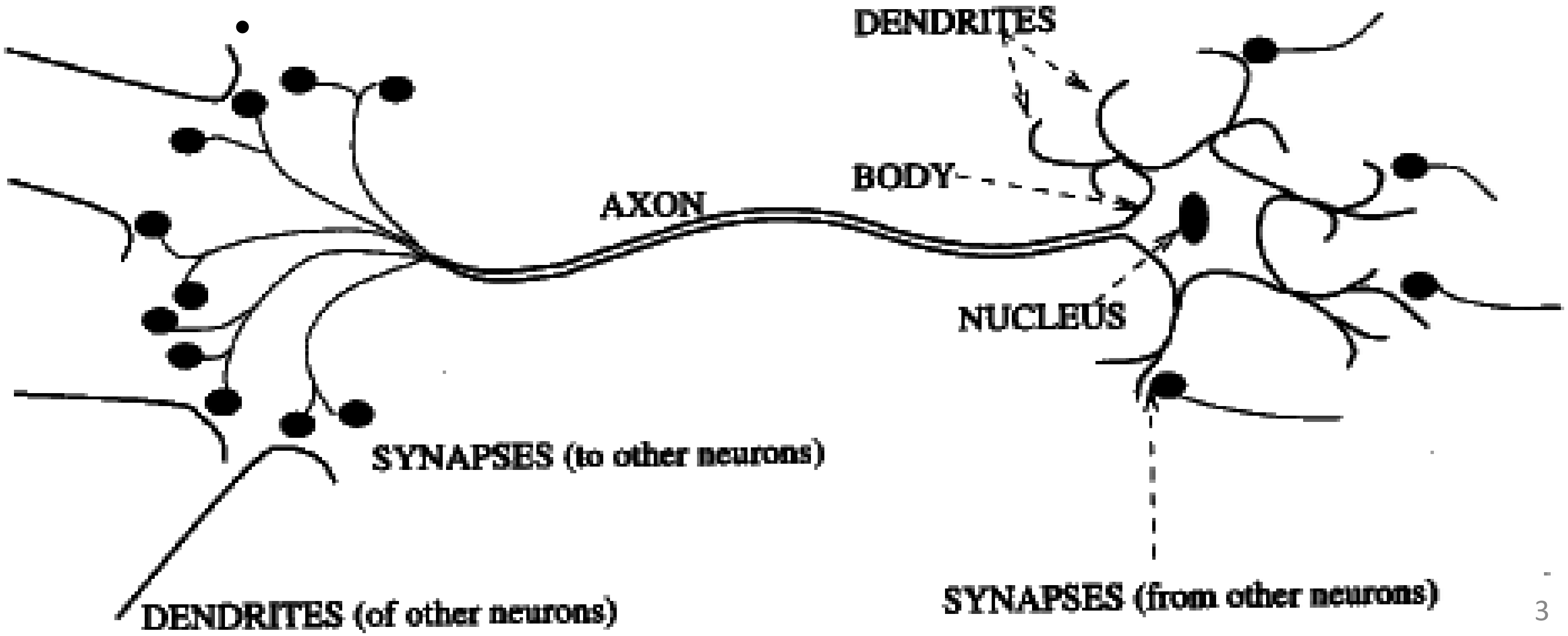
Artificial Neural Networks

Artificial Neural Networks

- What?
  - Computing Systems inspired by Biological Neural Networks.

Biological Neural Networks

- Nervous System
  - Biological Neural Networks
    - Biological Neurons
      - What?
        - Biological Neuron is an electrically excitable cell that processes and transmits information through electrical and chemical signals.



Biological Neural Networks

- Nervous System
  - Biological Neural Networks
    - Biological Neurons
      - 10 – 100 billion Neurons
      - connection to 100 – 10000 other neurons
      - 100 different types
      - layered structure

Biological Neural Networks

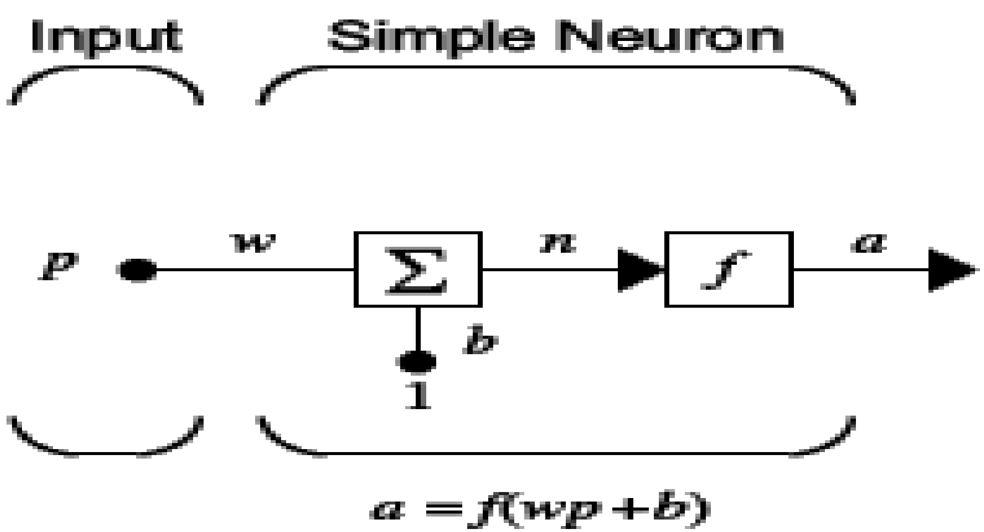
- Features
  - Parallel processing systems
  - Neurons are processing elements and each neuron performs some simple calculations
  - Neurons are networked
  - Each connection conveys a signal from one node (neuron) to another
  - Connection strength decides the extent to which a signal is amplified or diminished by a connection

Biological Neural Networks

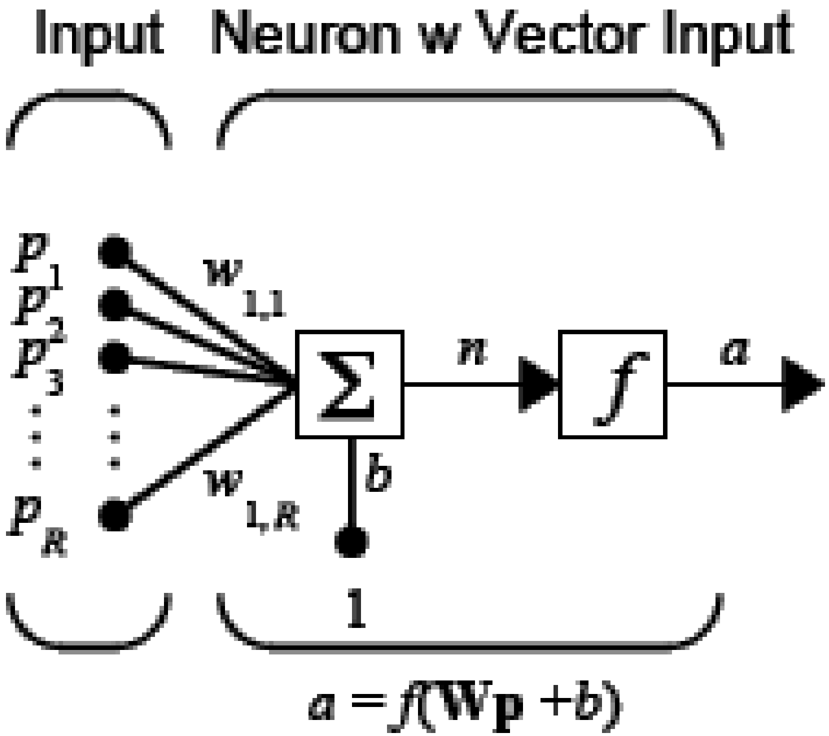
- Features (from our experience)
  - Ability to learn from experience and accomplish complex task without being programmed explicitly
    - Driving
    - Speaking using a particular language
    - Translation
    - Speaker Recognition
    - Face Recognition, etc…

Artificial Neuron Model

- An artificial neuron is a mathematical function regarded as a model of a biological neuron.
  - Remember: 1. BN is able to receive the amplified or diminished inputs from multiple dendrites 2. It is able to combine these inputs 3. It is able to process input and produce output
- Simple Neuron
  - Weight Function, Net Input Function & Transfer Function



Neuron with Vector Input

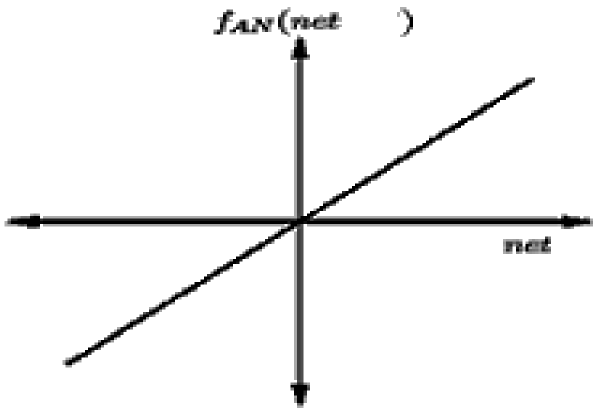


Where

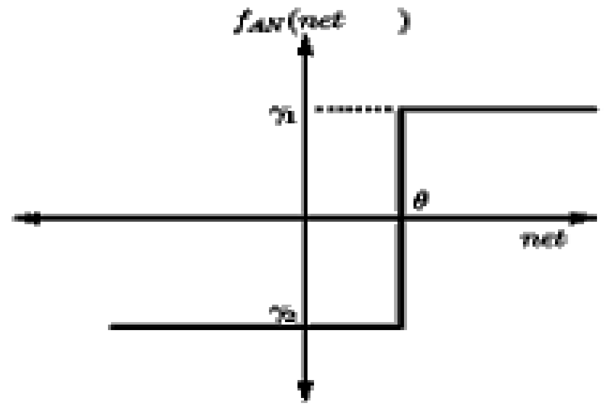
R = number of elements in input vector

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$$

Activation Functions

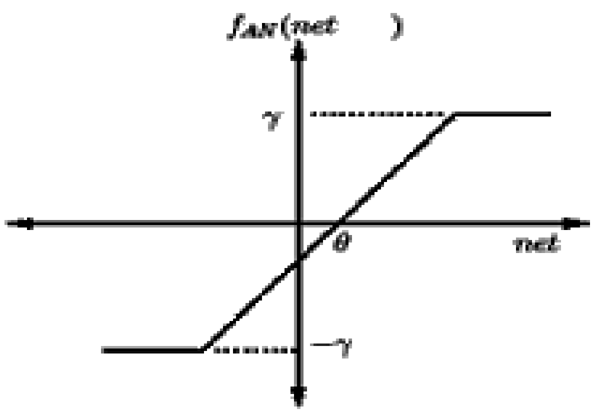


Linear function

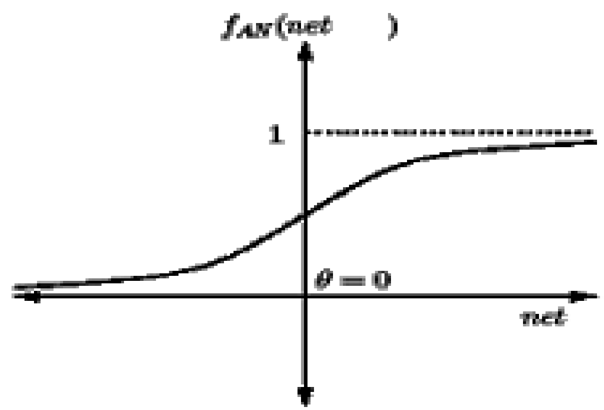


Step function

< & >=

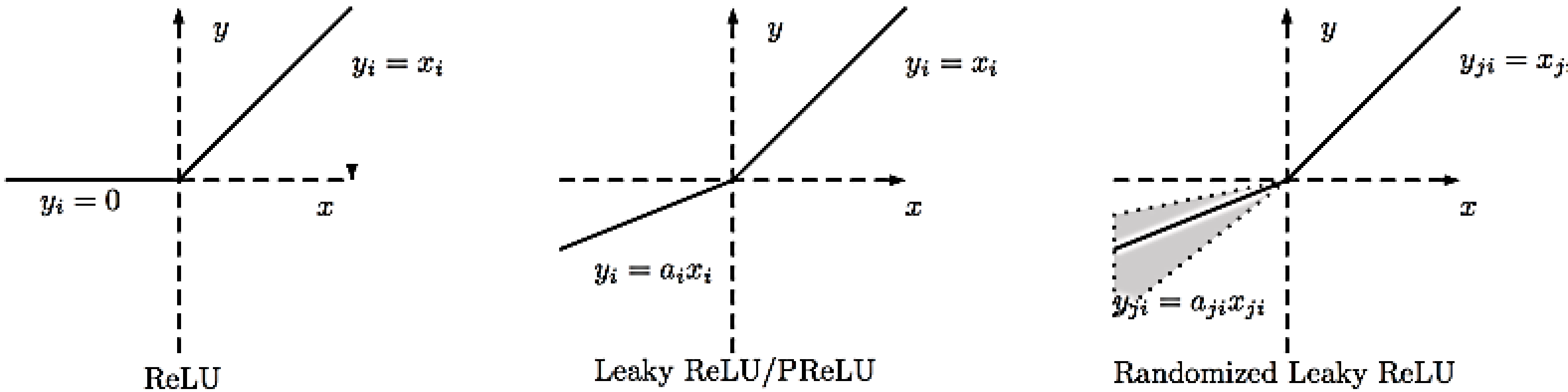


Ramp function



Sigmoid function

Activation Functions [12]



f(net) = max(0, net)

$0.01 * x_i / a_i * x_i$

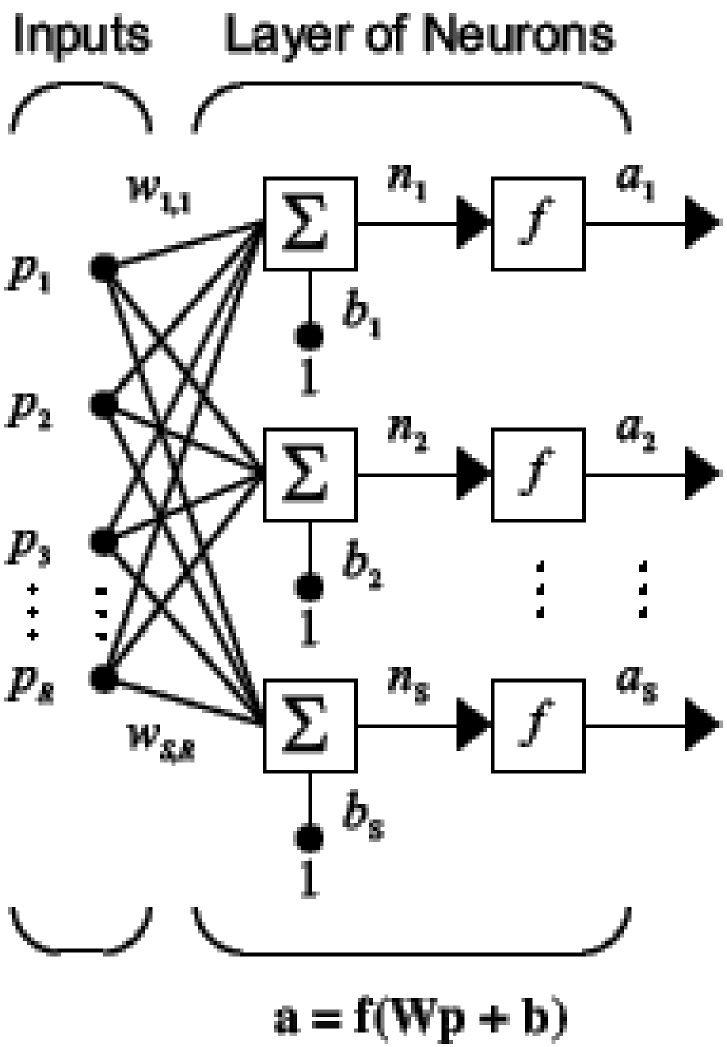
$$y_{ji} = \begin{cases} x_{ji} & \text{if } x_{ji} \geq 0 \\ a_{ji} x_{ji} & \text{if } x_{ji} < 0, \end{cases}$$

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0. \end{cases}$$

$$a_{ji} \sim U(l, u), l < u \text{ and } l, u \in [0, 1)$$

a_{ji} is a random number sampled from a uniform distribution U(l, u).

A Layer of Neurons

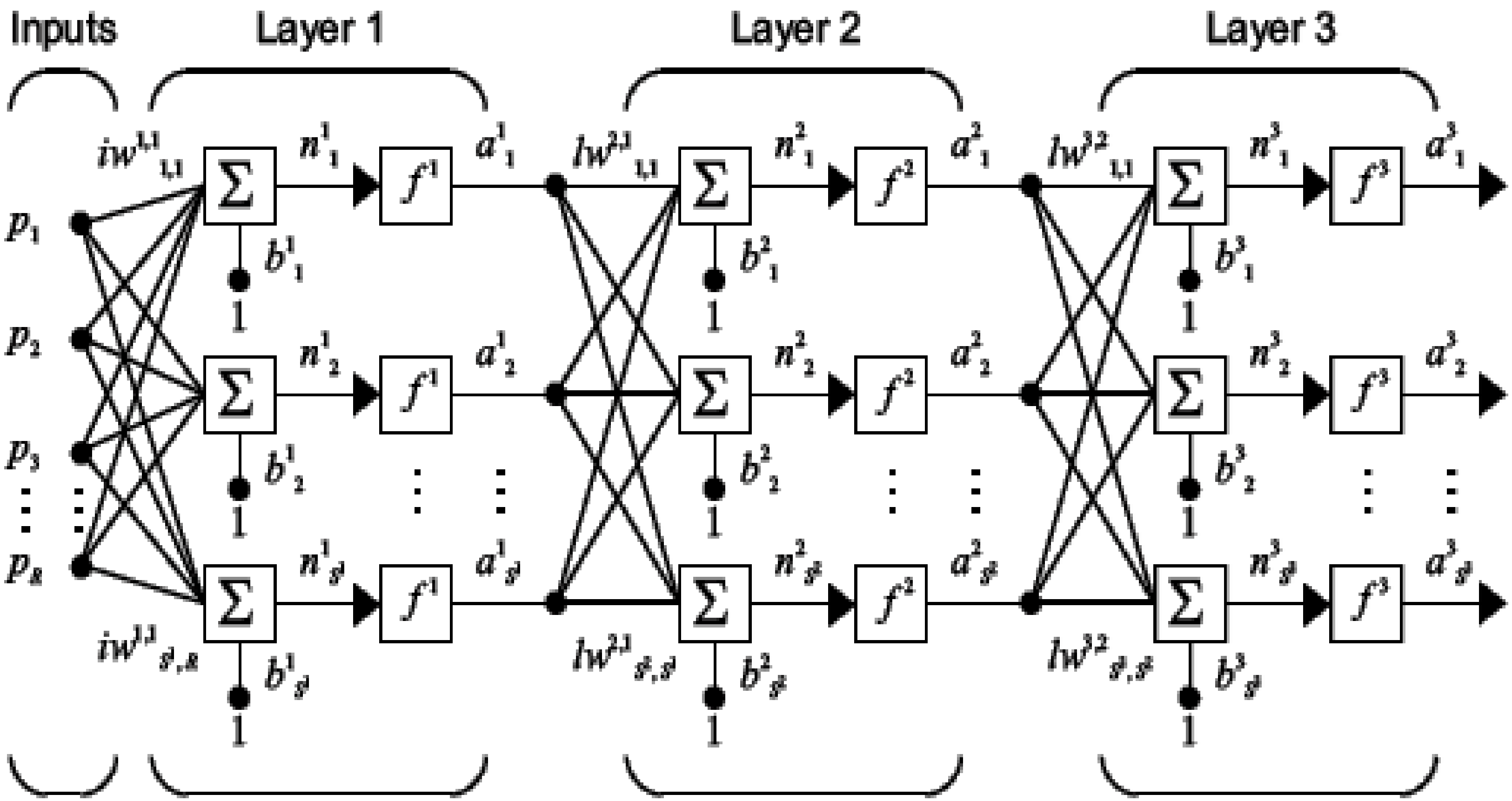


Where

R = number of elements in input vector

S = number of neurons in layer

Multiple Layers of Neurons

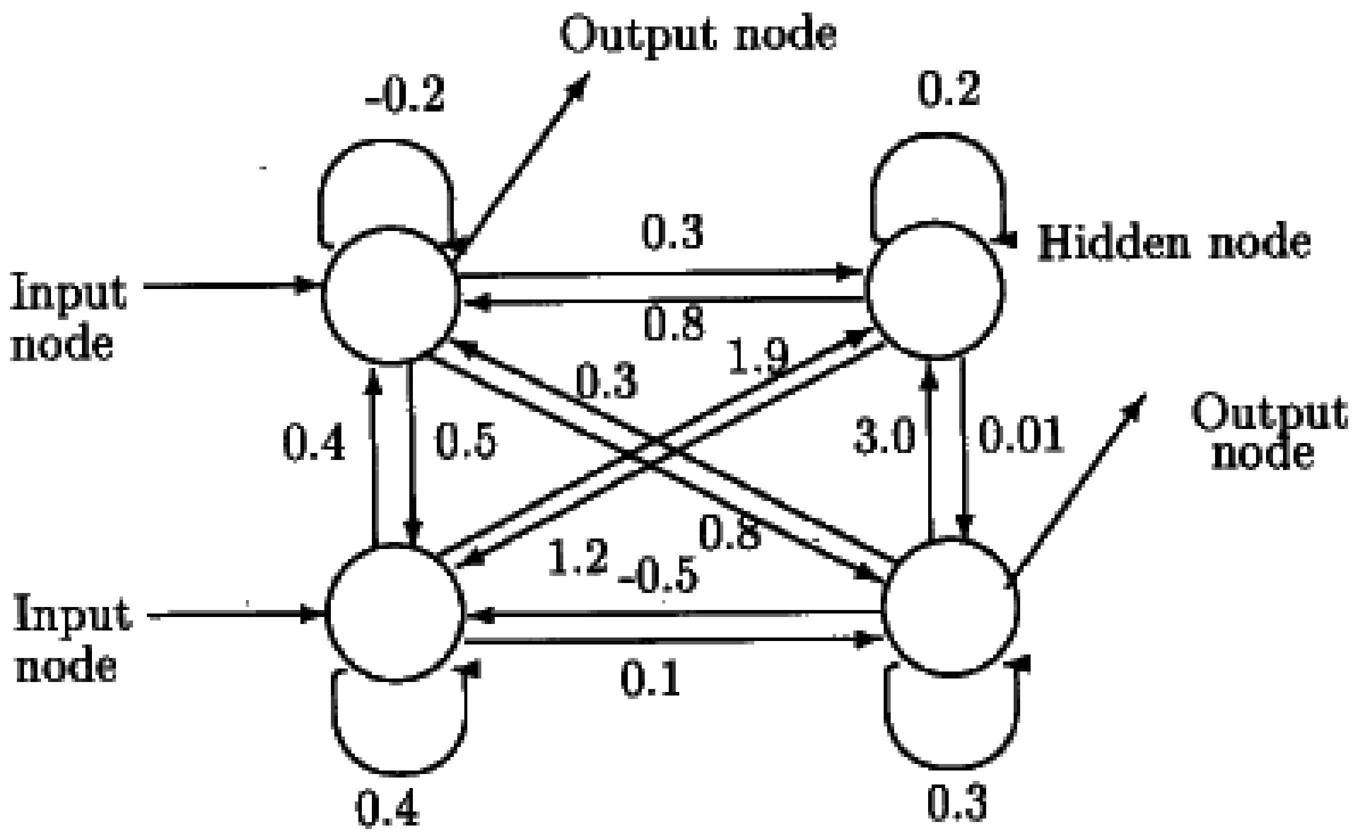


$$a^1 = f^1(IW^{1,1}p + b^1) \qquad a^2 = f^2(LW^{2,1}a^1 + b^2) \qquad a^3 = f^3(LW^{3,2}a^2 + b^3)$$

$$a^3 = f^3(LW^{3,2}f^2(LW^{2,1}f^1(IW^{1,1}p + b^1) + b^2) + b^3)$$

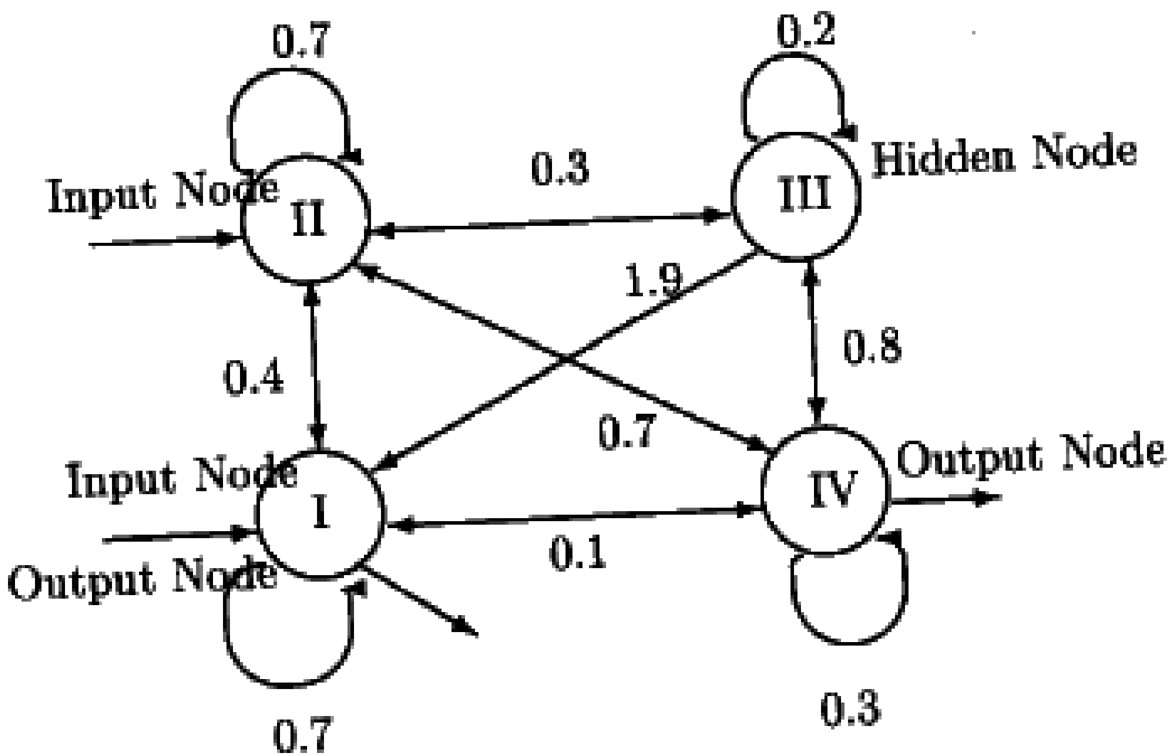
ANN Architectures

- Fully Connected Network (Asymmetric)



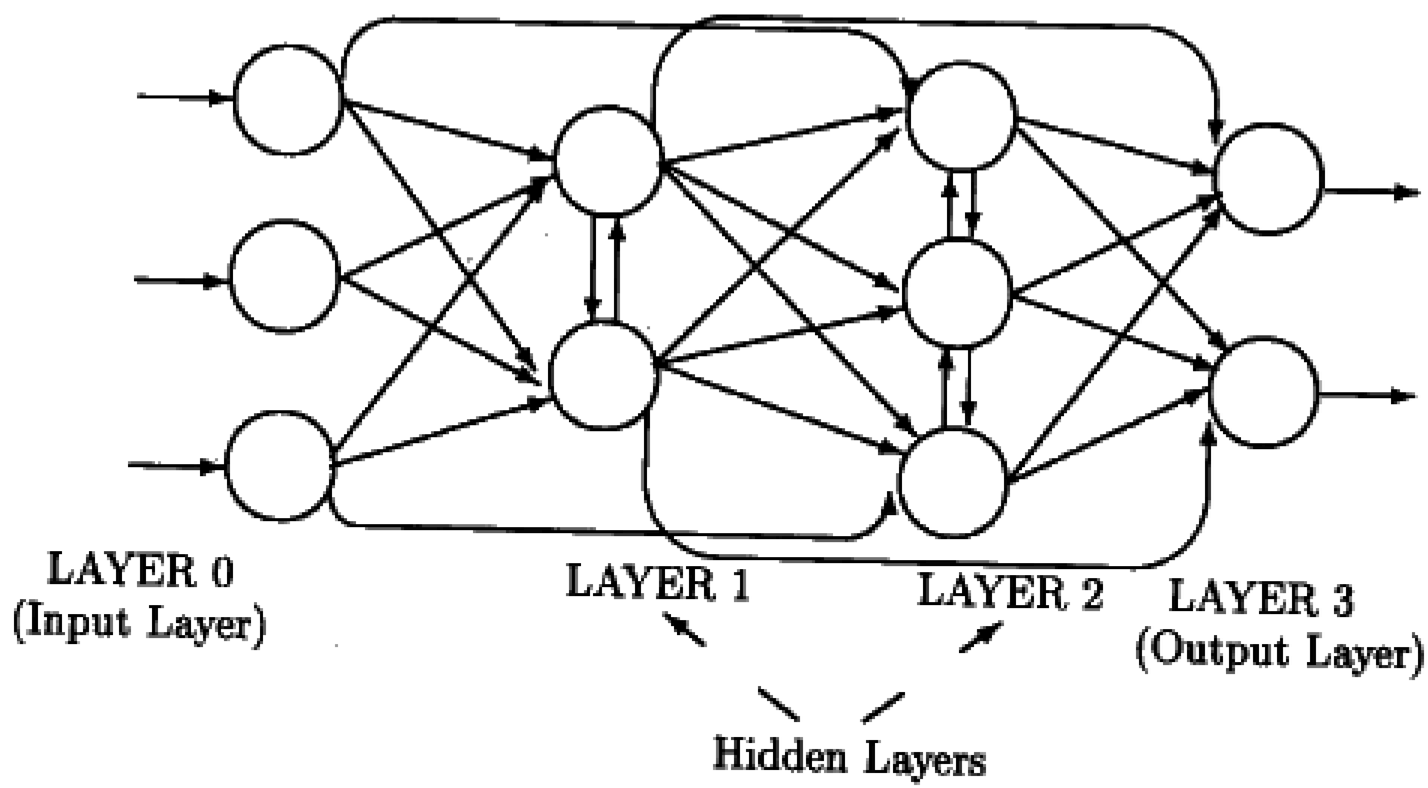
ANN Architectures

- Fully Connected Network (Symmetric)



ANN Architectures

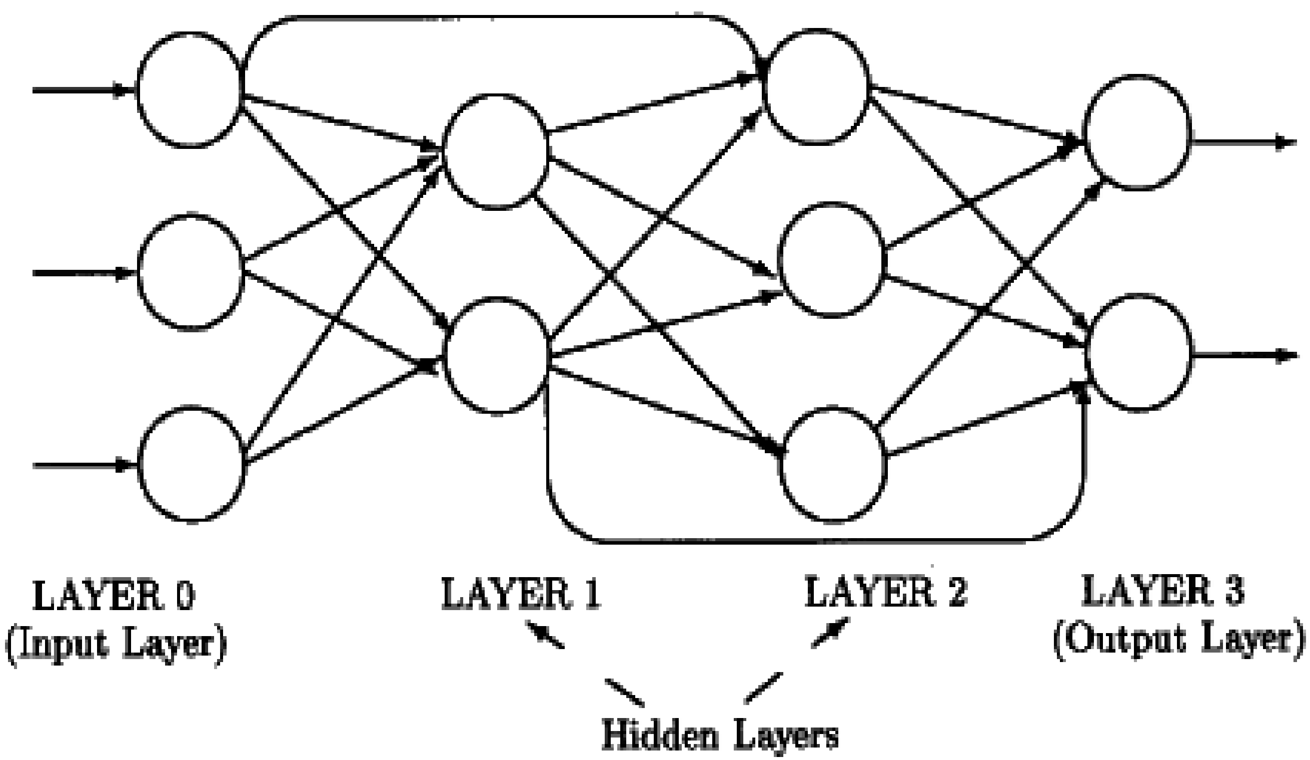
➢ Layered Network



These are networks in which nodes are partitioned into subsets called layers, with no connections that lead from layer j to layer k if j > k

ANN Architectures

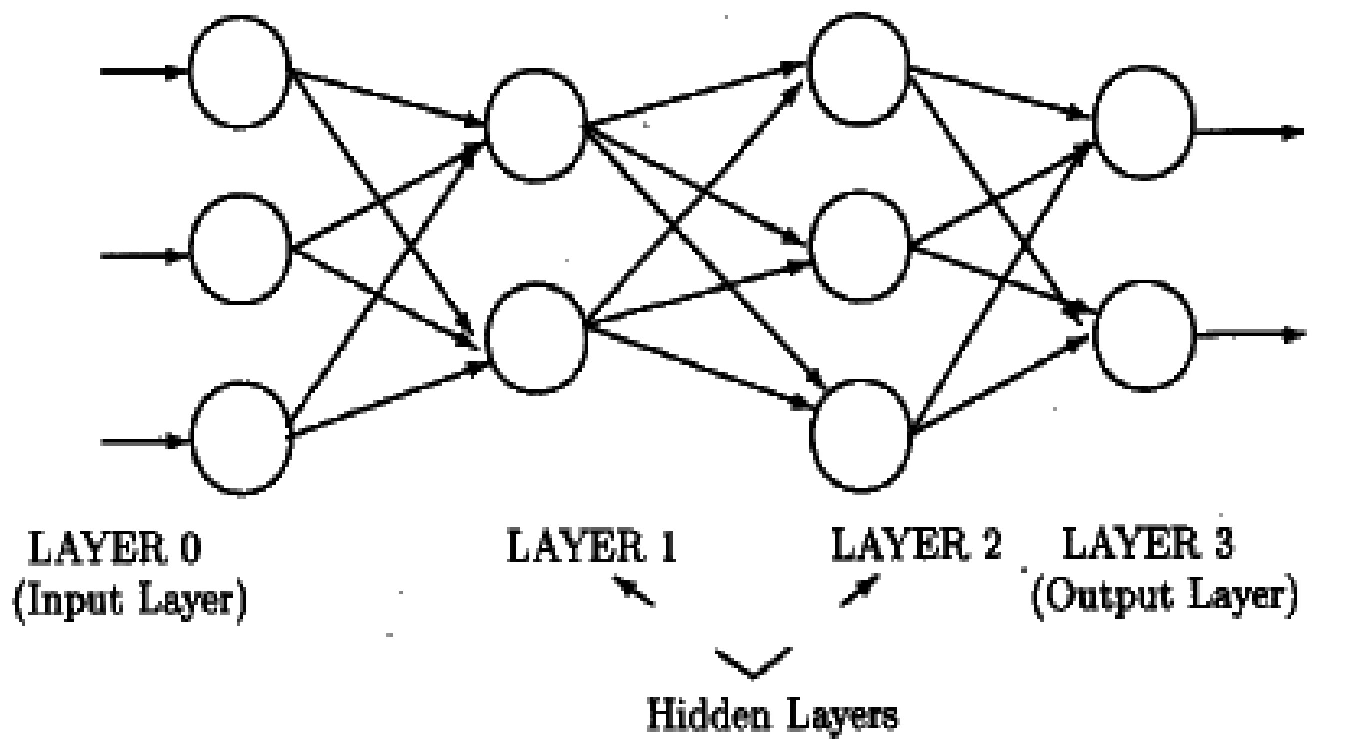
- Acyclic Network



These are subclass of layered networks with no intra-layer connections.

ANN Architectures

- Feedforward Network



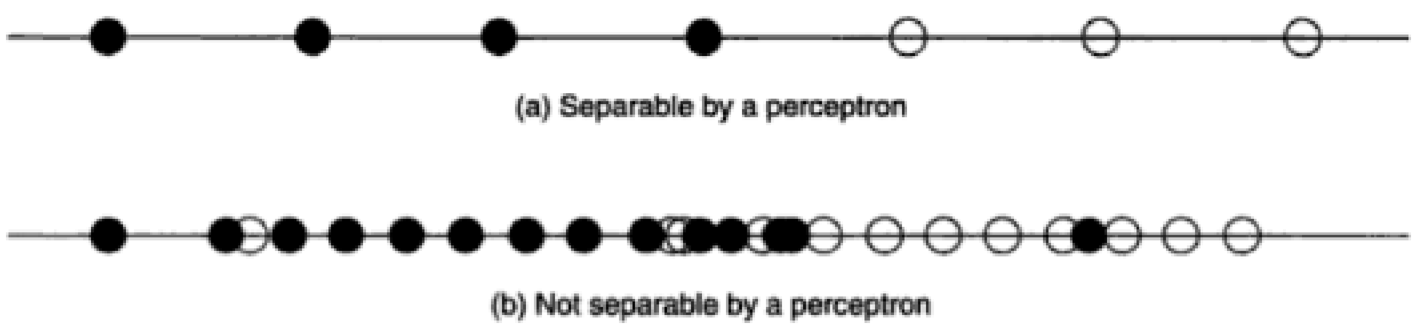
These are subclass of acyclic networks in which a connection is allowed from a node in layer i only to nodes in layer i + 1.

Learning in ANN

- Types of Learning
  - Supervised Learning
  - Unsupervised Learning

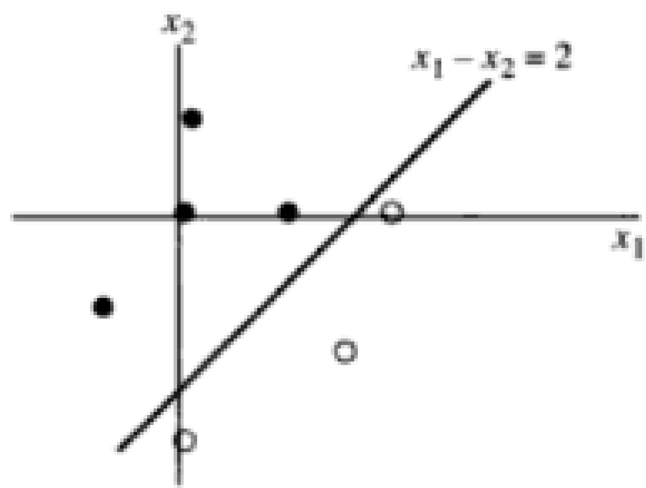
Linear Separability

- 1 – D Case
  - 7/5 Students data – Weight Values & Obese/Not Obese
    - (50, NO), (55, NO), (60, NO), (65, NO), (70, O), (75, O), (80, O) – Linearly Separable
    - (55, NO), (60, O), (65, NO), (70, O), (75, O) – Linearly Inseparable
  - Learning a separating point/line



Linear Separability

- 2 – D Case
  - Learning a separating line



(a)

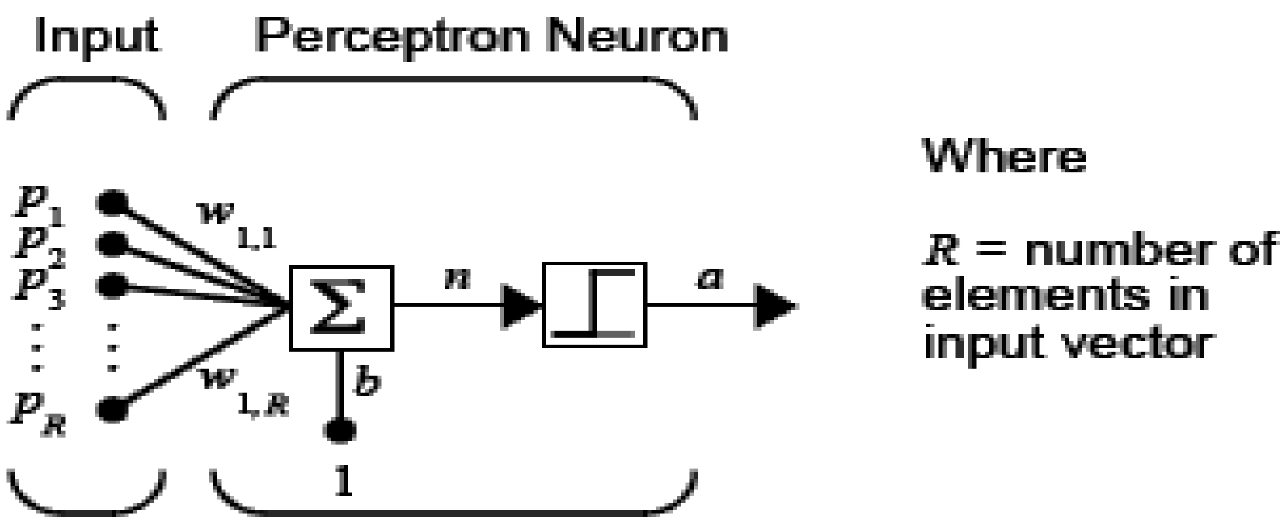
Linear Separability

- 3 – D Case
  - Learning a separating plane

- Higher Dimensional Case
  - Learning a separating hyperplane

Perceptron Model [6]

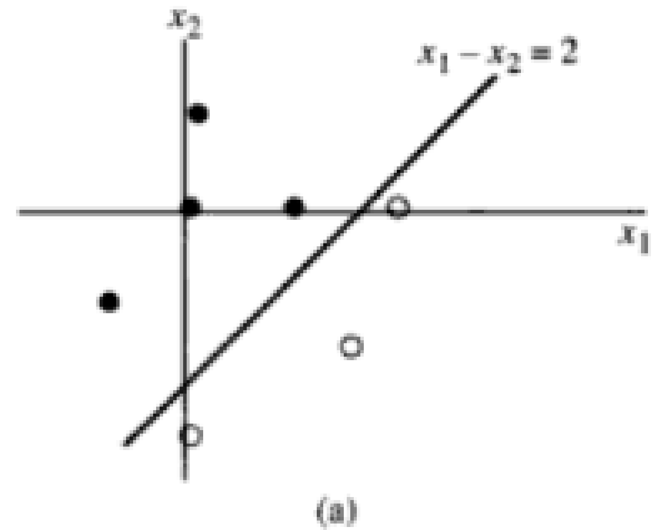
- What is Perceptron?
  - It is a machine which can learn (using examples) to assign input vectors to different classes.
- What can it do?
  - 2-class linear classification problem
    - What?
    - Process

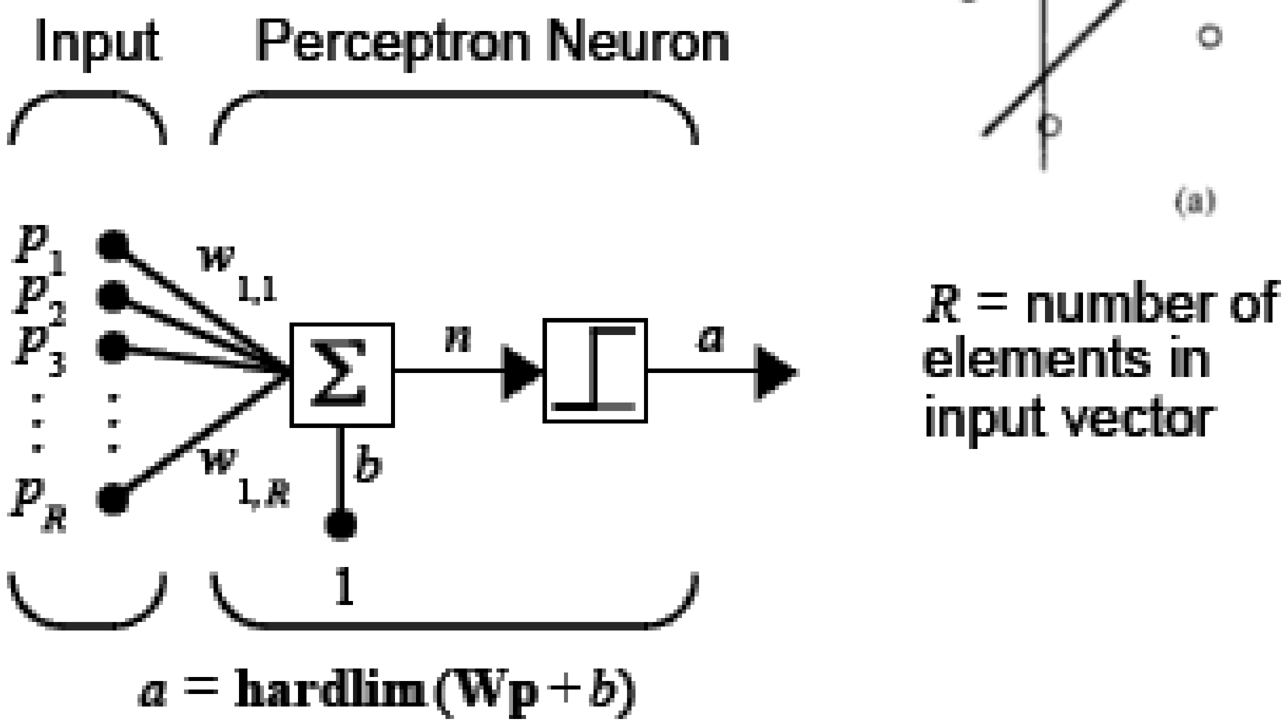


$a = \mathbf{hardlim}(\mathbf{Wp} + b)$ hardlim(n) = 1, if n >= 0; 0 otherwise.

Perceptron Learning Rule [5, 6]

- Learning Process





- $W_{new} = W_{old} + \eta e P$
- $b_{new} = b_{old} + \eta e$, where e = target - actual

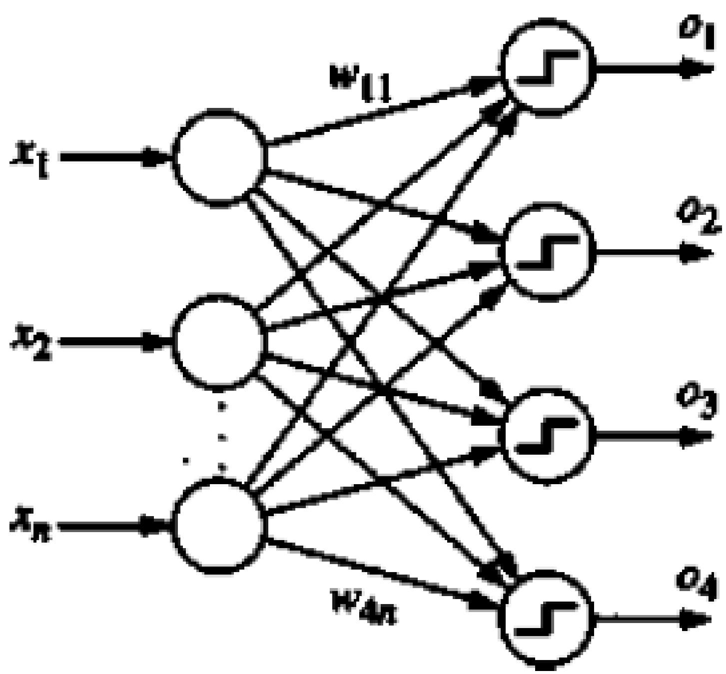
Numerical

- Assume 7 one dimensional input patterns {0.0, 0.17, 0.33, 0.50, 0.67, 0.83, 1.0}. Assume that first four patterns belong to class 0 (with desired output 0) and remaining patterns belong to class 1 (with desired output 1). Design a perceptron to classify these patterns. Use perceptron learning rule. Assume learning rate = 0.1 and initial weight and bias to be (-0.36) and (-0.1) respectively. Show computation for two epochs.

Some Issues

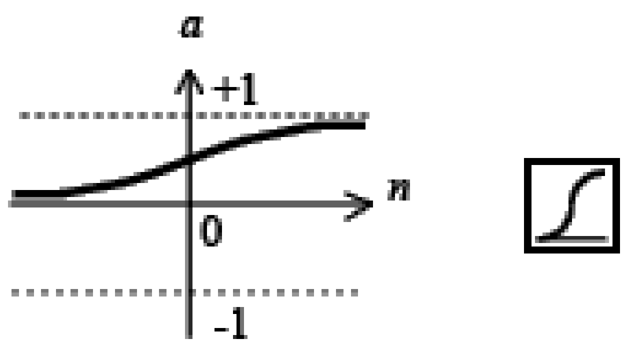
- Why to use bias?
- Termination Criterion
- Learning Rate
- Non-numeric Inputs
- Epoch

Multiclass Discrimination

- Layer of Perceptron
  - To distinguish among n classes, a layer of n perceptrons can be used
  - 

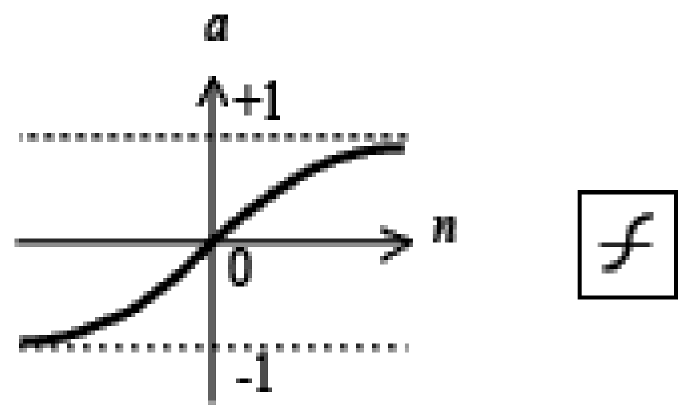
- A presented sample is considered to belong to i^{th} calss only if i^{th} output is 1 and remaining are 0.
- If all outputs are zero, or if more than one output value equals one, the network may be considered to have failed in classification task.

Multilayer Networks – Typical Transfer Functions

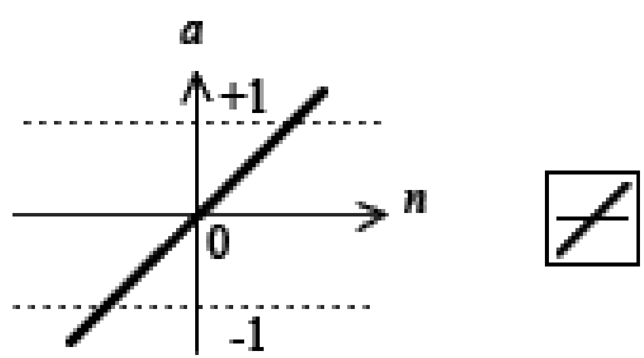


$a = logsig(n)$

Log-Sigmoid Transfer Function



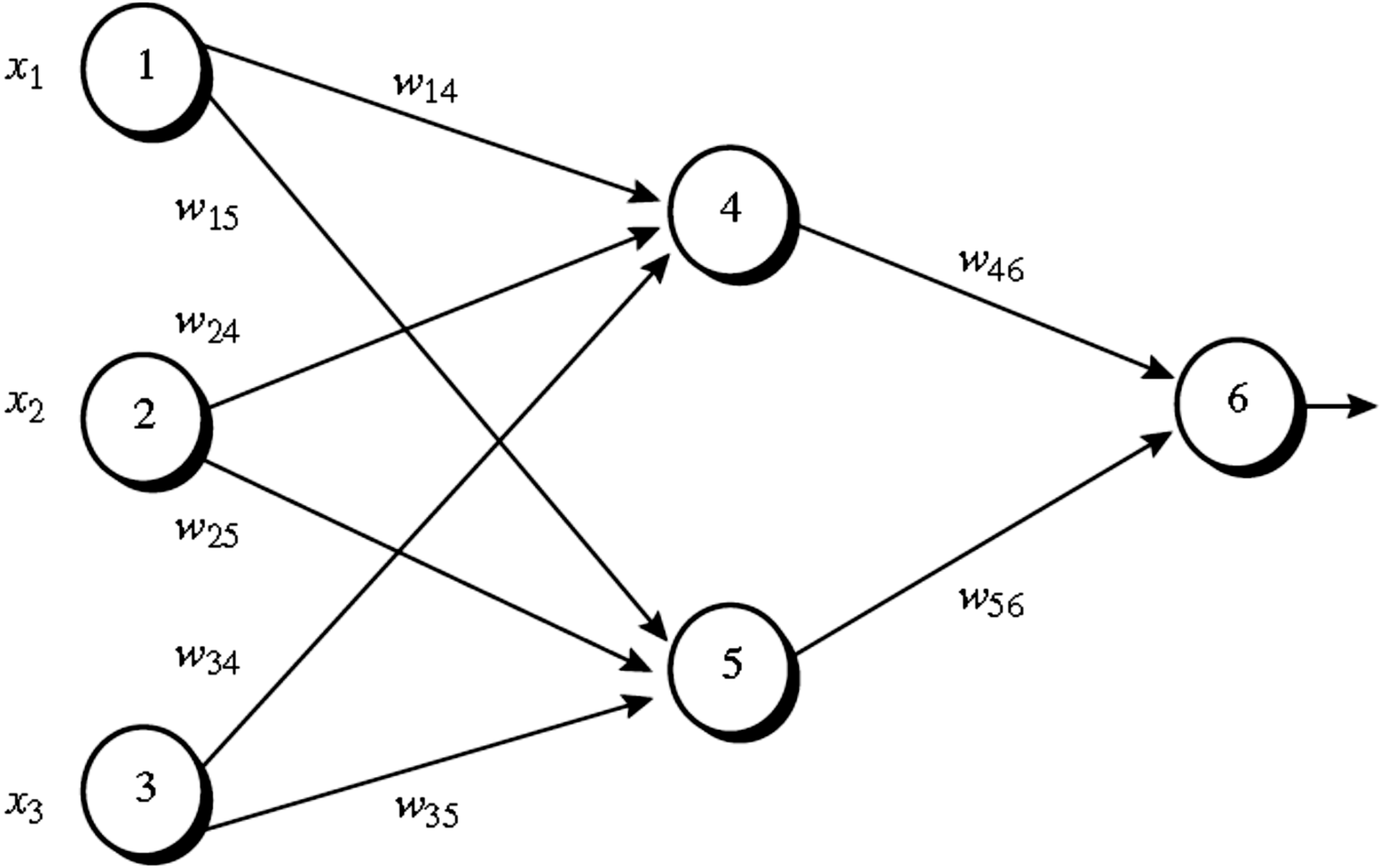
$a = tansig(n)$



$a = purelin(n)$

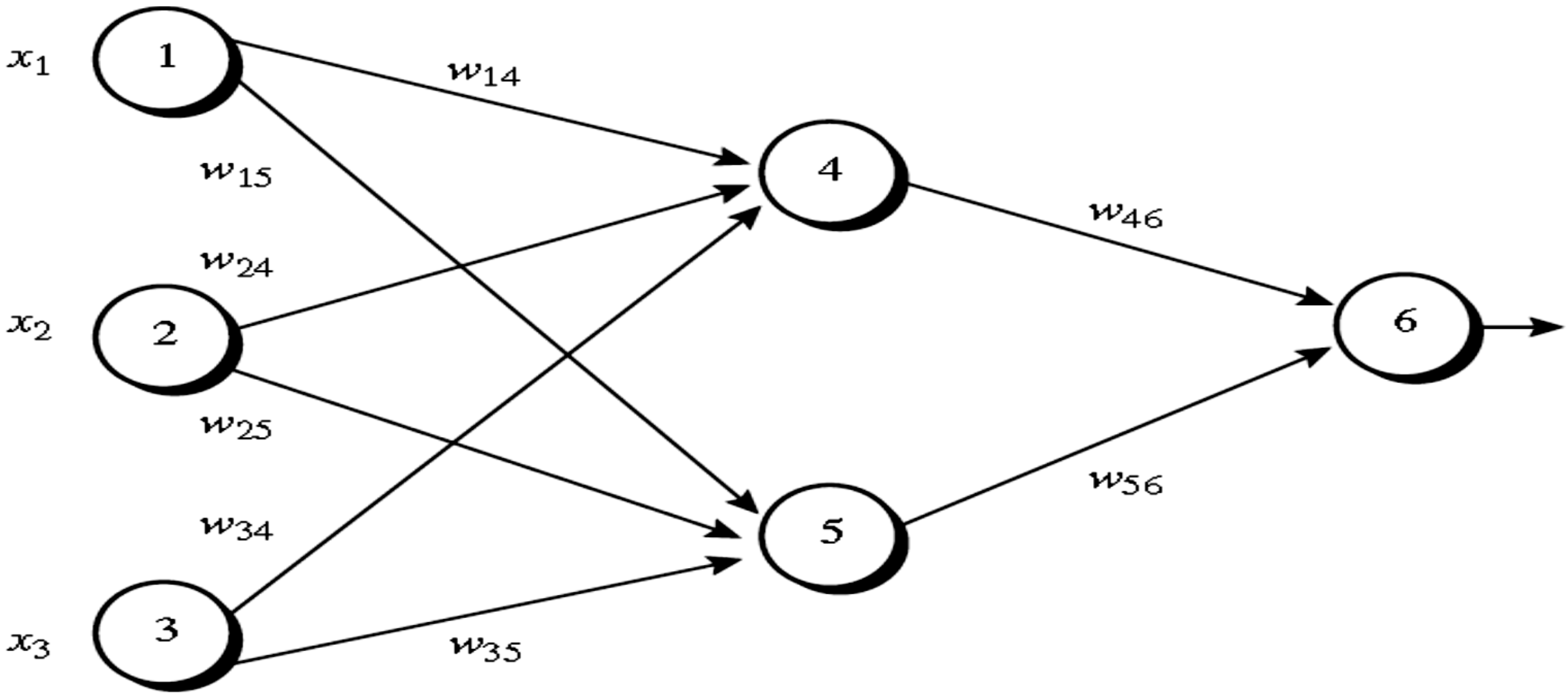
Linear Transfer Function

Example of Backpropagation



An example of a multilayer feed-forward neural network.

Example of Backpropagation



An example of a multilayer feed-forward neural network.

Class Label : 1

Initial input, weight, and bias values.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Example of Backpropagation

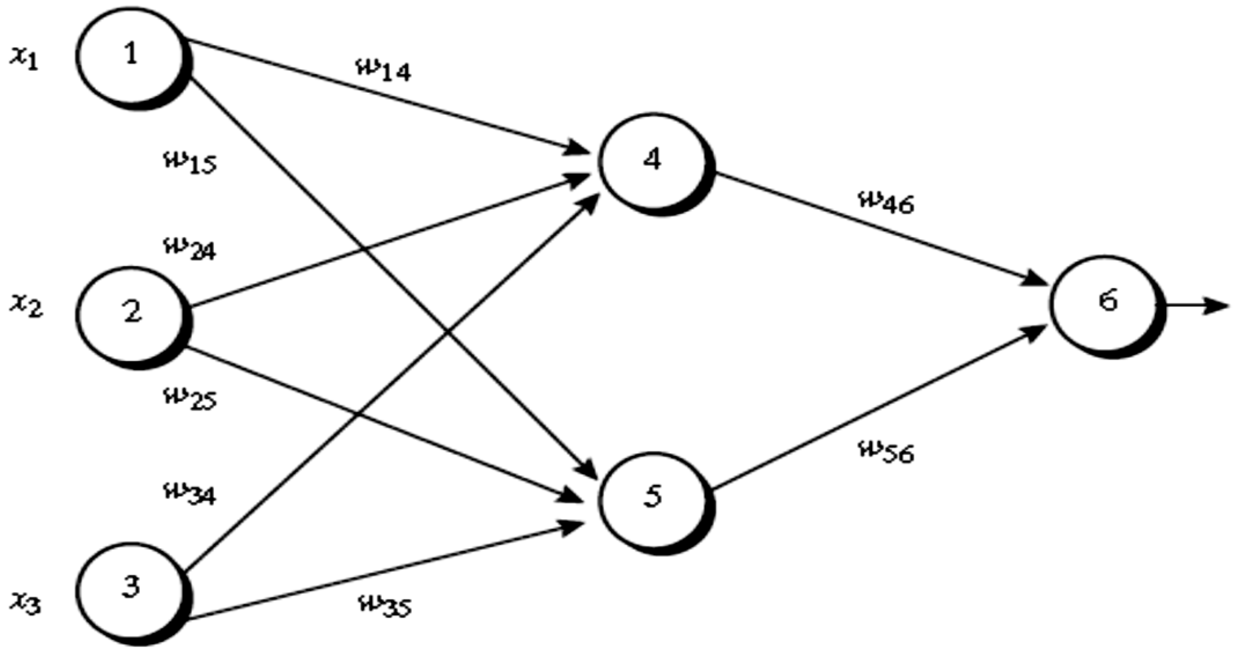


Figure 6.18 An example of a multilayer feed-forward neural network.

Table 6.3 Initial input, weight, and bias values.

Class Label : 1

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Table 6.4 The net input and output calculations.

Unit j	*Net input, I_j*	*Output, O_j*
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

Example of Backpropagation

- Backpropagation

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

$$Err_j = O_j(1-O_j)(T_j - O_j),$$

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk},$$

j	O_j
4	0.332
5	0.525
6	0.474

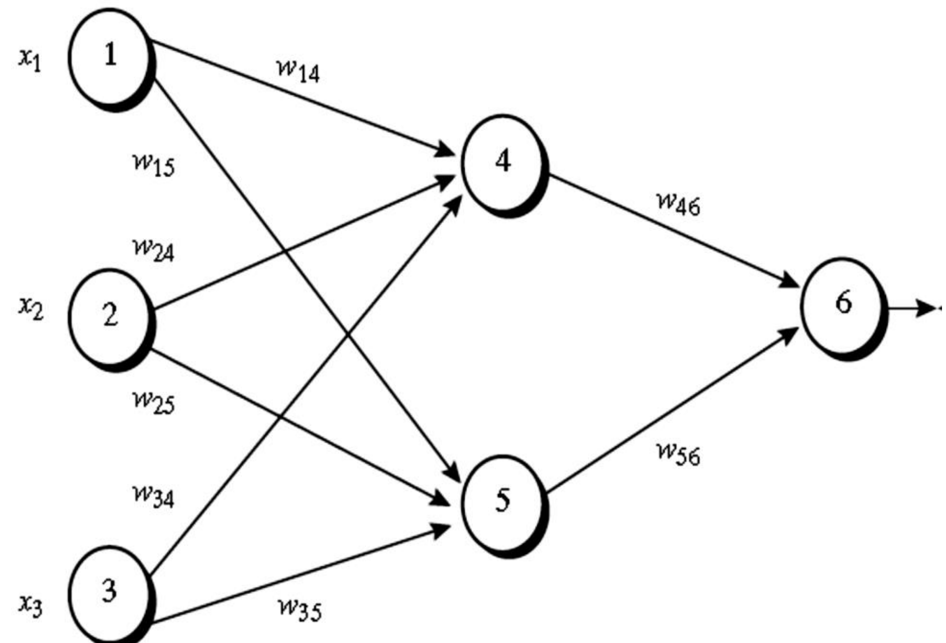


Table 6.5 Calculation of the error at each node.

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474) = 0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

Example of Backpropagation

- Backpropagation

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

$$\Delta w_{ij} = (l) Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

$$\Delta \theta_j = (l) Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

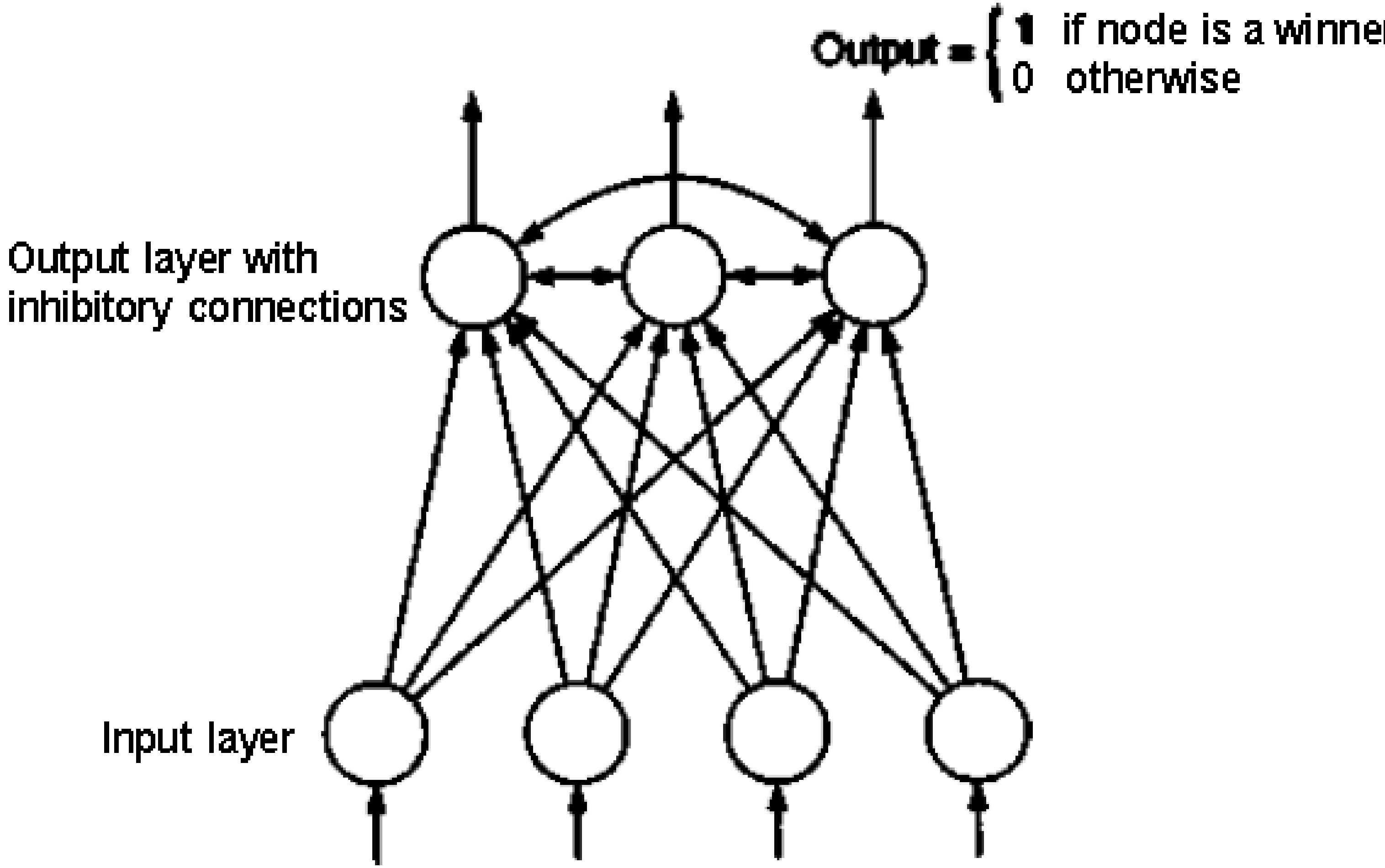
j	O_j	Err_j
4	0.332	-0.0087
5	0.525	-0.0065
6	0.474	0.1311

Table 6.6 Calculations for weight and bias updating.

Weight or bias	*New value*
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$

Unsupervised Learning NN

- Simple Competitive Learning Neural Network



- w_{new} vector = w_{old} vector + η(i/p vector – w_{old} weight vector)

Unsupervised Learning

- Simple Competitive Learning Neural Network
  - Samples: (1.1 1.7 1.8), (0 0 0), (0 0.5 1.5), (1 0 0), (0.5 0.5 0.5), (1 1 1) (Three dimensional inputs)
  - Neurons: A, B, C (in output layer)
  - W_A:0.2 0.7 0.3, W_B:0.1 0.1 0.9, W_C:1 1 1 (Randomly)
  - $\eta = 0.5$

I	Winner	I	Winner
1	C : 1.05, 1.35, 1.4	7	C: 1.05, 1.45, 1.5
2	A: 0.1, 0.35, 0.15	8	A: 0.25, 0.2, 0.15
3	B: 0.05, 0.3, 1.2	9	B: 0, 0.4, 1.35
4	A: 0.55, 0.2, 0.1	10	A: 0.6, 0.1, 0.1
5	A: 0.5, 0.35, 0.3	11	A: 0.55, 0.3, 0.3
6	C: 1, 1.2, 1.2	12	C: 1, 1.2, 1.25

- w_{new} vector = w_{old} vector + η(i/p vector – w_{old} weight vector)

Unsupervised Learning NN

- Simple Competitive Learning Neural Network
- Observations
  - Node a becomes repeatedly activated by the samples i2, i4 and i5, node B by i3 alone and node C by i1 and i6. The centroid of i2, i4 and i5 is (0.5, 0.2, 0.2), and convergence of the weight vector for node A towards this location is indicated by the progression.

  - The high value of a learning rate is causing substantial modification of a node position with every sample presentation. This is one reason for unsmooth convergence.

  - The network is sensitive to the choice of the exact distance function.

Unsupervised Learning NN

- Simple Competitive Learning Neural Network
- Observations
  - The initial value of the node positions also plays some role in determining which node is activated by which samples.

  - The result of network computations also depends on the sequence in which samples are presented to the network, especially when the learning rate is not very small.

Unsupervised Learning NN

- Self Organizing Maps

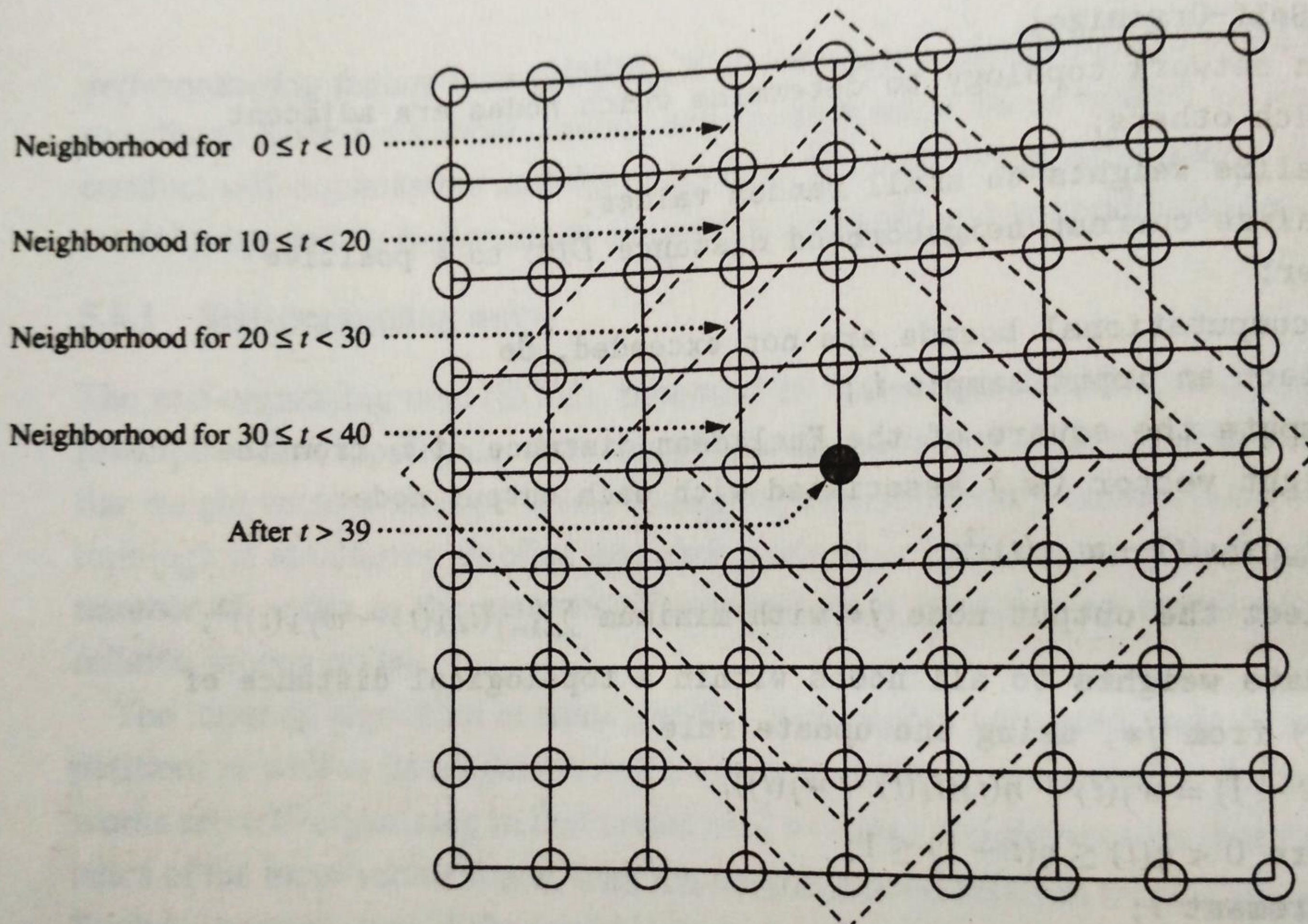
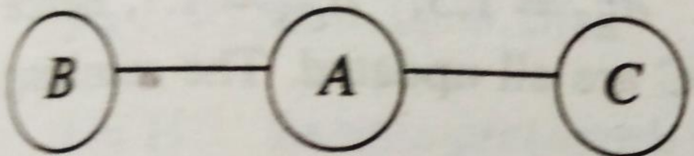


Figure 5.17
Time-varying neighborhoods of a node in an SOM network with grid topology: $D(t) = 4$ for $0 \le t < 10$, $D(t) = 3$ for $10 \le t < 20$, $D(t) = 2$ for $20 \le t < 30$, $D(t) = 1$ for $30 \le t < 40$, and $D(t) = 0$ for $t \ge 40$.

Unsupervised Learning NN

- Self Organizing Maps



Note that this topology is independent of the precise weight vectors chosen initially for the nodes. The training set $T = \{i_1 = (1.1, 1.7, 1.8),\ i_2 = (0, 0, 0), i_3 = (0, 0.5, 1.5),\ i_4 = (1, 0, 0),\ i_5 = (0.5, 0.5, 0.5),\ i_6 = (1, 1, 1)\}$. The initial weight vectors are given by

$$W(0) = \begin{pmatrix} w_A: & 0.2 & 0.7 & 0.3 \\ w_B: & 0.1 & 0.1 & 0.9 \\ w_C: & 1 & 1 & 1 \end{pmatrix}.$$

Let $D(t) = 1$ for one initial round of training set presentations (until $t = 6$), and $D(t) = 0$ thereafter. As in the LVQ example, let $\eta(t) = 0.5$ until $t = 6$, then $\eta(t) = 0.25$ until $t = 12$, and $\eta(t) = 0.1$ thereafter.

Unsupervised Learning NN

- Self Organizing Maps

$t = 1$: Sample presented: $i_1 = (1.1, 1.7, 1.8)$.

Squared Euclidean distance between A and i_1: $d_{A1}^2 = (1.1 - 0.2)^2 + (1.7 - 0.7)^2 + (1.8 - 0.3)^2 = 4.1$. Similarly, $d_{B1}^2 = 4.4$ and $d_{C1}^2 = 1.1$.

C is the "winner" since $d_{C1}^2 < d_{A1}^2$ and $d_{C1}^2 < d_{B1}^2$.

Since $D(t) = 1$ at this stage, the weights of both C and its neighbor A are updated according to equation 5.2. For example,

$$w_{A,1}(1) = w_{A,1}(0) + \eta(1) \cdot (x_{1,1} - w_{A,1}(0)) = 0.2 + (0.5)(1.1 - 0.2) = 0.65$$

The resulting weight matrix is

$$W(1) = \begin{pmatrix} w_A: & 0.65 & 1.2 & 1.05 \\ w_B: & 0.1 & 0.1 & 0.9 \\ w_C: & 1.05 & 1.35 & 1.4 \end{pmatrix}.$$

Note that the weights attached to B are not modified since B falls outside the neighborhood of the winner node C.

Unsupervised Learning NN

➢ Self Organizing Maps

$t = 2$: Sample presented: $i_2 = (0, 0, 0)$. $d^2_{A2} = 3$, $d^2_{B2} = 0.8$, $d^2_{C2} = 4.9$, hence B is the winner. The weights of both B and its neighbor A are updated. The resulting weight matrix is

$$W(2) = \begin{pmatrix} \mathbf{w}_A: & 0.325 & 0.6 & 0.525 \\ \mathbf{w}_B: & 0.05 & 0.05 & 0.45 \\ \mathbf{w}_C: & 1.05 & 1.35 & 1.4 \end{pmatrix}.$$

$t = 3$: Sample presented: $i_3 = (0, 0.5, 1.5)$. $d^2_{A3} = 1.1$, $d^2_{B3} = 1.3$, $d^2_{C3} = 1.7$, hence A is the winner. The weights of A and its neighbors B, C are all updated. The resulting weight matrix is

$$W(3) = \begin{pmatrix} \mathbf{w}_A: & 0.16 & 0.55 & 1.01 \\ \mathbf{w}_B: & 0.025 & 0.275 & 0.975 \\ \mathbf{w}_C: & 0.525 & 0.925 & 1.45 \end{pmatrix}.$$

Unsupervised Learning NN

- Self Organizing Maps

$t = 4$: Sample presented: $\boldsymbol{i}_4 = (1, 0, 0)$. $d^2_{A4} = 2$, $d^2_{B4} = 1.9$, $d^2_{C4} = 3.2$, hence B is the winner; both B and A are updated.

$$W(4) = \begin{pmatrix} \boldsymbol{w}_A: & 0.58 & 0.275 & 0.51 \\ \boldsymbol{w}_B: & 0.51 & 0.14 & 0.49 \\ \boldsymbol{w}_C: & 0.525 & 0.925 & 1.45 \end{pmatrix}.$$

$t = 5$: Sample presented: $\boldsymbol{i}_5$. A is the winner. All nodes are updated.

$$W(5) = \begin{pmatrix} \boldsymbol{w}_A: & 0.54 & 0.39 & 0.50 \\ \boldsymbol{w}_B: & 0.51 & 0.32 & 0.49 \\ \boldsymbol{w}_C: & 0.51 & 0.71 & 0.975 \end{pmatrix}.$$

$t = 6$: Sample presented: $\boldsymbol{i}_6$. C is the winner; both $\boldsymbol{w}_C$ and $\boldsymbol{w}_A$ are updated.

$$W(6) = \begin{pmatrix} \boldsymbol{w}_A: & 0.77 & 0.69 & 0.75 \\ \boldsymbol{w}_B: & 0.51 & 0.32 & 0.49 \\ \boldsymbol{w}_C: & 0.76 & 0.86 & 0.99 \end{pmatrix}.$$

Unsupervised Learning NN

- Self Organizing Maps

$t = 7$: η is now reduced to 0.25, and the neighborhood relation shrinks, so that only the winner node is updated henceforth. Sample presented: i_1. C is the winner; only w_C is updated.

w_C : (0.84 1.07 1.19).

$t = 8$: Sample presented: i_2. B is winner and w_B is updated.

w_B : (0.38 0.24 0.37).

$t = 9$: Sample presented: i_3. C is winner and w_C is updated.

w_C : (0.63 0.93 1.27).

$t = 10$: Sample presented: i_4. B is winner and w_B is updated.

w_B : (0.53 0.18 0.28).

$t = 11$: Sample presented: i_5. B is winner and w_B is updated.

w_B : (0.53 0.26 0.33).

$t = 12$: Sample presented: i_6. Weights of the winner node A are updated.

w_A : (0.83 0.77 0.81).

Unsupervised Learning NN

➢ Self Organizing Maps

$t = 13$: Now η is further reduced to 0.1.

Sample presented: i_1. Weights of the winner node C are updated.

w_C : (0.68 1.00 1.32).

$t = 14$: Sample presented: i_2. Weights of the winner node B are updated.

w_B : (0.47 0.23 0.30).

$t = 15$: Sample presented: i_3. Winner is C and w_C is updated. At this stage, the weight matrix is given by

$$W(15) = \begin{pmatrix} w_A: & 0.83 & 0.77 & 0.81 \\ w_B: & 0.47 & 0.23 & 0.30 \\ w_C: & 0.61 & 0.95 & 1.34 \end{pmatrix}.$$

Unsupervised Learning NN

- Self Organizing Maps

At the beginning of the training process, the Euclidean distances between various nodes were given by

$$|\mathbf{w}_A - \mathbf{w}_B| = 0.85, \ |\mathbf{w}_B - \mathbf{w}_C| = 1.28, \ |\mathbf{w}_A - \mathbf{w}_C| = 1.22.$$

The training process increases the relative distance between non-adjacent nodes (B, C), while the weight vector associated with A remains roughly in between B and C, with

$$|\mathbf{w}_A - \mathbf{w}_B| = 1.28, \ |\mathbf{w}_B - \mathbf{w}_C| = 1.75, \ |\mathbf{w}_A - \mathbf{w}_C| = 0.80.$$

The above example illustrates the extent to which nodes can move during the learning process, and how samples switch allegiance between nodes, especially in the early phases of computation. Computation continues in this manner until the network stabilizes, i.e., the same nodes continue to be the winners for the same input patterns, with the possible exception of input patterns equidistant from two weight vectors.

Classical Partitioning Methods

- The k-means Method

Algorithm: ***k*-means.** The k-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- k: the number of clusters,
- D: a data set containing n objects.

Output: A set of k clusters.

Method:

(1) arbitrarily choose k objects from D as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, i.e., calculate the mean value of the objects for each cluster;
(5) **until** no change;

Figure 7.2 The k-means partitioning algorithm.

Classical Partitioning Methods

➢ The k-means Method

Suppose that the data mining task is to cluster the following eight points (with (x, y) representing location) into three clusters:

$$A_1(2,10), A_2(2,5), A_3(8,4), B_1(5,8), B_2(7,5), B_3(6,4), C_1(1,2), C_2(4,9).$$

The distance function is Euclidean distance. Suppose initially we assign A_1, B_1, and C_1 as the center of each cluster, respectively. Use the *k-means* algorithm to show *only*

(a) The three cluster centers after the first round execution

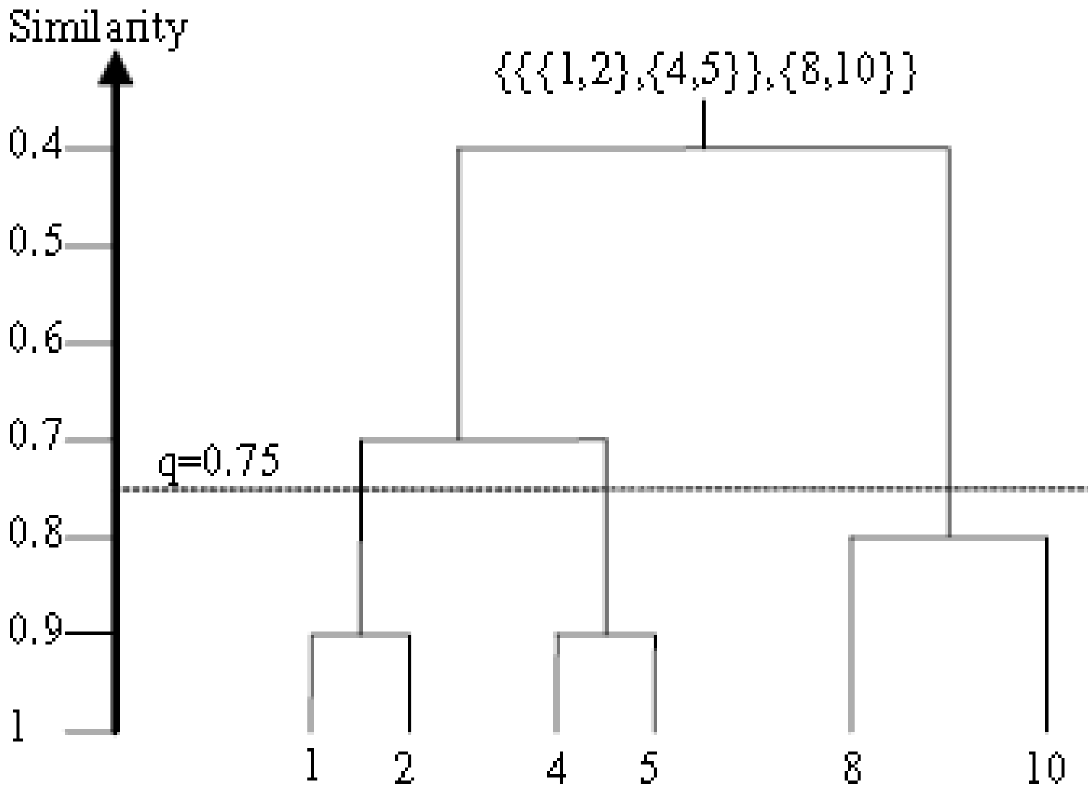
(b) The final three clusters

Classical Partitioning Methods

- The k-medoids Method

Hierarchical Methods

- Hierarchical Agglomerative Clustering



Hierarchical Methods

- Hierarchical Agglomerative Clustering

1. $G \leftarrow \{\{d\} | d \in S\}$ (initialize G with singleton clusters, each containing a document from S).
2. If $|G| \leq k$, then exit (stop if the desired number of clusters is reached).
3. Find $S_i, S_j \in G$ such that $(i, j) = \arg\max_{(i,j)} \text{sim}(S_i, S_j)$ (find the two closest clusters).
4. If $\text{sim}(S_i, S_j) < q$, exit (stop if the similarity of the closest clusters is less than q).
5. Remove S_i and S_j from G.
6. $G = G \cup \{S_i, S_j\}$ (merge S_i and S_j, and add the new cluster to the hierarchy).
7. Go to step 2.

References

- Data mining: concepts and techniques, J. Han, and M. Kamber. Morgan Kaufmann, (2006)

- Elements of Artificial Neural Networks, Kishan Mehrotra, Chilukuri K. Mohan, Sanjay Ranka. MIT Press, (1997)

- Matlab Neural Network Tollbox Documentation

Disclaimer

- These slides are not original and have been prepared from various sources for teaching purpose.