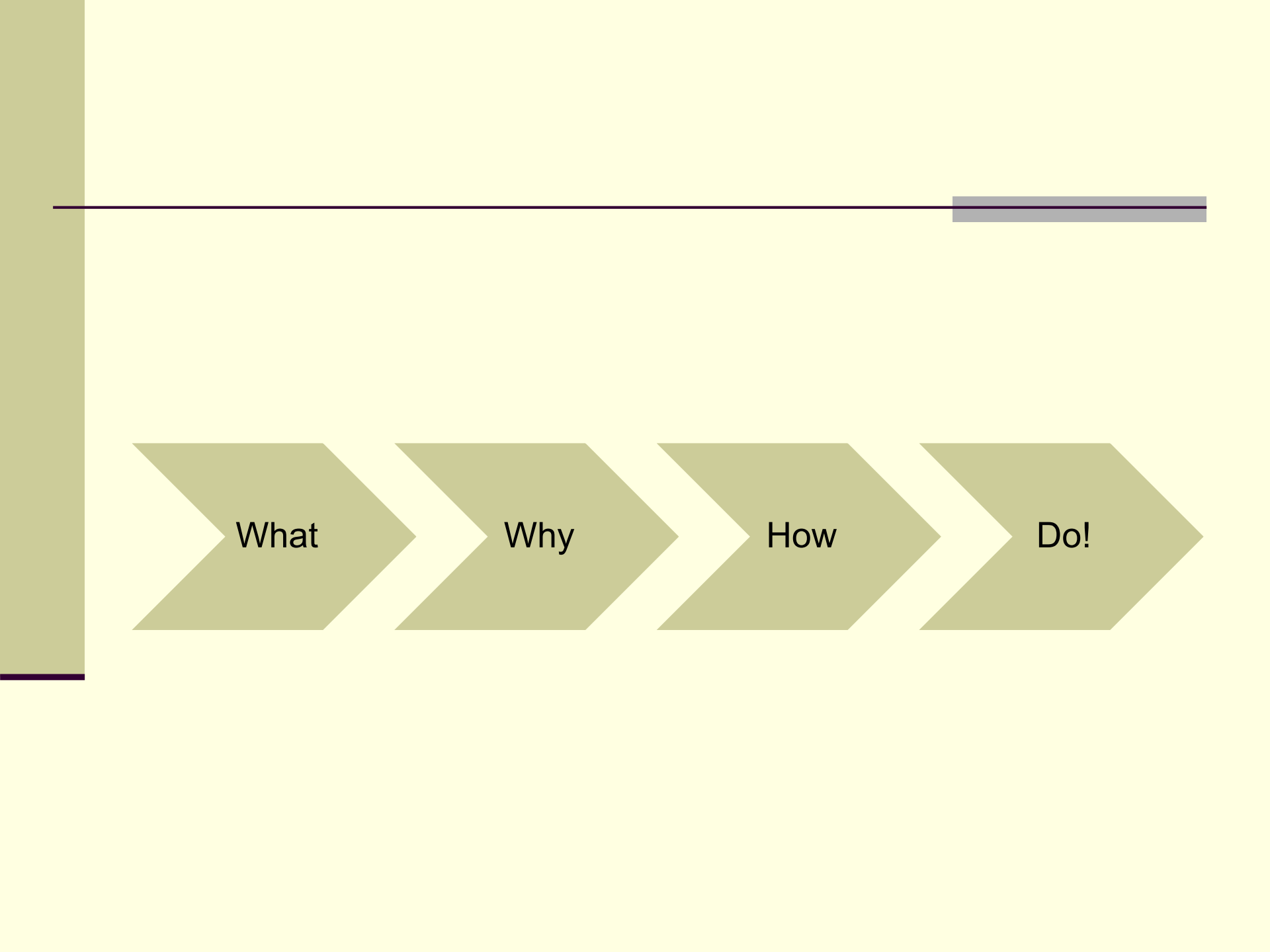


# Forecasting Stock Exchange value using Artificial Neural Network

Presented by: Amit Jena  
Supervisor: Mr. Mukul Priyadarshi



What

Why

How

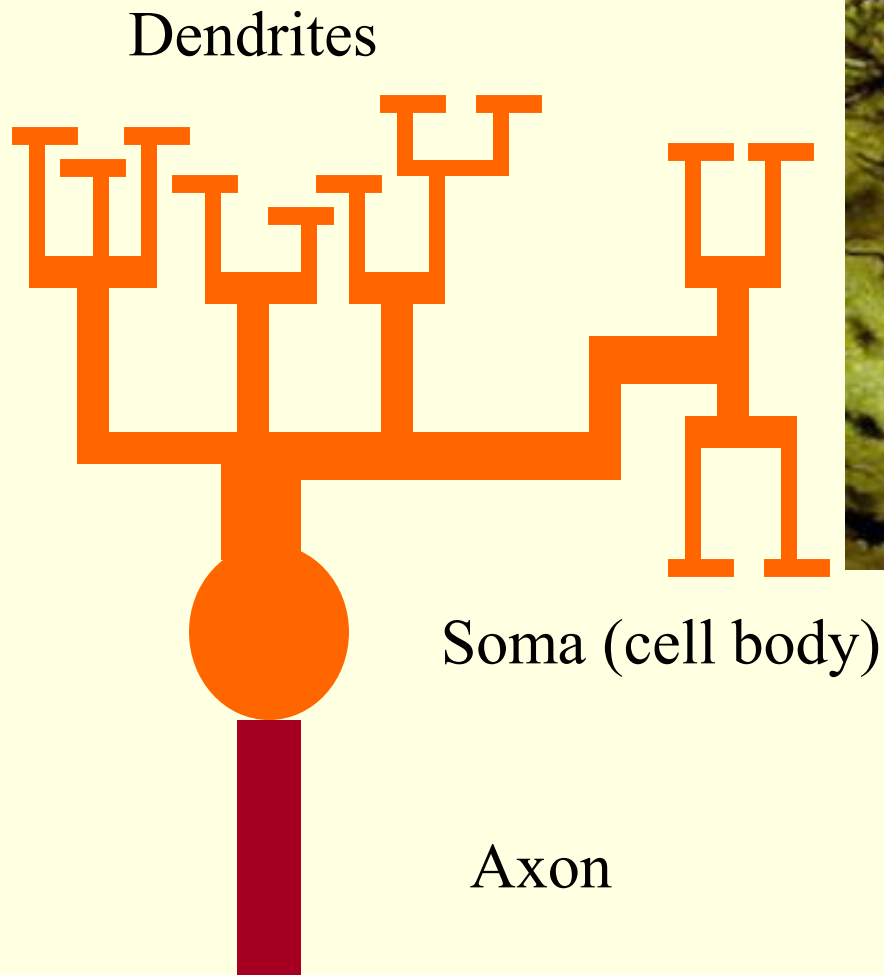
Do!



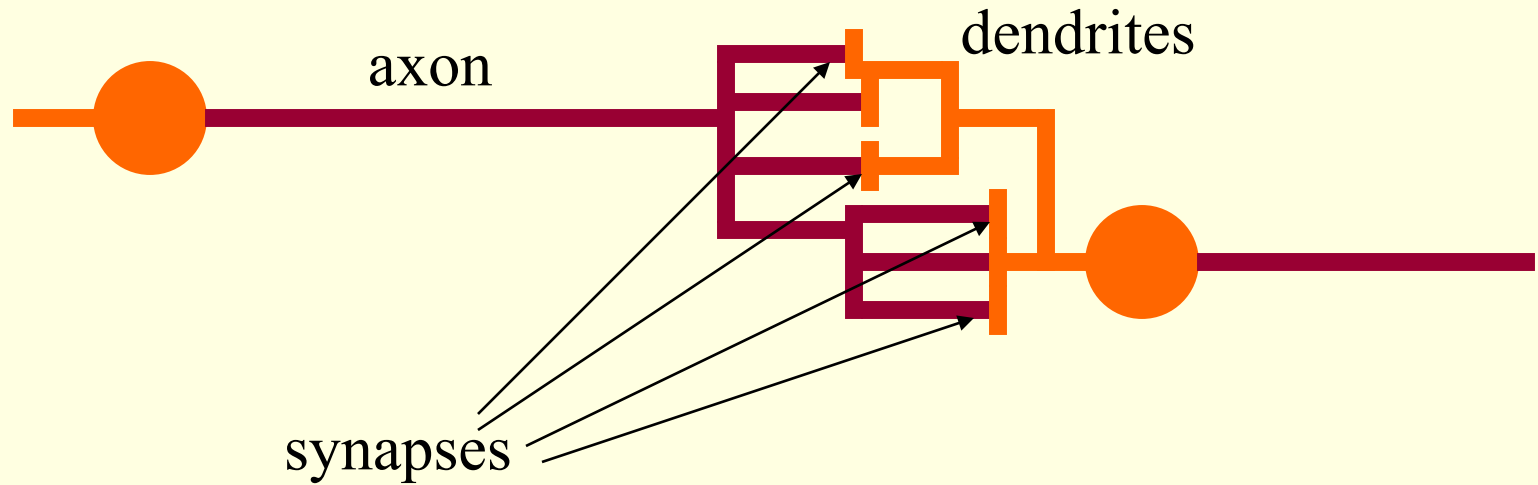
# **What is Artificial Neural Network?**

# Biological inspiration

---



# Biological inspiration



The information transmission happens at the synapses.

# Biological inspiration

---

The spikes travelling along the axon of the pre-synaptic neuron trigger the release of neurotransmitter substances at the synapse.

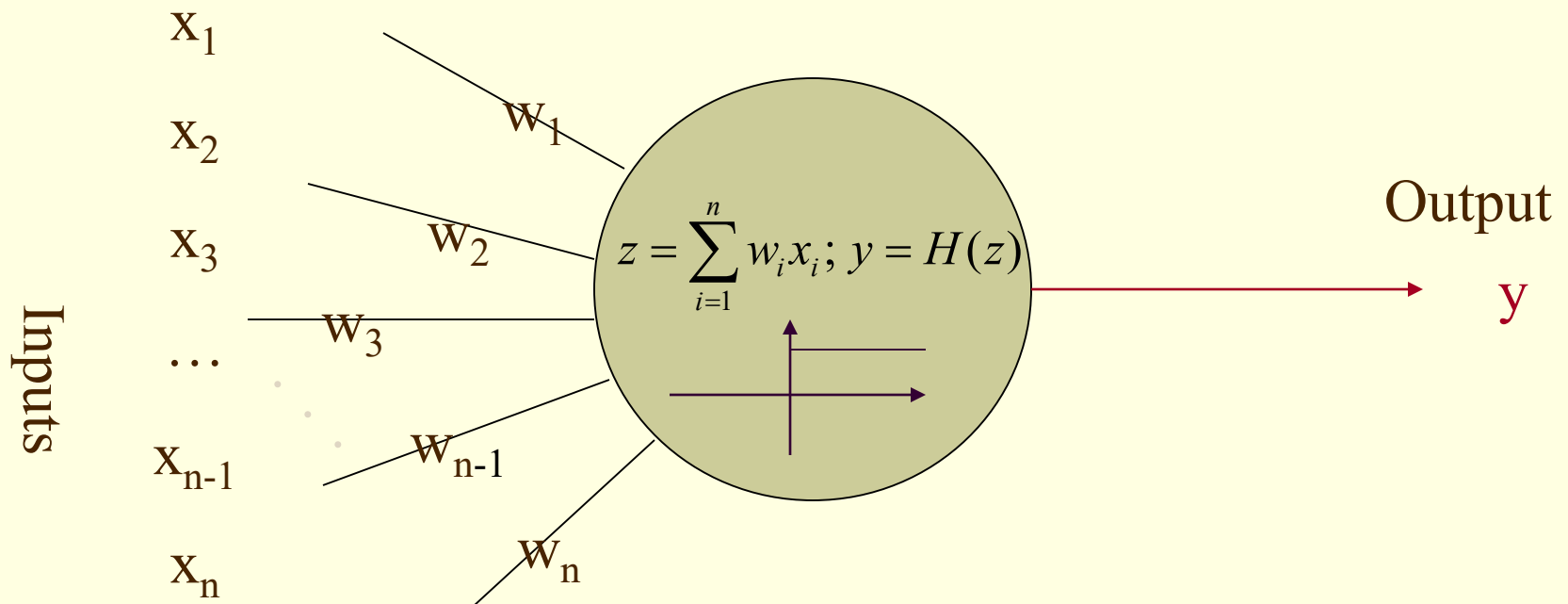
The neurotransmitters cause excitation or inhibition in the dendrite of the post-synaptic neuron.

The integration of the excitatory and inhibitory signals may produce spikes in the post-synaptic neuron.

The contribution of the signals depends on the strength of the synaptic connection.

# Artificial neurons

Neurons work by processing information. They receive and provide information in form of spikes.



The McCulloch-Pitts model

# Artificial neurons

---

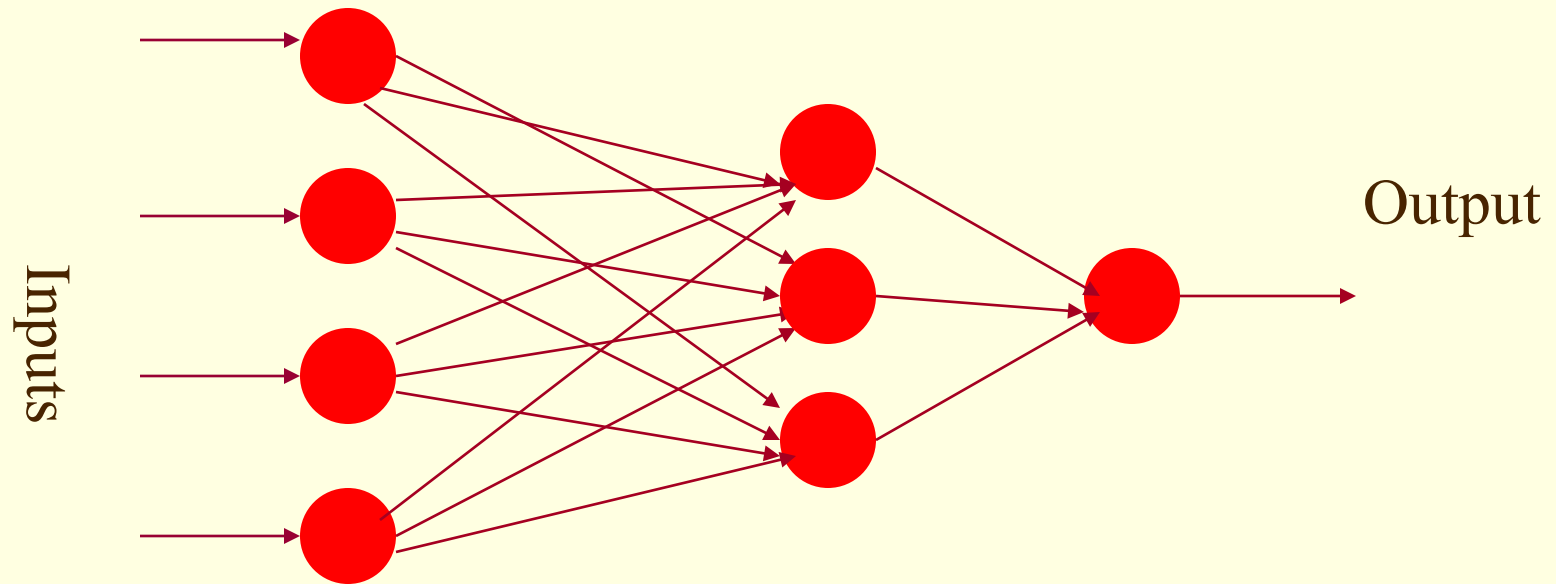
The McCulloch-Pitts model:

- spikes are interpreted as spike rates;
- synaptic strength are translated as synaptic weights;
  - excitation means positive product between the incoming spike rate and the corresponding synaptic weight;
  - inhibition means negative product between the incoming spike rate and the corresponding synaptic weight;



# Artificial neural networks

---



An artificial neural network is composed of many artificial neurons that are linked together according to a specific network architecture. The objective of the neural network is to transform the inputs into meaningful outputs.



Why to use ANN??





# Why to use ANN??

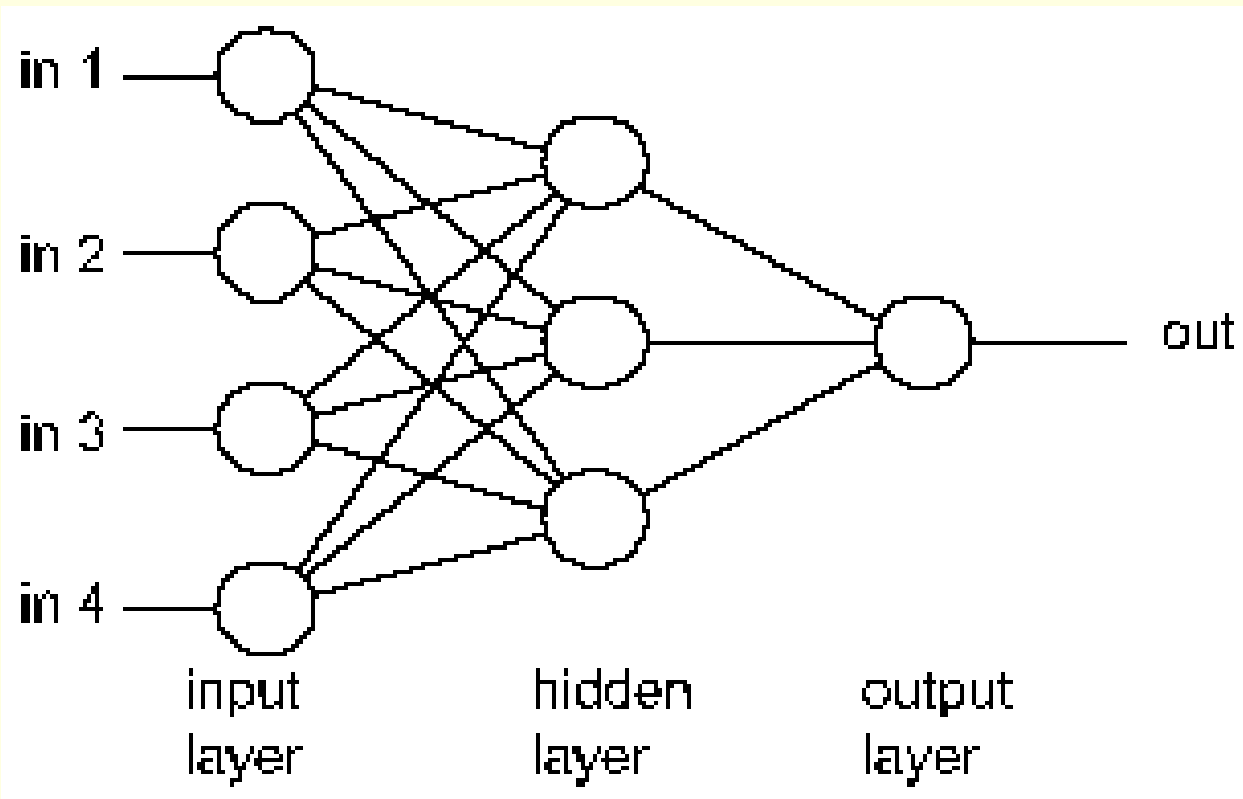
---

- Nonlinearity
- Input-Output mapping
- Adaptivity
- Evidential response
- Contextual Information
- Fault tolerance
- VLSI implementation
- Neurobiological Analogy

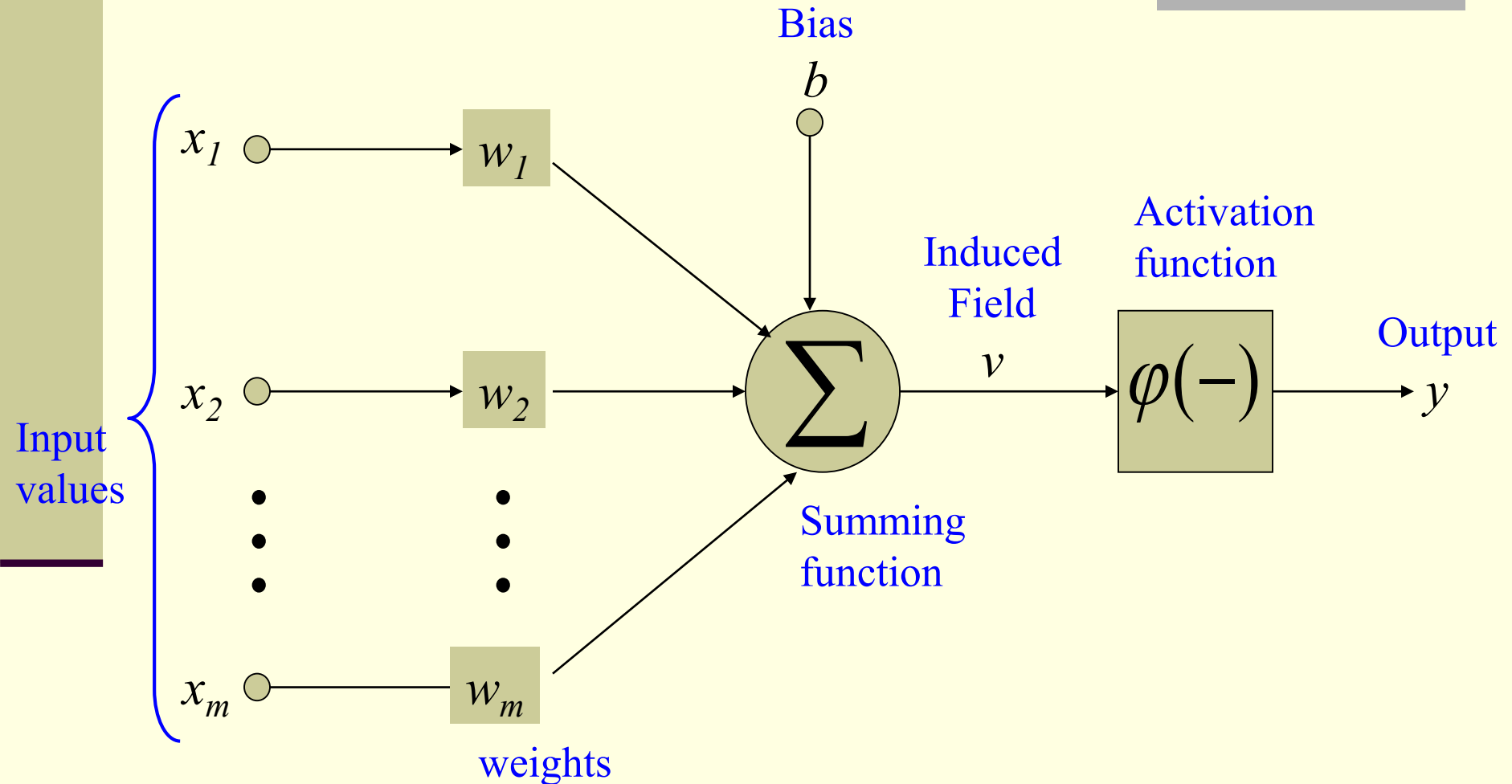


HOW do we do it?

# Basic ANN Architecture



# The Neuron Diagram



# Bias of a Neuron

- The bias  $b$  has the effect of applying a transformation to the weighted sum  $u$

$$v = u + b$$

- The bias is an external parameter of the neuron. It can be modeled by adding an extra input.
- $v$  is called **induced field** of the neuron

$$v = \sum_{j=0}^m w_j x_j$$

$$w_0 = b$$

# Neuron Models

- The choice of activation function  $\varphi$  determines the neuron model.

## Examples:

- step function: 
$$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > c \end{cases}$$

- sigmoid function:

$$\varphi(v) = z + \frac{1}{1 + \exp(-xv)}$$

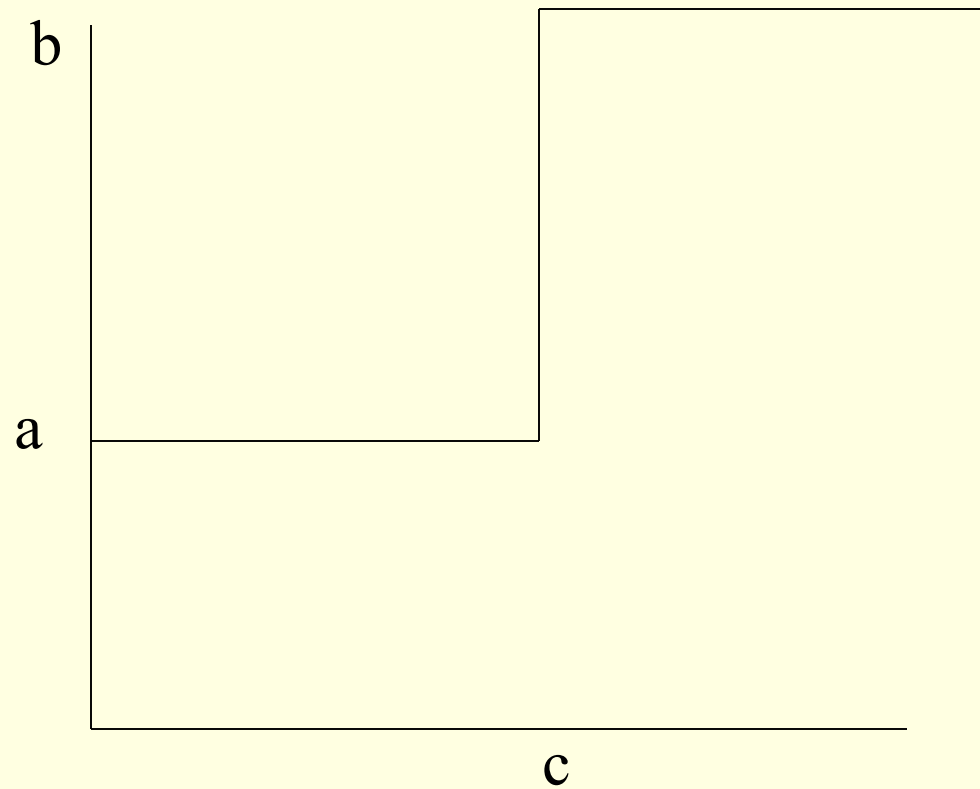
- Gaussian function:

$$\varphi(v) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{v - \mu}{\sigma}\right)^2\right)$$



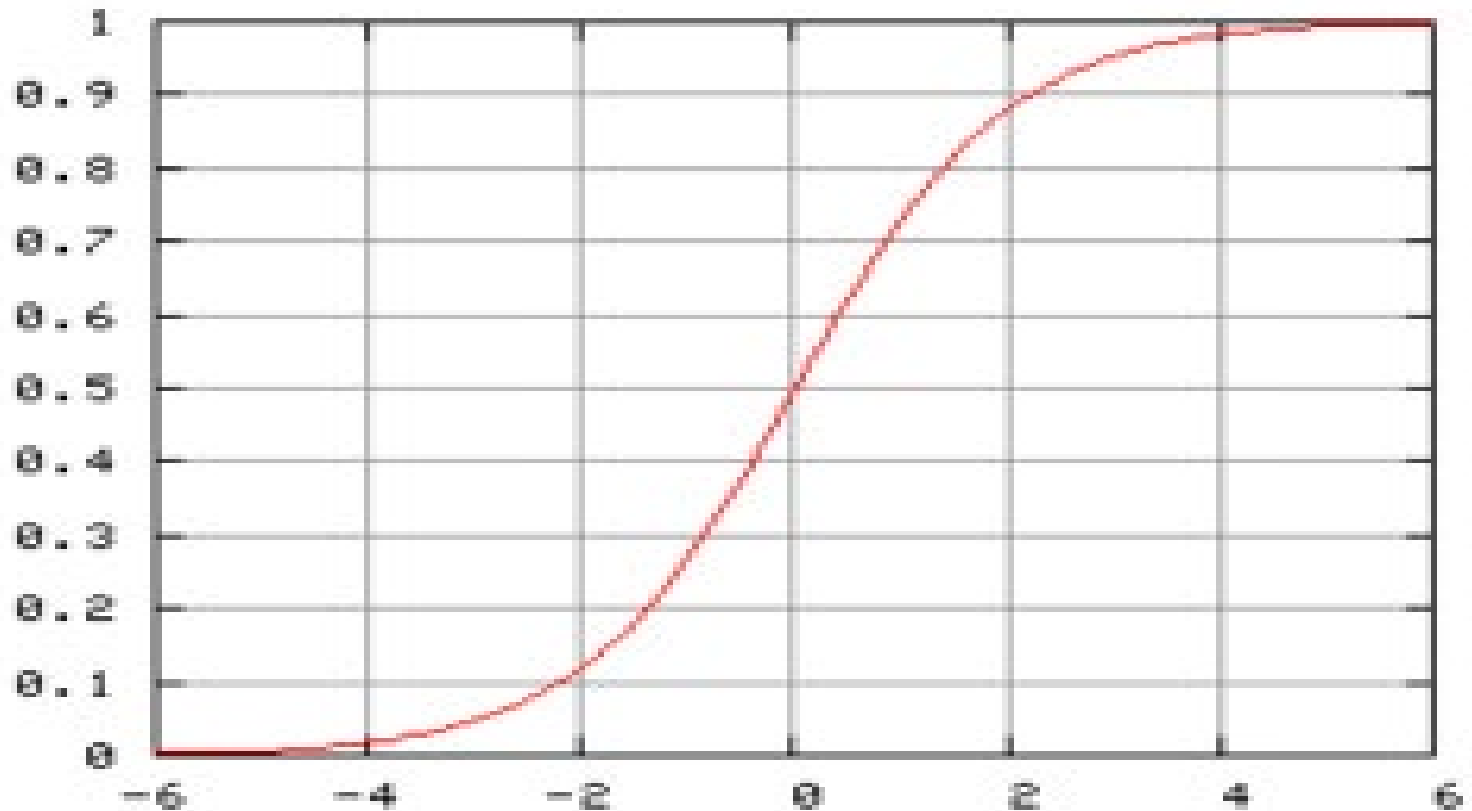
# Step Function

---



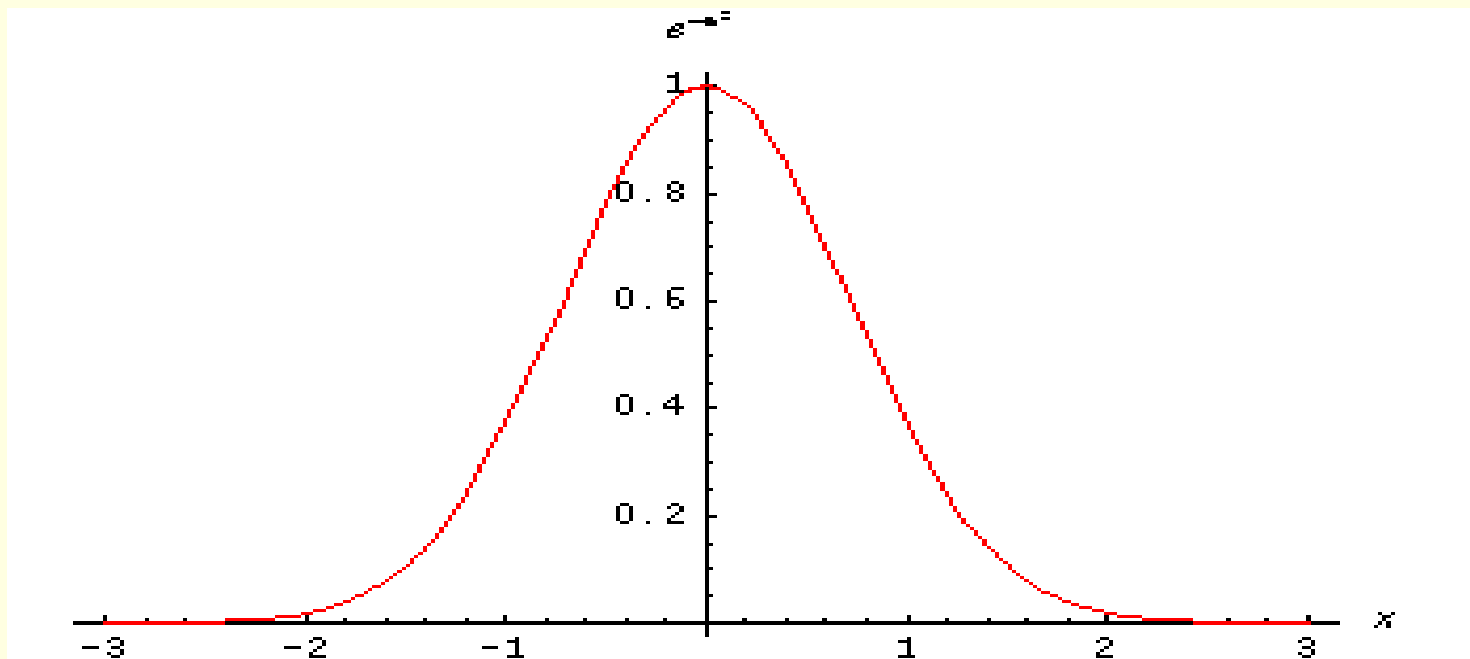
# Sigmoid function

---



- The **Gaussian function** is the probability function of the normal distribution. Sometimes also called the frequency curve.

$$f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2},$$



# Learning in the context of ANN

---

- Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place.

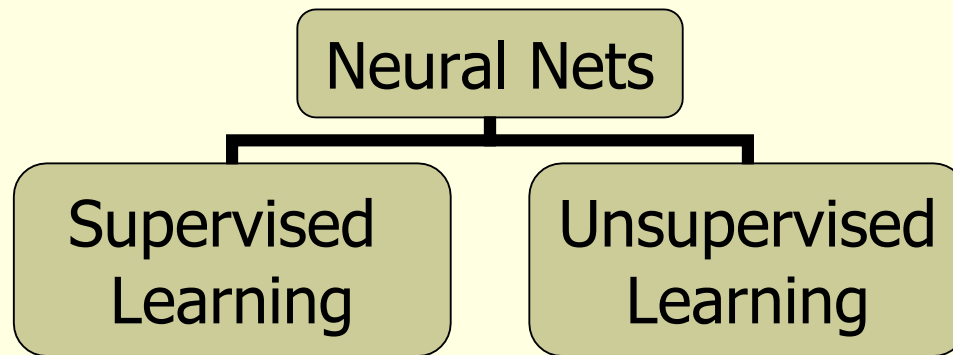
Stimulated

Undergoes  
changes

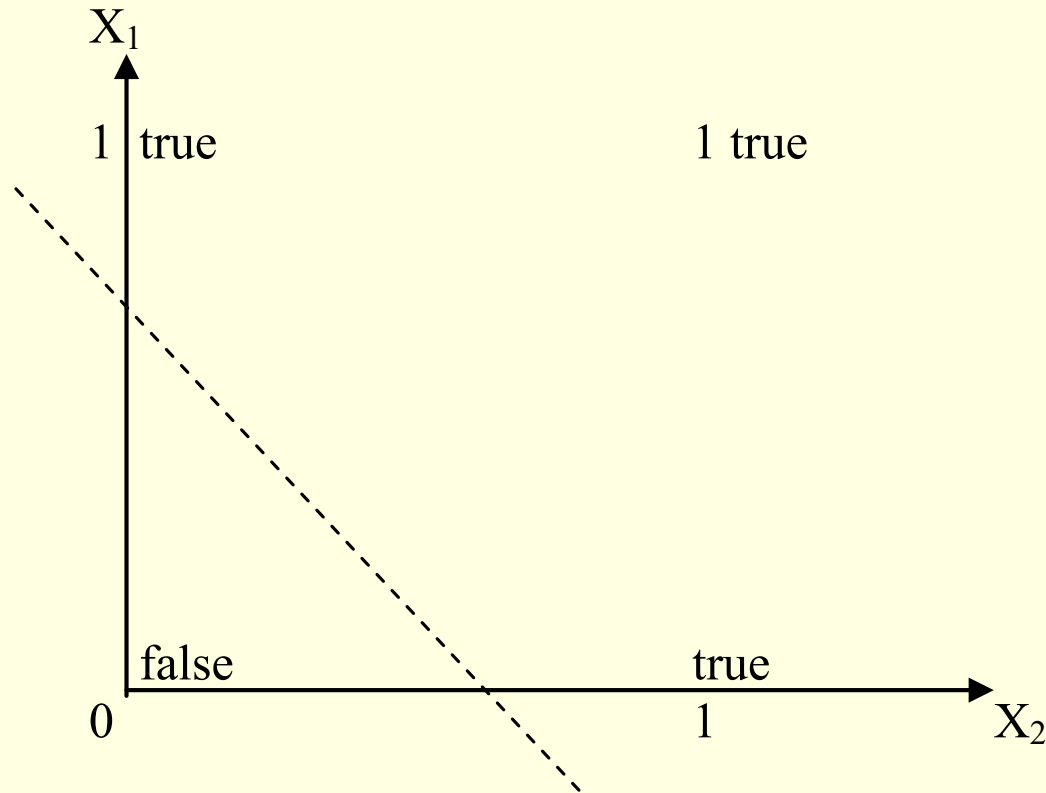
Responds in  
a new way

# Basic Types of Learning

---



# Boolean function OR – Linearly separable



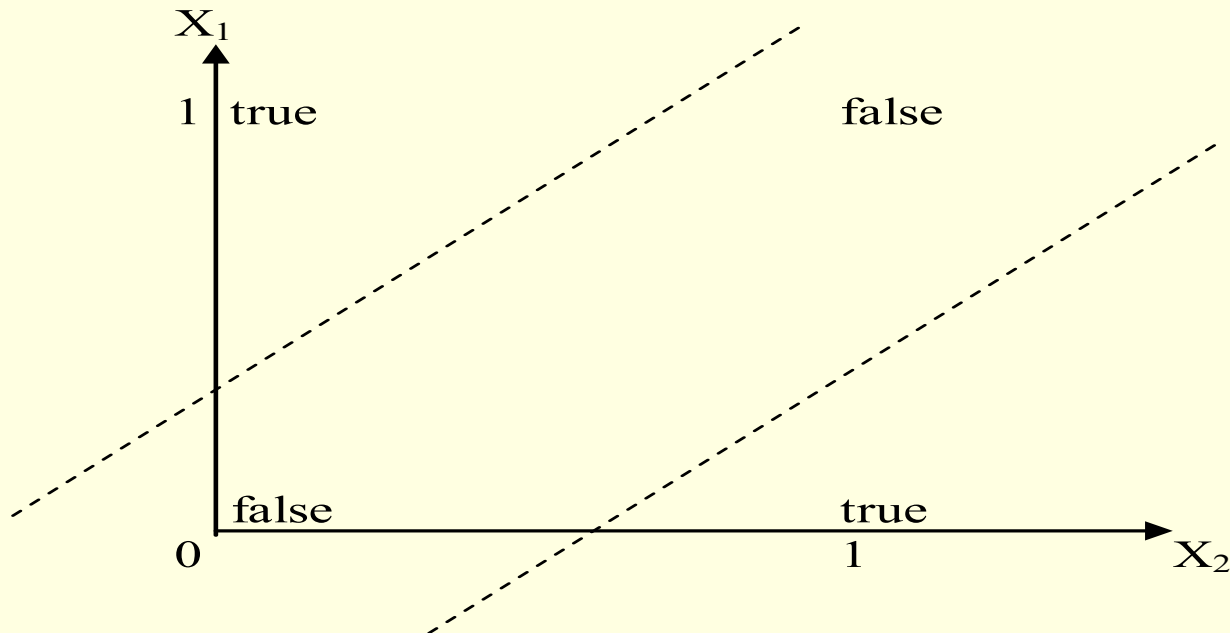
# XOR – Non linearly separable function

---

- A typical example of non-linearly separable function is the XOR that computes the logical **exclusive or**..
- This function takes two input arguments with values in  $\{0,1\}$  and returns one output in  $\{0,1\}$ ,
- Here 0 and 1 are encoding of the truth values **false** and **true**,
- The output is **true** if and only if the two inputs have different truth values.
- XOR is non linearly separable function which can not be modeled by perceptron.
- For such functions we have to use multi layer feed-forward network.

Input		Output
$X_1$	$X_2$	$X_1 \text{ XOR } X_2$
0	0	0
0	1	1
1	0	1
1	1	0

These two classes (true and false) cannot be separated using a line. Hence XOR is non linearly separable.





# FFNN for XOR

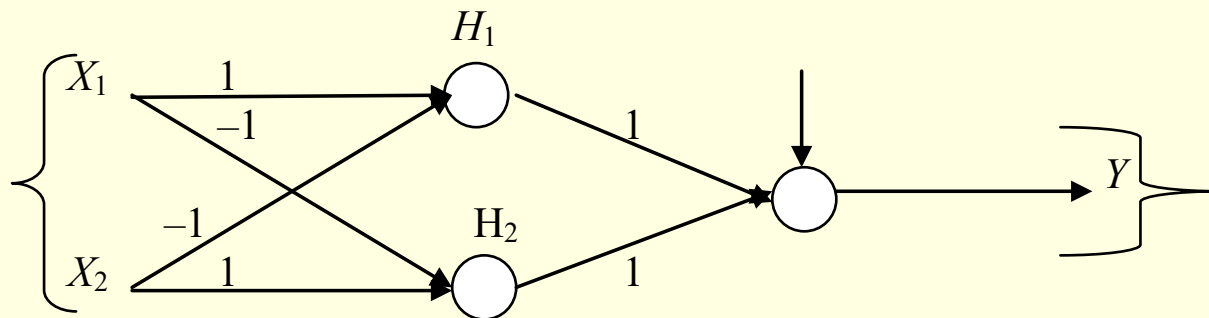
- The ANN for XOR has two hidden nodes that realizes this non-linear separation and uses the sign (step) activation function.
- Arrows from input nodes to two hidden nodes indicate the directions of the weight vectors  $(1,-1)$  and  $(-1,1)$ .
- The output node is used to combine the outputs of the two hidden nodes.

Input nodes

Hidden layer

Output layer

Output

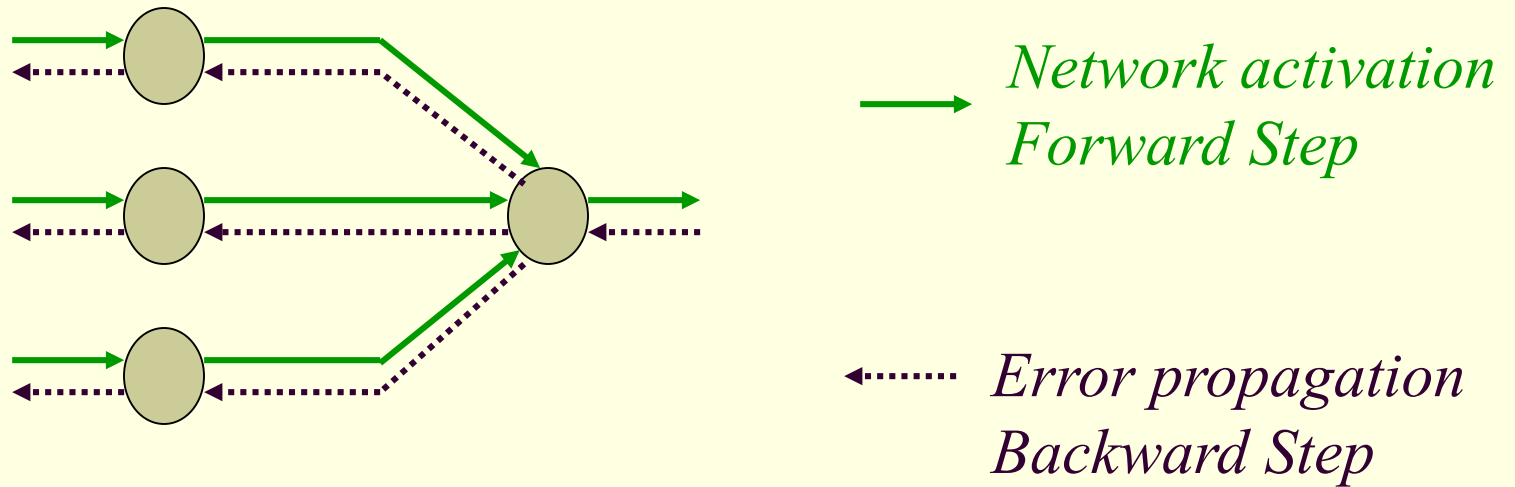


Inputs		Output of Hidden Nodes		Output	$X_1 \text{ XOR } X_2$
$X_1$	$X_2$	$H_1$	$H_2$	Node	
0	0	0	0	$-0.5 \rightarrow 0$	0
0	1	$-1 \rightarrow 0$	1	$0.5 \rightarrow 1$	1
1	0	1	$-1 \rightarrow 0$	$0.5 \rightarrow 1$	1
1	1	0	0	$-0.5 \rightarrow 0$	0

Since we are representing two states by 0 (false) and 1 (true), we will map negative outputs ( $-1$ ,  $-0.5$ ) of hidden and output layers to 0 and positive output ( $0.5$ ) to 1.

# Backpropagation

- Back-propagation training algorithm



- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.

# Total Mean Squared Error

---

- The error of output neuron  $k$  after the activation of the network on the  $n$ -th training example  $(x(n), d(n))$  is:

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

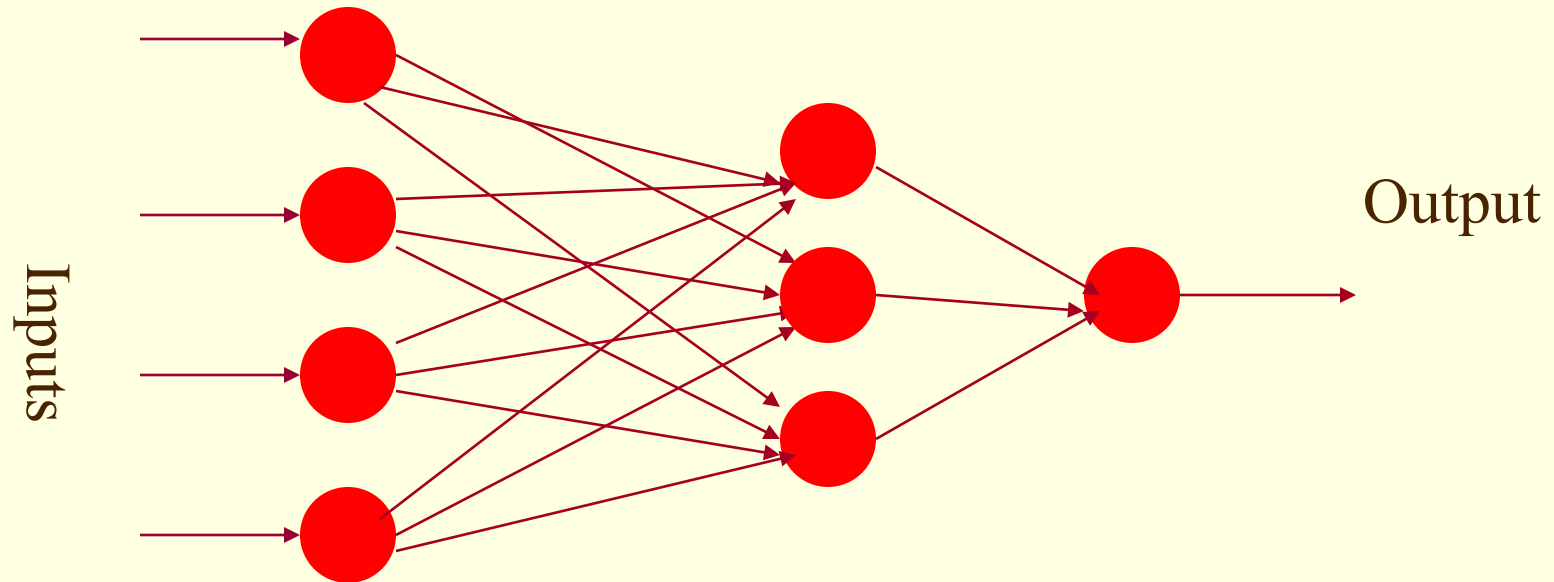
- The network error is the sum of the squared errors of the output neurons:

$$E(n) = \sum e_k^2(n)$$

- The total mean squared error is the average of the network errors of the training examples.

$$E_{AV} = \frac{1}{N} \sum_{n=1}^N E(n)$$

# Neural network mathematics



$$\begin{aligned} y_1^1 &= f(x_1, w_1^1) \\ y_2^1 &= f(x_2, w_2^1) \\ y_3^1 &= f(x_3, w_3^1) \\ y_4^1 &= f(x_4, w_4^1) \end{aligned} \quad y^1 = \begin{pmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \end{pmatrix} \quad \begin{aligned} y_1^2 &= f(y^1, w_1^2) \\ y_2^2 &= f(y^1, w_2^2) \\ y_3^2 &= f(y^1, w_3^2) \end{aligned} \quad y^2 = \begin{pmatrix} y_1^2 \\ y_2^2 \\ y_3^2 \end{pmatrix} \quad y_{Out} = f(y^2, w_1^3)$$

# Learning to approximate

---

Error measure:

$$E = \frac{1}{N} \sum_{t=1}^N (F(x_t; W) - y_t)^2$$

Rule for changing the synaptic weights:

$$\Delta w_i^j = -c \cdot \frac{\partial E}{\partial w_i^j} (W)$$

$$w_i^{j, new} = w_i^j + \Delta w_i^j$$

c is the learning parameter (usually a constant)

# Learning with a perceptron

---

Perceptron:  $y_{out} = w^T x$

Data:  $(x^1, y_1), (x^2, y_2), \dots, (x^N, y_N)$

Error:  $E(t) = (y(t)_{out} - y_t)^2 = (w(t)^T x^t - y_t)^2$

Learning:

$$w_i(t+1) = w_i(t) - c \cdot \frac{\partial E(t)}{\partial w_i} = w_i(t) - c \cdot \frac{\partial (w(t)^T x^t - y_t)^2}{\partial w_i}$$

$$w_i(t+1) = w_i(t) - c \cdot (w(t)^T x^t - y_t) \cdot x_i^t$$

$$w(t)^T x = \sum_{j=1}^m w_j(t) \cdot x_j^t$$

A perceptron is able to learn a linear function.

# Weight Update Rule

---

- The Backprop weight update rule is based on the gradient descent method:
  - It takes a step in the direction yielding the maximum decrease of the network error E.
  - This direction is the opposite of the gradient of E.
- Iteration of the Backprop algorithm is usually terminated when the sum of squares of errors of the output values for all training data in an epoch is less than some threshold such as 0.01

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$



# Stopping criterions

---

- Total mean squared error change:
  - Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range  $[0.1, 0.01]$ ).
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used then the part of the training set used for testing the network generalization will not be used for updating the weights.

# NN DESIGN ISSUES

---

- Data representation
- Network Topology
- Network Parameters
- Training
- Validation

# Data Representation

---

- Data representation depends on the problem.
- In general ANNs work on continuous (real valued) attributes. Therefore symbolic attributes are encoded into continuous ones.
- Attributes of different types may have different ranges of values which affect the training process.
- Normalization may be used, like the following one which scales each attribute to assume values between 0 and 1.

$$x_i = \frac{x_i - \min_i}{\max_i - \min_i}$$

for each value  $x_i$  of  $i^{\text{th}}$  attribute,  $\min_i$  and  $\max_i$  are the minimum and maximum value of that attribute over the training set.

# Network Topology

---

- The number of layers and neurons depend on the specific task.
- In practice this issue is solved by trial and error.
- Two types of adaptive algorithms can be used:
  - start from a large network and successively remove some neurons and links until network performance degrades.
  - begin with a small network and introduce new neurons until performance is satisfactory.

# Network parameters

---

- How are the weights initialized?
- How is the learning rate chosen?
- How many hidden layers and how many neurons?
- How many examples in the training set?

# Initialization of weights

- In general, initial weights are randomly chosen, with typical values between -1.0 and 1.0 or -0.5 and 0.5.
- If some inputs are much larger than others, random initialization may bias the network to give much more importance to larger inputs.
- In such a case, weights can be initialized as follows:

$$W_{ij} = \pm \frac{1}{2N} \sum_{i=1, \dots, N} \frac{1}{|x_i|}$$

For weights from the input to the first layer

$$W_{jk} = \pm \frac{1}{2N} \sum_{i=1, \dots, N} \frac{1}{\varphi(\sum w_{ij} x_i)}$$

For weights from the first to the second layer

# Choice of learning rate

---

- The right value of  $\eta$  depends on the application.
- Values between 0.1 and 0.9 have been used in many applications.
- Other heuristics is that adapt  $\eta$  during the training.

# Training

---

- Rule of thumb:
  - the number of training examples should be at least five to ten times the number of weights of the network.
- Other rule:

$$N > \frac{|W|}{(1 - a)}$$

|W| = number of weights  
a = expected accuracy on test set



# Convergence and Local Minima

---

- The error surface for multilayer networks may contain many different local minima.
- BP guarantees to converge local minima only.
- BP is a highly effective function approximator in practice.
  - The local minima problem found to be not severe in many applications.

## □ Notes

- Gradient descent over the complex error surfaces represented by ANNs is still poorly understood
- No methods are known to predict certainly when local minima will cause difficulties.
- We can use only heuristics for avoiding local minima.

# Heuristics for Alleviating the Local Minima Problem

---

- Add a momentum term to the weight-update rule.
- Train multiple networks using the same data, but initializing each network with different random weights.
  - Select the best network w.r.t the validation set
  - Make a committee of networks

# Why BP Works in Practice?

---

- Weights are initialized to values near zero.
- Early gradient descent steps will represent a very smooth function (approximately linear).
  - The sigmoid function is almost linear when the total input (weighted sum of inputs to a sigmoid unit) is near 0.
- The weights gradually move close to the global minimum.
- As weights grow in the later stage of learning, they represent highly nonlinear network functions.
- Gradient steps in this later stage move toward local minima in this region, which is acceptable.

# Hidden Layer Representations

---

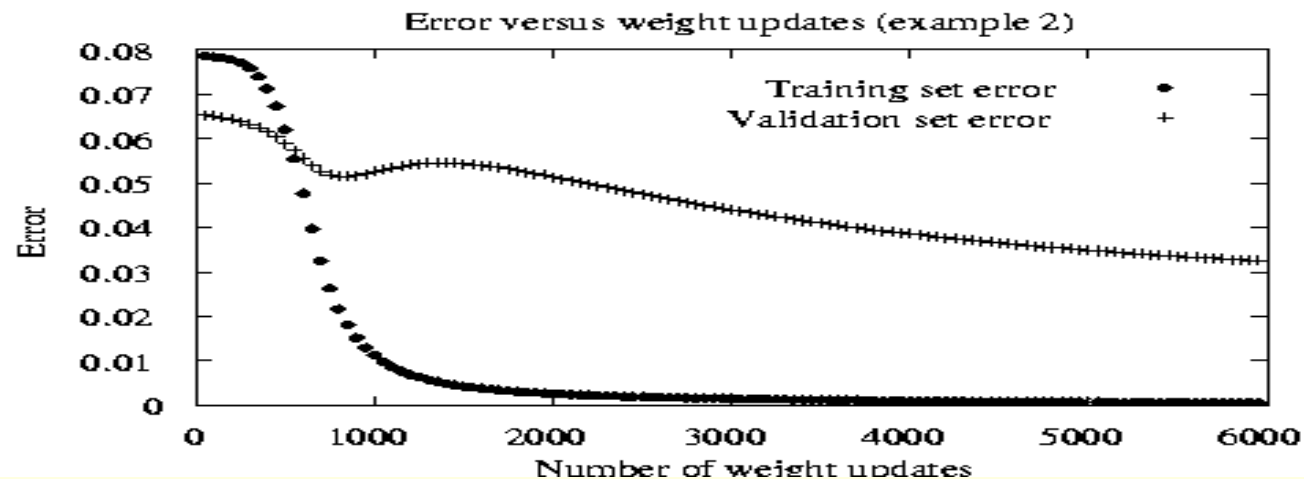
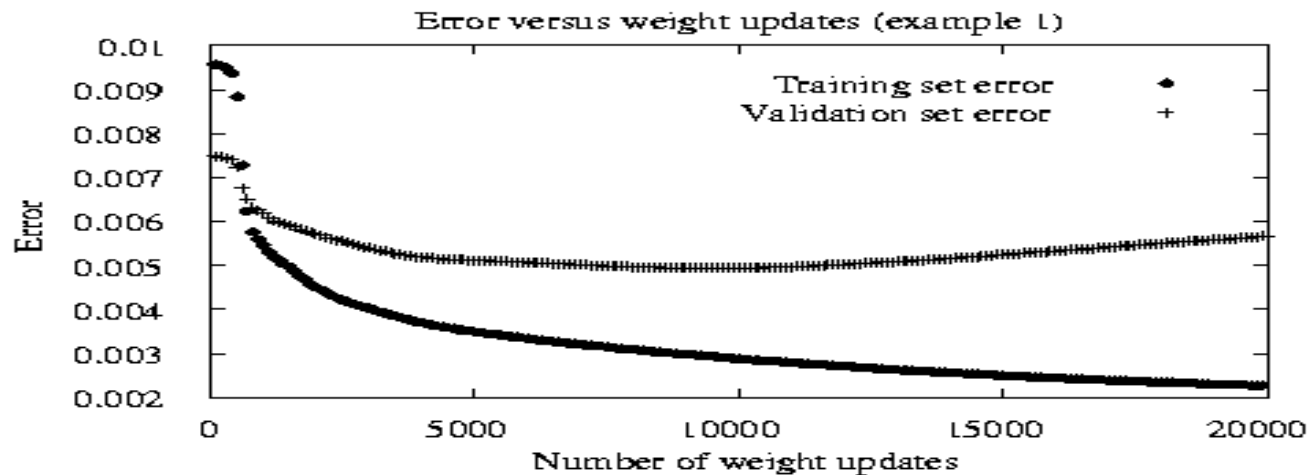
- BP has an ability to discover useful intermediate representations at the hidden unit layers inside the networks which capture properties of the input spaces that are most relevant to learning the target function.
- When more layers of units are used in the network, more complex features can be invented.
- But the representations of the hidden layers are very hard to understand for human.

# Generalization and Overfitting

---

- Continuing training until the training error falls below some predetermined threshold is a poor strategy since BP is susceptible to overfitting.
  - Need to measure the generalization accuracy over a validation set (distinct from the training set).
- Two different types of overfitting
  - Generalization error first decreases, then increases, even the training error continues to decrease.
  - Generalization error decreases, then increases, then decreases again, while the training error continues to decrease.

# Two Kinds of Overfitting Phenomena



# Techniques for Overcoming the Overfitting Problem

---

- Weight decay
  - Decrease each weight by some small factor during each iteration.
  - This is equivalent to modifying the definition of  $E$  to include a penalty term corresponding to the total magnitude of the network weights.
  - The motivation for the approach is to keep weight values small, to bias learning against complex decision surfaces.



---

- $k$ -fold cross-validation

- Cross validation is performed  $k$  different times, each time using a different partitioning of the data into training and validation sets
- The result are averaged after  $k$  times cross validation.



Now for some Practical Applications !



# We have used..

---

- Number of layers: 2 (i.e. 1 hidden layer with 20 neurons)
- Transfer function: Hyperbolic tangent sigmoid
- Network Type: Feed-forward network with backpropagation
- Adaption Learning functions: Gradient descent weight and bias learning function (learngd)
- Performance functions: Mean squared normalized error performance function (MSE)
- Training functions: Gradient descent backpropagation (traingd) and Levenberg-Marquardt
- backpropagation (trainlm)

---

■ It has two matrices:

1. stockInputs of order 16 x 330
2. stockTargets of order 1 x 330

# The work flow for the problem has seven primary steps:

---

- 1. Collect data
- 2. Create the network
- 3. Congure the network
- 4. Initialize the weights and biases
- 5. Train the network
- 6. Validate the network
- 7. Use the network

## The 16 input data values are the following:

---

- 1. CONSUMER DURABLES
- 2. CAPITAL GOODS
- 3. AUTO
- 4. BANKEX
- 5. FMCG
- 6. HEALTH CARE
- 7. IT
- 8. METAL
- 9. OIL and GAS
- 10. POWER
- 11. PSU
- 12. REALTY
- 13. TECK
- 14. USD TO INR
- 15. GBP TO INR
- 16. EURO TO INR



# Matlab Implementation.