Supervised Classification: Mathematical Formulation of Perceptron Learning Algorithm

Dr. Samit Ari

samit@nitrkl.ac.in

Professor

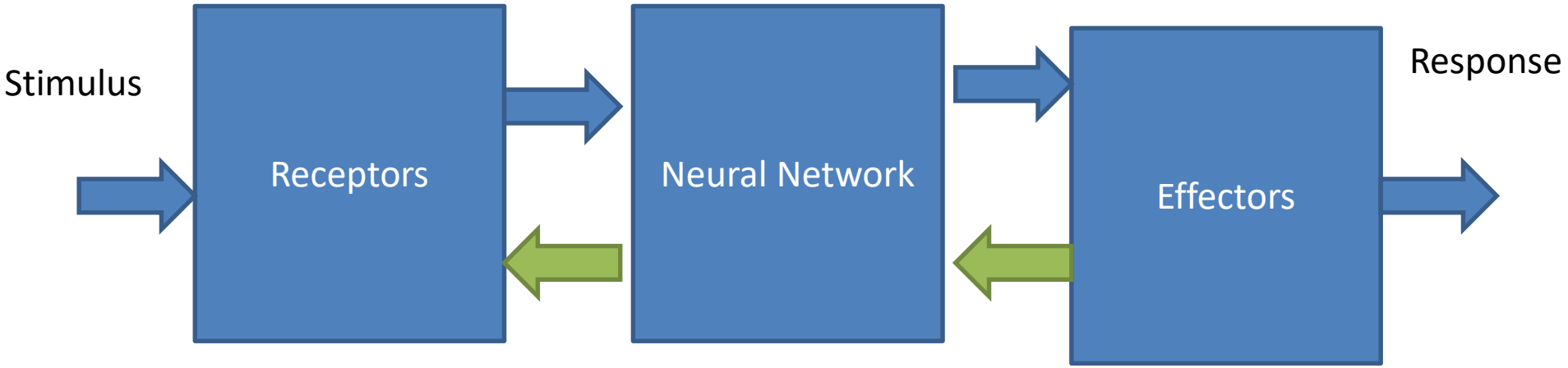
Department of Electronics & Communication Engineering

National Institute of Technology

Rourkela 769008

INDIA

Block diagram representation of nervous system



Introduction

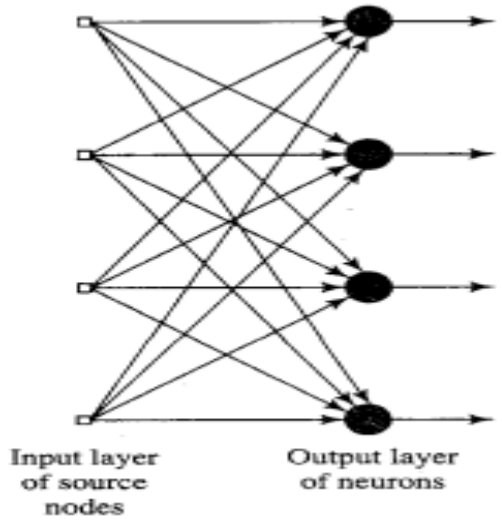
A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use.

It resembles the brain in two respects:

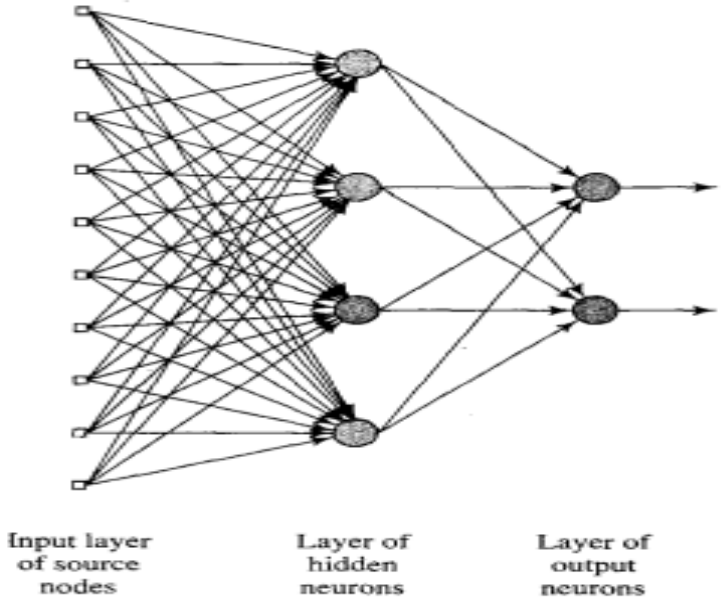
❖Knowledge is acquired by the network from its environment through a learning process.

❖Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

Architecture

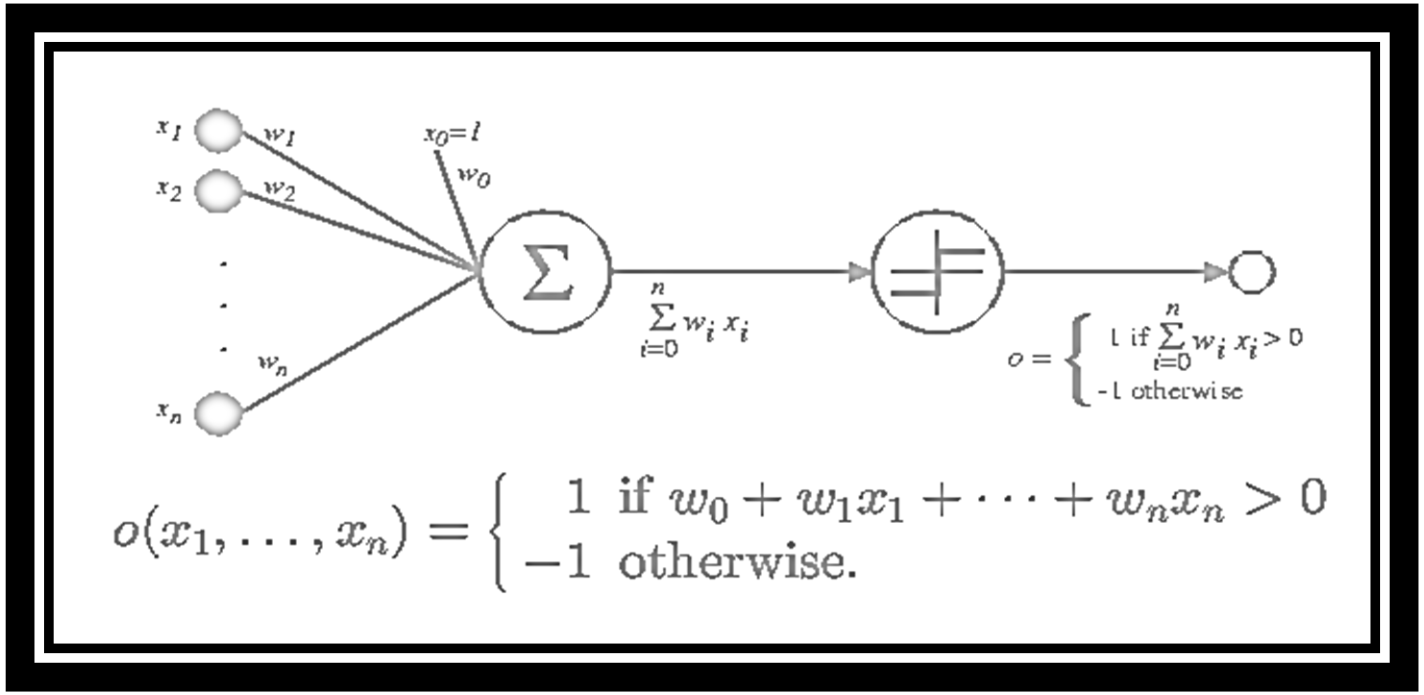


Feed forward network with a single layer of neurons



Feed-forward network with a hidden layer and one output layer

Architecture of a Neuron



- Input: a vector of real values
- Output: 1 or -1 (binary)
- Activation function: threshold function

Applications

Prediction: learning from past experience

pick the best stocks in the market

predict weather

identify people with cancer risk

Classification

Image processing

Predict bankruptcy for credit card companies

Risk assessment

Applications

Recognition

Pattern recognition: SNOOPE (bomb detector in U.S. airports)

Character recognition

Handwriting: processing checks

Data association

Not only identify the characters that were scanned but identify when the scanner is not working properly

Applications

Data Conceptualization

infer grouping relationships

e.g. extract from a database the names of those most likely to buy a particular product.

Data Filtering

e.g. take the noise out of a telephone signal, signal smoothing

Planning

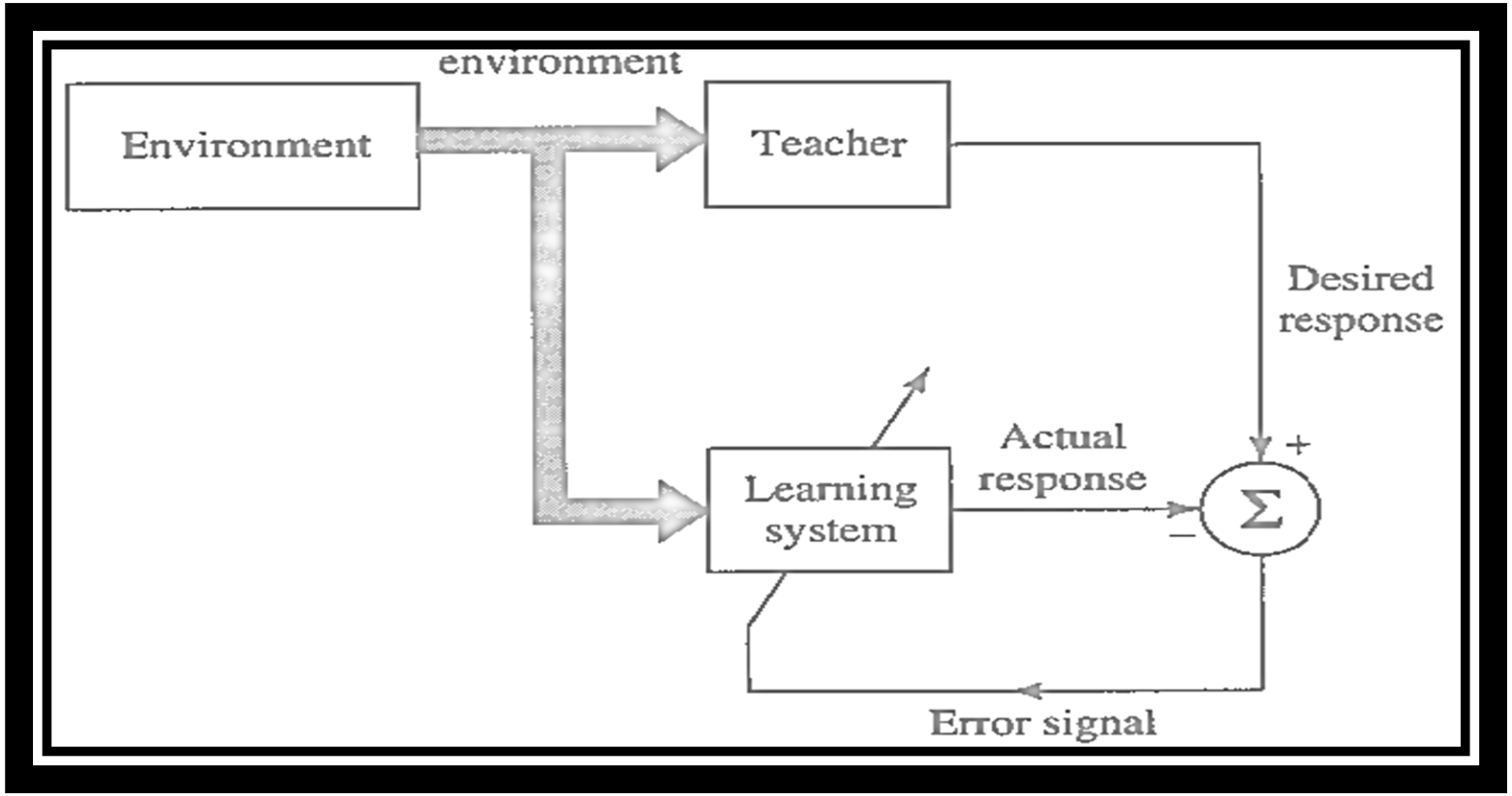
Unknown environments

Sensor data is noisy

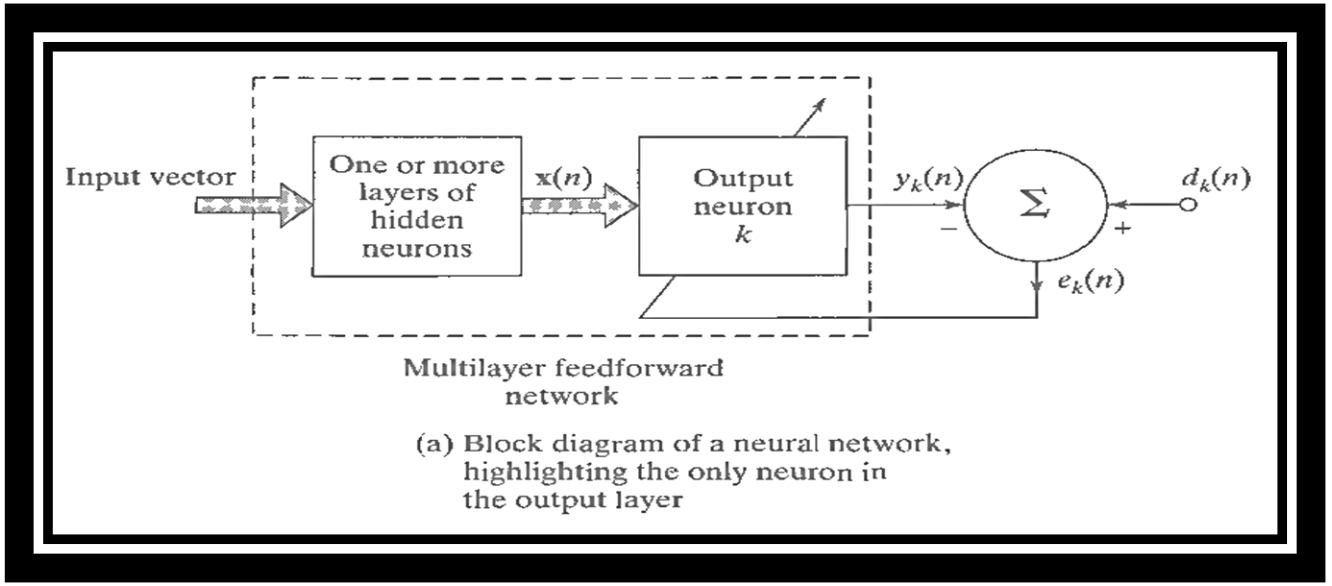
Fairly new approach to planning

Learning Paradigms

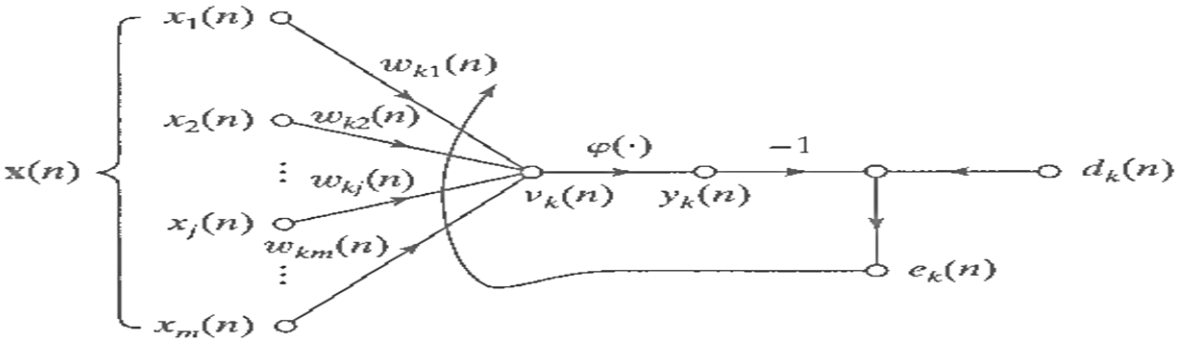
Learning with a Teacher (Supervised learning)



Error-Correction Learning



(a) Block diagram of a neural network, highlighting the only neuron in the output layer



(b) Signal-flow graph of output neuron

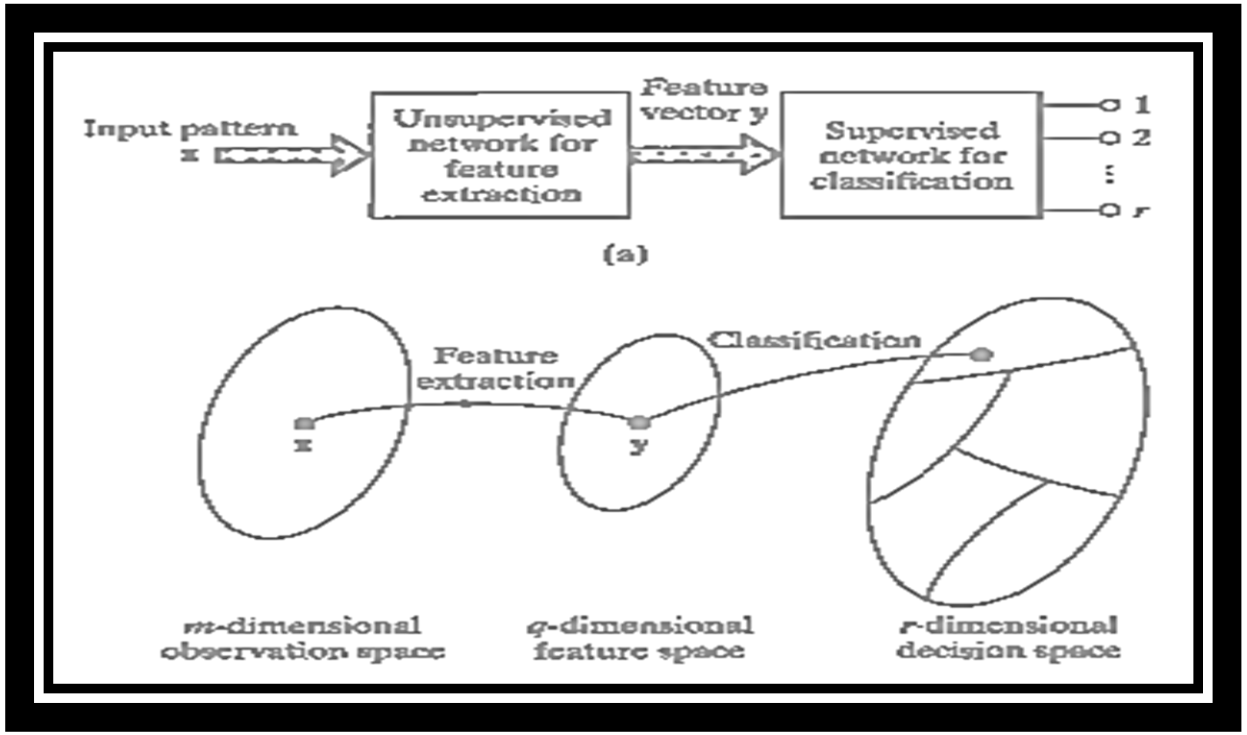
$$e_k(n) = d_k(n) - y_k(n)$$

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n)$$

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

The Issues of Learning Tasks (contd)

Pattern Recognition: The process whereby a received pattern/signal is assigned to one of a prescribed number of classes



Single layer and Multi layer Perceptron

Perceptron

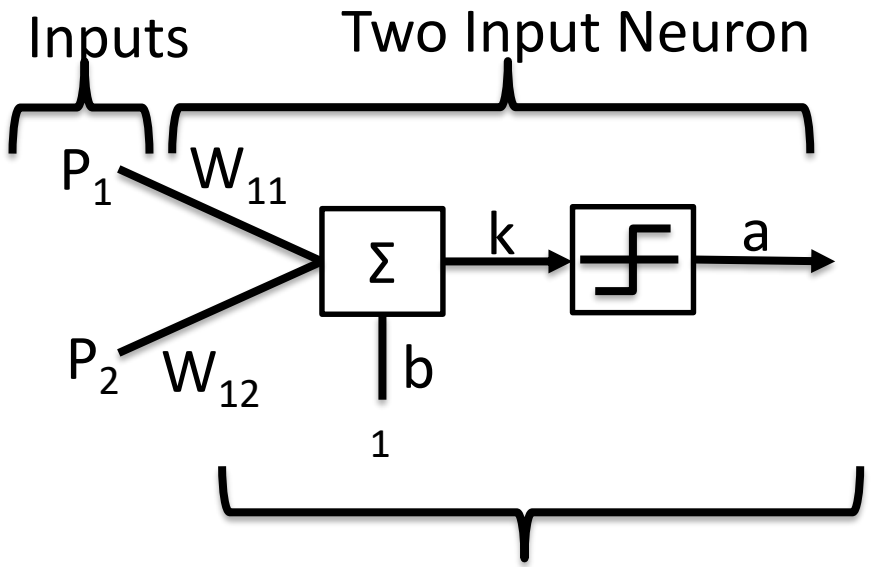
It is the simplest form of a neural network used for the classification of patterns said to be linearly separable.

✓A single artificial neuron that computes its weighted input and uses a threshold activation function.

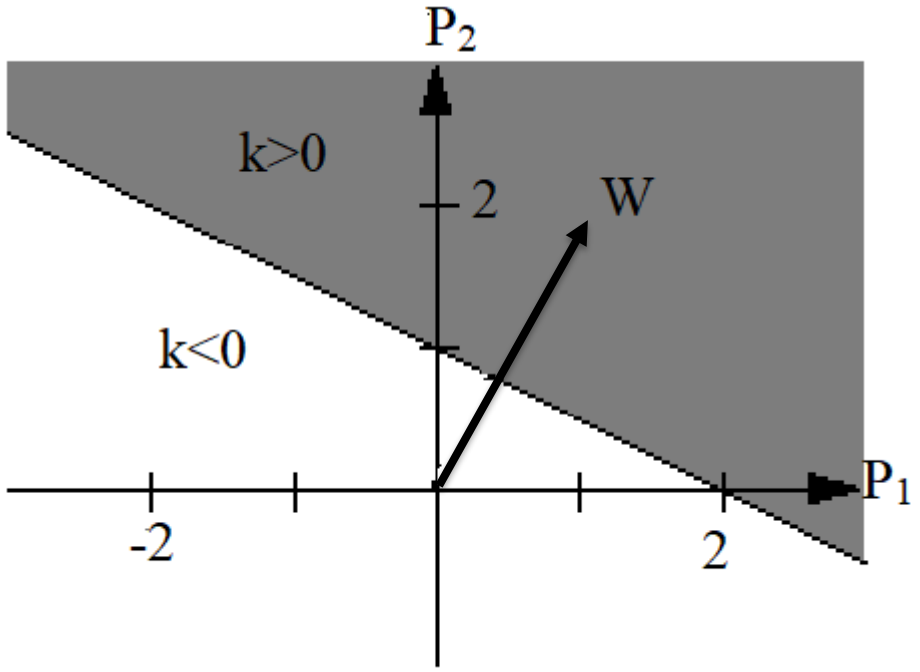
✓It effectively separates the input space into two categories by the hyper plane:

$$\mathbf{w}^T\mathbf{x} + b_i = 0$$

Two-Input Case



a = hardlim(W_p+b)



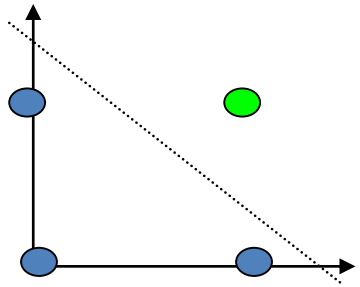
Decision Boundary $\mathbf{W}\mathbf{p} + b = 0$

Perceptron Limitations

- A single layer perceptron can only learn linearly separable problems.

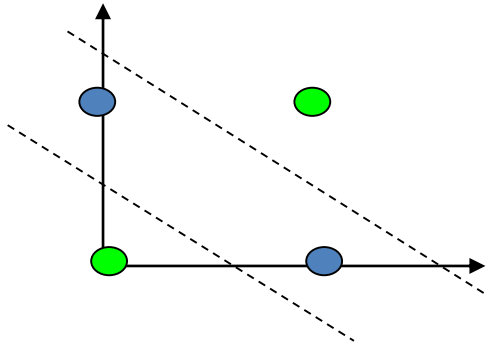
Boolean AND function is linearly separable,

whereas Boolean XOR function is not.



Boolean AND

✓



Boolean XOR

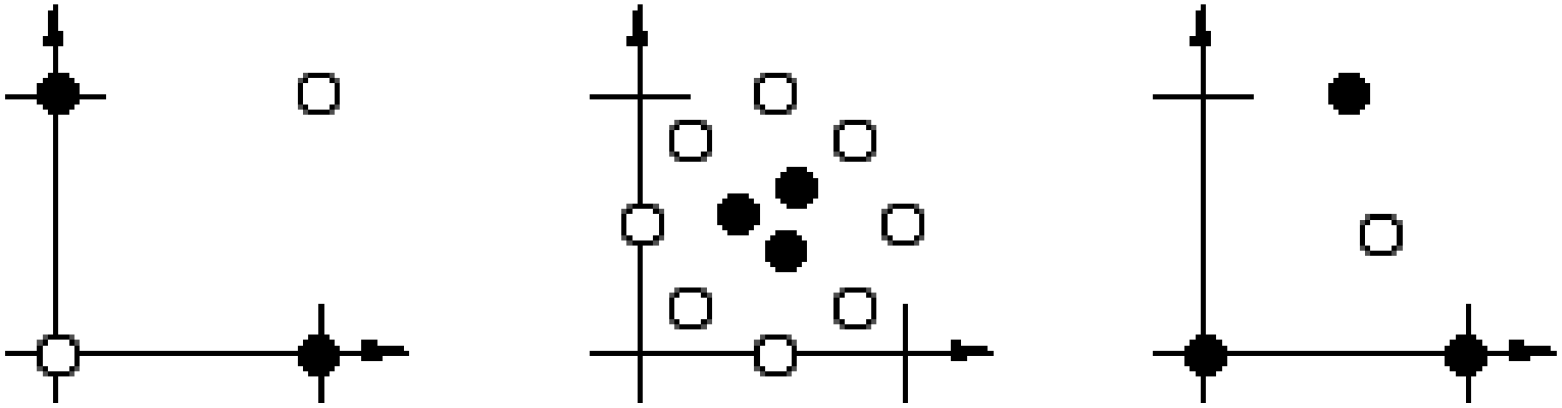
×

Perceptron Limitations (contd)

Linear Decision Boundary

$$_1 w^T p + b = 0$$

Linearly Inseparable Problems



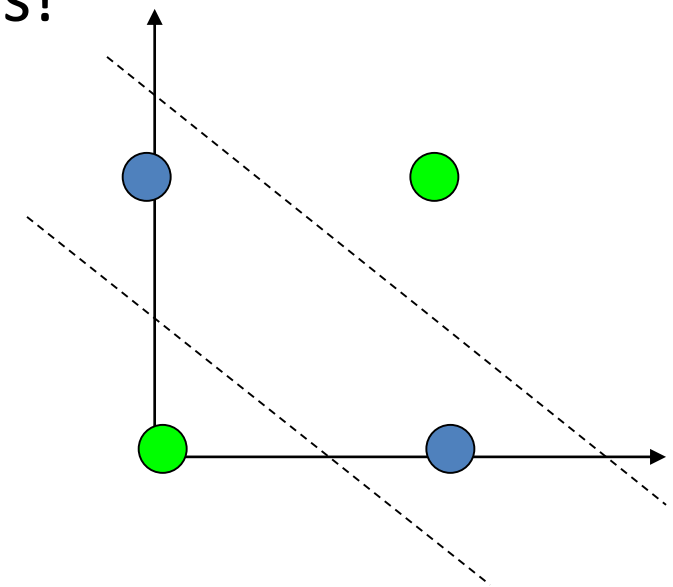
Perceptron Limitations (contd)

For a linearly not-separable problem:

Would it help if we use more layers of neurons?

What could be the learning rule for each neuron?

Yes!



Solution: Multilayer networks and the backpropagation learning algorithm

Perceptron Convergence Theorem

Theorem:

If the *n*th member of the training set, **x**(*n*), is correctly classified by the weight vector **w**(*n*) computed at the *n*th iteration of the algorithms, no correction is made to the weight vector of the perception in accordance with the rule:

$$w(n+1) = w(n) \qquad if\ w^T x(n) > 0\ and\ x(n)\ and\ x(n)\ belongs\ to\ class\ \zeta_1$$

$$w(n+1) = w(n) \qquad if\ w^T x(n) \leq 0\ and\ x(n)\ and\ x(n)\ belongs\ to\ class\ \zeta_2$$

Perceptron Convergence Theorem

2. Otherwise, the weight vector of the perception is updated in accordance with the rule

$$w(n+1) = w(n) - \eta(n)x(n) \qquad if\ w^T(n)x(n) > 0\ and\ x(n)\ belongs\ to\ class\ \zeta_2$$

$$w(n+1) = w(n) + \eta(n)x(n) \qquad if\ w^T(n)x(n) \leq 0\ and\ x(n)\ belongs\ to\ class\ \zeta_1$$

Where the *learning-rate parameter* $\eta(n)$ *control* the adjustment applied to the weight vector at iteration *n*

If $\eta(n) = \eta > 0$ where η is a constant independent of the iteration number *n*, we have a *fixed increment adaptation rule* for the perceptron.

Adaptive Filtering

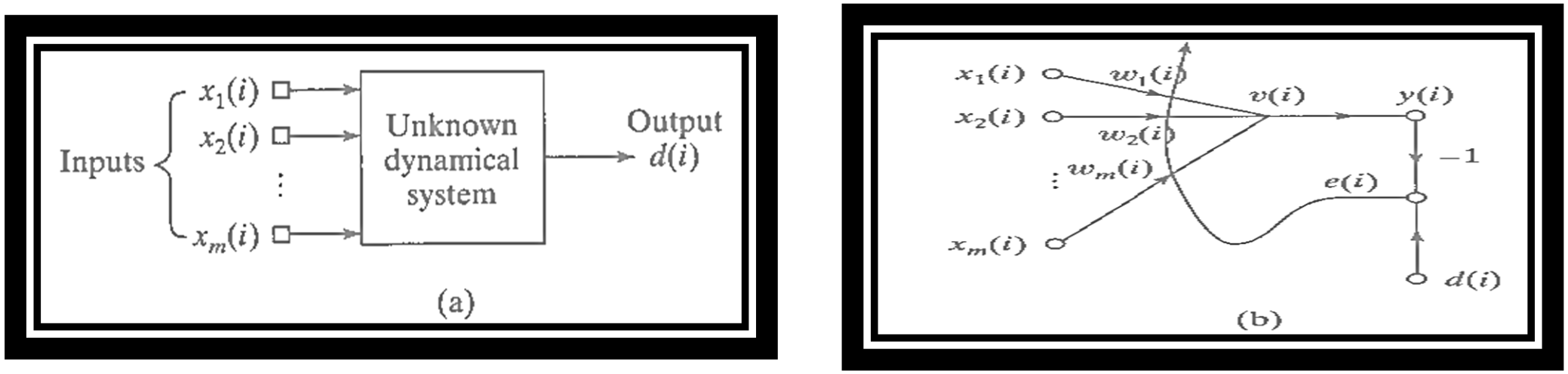


FIGURE (a) Unknown dynamical system (b) Signal-flow graph of adaptive model for the system

Adaptive Filtering

The external behavior of the system is described by:-

$$\mathcal{T}: \{\mathbf{x}(i), d(i); i = 1, 2, \ldots, n, \ldots\}$$

where

$$\mathbf{x}(i) = [x_1(i), x_2(i), \ldots, x_m(i)]^T$$

The stimulus $\mathbf{x}(i)$ can arise in one of two fundamentally different ways, one spatial and the other temporal:

- The m elements of $\mathbf{x}(i)$ originate at different points in space; in this case we speak of $\mathbf{x}(i)$ as a *snapshot* of data.
- The m elements of $\mathbf{x}(i)$ represent the set of present and $(m - 1)$ past values of some excitation that are *uniformly spaced in time*.

Adaptive Filtering

Figure b shows a signal-flow graph of the adaptive filter. Its operation consist of Two continuous processes :

1. *Filtering process*, which involves the computation of two signals:
   - An output, denoted by $y(i)$, that is produced in response to the m elements of the stimulus vector $\mathbf{x}(i)$, namely, $x_1(i), x_2(i), \ldots, x_m(i)$.
   - An error signal, denoted by $e(i)$, that is obtained by comparing the output $y(i)$ to the corresponding output $d(i)$ produced by the unknown system. In effect, $d(i)$ acts as a *desired response* or *target signal*.
2. *Adaptive process*, which involves the automatic adjustment of the synaptic weights of the neuron in accordance with the error signal $e(i)$.

Adaptive Filtering

$$y(i) = v(i) = \sum_{k=1}^{m} w_k(i) x_k(i)$$

$$y(i) = X^T(i) W(i)$$

$$where$$

$$W(i) = [w_1(i), w_2(i), w_2(i), \ldots . w_m(i)]^T$$

Typically, y(i) is different from d(i); hence, their comparison result in the error signal:

$$e(i) = d(i) - y(i)$$

Unconstrained Optimization Technique

Consider a cost function $\xi(w)$ that is continuously differentiable function of some unknown weight (parameter) vector **w** . The $\xi(w)$ maps the elements of **w** into real number

$$\xi(w^*) \leq \xi(w)$$

Minimum the cost function $\xi(w)$ with respect to the weight vector **w** .
The necessary condition for optimality is

$$\nabla \xi(w^*) = 0$$

Where ∇ is the gradient operator

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots \frac{\partial}{\partial w_n} \right]^T$$

Unconstrained Optimization Technique

And $\nabla\xi(w)$ is the gradient operator vector of the cost function

$$\nabla\xi = \left[\frac{\partial\xi}{\partial w_1}, \frac{\partial\xi}{\partial w_2}, \ldots \frac{\partial\xi}{\partial w_n}\right]^T$$

Starting with an initial guess denoted by **w**(0) ,generated a sequence of weight vector **w**(1) ,**w**(2),….such that the cost function $\xi(w)$ is reduced at each iteration of the algorithms as shown by

$$\xi(w(n+1)) < \xi(w(n))$$

Where **w**(n) is the old value of the weight vector and **w**(n+1) is its updated value

Method of steepest descent

The successive adjustments applied to the weight vector **w** are in the direction of steepest descent , that is, in a direction opposite to the gradient vector $\nabla \xi(w)$

$$g = \nabla \xi(w)$$

The steepest descent algorithms is formally described by

$$w(n+1) = w(n) - \eta g(n)$$

Where *n* is a positive constant called the step size or learning -rate-parameter , **g**(n) is the gradient vector evaluated at the point **w**(n)

$$\Delta w(n) = w(n+1) - w(n)$$
$$= -\eta g(n)$$

Method of steepest descent

We use first order taylor series expansion around **w**(n) to approximate $\xi(n+1)$

$$\xi(w(n+1)) = \xi(w(n)) + g^T(n)\Delta w(n)$$

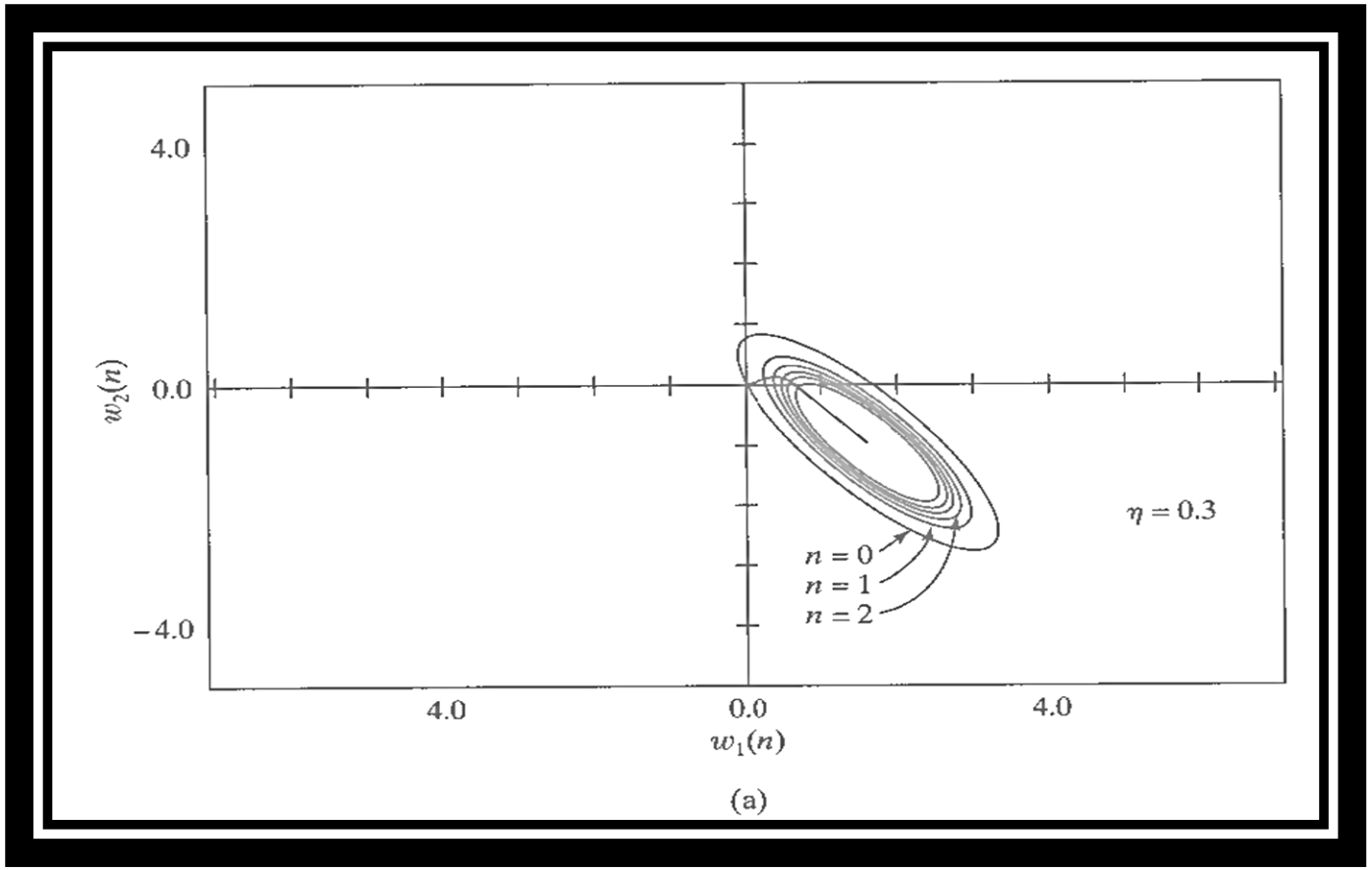
$$\xi(w(n+1)) = \xi(w(n)) - \eta g^T(n) g(n)$$

$$= \xi(w(n)) - \eta \| g(n) \|^2$$

The method of steepest descent converge to the optimal solution **w** slowly . Moreover , the learning –rate-parameter η has a profound influence on its convergence behavior

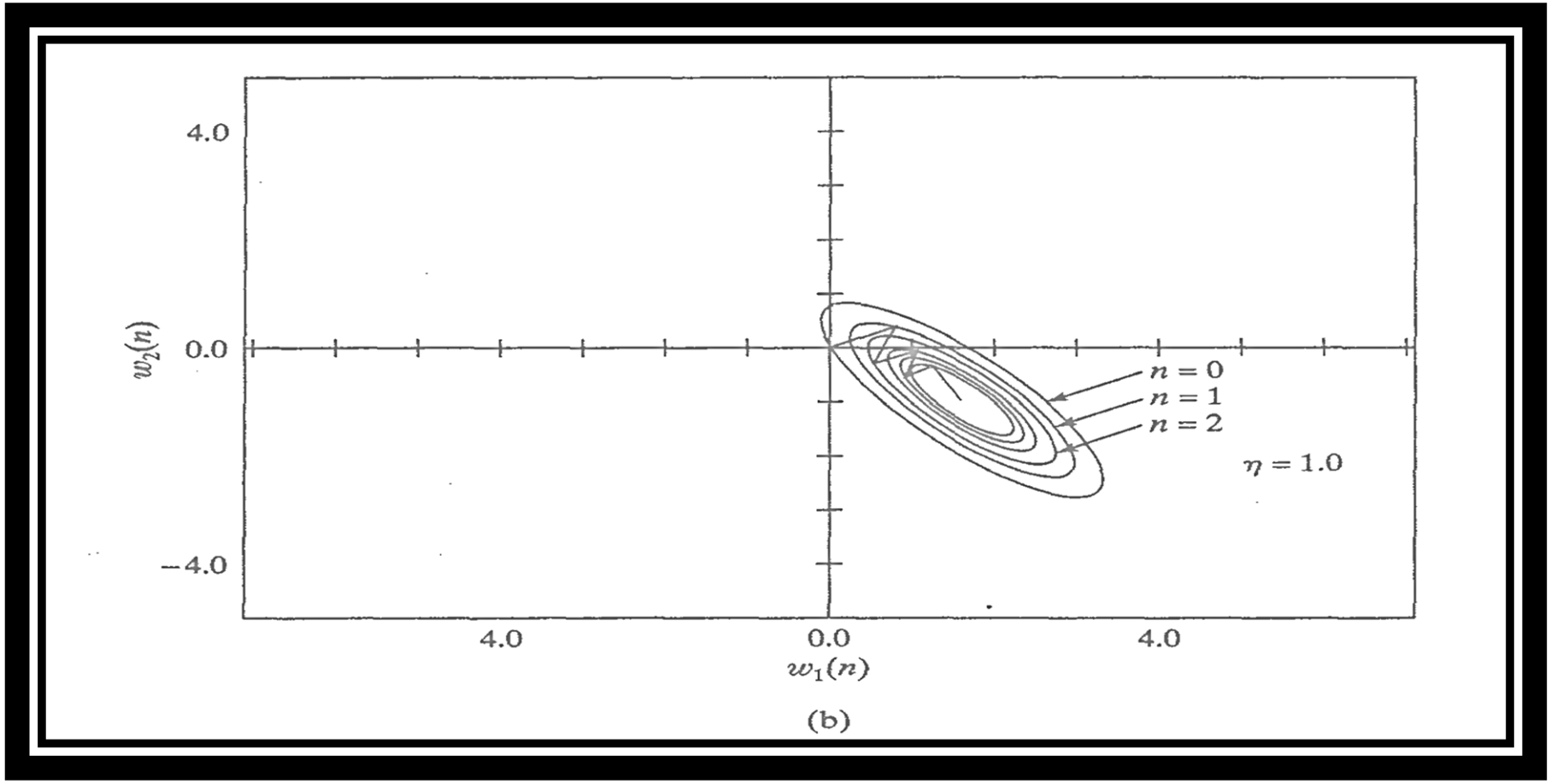
- When η is small , the transient response of the algorithms is overdamped
- When η is large , the transient response of the algorithms is underdamped,
- When η exceeds a certain critical value ,the algorithms become unstable,

Method of steepest descent



Trajectory of the method of steepest descent in a two-dimensional space for values of learning rate parameter (a) eta=0.3

Method of steepest descent



Trajectory of the method of steepest descent in a two-dimensional space for values of learning rate parameter (a) eta=1.0

LMS Algorithm

Based on the use of instantaneous values for cost function :

$$E(\mathbf{w}) = \frac{1}{2} e^2(n)$$

Differentiating with respect to

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = e(n) \frac{\partial e(n)}{\partial \mathbf{w}}$$

The error signal in LMS algorithm :

hence, $e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n)$

$$\frac{\partial e(n)}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)$$ so, $$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)e(n)$$

LMS Algorithm

Using $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}(n)}$ as an *estimate* for the gradient vector,

$$\hat{\mathbf{g}}(n) = -\mathbf{x}(n)e(n)$$

Using this for the gradient vector of steepest descent method, LMS algorithm as follows :

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$$

η: learning-rate parameter

The inverse of η is a *measure* of the memory of the LMS algorithm

When η is small, the adaptive process progress slowly, more of the past data are remembered and more accurate filtering action'

LMS Summary

Training Sample : Input signal vector = $\mathbf{x}(n)$

Desired response = $d(n)$

User-selected parameter : η

Initialization. Set $\hat{\mathbf{w}}(0) = \mathbf{0}$

Computation. For n = 1, 2, … , compute

$$e(n) = d(n) - \hat{\mathbf{w}}^T(n)\mathbf{x}(n)$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)e(n)$$

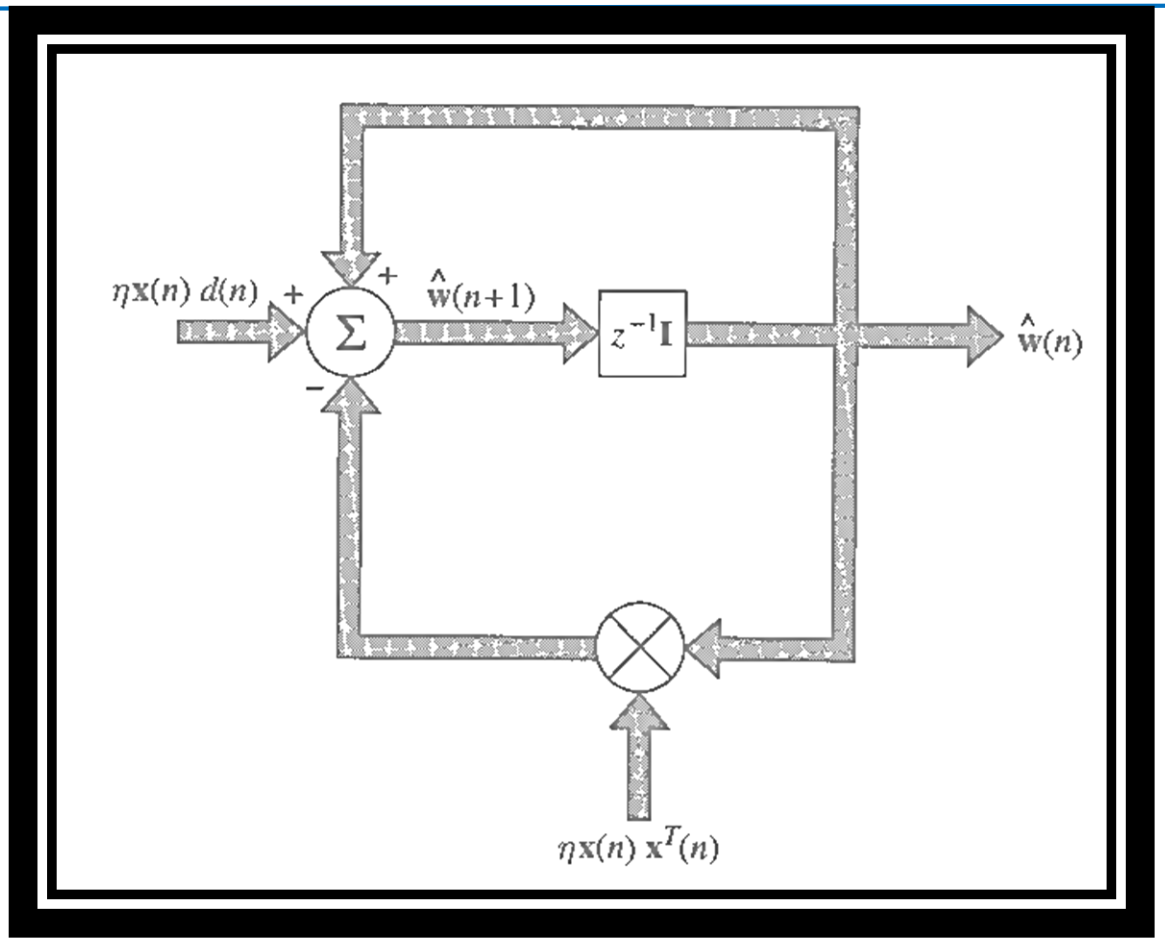
LMS Summary

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)[d(n) - \mathbf{x}^T(n)\hat{\mathbf{w}}(n)]$$
$$= [\mathbf{I} - \eta\mathbf{x}(n)\mathbf{x}^T(n)]\hat{\mathbf{w}}(n) + \eta\mathbf{x}(n)d(n)$$

where $\mathbf{I}$ is the identity matrix. In using the LMS algorithm, we recognize that

$$\hat{\mathbf{w}}(n) = z^{-1}[\hat{\mathbf{w}}(n+1)]$$

LMS Summary



Signal flow graph representation of the LMS algorithm

Convergence Consideration of the LMS algorithm

Two distinct quantities, η and $\mathbf{x}(n)$ determine the transmittance of feedback loop

With environment that supplies $\mathbf{x}(n)$, the selection of η is important for the LMS algorithm to be converge

Convergence of the mean

$$E\left[\hat{\mathbf{w}}(n)\right] \rightarrow \mathbf{w}_0 \quad \text{as} \quad n \rightarrow \infty$$

This is not a practical value

Convergence in the mean square $\quad E\left[e^2(n)\right] \rightarrow \text{constant} \quad \text{as} \quad n \rightarrow \infty$

Convergence condition for LMS algorithm in the mean square

$$0 < \eta < \frac{2}{\text{sum of meaa-square values of the sensor inputs}}$$

Learning Curves

Plot of the mean-square value of the estimation error, versus the number of iterations

Misadjustment

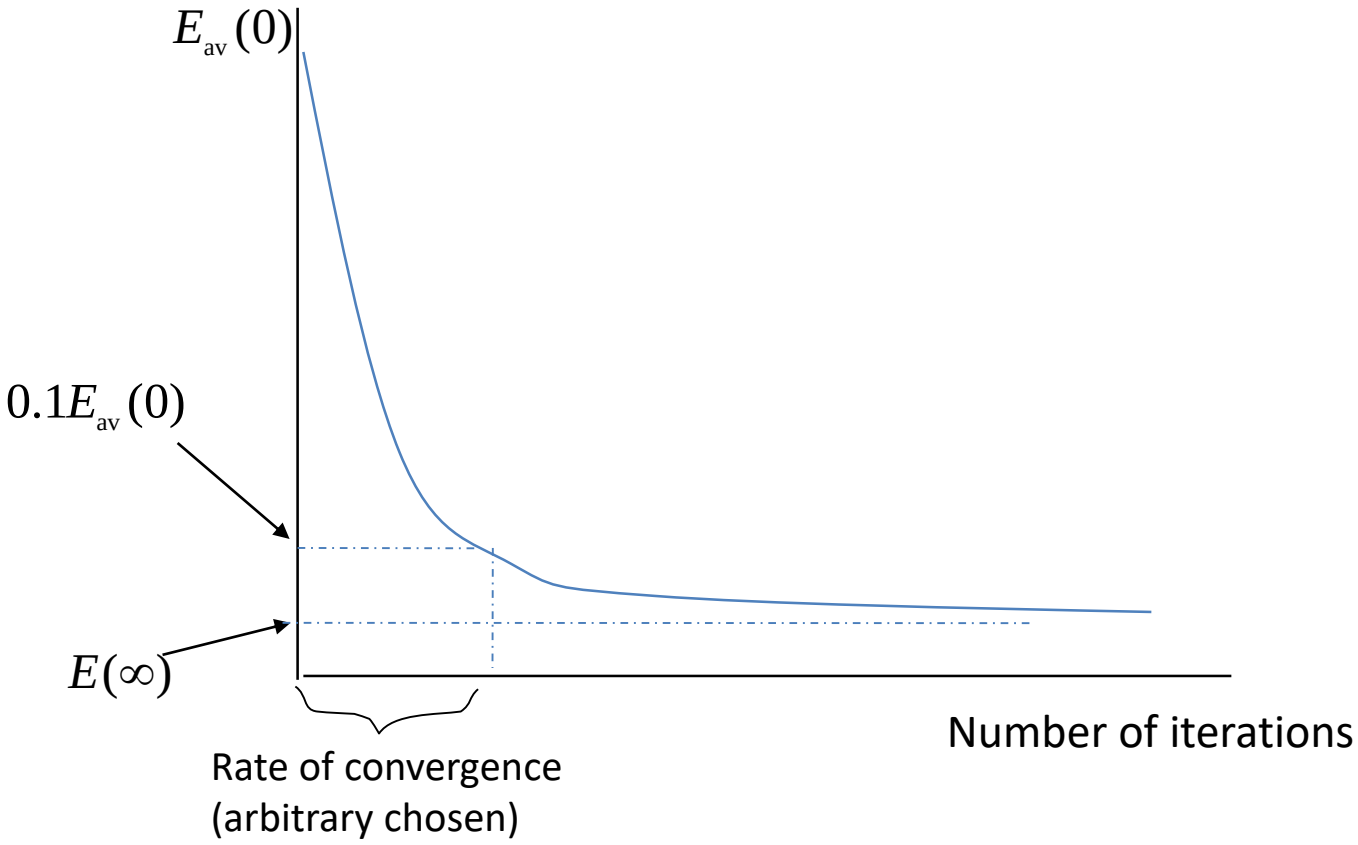
$$\mathrm{M} = \frac{E(\infty) - E_{\min}}{E_{\min}} = \frac{E(\infty)}{E_{\min}} - 1$$

The smaller M is compared to unity, the more *accurate* is the adaptive filtering action

For example, misadjustment of 10% means that the filter produces mean-square error that is 10% greater than the minimum mean-square error produced by corresponding Wiener filter

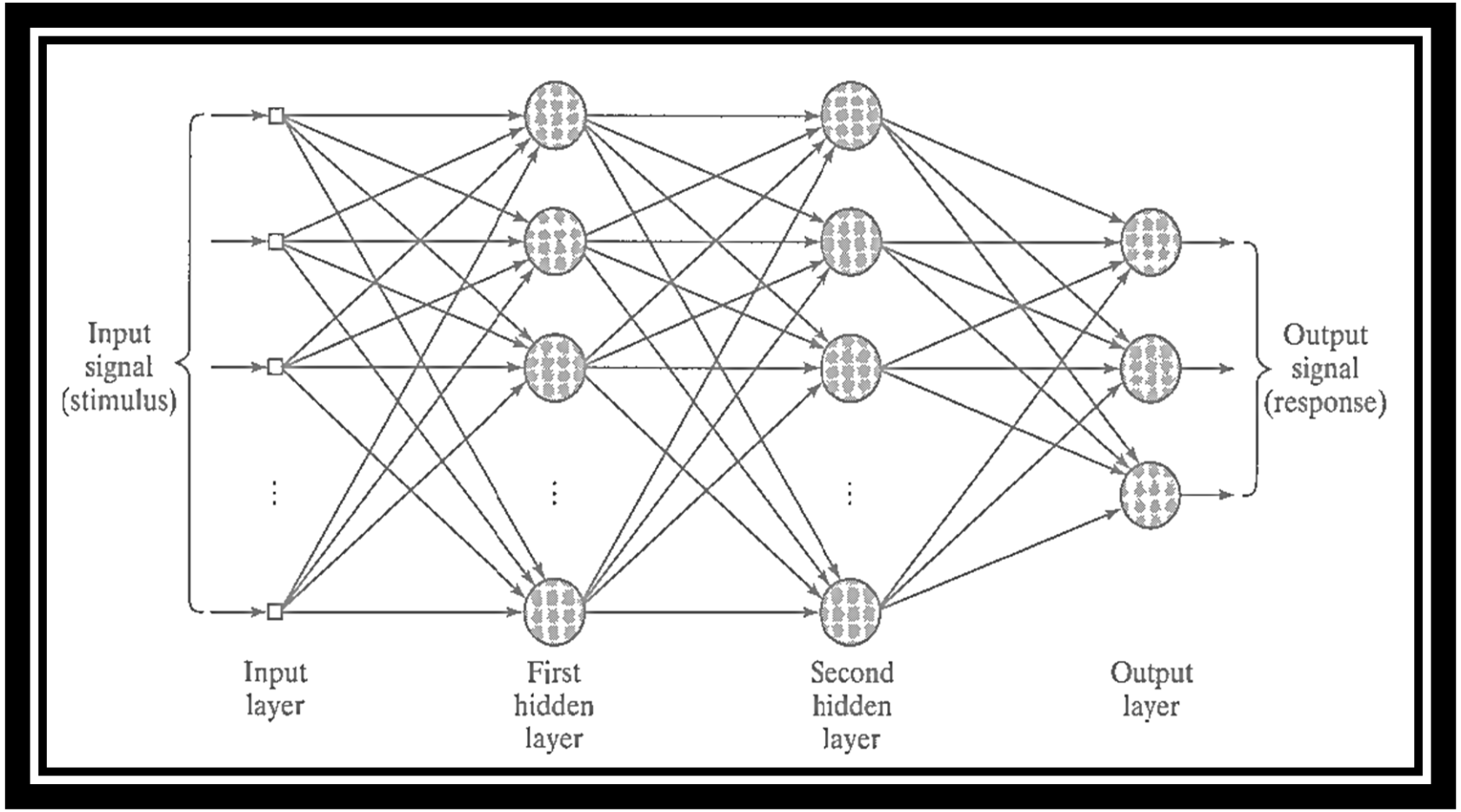
Increasing learning-rate to accelerate learning process, the misadjustment is increased. Conversely, similar result.

Learning Curves



Multilayer Percetrons

Multilayer Percetrons Architecture



Architectural graph of a multilayer perceptron with two hidden layers

Multilayer Percetrons Architecture

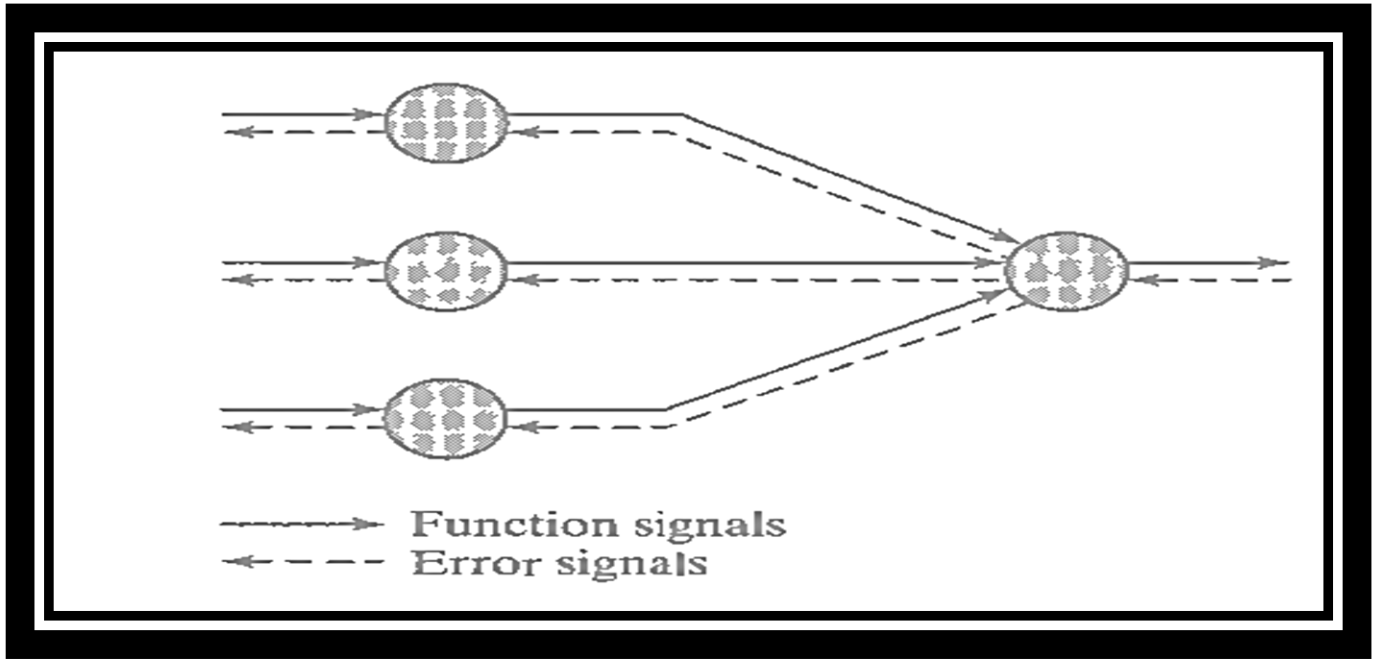


Illustration of the direction of two basic signal flows in a multilayer perceptron :forward propagation of function signals and back-propagation of error signals

Back Propagation Algorithm

The error signal at the output of neuron j at iteration n (i.e, presentation of the n^{th} training example) is defined by

$$e_j(n) = d_j(n) - y_j(n), \qquad neuron\ j\ is\ the\ output\ node$$

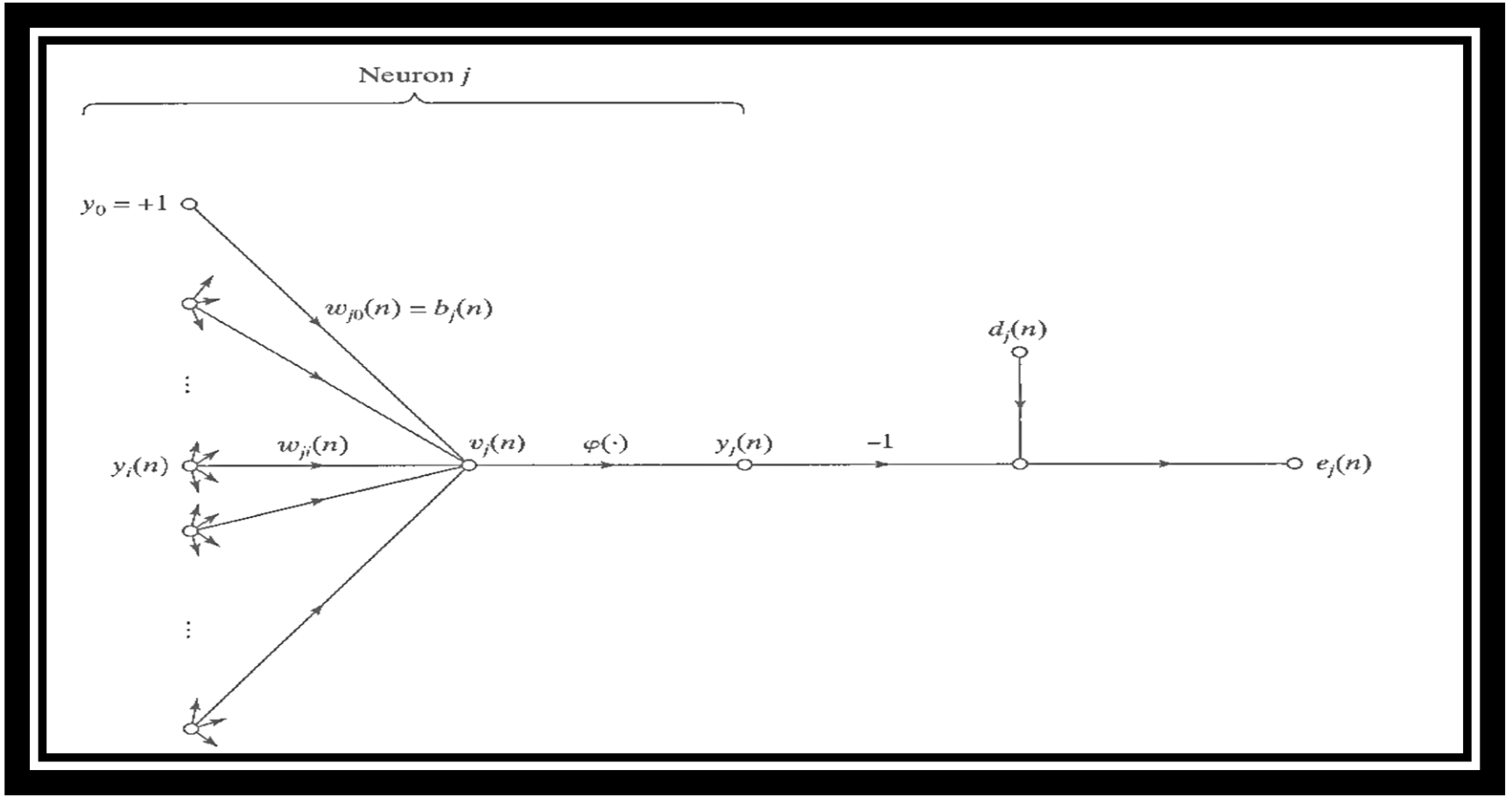
The total Energy

$$\xi(n) = \frac{1}{2}\sum_{j=C} e_j{}^2(n)$$

The average squared error energy

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N}\xi(n)$$

Back Propagation Algorithm



Signal flow graph highlighting the details of output neuron

Back Propagation Algorithm

The induce local field $v_j(n)$

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

$y_j(n)$ appearing at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(v_j(n))$$

According to chain rule

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$\frac{\partial \xi(n)}{\partial e_j(n)} = e_j(n), \qquad \frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

Back Propagation Algorithm

$$\frac{\partial y_j(n)}{\partial e_j(n)} = \varphi_j'(v_j(n))$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_j(n)$$

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n)$$

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)}$$

$$\Delta w_{ji}(n) = -\eta \delta_j(n) y_i(n)$$

Back Propagation Algorithm

where local gradient $\delta_j(n)$ *is defined by*

$$\delta_j(n) = \frac{\partial \xi(n)}{\partial v_j(n)}$$

$$= \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= -e_j(n)\varphi_j'(v_j(n))$$

$$\delta_j(n) = \frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

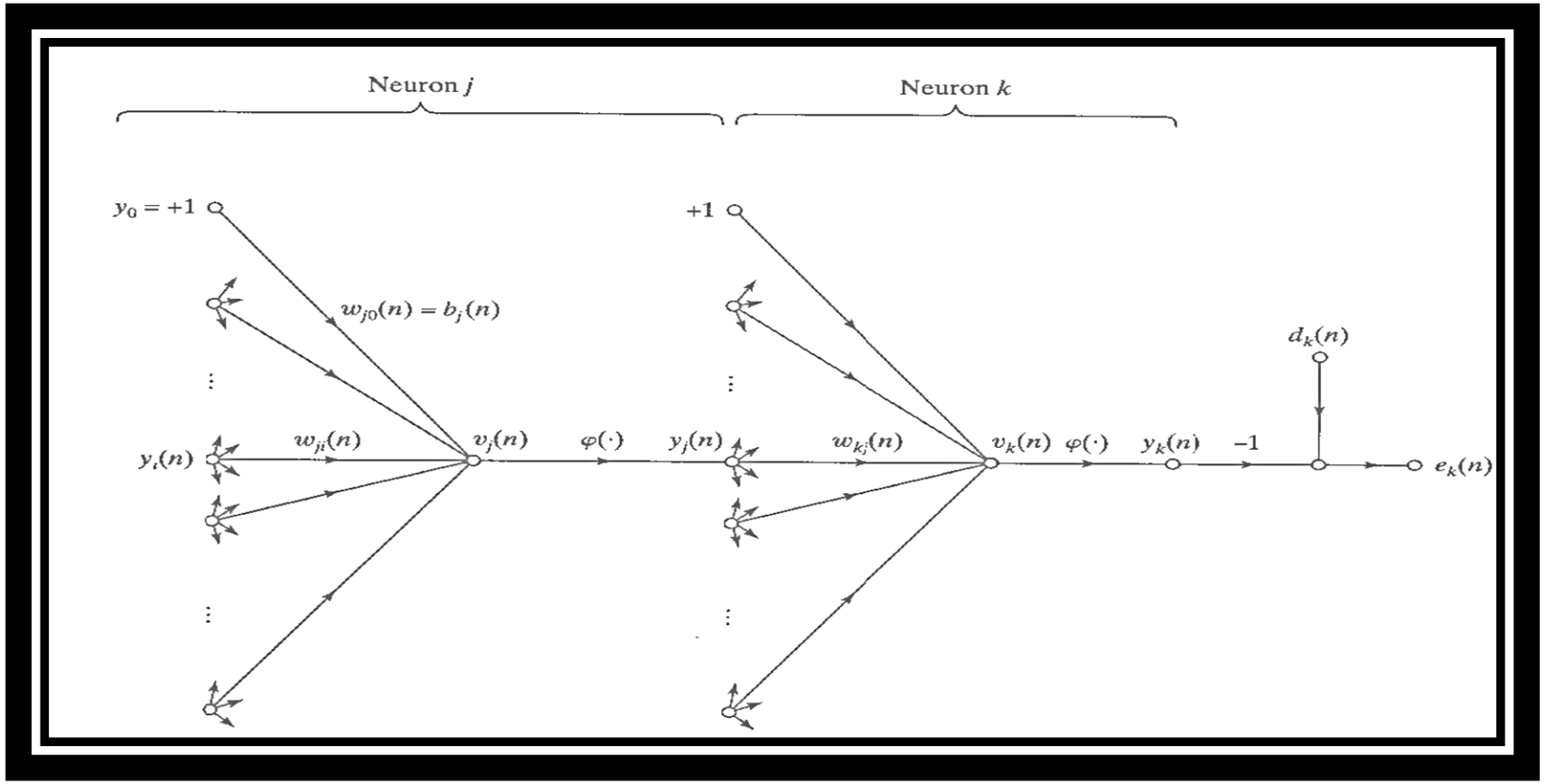
$$= \frac{\partial \xi(n)}{\partial y_j(n)} \varphi_j'(v_j(n))$$

Where j is the hidden neuron

Back Propagation Algorithm

$$\xi(n) = \frac{1}{2}\sum_{k \in C} e_k^{\,2}(n), \qquad \textit{neuron } k \textit{ is an output node}$$

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}$$



Signal-flow graph highlighting the details of output neuron k connected to hidden neuron j

Back Propagation Algorithm

$$\frac{\partial \xi(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

$$e_k(n) = d_k(n) - y_k(n)$$

$$= d_k(n) - \varphi_k(v_k(n)), \qquad \textit{neuron k output node}$$

$$\frac{\partial e_k(n}{\partial v_k(n)} = -\varphi_k'(v_k(n)),$$

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n)$$

Back Propagation Algorithm

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

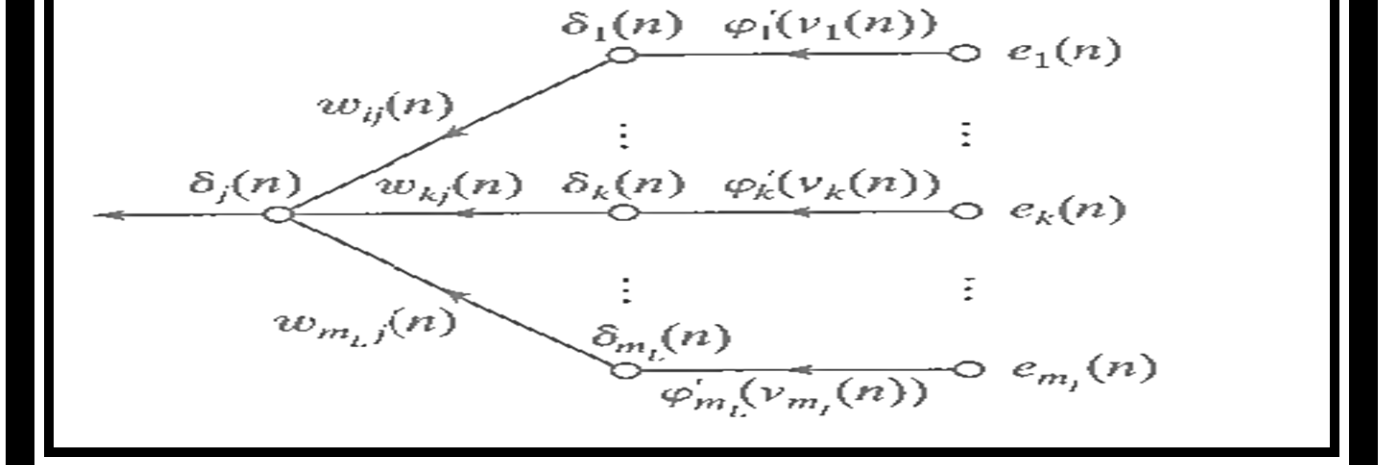
$$\frac{\partial \xi(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi_k'(v_k(n))w_{kj}(n)$$

$$= -\sum_k \delta_k(n)w_{kj}(n)$$

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n), \quad \text{neuron } j \text{ is hidden}$$

Back Propagation Algorithm

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{learning-} \\ \text{rate parameter} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} \text{input signal} \\ \text{of neuron } j \\ y_i(n) \end{pmatrix}$$



Signal-flow graph of a part of the adjoint system pertaining to back-propagation of error-signals

Back Propagation Algorithm

The two passes of Computation

The function signal appearing at the output of neuron j is computed as

$$y_j(n) = \varphi(v_j(n))$$

Where $v_j(n)$ is the induced local field of the neuron j, defined by

$$v_j(n) = \sum_{j=0}^{m} w_{ji}(n) y_i(n)$$

$$y_i(n) = x_i(n)$$

$$y_j(n) = O_j(n)$$

Activation Functions

1. Logistic function: This form of sigmoidal non linearity in its general form is defined by

$$\varphi_j(v_j(n)) = \frac{1}{1+\exp(-av_j(n))} \quad a>0 \ and \ -\infty < v_j(n) < \infty$$

$$\varphi_j'(v_j(n)) = \frac{a\exp(-av_j(n))}{[1+\exp(-av_j(n))]^2}$$

$$\varphi_j'(v_j(n)) = ay_j(n)[1-y_j(n)]$$

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$

$$= a[d_j(n)-o_j(n)]o_j(n)[1-o_j(n)], \quad neuron \ j \ is \ an \ output \ node.$$

$$\delta_j(n) = \varphi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n)$$

$$= ay_j(n)[1-y_j(n)]\sum_k \delta_k(n)w_{kj}(n), \quad neuron \ j \ is \ hidden$$

Activation Functions

1. Hyperbolic tangent function: another commonly used form of sigmoidal non linearity Is the hyperbolic Tangent function, which in its form most general form is defined by:

$$\varphi(v_j(n)) = a\tanh(bv_j(n)), \quad (a,b) > 0$$

$$\varphi'(v_j(n)) = ab\,\mathrm{sech}^2(bv_j(n))$$

$$= ab(1-\tanh^2(bv_j(n))$$

$$= \frac{b}{a}[a-y_j(n)][a+y_j(n)]$$

$$\delta_j(n) = e_j(n)\varphi'(v_j(n))$$

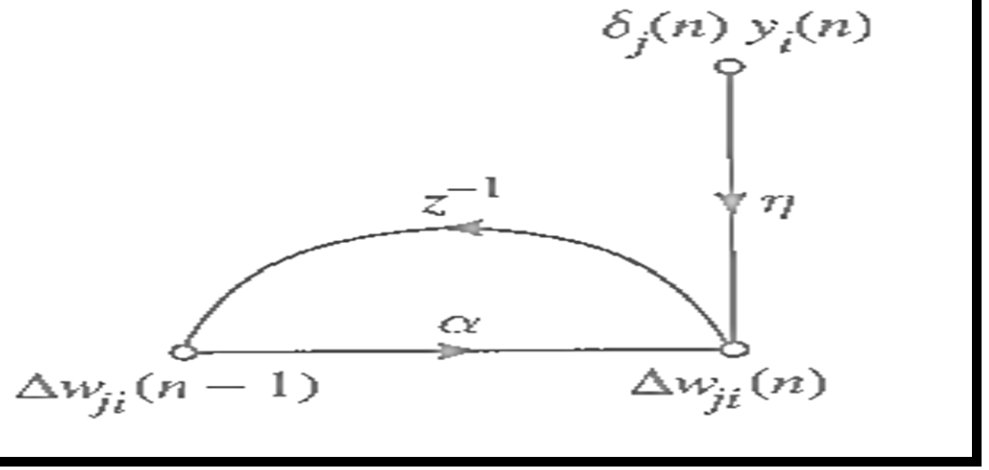
$$= \frac{b}{a}[d_j(n)-o_j(n)][a-o_j(n)][a+o_j(n)]$$

$$\delta_j(n) = \varphi'(v_j(n))\sum_k \delta_k(n)w_{jk}(n)$$

$$= \frac{b}{a}[a-y_j(n)][a+y_j(n)]\sum_k \delta_k(n)w_{jk}(n), \quad \textit{neuron } j \textit{ is hidden}$$

Activation Functions

RATE OF LEARNING



Signal flow graph illustrating the effect of momentum constant

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t)$$

THANK YOU!