\right)$$

$$\Delta e_k = - x_k\, \eta\, e_k \frac{x_k}{\|x_k\|^2}$$

$$\Delta e_k = - \eta\, e_k$$

We conclude that the error correction is proportional to error e_k itself. And with each iteration (k) reduces the error by a factor of the learning rate η.

Although alpha LMS is a linear error correction rule, in the sense that it makes error correction in proportion to the instantaneous pattern error, it also uses the instantaneous gradient estimate while attempting to minimize the mean squared error (MSE) over the training set T.

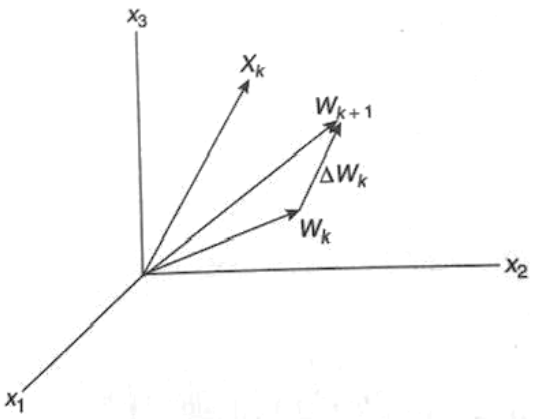


Fig. 5.21 *Weight update takes place in a direction of the input vector.*

α LMS working with a normalised training pattern set :

$$w_{k+1} = w_k + \eta\, e_k \frac{x_k}{\|x_k\|^2}$$

$$w_{k+1} = w_k + \eta \frac{e_k}{\|x_k\|} \frac{x_k}{\|x_k\|}$$

$$w_{k+1} = w_k + \eta \hat{e}_k \hat{x}_k$$

Where $\hat{e}_k = \frac{e_k}{\|x_k\|}$ and $e_k = d_k - w_k^T x_k$

So $\hat{e}_k = \frac{d_k - w_k^T x_k}{\|x_k\|}$

$$\Rightarrow \hat{e}_k = d_k - w_k \hat{x}_k$$

where d_k is the normalised desired value and x_k is the normalised input pattern, e_k is the error computed using the normalised training set. (Weight update takes place in the direction of the input vector in an attempt to reduce the error equation.)

Characteristics of LMS :

1. The LMS algorithm is **linear** with respect to adjustable parameters which makes the algorithm computationally efficient.
2. It is effective in performance.
3. It is simple to code and easy to build.
4. It is robust to noise or any external disturbances in the training input.

2. Derive the expression for the Weiner solution and MSE surface when the samples are drawn from a statistical stationary process.

Error correction rules such as the α LMS has the effect of reducing the error on individual patterns using simple linear error measures. More suitable error correction rule is the mean square error (MSE) function.

Until now we have assumed that the training set, T, is well defined in advance. It is finite and its component pattern completely specified beforehand. Often when the setting is stochastic (or incremental) , we have a sequence of samples x_k, d_k which are drawn from a statistical stationing process (an unchanging environment (output)) . Therefore if we want to adjust the neuron weight in response to some pattern dependent error measure, the error computation has to be based on the expectation of error ensemble (altogether)

[MSE is a risk function that calculates the average of the squared error values. A larger MSE indicates that the data points are dispersed widely around its central moment (mean), whereas a smaller MSE suggests the opposite. A smaller MSE is preferred because it indicates that your data points are dispersed closely around its central moment or mean.]

This can be done by taking the expectation of the squared error. (*Expectation E(x) is the expected or estimated value of x and can be thought of as the average / mean value attained by x. In other words , expectation is **a** generalization of the weighted average.*)

So we introduce a squared error on pattern X_k as

(recall $e_k = d_k - S_k$ and $S_k = X_k^T W_k$)

$$\varepsilon_k = \frac{1}{2} e_k^2$$

$$= \frac{1}{2} (d_k - x_k^T w_k)^2 \qquad (a-b)^2$$

$$\varepsilon_k = \frac{1}{2} (d_k^2 - 2 d_k x_k^T w_k - x_k^T w_k x_k^T w_k)$$

so mean squared error (MSE) would be the expectation of all e_k (denoted as Ek)

$$\varepsilon = E(\varepsilon_k)$$

$$\varepsilon = \frac{1}{2} E[d_k^2] - E[d_k x_k^T] w_k + E[x_k^T x_k] w_k w_k^T$$

Here assumption is that weight w_k is held fixed while computing the expectation and our objective here is to find the optimal weight vector w that minimizes mean squared error MSE (E). (Implies that we don't take the expected value of w_k (E $[w_k]$) in the expression , and consider it to be a constant.)

for convenience of expression let's use P^T as probability term for $E[d_k, x_k]$ as :

$$P^T = E[d_k x_k^T]$$

$$= E[d_k , d_k x_1^k , \ldots , d_k x_n^k]$$

where P^T is the cross correlation between desired scaler output d_k and input vector x_k. and let $R = E[x_k]$ which is the correlation matrix computed over pattern x_k .

$$R = E[x_k^T x_k]$$

Substitute R and P^T in ε

$$\varepsilon = \frac{1}{2} E[d_k^2] - P^T w_k + \frac{1}{2} w_k R w_k^T \qquad —(1)$$

eqn 1 plotting this would be exponential. Since MSE is a quadratic function and represents a bowl shaped surface. To find the optimal value by the computational gradient $\nabla\varepsilon$ and set it to 0.

$$\nabla\varepsilon = \frac{\partial\varepsilon}{\partial w_n} = 0 \quad \text{for k =0 to n}$$

So to find the minimum, Emin, we compute the gradient by differentiating eqn 1 wrt w

$$\varepsilon_{min} = \frac{\partial\varepsilon}{\partial w_k}$$

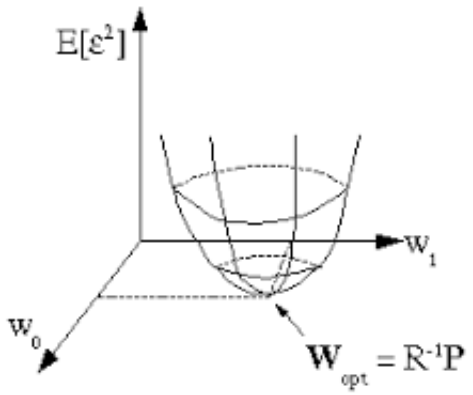
$$\varepsilon_{min} = E[d_k] - P^T + Rw_k \quad (E[d_k] = 0)$$

$$0 = -P^T + Rw_k$$

$$\Rightarrow Rw_k = P$$

$$\Rightarrow w_k = R^{-1}P$$

$$\hat{w} = R^{-1}P$$



This is called the Weiner - Hopf system and its solution is the Weiner solution $\hat{w}$ (given w_{opt} in diagram). This is the point in weight space that represents the minimum mean square error , $\varepsilon_{min.}$ So if we need to compute the optimal filter, we only need to compute the R^{-1} and P matrices. Substitute w_k to $\hat{w}$ in eqn (1)

$$\varepsilon_{min} = \frac{1}{2} E[d_k^2] - P^T \hat{w} + \frac{1}{2} \hat{w} R \hat{w}^T$$

$$= \frac{1}{2} E[d_k^2] - P^T R^{-1} P + \frac{1}{2} R^{-1} P * R * R^{-1} P$$

$$\varepsilon_{min} = \frac{1}{2} E[d_k^2] - \frac{1}{2} P^T \hat{w}$$

For the treatment of the weight update process, we reformulate the expression for mean square error in terms of deviation V = W - $\hat{w}$. substitute this in (1)

$$w = v + \hat{w}$$

$$\varepsilon = \frac{1}{2} E[d_k^2] - P^T w_k + \frac{1}{2} w_k R w_k^T \qquad —(1)$$

$$\varepsilon = \frac{1}{2} E[d_k^2] - P^T \hat{w} + \frac{1}{2} \hat{w} R \hat{w}^T$$

$$= \frac{1}{2} E[d_k^2] - P^T R^{-1} P + \frac{1}{2} R^{-1} P * R * R^{-1} P$$

$$\varepsilon = \varepsilon_{min} + \frac{1}{2} E[d_k^2] - \frac{1}{2} P^T \hat{w} \quad (\text{substitute } \varepsilon_{min})$$

$$\varepsilon = \varepsilon_{min} + \frac{1}{2}(w - \hat{w}) R (w - \hat{w}) \qquad —(A)$$

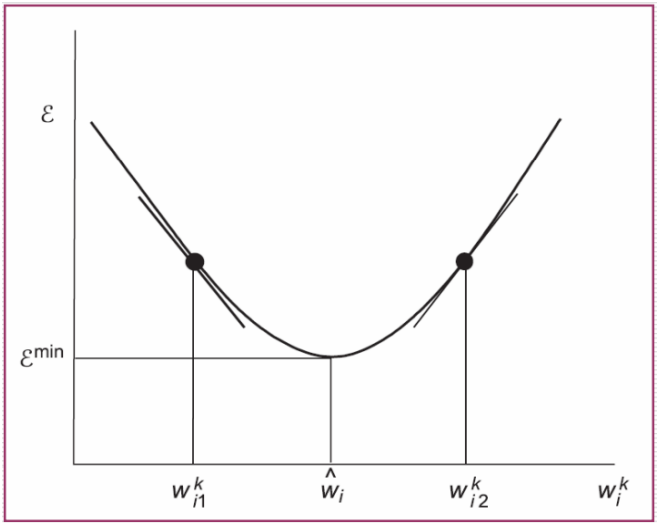
This is the equation for the weiner solution in terms of MSE .

3. Discuss weight update process in steepest descent search algorithm.

Steepest descent search is a straightforward method to find the Wiener solution $\widehat{w}$,through **iterative update of weights**. It uses **exact gradient information** available from the MSE surface to direct the search in weight space.

It provides an appropriate weight increment to w_i^k to push the error towards minimum which occurs at $\widehat{w_i}$ (minimum error $w_i^k = \widehat{w_i}$)

In order to move towards the minimum, the weight to be perturbed or tuned in a direction depends on which side of optimal weight $\widehat{w_i}$, the current weight value w_i^k lies (or where w_i^k is placed)



Suppose if the point is w_i^k lies to the left of $\widehat{w_i}$, say at w_{i1}^k, where the error gradient is negative , hence the algorithm increases weight w_i^k. If the point is at w_{i2}^k, the error gradient is positive and the algorithm decreases the weight w_i^k.

$$\text{If } \frac{\partial \varepsilon}{\partial w_i^k} < 0 \qquad (w_i^k < \widehat{w}) \text{ increase } w_i^k$$

$$\text{If } \frac{\partial \varepsilon}{\partial w_i^k} > 0 \qquad (w_i^k > \widehat{w}) \text{ decrease } w_i^k$$

It follows logically that the weight component should be updated in proportion with the negative of the gradient as follows :

$$W_i^{k+1} = w_k + \eta \left(- \frac{\partial \varepsilon}{\partial w_i^k}\right)$$

In the vector form, we represent it as :

$$w_{k+1} = w_k + \eta(-\nabla\varepsilon) \qquad \text{—(B)}$$

NOTE : Steepest descent search uses the exact gradient information at each step to decide the weight changes.

Condition for convergence :

Steepest descent is guaranteed to converge to the Wiener solution as long as η is maintained within the limit given as

$$0 < \eta < \frac{2}{\lambda_{max}}$$

Where λ_{max} is the maximum eigenvalue of the R matrix.(E [$x_k x_k^T$] =*R refers to the matrix of* $\widehat{w_i}$=$R^{-1}P$*)* When we differentiate the equation for weiner solution in terms of MSE (refer eqn (A)) and substitute the gradient, we get

$$\nabla\varepsilon = R(w - \widehat{w})$$

So, $w_{k+1} = w_k + \eta(-\nabla\varepsilon)$ (after substitution) is now

$$w_{k+1} = w_k + \eta R(w - \widehat{w})$$

4. Explain μ-LMS (approximate gradient descent) algorithm.

The real problem with steepest descent is that true gradient information is required and this is only available in situations where the dataset is completely specified in advance. For them, it's possible to compute R and P exactly. (P is the cross correlation between desired scaler output d_k and input vector x_k. and R = E[x_k] which is the correlation matrix computed over pattern x_k) And thus, the true gradient at every iteration is known.

However, when the dataset comprises a random stream of patterns, R and P cannot be computed accurately. To find a good approximation, one might have to examine a dataset for a reasonably large period of time and keep averaging out. This is not very appealing for the simple reason that we do not know how long to examine the stream to get a reliable estimate of R and P.

μ LMS algorithm breaks this computational barrier by using ε_k directly in place of $\varepsilon = E[\varepsilon_k]$

$$\varepsilon_k = \frac{1}{2} e_k^2$$

Differentiating this ,

$$\frac{\partial \varepsilon_k}{\partial w} = e_k \frac{\partial e_k}{\partial w}$$

$(e_k = d_k - x_k^T w_k)$ differentiating this wrt 'w' gives $-x_k$

$$\frac{\partial \varepsilon_k}{\partial w} = -e_k x_k$$

Substitute this in gradient descent equation (B)

$$w_{k+1} = w_k + \eta (-\nabla \varepsilon) \quad -(B)$$

$$w_{k+1} = w_k + \eta e_k x_k$$

We can see that this eqn is not normalised unlike the α LMS,

The gradient here approximately reaches to infinity, in the long run. So, μ LMS algorithm is said to be convergent in the mean square if the average squared error becomes constant as the number of iterations approaches infinity.

$$\lim_{k \to \infty} \nabla \varepsilon = \nabla \varepsilon$$

μ LMS is identical to steepest descent and it searches for a weiner solution but not as smoothly as the steepest descent algorithm. μ LMS is a stochastic gradient algorithm.

Its trajectory follows random path towards the optimum through performing random walk about the optimum $\widehat{w}$.

Condition for convergence :

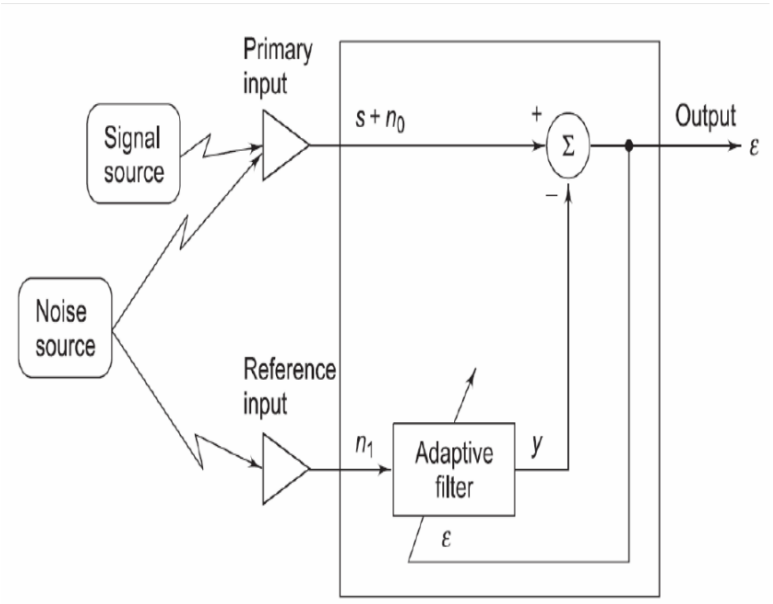
μ LMS (approximate descent) is guaranteed to converge to the Wiener solution as long as η is maintained within the limit given as

$$0 < \eta < \frac{2}{\lambda_{max}}$$

Where λ_{max} is the maximum eigenvalue of the R matrix ($E [x_k x_k^T]$).

5. Considering an example, explain the application of the LMS algorithm in noise cancellation.

The field of adaptive signal processing has benefitted alot from the LMS algorithm (adaptive linear combiner ALC) . In a filter with n signal samples input to the ALC, generates output by computing the inner product of $x_k w_k$. The filtered output is simply this inner product - a linear combination of current and past signal samples. The LMS procedure is employed to adjust the weights over time, so that output matches the desired response.



Problem in signal processing is the **removal of noise** 'n_0' from the signal 's'. This adaptive noise cancellation approach can be used only if a reference signal is available that contains a noise component 'm' which is correlated with the noise 'n_0' .

The adaptive noise canceller subtracts the filtered reference signal from the noisy input ,thereby making the output of the canceller an error signal. Assume that s , n_0 , n_1 and y are statistically independent and stationary with the zero mean signal ,

$\varepsilon = s + n_0 + y$

Squaring on both the sides,

$\varepsilon^2 = s^2 + (n_0 + y)^2 + 2\ s\ (n_0 - y\ )$

The point to be noted is n_0 and n_1 are correlated but s and n_0 and y are uncorrelated (depends on each other). Taking the estimation on both the sides,

$E\ [\ \varepsilon^2\ ] = E[\ s^2\ ] + E\ [\ (n_0 + y)^2\ ] + 2\ E\ [\ s\ (n_0 - y\ )\ ]$

Since s, n_0 ,y are uncorrelated, $s(n_0 - y\ ) = 0$

$\Rightarrow E\ [\ \varepsilon^2\ ] = E[\ s^2\ ] + E\ [\ (n_0 + y)^2\ ]$

By changing the weights of adaptive filter we cannot change signal 's' or its expectation, so we minimize the average error,

Minimizing $E\ [\ \varepsilon^2\ ]$ implies minimizing $E\ [\ (n_0 + y)^2\ ]$,

After minimizing $E\ [\ (n_0 + y)^2\ ]$ really means that the output 'y' of the adaptive filter becomes a good estimate of noise 'n_0' . Simultaneously, estimation of $E\ [\ (\varepsilon - S\ )^2\ ]$ is also minimized. LMS adaptation of filter causes output ε to be the least squares estimate of the input signal s, since the noise gets subtracted.

When y approaches n_0 the output ε becomes similar to s

Monitoring the fetal heart rate using ECG is an important application domain of adaptive noise cancellation. In fetal heart rate measurement, muscle movement and mother heartbeat are the noise. Noise cancellation system include the following procedure

1. Reference input measurement, (figure(a)) , set Q for chest leads are used to measure the maternal heartbeat.
2. Primary input measurement (figure(a)) A single abdominal lead is used to measure a mixture of the maternal and fetal ECG which serves as primary input. The reference input is adaptively filtered and subtracted from the fetal ECG signal to get a noise free version of the fetal signal.

6. Derive Backpropagation algorithm.

Algorithm :

1.Select a pattern Xk from the training set T present it to the network.

2.Forward Pass: Compute activations and signals of input, hidden and output neurons in that sequence.

3.Error Computation: Compute the error over the output neurons by comparing the generated outputs with the desired outputs.

4.Compute Weight Changes: Use the error to compute the change in the hidden to output layer weights, and the change in input to hidden layer weights such that a global error measure gets reduced.

5.Update all weights of the network

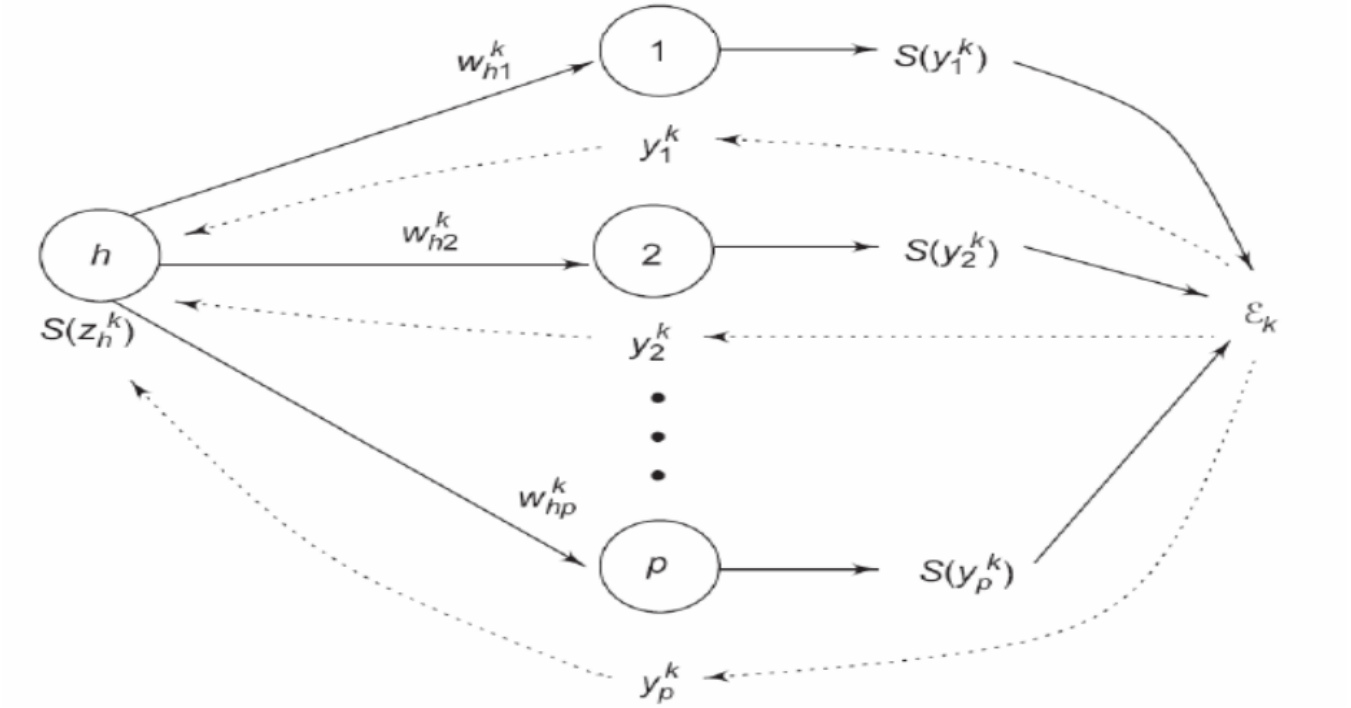
Hidden to output layer weights

$$w_{hj}^{k+1} = w_{hj}^{k} + \Delta w_{hj}^{k}$$

Input to hidden layer weights

$$w_{ih}^{k+1} = w_{ih}^{k} + \Delta w_{ih}^{k}$$

6. Repeat Steps 1 through 5 until the global error falls below a predefined threshold.



1.Computation of Neuronal signal

a. Input Layer

$$\mathcal{S}(x_i^k) = x_i^k, \ i = 1, \ldots, n$$

$$\mathcal{S}(x_0^k) = x_0^k = 1$$

b. Hidden Layer

$$z_h^k = \sum_{i=0}^{n} w_{ih}^k \mathcal{S}(x_i^k) = \sum_{i=0}^{n} w_{ih}^k x_i^k, \ h = 1, \ldots, q$$

$$\mathcal{S}(z_h^k) = \frac{1}{1 + e^{-z_h^k}}, \ h = 1, \ldots, q$$

$$\mathcal{S}(z_0^k) = 1, \ \forall k$$

c. Output Layer

$$y_j^k = \sum_{h=0}^{q} w_{hj}^k \mathcal{S}(z_h^k), \ j = 1, \ldots, p$$

$$\mathcal{S}(y_j^k) = \frac{1}{1 + e^{-y_j^k}}, \ j = 1, \ldots, p$$

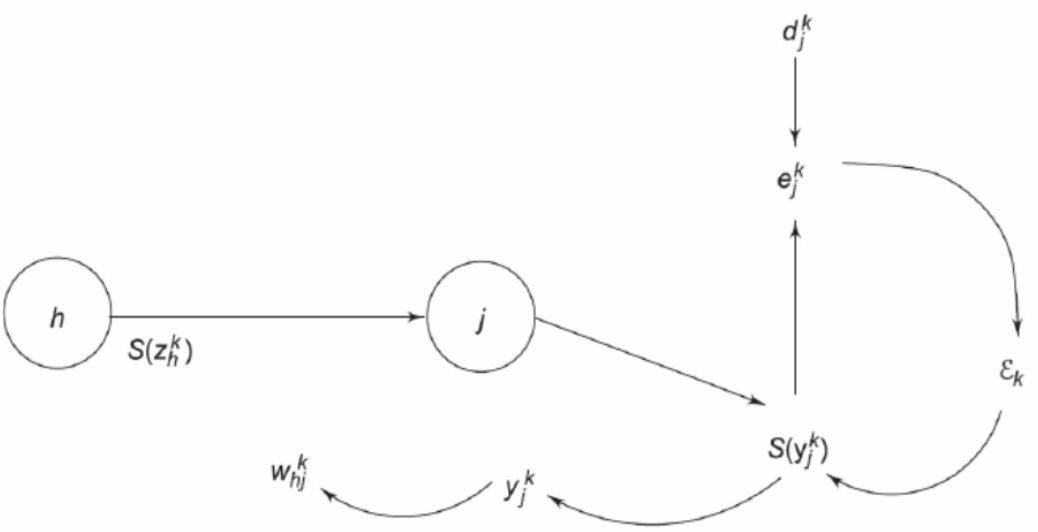
Here we use linear activation for input layer neurons.

Hidden and output layer neurons are sigmoidal functions.

A training dataset assumed is $T = \{X_k , D_k \}\ ^{Q}_{k=1}$

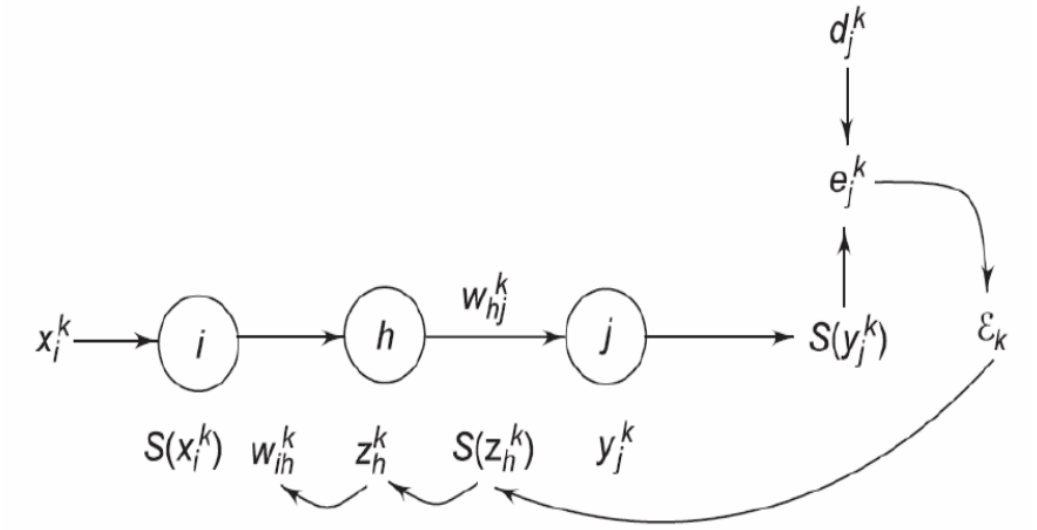
2.Computation of Error Gradients

a. Gradient computation for Hidden to output



$$\frac{\partial \mathcal{E}_k}{\partial w_{hj}^k} = \frac{\partial \mathcal{E}_k}{\partial \mathcal{S}(y_j^k)} \frac{\partial \mathcal{S}(y_j^k)}{\partial y_j^k} \frac{\partial y_j^k}{\partial w_{hj}^k}$$

b. Input to Hidden layer weight gradient layer weights



$$\frac{\partial \mathcal{E}_k}{\partial w_{ih}^k} = \frac{\partial \mathcal{E}_k}{\partial \mathcal{S}(z_h^k)} \frac{\partial \mathcal{S}(z_h^k)}{\partial z_h^k} \frac{\partial z_h^k}{\partial w_{ih}^k}$$

3.Weight Update

1. *For hidden to output layer weights:*

$$\begin{aligned} w_{hj}^{k+1} &= w_{hj}^{k} + \Delta w_{hj}^{k} \\ &= w_{hj}^{k} + \eta\left(-\frac{\partial \mathcal{E}_k}{\partial w_{hj}^{k}}\right) \\ &= w_{hj}^{k} + \eta\delta_j^k \mathcal{S}(z_h^k) \end{aligned}$$

2. *For input to hidden layer weights:*

$$\begin{aligned} w_{ih}^{k+1} &= w_{ih}^{k} + \Delta w_{ih}^{k} \\ &= w_{ih}^{k} + \eta\left(-\frac{\partial \mathcal{E}_k}{\partial w_{ih}^{k}}\right) \\ &= w_{ih}^{k} + \eta\delta_h^k x_i^k \end{aligned}$$

(refer notes for detail)

6. Write a note on the factors which improves the training and operational performance of BP algorithm. (Factors to be considered-Pattern mode or Batch mode, Bipolar signal function, Initialization of network weights , data pre-processing, adjustment of learning rate, Error criteria for termination of learning, regularization, selection of network architecture)

Pattern and Batch mode training :

Pattern Mode:

Present a single pattern , Compute local gradients , Change the network weights ,

Given Q training patterns {Xi,Di } i=1 Q , and some initial neural network N 0, pattern mode training generates a sequence of Q neural networks N 1, . . . , N Q over one epoch of training by incremental computation of error to tune the weights.

Batch Mode (true gradient descent) :

Collect the error gradients over an entire epoch , Change the weights of the initial neural network N 0 in one shot. Takes a whole batch of training data and gives an estimation of NN.

Bipolar Signal function :

Introducing a bipolar signal function such as the hyperbolic tangent function can cause a significant speed up in the network convergence. (instead of sigmoid)

Weight Inititialization :

- Choose small random values within some interval $[-\varepsilon, +\varepsilon]$. (Identical initial values can lead to network paralysis—the network learns nothing.)
- May be incorrectly interpreted as a local minimum.
- Signal values are close to the 0 or 1; signal derivatives are infinitesimally small.
- Weight changes are negligibly small.
- Small weight changes allow the neuron to escape from incorrect saturation only after a very long time.
- Randomization of network weights helps avoid these problems.
- For bipolar signal functions it is useful to randomize weights depending on the individual neuron.

Adjusting learning rates :

- For small learning rates, convergence to the local minimum in question is guaranteed but may lead to long training times.
- If network learning is non-uniform, and we stop before the network is trained to an error minimum, some weights will have reached their final "optimal" values; others may not have.
- In such a situation, the network might perform well on some patterns and very poorly on others.

If we assume that the error function can be approximated by a quadratic then we can make the following observations.

- An optimal learning rate reaches the error minimum in a single learning step.
- Rates that are lower take longer to converge to the same solution.
- Rates that are larger but less than twice the optimal learning rate converge to the error minimum but only after much oscillation.
- Learning rates that are larger than twice the optimal value will diverge from the solution.

Error criteria for termination of learning :

Compare the value of squared error averaged over one epoch. If it's typically 0.01 or as low as 0.0001 then stop.

Check the generalization ability of the network. The network generalizes well if it is able to predict correct or near correct outputs for unseen inputs.

Selection of network architecture :

A three-layered network can approximate any continuous function.

Problem with multilayered networks using one hidden layer:

- Neurons tend to interact with each other globally
- Interactions make it difficult to generate approximations of arbitrary accuracy.

With two hidden layers the curve-fitting process is easier:

- The first hidden layer extracts local features of the function (as binary threshold neurons partition the input space into regions.)
- Global features are extracted in the second hidden layer.