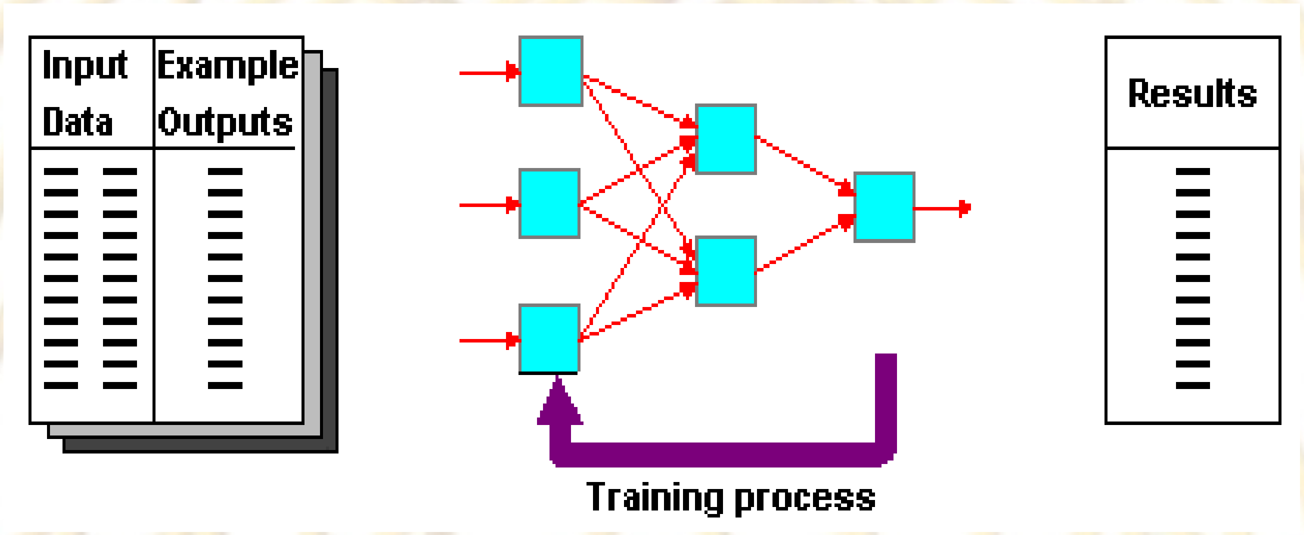
CHAPTER 3

SUPERVISED LEARNING NETWORK

DEFINITION OF SUPERVISED LEARNING NETWORKS

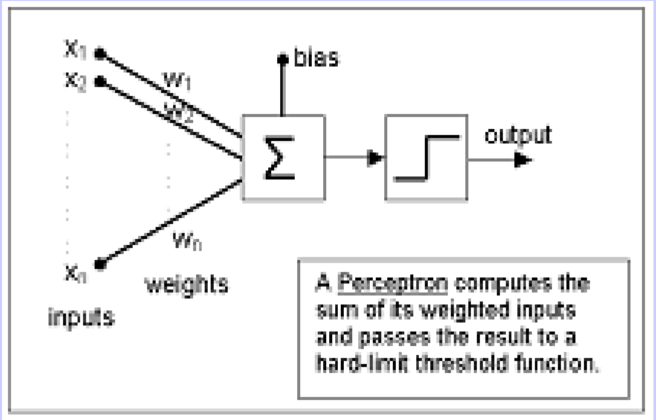
- Training and test data sets

- Training set; input & target are specified



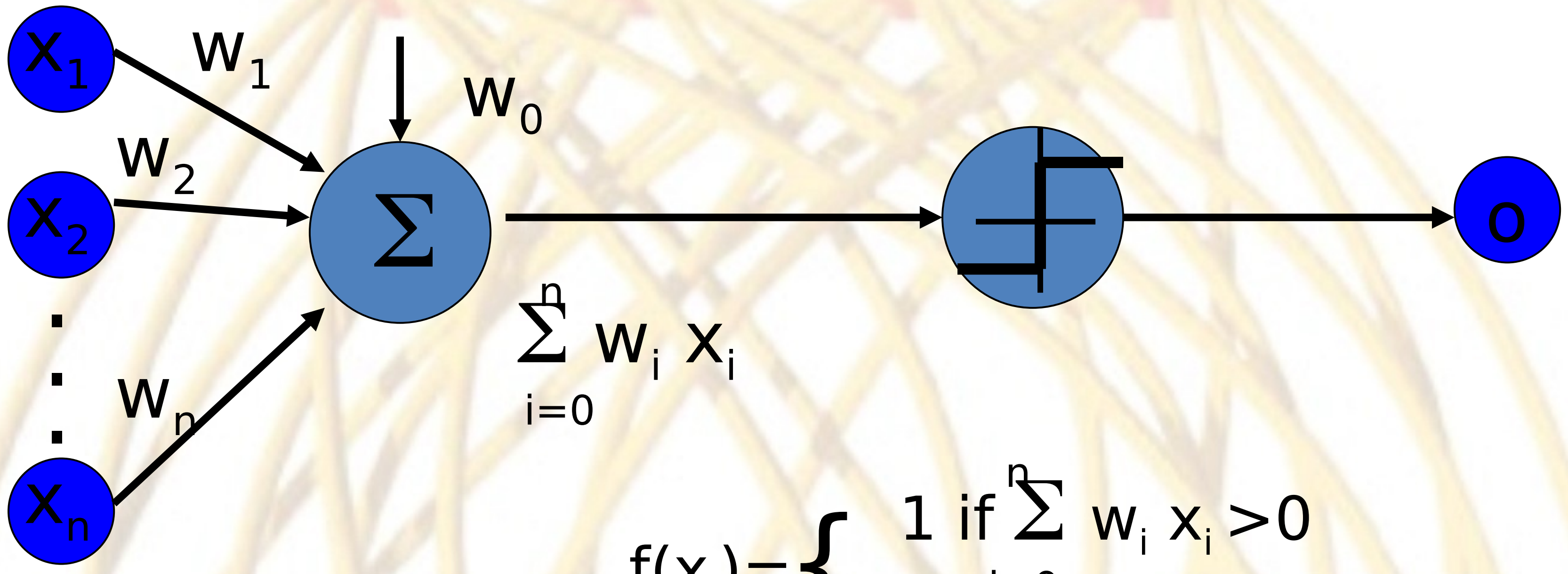
PERCEPTRON NETWORKS

- One type of NN system is based on the "perceptron".
- A perceptron computes a sum of weighted combination of its inputs, if the sum is greater than a certain threshsold (bias), then it ouputs a "1", else a "-1".



A Perceptron computes the sum of its weighted inputs and passes the result to a hard-limit threshold function.

➢ Linear threshold unit (LTU)



$$f(x_i) = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

PERCEPTRON LEARNING

$w_i = w_i + \Delta w_i$

$\Delta w_i = \eta\ (t - o)\ x_i$

where

$t = c(x)$ is the target value,

o is the perceptron output,

η Is a small constant (e.g., 0.1) called learning rate.

- If the output is correct ($t = o$) the weights wi are not changed

- If the output is incorrect ($t \neq o$) the weights wi are changed such that the output of the perceptron for the new weights is closer to t.

- The algorithm converges to the correct classification
  - if the training data is linearly separable
  - ∀ η is sufficiently small

LEARNING ALGORITHM

- **Epoch** : Presentation of the entire training set to the neural network.

- In the case of the AND function, an epoch consists of four sets of inputs being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1]).

- **Error**: The error value is the amount by which the value output by the network differs from the target value. For example, if we required the network to output 0 and it outputs 1, then Error = -1.

- **Target Value, T** : When we are training a network we not only present it with the input but also with a value that we require the network to produce. For example, if we present the network with [1,1] for the AND function, the training value will be 1.

- **Output , O** : The output value from the neuron.

- **Ij** : Inputs being presented to the neuron.

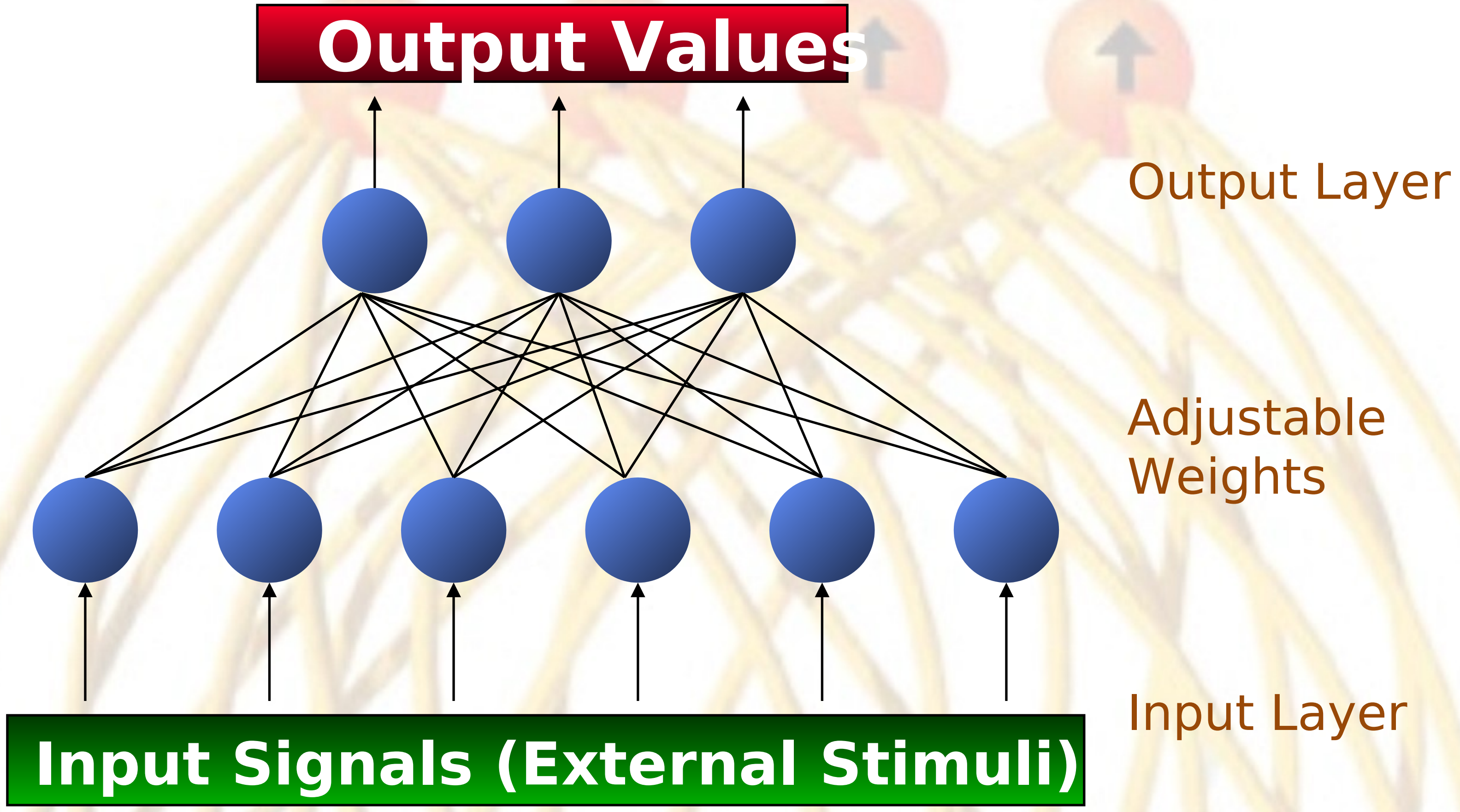
- **Wj** : Weight from input neuron (Ij) to the output neuron.

- **LR** : The learning rate. This dictates how quickly the network converges. It is set by a matter of experimentation. It is typically 0.1.

TRAINING ALGORITHM

- Adjust neural network weights to map inputs to outputs.
- Use a set of sample patterns where the desired output (given the inputs presented) is known.
- The purpose is to learn to
  - Recognize features which are common to good and bad exemplars

MULTILAYER PERCEPTRON



LAYERS IN NEURAL NETWORK

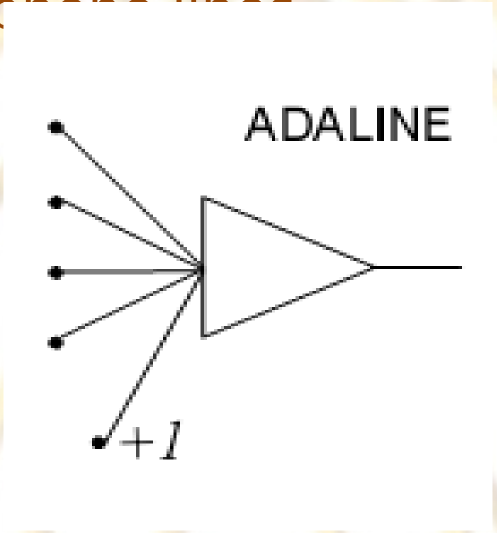
- **The input layer:**
  - Introduces input values into the network.
  - No activation function or other processing.

- **The hidden layer(s):**
  - Performs classification of features.
  - Two hidden layers are sufficient to solve any problem.
  - Features imply more layers may be better.

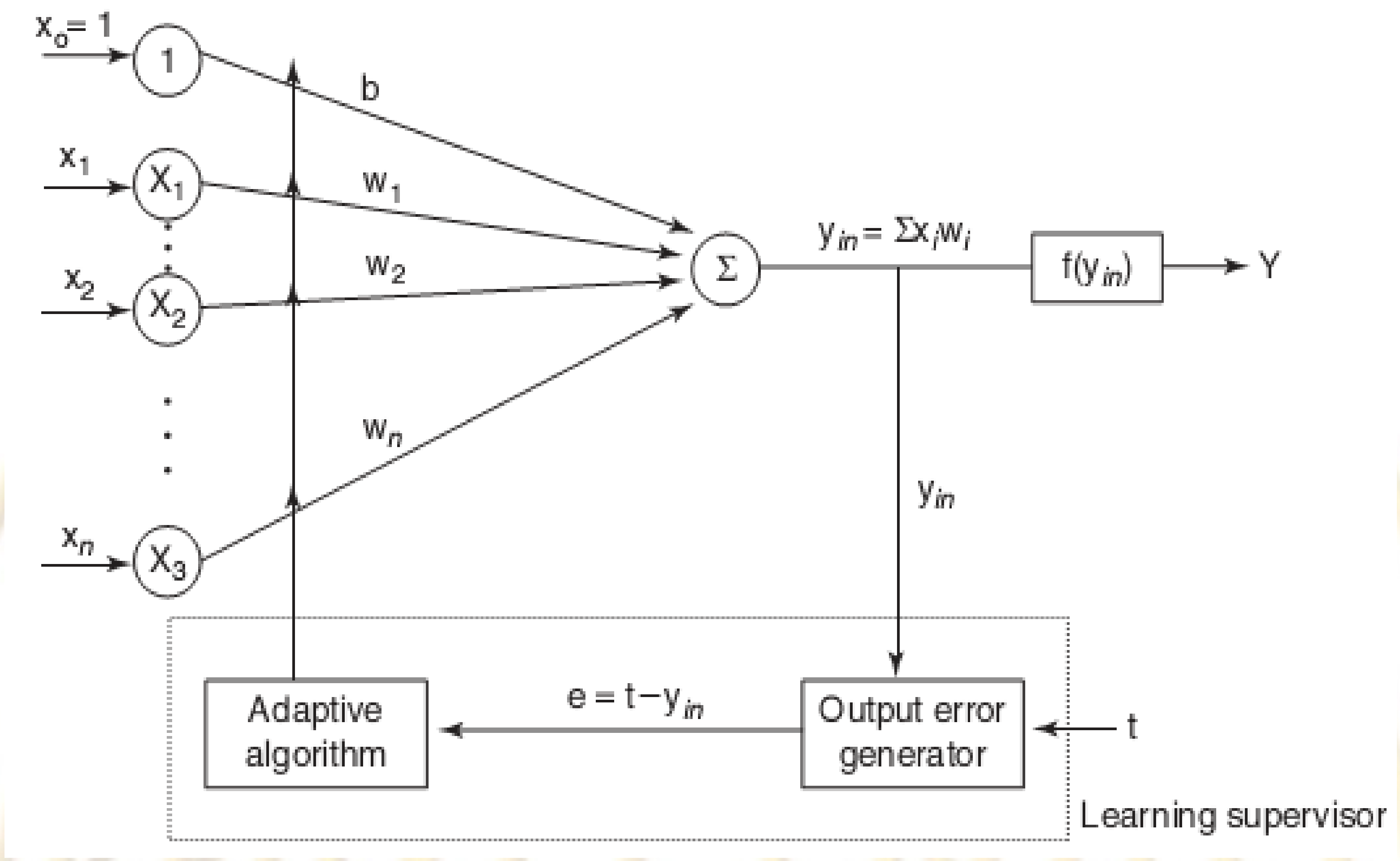
- **The output layer:**
  - Functionally is just like the hidden layers.
  - Outputs are passed on to the world outside the neural network.

ADAPTIVE LINEAR NEURON (ADALINE)

In 1959, Bernard Widrow and Marcian Hoff of Stanford developed models they called ADALINE (Adaptive Linear Neuron) and MADALINE (Multilayer ADALINE). These models were named for their use of Multiple ADAptive LINear Elements. MADALINE was the first neural network to be applied to a real world problem. It is an adaptive filter which eliminates echoes on phone lines.



ADALINE MODEL



ADALINE LEARNING RULE

Adaline network uses Delta Learning Rule. This rule is also called as Widrow Learning Rule or Least Mean Square Rule. The delta rule for adjusting the weights is given as (i = 1 to n):

$$\Delta w_i = \alpha(t - y_{in})x_i$$

Δw_i = weight change

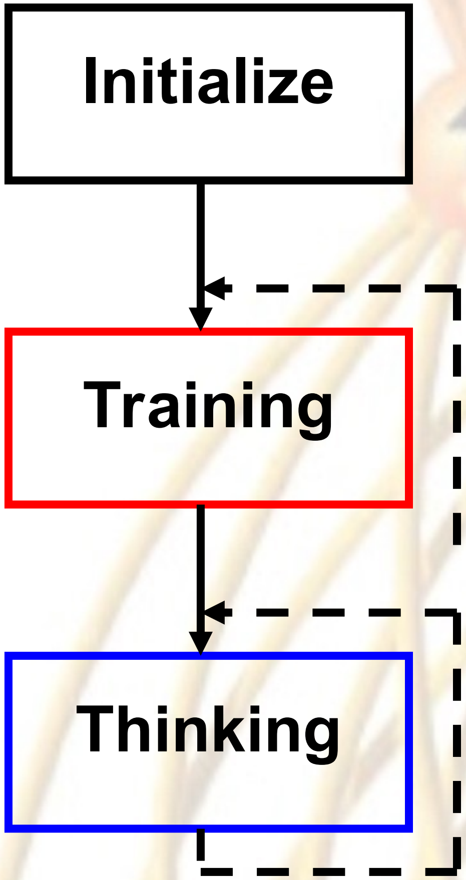
α = learning rate

x = vector of activation of input unit

y_{in} = net input to output unit, i.e., $Y = \sum_{i=1}^{n} x_i w_i$

t = target output

USING ADALINE NETWORKS



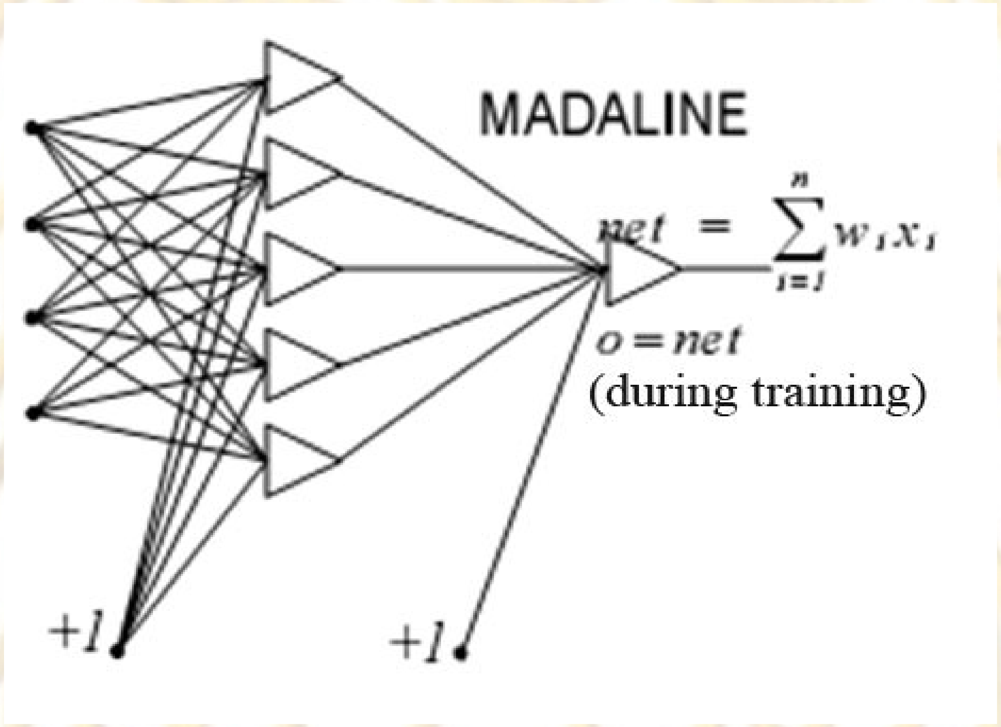
- Initialize
  - Assign random weights to all links

- Training
  - Feed-in known inputs in random sequence
  - Simulate the network
  - Compute error between the input and the output (Error Function)
  - Adjust weights (Learning Function)
  - Repeat until total error < ε

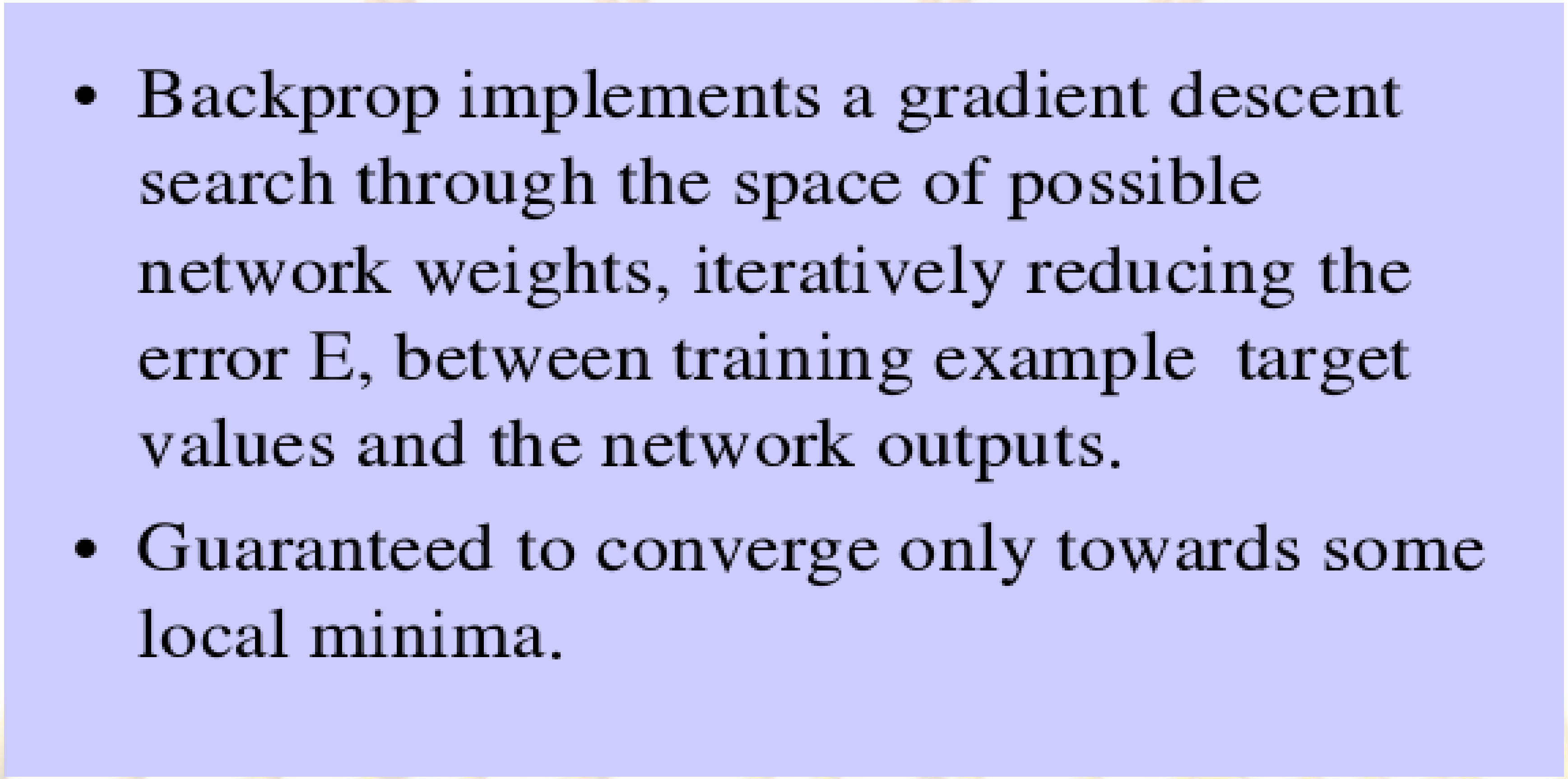
- Thinking
  - Simulate the network
  - Network will respond to any input
  - Does not guarantee a correct solution even for trained inputs

MADALINE NETWORK

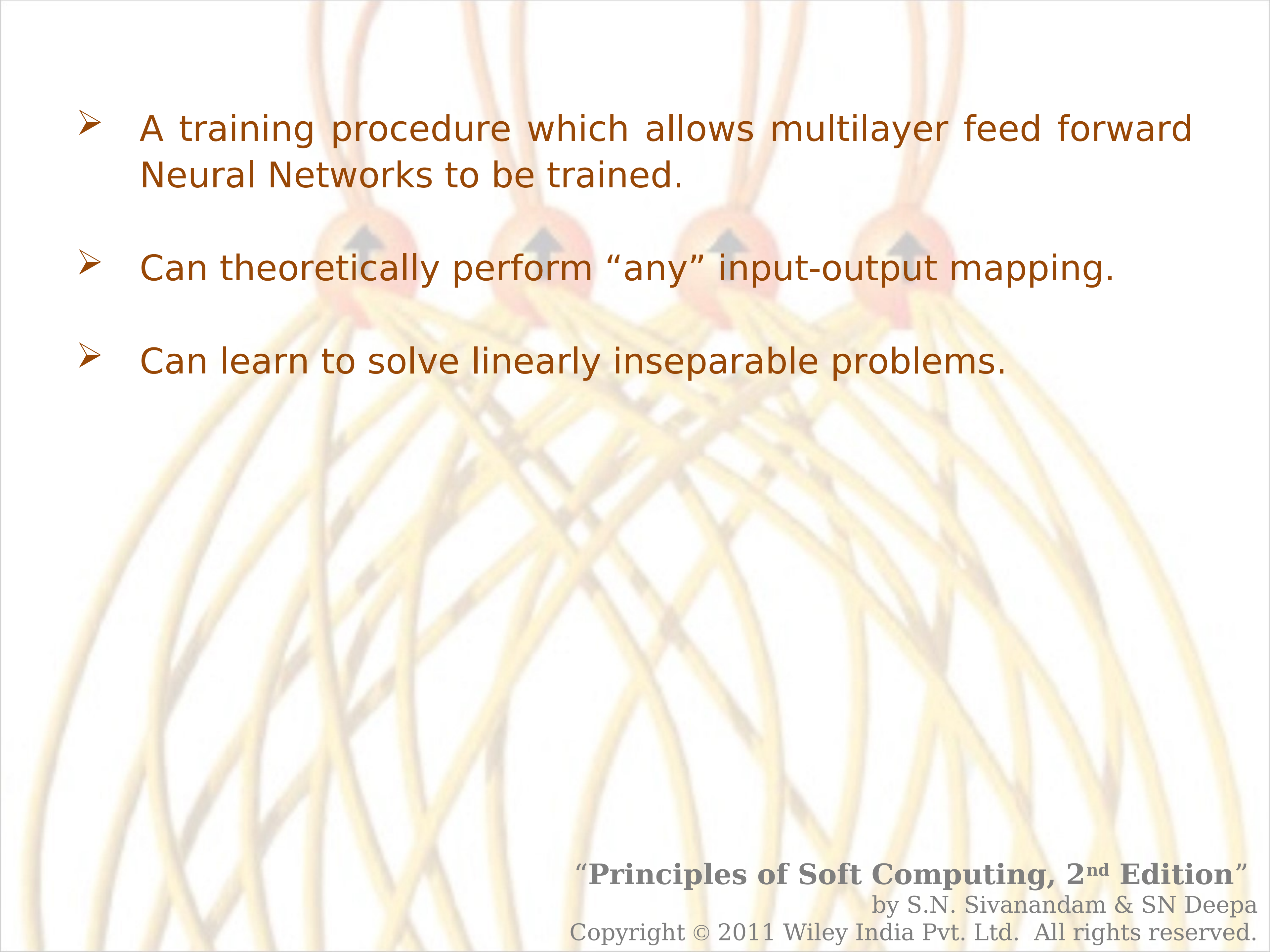
MADALINE is a Multilayer Adaptive Linear Element. MADALINE was the first neural network to be applied to a real world problem. It is used in several adaptive filtering process.



BACK PROPAGATION NETWORK

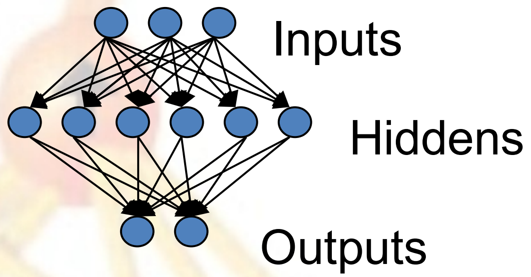


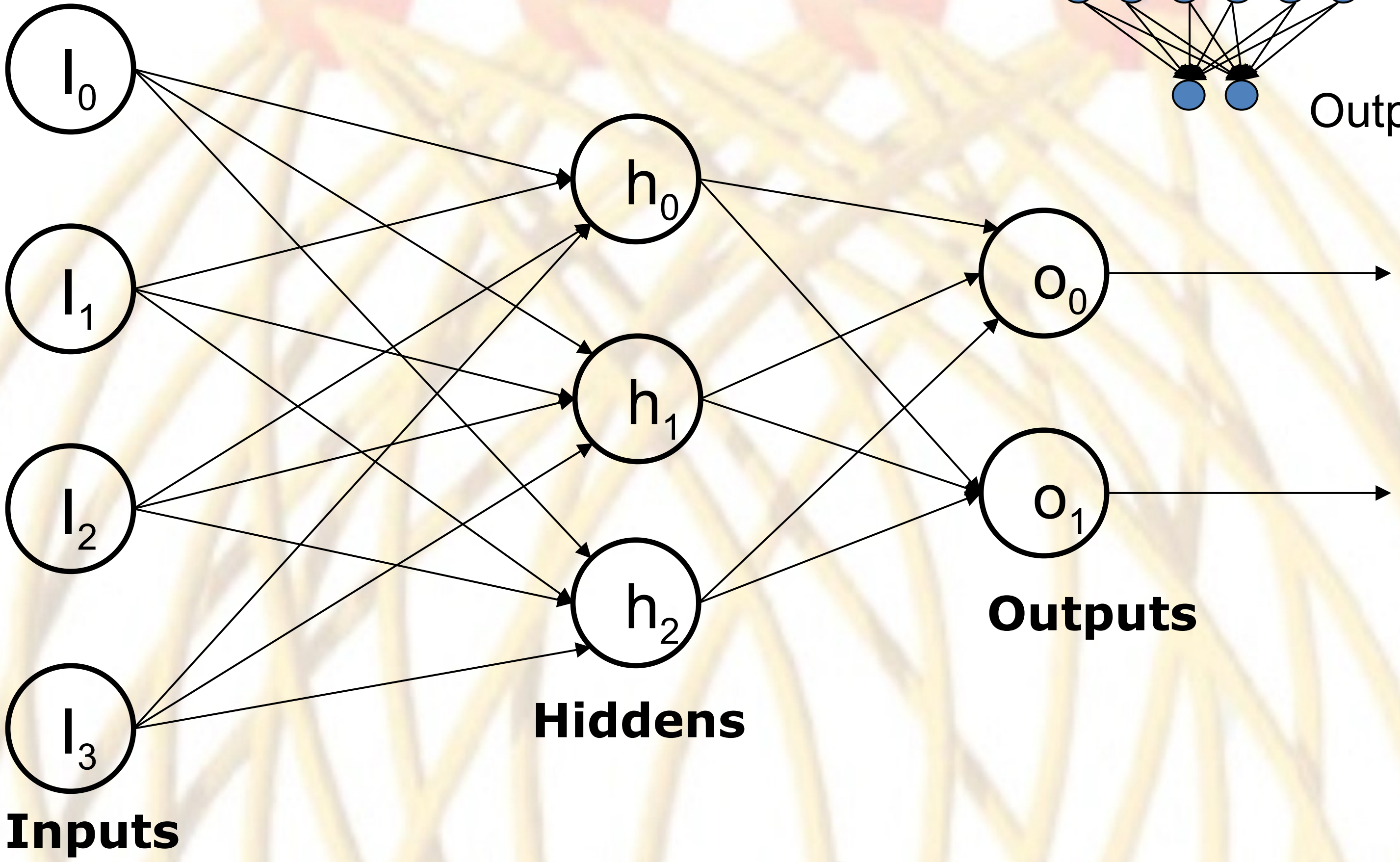
- Backprop implements a gradient descent search through the space of possible network weights, iteratively reducing the error E, between training example target values and the network outputs.
- Guaranteed to converge only towards some local minima.



- A training procedure which allows multilayer feed forward Neural Networks to be trained.
- Can theoretically perform “any” input-output mapping.
- Can learn to solve linearly inseparable problems.

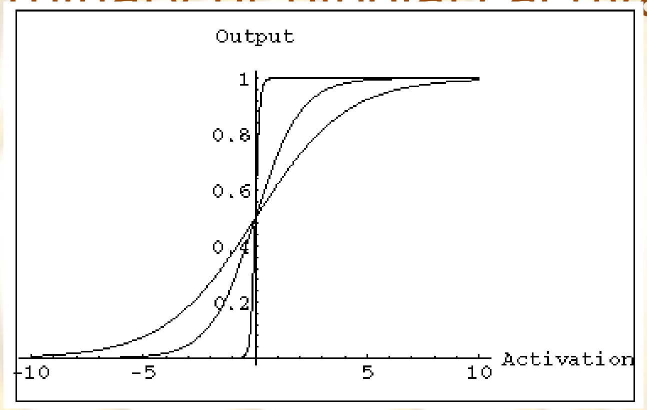
MULTILAYER FEEDFORWARD NETWORK

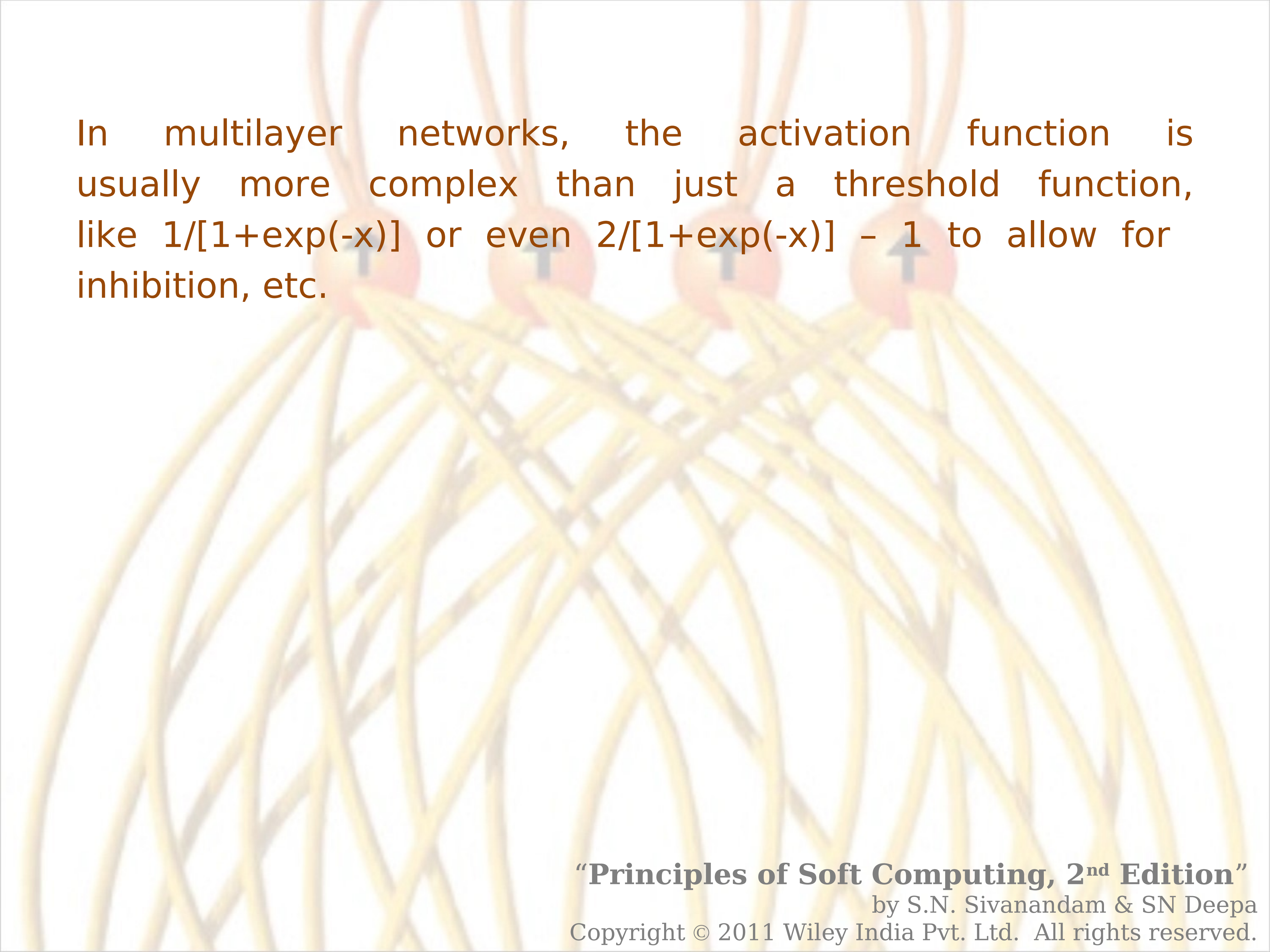




MULTILAYER FEEDFORWARD NETWORK: ACTIVATION AND TRAINING

- For feed forward networks:
  - A continuous function can be
  - differentiated allowing
  - gradient-descent.
  - Back propagation is an example of a gradient-descent technique.
  - Uses sigmoid (binary or bipolar) activation function.





In multilayer networks, the activation function is usually more complex than just a threshold function, like 1/[1+exp(-x)] or even 2/[1+exp(-x)] – 1 to allow for inhibition, etc.

GRADIENT DESCENT

- Gradient-Descent(training_examples, η)

- Each training example is a pair of the form $<(x_1,\ldots x_n),t>$ where $(x_1,\ldots,x_n)$ is the vector of input values, and t is the target output value, η is the learning rate (e.g. 0.1)

- Initialize each wi to some small random value

- Until the termination condition is met, Do
  - Initialize each Δwi to zero
  - For each $<(x_1,\ldots x_n),t>$ in training_examples Do

- ✓ Input the instance (x1,…,xn) to the linear unit and compute the output o
- ✓ For each linear unit weight wi Do

- $\Delta w_i = \Delta w_i + \eta$ (t-o) xi
- For each linear unit weight wi Do
- $w_i = w_i + \Delta w_i$

MODES OF GRADIENT DESCENT

- Batch mode : gradient descent

  $w=w - \eta \nabla ED[w]$ over the entire data D

  $ED[w]=1/2\Sigma d(t_d-o_d)2$

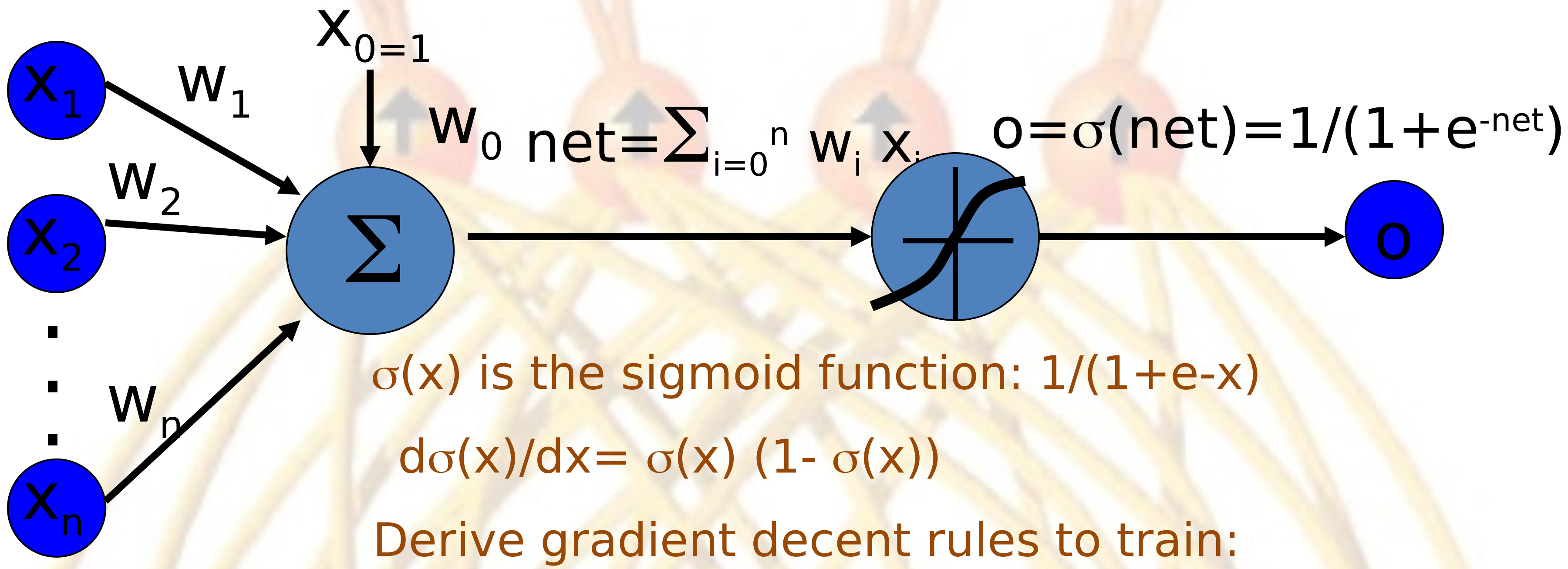
- Incremental mode: gradient descent

  $w=w - \eta \nabla E_d[w]$ over individual training examples d

  $Ed[w]=1/2 (t_d-o_d)2$

- Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if η is small enough.

SIGMOID ACTIVATION FUNCTION



$x_{0=1}$
x_1
x_2
x_n
w_1
w_2
w_n
w_0
Σ
$net=\Sigma_{i=0}^{n} w_i x_i$
$o=\sigma(net)=1/(1+e^{-net})$
o

$\sigma(x)$ is the sigmoid function: $1/(1+e-x)$

$d\sigma(x)/dx= \sigma(x)\ (1- \sigma(x))$

Derive gradient decent rules to train:

- one sigmoid function
  $\partial E/\partial w_i = -\Sigma d(td\text{-}od)\ od\ (1\text{-}od)\ xi$
- Multilayer networks of sigmoid units

backpropagation

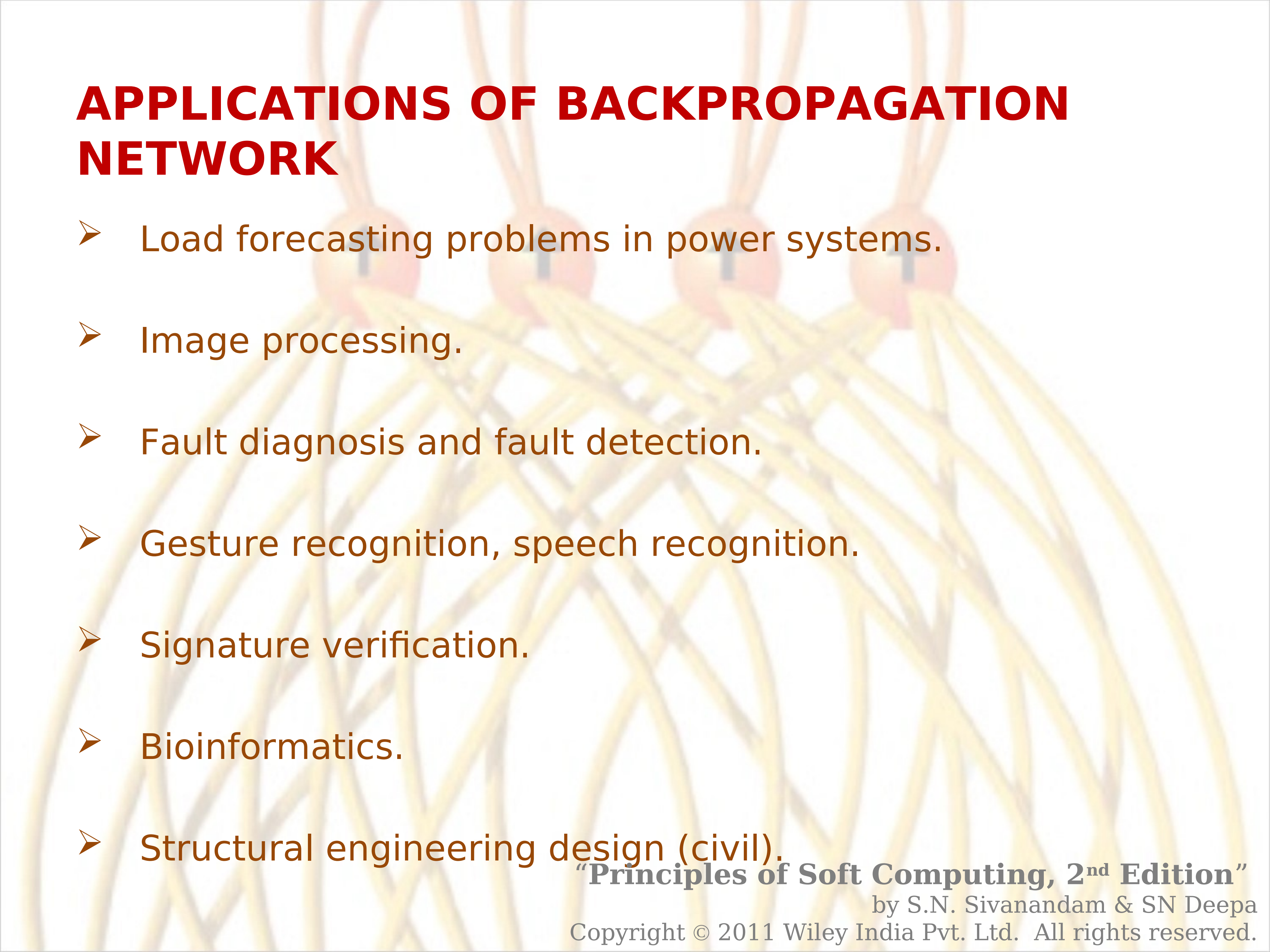
BACKPROPAGATION TRAINING ALGORITHM

- Initialize each wi to some small random value.
- Until the termination condition is met, Do
  - For each training example <(x1,…xn),t> Do
  - Input the instance (x1,….,xn) to the network and compute the network outputs ok
  - For each output unit k
    - $\delta k=ok(1-ok)(tk-ok)$
  - For each hidden unit h
    - $\delta h=oh(1-oh)\ \Sigma k\ wh,k\ \delta k$
  - For each network weight w,j Do
  - wi,j=wi,j+Δwi,j where
    - $\Delta wi,j= \eta\ \delta j\ xi,j$

BACKPROPAGATION

- Gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum -in practice often works well (can be invoked multiple times with different initial weights)
- Often include weight momentum term

  $$\Delta wi,j(t) = \eta\ \delta j\ xi,j + \alpha\ \Delta wi,j\ (t-1)$$
- Minimizes error training examples
- Will it generalize well to unseen instances (over-fitting)?
- Training can be slow typical 1000-10000 iterations (use Levenberg-Marquardt instead of gradient descent)

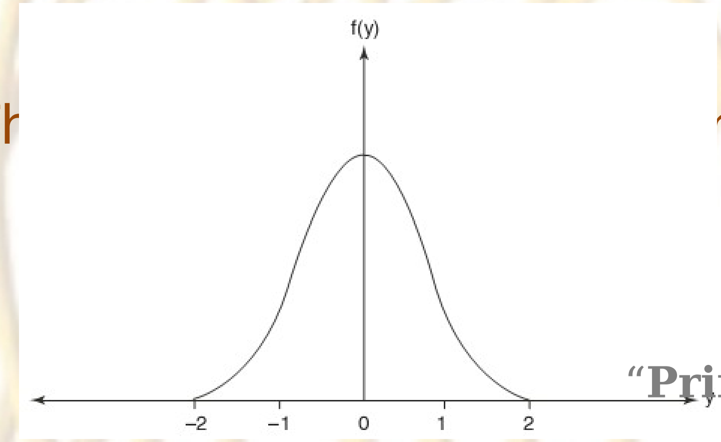


APPLICATIONS OF BACKPROPAGATION NETWORK

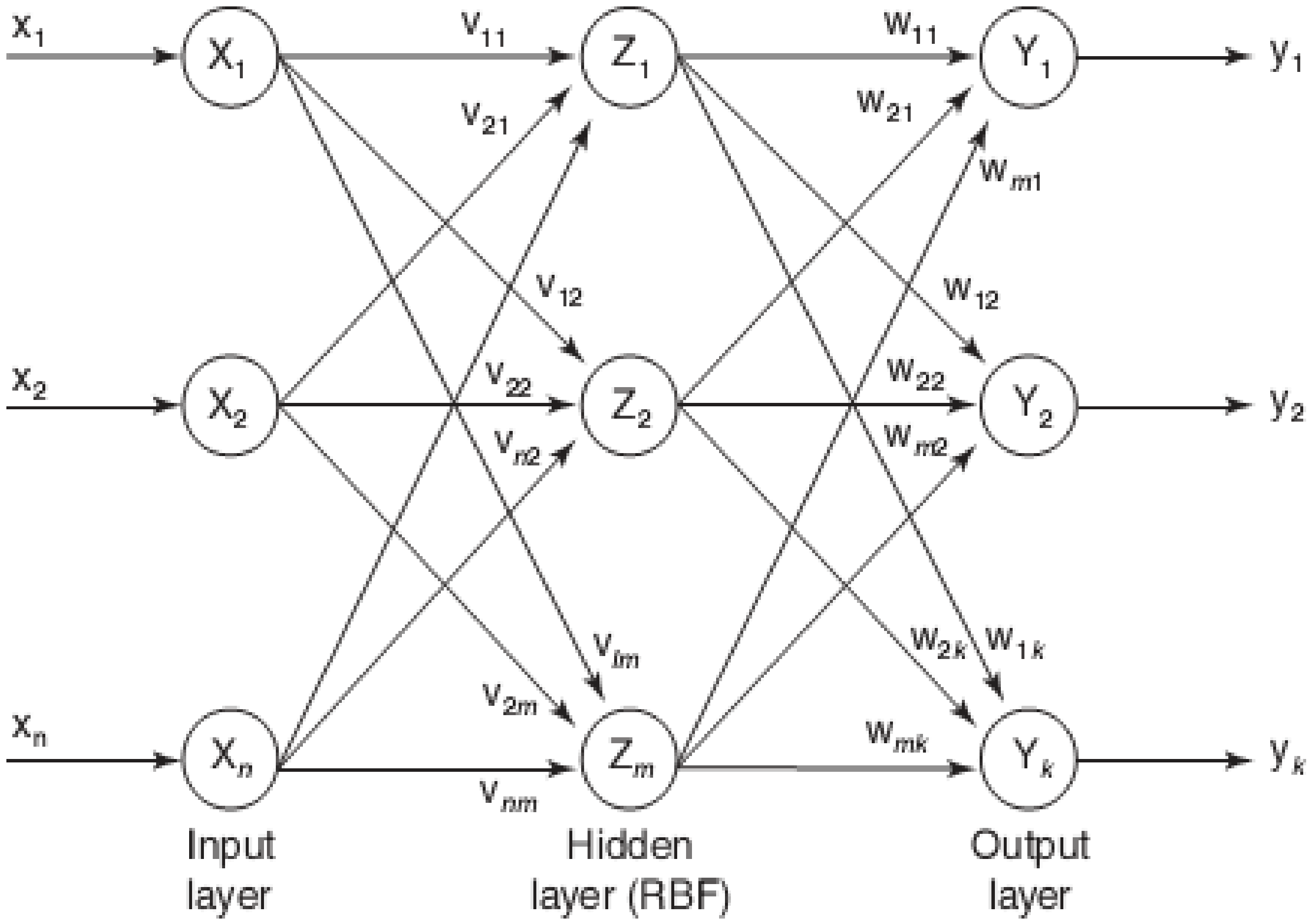
- Load forecasting problems in power systems.
- Image processing.
- Fault diagnosis and fault detection.
- Gesture recognition, speech recognition.
- Signature verification.
- Bioinformatics.
- Structural engineering design (civil).

RADIAL BASIS FUCNTION NETWORK

- The radial basis function (RBF) is a classification and functional approximation neural network developed by M.J.D. Powell.
- The network uses the most common nonlinearities such as sigmoidal and Gaussian kernel functions.
- The Gaussian functions are also used in regularization networks.
- Th ne $f(y) = e^{-y^2}$ as



RADIAL BASIS FUCNTION NETWORK



SUMMARY

This chapter discussed on the several supervised learning networks like

- Perceptron,
- Adaline,
- Madaline,
- Backpropagation Network,
- Radial Basis Function Network.

Apart from these mentioned above, there are several other supervised neural networks like tree neural networks, wavelet neural network, functional link neural network and so on.