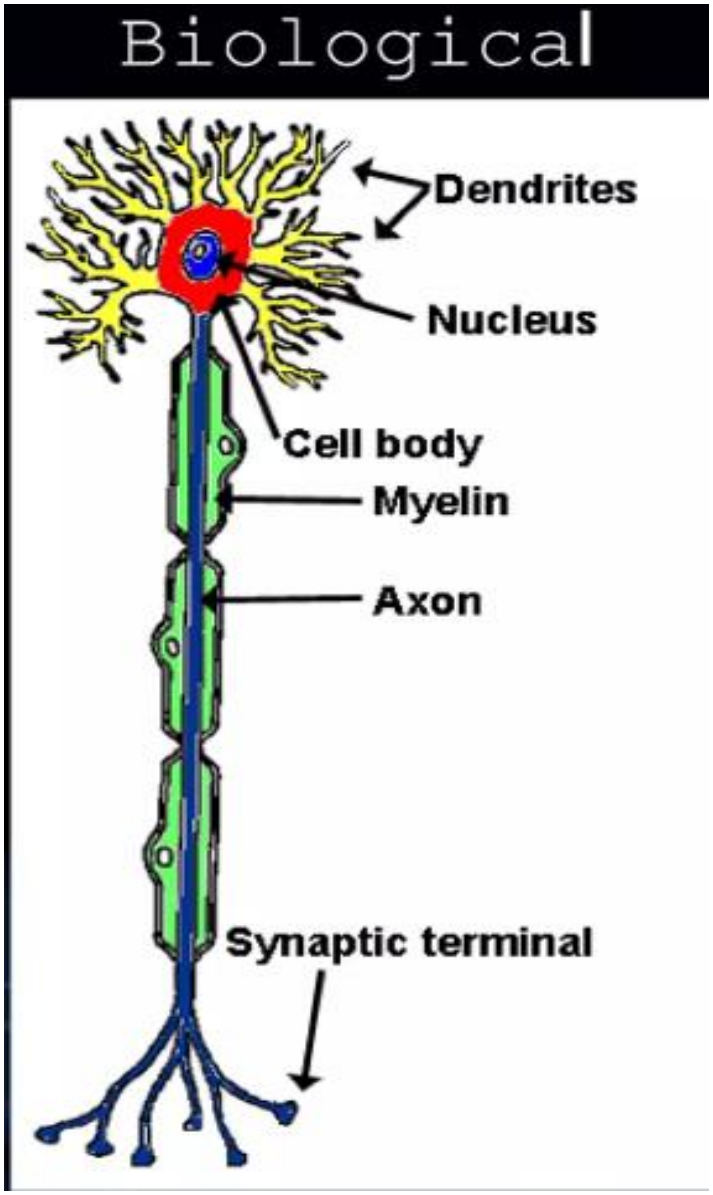
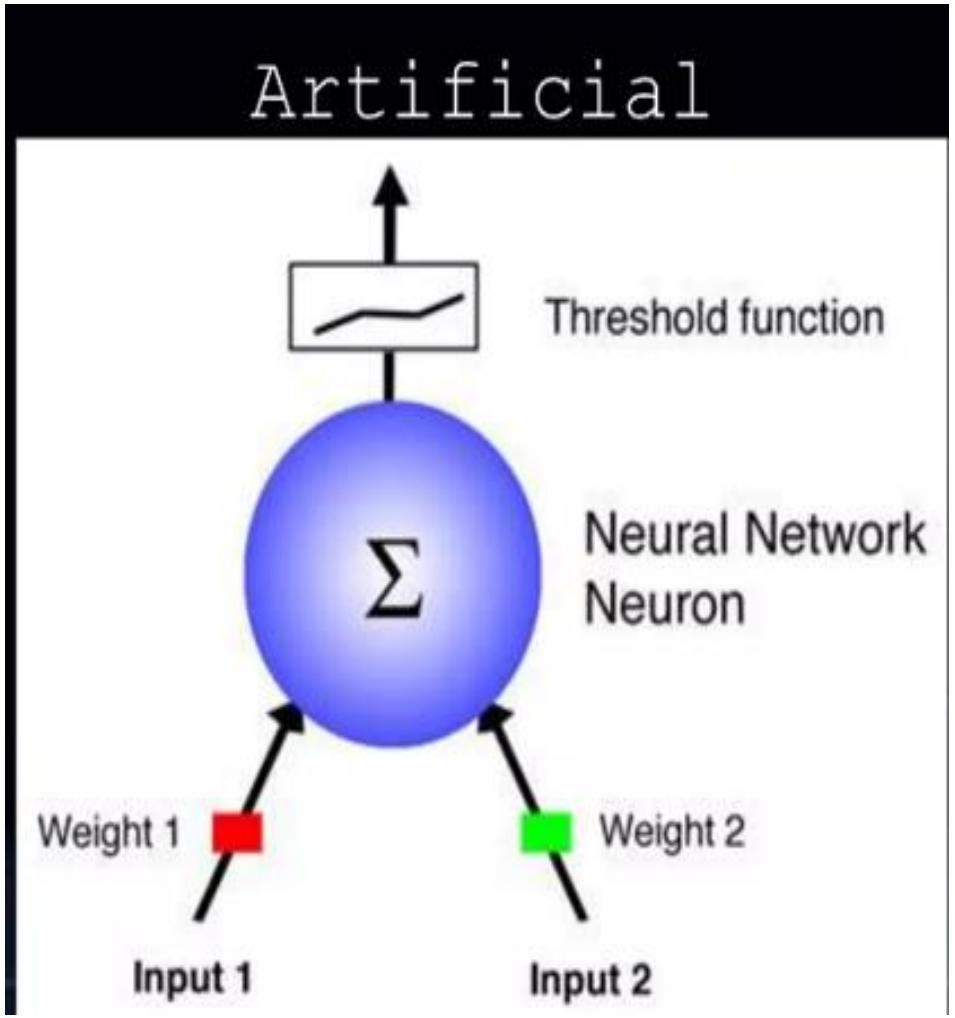
Neural Networks

Biological vs Artificial Neuron





Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Neuron in ANNs tend to have fewer connections than biological neurons.
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.

Contd..

- An Artificial Neural Network is specified by:
  - **neuron model** : the information processing unit of the NN,
  - **an architecture** : a set of neurons and links connecting neurons. Each link has a weight,
  - **a learning algorithm** : used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.
- The aim is to obtain a NN that is trained and generalizes well.
- It should behaves correctly on new instances of the learning task.

Neuron

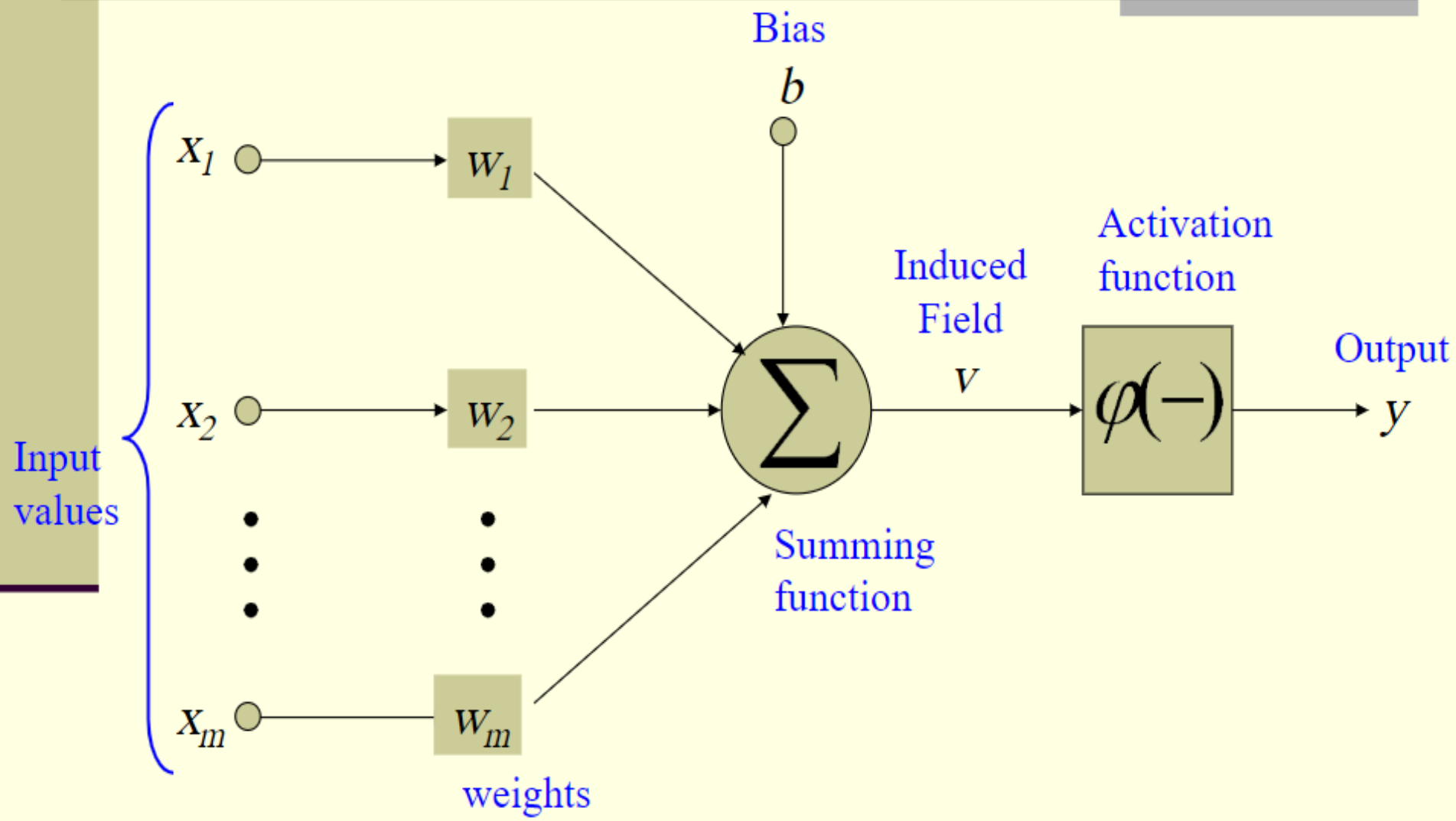
- The neuron is the basic information processing unit of a NN. It consists of:
  1. A set of links, describing the neuron inputs, with weights W_1, W_2, …, W_m
  2. An adder function (linear combiner) for computing the weighted sum of the inputs:
     (real numbers)

$$u = \sum_{j=1}^{m} w_j x_j$$

  3. Activation function φ for limiting the amplitude of the neuron output. Here 'b' denotes bias.

$$y = \varphi(u + b)$$

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 φ determines the neuron model.

Examples:

- step function:

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

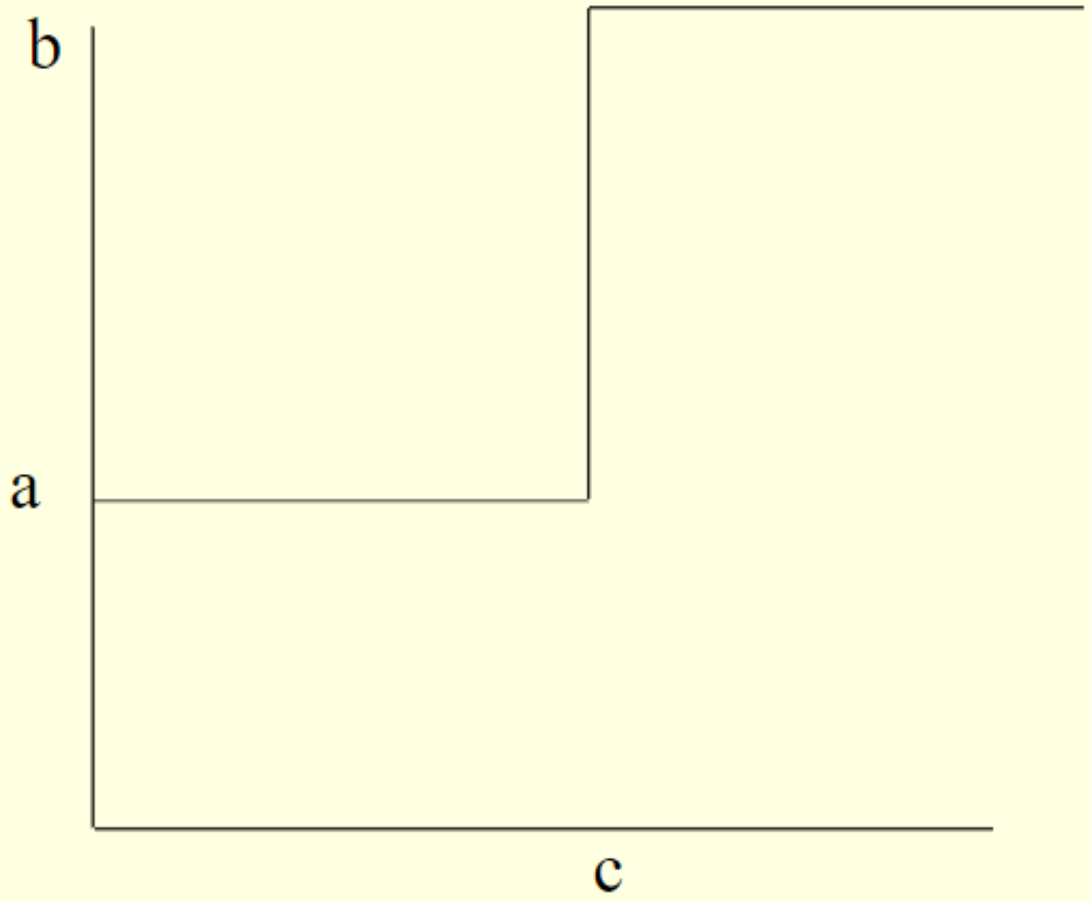
- ramp function:

$$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > d \\ a + ((v - c)(b - a)/(d - c)) & \text{otherwise} \end{cases}$$

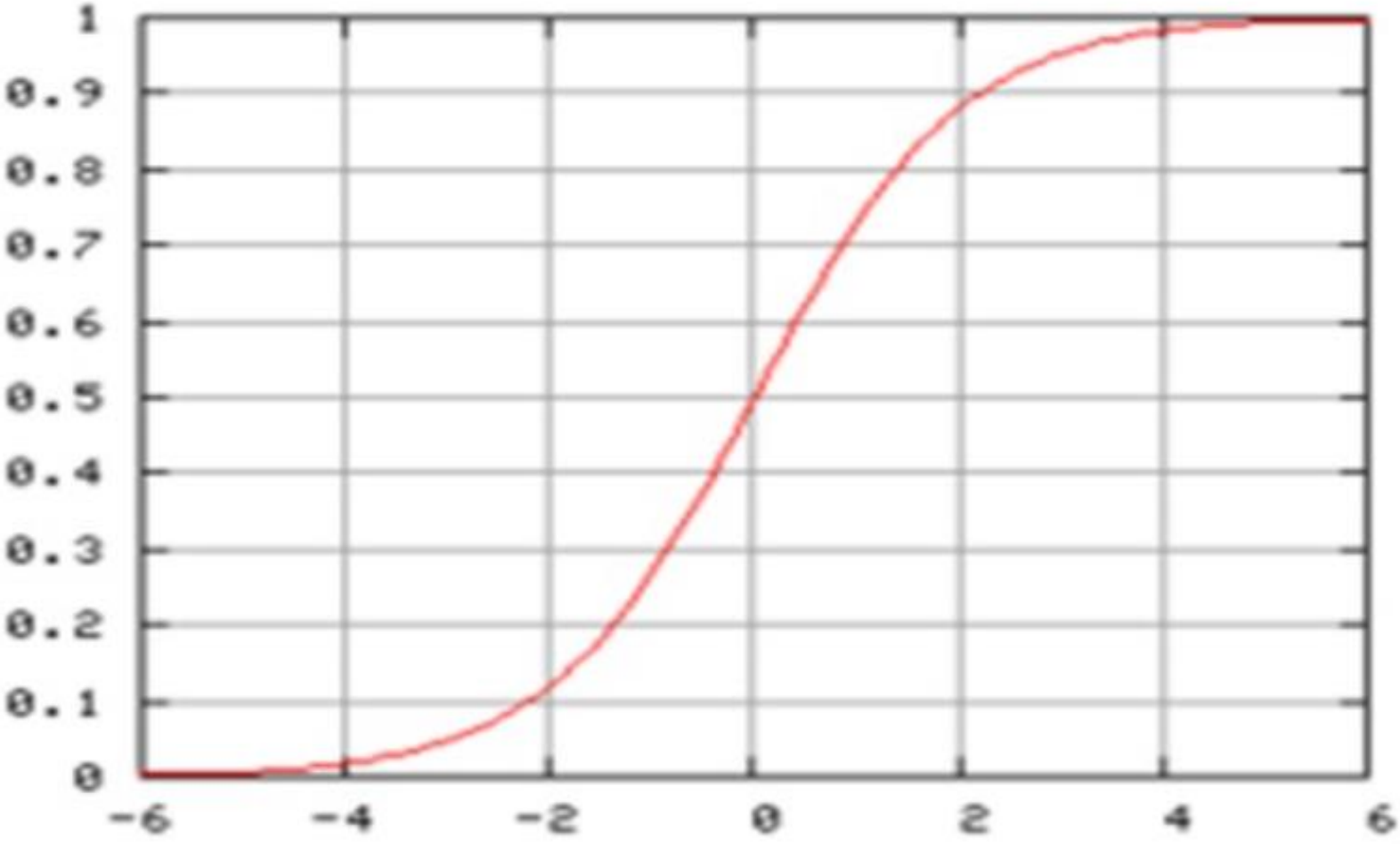
- sigmoid function with z,x,y parameters

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

Step Function



Sigmoid function

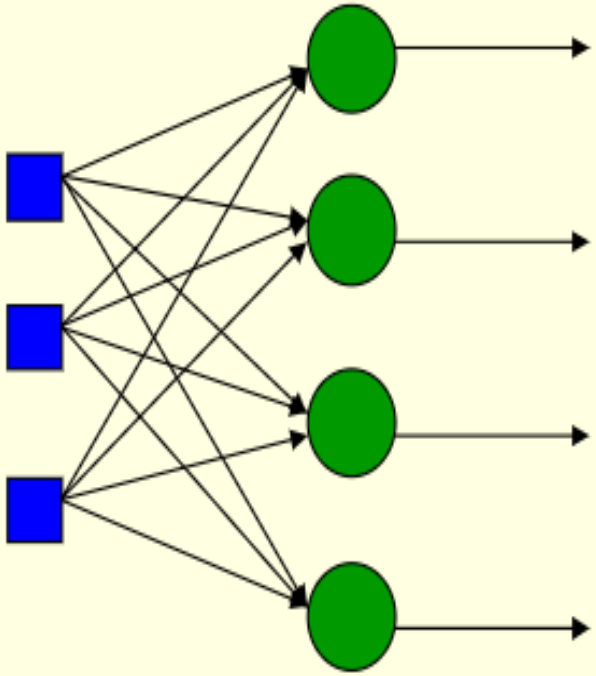


Network Architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

Input layer of source nodes



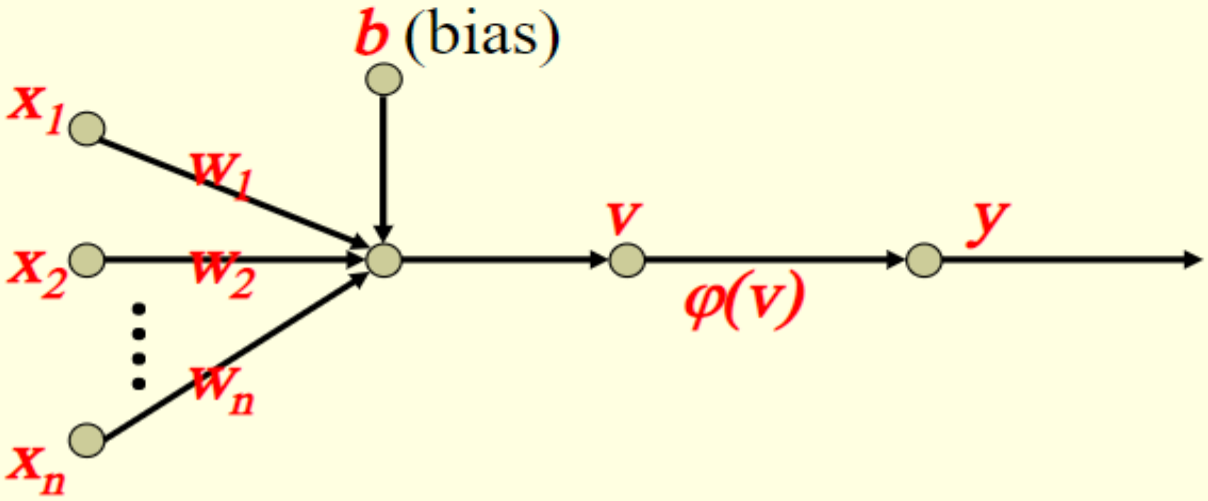
Output layer of neurons

Perceptron: Neuron Model

(Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

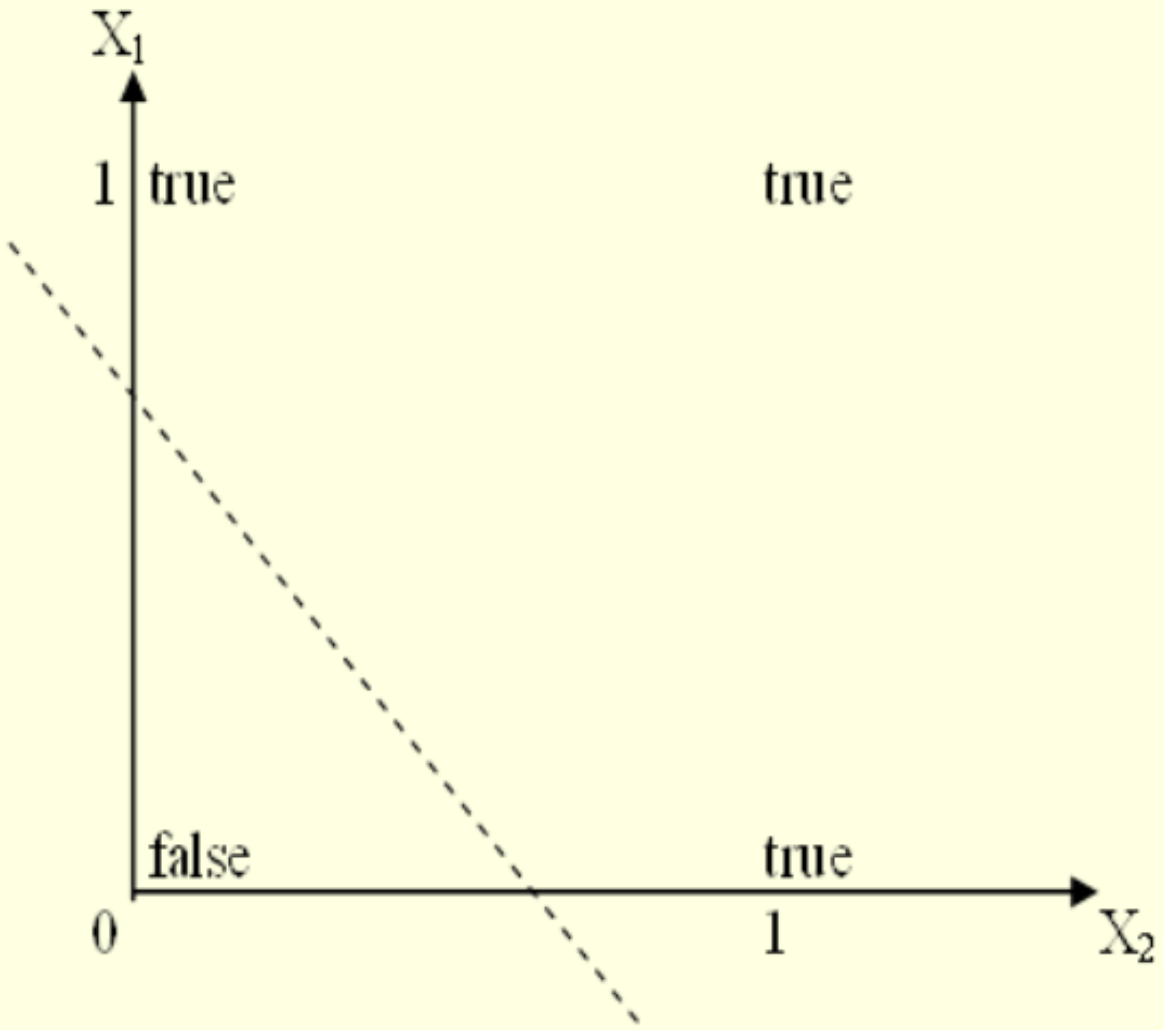
$$\varphi(v) = \begin{cases} +1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$



Perceptron for Classification

- The perceptron is used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- The perceptron can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.
- Given training examples of classes C_1, C_2 train the perceptron in such a way that :
  - *If the output of the perceptron is +1 then the input is assigned to class C_1*
  - *If the output is -1 then the input is assigned to C_2*

Boolean function OR – Linearly separable



Learning Process for Perceptron

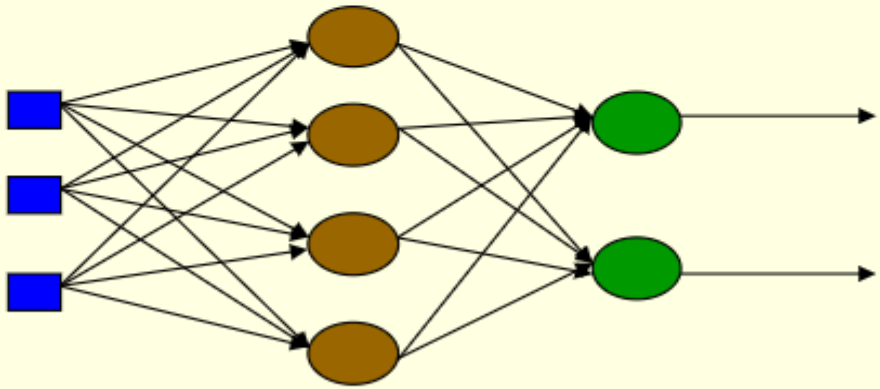
- Initially assign random weights to inputs between -0.5 and +0.5
- Training data is presented to perceptron and its output is observed.
- If output is incorrect, the weights are adjusted accordingly using following formula.

$$wi \leftarrow wi + (a^* xi^* e)$$, where 'e' is error produced and 'a' ($-1 < a < 1$) is learning rate

  - 'a' is defined as 0 if output is correct, it is +ve, if output is too low and –ve, if output is too high.
  - Once the modification to weights has taken place, the next piece of training data is used in the same way.
  - Once all the training data have been applied, the process starts again until all the weights are correct and all errors are zero.
  - Each iteration of this process is known as an epoch.

Multi layer feed-forward NN (FFNN)

- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning tasks.



3-4-2 Network

FFNN NEURON MODEL

- The classical learning algorithm of FFNN is based on the gradient descent method.
- For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- The activation function for node i may be defined as a simple form of the **sigmoid function** in the following manner:

$$\varphi(Vi) = \frac{1}{1+e^{(-A*Vi)}}$$

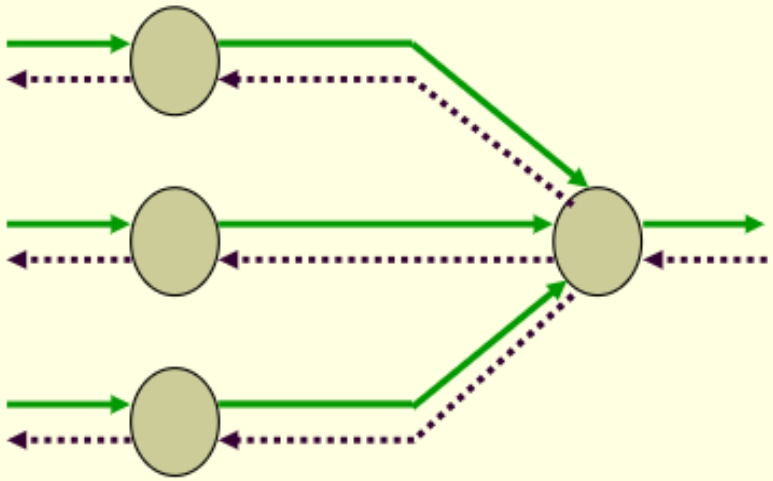
where $A > 0$, $V_i = \sum W_{ij} * Y_j$, such that W_{ij} is a weight of the link from node i to node j and Y_j is the output of node j.

Training Algorithm: Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

Backpropagation

- Back-propagation training algorithm



Network activation
Forward Step

Error propagation
Backward Step

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

Contd..

- Consider a network of three layers.
- Let us use i to represent nodes in input layer, j to represent nodes in hidden layer and k represent nodes in output layer.
- w_{ij} refers to weight of connection between a node in input layer and node in hidden layer.
- The following equation is used to derive the output value Yj of node j

$$Yj = \frac{1}{1 + e^{-X_j}}$$

where, $X_j = \sum x_i \cdot w_{ij} - \theta_j$, $1 \leq i \leq n$; n is the number of inputs to node j, and θ_j is threshold for node j

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

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

Backprop learning algorithm (incremental-mode)

n=1;

initialize **weights** randomly;

while (stopping criterion not satisfied or n <max_iterations)

for each example (**x**,*d*)

- run the network with input x and compute the output y
- update the weights in backward order starting from those of the output layer:

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

with Δw_{ji} computed using the (generalized) Delta rule

end-for

n = n+1;

end-while;

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 used for updating the weights.

Choice of learning rate

- The right value of η depends on the application.
- Values between 0.1 and 0.9 have been used in many applications.
- Other heuristics is that adapt η during the training as described in previous slides.

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:

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

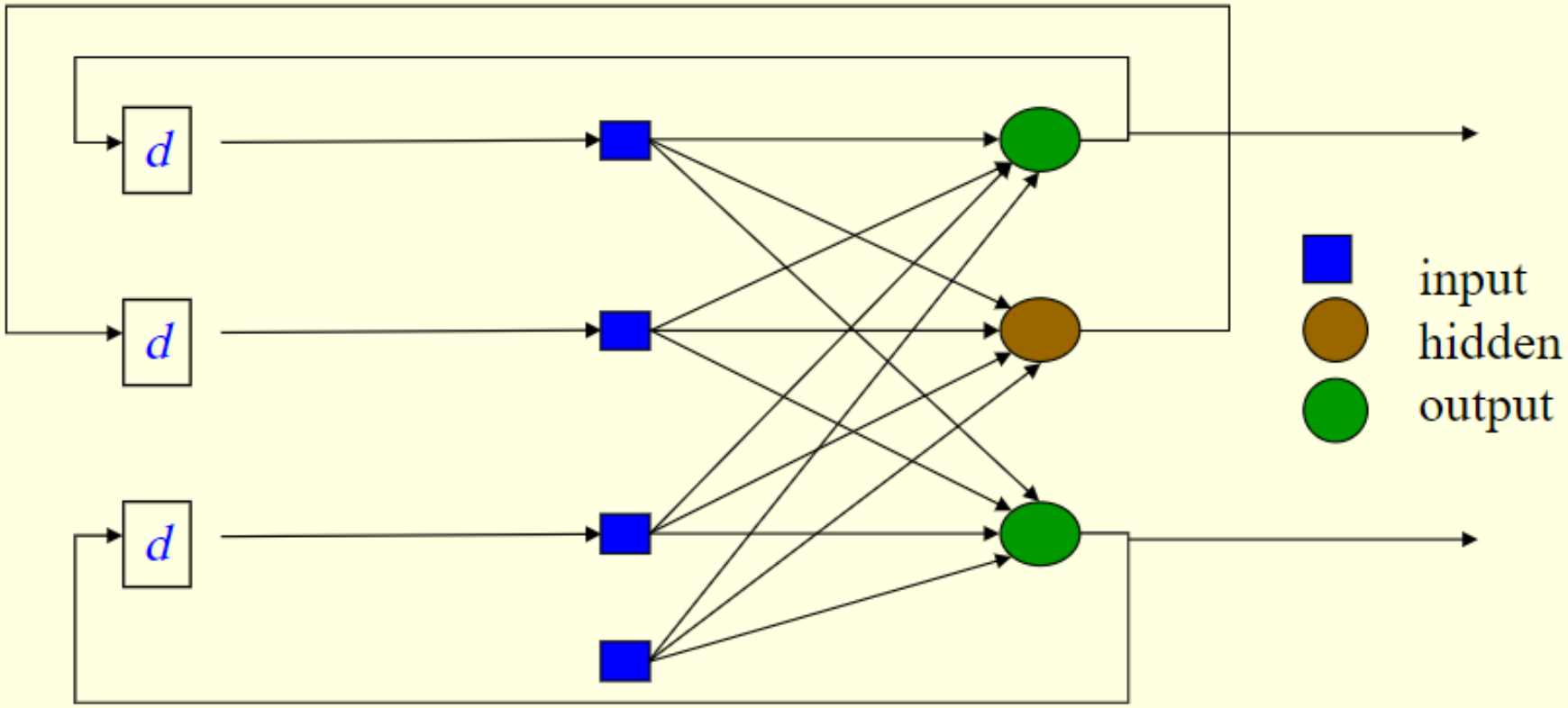
|W|= number of weights
a=expected accuracy on test set

Recurrent Network

- **FFNN** is acyclic where data passes from input to the output nodes and not vice versa.
  - Once the FFNN is trained, its state is fixed and does not alter as new data is presented to it. It does not have memory.
- **Recurrent network** can have connections that go backward from output to input nodes and models dynamic systems.
  - In this way, a recurrent network's internal state can be altered as sets of input data are presented. It can be said to have memory.
  - It is useful in solving problems where the solution depends not just on the current inputs but on all previous inputs.
- Applications
  - predict stock market price,
  - weather forecast

Recurrent Network Architecture

- Recurrent Network with ***hidden neuron*** : unit delay operator *d* is used to model a dynamic system



Learning and Training

- During learning phase,
  - a recurrent network feeds its inputs through the network, including feeding data back from outputs to inputs
  - process is repeated until the values of the outputs do not change.
- This state is called equilibrium or stability
- Recurrent networks can be trained by using back-propagation algorithm.
- In this method, at each step, the activation of the output is compared with the desired activation and errors are propagated backward through the network.
- Once this training process is completed, the network becomes capable of performing a sequence of actions.