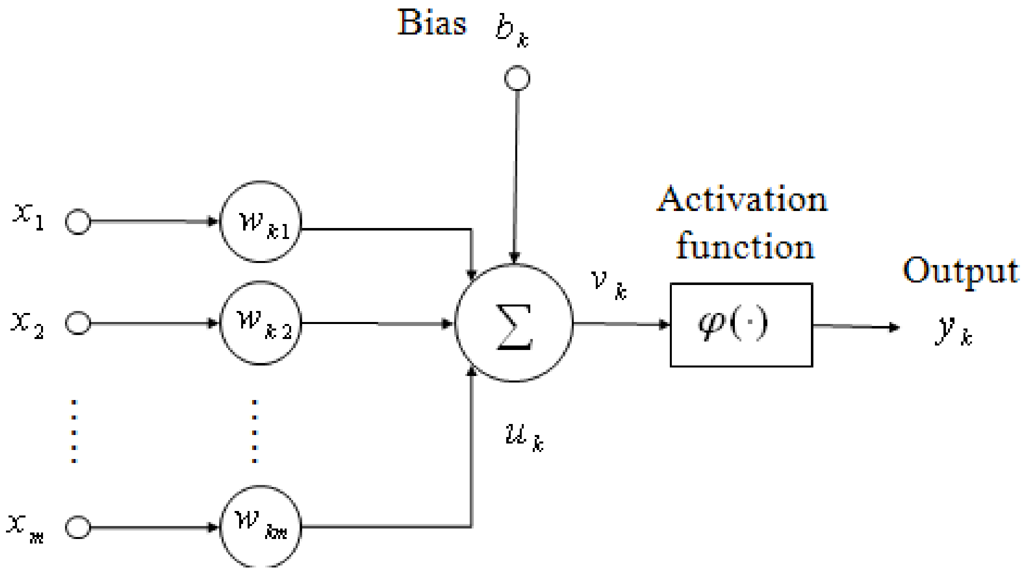
Deep Learning

Basics in Neural Net/ Backpropagation



HYUNSOO LEE

Neural Network

- Basics in Neural Network
  - Related fields
    - Machine Learning
    - Pattern Analysis
  - Motivation
    - Current Mathematical control ➔ "Too load"
    - Another method ➔ "Think like "Human's brain"

- Reference
  - Simon Haykin, "Neural Network : A Comprehensive Foundation", 2nd , Prentice-Hall, 1999

Human Brain (1)

- Neuron
  - Dendrite
  - Soma
  - Axon terminal
  - Contact dendrite

Human Brain (2)

- Human brain : Arbib (1987)
  - Stimulus → Receptors ↔ (Neural Net) ↔ Effectors → Response
  - Pioneer : Santiago Ramon y Cajal, a Spanish Neuroanatomist who introduced neurons as a fundamental unit of brain function
  - Neuron are slow : 10^{-3} Vs 10^{-9}
  - Highly energy efficient : 10^{-16} J Vs 10^{-6} J
  - Huge number of neurons and connections :
    - 10^{10} Neurons & 6 X 10^{13} Connection

Human Brain (3)

- 5 or 6

Human Brain (4)

- Human brain “Computes” in an entirely different way from conventional digital computers

- The brain is highly complex, nonlinear and parallel

- Organization or neurons to perform task much faster than computers
  - Typical time taken in visual recognition tasks is 100~200ms

- Key features of the biological brain
  - **experience** shapes the wiring through **plasticity** and hence **learning** becomes the central issues in neural networks

Neural Network
as an adaptive machine

- Neural Network
  - 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
  - Neural network resemble the brain
    - Knowledge is acquired from the environment through a **learning process**
    - Connection strengths , known as synaptic weights, are used to store the acquired knowledge

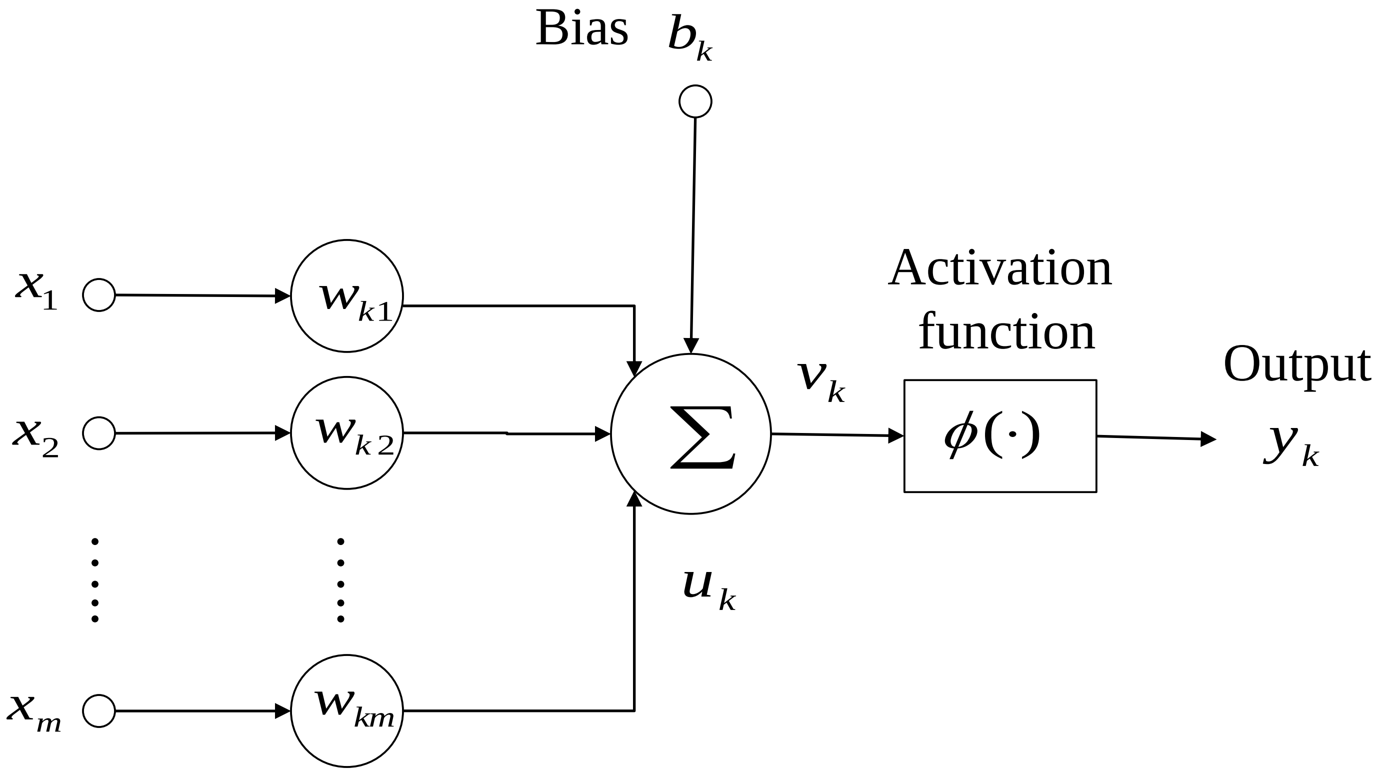
→ Leaning algorithm, weights, topology

Benefits of neural network

- Nonlinearity : distributed nonlinearity
- Input-output mapping
- Adaptivity : retain / adapt
- Evidential response : Decision + confidence
- Contextual Information
- Fault tolerance : Performance degrades gracefully
- VLSI implementability : Network of simple component
- Uniformity of analysis and design / Modular design
- Neurobiological analogy

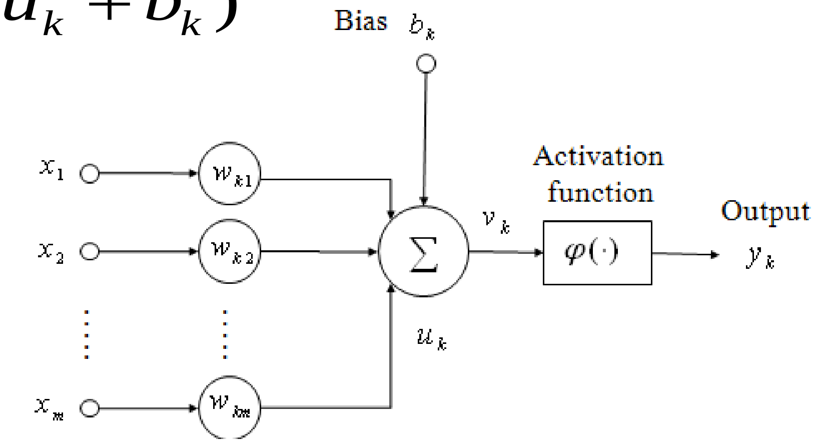
Models of Neurons (1)

- Modeling of Neuron



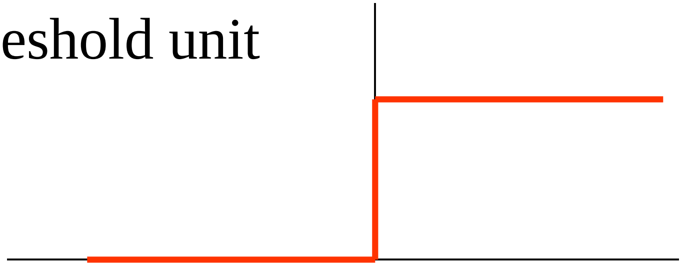
Models of Neurons (2)

- Modeling
  - Weight : $w_{kj} : j \rightarrow k$
  - Summing Junction: $u_k = \sum_{j=1}^{m} w_{kj} \cdot x_j$
  - Activation function: $y_k = \phi(u_k + b_k)$
  - Bias : $v_k = u_k b_k$ $v_k = u_k + b_k$



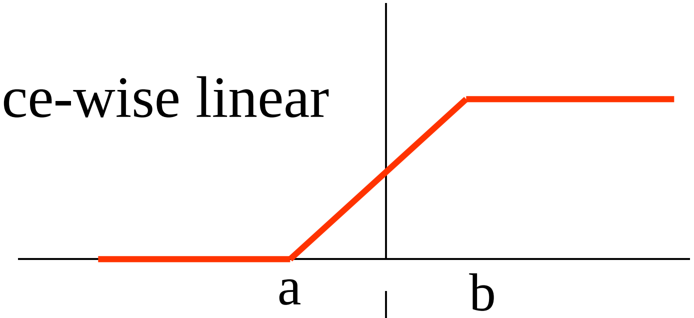
Activation function : $\phi(\cdot)$

- Activation function
  - Threshold unit



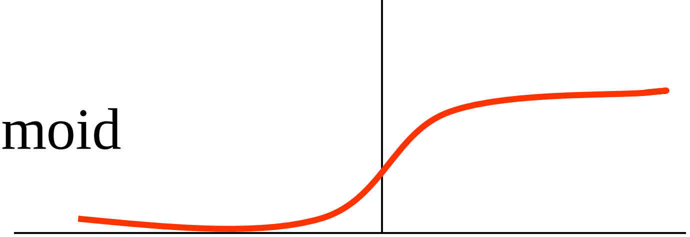
$$\phi(\cdot) = \begin{cases} 1 & if \ \ v \geq 0 \\ 0 \ , & v < 0 \end{cases}$$

  - Piece-wise linear



$$\phi(\cdot) = \begin{cases} 1 & v \geq b \\ -\dfrac{1}{b-a}(v-a) \ , & a < v < b \\ 0 & a \leq v \end{cases}$$

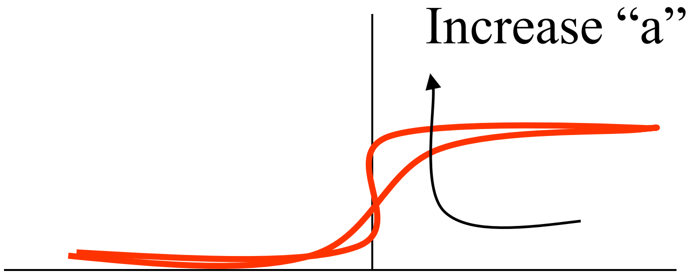
  - Sigmoid



$$\phi(v) = \frac{1}{1+e^{-av}}$$

Sigmoid, more

- Logistics function
  - a : slope parameter



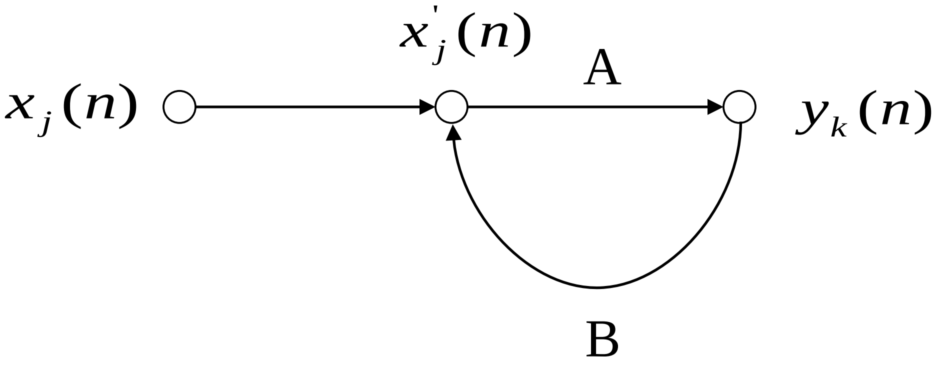
$$\phi(v) = \frac{1}{1+e^{-av}}$$

$$\phi'(v) = a \cdot \phi(v) \cdot (1-\phi(v))$$

- And other many functions

Feedback (1)

- And, feedback

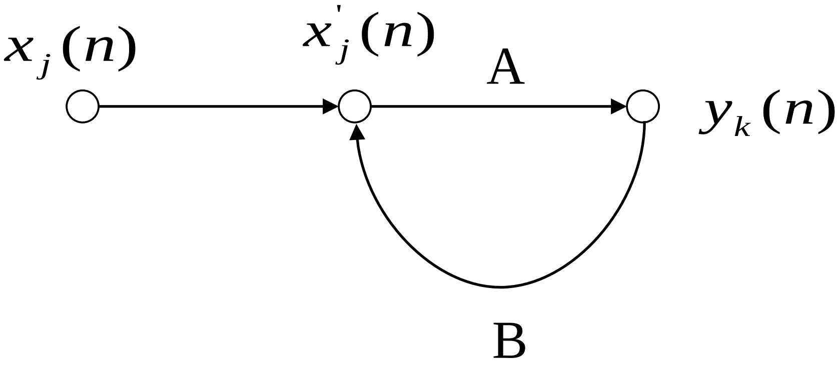


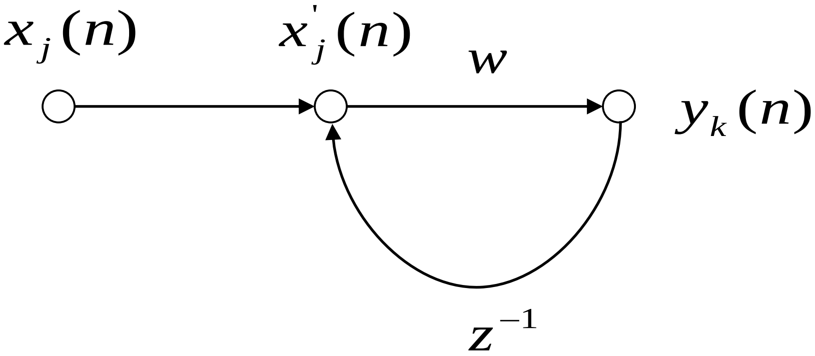
$$y_k(n) = A \cdot [x'_j(n)]$$

$$x'_j(n) = x_j(n) + B \cdot [y_k(n)]$$

$$y_k(n) = \frac{A}{1 - AB} x_j(n)$$

Feedback (2)





$$\frac{A}{1-AB}=\frac{w}{1-wz^{-1}}=w(1-wz^{-1})^{-1}=w\sum_{l=0}^{\infty}w^{l}z^{-l}$$

$$y_k(n)=w\sum_{l=0}^{\infty}w^{l}z^{-l}\cdot[x_j(n)]$$

$$z^{-l}\cdot[x_j(n)]=x_j(n-l) \qquad y_k(n)=\sum_{l=0}^{\infty}w^{l+1}\cdot[x_j(n-l)]$$

Feedback (3)

- Role of “w”
  - Case 1 : converge

$$|w| < 1$$

  - Case 2 : linearly diverge

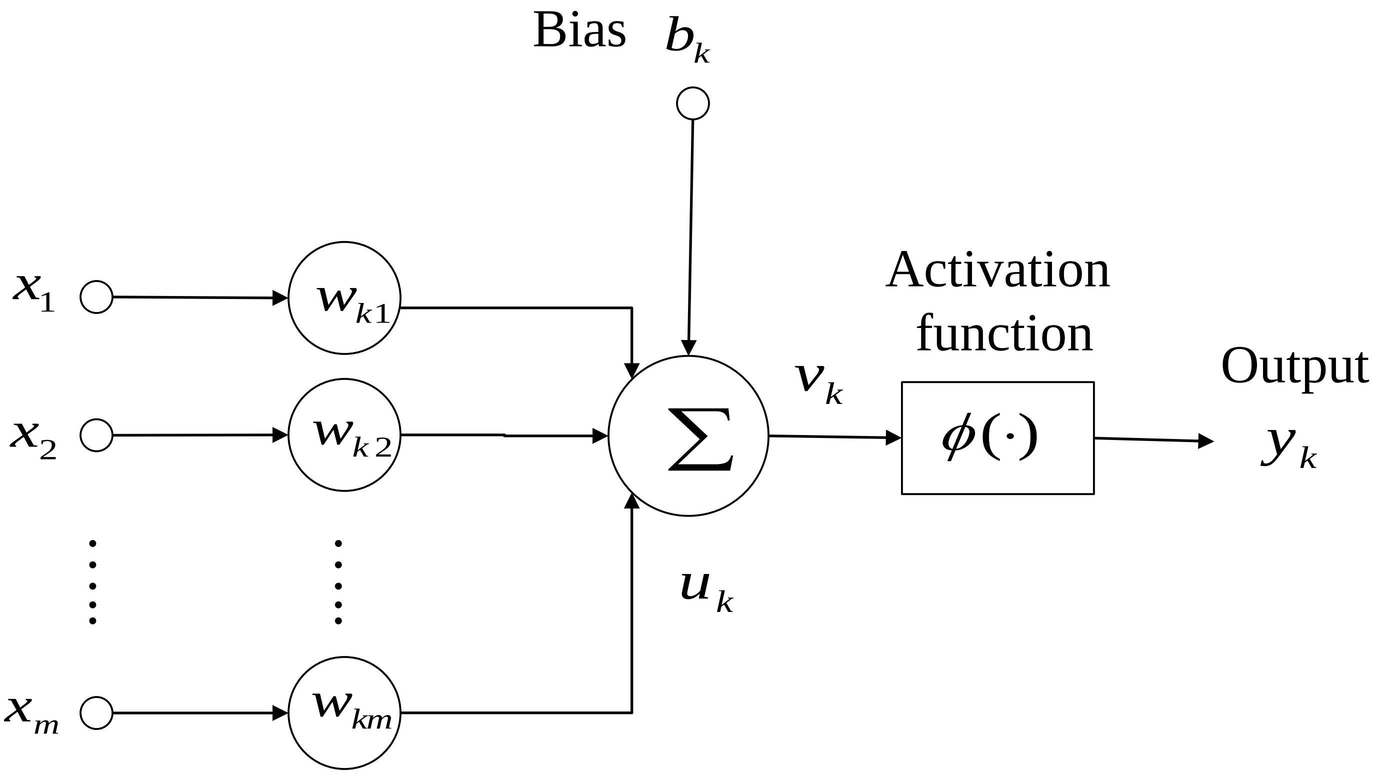
$$|w| = 1$$

  - Case 3 : expontially diverge

$$|w| > 1$$

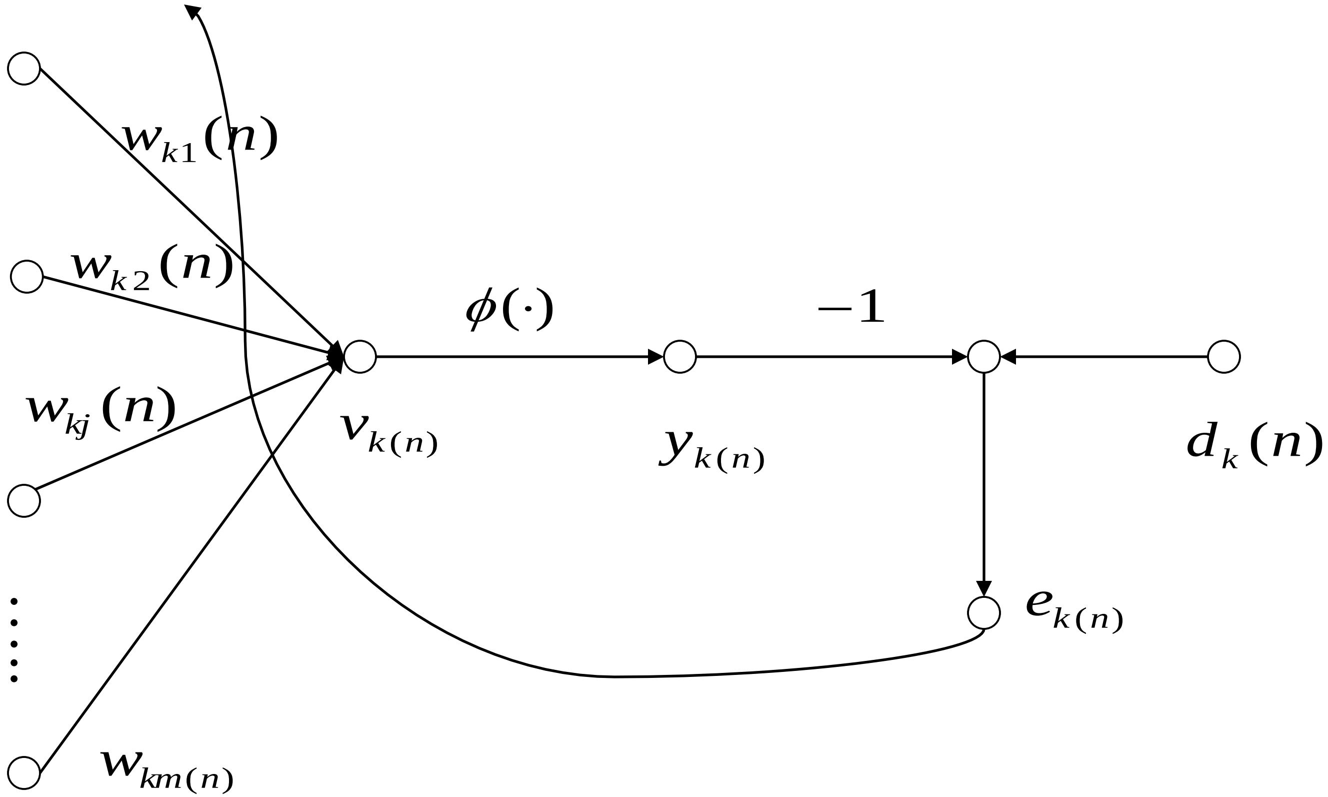
$$y_k(n) = \sum_{l=0}^{\infty} w^{l+1} \cdot [x_j(n-l)]$$

Learning (1)


Bias b_k
x_1
x_2
x_m
w_{k1}
w_{k2}
w_{km}
$\sum$
v_k
u_k
Activation function
$\phi(\cdot)$
Output
y_k

Learning (2)

- Redesign



Learning (3)

- Error

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

- Learning from “Error”

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

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

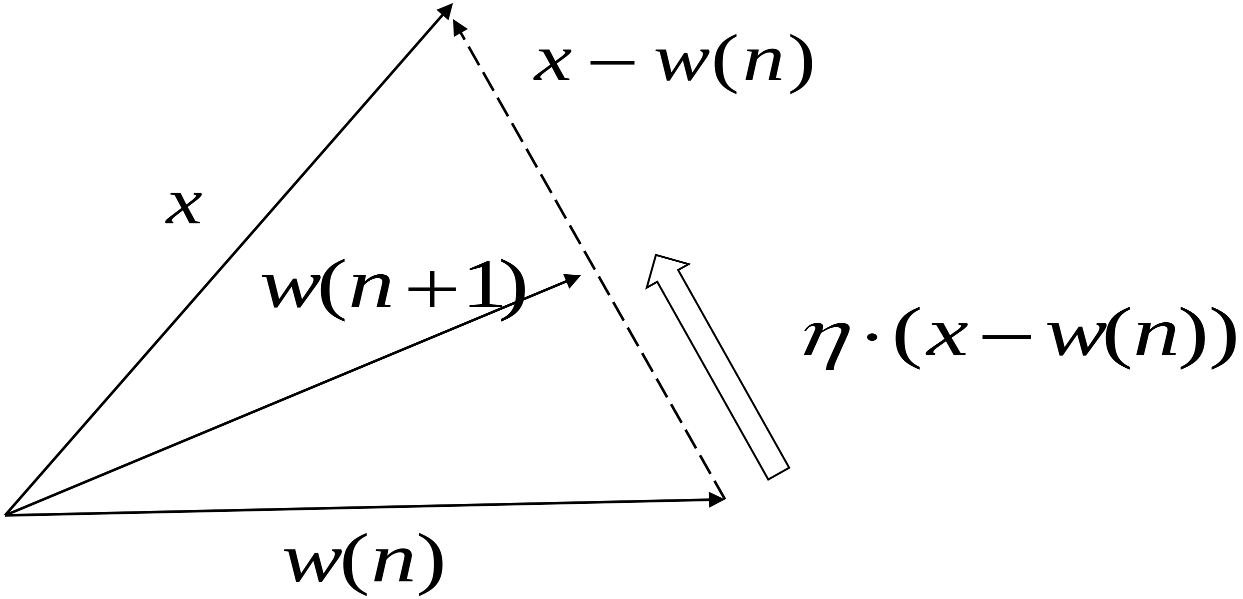
Hebbian Learning

- Donald Hebb’s postulate of leraning (1949)
  - “When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A’s efficiency as one of the cells firing B, is increased”

- Hebbian Synapse
  - If two neurons on either side of a synapse are activated simultaneously, the synapse is strengthened
  - If they are activated are asynchronously, the synapse is weakened or eliminated.

Competitive learning

- Competitive learning → adaption



$$\Delta w_{kj} = \begin{cases} \eta \cdot (x_j - w_{kj}) & , \; if \;\; k \;\; is \;\; winner \\ 0 & \;\; ow \end{cases}$$

Basics of Neural Network

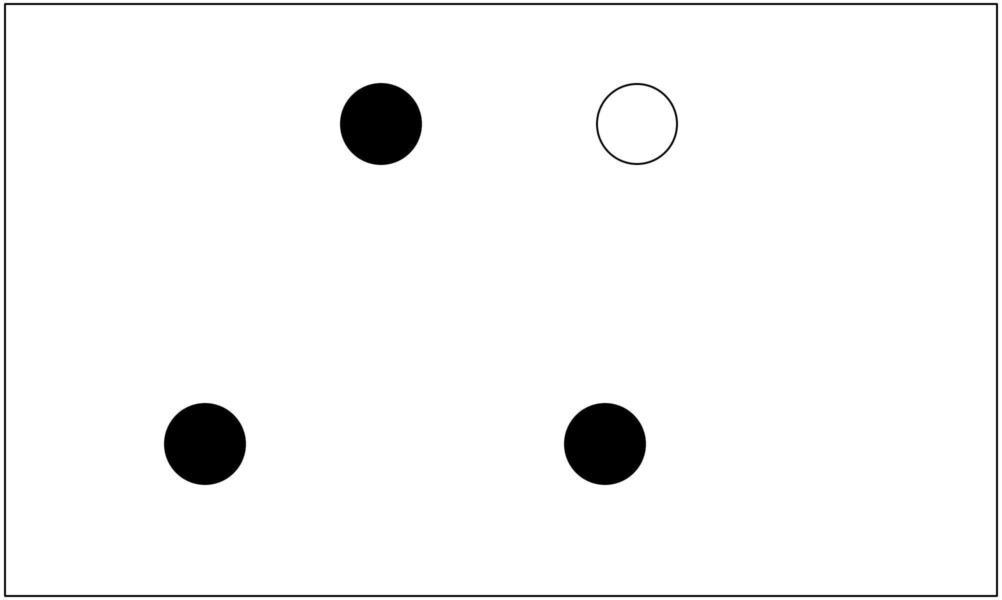
- VC Dimension
- Nonlinear programming
  - Unconstrained optimization techniques
    - Steepest descent
    - Newton's method
    - Gauss-Netwon's method
    - Linear Least Square filter
    - Least Mean Square algorithm

VC Dimension (1)

- “Shattering” a set of instance
  - A dichotomy of a set *S* is a partition of *S* into two disjoint subset.
  - A set of instances *S* is ***shattered*** by a function class F if and only if for every dichotomy of S there exists some function in F consistent with this dichotomy

VC Dimension (2)

- Shattering



VC Dimension (3)

- VC Dimension
  - The Vapnik – Chervonenkis dimension
  - VC (F), a function class F defined over sample space X is the size of the largest finite subset of X shattered by F
  - If arbitrarily large finite set of X can be shattered by F, then VC (F) = "infinite"

VC Dimension (4)

- VC Dimension of Linear Decision Surface
  - When F is a set of lines, and S a set of points, VC (F) =
  - Set of size 4 cannot be shattered, for any combination of points

Unconstrained Optimization Techniques

- How can we adjust W(i) to gradually minimize e(i)?
  - Note that $e(i) = d(i) - y\ (i) = d(i) - x^T w(i)$
  - in other words, we want to minimize the cost function $\varepsilon(w)$ with respect to the weight vector w → find optimal solution w^*
  - Necessary condition for optimality

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

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

$$\nabla \varepsilon(w^*) = \left[\frac{\partial \varepsilon}{\partial w_1}, \frac{\partial \varepsilon}{\partial w_2}, \cdots, \frac{\partial \varepsilon}{\partial w_m} \right]^T$$

Steepest Descent (1)

- Iterative update algorithm

$$\min \varepsilon(w) \Rightarrow \quad \varepsilon(w(n+1)) < \varepsilon(w(n))$$

- Define the gradient vector , if $\nabla \varepsilon(w(n)) = g(n)$

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

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

Steepest Descent (2)

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

$$\boxed{f(x) = f(a) + f^{'}(a)(x-a) + \frac{f^{''}(a)(x-a)^2}{2!} + \cdots}$$

$$\varepsilon(w(n+1)) \approx \varepsilon(w(n)) + g^T(n)\Delta w(n)$$

$$\begin{aligned}\varepsilon(w(n+1)) &\approx \varepsilon(w(n)) - g^T(n)\cdot\eta\cdot g(n)\\ &= \varepsilon(w(n)) - \eta \| g(n) \|^2\end{aligned}$$

Steepest Descent (3)

- Example : $\min\ x_1^2 + x_2^2$

Newton’s method

- Newton’s method
  - A extension of steepest descent → second order term in the Taylor series is used
  - It is generally faster and shows a less erratic meandering compared to the steepest descent method
  - There are certain conditions to be met though, such as the Hessian matrix $\nabla^2 \varepsilon(w)$

Gauss-Newton method (1)

- Energy function → Sum of error square

$$\varepsilon(w) = \frac{1}{2}\sum_{i=1}^{n} e_i(w)^2$$

$$e_i(w) = e_i(w_k) + \left[\frac{\partial e_i}{\partial w}\right]_{w=w_k}^{T} \cdot (w - w_k)$$

$$e_i(w) = e_i(w_k) + J_e(w_k)\cdot(w - w_k)$$

Gauss-Newton method (2)

- Jacobian Matrix $J_e(w)$

$$J_e(w) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(1)}{\partial w_n} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_n} \end{bmatrix}$$

Gauss-Newton method (3)

- Example

$$e(x,y) = \begin{bmatrix} e_1(x,y) \\ e_2(x,y) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 \\ \cos(x) + \sin(y) \end{bmatrix}$$

$$J_e(w) = \begin{bmatrix} \dfrac{\partial e_1(x,y)}{\partial x} & \dfrac{\partial e_1(x,y)}{\partial y} \\ \dfrac{\partial e_2(x,y)}{\partial x} & \dfrac{\partial e_2(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x & 2y \\ -\sin(x) & \cos(y) \end{bmatrix}$$

  - For (x,y) = $(0.5\pi, \pi)$

$$J_e(w) = \begin{bmatrix} \pi & 2\pi \\ -1 & -1 \end{bmatrix}$$

Gauss-Newton method (4)

- Again, $e_i(w) = e_i(w_k) + J_e(w_k)\cdot(w - w_k)$

$$\varepsilon(w) = \frac{1}{2}\sum_{i=1}^{n} e_i(w)^2$$

$$\| e_i(w) \|^2 = \| e_i(w_k) \|^2 + 2e(w_k)^T \cdot J_e(w_k)\cdot(w - w_k)$$
$$(w - w_k)^T \cdot J_e^T(w_k)\cdot J_e(w_k)\cdot(w - w_k)$$

$$J_e^T(w_k)\cdot e(w_k) + J_e^T(w_k)\cdot J_e(w_k)\cdot(w - w_k) = 0$$

$$w = w_k - (J_e^T(w_k)\cdot J_e(w_k))^{-1}\cdot J_e^T(w_k)\cdot e(w_k)$$

Linear Least-Square Filter (1)

- Given m input and 1 output function $y(i) = \phi(x_i^T \cdot w_i)$

$$e(w) = d - [x_1, x_2, \cdots, x_n]^T \cdot w \qquad \phi(x) = x$$

$$d = [d_1, d_2, \cdots, d_n]^T$$

$$e(w) = d - X \cdot w$$

$$\nabla e(w) = -X^T$$

$$\begin{aligned} w &= w_k + (X^T X)^{-1} X^T (d - X w_k) \\ &= w_k + (X^T X)^{-1} X^T d - (X^T X)^{-1} X^T X w_k) \\ &= (X^T X)^{-1} X^T d \end{aligned}$$

Linear Least-Square Filter (2)

- Characteristics of LLS
  - X does not need to be a square matrix
  - The Jacobian of the error function only depends on the input, and is invariant w.r.t. the weight w.
  - Pseudo-inverse : X^{+}

$$w = (X^{T} X)^{-1} X^{T} d = X^{+} d$$

Linear Least-Square Filter (3)

- Example
  - X and d

$$X = \begin{bmatrix} 10 & 7 \\ 3 & 7 \\ 3 & 6 \\ 5 & 4 \end{bmatrix} \qquad d = \begin{bmatrix} 40 \\ 36 \\ 31 \\ 22 \end{bmatrix} \qquad w = ?$$

Least-Mean-Square algorithm (1)

- From energy function

$$\varepsilon(w) = \frac{1}{2}\sum_{i=1}^{n} e_i(w)^2$$

$$\frac{\partial \varepsilon(w)}{\partial w} = e(w) \cdot \frac{\partial e(w)}{\partial w}$$

$$e = d - x^T \cdot w$$

$$\frac{\partial e(w)}{\partial w} = -x \qquad \frac{\partial \varepsilon(w)}{\partial w} = -xe(w)$$

$$w_{n+1} = w_n + \eta \cdot x_n \cdot e_n$$

Least-Mean-Square algorithm (2)

- Handling of η
  - The main problem arise because of the fixed η
  - One solution : Use a time varying learning rate $\eta(n) = \dfrac{c}{n}$
  - More better alternative → use hybrid method called search-then-converge

$$\eta(n) = \frac{\eta_0}{1 + n/\tau}$$

- Where τ : big number

The Perceptron model

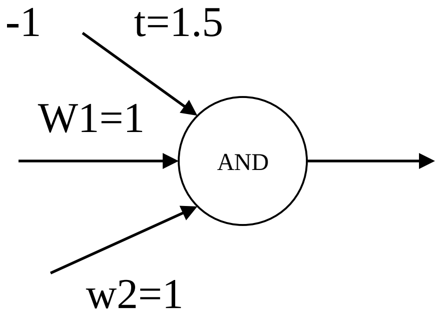
- Non-linear neuron model (McCulloch-Pitts model)
  - Objective : classify input vectors into two classes

$$v = \sum_{i=1}^{m} w_i x_i + b$$

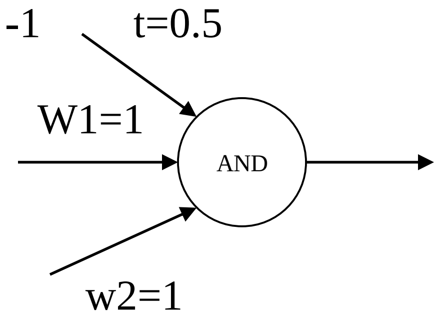
$$y = \phi(v) = \begin{cases} 1 & if \;\; v > 0 \\ 0 & if \;\; v \leq 0 \end{cases}$$

Boolean logic gates with Perceptron (1)

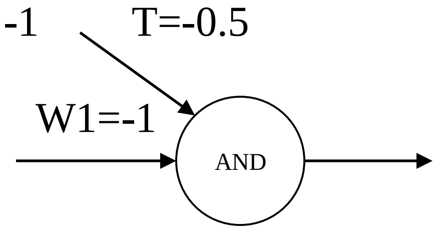
- "And" gate



- "Or" gate

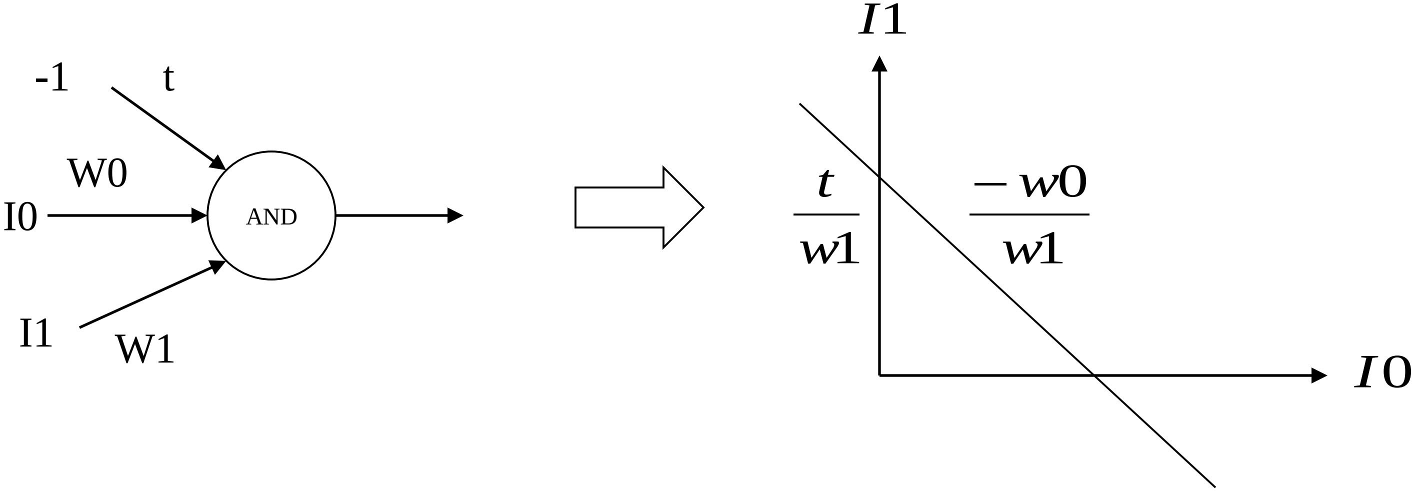


- "Not"



Boolean logic gates with Perceptron (2)

- Mechanism of Perceptron

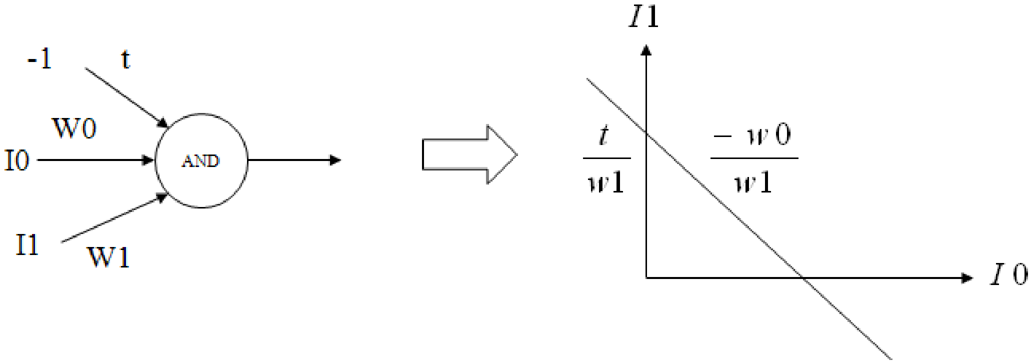


$$w_0 \cdot I_0 + w_1 \cdot I_1 - t > 0 \Rightarrow 1$$

$$w_0 \cdot I_0 + w_1 \cdot I_1 - t \leq 0 \Rightarrow 0$$

Boolean logic gates with Perceptron (3)

- Geometric interpretation



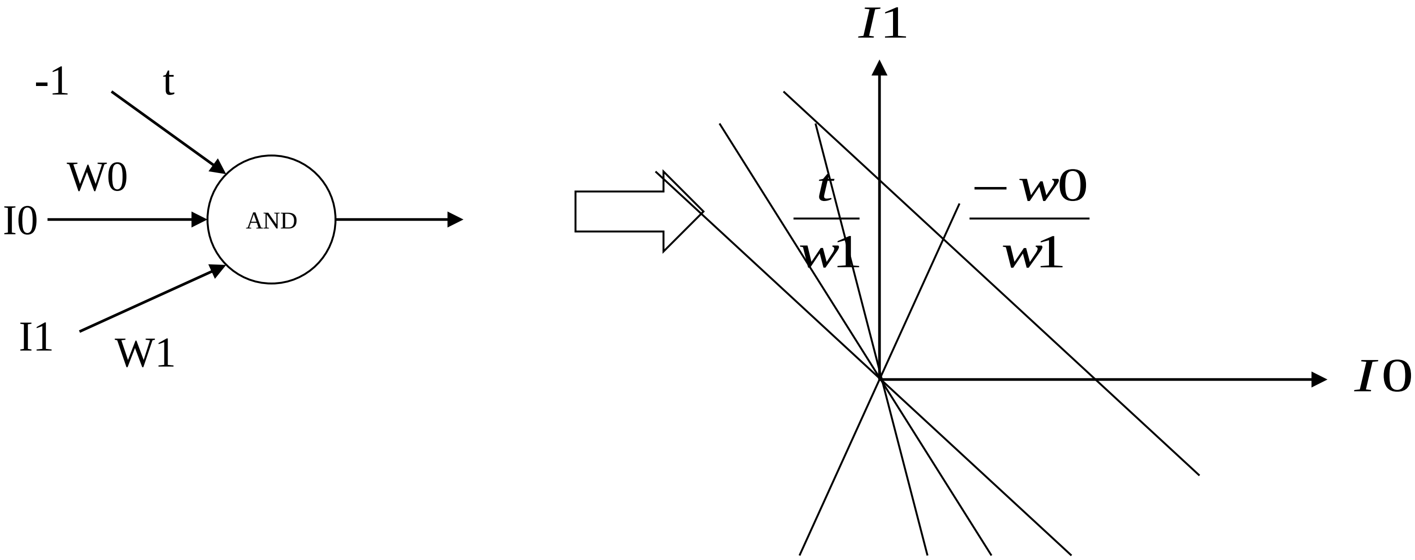
$$w_0 \cdot I_0 + w_1 \cdot I_1 - t > 0 \;\Rightarrow 1$$

$$I_1 > \frac{-w_0}{w_1} I_0 + \frac{t}{w_1}$$

$$y > \frac{-w_0}{w_1} x + \frac{t}{w_1}$$

The role of Bias

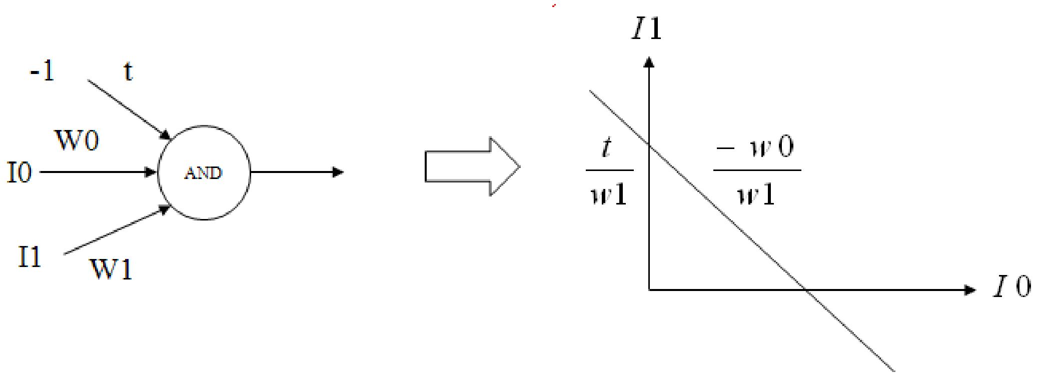
- Bias



  - Without the bias (t=0), learning is limited to adjustment of the slope of the separating line passing through the origin

Limitation of Perceptron

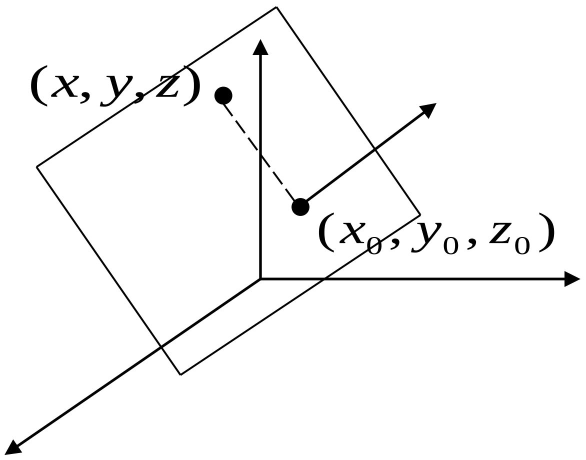
- Limitation

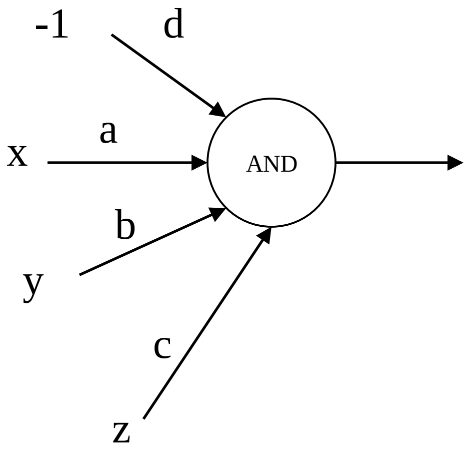


  - Only functions where the 0 points and 1 points are clearly linearly separable can be represented by perceptrons
  - The geometric interpretation is generalizable to function of n arguments, i.e. perceptron with n inputs plus on threshold (or bias) unit.

Generalization to n-dimension

- In N dimension





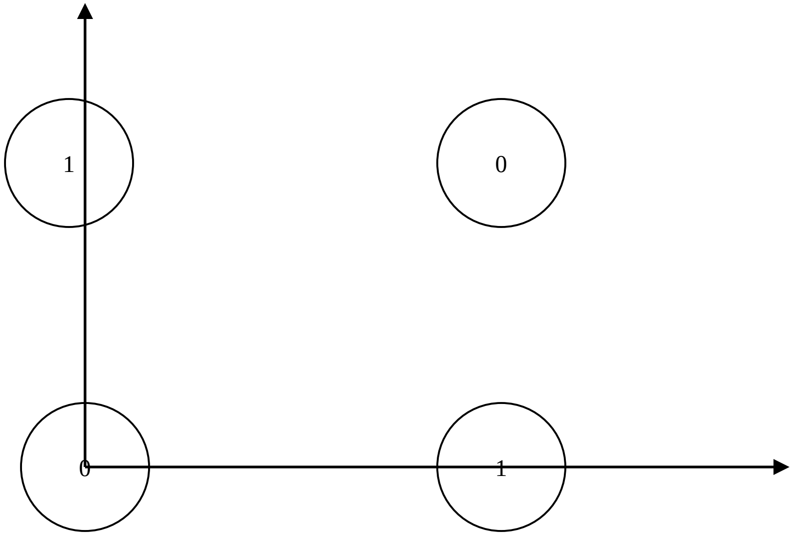
  - In 3D, perceptron has a role of cutting plane
  - For n-D input space, the decision boundary becomes a (n-1)-Hyperplane (1-D less than the input space)

Linear Separability (1)

- Characteristics of perceptron
  - For function that take integer or real values are arguments and output either 0 or 1
  - Perceptron cannot represent such a function – not linear - separable

Linear Separability (2)

- In “XOR” gate : Minsky and Papert (1969)



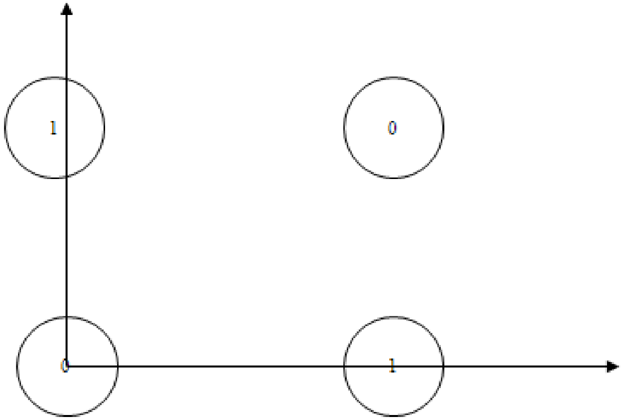
  - Perceptron cannot represent XOR

Linear Separability (3)

- In detail, XOR gate

	I0	I1	XOR
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

⇨



$$w_0 \cdot I_0 + w_1 \cdot I_1 - t > 0 \;\Rightarrow 1$$

Perceptron learning rule (1)

- Objective
  - Given a linearly separable set of inputs that can belong to class C1 or C2
  - The goal of perceptron learning is to have

$$w^T x > 0 \;\Rightarrow C_1$$

$$w^T x \leq 0 \;\Rightarrow C_2$$

- Terminating condition
  - All input satisfying $w(n)^T x > 0 \;\Rightarrow C_1$

$$w(n)^T x \leq 0 \;\Rightarrow C_2$$

  - Then

$$w(n+1) = w(n)$$

Perceptron learning rule (2)

- Learning rule

$$w(n+1) = w(n) - \eta(n) \cdot x(n) \Leftarrow w^T x > 0 \ and \ x \in C_2$$

$$w(n+1) = w(n) + \eta(n) \cdot x(n) \Leftarrow w^T x \leq 0 \ and \ x \in C_1$$

  - Or, simply

$$w(n+1) = w(n) + \eta(n) \cdot e(n) \cdot x(n)$$

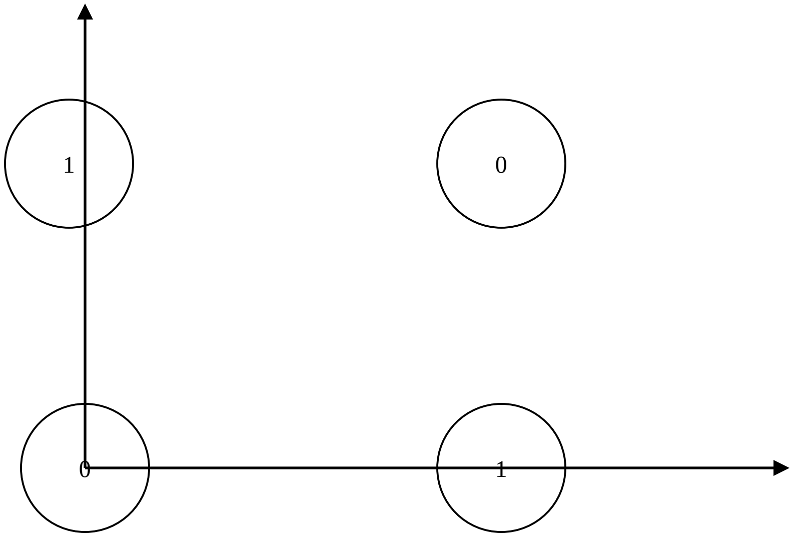
$$e(n) = d(n) - y(n)$$

Summary of LMS

- Summary
  - Adaptive filter using LMS algorithm and perceptron are closely related
  - LMS and perceptron are different, however, since one uses linear activation and the other hard limiters
  - LMS is used in continuous learning, while perceptrons are trained for only a finite number of steps
  - Single-neuron or single-layer has severe limits : how can multiple layers help?

Again, XOR gate

- How can we handle XOR gate problem with perceptron concept?



Simple Review (1)

- VC Dimension
  - VC (Line) = 3
  - VC (Triangle) = ?
  - VC (Rectangle) = ?

- Unconstraint Nonlinear Programming
  - Steepest Descent
  - Newton
  - Gauss-Newton
  - L.L.S Filter

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

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

Simple Review (1)

- Learning Process
  - Linear Least Square Filter

$$w = (X^T X)^{-1} X^T d = X^+ d$$

  - Linear Mean Square

$$\varepsilon(w) = \frac{1}{2}\sum_{i=1}^{n} e_i(w)^2$$

$$e = d - x^T \cdot w$$

$$\frac{\partial \varepsilon(w)}{\partial w} = e(w) \cdot \frac{\partial e(w)}{\partial w}$$

$$\frac{\partial e(w)}{\partial w} = -x$$

$$\frac{\partial \varepsilon(w)}{\partial w} = -xe(w)$$

$$w_{n+1} = w_n + \eta \cdot x_n \cdot e_n$$

Multi layer Perceptron (1)

- “From Haykin’s book”

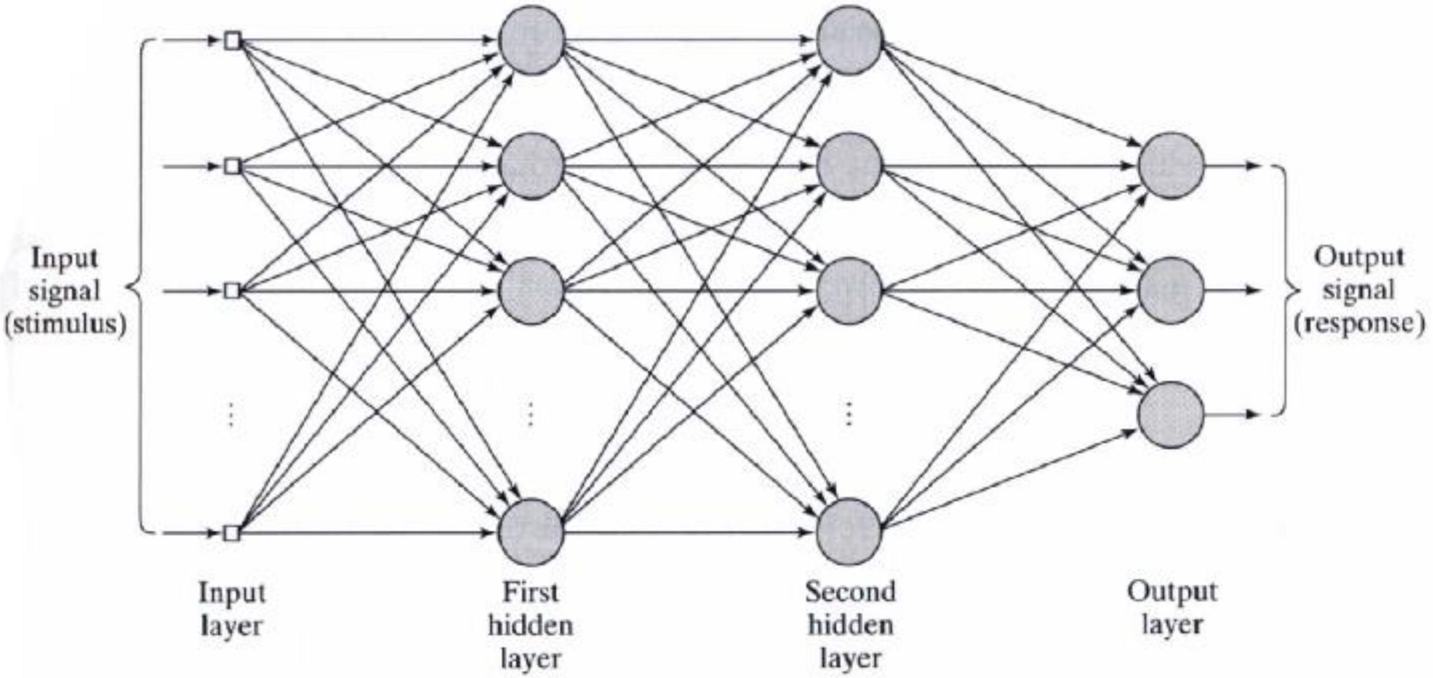


FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

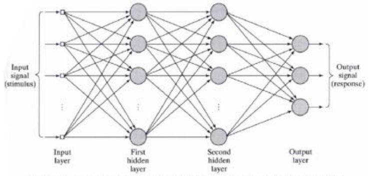


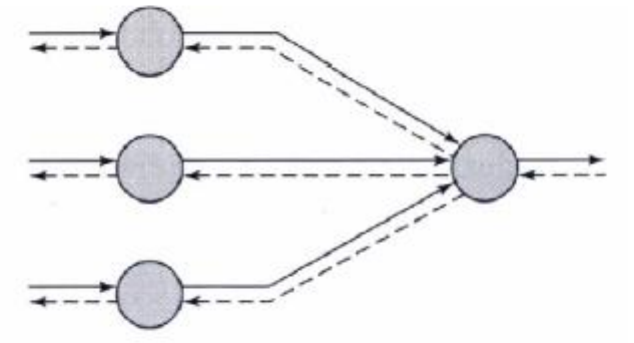
FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

Multi layer Perceptron (2)

- Characteristics of M.L.P
  - Learning algorithm : "Backpropagation" algorithm
    - Forward pass : activate the network, layer by layer
    - Backward pass : error signal backpropagates
      - From output to hidden
      - From hidden to input
  - Activation function : sigmoid

$$y_j = \frac{1}{1+e^{-av_j}}$$

  - It can have many hidden layers



Rearrangement of Activation function

$$y_j = \frac{1}{1+e^{-av_j}}$$

$$\phi(v_j) = \frac{1}{1+e^{-av_j}}$$

$$\phi'(v_j) = a\phi(v_j)(1-\phi(v_j))$$

$$\phi'(v_j) = \phi(v_j)(1-\phi(v_j))$$

In Haykin's MLP model
(that is, a=1)

Redesign

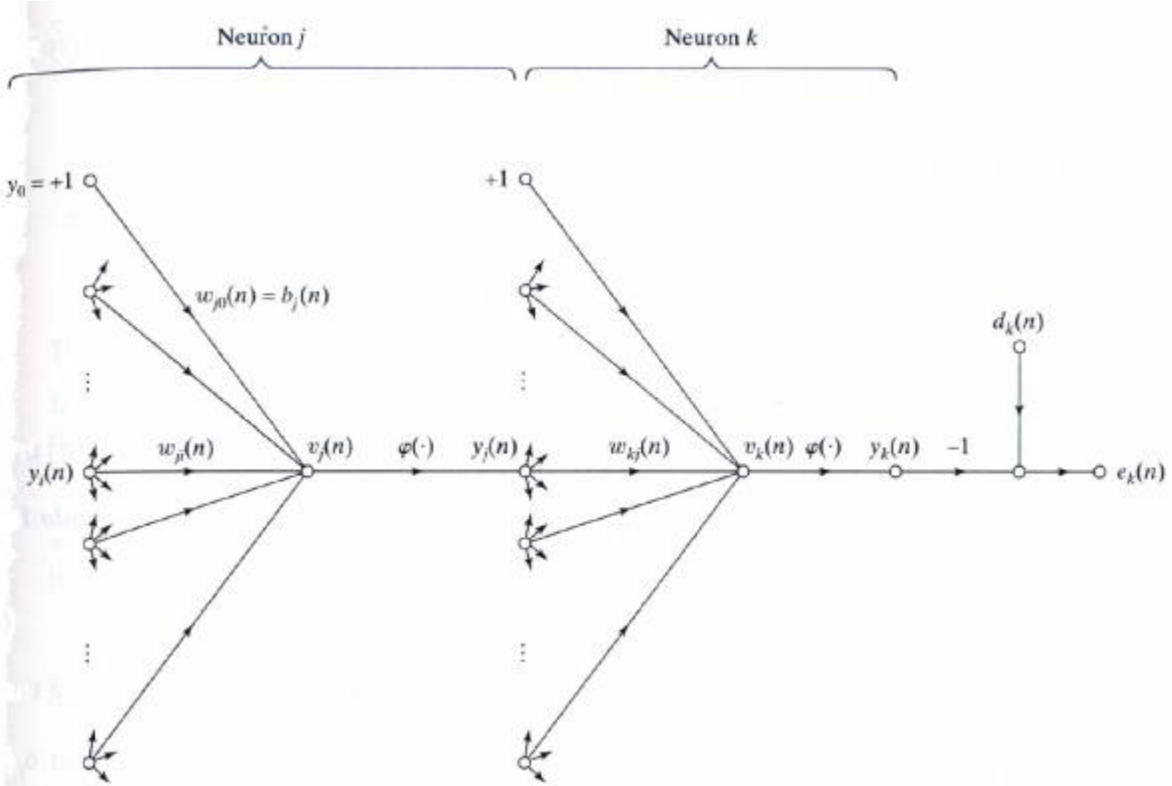
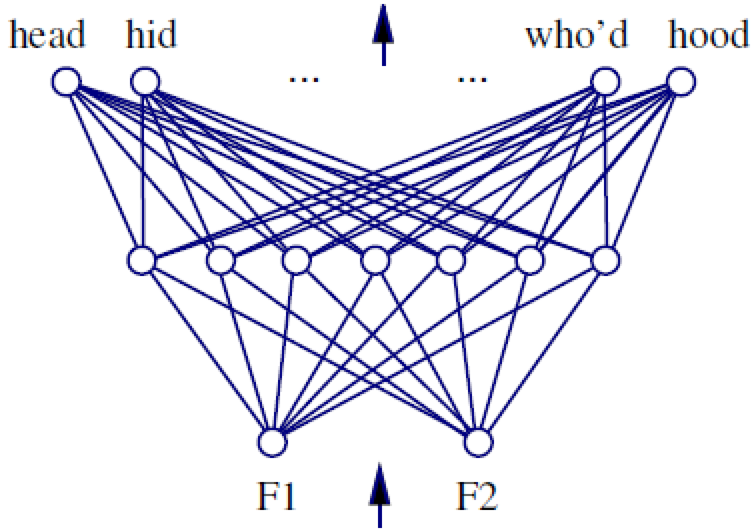
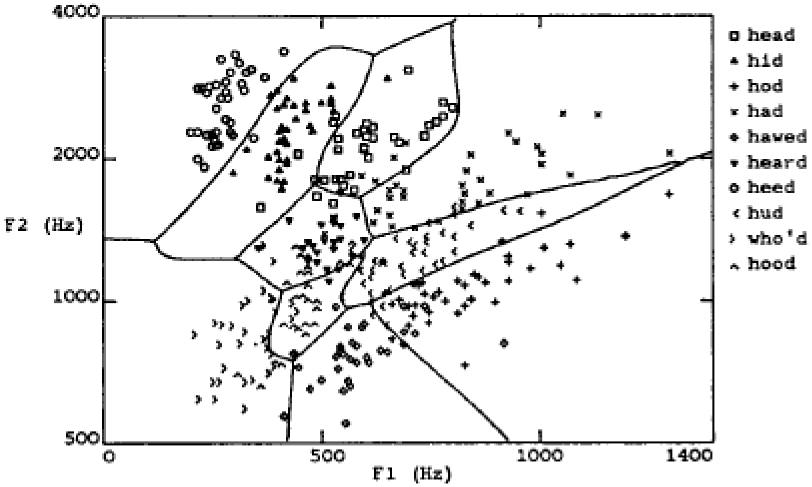


FIGURE 4.4 Signal-flow graph highlighting the details of output neuron *k* connected to hidden neuron *j*.

MLP and Backpropagation (1)

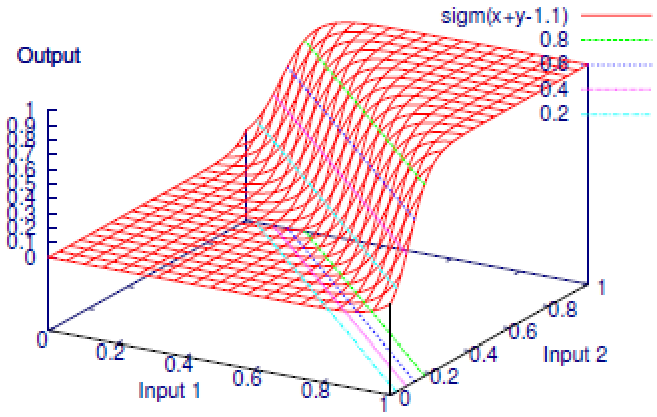
- Nonlinear decision surfaces



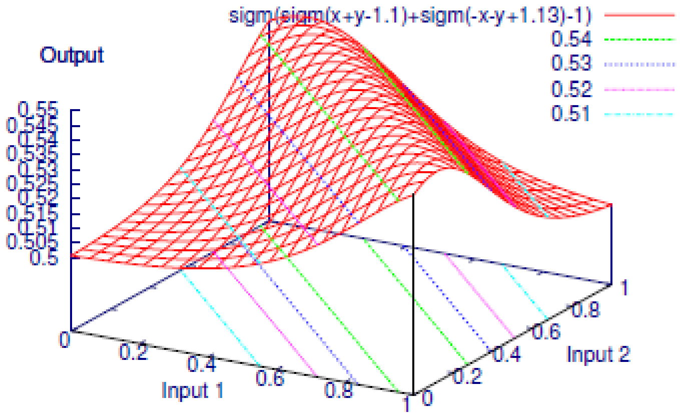


MLP and Backpropagation (2)

- In XOR gate



<One output>



<Two Hidden, One Output>

Industrial Engineering @ Kumoh National Institute of Technology

Error Gradient for a single sigmoid unit (1)

$$\{(x_k, d_k)\}_{k=1}^{n}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_k (d_k - y_k)^2$$

$$= \frac{1}{2}\sum_k \frac{\partial}{\partial w_i}(d_k - y_k)^2$$

$$= \frac{1}{2}\sum_k 2(d_k - y_k)\frac{\partial}{\partial w_i}(d_k - y_k)$$

$$= \sum_k (d_k - y_k)\left(\frac{-\partial y_k}{\partial w_i}\right)$$

$$= -\sum_k (d_k - y_k)\left(\frac{\partial y_k}{\partial v_k}\right)\left(\frac{\partial v_k}{\partial w_i}\right)$$

Error Gradient for a single sigmoid unit (2)

$$\frac{\partial E}{\partial w_i} = -\sum_k (d_k - y_k)\left(\frac{\partial y_k}{\partial v_k}\right)\left(\frac{\partial v_k}{\partial w_i}\right)$$

$$\frac{\partial y_k}{\partial v_k} = \frac{\partial \phi(v_k)}{\partial v_k} = y_k(1 - y_k)$$

$$\frac{\partial v_k}{\partial w_i} = \frac{\partial \phi(x_k^T w)}{\partial w_i} = x_{i,k}$$

$$\frac{\partial E}{\partial w_i} = -\sum_k (d_k - y_k) y_k (1 - y_k) x_{i,k}$$

Backpropagation (1)

- For output unit, j

$$\delta_j \leftarrow y_j(1-y_j)(d_j - y_j)$$

- For hidden unit, h

$$\delta_h \leftarrow y_h(1-y_h) \sum_{j \in output} w_j \delta_j$$

- Update weight

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

$$\Delta w_{ji} = \eta \delta_j x_i$$

Backpropagation (2)

- For output unit, j

$$\delta_j \leftarrow \underbrace{y_j(1-y_j)}_{\phi'(v_j)}\underbrace{(d_j - y_j)}_{Error}$$

- For hidden unit, h

$$\delta_h \leftarrow \underbrace{y_h(1-y_h)}_{\phi'(v_j)}\ \underbrace{\sum_{j \in output} w_j \delta_j}_{\text{Backpropagated error}}$$

Derivation of Δw : output unit (1)

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

$$E(w) \equiv \frac{1}{2} \sum_{j \in outputs} (d_j - y_j)^2$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial v_j}$$

$$\frac{\partial E}{\partial y_j} = \frac{\partial}{\partial y_j} \frac{1}{2} \sum_{j \in outputs} (d_j - y_j)^2$$

$$= \frac{\partial}{\partial y_j} \frac{1}{2} (d_j - y_j)^2$$

$$= 2 \frac{1}{2} (d_j - y_j) \frac{\partial (d_j - y_j)}{\partial y_j}$$

$$= -(d_j - y_j)$$

Derivation of Δw : output unit (2)

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} = -(d_j - y_j)\frac{\partial y_j}{\partial v_j}$$

$$\frac{\partial y_j}{\partial v_j}$$

$$y_j = \phi(v_j)$$

$$\frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} = -(d_j - y_j)y_j(1 - y_j)$$

$$\phi'(v_j) = y_j(1 - y_j)$$

$$\frac{\partial y_j}{\partial v_j} = y_j(1 - y_j)$$

Derivation of Δw : output unit (3)

- Finally,

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_{ji}} = -(d_j - y_j)y_j(1-y_j)$$

$$\frac{\partial v_j}{\partial w_{ji}} = x_i$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial v_j}\frac{\partial v_j}{\partial w_{ji}} = \underbrace{-(d_j - y_j)y_j(1-y_j)}_{\delta_j = error \times \phi'(net)}\underbrace{x_i}_{\text{input}}$$

Derivation of Δw : hidden unit (1)

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial v_j}\frac{\partial v_j}{\partial w_{ji}} = \frac{\partial E}{\partial v_j} x_i$$

$$\frac{\partial E}{\partial v_j} = \sum_{k \in downstream(j)} \frac{\partial E}{\partial v_k}\frac{\partial v_k}{\partial v_j}$$

$$= \sum_{k \in downstream(j)} -\delta_k \frac{\partial v_k}{\partial v_j}$$

$$= \sum_{k \in downstream(j)} -\delta_k \frac{\partial v_k}{\partial y_j}\frac{\partial y_j}{\partial v_j}$$

$$= \sum_{k \in downstream(j)} -\delta_k w_{kj} \frac{\partial y_j}{\partial v_j}$$

$$= \sum_{k \in downstream(j)} -\delta_k w_{kj} y_j (1 - y_j)$$

Derivation of Δw : hidden unit (2)

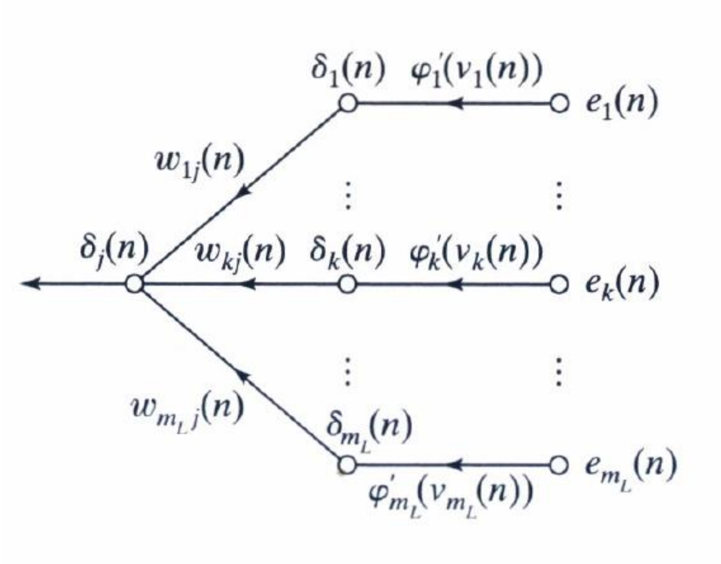
- Finally

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial v_j}\frac{\partial v_j}{\partial w_{ji}} = \frac{\partial E}{\partial v_j} x_i$$

$$\frac{\partial E}{\partial v_j} = \sum_{k \in downstream(j)} -\delta_k w_{kj} \underbrace{y_j(1-y_j)}_{\phi'(net)}$$

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = \eta \underbrace{\left[\underbrace{y_j(1-y_j)}_{\phi'(net)} \underbrace{\sum_{k \in downstream(j)} \delta_k \; w_{kj}}_{\text{error}} \right]}_{\delta_j} x_i$$

Summary



$$\underbrace{\Delta w_{ji}(n)}_{\text{Weight correction}} = \underbrace{\eta}_{\text{Learning rate}} \cdot \underbrace{\delta_j(n)}_{\text{Local gradient}} \cdot \underbrace{y_i(n)}_{\text{Input signal}}$$