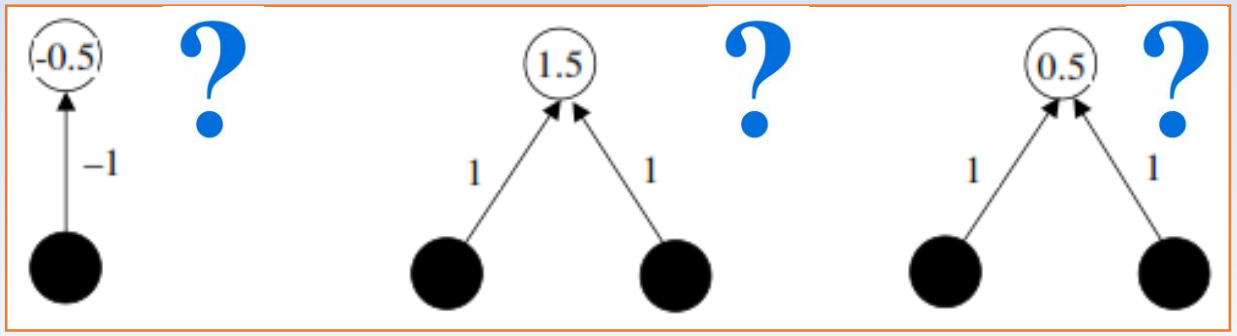
Notes on Neural Networks

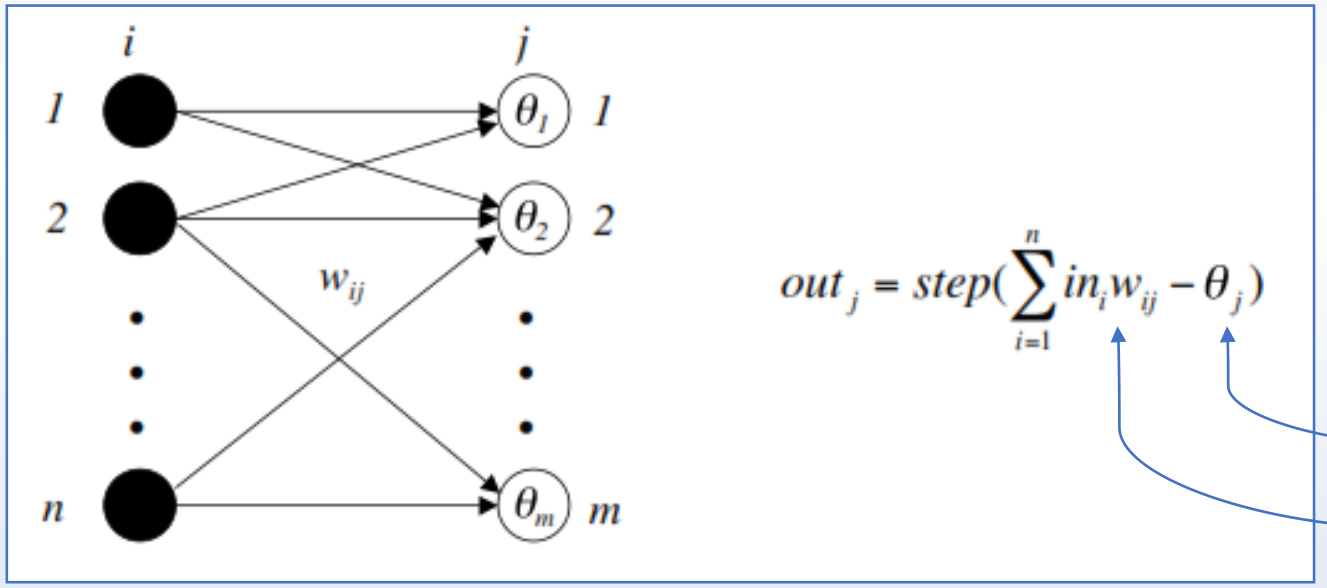
B
- Breadboard
+ Blackboard
approach

McCulloch-Pitts Neuron

$$out_i = step(\sum_{k=1}^{n} in_{ki} - \theta_i)$$



The Perceptron



$$out_j = step(\sum_{i=1}^{n} in_i w_{ij} - \theta_j)$$

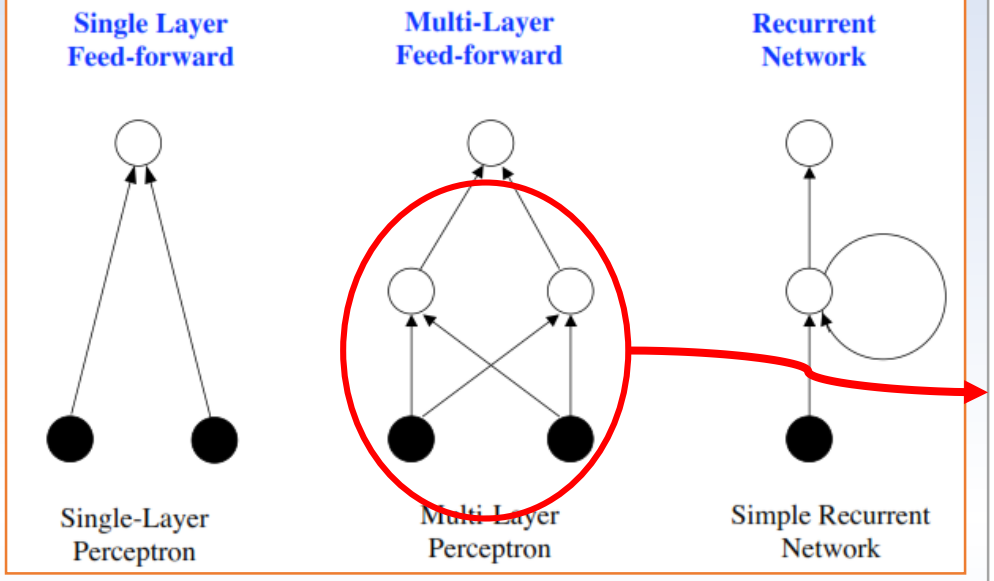
Bias/Threshold

Weights

XOR ?

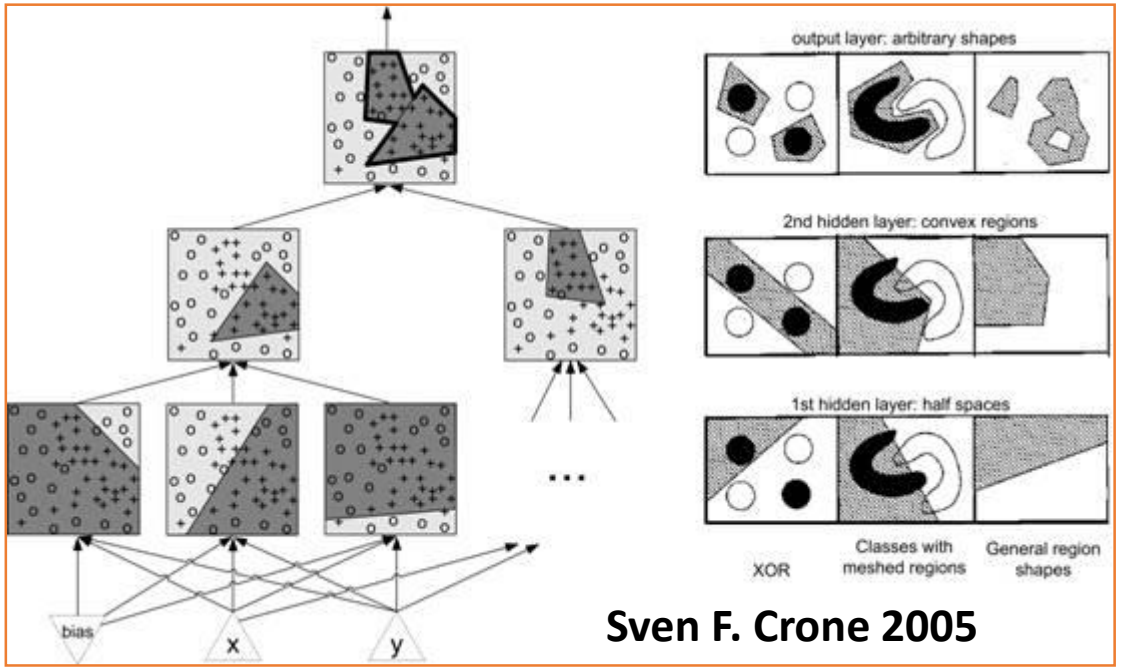
non-linearly separable.

Examples of Network Architecture Types

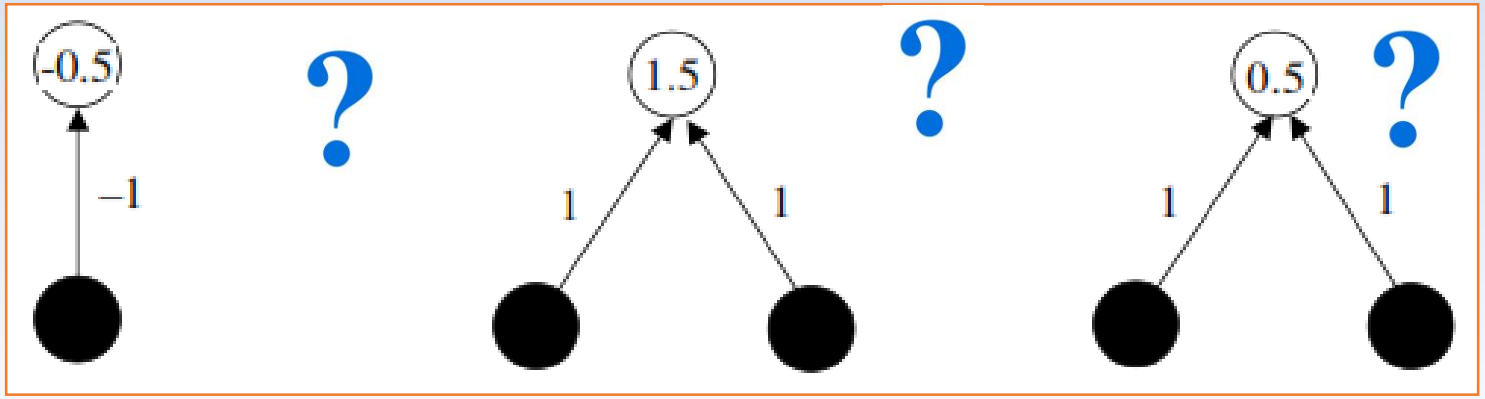


Decision Hyperplanes and Linear Separability

$$w_1 in_1 + w_2 in_2 + \ldots + w_n in_n - \theta = 0$$



Sven F. Crone 2005



NOT

in	*out*
0	1
1	0

AND

in_1	in_2	*out*
0	0	0
0	1	0
1	0	0
1	1	1

OR

in_1	in_2	*out*
0	0	0
0	1	1
1	0	1
1	1	1

Finding Weights Analytically for the AND Network

$$out = step(w_1 in_1 + w_2 in_2 - \theta)$$

in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$\Rightarrow$

$$\begin{aligned} w_1 0 + w_2 0 - \theta &< 0 \\ w_1 0 + w_2 1 - \theta &< 0 \\ w_1 1 + w_2 0 - \theta &< 0 \\ w_1 1 + w_2 1 - \theta &\geq 0 \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} \theta &> 0 \\ w_2 &< \theta \\ w_1 &< \theta \\ w_1 + w_2 &\geq \theta \end{aligned}$$

AND

in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

Finding Weights Analytically

$$out = step(w_1 in_1 + w_2 in_2 - \theta)$$

XOR

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

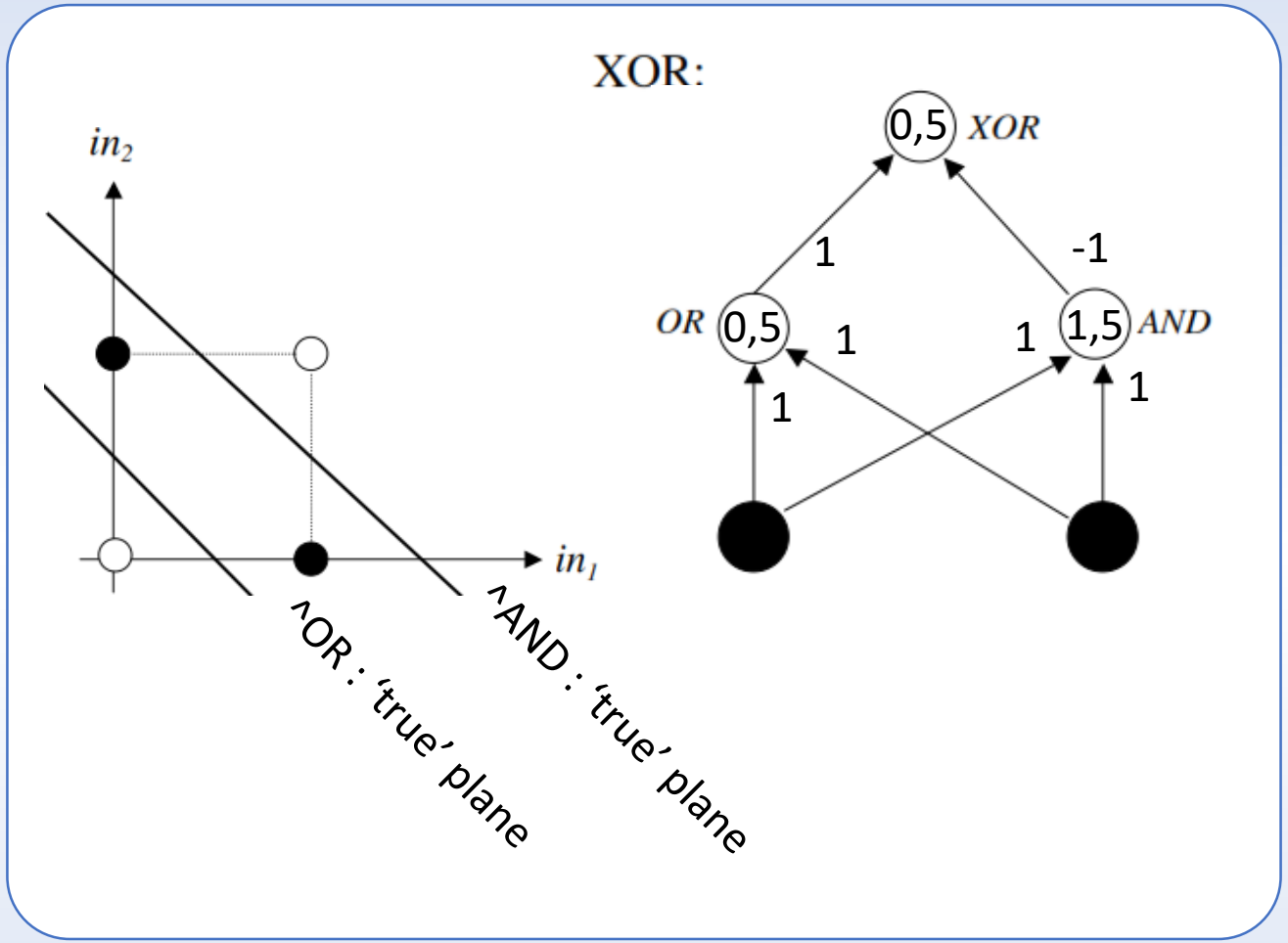
$\Rightarrow$

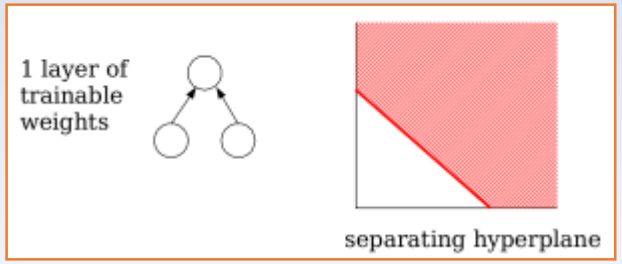
$$\begin{aligned} w_1 0 + w_2 0 - \theta &< 0 \\ w_1 0 + w_2 1 - \theta &\geq 0 \\ w_1 1 + w_2 0 - \theta &\geq 0 \\ w_1 1 + w_2 1 - \theta &< 0 \end{aligned}$$

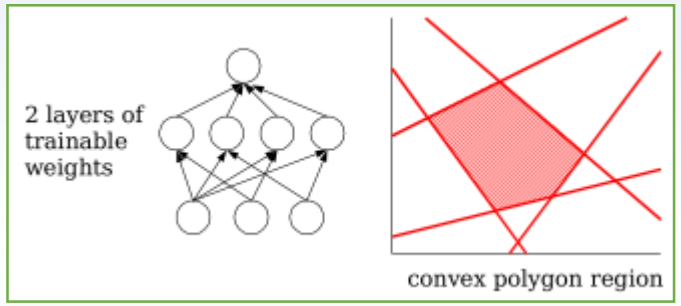
$\Rightarrow$

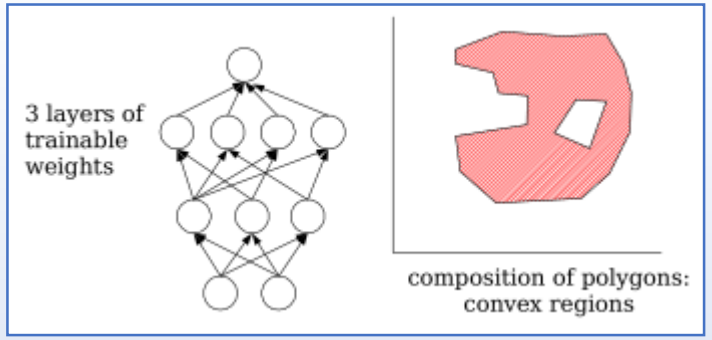
$$\begin{aligned} \theta &> 0 \\ w_2 &\geq \theta \\ w_1 &\geq \theta \\ w_1 + w_2 &< \theta \end{aligned}$$

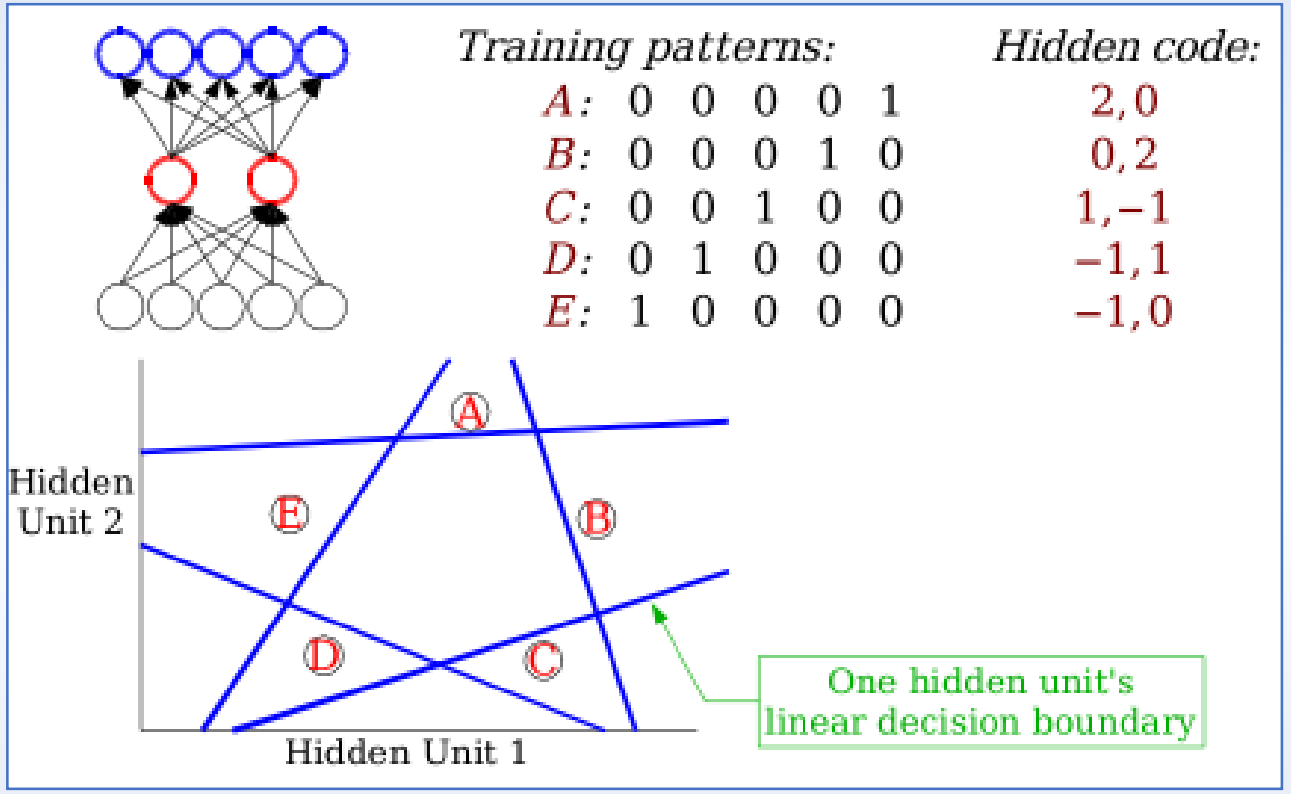
?







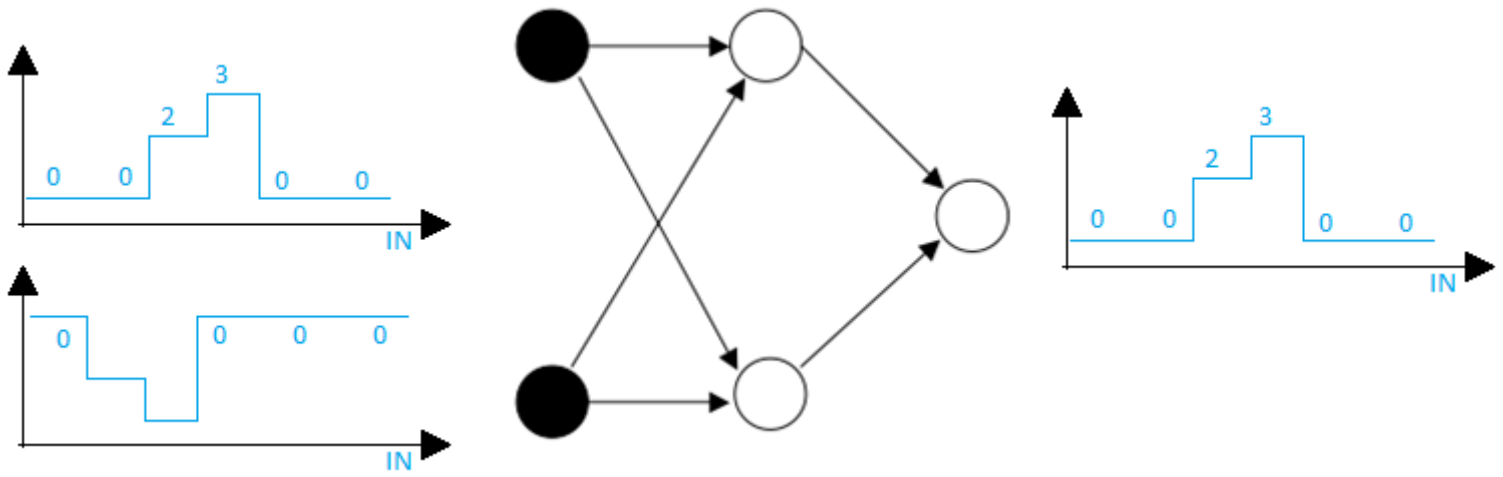




Dave Touretzky https://www.cs.cmu.edu/afs/cs/academic/class/15782-f06/

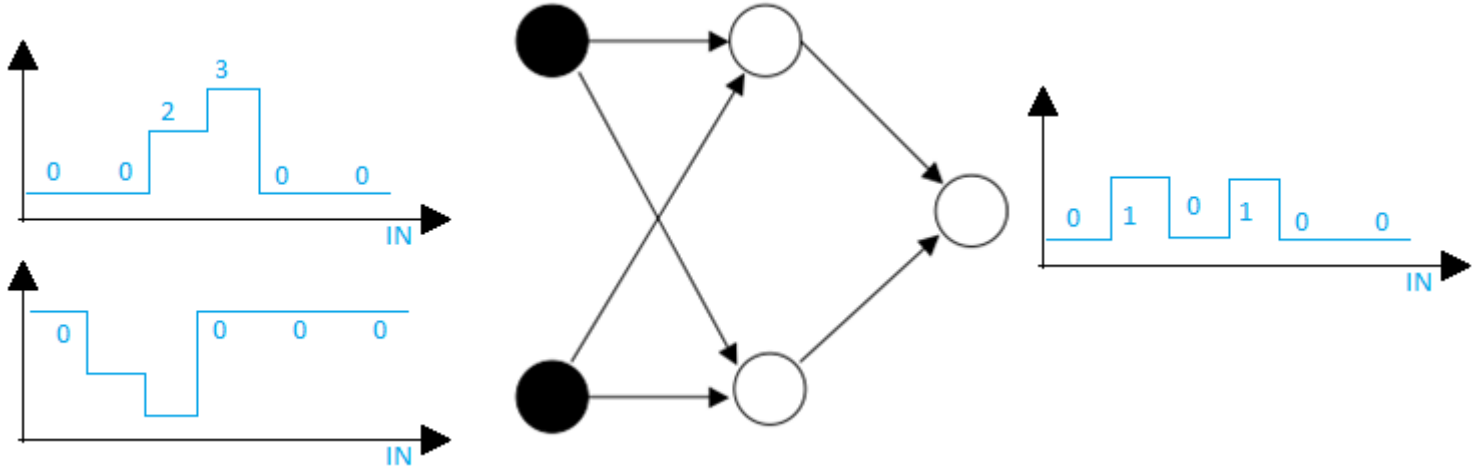
Training a Neural Network

network weights at time t are $w_{ij}(t)$ $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$



Training a Neural Network

network weights at time t are $w_{ij}(t)$ $\quad w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$



Perceptron learning

$$\Delta w_{ij} = \eta.(targ_j - out_j).in_i$$

Assuming one output neuron, the squared error function is:

$$E = \frac{1}{2}(t - y)^2,$$

where

E is the squared error,

t is the target output for a training sample, and

y is the actual output of the output neuron.

$$E = (t - y)^2,$$

where E is the discrepancy or error

Assuming one output neuron, the squared error function is:

$$E = \frac{1}{2}(t - y)^2,$$

where

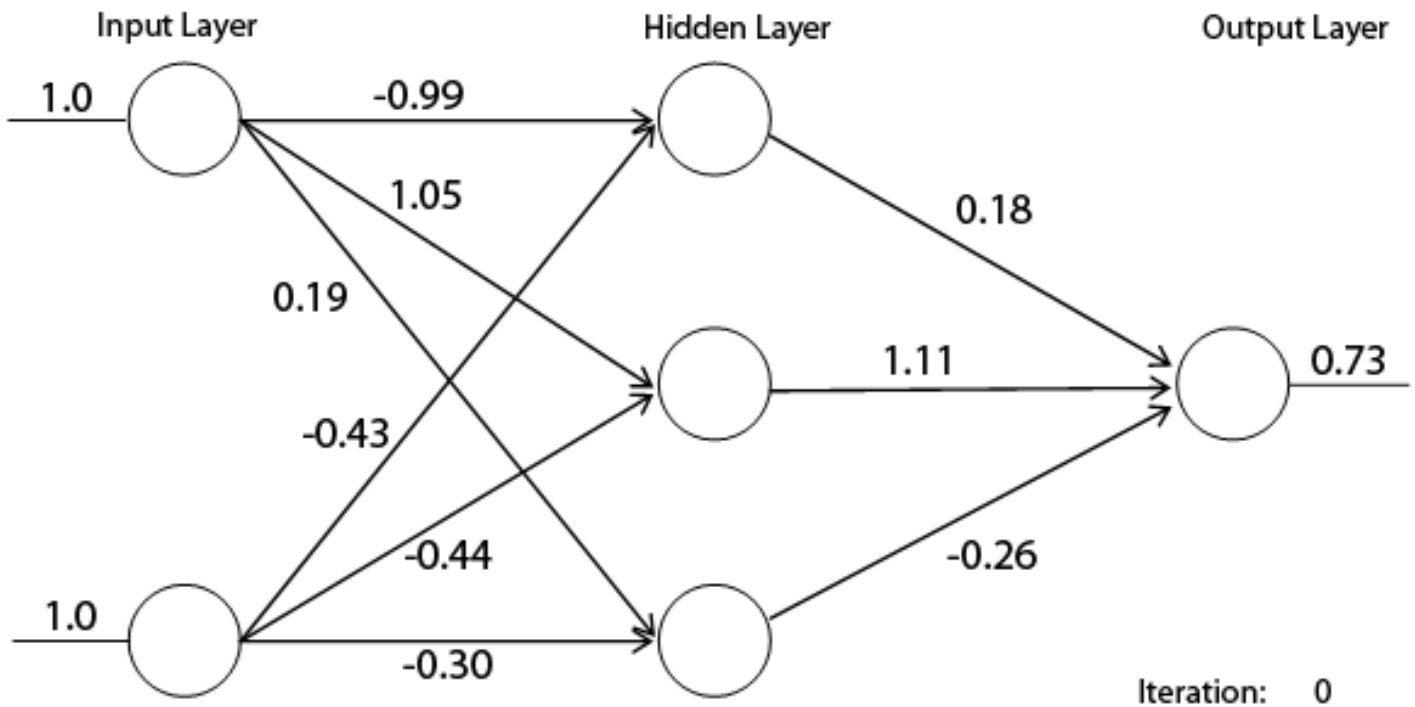
E is the squared error,

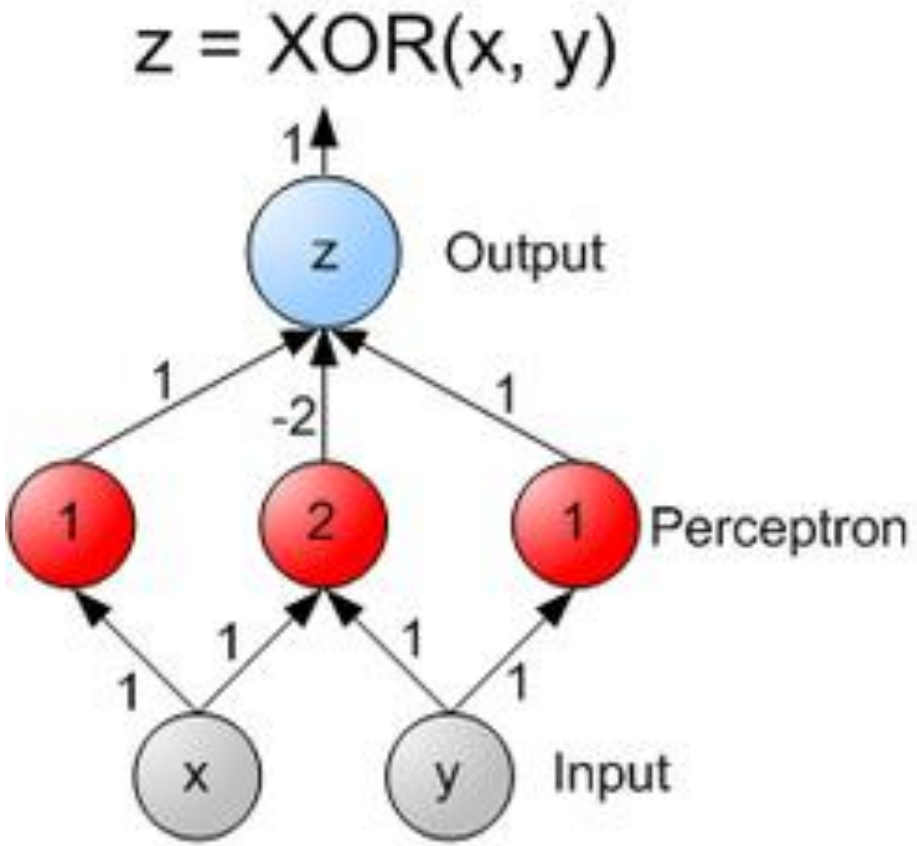
t is the target output for a traini

y is the actual output of the out

$$E = (t - y)^2,$$

where E is the discrepancy or error




z = XOR(x, y)
1
z
Output
1
-2
1
1
2
1
Perceptron
1
1
1
1
x
y
Input

Assuming one output neuron, the squared error function is:

$$E = \tfrac{1}{2}(t - y)^2,$$

where

E is the squared error,
t is the target output for a training sample, and
y is the actual output of the output neuron.

$$E = (t - y)^2,$$

where E is the discrepancy or error

Sum Squared Error (SSE)

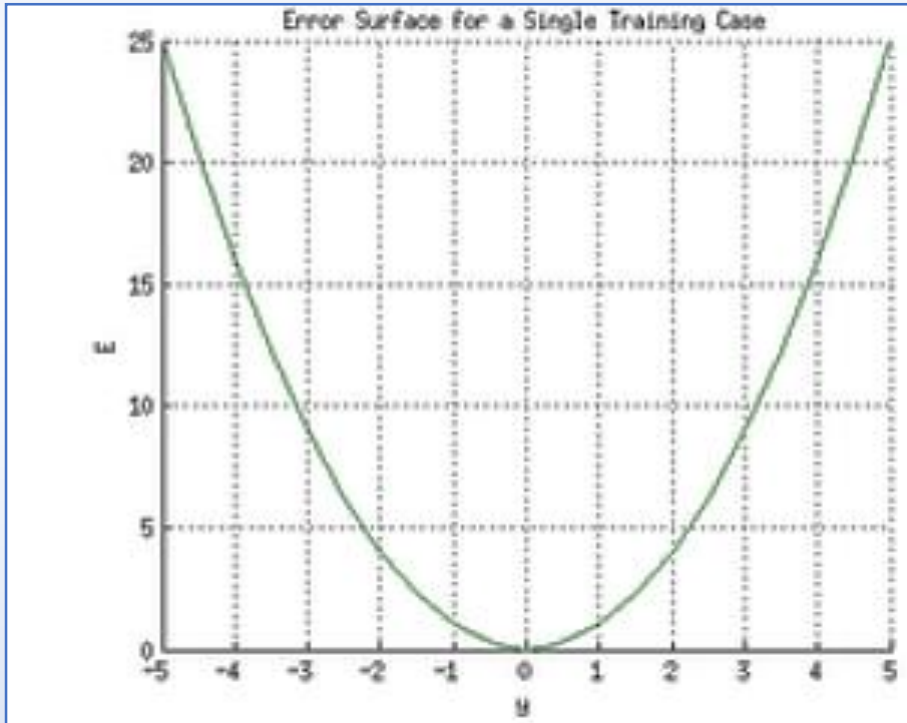
$$E_{SSE}(w_{ij}) = \tfrac{1}{2}\sum_{p}\sum_{j}\left(targ_j - out_j\right)^2$$

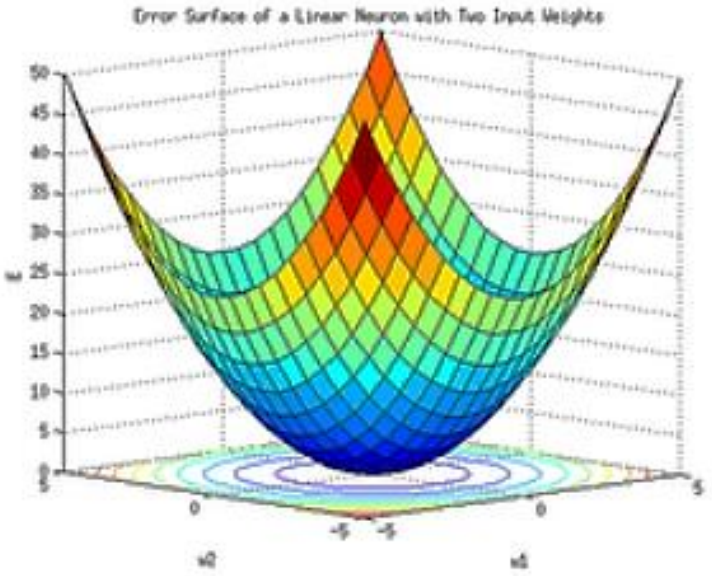
desired outputs $targ_j$
actual outputs out_j
output units j
training patterns p

Error Minimisation

$$E_{SSE}(w_{ij}) = \frac{1}{2}\sum_{p}\sum_{j}\left(targ_j - out_j\right)^2$$

$$\Delta w_{kl} = -\eta \frac{\partial E(w_{ij})}{\partial w_{kl}}$$





Error surface of a linear neuron with two input weights

$$\Delta w_{kl} = -\eta \frac{\partial E(w_{ij})}{\partial w_{kl}} \qquad E_{SSE}(w_{ij}) = \frac{1}{2}\sum_p \sum_j \left(targ_j - out_j\right)^2$$

gradient descent learning

$$\Delta w_{kl} = -\eta \frac{\partial}{\partial w_{kl}} \left[\frac{1}{2} \sum_p \sum_j \left(targ_j - f(\sum_i in_i w_{ij}) \right)^2 \right]$$

$$\Delta w_{kl} = -\eta \left[\frac{1}{2} \sum_p \sum_j \frac{\partial}{\partial w_{kl}} \left(targ_j - f(\sum_i in_i w_{ij}) \right)^2 \right]$$

$$\Delta w_{kl} = -\eta \left[\frac{1}{2} \sum_p \sum_j 2 \left(targ_j - f(\sum_i in_i w_{ij}) \right) \left(-\frac{\partial}{\partial w_{kl}} f(\sum_m in_m w_{mj}) \right) \right]$$

$$\Delta w_{kl} = \eta \left[\sum_p \sum_j \left(targ_j - f(\sum_i in_i w_{ij}) \right) \left(f'(\sum_n in_n w_{nj}) \frac{\partial}{\partial w_{kl}} (\sum_m in_m w_{mj}) \right) \right]$$

$$\Delta w_{kl} = \eta \left[\sum_p \sum_j \left(targ_j - f(\sum_i in_i w_{ij}) \right) \left(f'(\sum_n in_n w_{nj}) (\sum_m in_m \frac{\partial w_{mj}}{\partial w_{kl}}) \right) \right]$$

$$\Delta w_{kl} = \eta \left[\sum_p \sum_j \left(targ_j - f(\sum_i in_i w_{ij}) \right) \left(f'(\sum_n in_n w_{ij}) (\sum_m in_m \delta_{mk} \delta_{jl}) \right) \right]$$

$$\Delta w_{kl} = \eta \left[\sum_p \sum_j \left(targ_j - f(\sum_i in_i w_{ij}) \right) \left(f'(\sum_n in_n w_{nj}) (in_k \delta_{jl}) \right) \right]$$

$$\Delta w_{kl} = \eta \left[\sum_p \left(targ_l - f(\sum_i in_i w_{il}) \right) \left(f'(\sum_n in_n w_{nl}) (in_k) \right) \right]$$

$$\Delta w_{kl} = \eta \sum_p \left(targ_l - out_l \right) . f'(\sum_n in_n w_{nl}) . in_k$$

Delta Rule

$$\Delta w_{kl} = \eta \sum_p (targ_l - out_l).in_k$$

$$\Delta w_{kl} = -\eta \frac{\partial E(w_{ij})}{\partial w_{kl}}$$

$$E_{SSE}(w_{ij}) = \tfrac{1}{2} \sum_p \sum_j \left(targ_j - out_j\right)^2$$

$$\Delta w_{kl} = -\eta \frac{\partial}{\partial w_{kl}} \left[\tfrac{1}{2} \sum_p \sum_j \left(targ_j - f(\sum_i in_i w_{ij}) \right)^2 \right] \longrightarrow \Delta w_{kl} = \eta \sum_p (targ_l - out_l).f'(\sum_n in_n w_{nl}).in_k$$

gradient descent learning

Gradient Descent – General Weight Update Equation

$$\Delta w_{kl} = -\eta \frac{\partial E(\{w_{ij}\})}{\partial w_{kl}} \quad *$$

Gradient Descent Updates for Single Layer Perceptron

$$\Delta w_{kl} = \eta \sum_{p} (targ_l - out_l) in_k \quad *$$

Back-propagation with Momentum

$$delta_k^{(N)} = \left(targ_k - out_k^{(N)}\right)$$

$$delta_k^{(n)} = \left(\sum_k delta_k^{(n+1)} . w_{lk}^{(n+1)}\right) . f'\left(\sum_j out_j^{(n-1)} w_{jk}^{(n)}\right)$$

$$\Delta w_{kl}^{(n)}(t) = \eta \sum_p delta_l^{(n)}(t) . out_k^{(n-1)}(t) + \alpha . \Delta w_{kl}^{(n)}(t-1)$$

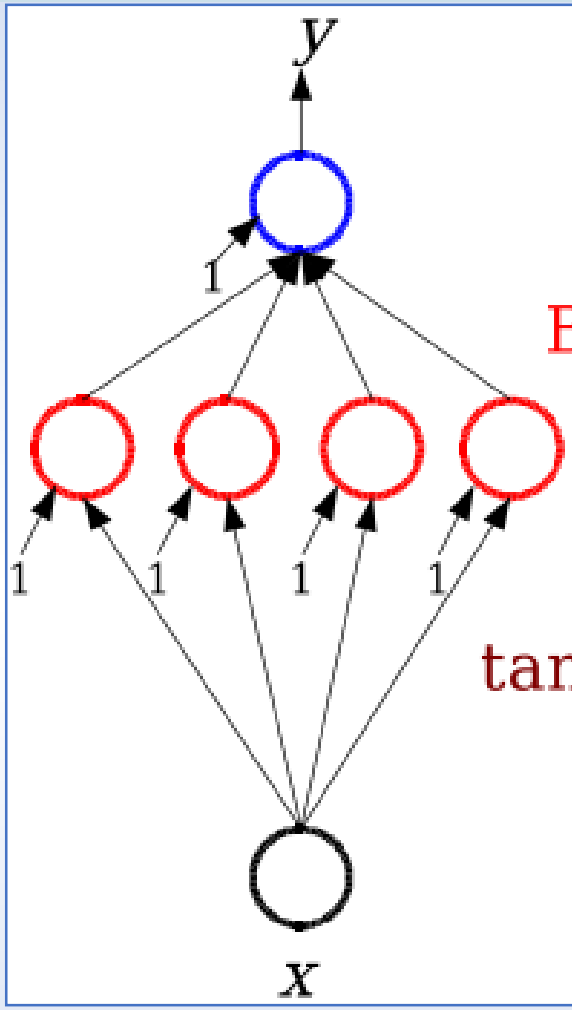
training pattern labels p

$$\sum_p \left(targ_{pl} - \sum_i in_{pi} w_{il} \right) in_{pk} = 0$$

? SELECT DESIRED PATTERN

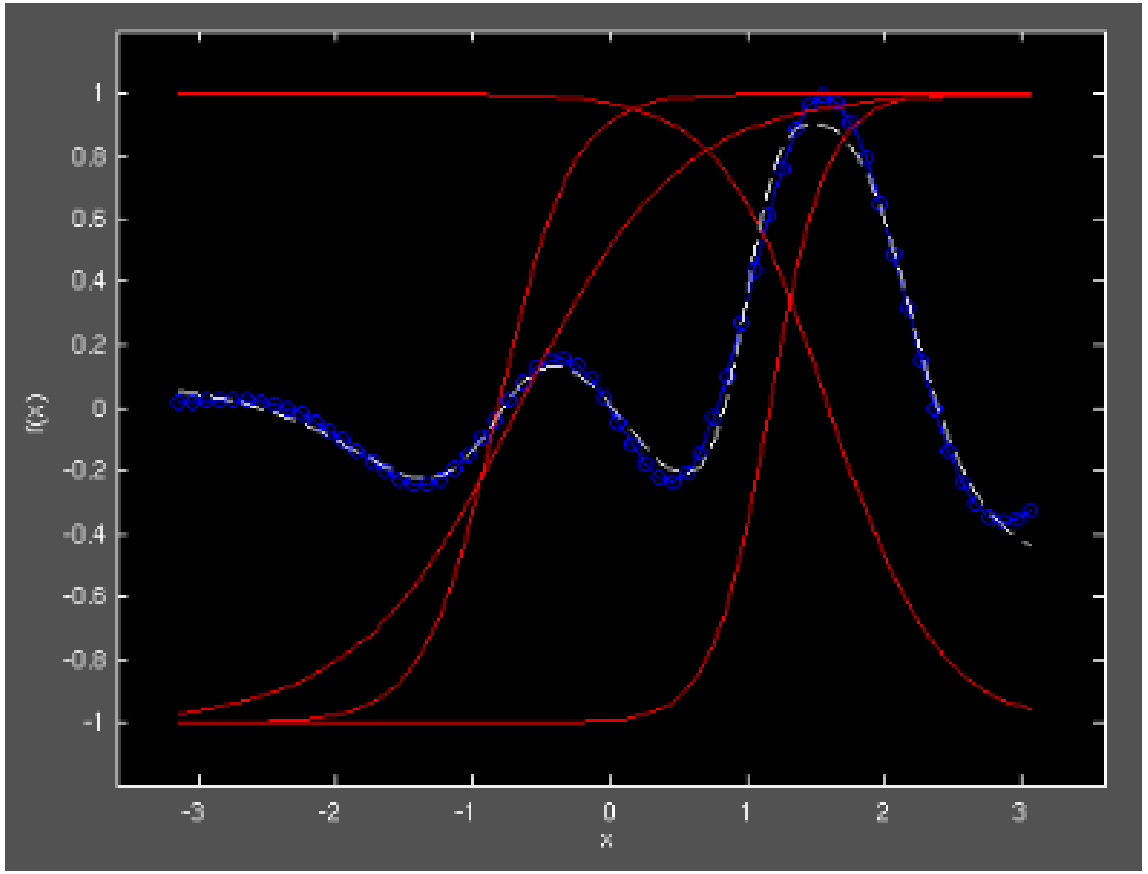
? VECTORIZE BACKPROPAGATION

THANKS

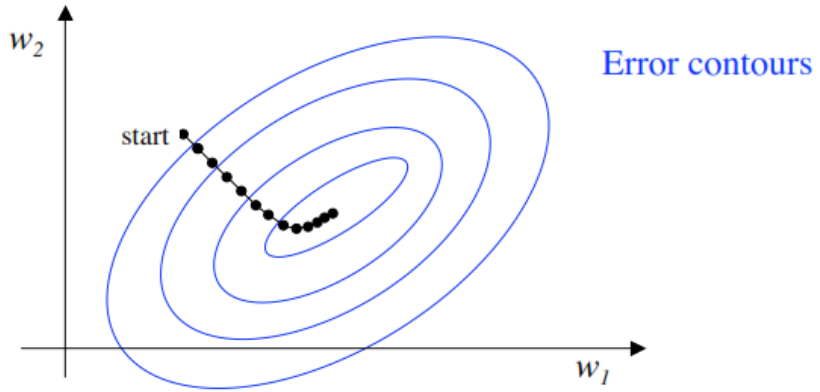


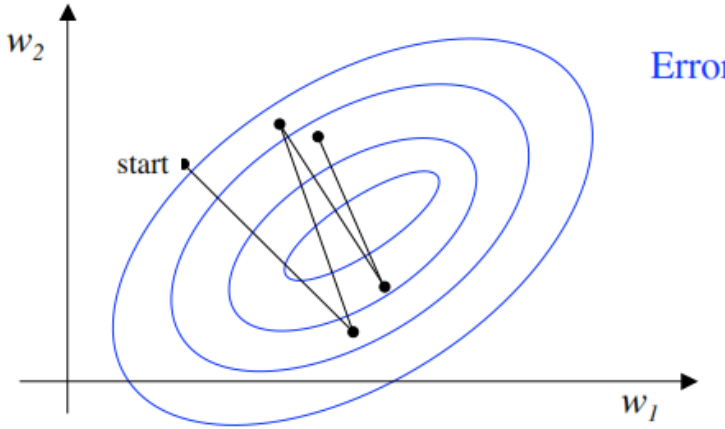
Bumps from which we compose f(x)

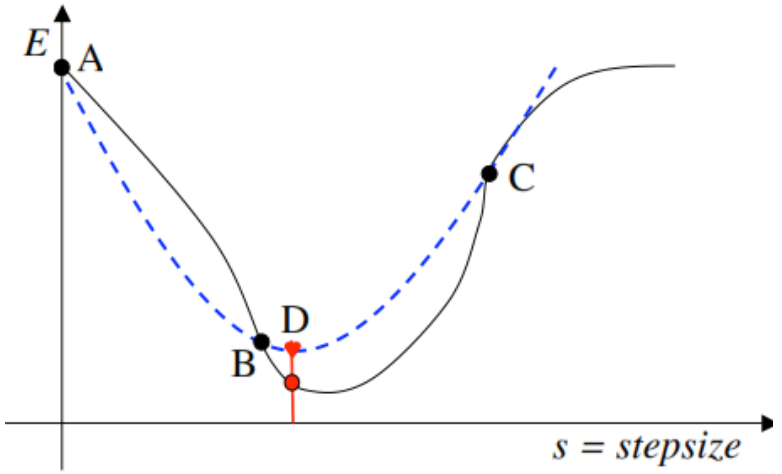
$$\tanh(w_0 + w_1 x)$$



Dave Touretzky https://www.cs.cmu.edu/afs/cs/academic/class/15782-f06/







$$y = x_1 w_1 + x_2 w_2,$$

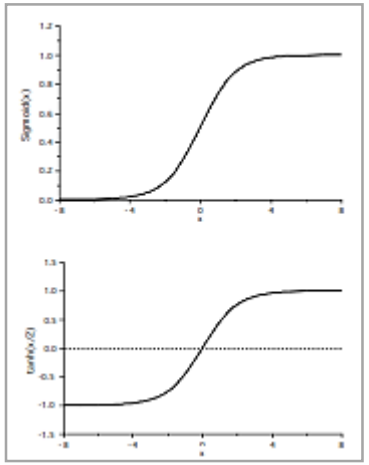
$$o_i = \varphi(\mathrm{net}_i) = \varphi\left(\sum_{k=1}^{n} w_{kj} o_k\right)$$

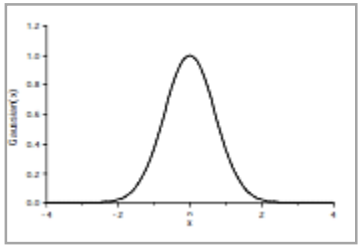
$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{d\varphi}{dz}(z) = \varphi(z)(1 - \varphi(z))$$

Radial Basis Function Networks

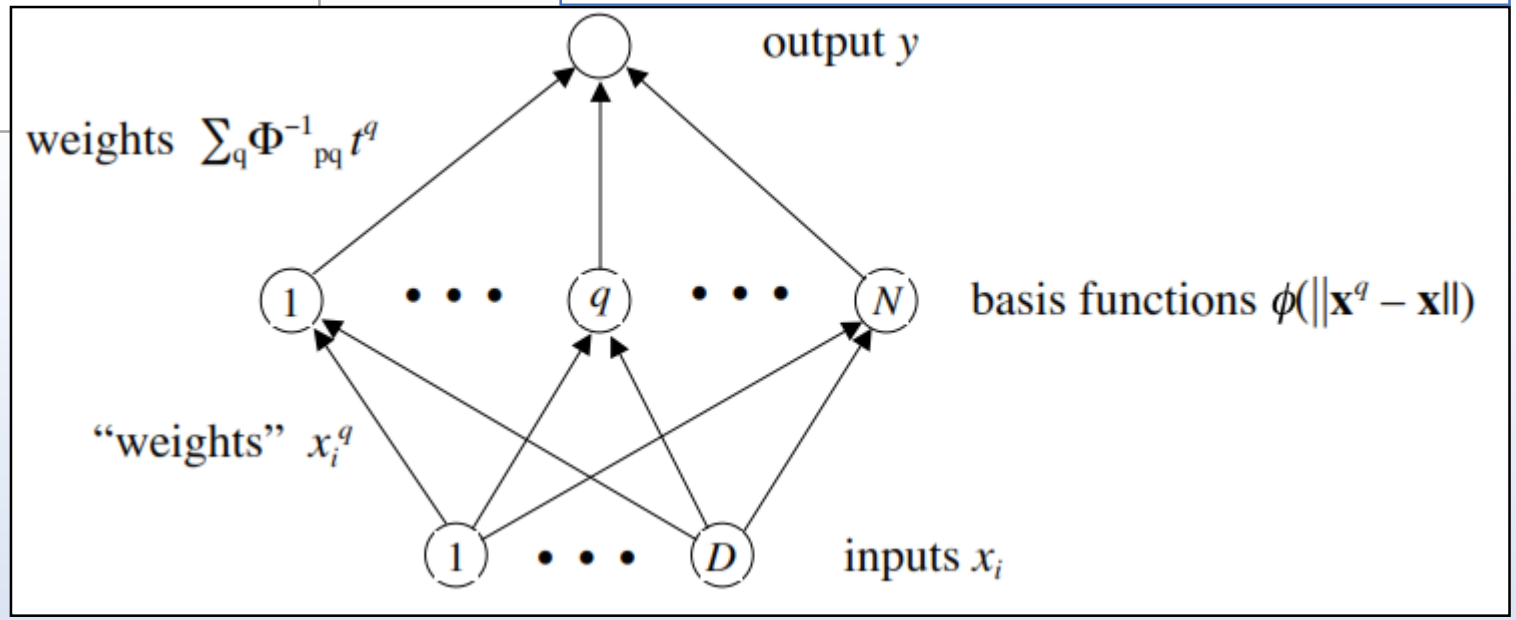
Activation/Transfer Function





Gaussian Radial

$$f(\mathbf{x}) = \sum_{p=1}^{N} w_p \phi_p(\mathbf{x}) = \sum_{p=1}^{N} w_p \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}^p\|^2}{2\sigma^2}\right)$$



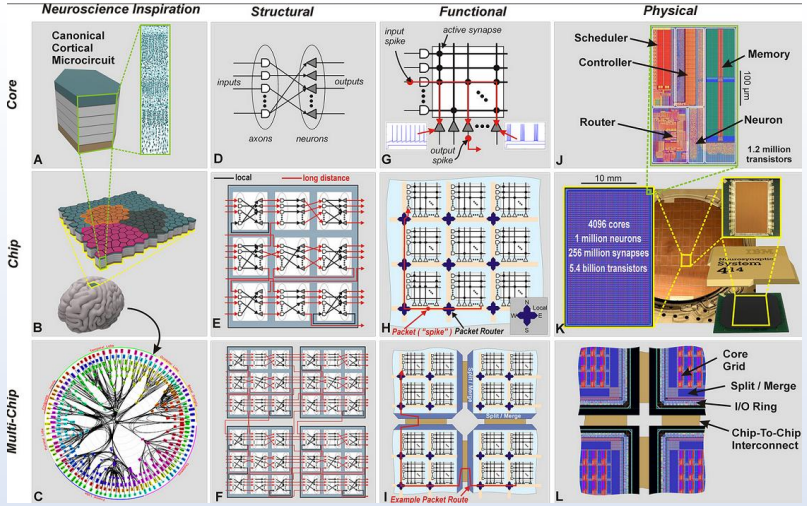
Intervallo 1

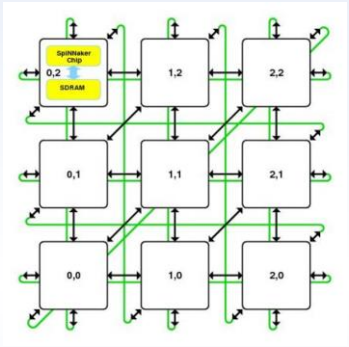
http://apt.cs.manchester.ac.uk/projects/SpiNNaker

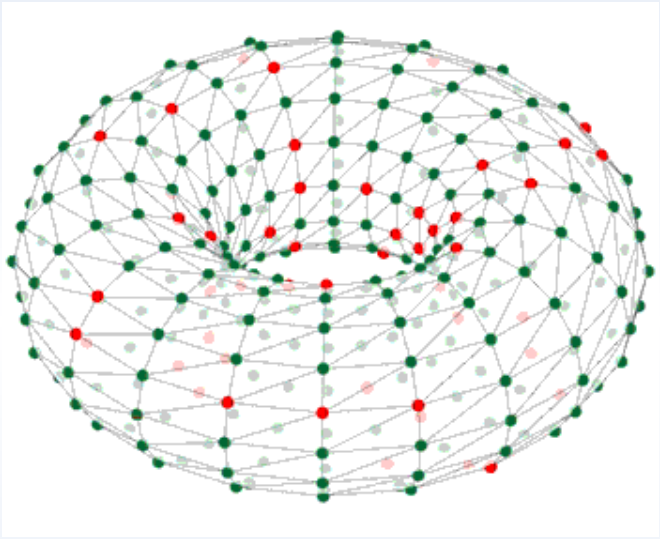
http://www.eenewseurope.com/news/spinnaker-neuromorphic-supercomputer-reaches-one-million-cores-0

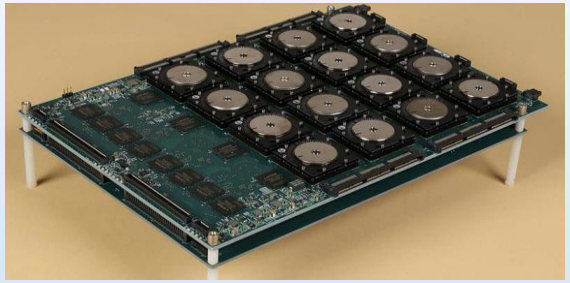
***Spi**king **N**eural **N**etwork **A**rchitecture*

DARPA SyNAPSE
IBM TrueNorth
2014









Intervallo 1

http://www.eenewseurope.com/news/cea-leti-proves-rram-based-tcams-viable-neuromorphic-processors