CS 366: Artificial Intelligence

Perceptron

Two paradigms of AI

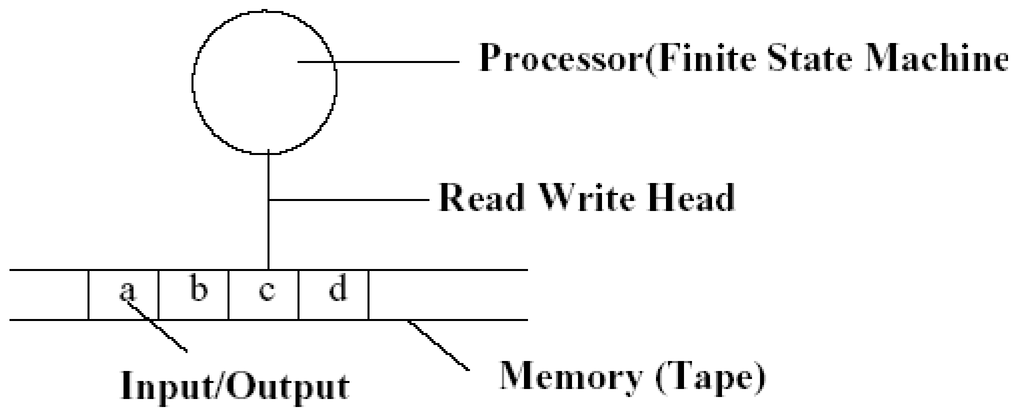
- Symbolic and Connectionist
- **Symbolic AI**
  - Physical Symbol System Hypothesis
  - Every intelligent system can be constructed by storing and processing symbols and nothing more is necessary
- **Connectionist AI**
  - Distributed method to represent knowledge
  - Inspired by human brains

Symbolic AI

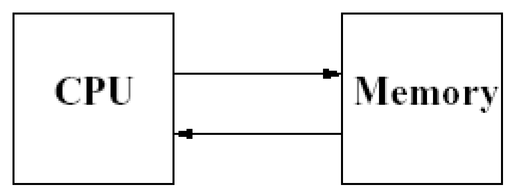
Symbolic AI has a bearing on models of computation such as

Turing Machine
Von Neumann Machine
Lambda calculus

Turing Machine & Von Neumann Machine



Turing machine



VonNeumann Machine

Challenges to Symbolic AI

Motivation for challenging Symbolic AI

A large number of computations and information process tasks that living beings are comfortable with, are not performed well by computers!

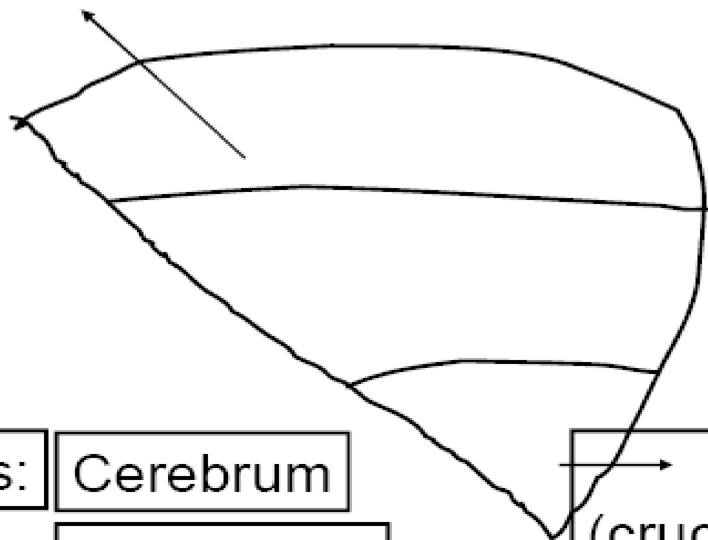
The Differences

Brain computation in living beings	TM computation in computers
Pattern Recognition	Numerical processing
Learning oriented	Programming oriented
Distributed & Parallel processing	Centralized & serial processing
Content addressable	Location addressable

The human brain

Seat of consciousness and cognition

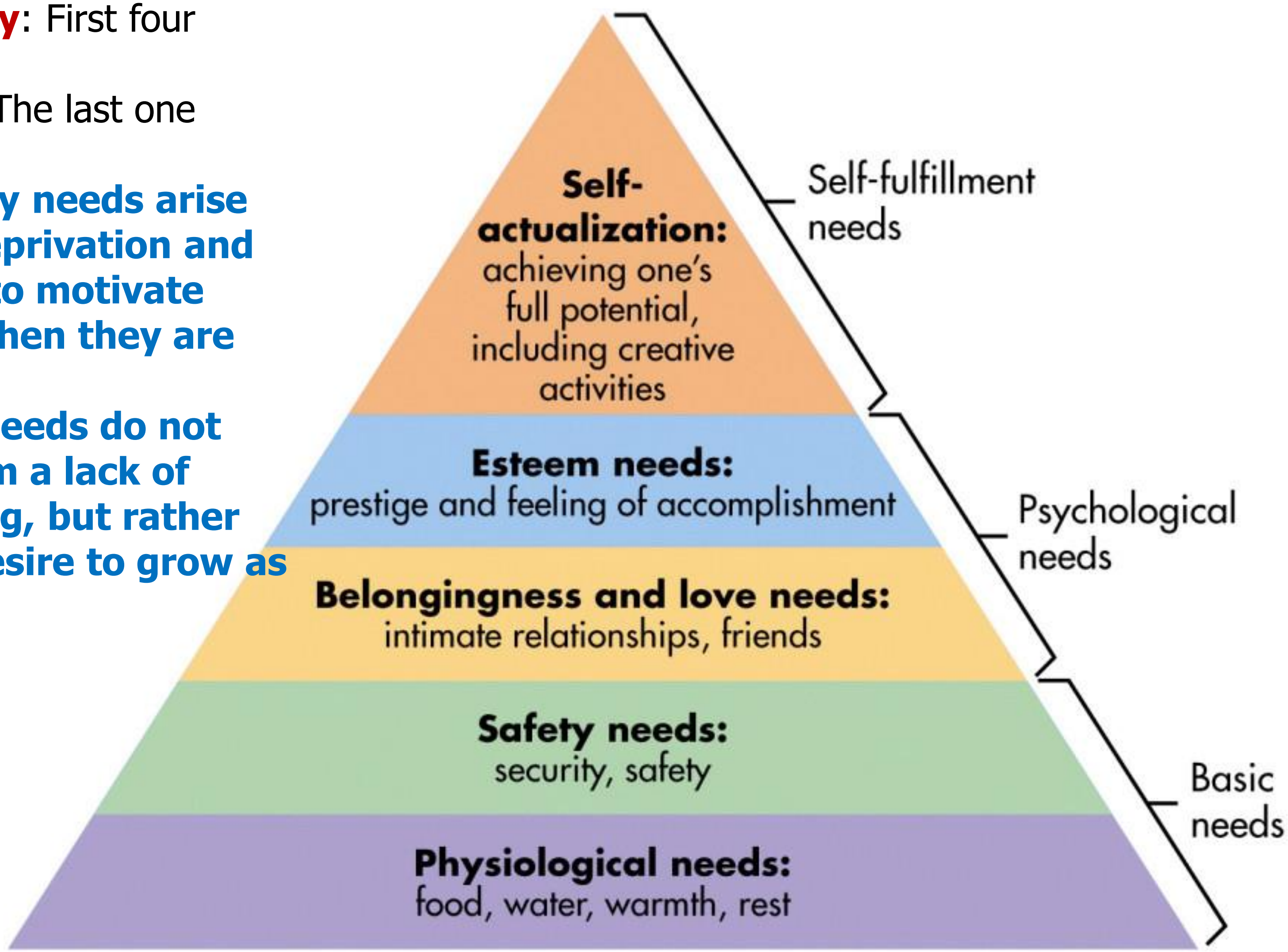
Perhaps the most complex information processing machine in nature


Higher brain (responsible for higher needs)
3- Layers:
Cerebrum
Cerebellum
Higher brain
Cerebrum
(crucial for survival)

Maslow Hierarchy of Need- 5 tier

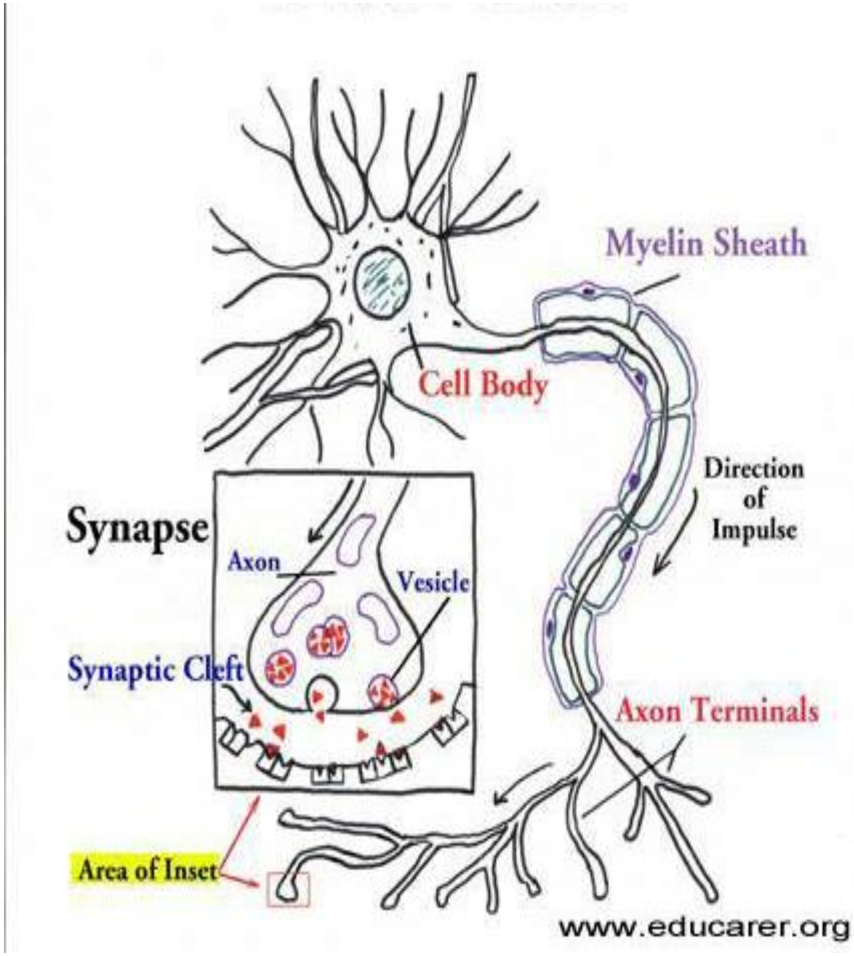
- **Deficiency**: First four levels
- **Growth**: The last one

- **Deficiency needs arise due to deprivation and are said to motivate people when they are unmet**
- **Growth needs do not stem from a lack of something, but rather from a desire to grow as a person**



Neuron - "classical": Fundamental Units of Human Brain

- **Dendrites**
  - Receiving stations of neurons
  - Don't generate action potentials
- **Cell body**
  - Site at which information received is integrated
- **Axon**
  - Generate and relay action potential
  - Terminal
    - Relays information to next neuron in the pathway

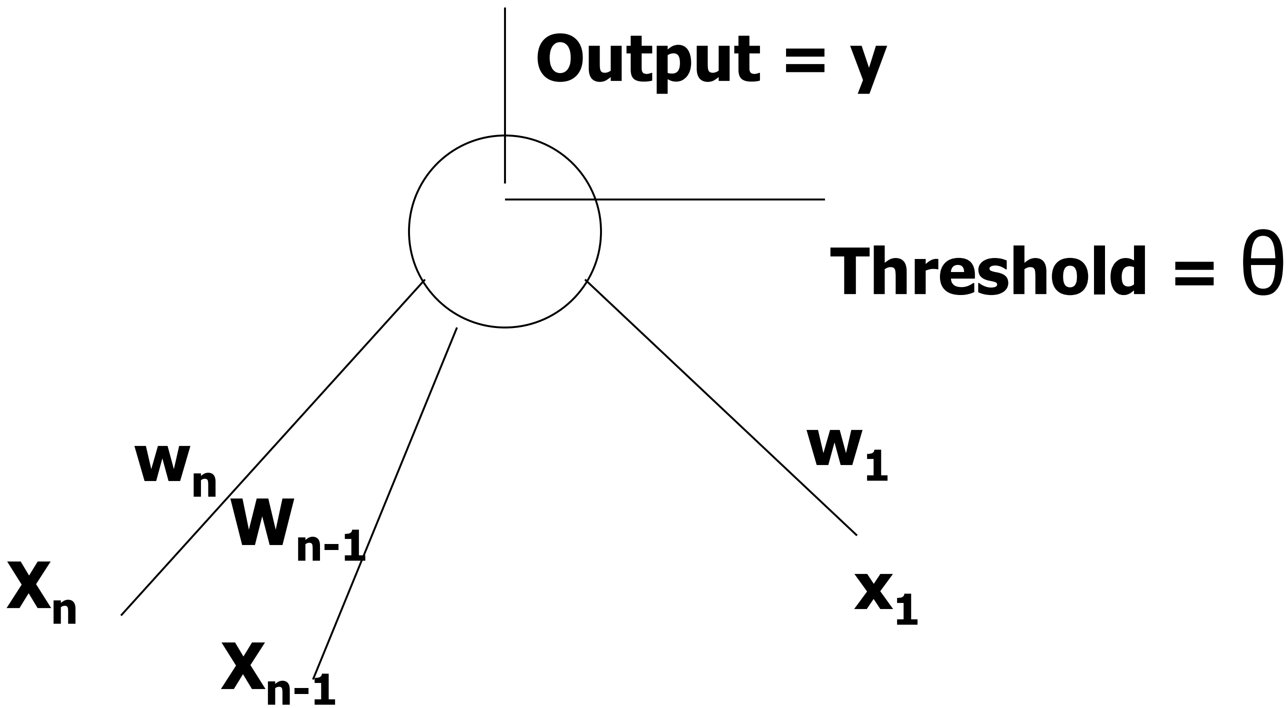


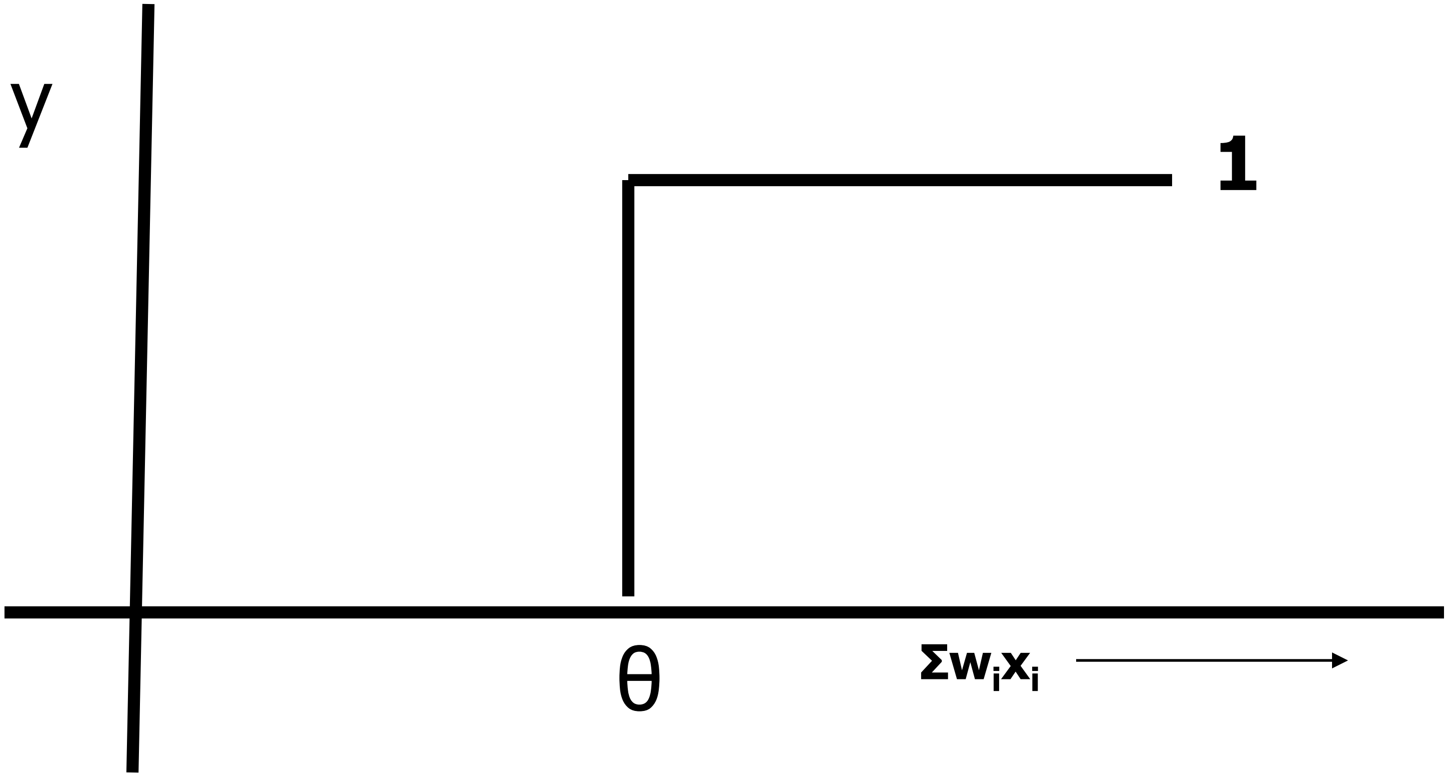
http://www.educarer.com/images/brain-nerve-axon.jpg

Perceptron

The Perceptron Model

A perceptron is a computing element with input lines having associated weights and the cell having a threshold value. The perceptron model is motivated by the biological neuron





Step function / Threshold function

y = 1 for $\Sigma w_i x_i >= \theta$

=0 otherwise

Features of Perceptron

- Input-output behavior is discontinuous and the derivative does not exist at $\boldsymbol{\Sigma w_i x_i = \theta}$

- $\boldsymbol{\Sigma w_i x_i - \theta}$ is the net input denoted as net

- Referred to as a linear threshold element - *linearity because of* ***x*** *appearing with power* ***1***

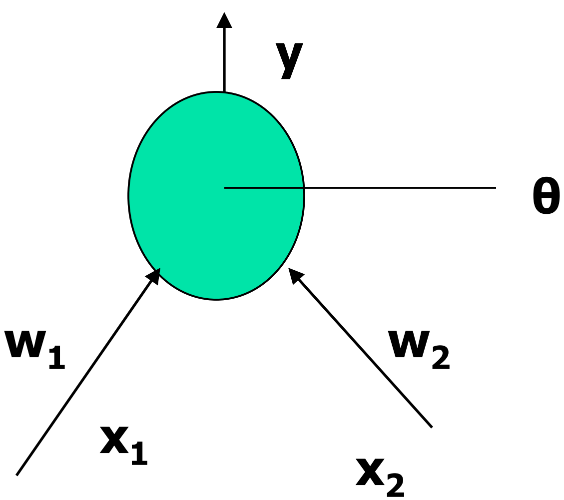
y= f(net): Relation between y and net is non-linear

Computation of Boolean functions

AND of 2 inputs

X1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

The parameter values (weights & thresholds) need to be found



Computing parameter values

$w1 * 0 + w2 * 0 \ <= \theta \Rightarrow \theta >= 0$; since $y=0$

$w1 * 0 + w2 * 1 \ <= \theta \Rightarrow w2 <= \theta$; since $y=0$

$w1 * 1 + w2 * 0 \ <= \theta \Rightarrow w1 <= \theta$; since $y=0$

$w1 * 1 + w2 * 1 > \theta \Rightarrow w1 + w2 > \theta$; since $y=1$

$w1 = w2 = \theta = 0.5$

satisfy these inequalities and find parameters to be used for computing AND function

Other Boolean functions

- OR can be computed using values of $w1 = w2 = 1$ and $\theta = 0.5$

- XOR function gives rise to the following inequalities:

$w1 * 0 + w2 * 0 \leq \theta \Rightarrow \theta \geq 0$

$w1 * 0 + w2 * 1 > \theta \Rightarrow w2 > \theta$

$w1 * 1 + w2 * 0 > \theta \Rightarrow w1 > \theta$

$w1 * 1 + w2 * 1 \leq \theta \Rightarrow w1 + w2 \leq \theta$

No set of parameter values satisfy these inequalities

Perceptron training

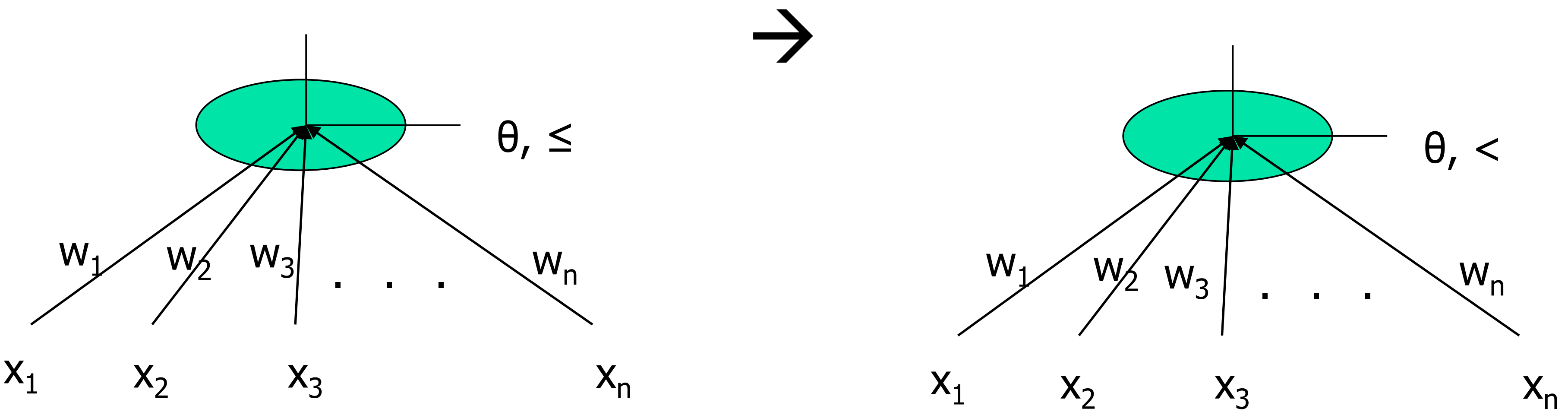
Perceptron Training Algorithm (PTA)

Preprocessing:

1. The computation law is modified to

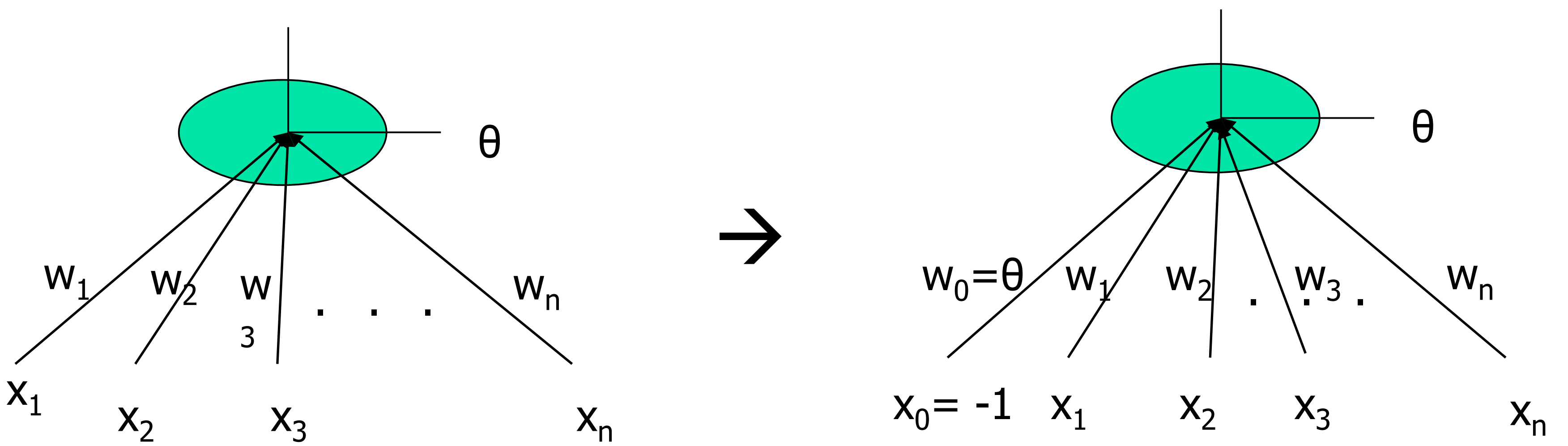
$$y = 1 \text{ if } \sum w_i x_i > \theta$$

$$y = 0 \text{ if } \sum w_i x_i < \theta$$



PTA – preprocessing cont…

2. Absorb θ as a weight



3. Negate all the zero-class examples (***why?***)

Example to demonstrate preprocessing

OR perceptron

1-class <1,1> , <1,0> , <0,1>
0-class <0,0>

Augmented x vectors:-
1-class <-1,1,1> , <-1,1,0> , <-1,0,1>
0-class <-1,0,0>

Negate 0-class:- <1,0,0>

Example to demonstrate preprocessing cont..

Now the vectors are

	x_0	x_1	x_2
X_1	-1	0	1
X_2	-1	1	0
X_3	-1	1	1
X_4	1	0	0

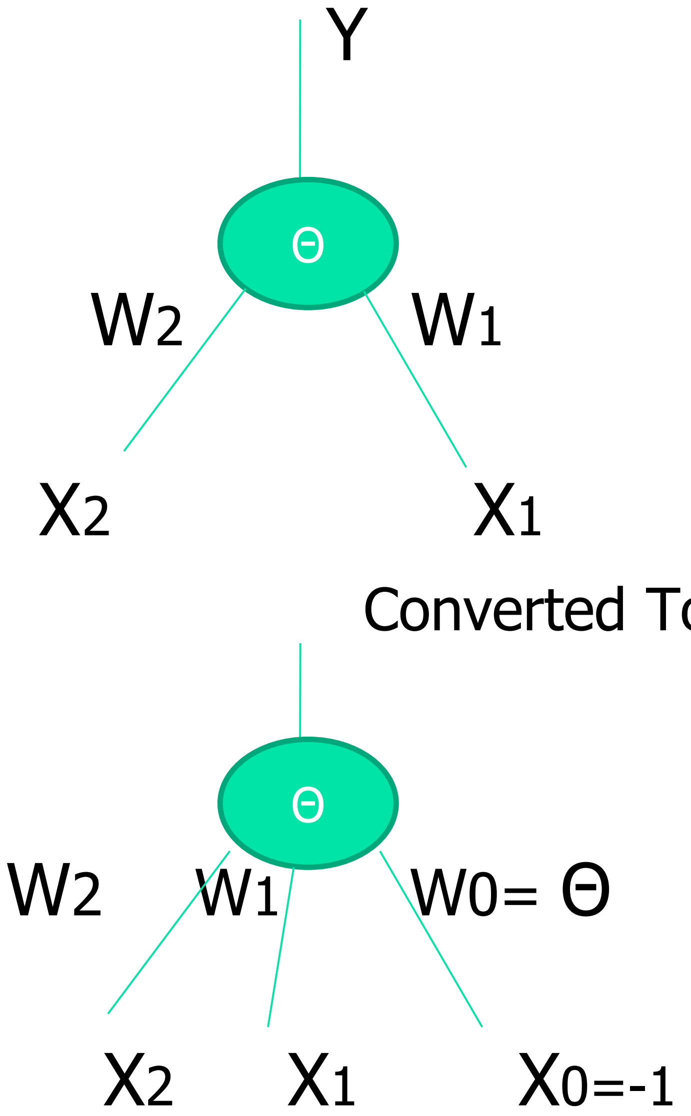
Perceptron Training Algorithm

1. Start with a random value of w
   ex: <0,0,0…>
2. Test for $wx_i > 0$
   If the test succeeds for i=1,2,…n
   then return w
3. Modify w, $w_{next} = w_{prev} + x_{fail}$

PTA on NAND

NAND:

X_2	X_1	Y
0	0	1
0	1	1
1	0	1
1	1	0



Preprocessing

NAND Augmented:

X_2	X_1	X_0	Y
0	0	-1	1
0	1	-1	1
1	0	-1	1
1	1	-1	0

NAND-0 class Negated

	X_2	X_1	X_0
V_0:	0	0	-1
V_1:	0	1	-1
V_2:	1	0	-1
V_3:	-1	-1	1

Vectors for which $W = \langle W2\ W1\ W0 \rangle$ has to be found such that $W.\ V_i > 0$

PTA Algo steps

Algorithm:

1. Initialize and Keep adding the failed vectors until $W.\ V_i > 0$ is true

Step 0:

$W = \langle 0, 0, 0 \rangle$

$W_1 = \langle 0, 0, 0 \rangle + \langle 0, 0, -1 \rangle$ {V_0 Fails}

$= \langle 0, 0, -1 \rangle$

$W_2 = \langle 0, 0, -1 \rangle + \langle -1, -1, 1 \rangle$ {V_3 Fails}

$= \langle -1, -1, 0 \rangle$

$W_3 = \langle -1, -1, 0 \rangle + \langle 0, 0, -1 \rangle$ {V_0 Fails}

$= \langle -1, -1, -1 \rangle$

$W_4 = \langle -1, -1, -1 \rangle + \langle 0, 1, -1 \rangle$ {V_1 Fails}

$= \langle -1, 0, -2 \rangle$

Trying convergence

$W_5 = \langle -1, 0, -2 \rangle + \langle -1, -1, 1 \rangle$ $\{V_3 \text{ Fails}\}$

$= \langle -2, -1, -1 \rangle$

$W_6 = \langle -2, -1, -1 \rangle + \langle 0, 1, -1 \rangle$ $\{V_1 \text{ Fails}\}$

$= \langle -2, 0, -2 \rangle$

$W_7 = \langle -2, 0, -2 \rangle + \langle 1, 0, -1 \rangle$ $\{V_0 \text{ Fails}\}$

$= \langle -1, 0, -3 \rangle$

$W_8 = \langle -1, 0, -3 \rangle + \langle -1, -1, 1 \rangle$ $\{V_3 \text{ Fails}\}$

$= \langle -2, -1, -2 \rangle$

$W_9 = \langle -2, -1, -2 \rangle + \langle 1, 0, -1 \rangle$ $\{V_2 \text{ Fails}\}$

$= \langle -1, -1, -3 \rangle$

Trying convergence

$W_{10} = <-1, -1, -3> + <-1, -1, 1>$ {V_3 Fails}
$= <-2, -2, -2>$
$W_{11} = <-2, -2, -2> + <0, 1, -1>$ {V_1 Fails}
$= <-2, -1, -3>$
$W_{12} = <-2, -1, -3> + <-1, -1, 1>$ {V_3 Fails}
$= <-3, -2, -2>$
$W_{13} = <-3, -2, -2> + <0, 1, -1>$ {V_1 Fails}
$= <-3, -1, -3>$
$W_{14} = <-3, -1, -3> + <0, 1, -1>$ {V_2 Fails}
$= <-2, -1, -4>$

W15 = <-2, -1, -4> + <-1, -1, 1> {V3 Fails}
= <-3, -2, -3>
W16 = <-3, -2, -3> + <1, 0, -1> {V2 Fails}
= <-2, -2, -4>
W17 = <-2, -2, -4> + <-1, -1, 1> {V3 Fails}
= <-3, -3, -3>
W18 = <-3, -3, -3> + <0, 1, -1> {V1 Fails}
= <-3, -2, -4>

W2 = -3, W1 = -2, W0 = Θ = -4

Succeeds for all vectors

Statement of Convergence of PTA

- Statement:

  Whatever be the initial choice of weights and whatever be the vector chosen for testing, PTA converges if the vectors are from a linearly separable function.

Proof of Convergence of PTA

- Suppose w_n is the weight vector at the n^{th} step of the algorithm
- At the beginning, the weight vector is w_0
- Go from w_i to w_{i+1} when a vector X_j fails the test $w_i X_j > 0$ and update w_i as

$$w_{i+1} = w_i + X_j$$

- Since Xjs form a linearly separable function,

$$\exists\ w^* \text{ s.t. } w^* X_j > 0\ \forall j$$

Proof of *Convergence* of PTA (cntd.)

- Consider the expression

$$G(w_n) = \frac{w_n \cdot w^*}{|w_n|}$$

where w_n = weight at ***nth*** iteration

- $G(w_n) = \dfrac{|w_n| \cdot |w^*| \cdot \cos\theta}{|w_n|}$

where θ = angle between w_n and w^*

- $G(w_n) = |w^*| \cdot \cos\theta$
- $G(w_n) \leq |w^*|$ (as $-1 \leq \cos\theta \leq 1$)

Behavior of Numerator of G

$w_n \,.\, w^* \;=\; (w_{n-1} + X^{n-1}_{fail}) \,.\, w^*$

$= w_{n-1} \,.\, w^* + X^{n-1}_{fail} \,.\, w^*$

$= (w_{n-2} + X^{n-2}_{fail}) \,.\, w^* + X^{n-1}_{fail} \,.\, w^* \; .....$

$= w_0 \,.\, w^* + (X^0_{fail} + X^1_{fail} + ... + X^{n-1}_{fail}).\, w^*$

$w^*.X^i_{fail}$ is always positive: *note carefully*

- Suppose $|X_j| \geq \delta$, where δ is the minimum magnitude
- Num of G $\geq |w_0 \,.\, w^*| + n\,\delta \,.\, |w^*|$
- So, numerator of G grows with n

Behavior of Denominator of G

- $|w_n| = \sqrt{w_n \cdot w_n}$

$= \sqrt{(w_{n-1} + X^{n-1}_{fail})^2}$

$= \sqrt{(w_{n-1})^2 + 2.\, w_{n-1}.\, X^{n-1}_{fail} + (X^{n-1}_{fail})^2}$

$\leq \sqrt{(w_{n-1})^2 + (X^{n-1}_{fail})^2}$ (as $w_{n-1}.\, X^{n-1}_{fail} \leq 0$)

$\leq \sqrt{(w_0)^2 + (X^0_{fail})^2 + (X^1_{fail})^2 + \ldots + (X^{n-1}_{fail})^2}$

- $|X_j| \leq \rho$ (max magnitude)
- So, Denom $\leq \sqrt{(w_0)^2 + n\rho^2}$

Some Observations

- Numerator of G grows as n
- Denominator of G grows as $\sqrt{n}$

  => Numerator grows faster than denominator

If PTA does not terminate, $G(w_n)$ values will become unbounded

Some Observations contd.

- But, as $|G(w_n)| \leq |w^*|$ which is finite, this is impossible!
- Hence, PTA has to converge

Proof is due to Marvin Minsky

A Problem that can be solved using the proof of PTA

Problem: *If a weight repeats while training the perceptron, then the function is not linearly separable.*

Proof

Let us prove first $w_n . w^*$ is an increasing function

From the proof of convergence of PTA, we can write

$$w_n . w^* = (w_{n-1} + X^{n-1}_{fail}) . w^*$$
$$= w_{n-1} . w^* + w^* . X^{n-1}_{fail}$$

Since w^* is optimal weight vector therefore:

$$w^* . X^{n-1}_{fail} > 0$$

Proof cntd.

Because in every iteration we are adding +ve number $w^*. X^{n-1}{}_{fail}$

Therefore:

$$w_n . w^* > w_{n-1} . w^* \qquad (1)$$

Hence $w_n . w^*$ is an increasing function

According to the claim made by theorem, if weight repeats then the weight w_i at a given iteration i, will be equal to the weight w_{i+k} at a given iteration $(i+k)$ where k is a +ve number

$$w_{i} = w_{i+k}$$

Proof cntd.

Therefore:

$$w_i \,.\, w^* = w_{i+k} \,.\, w^* \qquad (2)$$

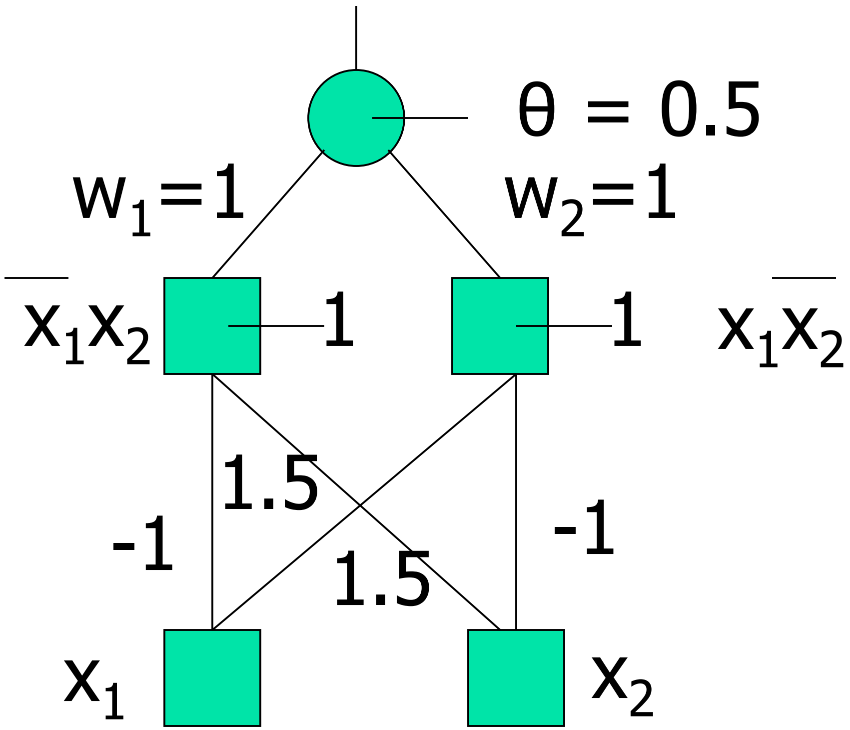
(2) contradicts the (1)

Hence no w* exists

So function is not linearly separable.

Feedforward Network and Backpropagation

Example - XOR



Gradient Descent Technique

- Let E be the error at the output layer

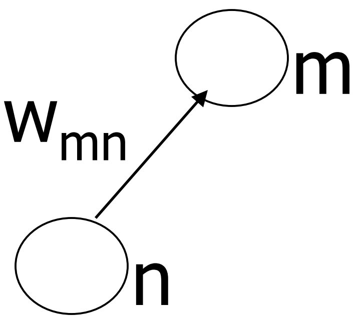
$$E = \frac{1}{2} \sum_{j=1}^{p} \sum_{i=1}^{n} (t_i - o_i)_j^2$$

- t_i = target output; o_i = observed output

- i is the index going over n neurons in the outermost layer
- j is the index going over the p patterns (1 to p)
- Ex: XOR:– p=4 and n=1

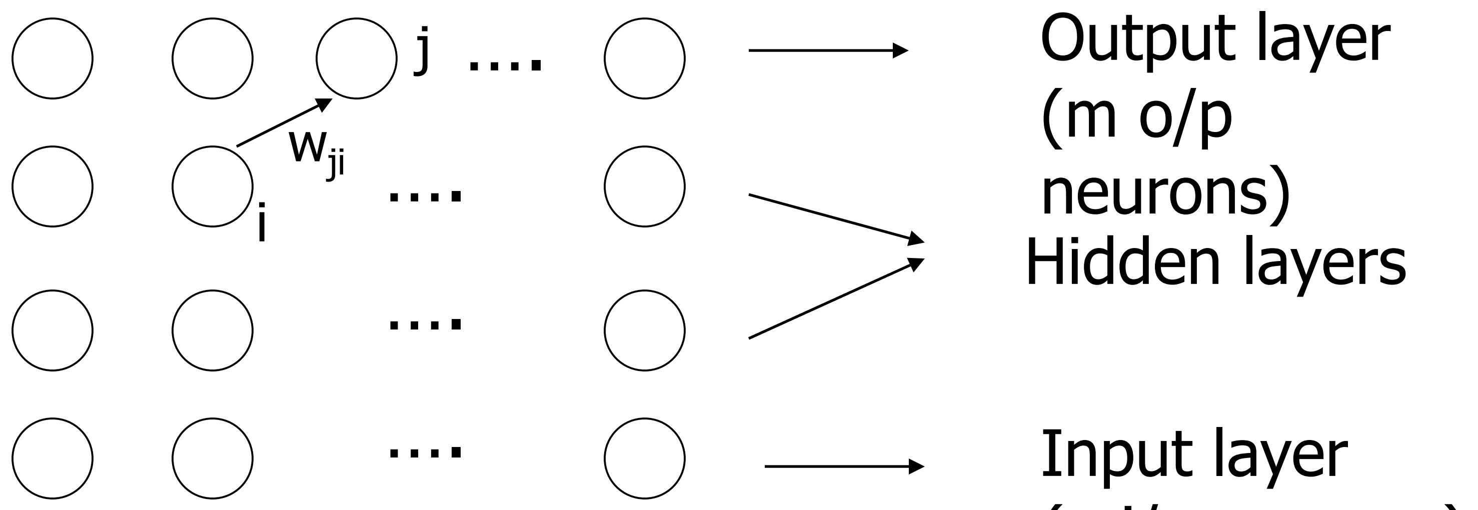
Weights in a FF NN

- w_{mn} is the weight of the connection from the n^{th} neuron to the m^{th} neuron
- E *vs* $\overline{W}$ surface is a complex surface in the space defined by the weights w_{ij}
- $-\frac{\delta E}{\delta w_{mn}}$ gives the direction in which a movement of the operating point in the w_{mn} co-ordinate space will result in maximum decrease in error



$$\Delta w_{mn} \propto -\frac{\delta E}{\delta w_{mn}}$$

Backpropagation algorithm



- Fully connected feed forward network
- Pure FF network (*no jumping of connections over layers*)

Gradient Descent Equations

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}} (\eta = \text{learning rate, } 0 \le \eta \le 1)$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}} (net_j = \text{input at the j}^{th} \text{ layer})$$

$$\frac{\delta E}{\delta net_j} = -\delta j$$

$$\Delta w_{ji} = \eta \delta j \frac{\delta net_j}{\delta w_{ji}} = \eta \delta j o_i$$

Backpropagation – **for outermost layer**

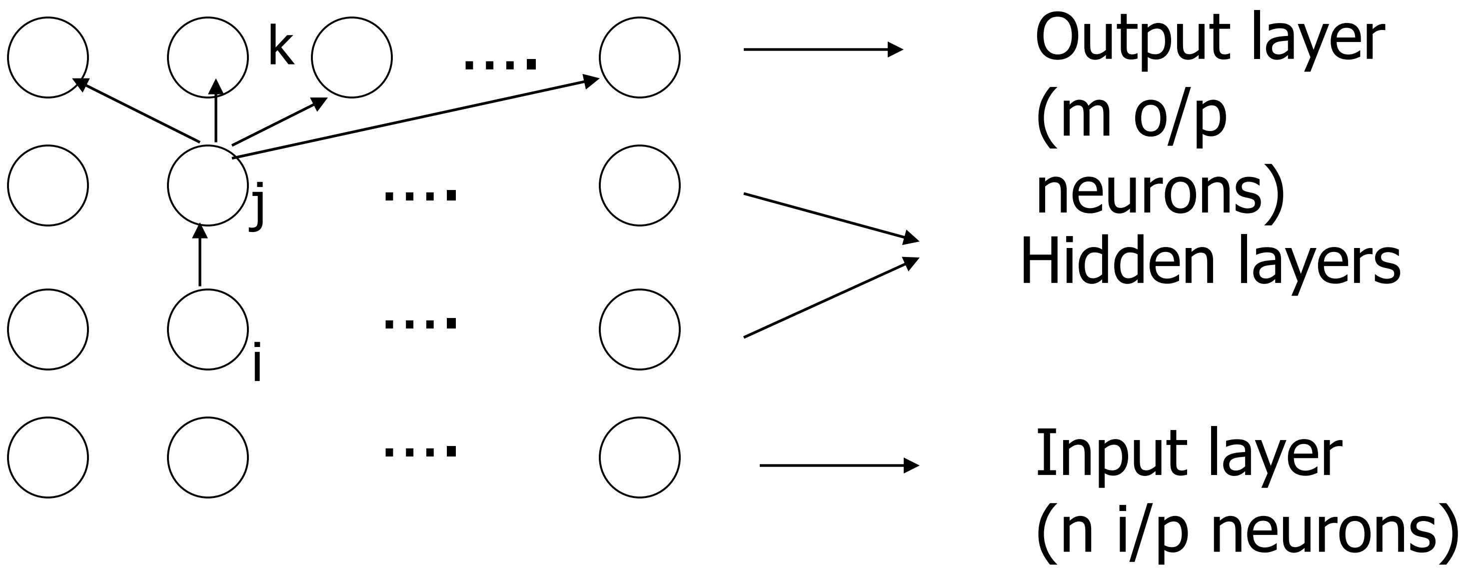
$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} (net_j = \text{input at the j}^{th} \text{ layer})$$

$$E = \frac{1}{2} \sum_{p=1}^{m} (t_p - o_p)^2$$

$$\text{Hence, } \delta j = -(-(t_j - o_j) o_j (1 - o_j))$$

$$\Delta w_{ji} = \eta (t_j - o_j) o_j (1 - o_j) o_i$$

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j}$$

$$= -\frac{\delta E}{\delta o_j} \times o_j(1 - o_j)$$

This recursion can give rise to vanishing and exploding Gradient problem

$$= -\sum_{k \in \text{next layer}} \left(\frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j}\right) \times o_j(1 - o_j)$$

$$\text{Hence, } \delta_j = -\sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times o_j(1 - o_j)$$

$$= \sum_{k \in \text{next layer}} (w_{kj}\delta_k) o_j(1 - o_j)$$

General Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \quad \text{for hidden layers}$$

References

- Pattern Recognition and Machine Learning. Christopher M. Bishop, Springer

Thank you