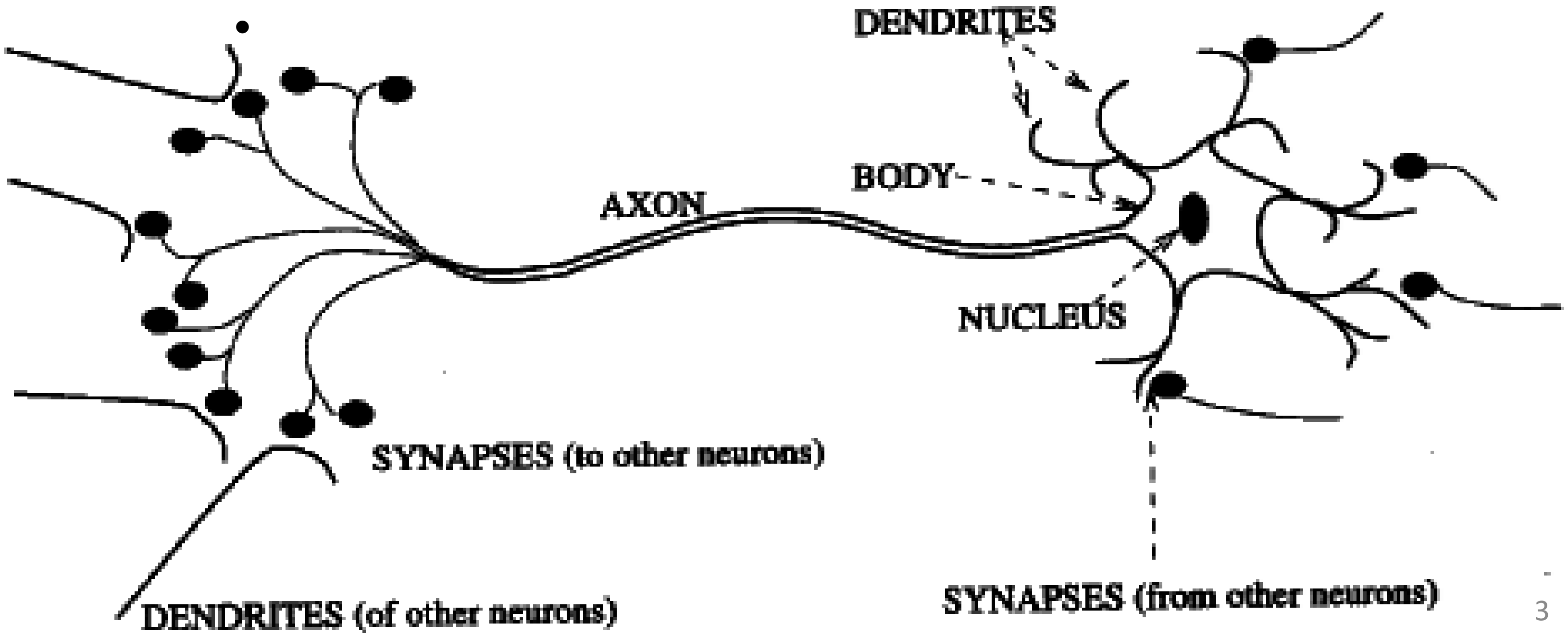
Artificial Neural Networks

Artificial Neural Networks

- What?
  - Computing Systems inspired by Biological Neural Networks.

Biological Neural Networks

- Nervous System
  - Biological Neural Networks
    - Biological Neurons
      - What?
        - Biological Neuron is an electrically excitable cell that processes and transmits information through electrical and chemical signals.



Biological Neural Networks

- Nervous System
  - Biological Neural Networks
    - Biological Neurons
      - 10 – 100 billion Neurons
      - connection to 100 – 10000 other neurons
      - 100 different types
      - layered structure

Biological Neural Networks

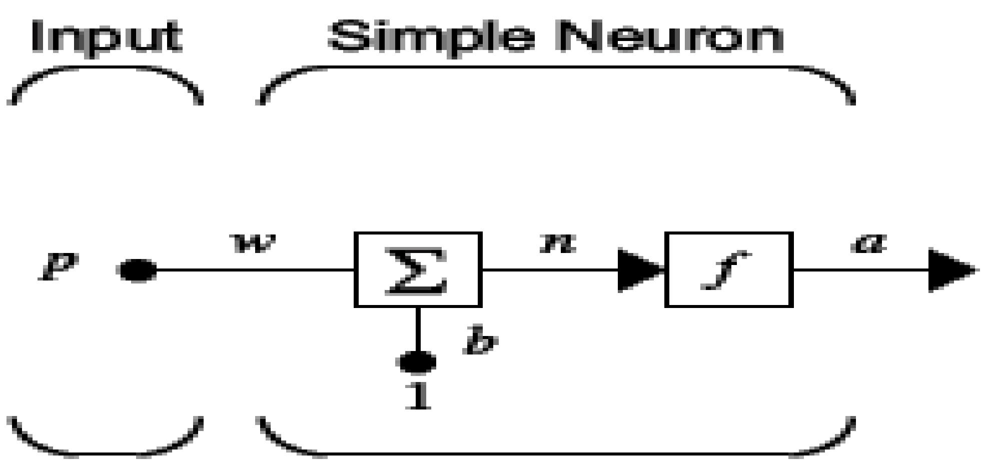
- Features
  - Parallel processing systems
  - Neurons are processing elements and each neuron performs some simple calculations
  - Neurons are networked
  - Each connection conveys a signal from one node (neuron) to another
  - Connection strength decides the extent to which a signal is amplified or diminished by a connection

Biological Neural Networks

- Features (from our experience)
  - Ability to learn from experience and accomplish complex task without being programmed explicitly
    - Driving
    - Speaking using a particular language
    - Translation
    - Speaker Recognition
    - Face Recognition, etc…

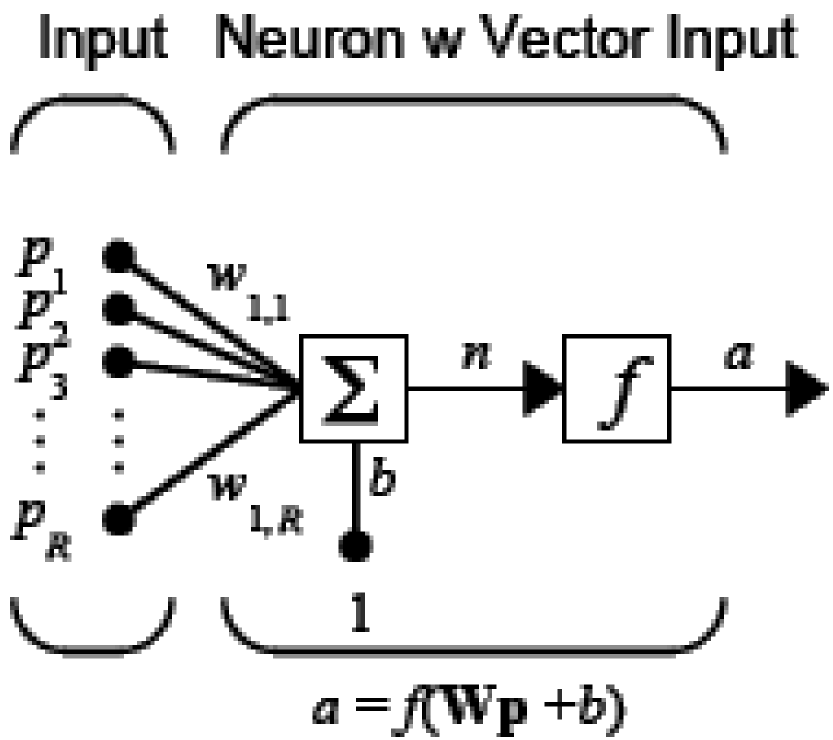
Artificial Neuron Model

- An artificial neuron is a mathematical function regarded as a model of a biological neuron.
  - Remember: 1. BN is able to receive the amplified or diminished inputs from multiple dendrites 2. It is able to combine these inputs 3. It is able to process input and produce output
- Simple Neuron
  - Weight Function, Net Input Function & Transfer Function



$a = f(wp + b)$

Neuron with Vector Input

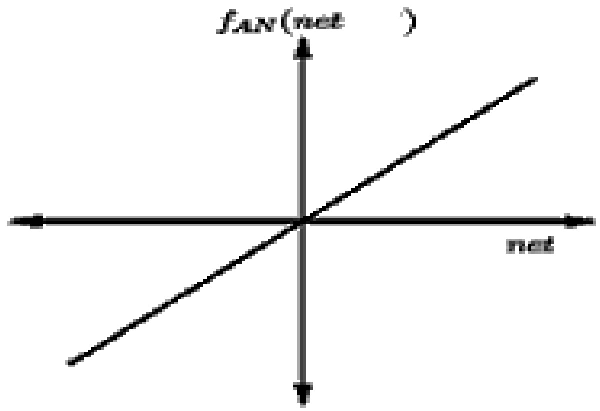


Where

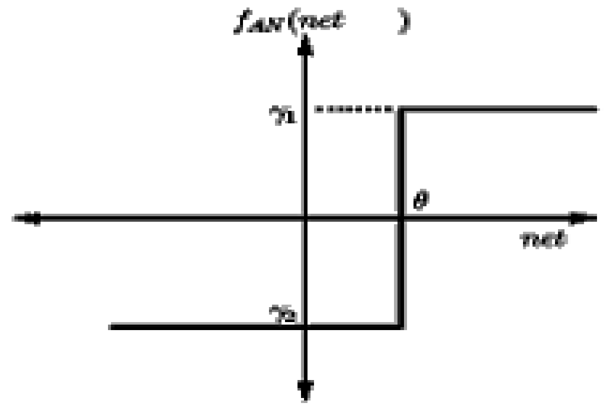
R = number of elements in input vector

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$$

Activation Functions

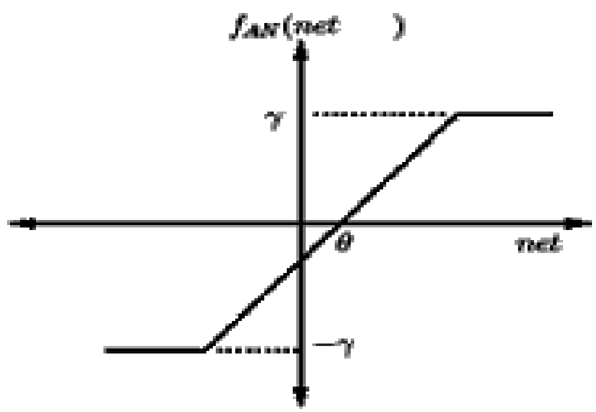


Linear function

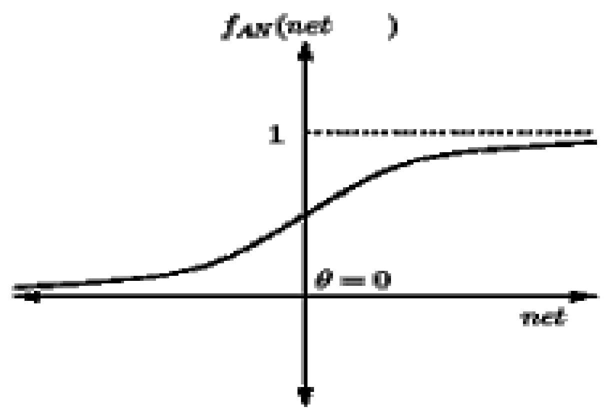


Step function

< & >=

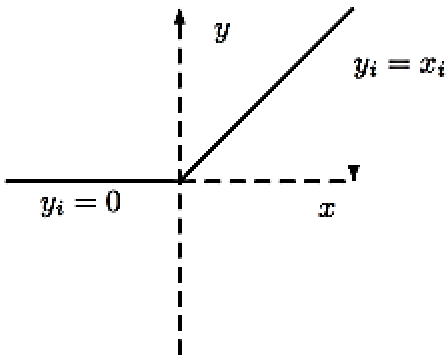


Ramp function

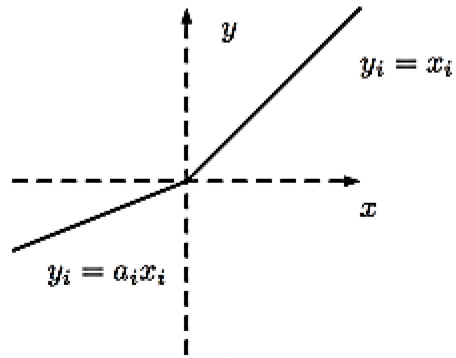


Sigmoid function

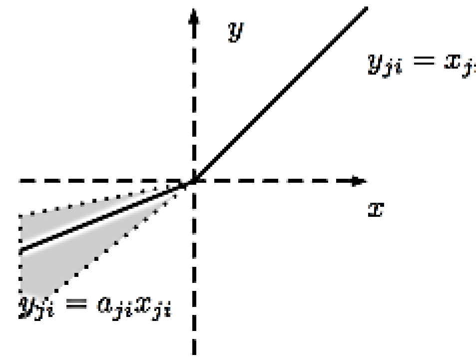
Activation Functions [12]



ReLU



Leaky ReLU/PReLU



Randomized Leaky ReLU

f(net) = max(0, net)

$0.01*x_i / a_i*x_i$

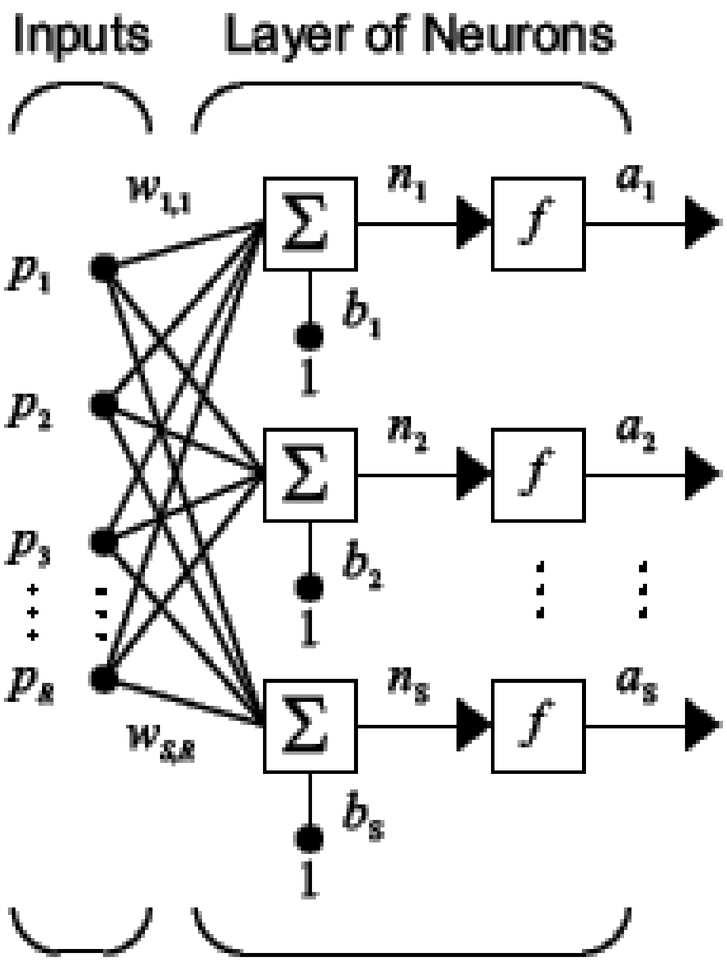
$$y_{ji} = \begin{cases} x_{ji} & \text{if } x_{ji} \geq 0 \\ a_{ji} x_{ji} & \text{if } x_{ji} < 0, \end{cases}$$

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0. \end{cases}$$

$$a_{ji} \sim U(l, u), l < u \text{ and } l, u \in [0, 1)$$

a_{ji} is a random number sampled from a uniform distribution U(l, u).

A Layer of Neurons



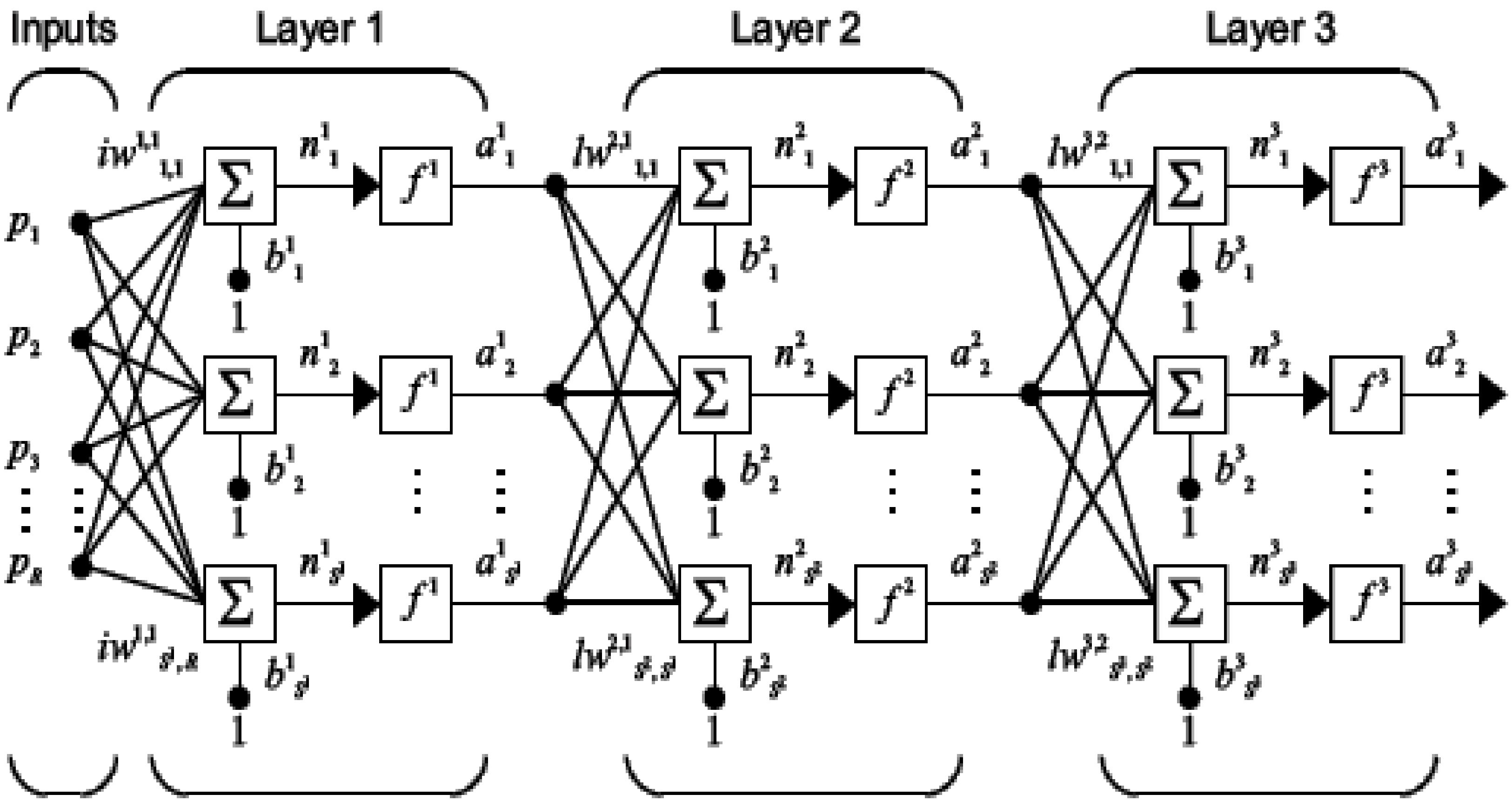
$$a = f(Wp + b)$$

Where

R = number of elements in input vector

S = number of neurons in layer

Multiple Layers of Neurons

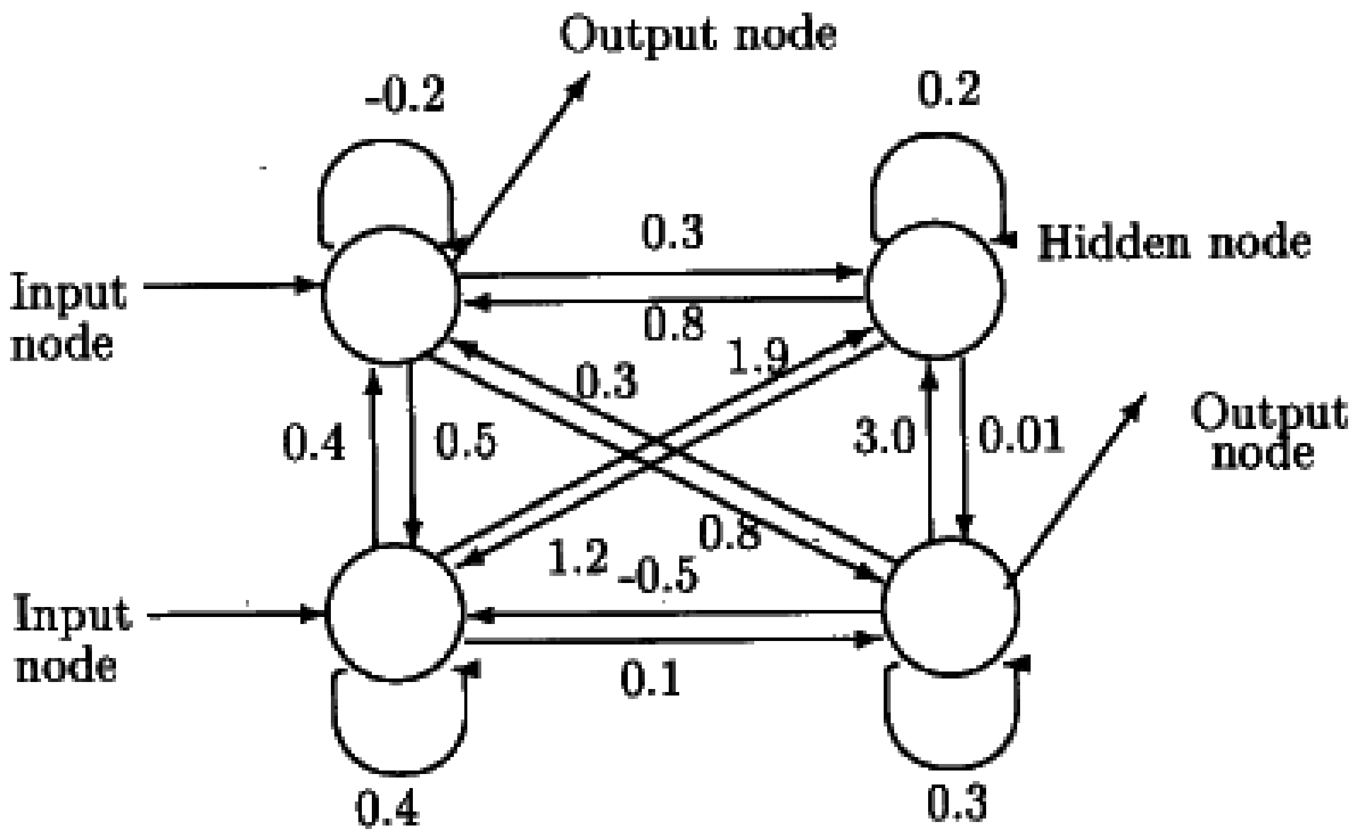


$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{a}^1+\mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{a}^2+\mathbf{b}^3)$$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

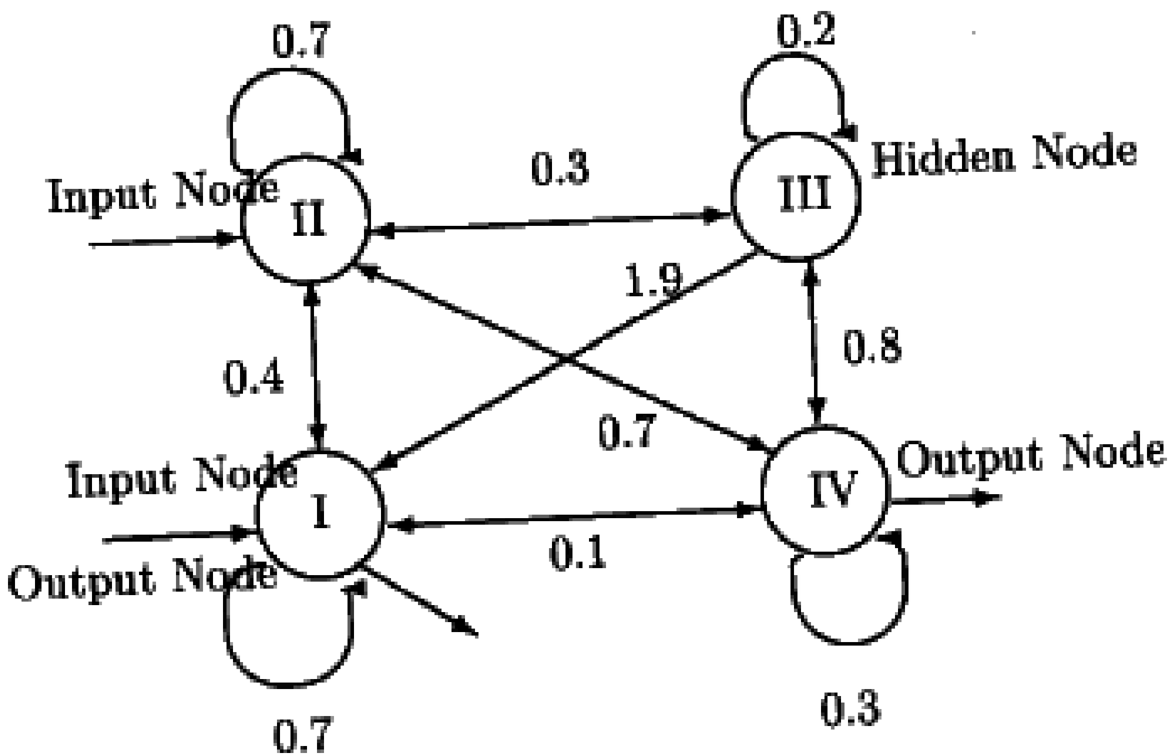
ANN Architectures

- Fully Connected Network (Asymmetric)



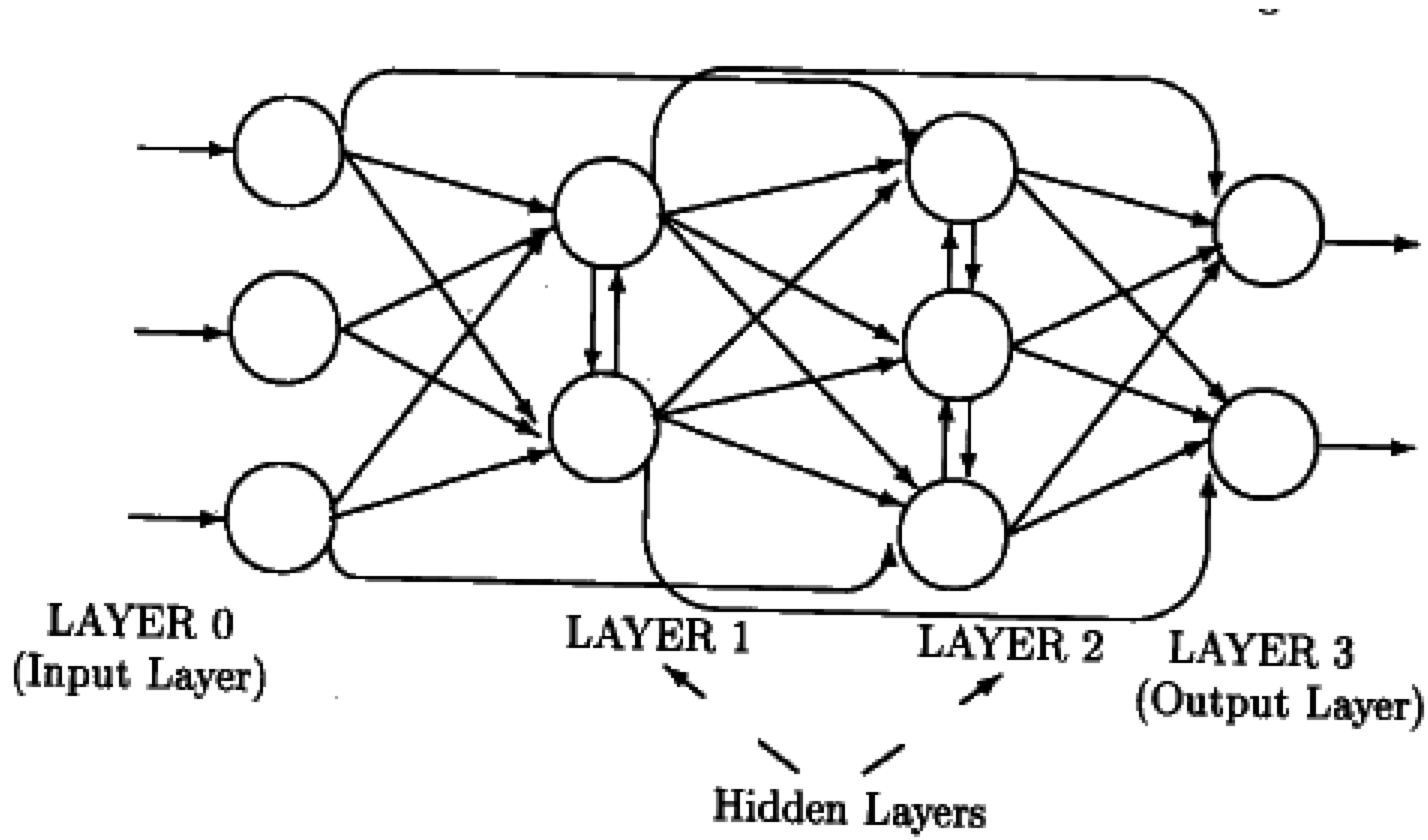
ANN Architectures

- Fully Connected Network (Symmetric)



ANN Architectures

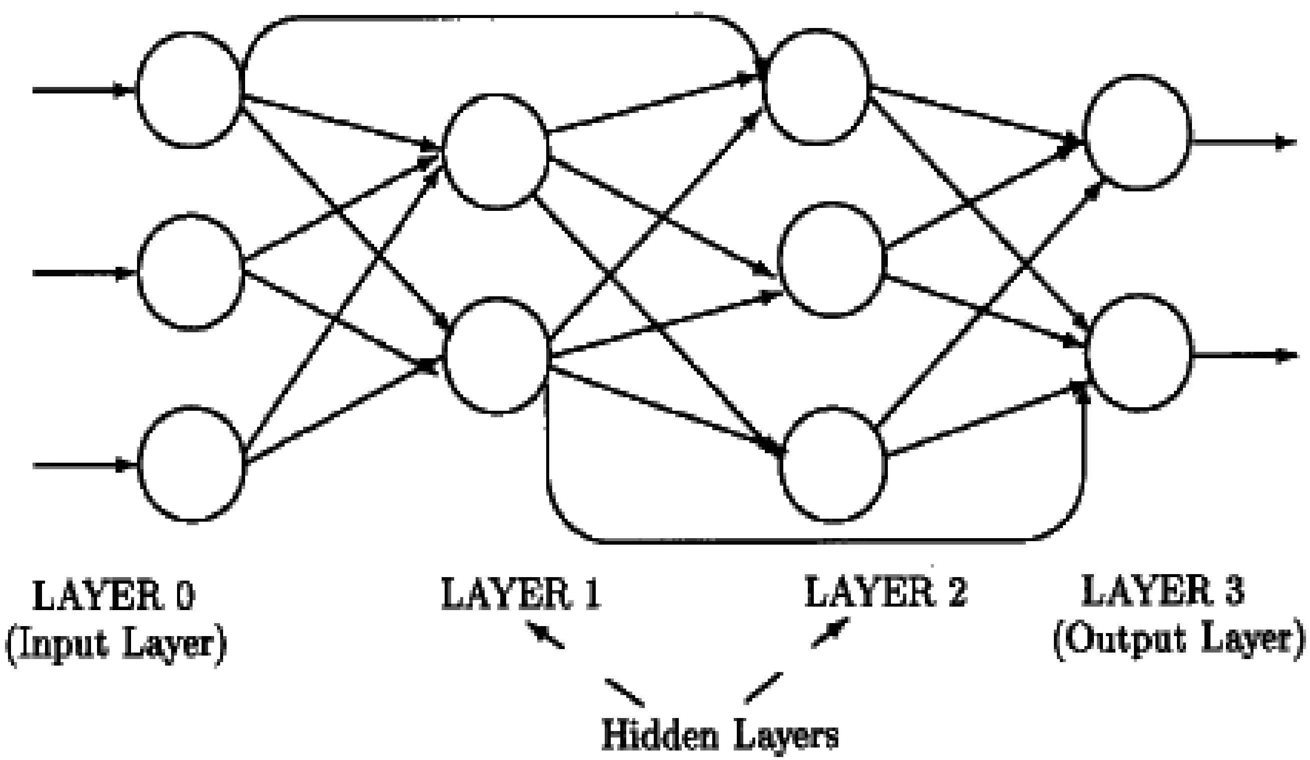
➢ Layered Network



These are networks in which nodes are partitioned into subsets called layers, with no connections that lead from layer j to layer k if j > k

ANN Architectures

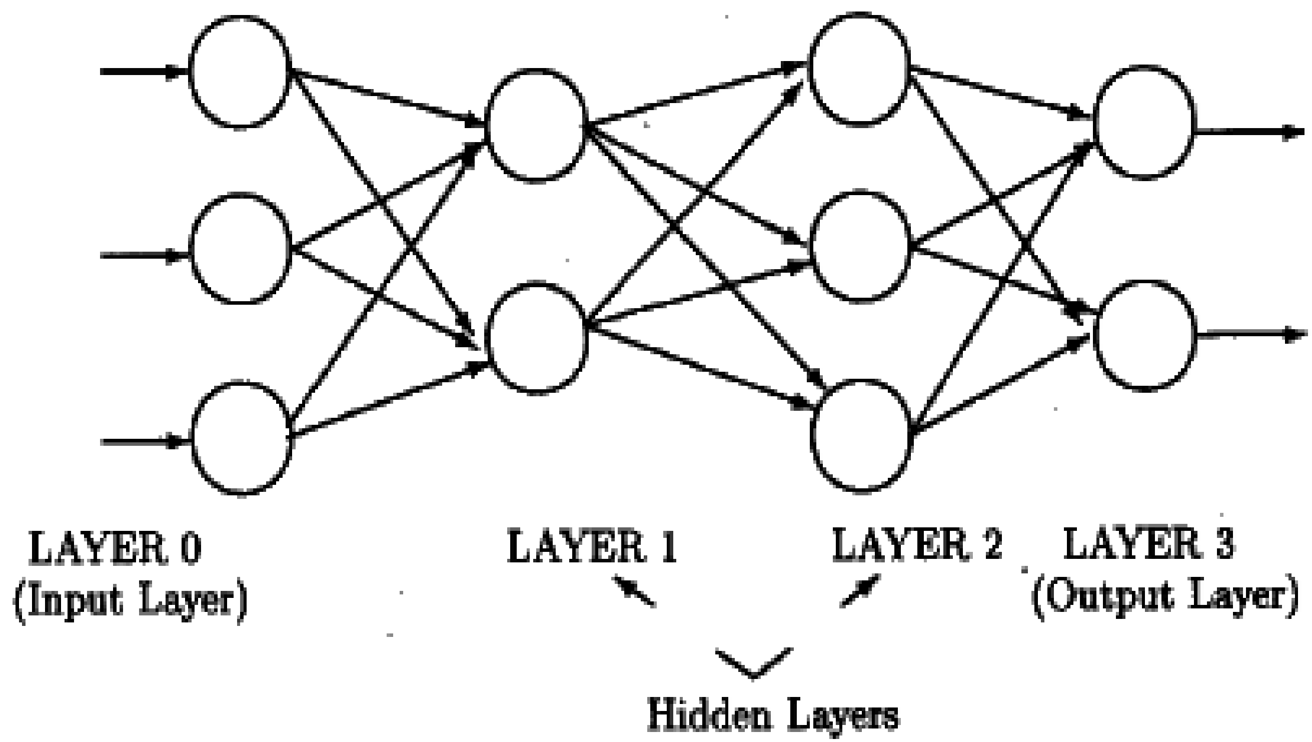
- Acyclic Network



These are subclass of layered networks with no intra-layer connections.

ANN Architectures

- Feedforward Network



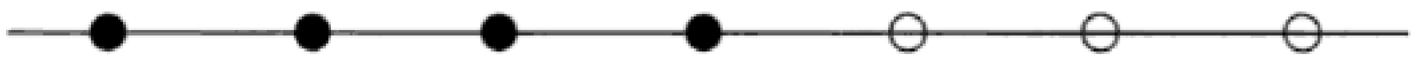
These are subclass of acyclic networks in which a connection is allowed from a node in layer i only to nodes in layer i + 1.

Learning in ANN

- Types of Learning
  - Supervised Learning
  - Unsupervised Learning

Linear Separability

- 1 – D Case
  - 7/5 Students data – Weight Values & Obese/Not Obese
    - (50, NO), (55, NO), (60, NO), (65, NO), (70, O), (75, O), (80, O) – Linearly Separable
    - (55, NO), (60, O), (65, NO), (70, O), (75, O) – Linearly Inseparable
  - Learning a separating point/line

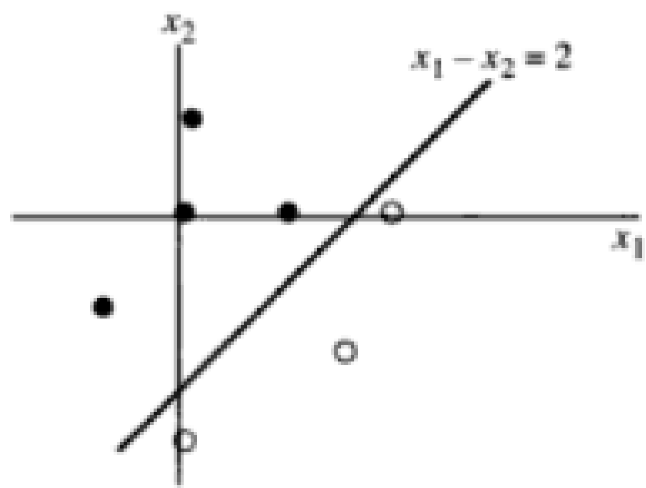


(a) Separable by a perceptron

(b) Not separable by a perceptron

Linear Separability

- 2 – D Case
  - Learning a separating line



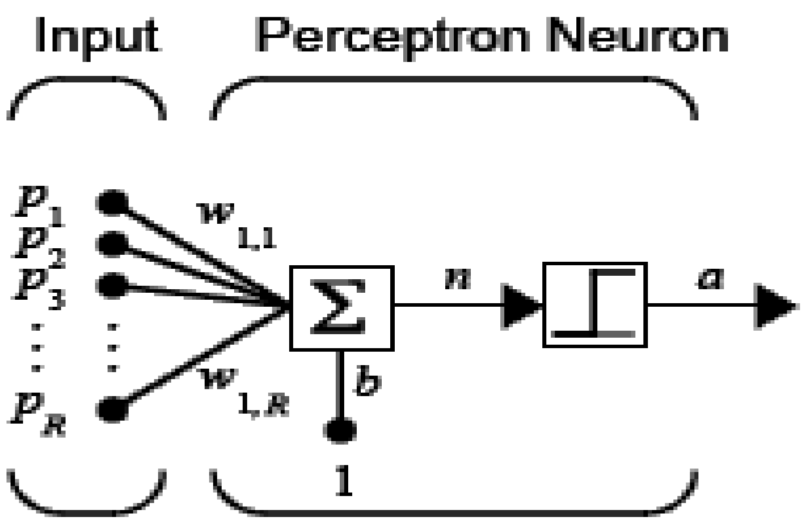
(a)

Linear Separability

- 3 – D Case
  - Learning a separating plane
- Higher Dimensional Case
  - Learning a separating hyperplane

Perceptron Model [6]

- What is Perceptron?
  - It is a machine which can learn (using examples) to assign input vectors to different classes.
- What can it do?
  - 2-class linear classification problem
    - What?
    - Process



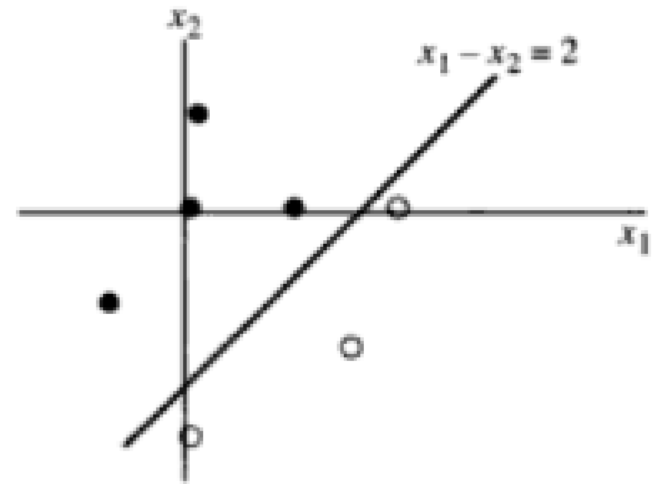
Where

R = number of elements in input vector

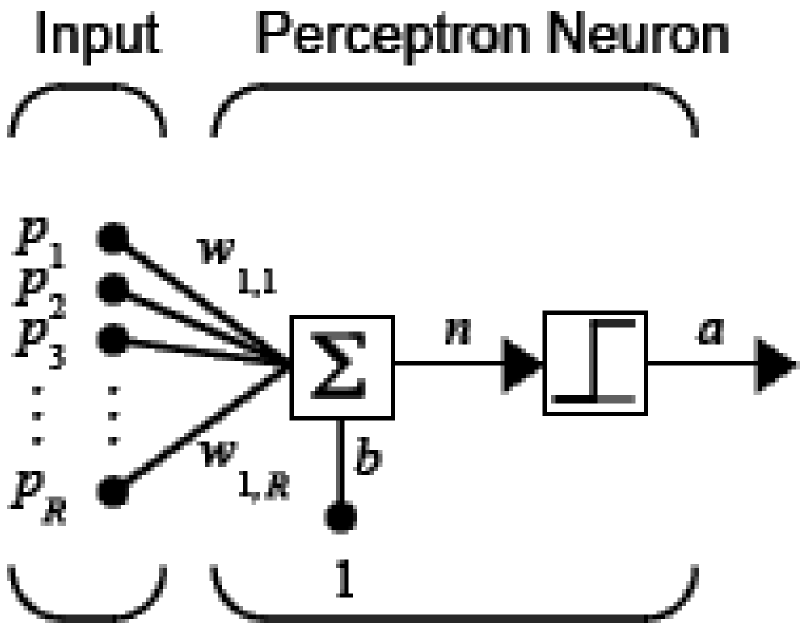
$a = \mathbf{hardlim}(\mathbf{Wp} + b)$ hardlim(n) = 1, if n >= 0; 0 otherwise.

Perceptron Learning Rule [5, 6]

- Learning Process



(a)



$a = \mathbf{hardlim}(\mathbf{Wp} + b)$

R = number of elements in input vector

- $W_{new} = W_{old} + \eta e P$
- $b_{new} = b_{old} + \eta e$, where e = target - actual

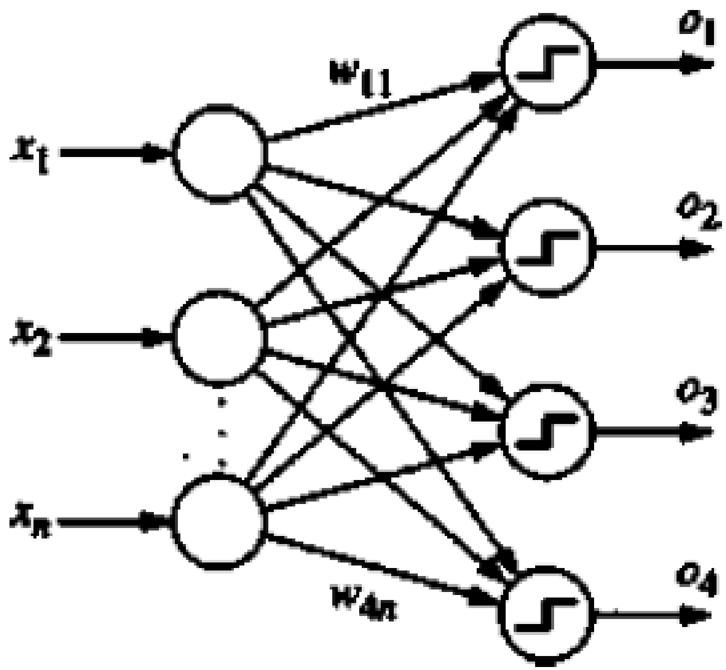
Numerical

- Assume 7 one dimensional input patterns {0.0, 0.17, 0.33, 0.50, 0.67, 0.83, 1.0}. Assume that first four patterns belong to class 0 (with desired output 0) and remaining patterns belong to class 1 (with desired output 1). Design a perceptron to classify these patterns. Use perceptron learning rule. Assume learning rate = 0.1 and initial weight and bias to be (-0.36) and (-0.1) respectively. Show computation for two epochs.

Some Issues

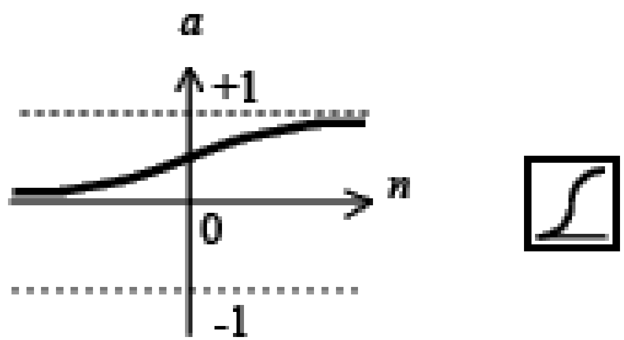
- Why to use bias?
- Termination Criterion
- Learning Rate
- Non-numeric Inputs
- Epoch

Multiclass Discrimination

- Layer of Perceptron
  - To distinguish among n classes, a layer of n perceptrons can be used
  - 

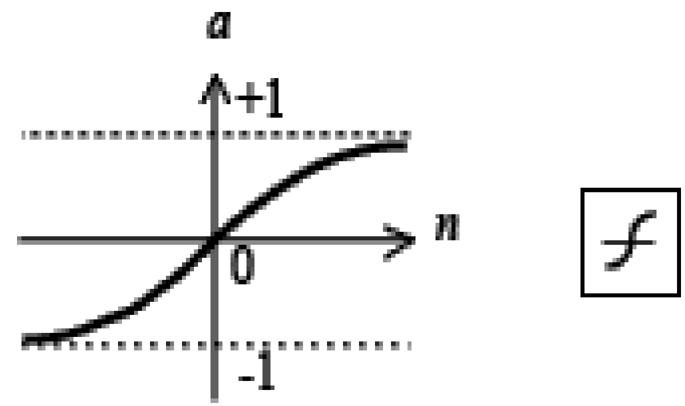
- *A* presented sample is considered to belong to i^{th} class only if i^{th} output is 1 and remaining are 0.
- If all outputs are zero, or if more than one output value equals one, the network may be considered to have failed in classification task.

Multilayer Networks – Typical Transfer Functions

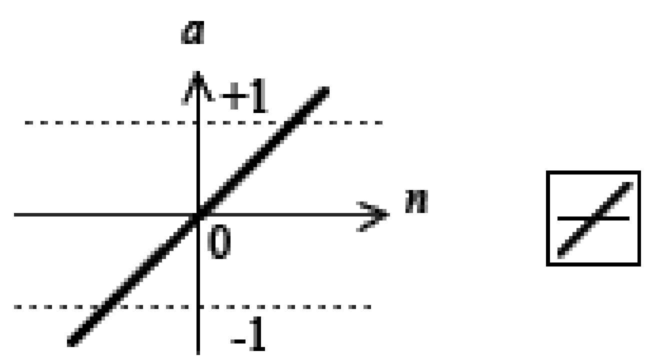


$a = logsig(n)$

Log-Sigmoid Transfer Function



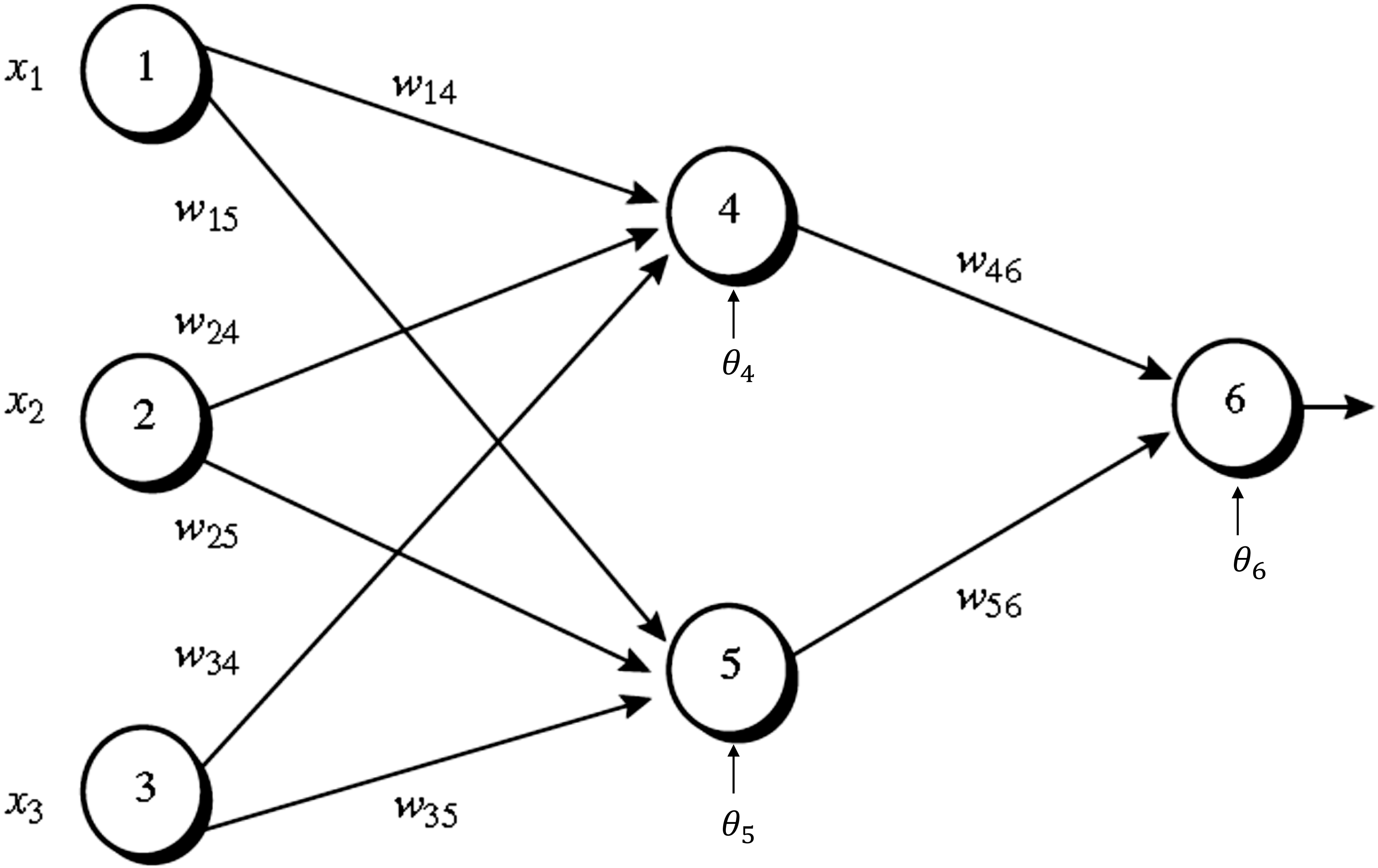
$a = tansig(n)$



$a = purelin(n)$

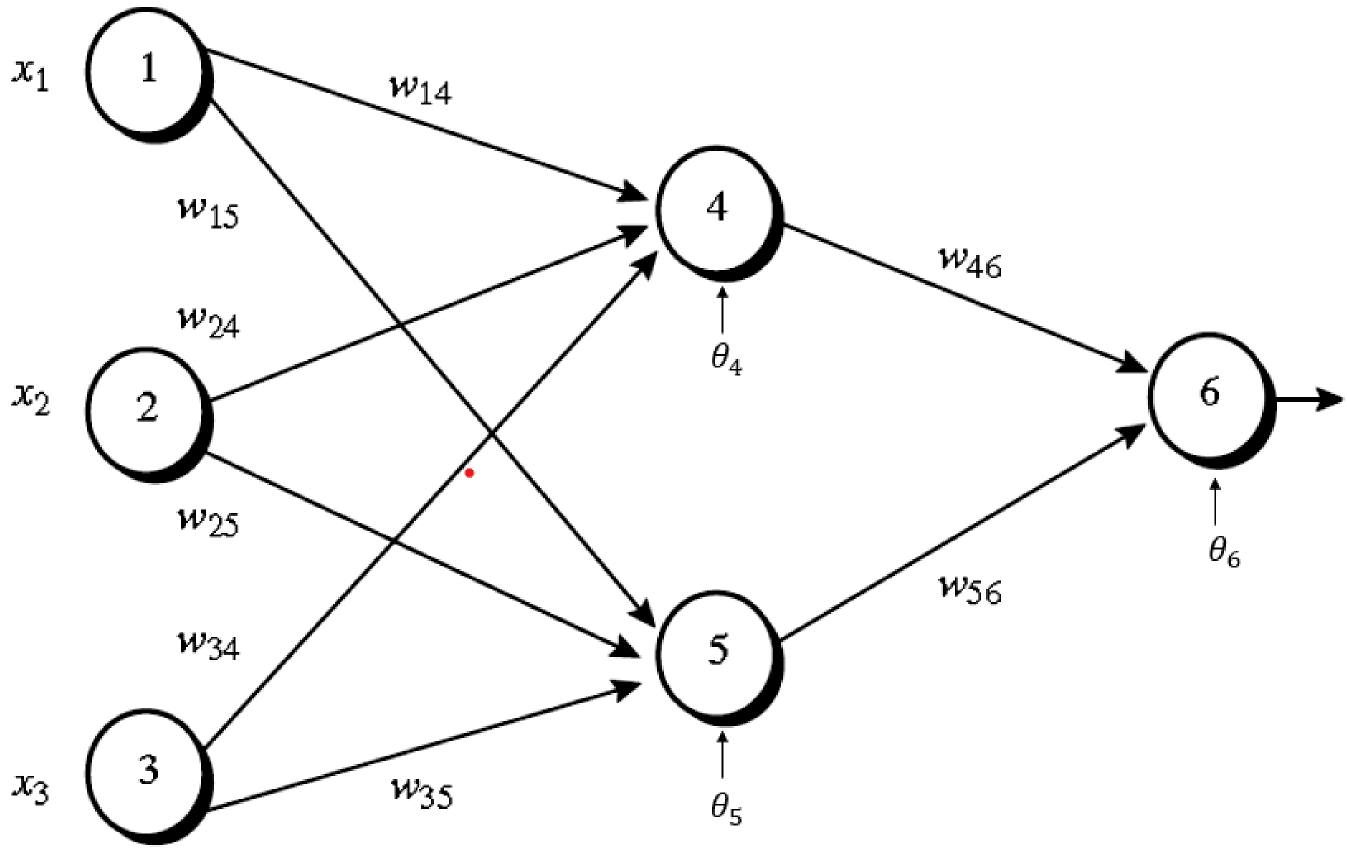
Linear Transfer Function

Example of Backpropagation



An example of a multilayer feed-forward neural network.

Example of Backpropagation



Class Label : 1

Initial input, weight, and bias values.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Example of Backpropagation

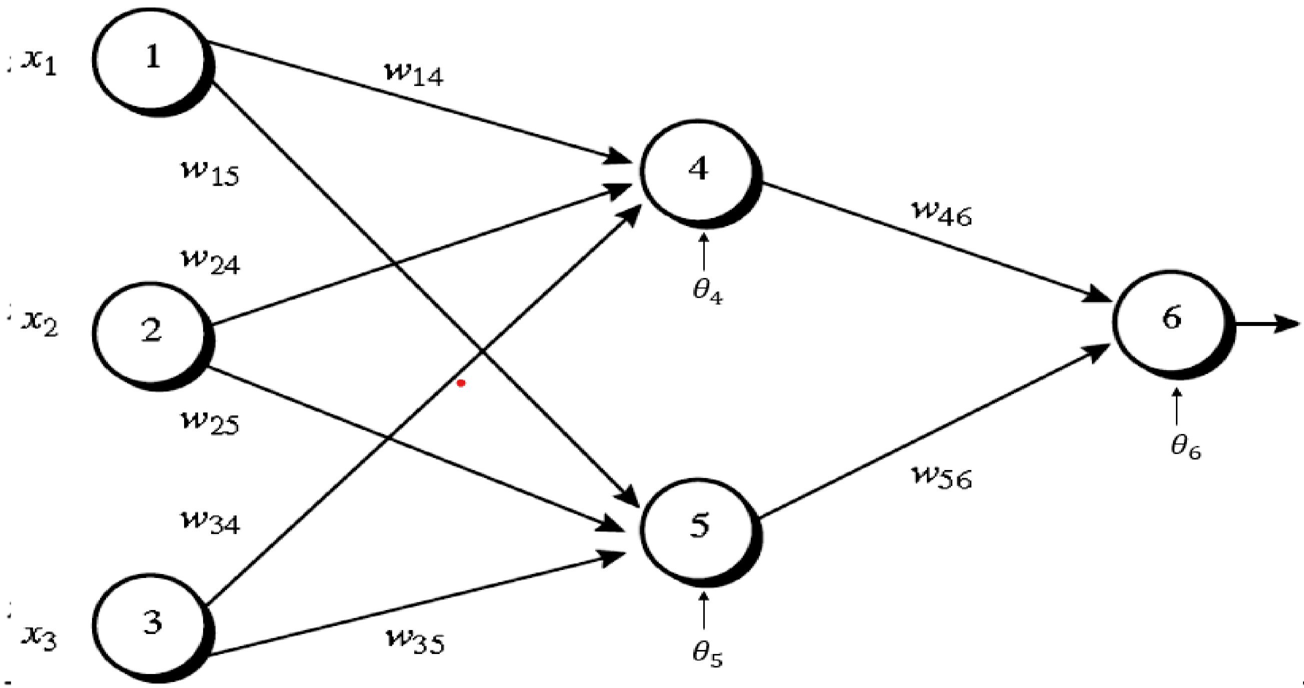


Figure 6.18 An example of a multilayer feed-forward neural network.

Table 6.3 Initial input, weight, and bias values.

Class Label : 1

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Table 6.4 The net input and output calculations.

Unit j	Net input, I_j	Output, O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

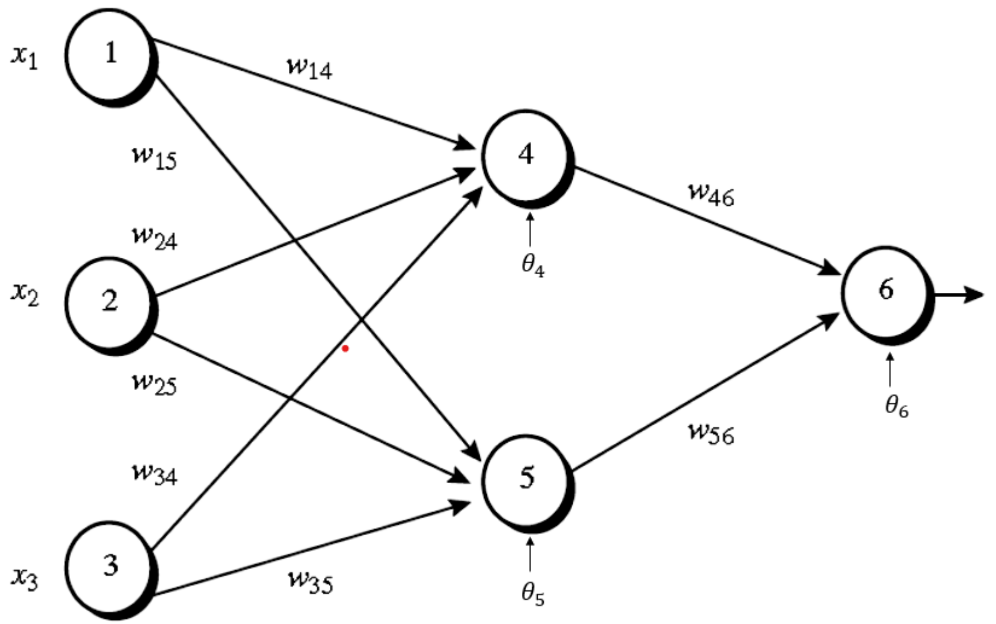
Example of Backpropagation

- Backpropagation

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

$$Err_j = O_j(1-O_j)(T_j-O_j),$$

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk},$$



j	O_j
4	0.332
5	0.525
6	0.474

Table 6.5 Calculation of the error at each node.

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474) = 0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

Example of Backpropagation

- Backpropagation

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

$$\Delta w_{ij} = (l) Err_j O_i$$

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

$$\Delta \theta_j = (l) Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

j	O_j	Err_j
4	0.332	-0.0087
5	0.525	-0.0065
6	0.474	0.1311

Table 6.6 Calculations for weight and bias updating.

Weight or bias	*New value*
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

Binary Cross-Entropy

Code

```
def CrossEntropy(yHat, y):
    if y == 1:
      return -log(yHat)
    else:
      return -log(1 - yHat)
```

Math

In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

$$-(y\log(p) + (1-y)\log(1-p))$$

Categorical Cross-Entropy

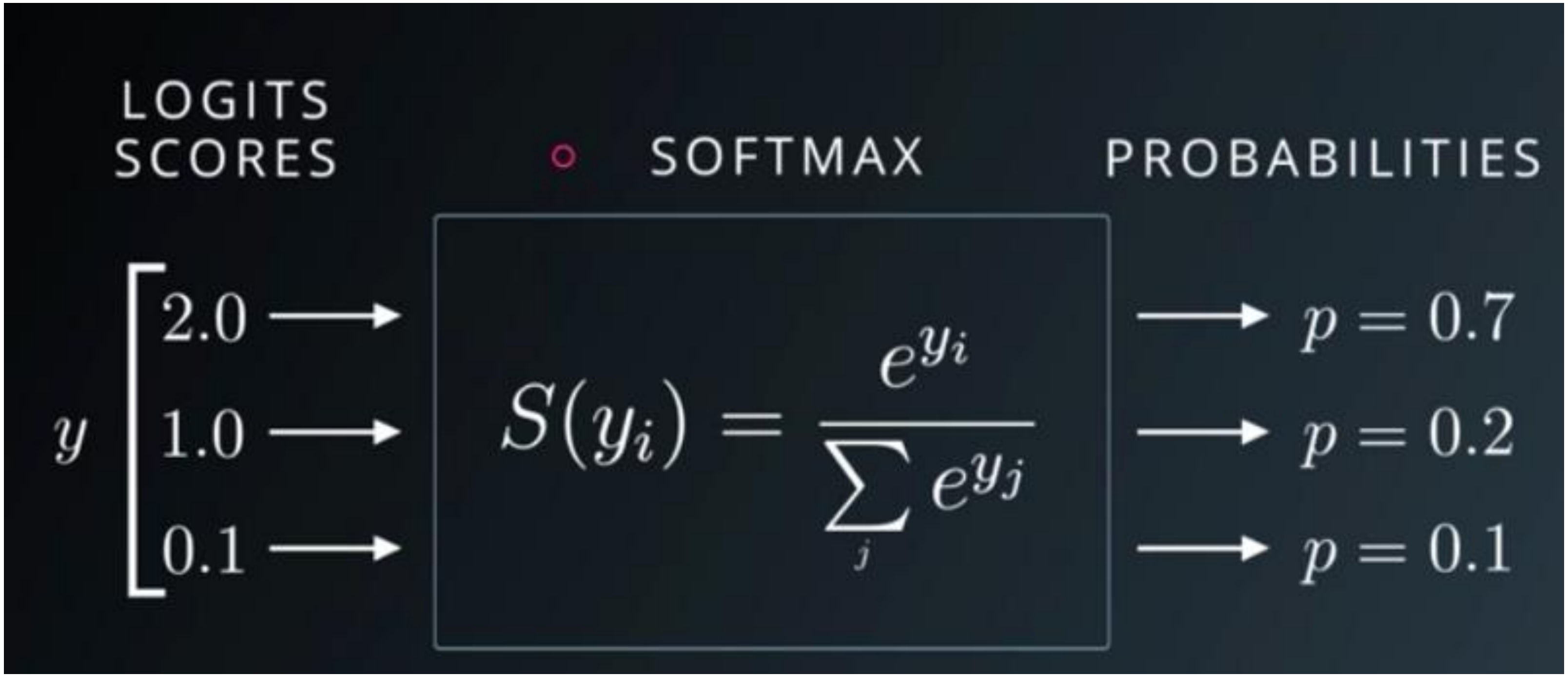
If $M > 2$ (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Note

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Softmax Activation Function



Disclaimer

- These slides are not original and have been prepared from various sources for teaching purpose.