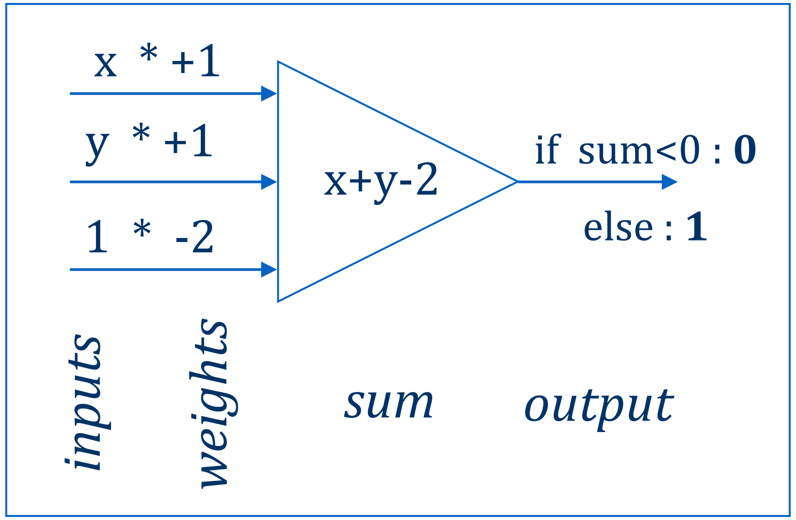
ANN

Sonal Ghanshani

Example

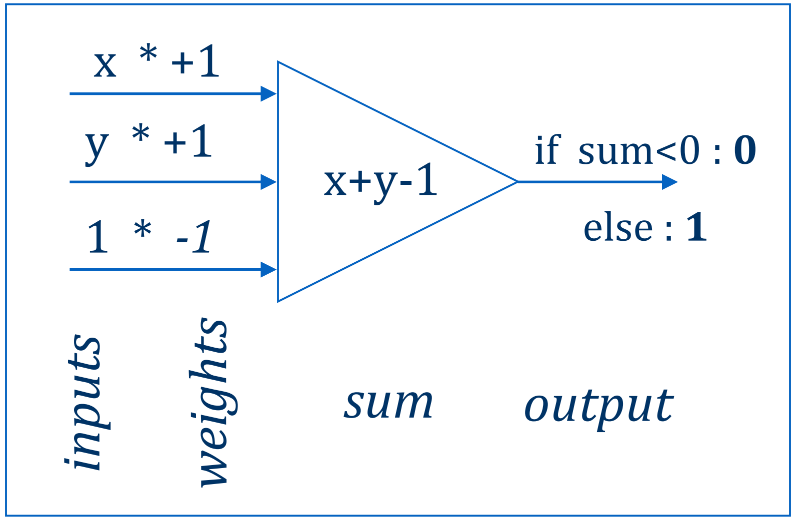


Truth Table for Logical AND

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

inputs *output*

Example

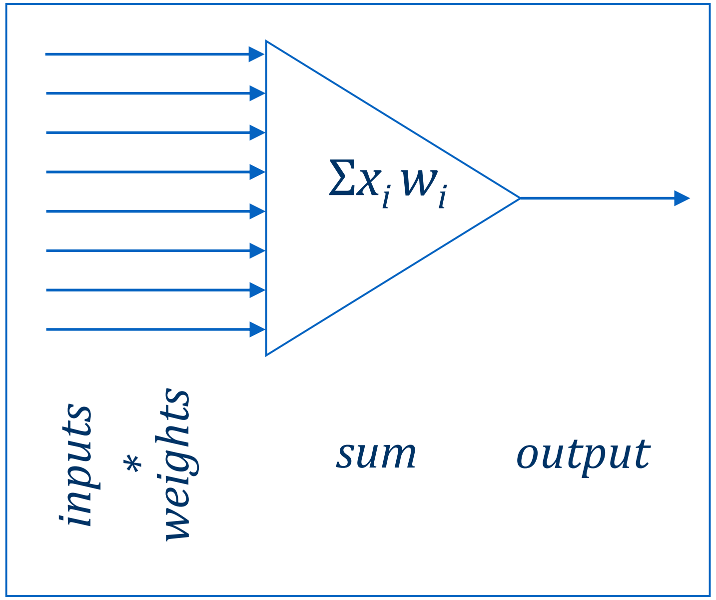


Truth Table for Logical OR

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

inputs *output*

Example



It obeyed the following rule:

If the sum of the weighted inputs exceeds a threshold, output 1, else output -1.

1 if $\Sigma\ input_i * weight_i > threshold$

-1 if $\Sigma\ input_i * weight_i < threshold$

Linear Neurons

The neuron has a real-valued output which is a weighted sum of its inputs

weight

vector

$$\hat{y} = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$

input

vector

Neuron's estimate of the desired output

The aim of learning is to minimize the discrepancy between the desired output and the actual output

Delta Rule

- Define the error as the squared residuals summed over all training cases:
- Now differentiate to get error derivatives for weights
- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases

$$E = \tfrac{1}{2}\sum_n (y_n - \hat{y}_n)^2$$

$$\frac{\partial E}{\partial w_i} = \tfrac{1}{2}\sum_n \frac{\partial \hat{y}_n}{\partial w_i}\frac{\partial E_n}{\partial \hat{y}_n}$$

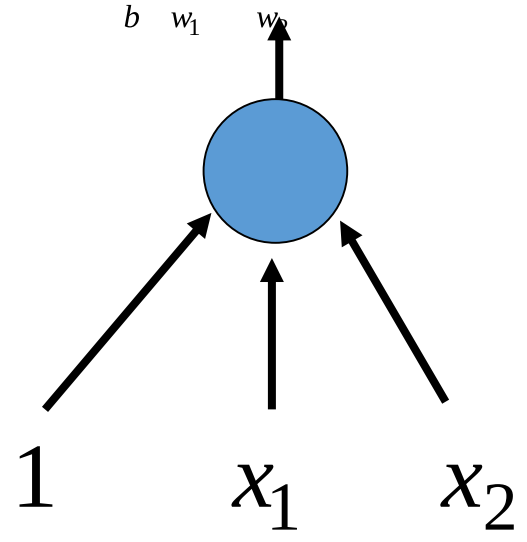
$$= -\sum_n x_{i,n}\,(y_n - \hat{y}_n)$$

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i}$$

Linear Neuron

- A linear neuron is a more flexible model if we include a bias.
- We can avoid having to figure out a separate learning rule for the bias by using a trick:
    - A bias is exactly equivalent to a weight on an extra input line that always has an activity of 1.

$$\hat{y} = b + \sum_i x_i w_i$$



Transfer Functions

- Determines the output from a summation of the weighted inputs of a neuron.
- Maps any real numbers into a domain normally bounded by 0 to 1 or -1 to 1, i.e. squashing functions. Most common functions are sigmoid functions:

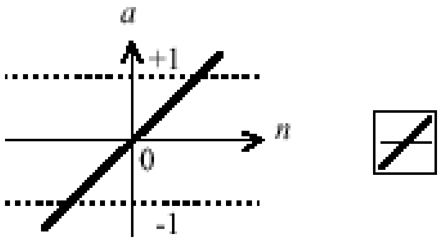
$$O_j = f_j\left(\sum_i w_{ij} x_i\right)$$

logistic: $$f(x) = \frac{1}{1+e^{-x}}$$

hyperbolic tangent: $$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

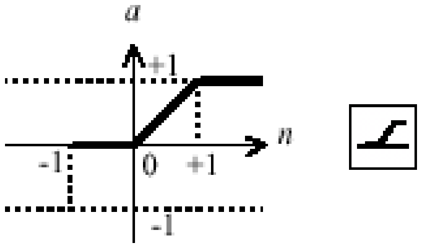
Activation function

- The activation function is generally non-linear.
- Linear functions are limited because the output is simply proportional to the input.



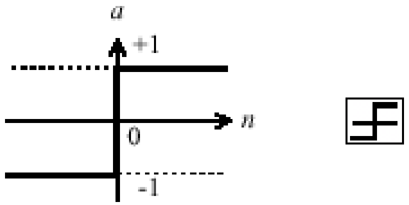
$a = purelin(n)$

Linear Transfer Function



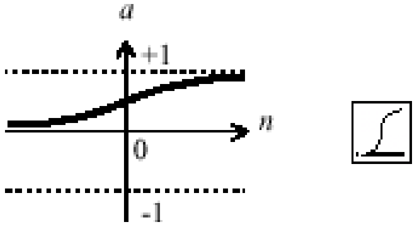
$a = satlin(n)$

Satlin Transfer Function



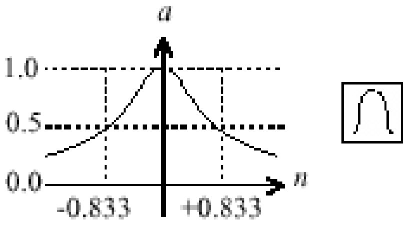
$a = hardlims(n)$

Symmetric Hard Limit Trans. Funct.



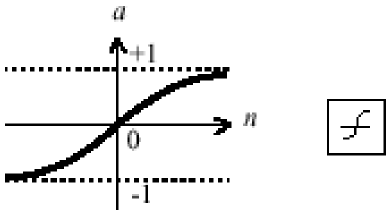
$a = logsig(n)$

Log-Sigmoid Transfer Function



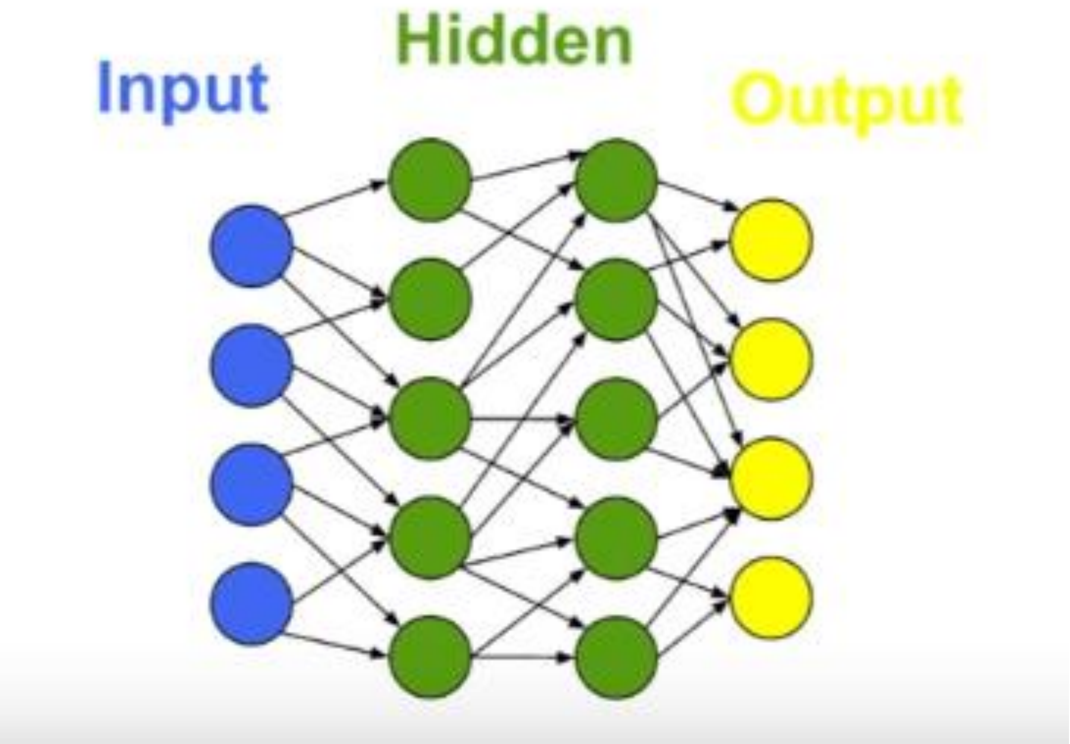
$a = radbas(n)$

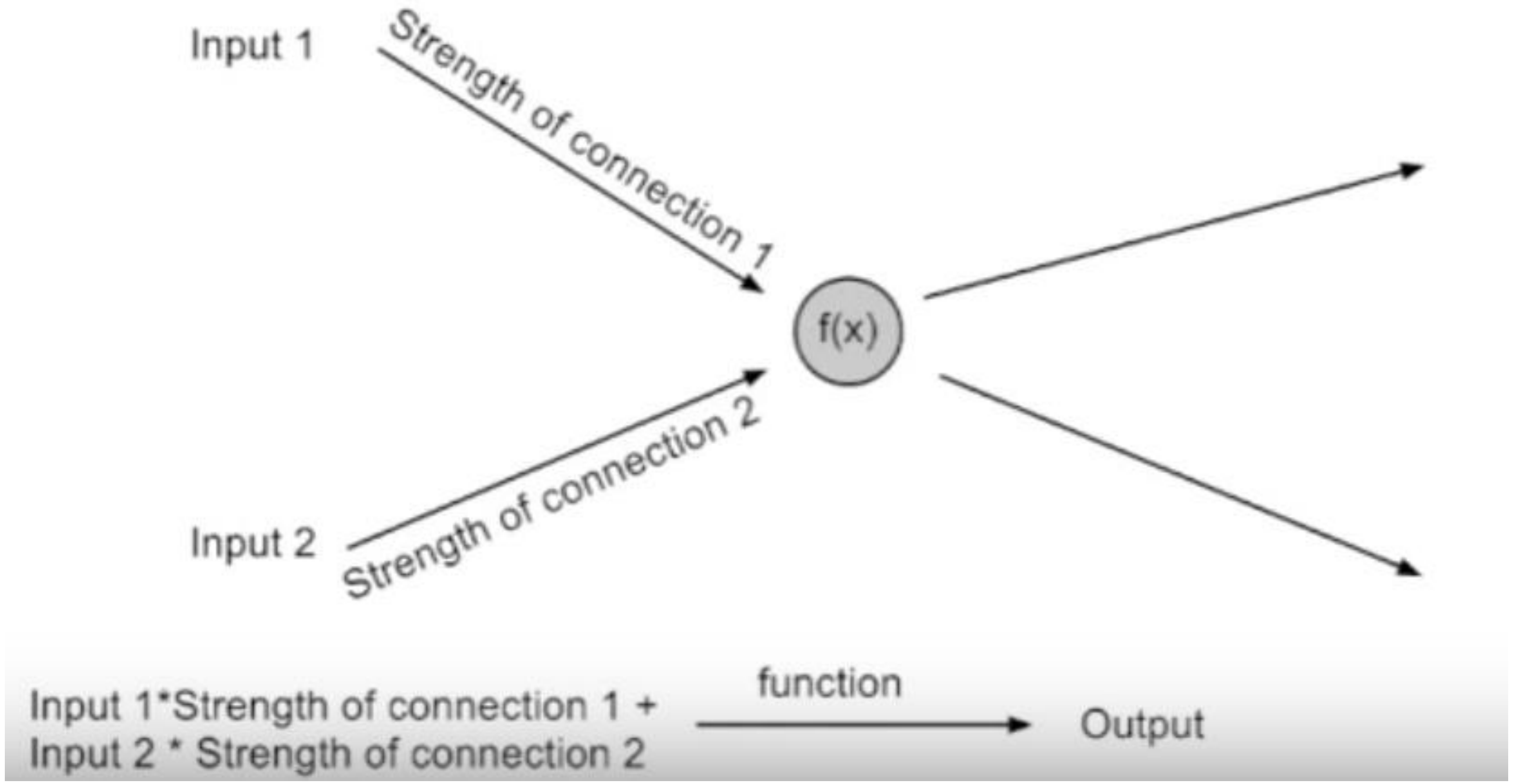
Radial Basis Function



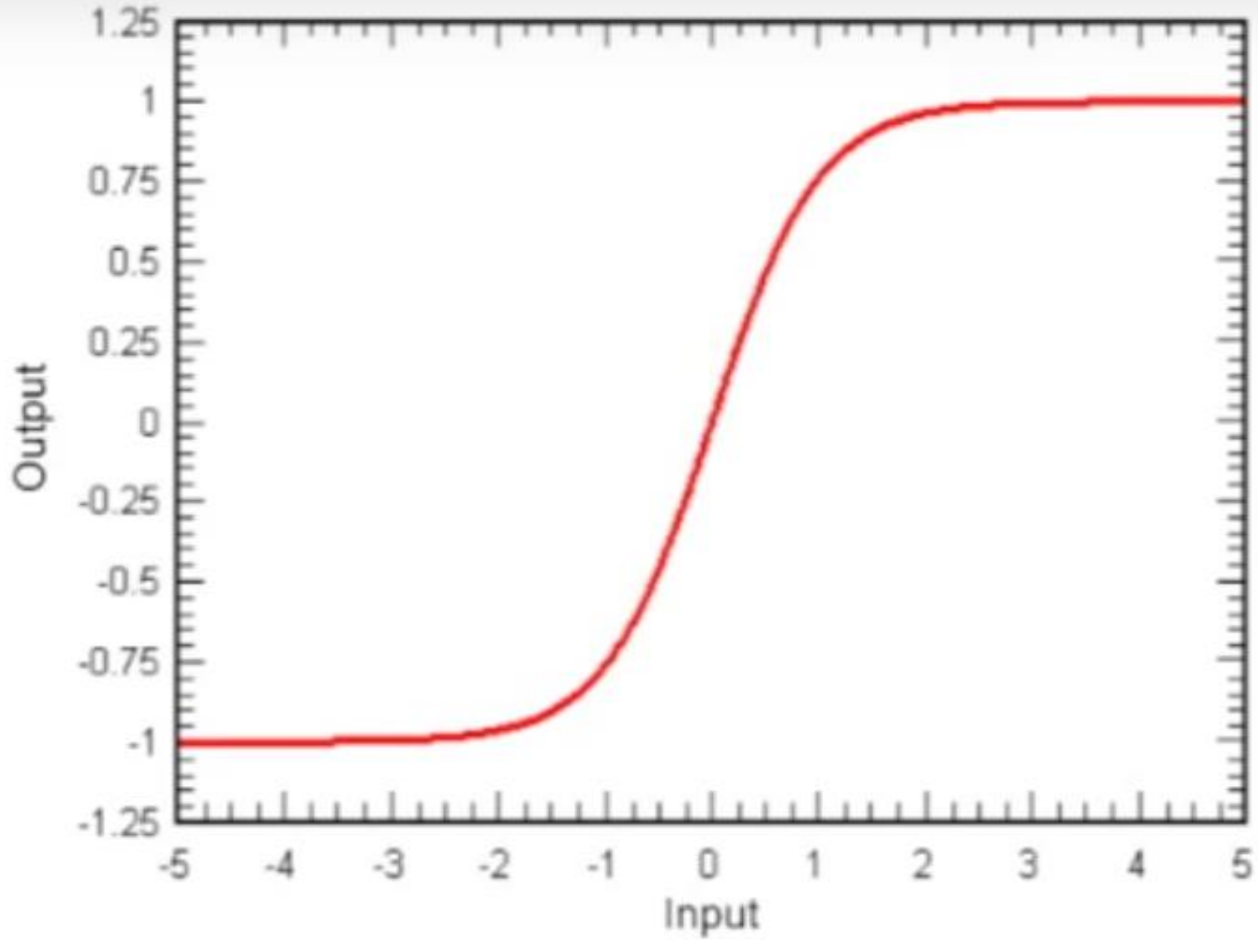
$a = tansig(n)$

Tan-Sigmoid Transfer Function

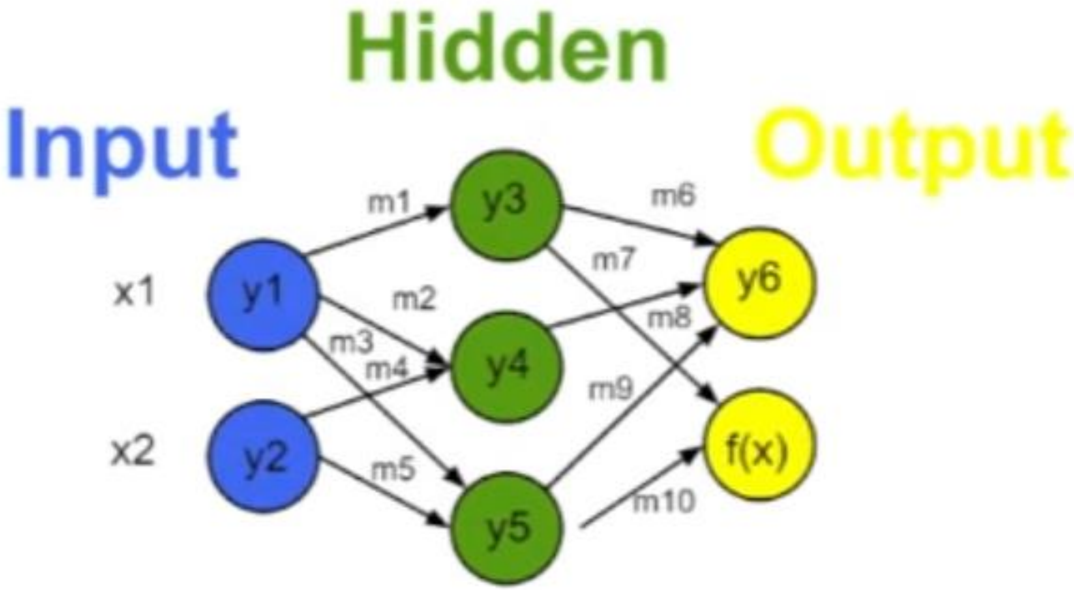

Input
Hidden
Output


Input 1
Strength of connection 1
f(x)
Input 2
Strength of connection 2
Input 1*Strength of connection 1 +
Input 2 * Strength of connection 2
function
Output

Tanh



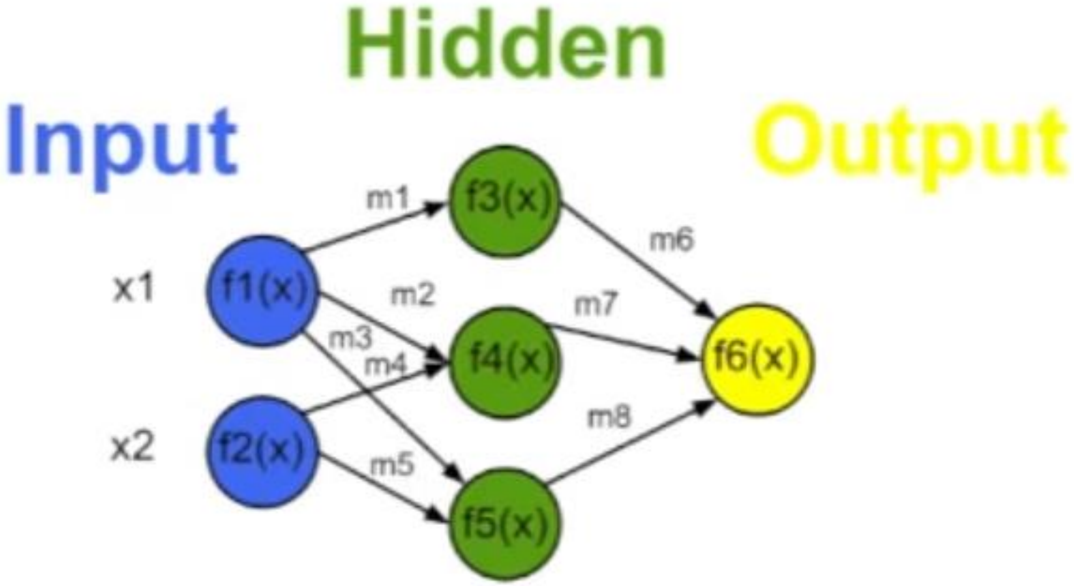
Input 1*Strength of connection 1 +
Input 2 * Strength of connection 2 $\xrightarrow{\text{function}}$ Output



$y1 = f(x1)$ $\quad$ $y3 = f(m1 * y1)$ $\quad$ $y6 = f(m6*y3 + m7*y4 + m8*y5)$

$y2 = f(x2)$ $\quad$ $y4 = f(m2 * y1 + m4 *y2)$ $\quad$ $y7 = f(m7*y3 + m10*y5)$

$y5 = f(m3 * y1 + m5* y2)$



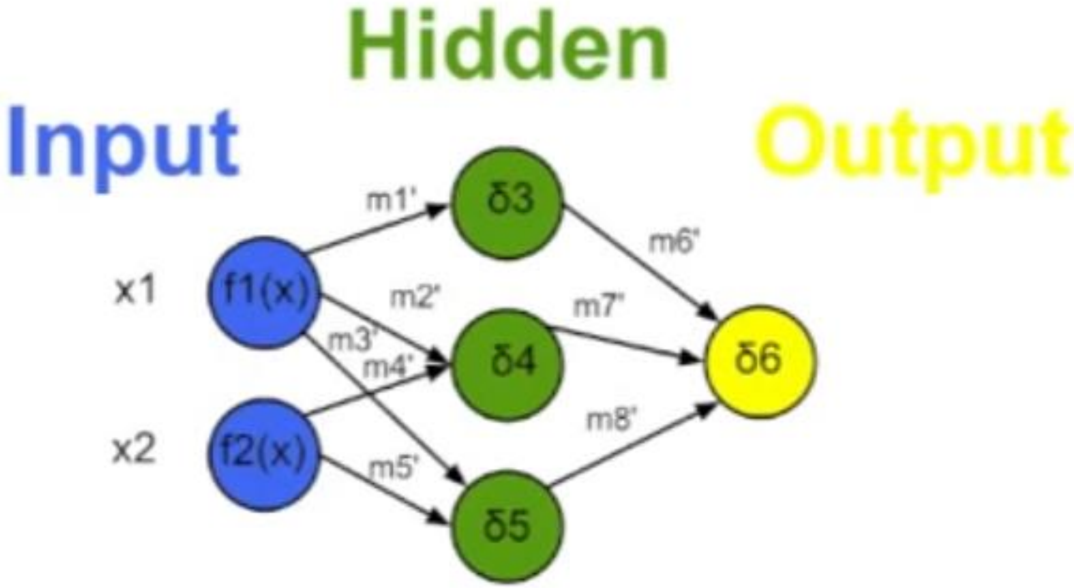
We know these are right, so no error

$\delta 3 = \delta * m6$

$\delta 4 = \delta * m7$

$\delta 5 = \delta * m8$

δ = actual output - desired output



Hidden
Input
Output
x1
x2
f1(x)
f2(x)
m1'
m2'
m3'
m4'
m5'
m6'
m7'
m8'
δ3
δ4
δ5
δ6