Computational Intelligence:

Computational intelligence (CI) is a set of Nature inspired computational methodologies and approaches to address complex problems of the real world applications to which traditional methodologies and approaches are ineffective or infeasible. It primarily includes Fuzzy logic systems, Neural Networks and Evolutionary Computation. In addition, CI also embraces techniques that stem from the above three or gravitate around one or more of them, such as Swarm intelligence and Artificial immune systems which can be seen as a part of Evolutionary Computation.

Evolutionary Computation

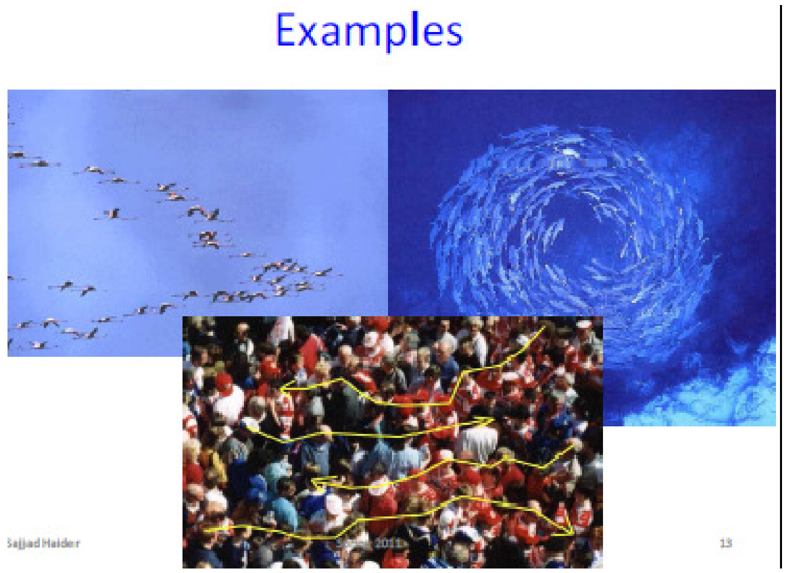
In computer science, evolutionary computation is a subfield of artificial intelligence (more particularly computational intelligence) that involves combinatorial
optimization problems. Evolutionary techniques mostly involve metaheuristic optimization algorithms such as:
Evolutionary algorithms (comprising genetic algorithms, evolutionary programming, evolution strategy and Genetic programming) Swarm intelligence (comprising ant colony optimization and particle swarm optimization and in a lesser extent Artificial immune systems, Cultural
algorithms, Differential evolution, Harmony search algorithm, etc.

In artificial intelligence, an evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm. An EA uses some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in population, and the fitness function determines the environment within which the solutions "live" (see also cost function). Evolution of the population then takes place after the repeated application of the
above operators.

Swarm Intelligence

There are two popular swarm inspired methods in computational intelligence areas:
– Ant colony optimization (ACO)

– Particle swarm optimization (PSO)
ACO was inspired by the behaviors of ants and has many successful applications in discrete optimization problems. The particle swarm concept originated as a simulation of
simplified social system. The original intent was to graphically simulate the choreography of bird of a bird block or fish school. However, it was found that particle swarm model can be used as optimizer



Artificial Neural Networks

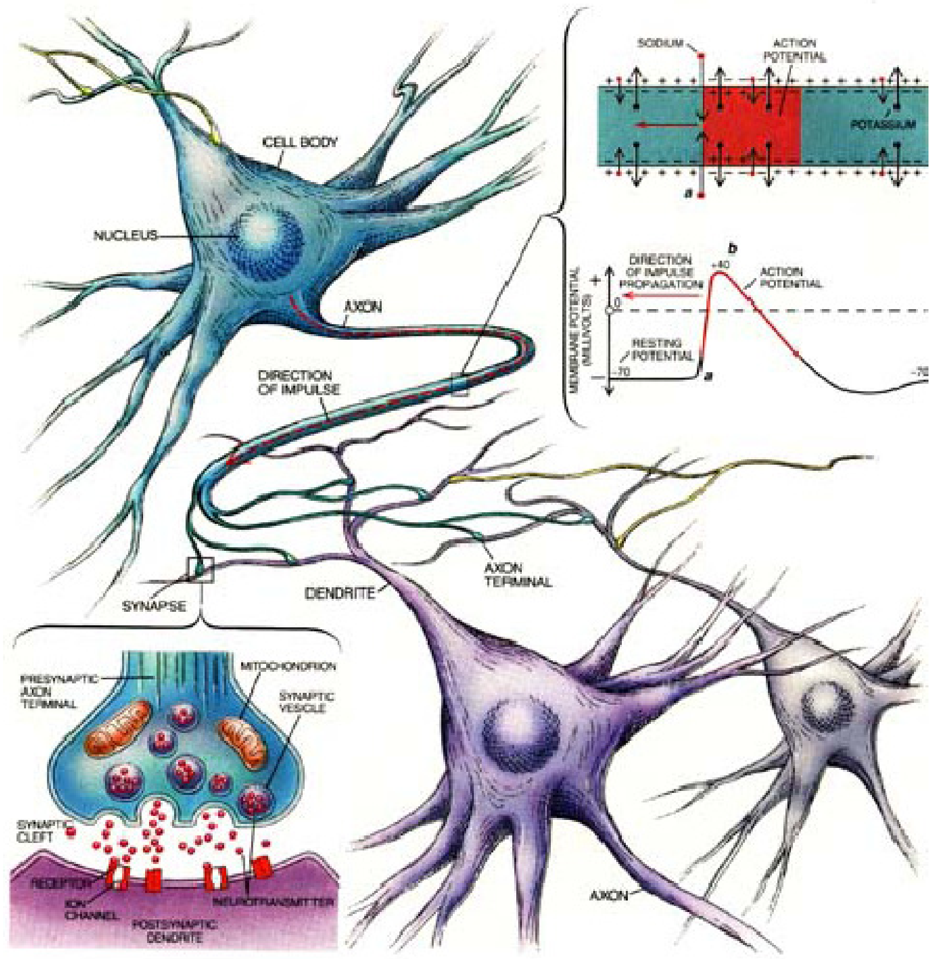
Neural networks are biologically motivated computing structures that are conceptually modeled after the brain. The neural network is made up of a highly connected network of individual computing elements (mimicking neurons) that collectively can be used to solve interesting and difficult problems.

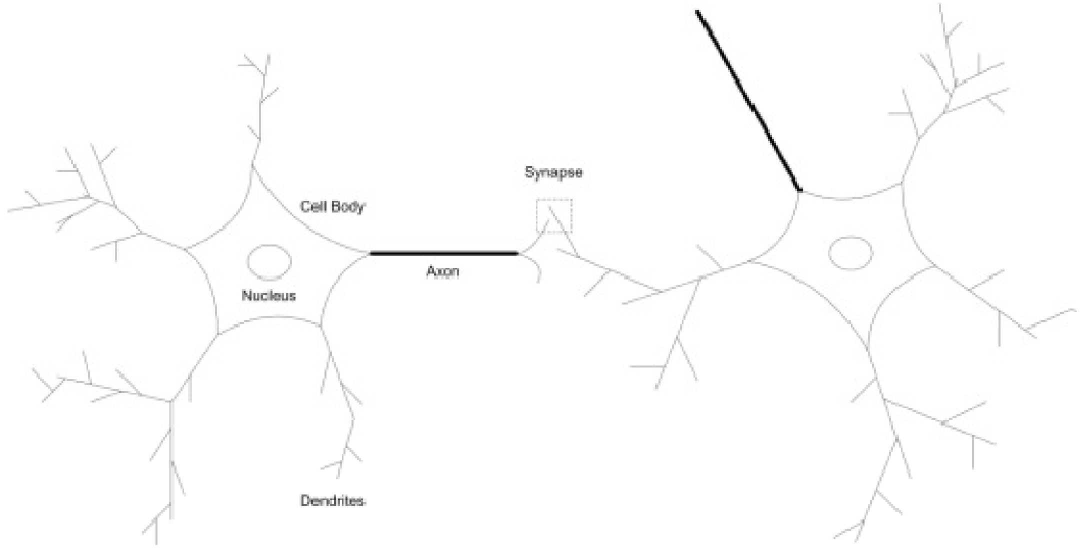
How Our Brain Works?

While neural networks are modeled after our understanding of the way in which our brain works, surprisingly little is known about how our brains actually function.

Through various types of inspection, we can see our brain in operation, but because of the massive number of neurons and interconnections between these neurons, how it works remains a mystery (though many theories exist).

Neurons Inside Our Body


SODIUM
ACTION POTENTIAL
POTASSIUM
a
CELL BODY
NUCLEUS
AXON
DIRECTION OF IMPULSE
DIRECTION OF IMPULSE PROPAGATION
b
+40
ACTION POTENTIAL
MEMBRANE POTENTIAL (MILLIVOLTS)
+
0
RESTING POTENTIAL
-70
a
-70
SYNAPSE
DENDRITE
AXON TERMINAL
PRESYNAPTIC AXON TERMINAL
MITOCHONDRION
SYNAPTIC VESICLE
SYNAPTIC CLEFT
RECEPTOR
ION CHANNEL
NEUROTRANSMITTER
POSTSYNAPTIC DENDRITE
AXON


Synapse
Cell Body
Nucleus
Axon
Dendrites

We are born with about 100 billion neurons. A neuron may connect to as many as 100,000 other neurons. Signals “move” via electrochemical signals. The synapses release a chemical transmitter –the sum of which can cause a threshold to be reached –causing the neuron to “fire”

Definition of Neurons (from Wikipedia)

•Neurons are responsive cells in the nervous system that process and transmit information by chemical signals within the neuron.

•A number of different types of neurons exist: sensory neurons respond to touch, sound, light and numerous other stimuli affecting cells of the sensory organs that then send signals to the spinal cord and brain.

•Motor neurons receive signals from the brain and spinal cord and cause muscle contractions and affect glands.

•Inter-neurons connect neurons to other neurons within the brain and spinal cord.

•Neurons respond to stimulus and communicate the presence of that stimuli to the central nervous system, which processes that information and sends responses to other parts of the body for action.

Birth of Artificial Neural Networks

•The story of neural networks is interesting because, like AI itself, it’s one of grand visions, eventual disappointment, and finally, silent adoption.

•In 1943, McCulloch and Pitts developed a neural network model based on their understanding of neurology, but the models were typically limited to formal logic simulations (simulating binary operations). McCulloch and Pitts (1943) proposed a model of a neuron , McCulloch-Pitts Model. Five physical assumptions:

1. The activity of a neuron is an all or none process.
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron. The only significant delay within the nervous system is synaptic delay.

3. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time.
4. The structure of the net does not change with time

Rosenblatt (1958) introduced a new approach to the pattern recognition problem; *perceptron which could classify patterns by modifying its connections. Attracted attention of engineers and physicists, using model of biological vision.* Widrow and Hoff (1960) introduced the LMS algorithm formulated the *Adaline*(Adaptive linear element). Minsky (1961)written a paper contains a large section on what is now termed neural networks.

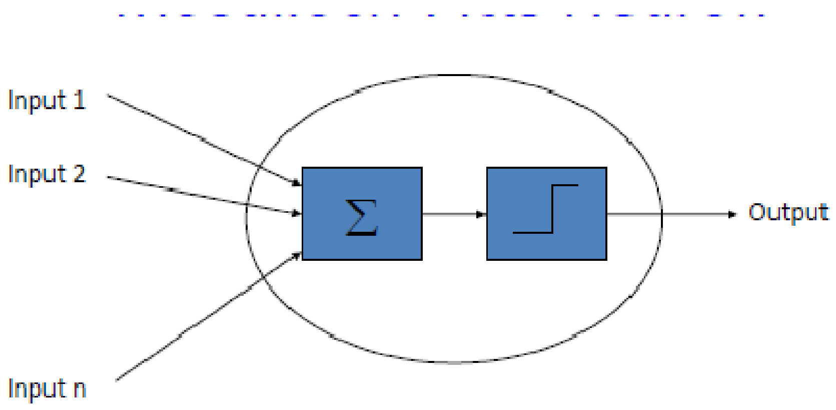
Widrow (1962) proposed Madaline (Multiple - adaline) with his students.
Minsky and Papert (1969) demonstrated fundamental limits on what one-layer perceptrons can compute exclusive-OR problem. In 1979 “decade of dormancy” for neural networks from physics and engineering, perspective. Grossberg (1980) established a principle of self-organization, called adaptive Resonance provides the basis of ART (Adaptive Resonance Theory).

Hopfield (1982) used the idea of energy function to formulate the computation performed by recurrent networks dynamical stable networks. Kirkpatrick, Galatt, Vecchi (1983) described a simulated annealing method Boltzmann learning: stochastic learning algorithm. Barto, Sutton, Anderson (1983) proposed reinforcement learning. Rumelhart, Hinton, Williams (1986) reported the *back-propagation algorithm.* Broomhead, Lowe (1988) proposed the *radial basis function* (RBF) network.

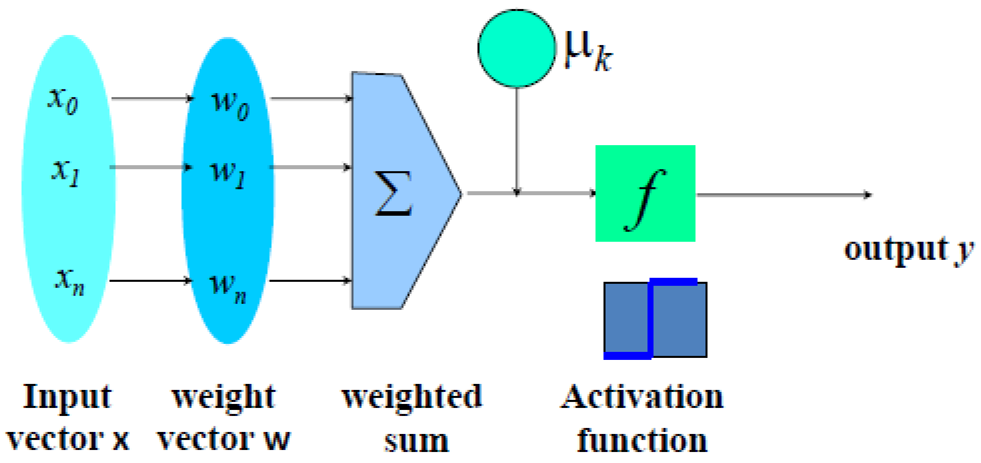
Artificial Neuron

An artificial neuron is an information-processing unit that is fundamental to the operation of an ANN. It consists of three basic elements:
1. A set of connecting links from different inputs, each of which is characterized by a weight or strength. In general, the weights of an artificial neuron may lie in a range that includes negative as well as positive values.
2. An adder for summing the input signals weighted by the respective synaptic strengths.
3. An activation function for limiting the amplitude of the output of a neuron.

McCulloch-Pitts Neuron



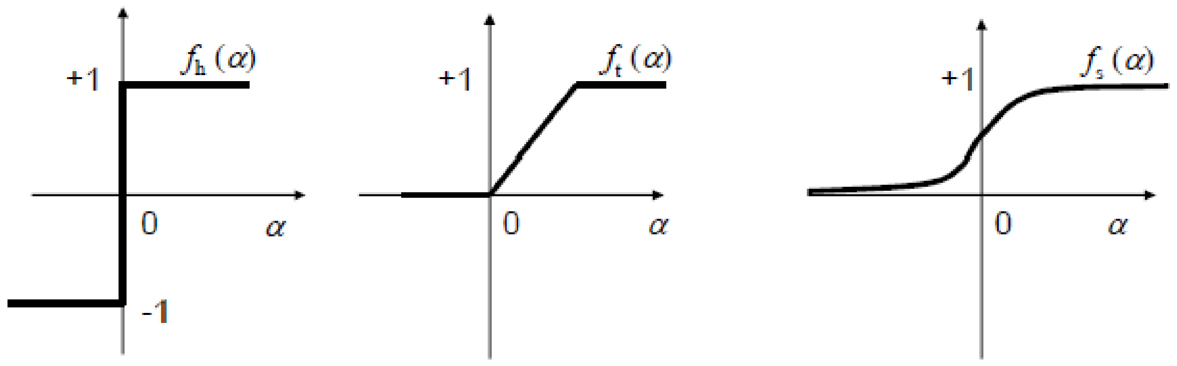
A set of synapses (connections) brings in activations from other neurons.
A processing unit sums the inputs, and then applies a non-linear activation function.
An output line transmits the results to other neurons.



The *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping.

Activation functions

- Types of nonlinear activation functions



Hard limiter function

Threshold logic function

Sigmoid function

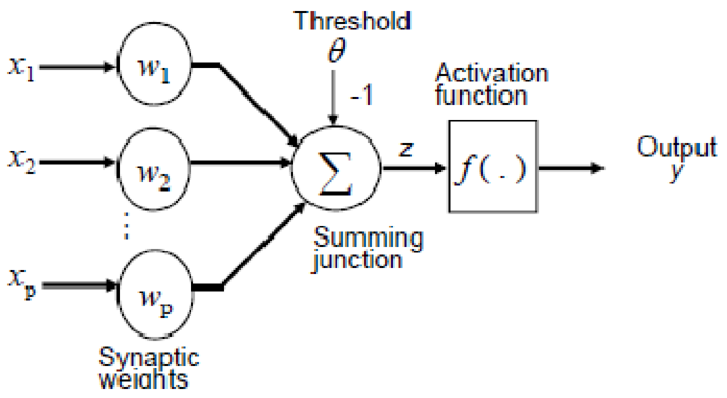
The Adaptive Linear Combiner (ALC) formed part of the two earliest ANNs : the Perceptron and the ADALINE

Perceptron : consists of a single neuron with adjustable synaptic weights

Developed by F. Rosenblatt (1958)

Used for the classification of a special type of patterns said to be linearly separable

Hard-limiting activation function is used



Linear separability of perceptron

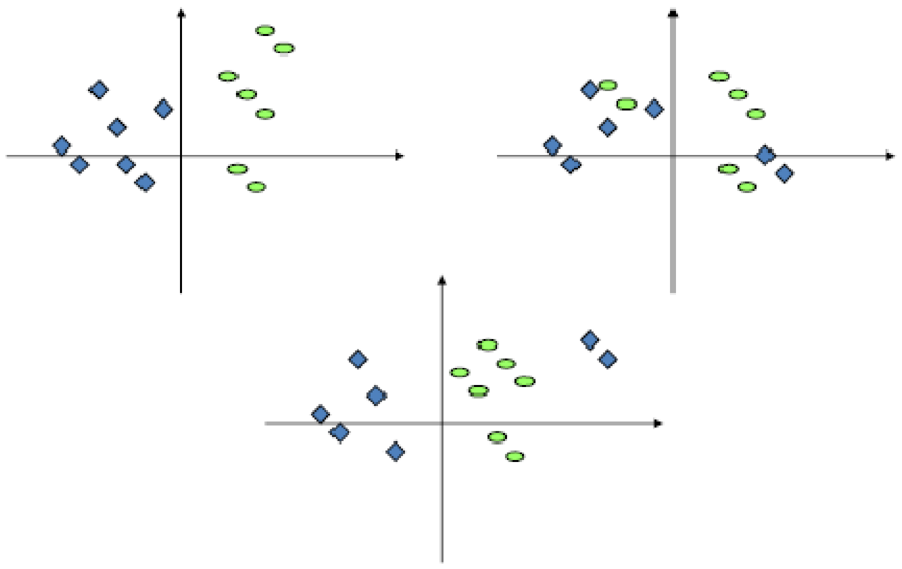
- From the model, we thus find that the linear combiner output (i.e hard limiter input) is :

$$z = \sum_{i=1}^{p} w_i x_i - \theta$$

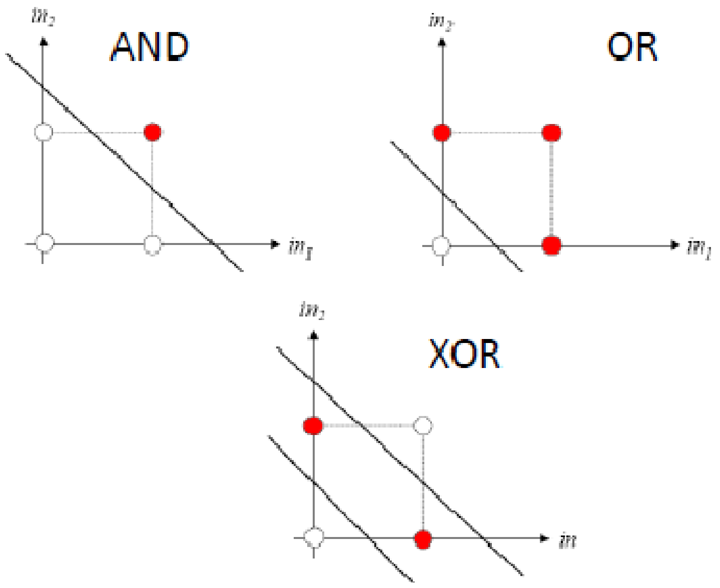
- Decision regions in the case of elementary perceptron; two decision regions separated by a hyperplane defined by

$$\sum_{i=1}^{p} w_i x_i - \theta = 0$$

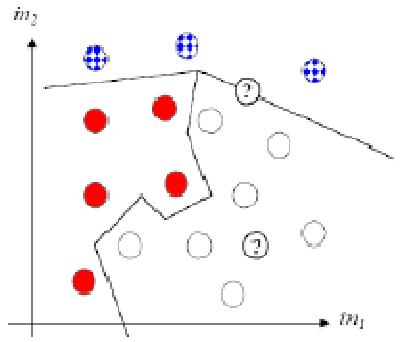
- By expanding the output (computation) layer of the perceptron tc include more than one neuron, we may correspondingly form classification with more than two classes.

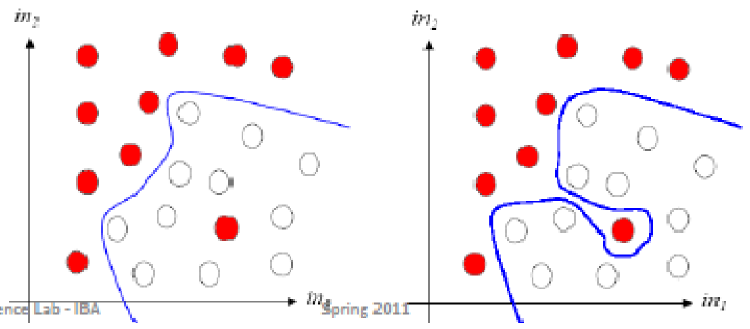


AND, OR, XOR



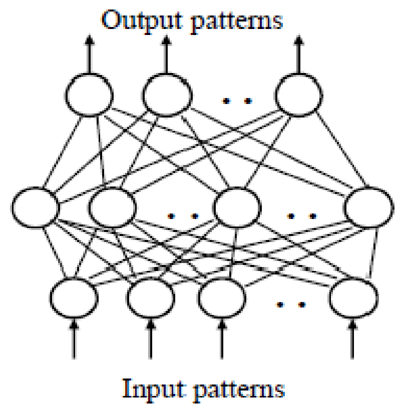
Decision Boundaries





Multilayer perceptrons

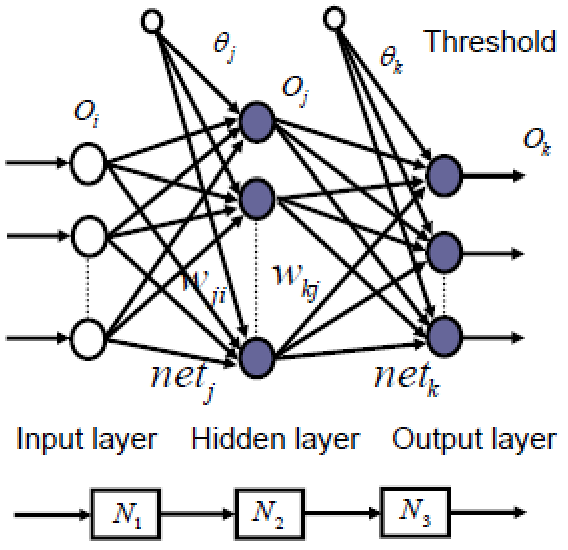
- Features
  - feed forward neural networks
  - several layers
  - no inter-connection links in same layer
  - connection only in the neighboring layer
  - Error backpropagation learning



The model of each neuron in the network includes a non-linearity at the output end.

The network contains one or more layers of hidden neurons.

The network exhibits a high degree of connectivity.



● Nonlinear neuron

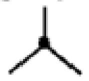
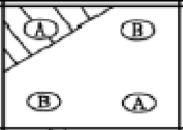
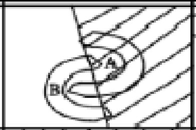
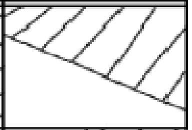
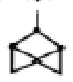
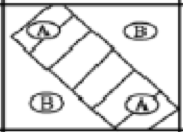
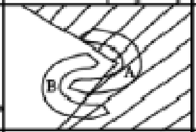
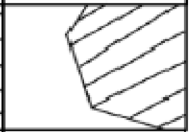
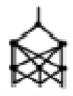
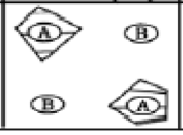
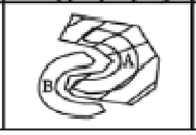
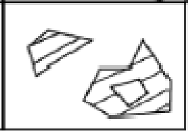
○ Linear neuron

$$net_j = \sum_i w_{ji} o_i$$

$$o_j = f_j(net_j)$$

f_j : non – decreasing and differenti able

The number of layers and decision regions

N layers	Complexity	Exclusive -or	Classes with Meshed Regions	Most General Region Shapes
Single-layer j i 	Half Plane Bounded by Hyperplane			
Two-layer k j i 	Convex Open or Closed Regions			
Three-layer l k j i 	Arbitrary (Complexity limited by Number of Units)			

Multilayer Error Correction Adaptation

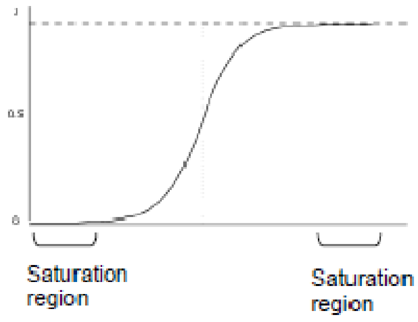
•Credit assignment problem solved by using non-decreasing and continuously differentiable activation function for PEs

•In our example, we use linear output PEs and nonlinear hidden PEs with sigmoid activations:

$$f(x) = \frac{1}{1+e^{-x}}$$

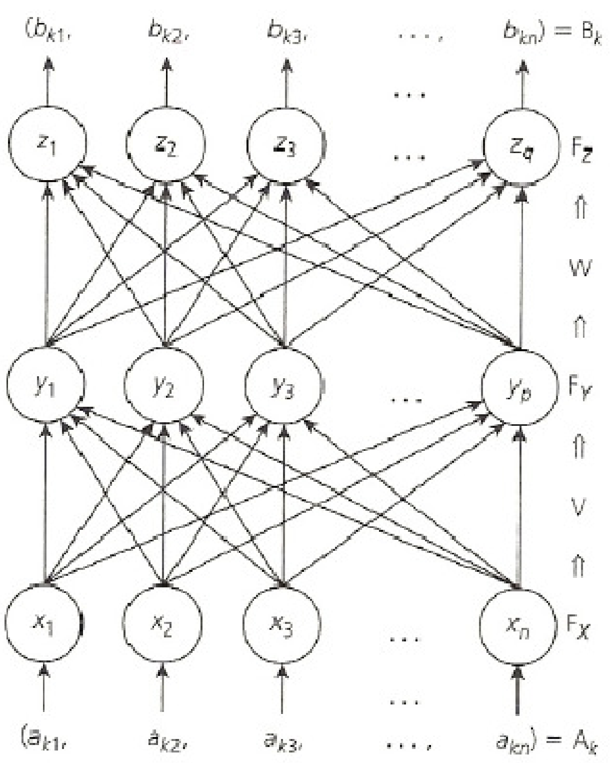
$$f'(x) = -\frac{1}{(1+e^{-x})^2}\cdot(1+e^{-x})' = -\frac{1}{(1+e^{-x})^2}\cdot(-e^{-x})$$

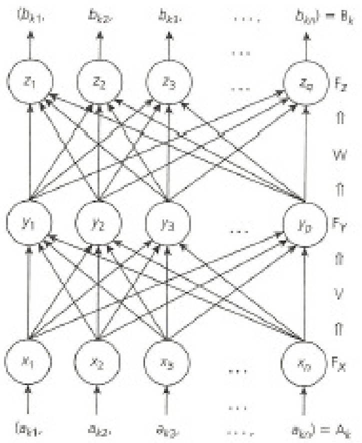
$$= \frac{1}{1+e^{-x}}\cdot\frac{e^{-x}}{1+e^{-x}} \quad = f(x)(1-f(x))$$



•We will also use chain rule of differentiation

$$\text{if } z = f(y), y = g(x), x = h(t) \text{ then} \frac{dz}{dt} = \frac{dz}{dy}\cdot\frac{dy}{dx}\cdot\frac{dx}{dt} = f'(y)g'(x)h'(t)$$





Minimize cost (error) function :

$$E = 0.5\sum_{k=1}^{m}\sum_{j=1}^{q}\left(b_{kj} - z_{kj}\right)^2$$

Summed over neurons and patterns

For linear output neuron,

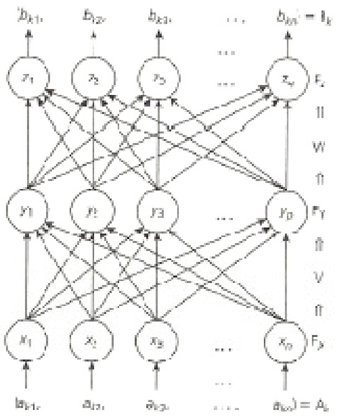
$$z_{kj} = \sum_{i=1}^{p} y_{ki}w_{ji} = f_l\left(r_{kj}\right), \quad where\ r_{kj} = \sum_{i=1}^{p} y_{ki}w_{ji}$$

For nonlinear hidden PE,

$$y_{ki} = f_n\left(\underset{\nwarrow \text{inputs}}{\sum_{h=1}^{n}} a_{kh}v_{ih}\right) = f_n(r_{ki}), \quad where\ r_{ki} = \sum_{h=1}^{n} a_{kh}v_{ih}$$

$$f_n(r_{ki}) = \frac{1}{1+e^{-r_{ki}}}$$

- Move in the direction opposite to error (cost) gradient to a minimum
- Output weights Fy to Fz are adjusted using chain rule of differentiation:
- (Pattern – *k*, *j* – index of neuron in the output layer)

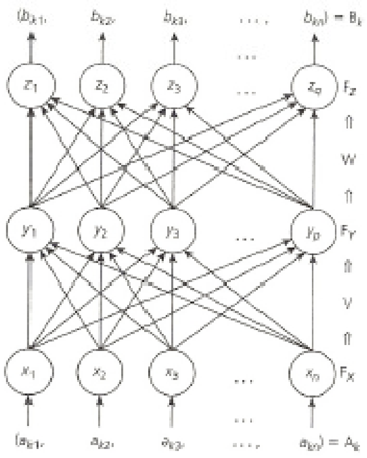


b_{k1}, b_{k2}, b_{k3}, ..., b_{kn}) = B_k
z_1 z_2 z_3 ... z_q
F_Z
W
y_1 y_2 y_3 ... y_p
F_Y
V
x_1 x_2 x_3 ... x_n
F_X
(a_{k1}, a_{k2}, a_{k3}, ..., a_{kn}) = A_k

$$\frac{\partial E_{kj}}{\partial w_{ji}} = \frac{\partial E_{kj}}{\partial z_{kj}} \frac{\partial z_{kj}}{\partial w_{ji}}$$

$$\frac{\partial E_{kj}}{\partial w_{ji}} = \frac{\partial}{\partial z_{kj}}\left[\frac{1}{2}\left(b_{kj} - z_{kj}\right)^2\right] \frac{\partial}{\partial w_{ji}}\left[\sum_{i=1}^{p} w_{ji} y_{ki}\right]$$

$$= -(b_{kj} - z_{kj}) \sum_{i=1}^{p} y_{ki} = -\delta_{kj} \sum_{i=1}^{p} y_{ki}$$



(b_{k1}, b_{k2}, b_{k3}, ..., b_{kn}) = B_k
z_1 z_2 z_3 ... z_q
F_Z
W
y_1 y_2 y_3 ... y_p
F_Y
V
x_1 x_2 x_3 ... x_n
F_X
(a_{k1}, a_{k2}, a_{k3}, ..., a_{kn}) = A_k

Next step: adjust weights **V** between input and hidden layers

Define error assigned to a hidden neuron, where r_{ki} is the net input to the hidden neuron

$$\delta_{ki} \equiv -\partial E_k / r_{ki}$$

$$\frac{\partial E_k}{\partial v_{ih}} = \frac{\partial E_k}{\partial r_{ki}} \frac{\partial r_{ki}}{\partial v_{ih}} = -\delta_{ki} a_{kh}$$

The key question is how to compute the δ_{ki} values for hidden neurons

But $\partial y_{ki} / \partial r_{ki} = f_n'\left(r_{ki}\right)$ is the derivative of the sigmoid activation function.

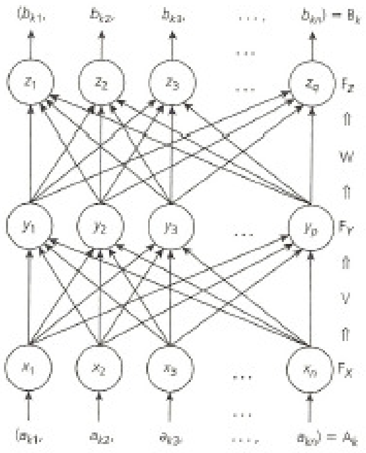
Now use the chain rule twice:

$$\delta_{ki} = -\frac{\partial E_k}{\partial r_{ki}} = -\frac{\partial E_k}{\partial y_{ki}} \frac{\partial y_{ki}}{\partial r_{ki}} = -\frac{\partial E_k}{\partial y_{ki}} f_n'(r_{ki})$$

but $\frac{\partial E_k}{\partial y_{ki}} = \sum_j \frac{\partial E_k}{\partial r_{kj}} \frac{\partial r_{kj}}{\partial y_{ki}} =$

$$\sum_j \frac{\partial E_k}{\partial r_{kj}} \frac{\partial}{\partial y_{ki}} \left(\sum_i y_{ki} w_{ji} \right) = -\sum_j \delta_{kj} w_{ji}$$

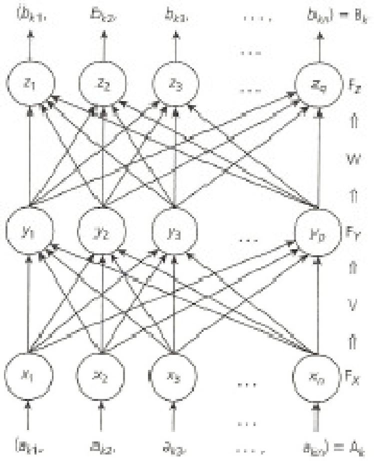
so therefore $\delta_{ki} = f_i'(r_{ki}) \sum_j \delta_{kj} w_{ji}$

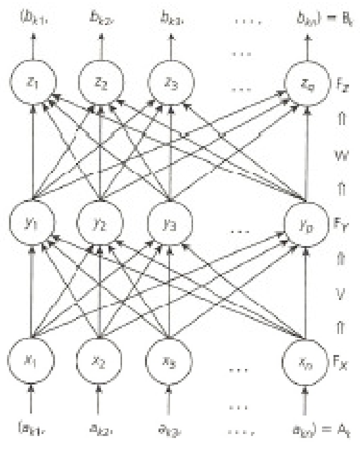


But $f_n'(r_{ki}) = \partial y_{ki} / \partial r_{ki} = y_{ki}(1 - y_{ki})$

So the error assigned to a hidden PE is

$$\underset{\text{Hidden PE}}{\delta_{ki}} = y_{ki}(1 - y_{ki}) \sum_j \underset{\text{Output PE}}{\delta_{kj}} w_{ji}$$





Backpropagation weight adjustments are:

$$w_{ji}^{new} = w_{ji}^{old} - \alpha \frac{\partial E}{\partial w_{ji}} = w_{ji}^{old} + \alpha \sum_k \delta_{kj} y_{ki}$$

$$v_{ih}^{new} = v_{ih}^{old} - \beta \frac{\partial E}{\partial v_{ih}} = v_{ih}^{old} + \beta \sum_k \delta_{ki} a_{kh}$$

Where $\alpha = \beta = \eta$ - learning rate

Back propagation algorithm

1. Present a training sample to the neural network.
2. Forward propagate the input pattern and compute output.
3. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
4. Calculate error in the hidden layer
5. Adjust the weights of each neuron in the output layer.
6. Adjust the weights of each neuron in the hidden layer.
7. Repeat from step 1 until convergence.

Back propagation/MLP Features

Training:

Back propagation procedure

Gradient descent strategy (usual problems)

Prediction:

Compute outputs based on input vector & weights

Pros:

Very general, Fast prediction

Cons:

Training can be VERY slow (1000's of epochs), Over fitting

Unstable convergence on noisy data!

Training Strategies

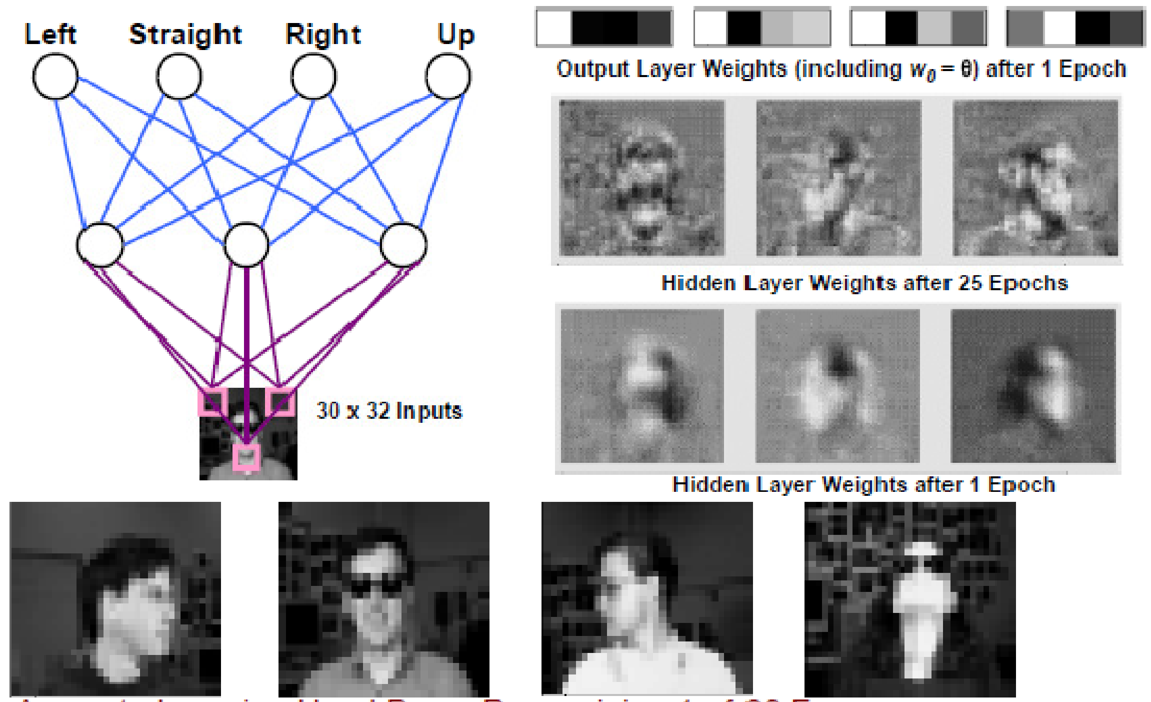
Online training:

Update weights after each sample

Offline (batch training):

Compute error over all samples. Then update weights, Online training "noisy" Sensitive to individual instances. However, may escape local minima.

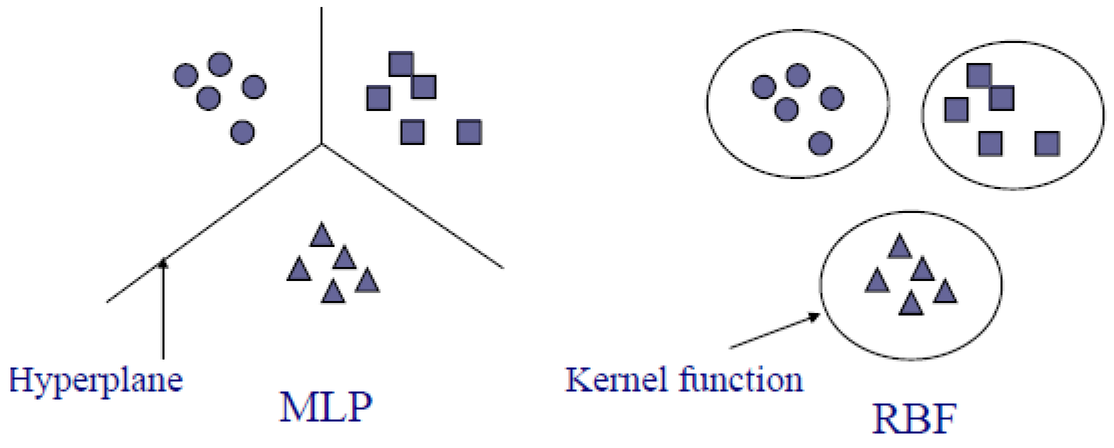
Neural Nets for Face Recognition


Left
Straight
Right
Up
30 x 32 Inputs
Output Layer Weights (including $w_0 = \theta$) after 1 Epoch
Hidden Layer Weights after 25 Epochs
Hidden Layer Weights after 1 Epoch

90% Accurate Learning Head Pose, Recognizing 1-of-20 Faces

http://www.cs.cmu.edu/~tom/faces.html

Radial Basis Function Networks

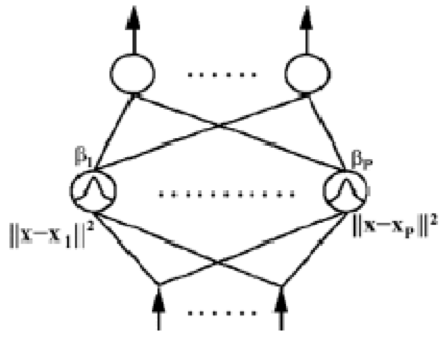


RBF networks are more suitable for probabilistic pattern classification

The probability density function (also called conditional density function or likelihood) of the k-th class is defined as

$$p(\vec{x} \mid C_k)$$

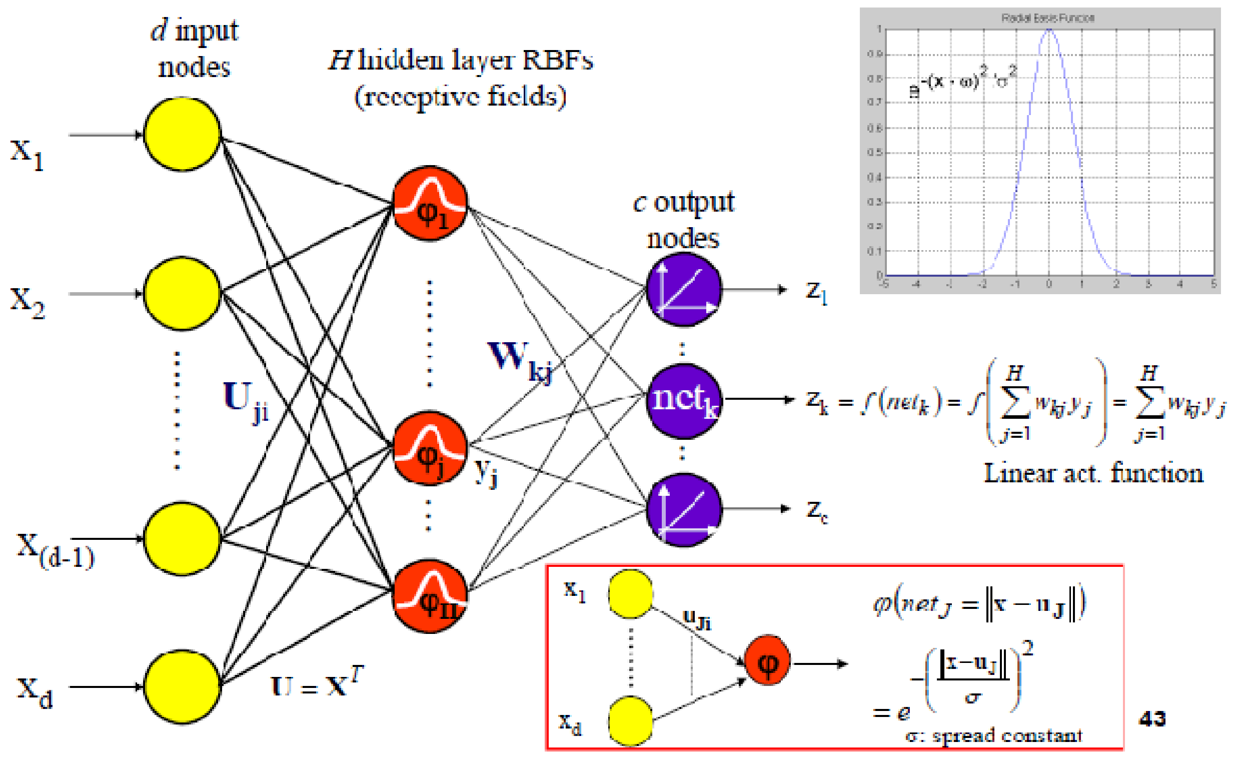
RBF Neural Network



- One hidden layer of RBF nodes
    - Each hidden node corresponds to a basis function center
- The output layer of linear nodes computes the sum of the weighted outputs from the RBF nodes

The centers and widths of the RBF Gaussian kernels are deterministic functions of the training data;

RBF: Gaussian basis function



d input nodes
H hidden layer RBFs (receptive fields)
c output nodes
x_1
x_2
$x_{(d-1)}$
x_d
U_{ji}
$U = X^T$
φ_1
φ_j
φ_H
y_j
W_{kj}
net_k
z_1
z_c
$z_k = f(net_k) = f\left(\sum_{j=1}^{H} w_{kj} y_j\right) = \sum_{j=1}^{H} w_{kj} y_j$
Linear act. function
Radial Basis Function
$e^{-(x-\omega)^2/\sigma^2}$
u_{Ji}
φ
$\varphi(net_J = \|x - u_J\|)$
$= e^{-\left(\frac{\|x-u_J\|}{\sigma}\right)^2}$
σ: spread constant
43

What do these parameters represent?
Physical meanings:
 The radial basis function for the hidden layer. This is a simple nonlinear mapping function (typically Gaussian) that transforms the *d*- dimensional input patterns to a (typically higher) *H*-dimensional space. The complex decision boundary will be constructed from linear combinations (weighted sums) of these simple building blocks.

uji: The weights joining the first to hidden layer. These weights constitute the center points of the radial basis functions.
The spread constant(s). These values determine the spread (extend) of each radial basis function.

Wjk: The weights joining hidden and output layers. These are the weights which are used in obtaining the linear combination of the radial basis functions. They determine the relative amplitudes of the RBFs when they are combined to form the complex function.

Unsupervised Learning

Objectives:

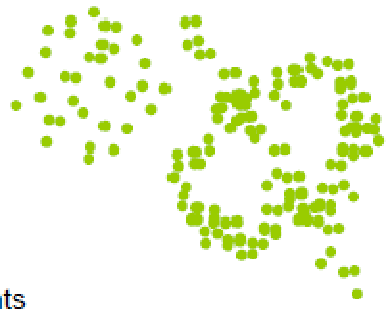
- Clustering: *grouping points* (x) into inherent regions of mutual similarity
- Vector quantization: *discretizing continuous space* with best labels
- Dimensionality reduction: *projecting many attributes* down to a few
- Feature extraction: *constructing (few) new attributes* from (many) old ones

Intuitive Idea

- Want to map independent variables (x) to dependent variables ($y = f(x)$)
- *Don't always know what "dependent variables" (y) are*
- Need to discover y based on numerical criterion (e.g., distance metric)

Clustering

A Mode of Unsupervised Learning



- Given: a collection of data points
- Goal: *discover structure* in the data
  - *Organize data into sensible groups* (how many here?)
  - Criteria: convenient and valid organization of the data
  - NB: not necessarily rules for classifying future data points
- Cluster analysis: study of algorithms, methods for discovering this structure

Cluster: Informal and Formal Definitions

- Set whose entities are alike and are different from entities in *other* clusters
- Aggregation of points in the instance space such that distance between any two points in the cluster is less than the distance between any point in the cluster and any point not in it

Kohonen nets

MATCHING AND SELF-ORGANIZING NET\

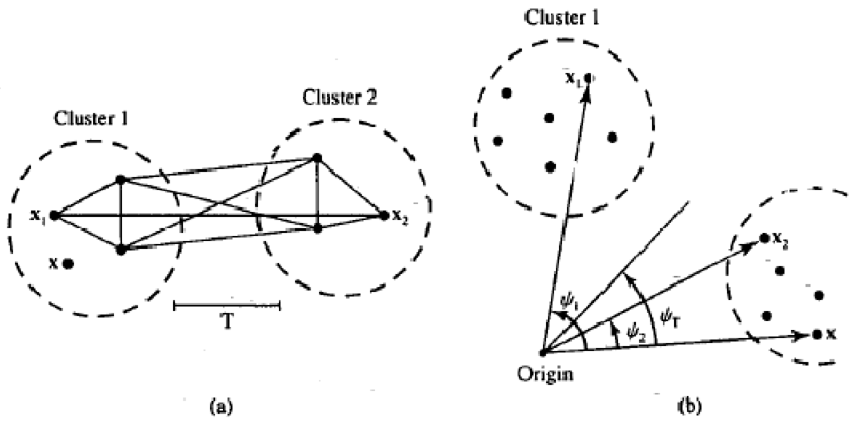
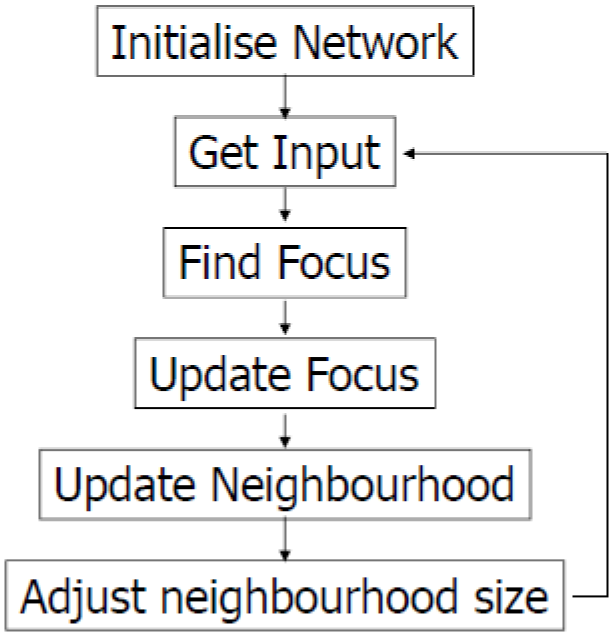


Figure 7.5 Measures of similarity for clustering data: (a) distance and (b) a normalized s product.

SOM Algorithm



Kohonen nets

- Finding the focus

$$\min_j(\| x - w_j \|)$$

x: input

w_j: jth neuron

- Updating neighbourhood

$$w_j = w_j + \eta(x - w_j)$$

η: Learning rate

w_j: jth neuron for all neurons within neighbourhood

Kohonen nets

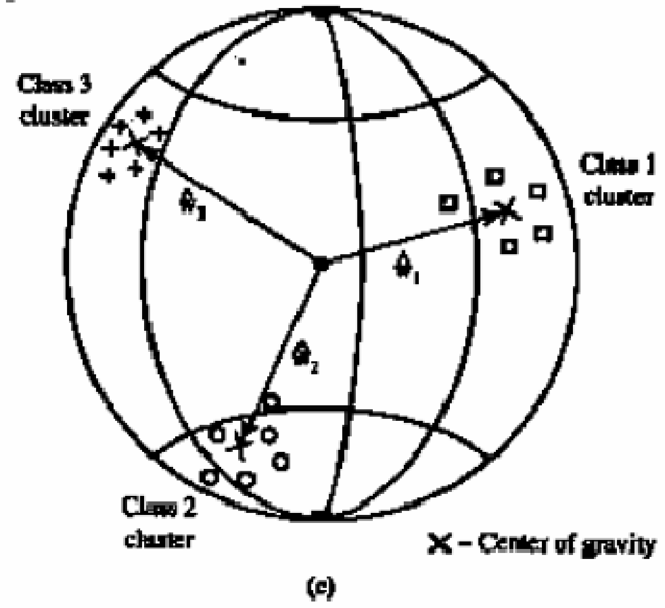
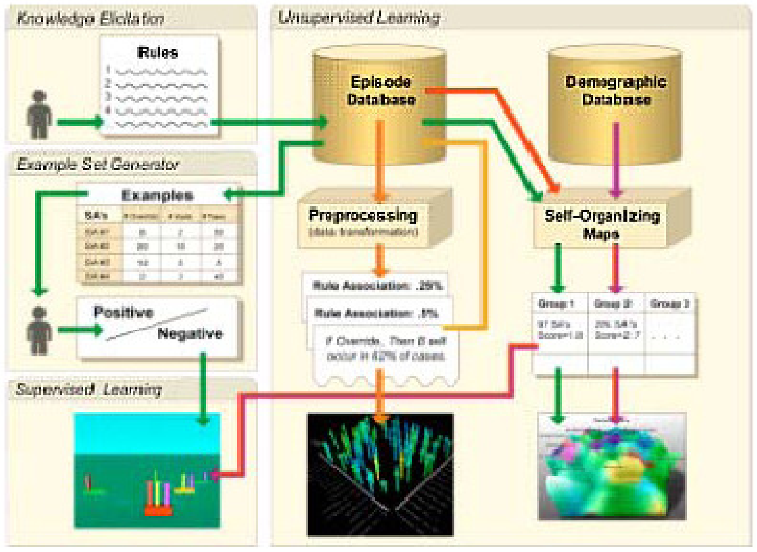


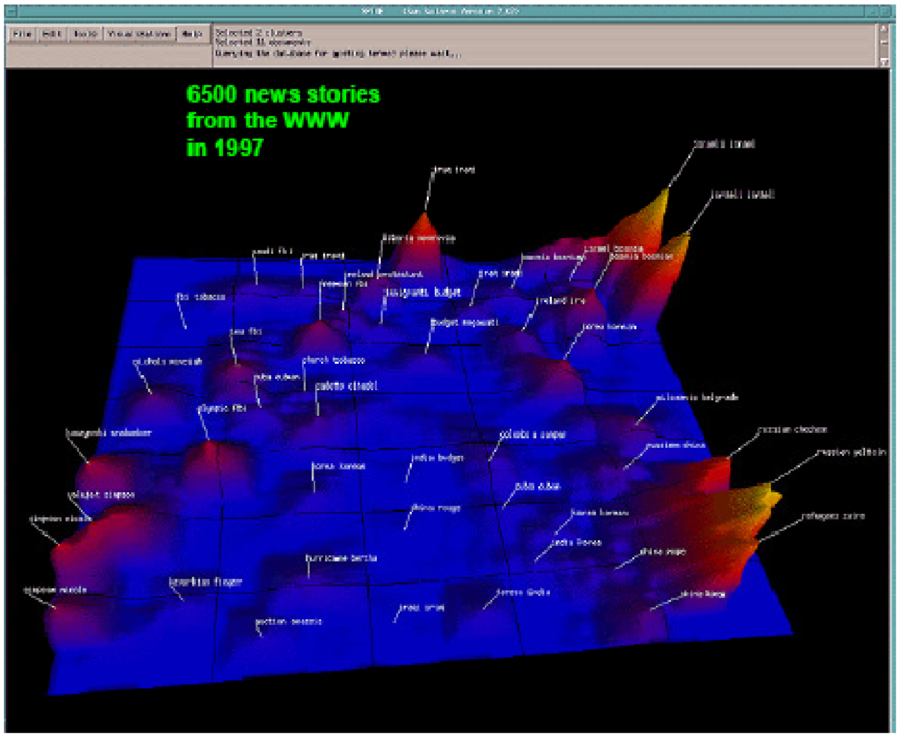
Figure 7.6 Winner-take-all learning rule: (a) learning layer, (b) vector diagram, and (c) wei
vectors on a unity sphere for $p = n = 3$.

Clustering: Applications



Transactional Database Mining

Clustering: Applications



ThemeScapes - http://www.cartia.com

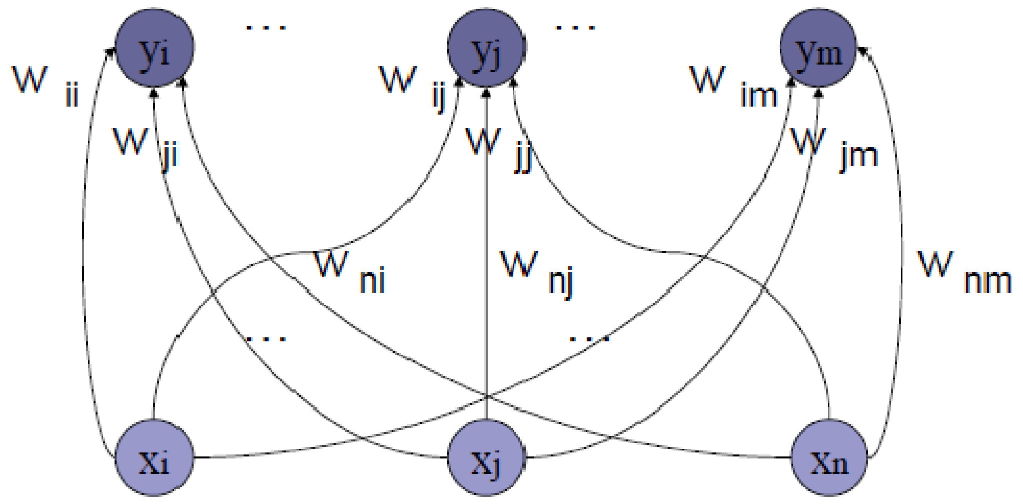
Categorization of text documents

Kohonen nets: VQ/LVQ

- Unsupervised or supervised learning: Vector Quantization / Learning Vector Quantization.

- LVQ intended for statistical classification. It uses pre-assigned cluster labels to data items to facilitate the two dimensional transformation so as to minimize the average expected misclassification probability.
- Each output unit represents a particular class.
- After training, an LVQ net classifies an input vector by assigning it to the same class as the output unit that has its weight vector (reference vector) closest to the input vector.

Kohonen nets: LVQ



LVQ algorithm

The motivation for the algorithm for the LVQ net is to find the output unit that is closest to the input vector.

```
Step0 : Initialize reference vectors and learning rate, a(0)
Step1 : While stopping condition = false do 2-6
Step2 : For each training input vector x,do 3-4
Step3 : Find J so that $|| x-w_j ||$=min
Step4 : Update $w_j$ as follows:
            □ If T = $C_j$, then
                ▪ $w_j(new) = w_j(old) + a[x – w_j(old)]$ ;
            □ If T != $C_j$, then
                ▪ $w_j(new) = w_j(old) - a[x – w_j(old)]$ ;
Step5 : Reduce learning rate
Step6 : Test stopping condition: The condition may specify a fixed
    number of iterations or the learning rate reaching a sufficiently small
    value
```