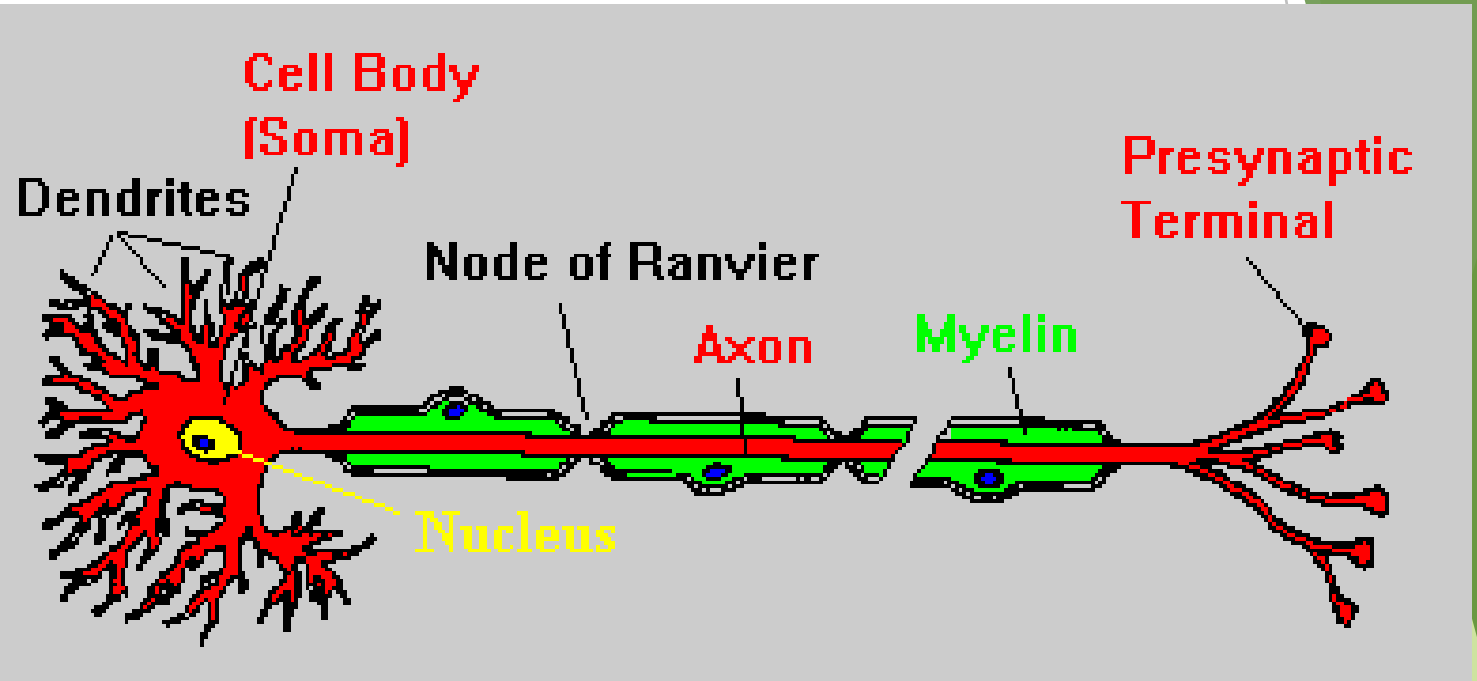
Introduction to Neural Networks

The Biological Neuron



The human brain is made of about 100 billions of such neurons.

Characteristics of Biological Neural Networks

1) Massive connectivity
2) Nonlinear, Parallel, Robust and Fault Tolerant
3) Capability to adapt to surroundings
4) Ability to learn and generalize from known examples
5) Collective behavior is different from individual behavior

Artificial Neural Networks mimics some of the properties of the biological neural networks

Some Properties of Artificial Neural Networks

Assembly of simple processors

Information stored in connections – No Memory

Massively Parallel

Massive connectivity

Fault Tolerant

Learning and Generalization Ability

Robust

Individual dynamics different from group dynamics

*All these properties may **not** be present in a particular network*

Network Characteristics

Neural Network Characterized by:

1) Architecture
2) Learning (update scheme of weights and/or outputs)

Architecture

Layered (single /multiple): Feed forward – *MLP, RBF*

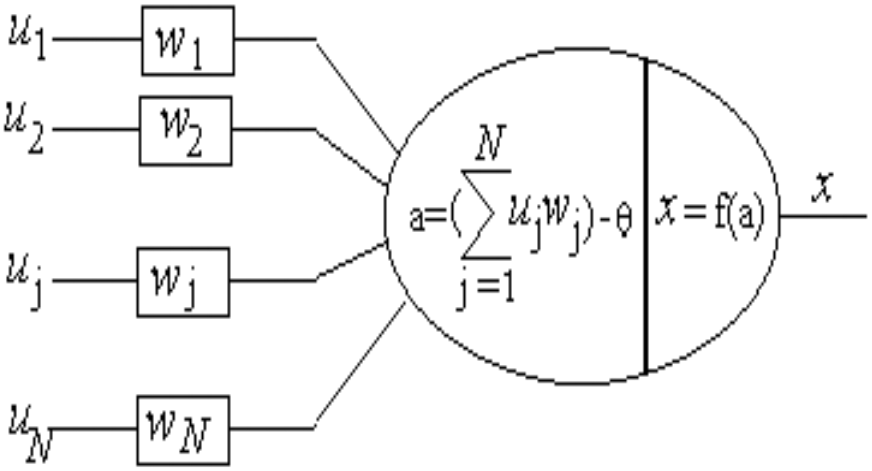
Recurrent : At least one feedback loop – Hopfield

Competitive : p – dimensional array of neurons with a set of nodes supplying input to each element of the array – *LVQ, SOFM*

Learning

- **Supervised** : In Presence of a teacher
- **Unsupervised or Self-Organized** : No teacher
- **Reinforcement**: Trial and error, no teacher, but can asses the situations – reinforcement signals.

Model of an Artificial Neuron



$\boldsymbol{u^T} = (u_1, u_2, \ldots, u_N)$ The input vector

$\boldsymbol{w^T} = (w_1, w_2, \ldots, w_N)$ The weight vector

Activation Functions

1) **Threshold Function**

$f(v) = 1$ if $v \geq 0$

$= 0$ otherwise

2) **Piecewise-Linear Function**

$f(v) = 1$ if $v \geq \frac{1}{2}$

$= v$ if $\frac{1}{2} > v > -\frac{1}{2}$

$= 0$ otherwise

3) **Sigmoid Function**

$f(v) = 1/\{1 + exp(-av)\}$

etc..

Perceptron Learning Algorithm

Assume we are given a data set $X=\{(\boldsymbol{x}_1,y_1),....,(\boldsymbol{x}_l,y_l)\}$, where $\boldsymbol{x} \in R^n$ and y = {1,-1}.

Assume X is linearly separable i.e.:

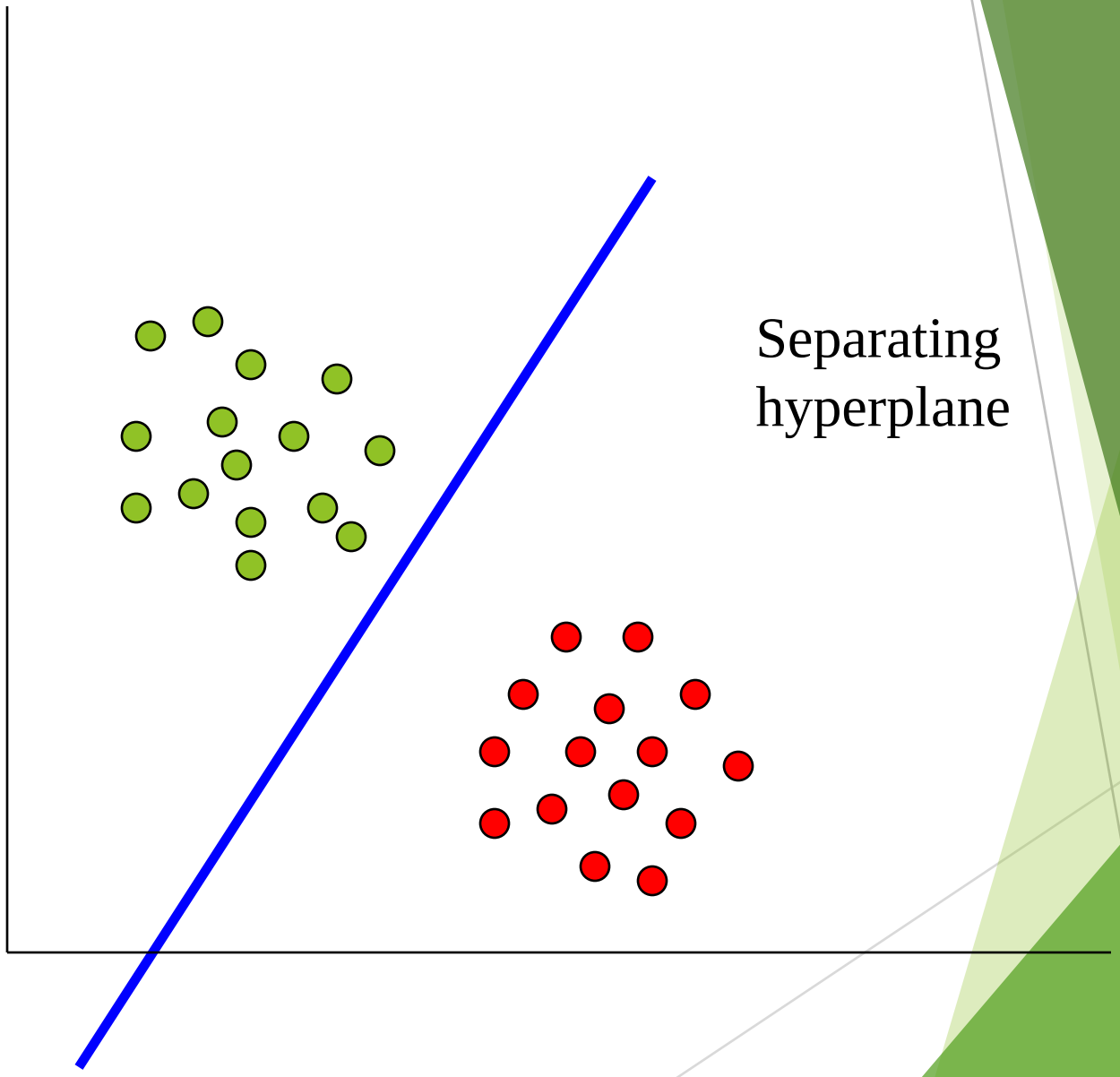
There exists a $\boldsymbol{w}$ and b, such that

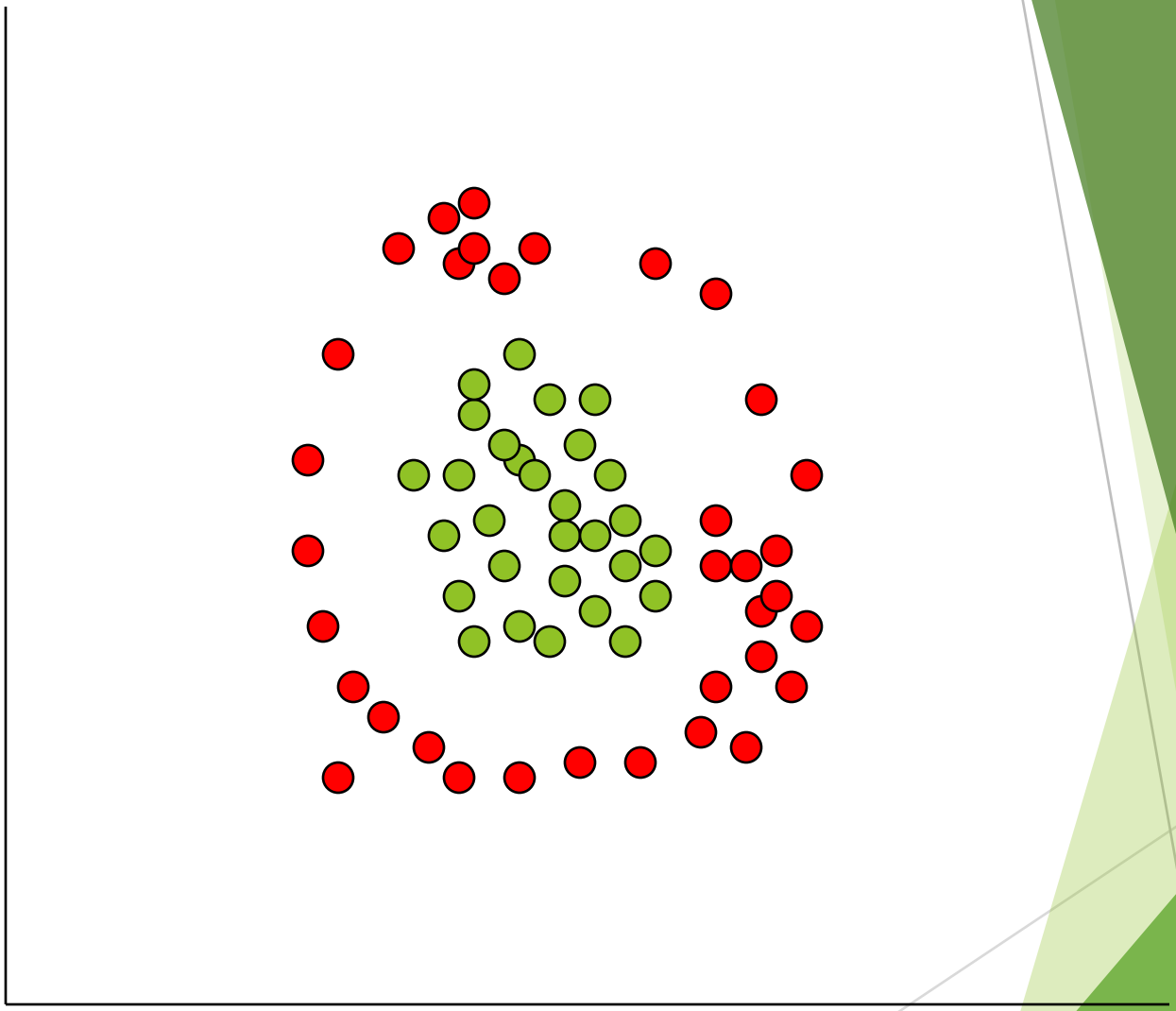
$$(\boldsymbol{w}^T \boldsymbol{x}_i + b)y_i > 0, \text{ for all } i$$

Classification of X means finding a $\boldsymbol{w}$ and b such that

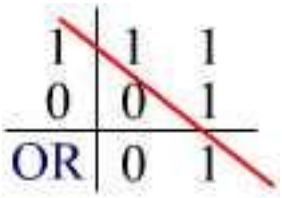
$$(\boldsymbol{w}^T \boldsymbol{x}_i + b)y_i > 0, \quad \text{for all } i$$

A perceptron can classify X in a finite number of steps

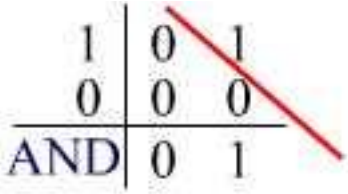

Separating
hyperplane



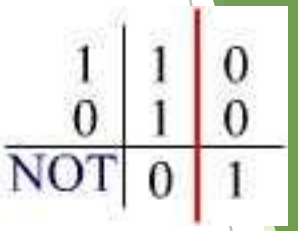
Linearly separable



	1	1
0	0	1
OR	0	1



	0	1
0	0	0
AND	0	1



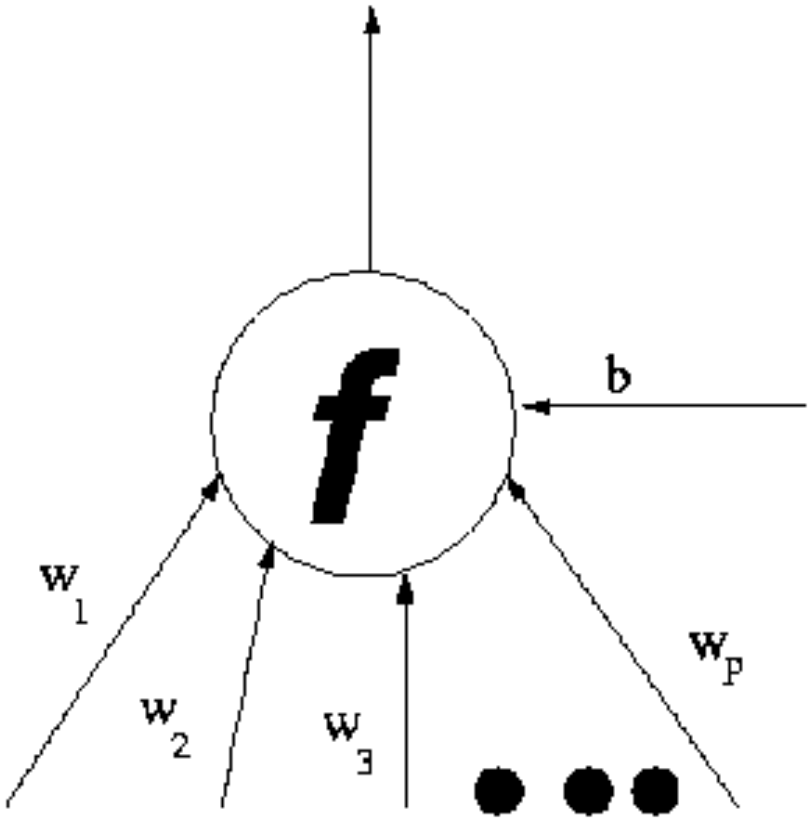
	1	0
0	1	0
NOT	0	1

OR, AND and NOT are linearly separable Boolean Functions

	1	0
0	0	1
XOR	0	1

XOR is not linearly separable

Perceptron Learning Algorithm (Contd.)



$f(net_i) = 1$ if $net_i > 0$

$f(net_i) = -1$ otherwise

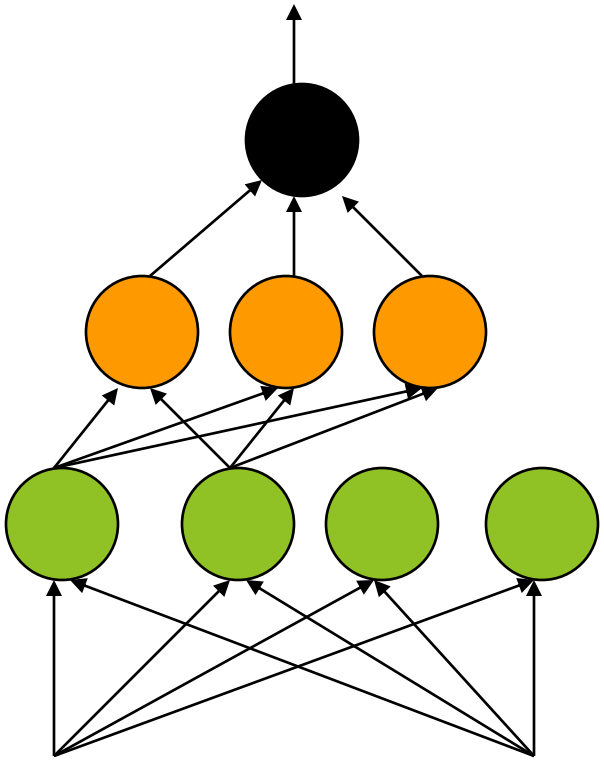
$net_i = \boldsymbol{w}^T \boldsymbol{x}_i$

Starting with $\boldsymbol{w}^{(0)}=0$ we follow the following learning rule:

$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha\, y_i\, \boldsymbol{x}_i$

for each misclassified point $\boldsymbol{x}_i$

The Multilayered Perceptron



MLPs are layered feed-forward networks.

The n-th layer is fully connected with the (n+1)-th layer.

They are widely used for learning input-output mappings from data which has varied scientific and engineering applications.

Each node in an MLP behaves like a perceptron with a sigmoidal activation function.

Multilayered Perceptrons (Contd.)

An MLP can learn efficiently any input-output mapping.

Suppose we have a training set

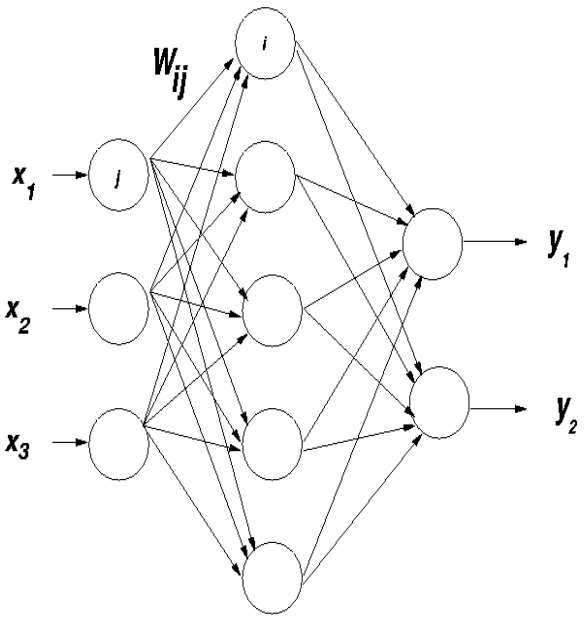
$X=\{(\boldsymbol{x}_1,\boldsymbol{y}_1),....,(\boldsymbol{x}_n,y_n)\}$, where $\boldsymbol{x}\in R^p$ and $\boldsymbol{y}\in R^q$.

There is an unknown functional relationship between ***x*** and ***y.***

Say, $\boldsymbol{y = F(x).}$

Our objective is to learn ***F,*** given X.

Multilayered Perceptrons (Contd.)



When an input vector is given to an MLP it computes a function. The function ***F**** which the MLP computes has the weights and biases of each nodes as a parameter. Let ***W*** be a vector which contains all the weights and biases associated with the MPL as its elements**,** thus the MLP computes the function ***F*(W,x).***

Our objective would be to find such a ***W*** which minimizes

$$E = \tfrac{1}{2} \sum_i \|F^*(W,x_i) - y_i\|^2$$

The Gradient descent algorithm

Let $\boldsymbol{w} = (w_1, \ldots, w_N)^T$ be a vector of N adjustable parameters.

Let $J(\boldsymbol{w})$ be a scalar cost function, with the following properties :

1) **Smoothness**: The cost function $J(\boldsymbol{w})$ is twice differentiable with respect to any pair (w_j, w_j) for $1 \le i \le j \le N$.

2) **Existence of Solution:** At least one parameter vector $\boldsymbol{w}_{opt} = (w_{1,opt}, \ldots, w_{N,opt})^T$ exists, such that

a) $$\frac{\partial J(w_{opt})}{\partial w_i} = 0 \qquad 1 \le i \le N$$

b) The N×N Hessian Matrix H($\boldsymbol{w}$) with entries $h_{ij}(\boldsymbol{w})$

$$h_{ij}(w) = \frac{\partial^2 J(w)}{\partial w_i \partial w_j}$$

Is positive definite for $\boldsymbol{w} = \boldsymbol{w}_{opt}$

The Gradient Descent Algorithm (Contd.)

The minimizer for J can be found as

$$w(k+1) = w(k) - \mu(k)\frac{\partial J(w(k))}{\partial w}$$

Where ***w(0)*** is any initial parameter vector and $\mu(k)$ is a positive values sequence of step sizes.

This optimization procedure may lead to a **local minima** of the cost function *J*.

Training the MLP

The weights of a MLP which minimizes the error *E* can also be found by the gradient descent algorithm. This method when applied to a MLP is called the backpropagation which have two passes.

Forward pass: where the output is calculated

Backward pass: According to the error the weights are updated

Modes of update:

Batch Update

Online Update

Multilayered Perceptron (Contd.)

Some important issues:

How big should be my network ?

No specific answer is known till date. The size of the network depends on the complexity of the problem at hand and the training accuracy which is desired. A good training accuracy does not always means a good network. If the number of free parameters of the network is almost the same as the number of data points, the network tends to memorize the data and gives *bad* generalization.

How many hidden layers to use ?

It has been proved that a single hidden layer is sufficient to do any mapping task. But still experience shows that multiple hidden layers may be sometimes simplify learning.

Can a trained network generalize on all data points ?

No, it can generalize only on data points which lies within the boundary of the training sample. The output given by an MLP is never reliable on data points far away from the training sample.

Can I get the explicit functional form of the relationship that exists in my data from the trained MLP?

No, one may write a functional form of nested sigmoids, but it will (in almost all cases) be far from useful. MLPs are **black-boxes,** one cannot retrieve the rules which governs the input-output mapping from a trained MLP by any easy means.

Generalization

A network is said to generalize well if it produces correct output (or nearly so) for a input data point never used to train the network.

The training of an MLP may be viewed as a “curve fitting” problem. The network performs useful generalization (interpolation) as MLPs with continuous activation functions leads to continuous outputs.

If an MLP have too many free parameters compared to the diversity in the data, the network may tend to memorize the training data.

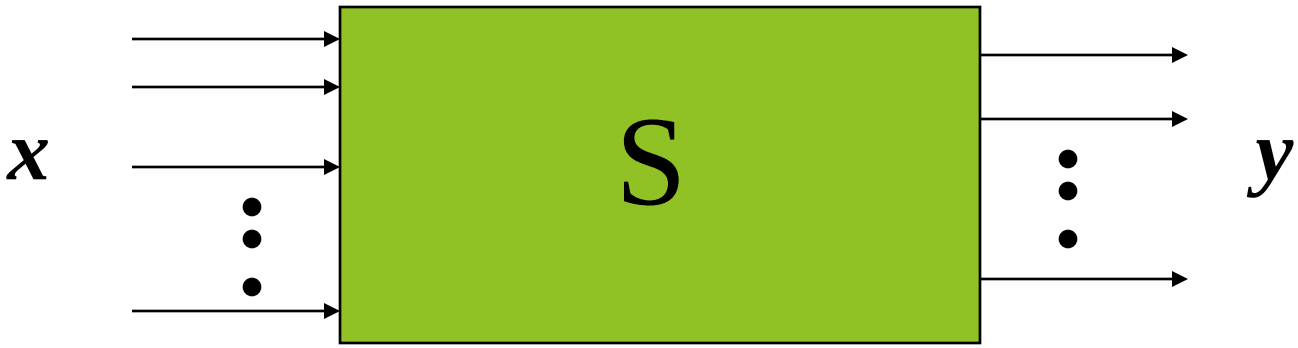
Generalization ability depends on:

1) Representativeness of the training set
2) The architecture of the network
3) The complexity of the problem

Some applications

1) Function approximation
2) Classification
   a) Land Cover classification for remotely sensed images
   b) Optical Character Recognition
   many more !!
3) Dimensionality Reduction

Function approximation



The system S can be any type of system with numerical input and output.

Classification

Classifiers are functions of special types which do not have numerical outputs but have *class labels* as outputs.

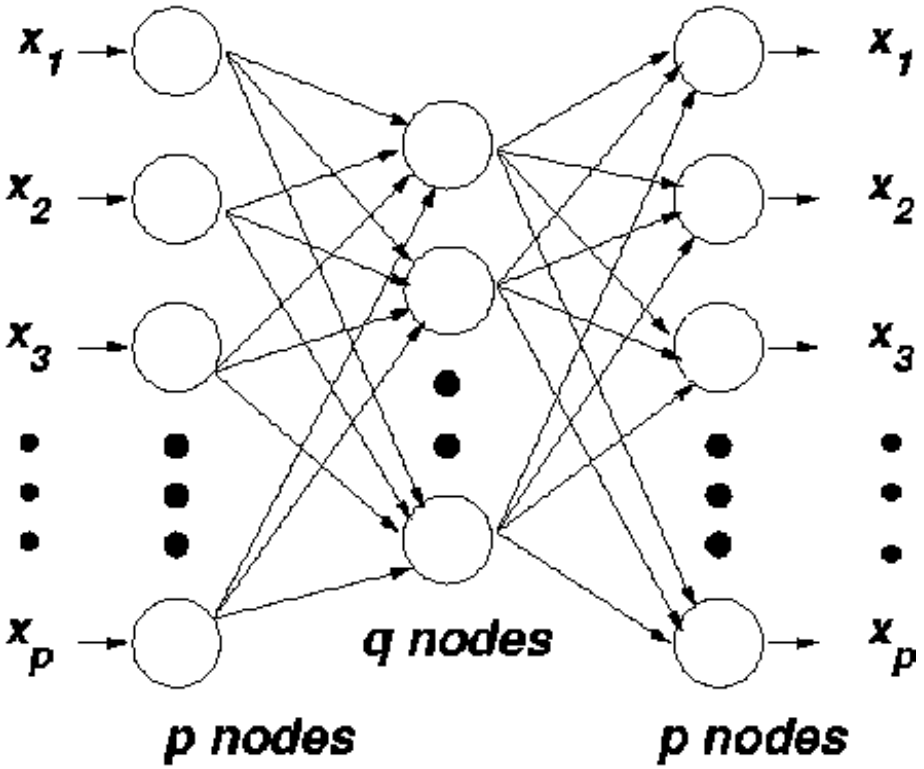
$$D: R^p \rightarrow N_{pc}$$

The class labels can be numerically coded and thus an MLP may be used to learn a classification problem.

Example: We may code three different classes as

0 0 1 -- Class1
0 1 0 -- Class2
1 0 0 – Class3

Dimensionality Reduction by MLP



There can be other approaches too !!

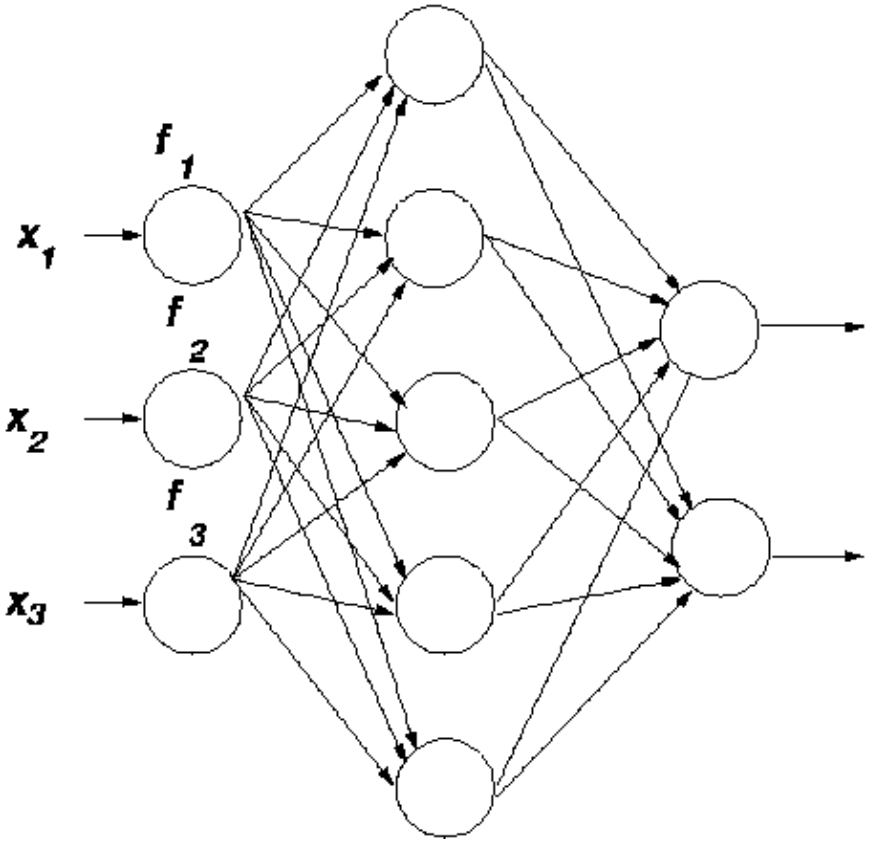
Both the input and output nodes contains p nodes and the hidden layer contain q nodes. Here $q<p$.

A pattern $\boldsymbol{x} = (x_1,...,x_p)$ is presented to the network with the same target $\boldsymbol{x}$.

If the output from the hidden layer of the trained network is tapped, then we get a transformed set of feature vectors $\boldsymbol{y} \in R^q$

But, these feature vectors y are not interpretable.

Online Feature Selection by MLP



Associate with each input node *i* a multiplier f_i.

f_i takes values in [0,1].

f_i's takes values near one for good features and near zero for bad/redundant ones.

A good choice

$f_i = f(\lambda_i) = 1/(1+e^{-\lambda i})$

λ_i's are learnable.

Initialization.