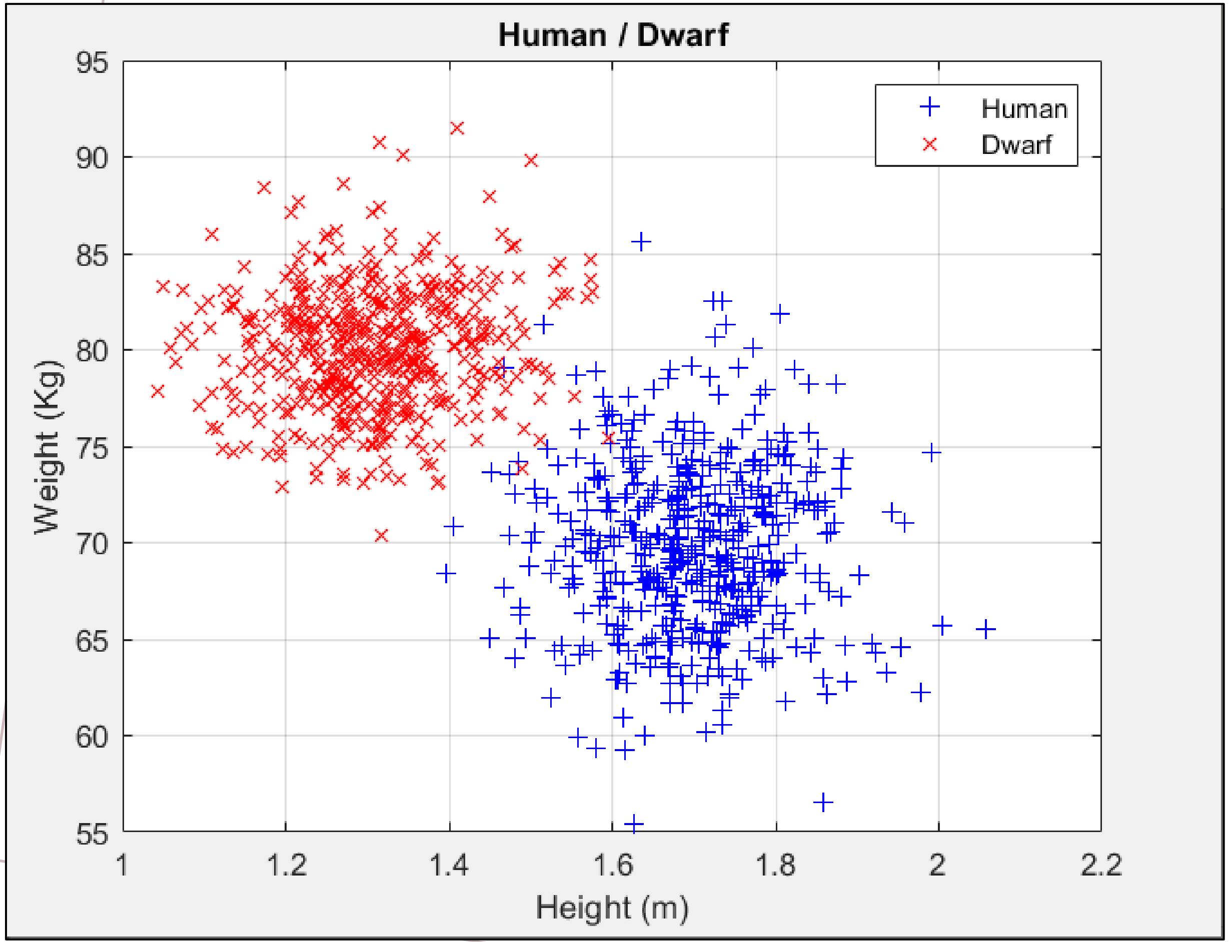
312-3302 ARTIFICIAL INTELLIGENCE

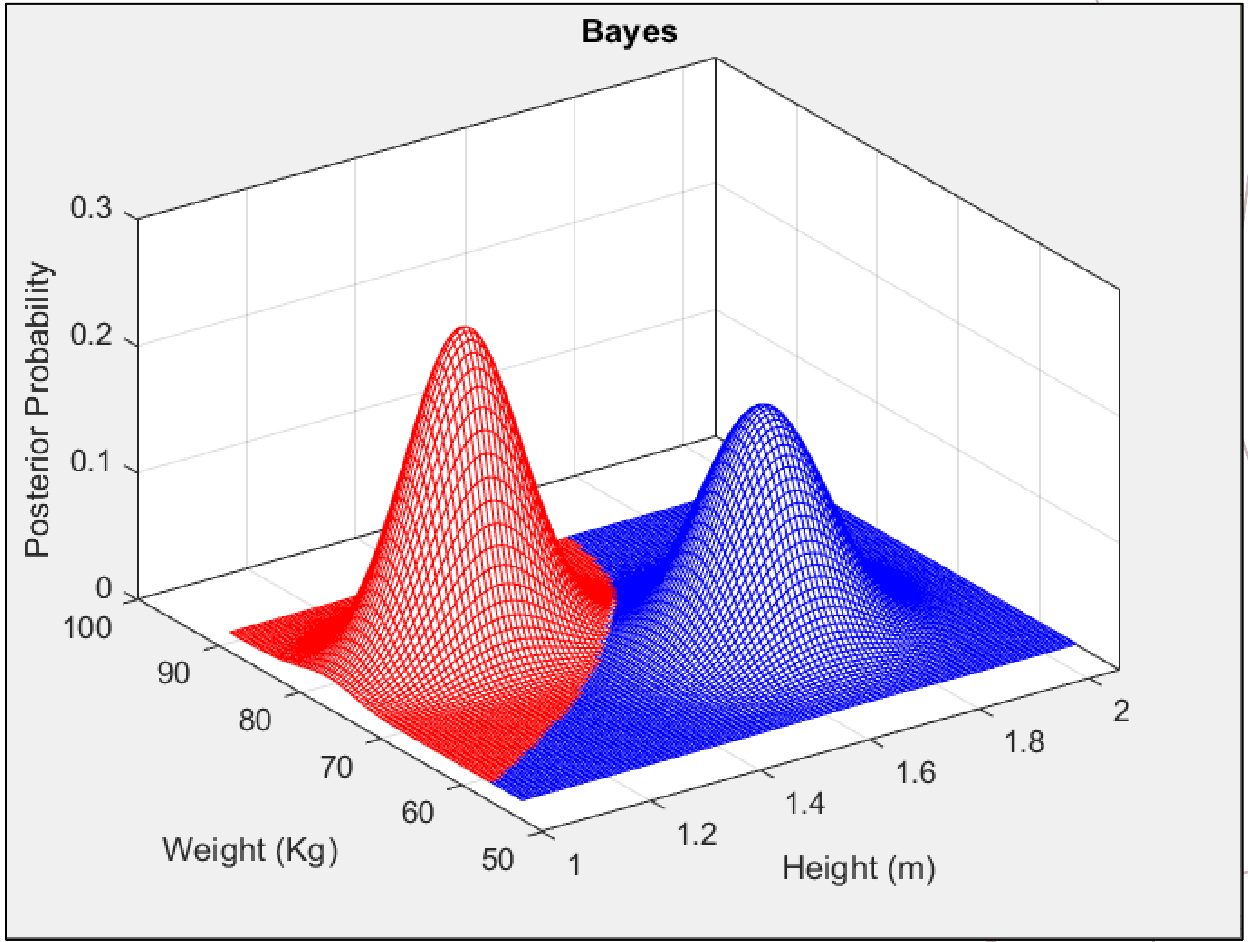
Lecture 4

Perceptron

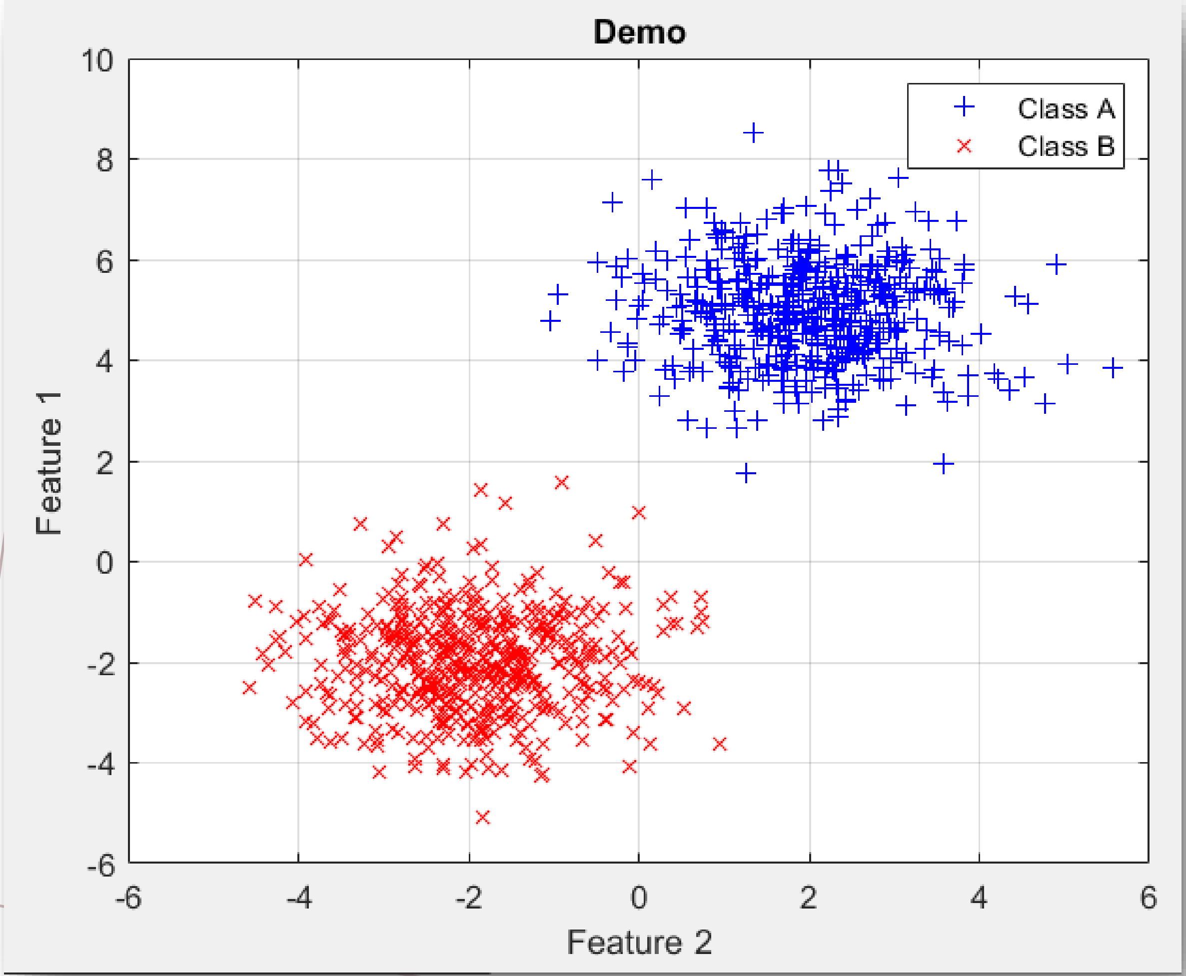
Decision Hyperplane

Hyperplane is a subspace whose dimension is <u>one less than</u> that of its ambient space.

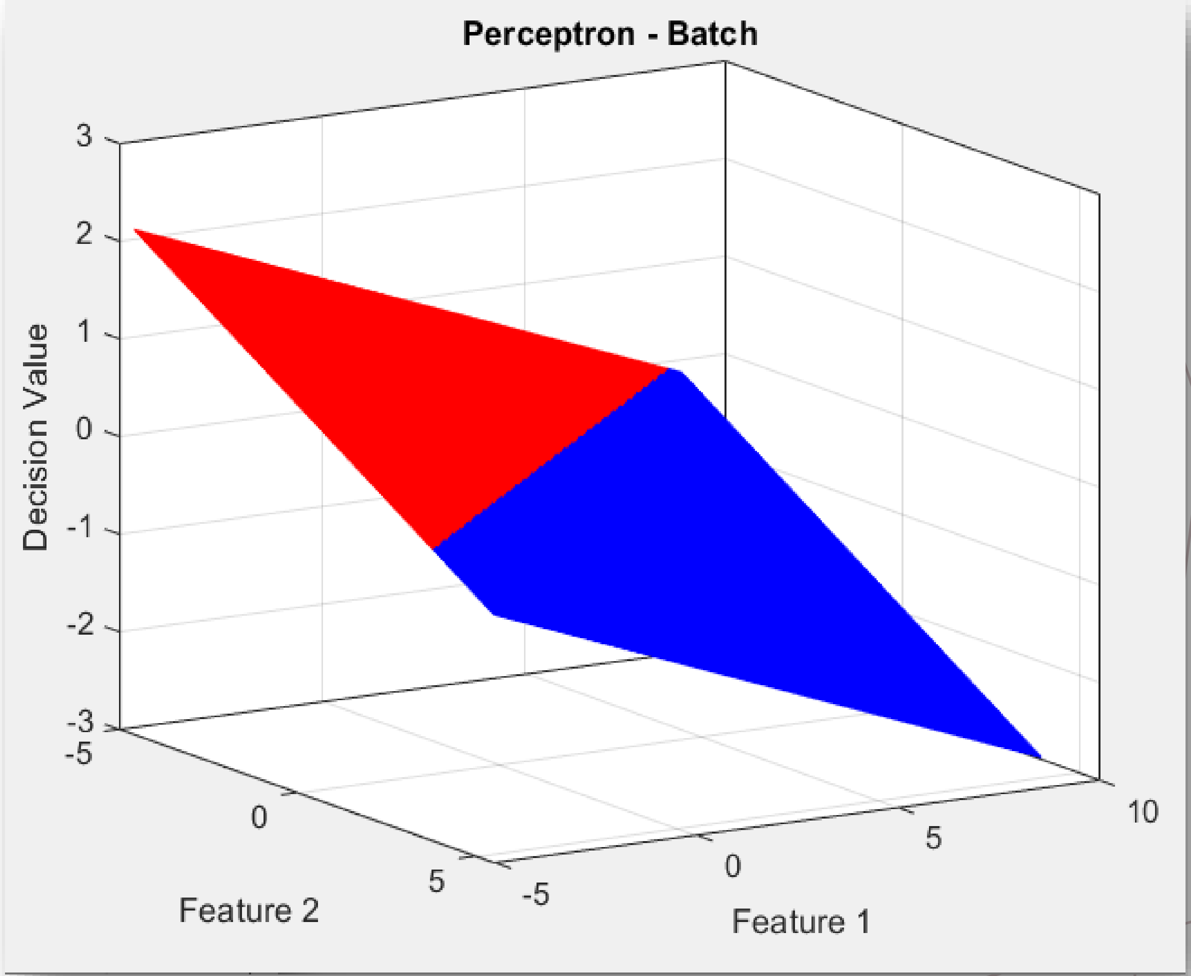




Perceptron

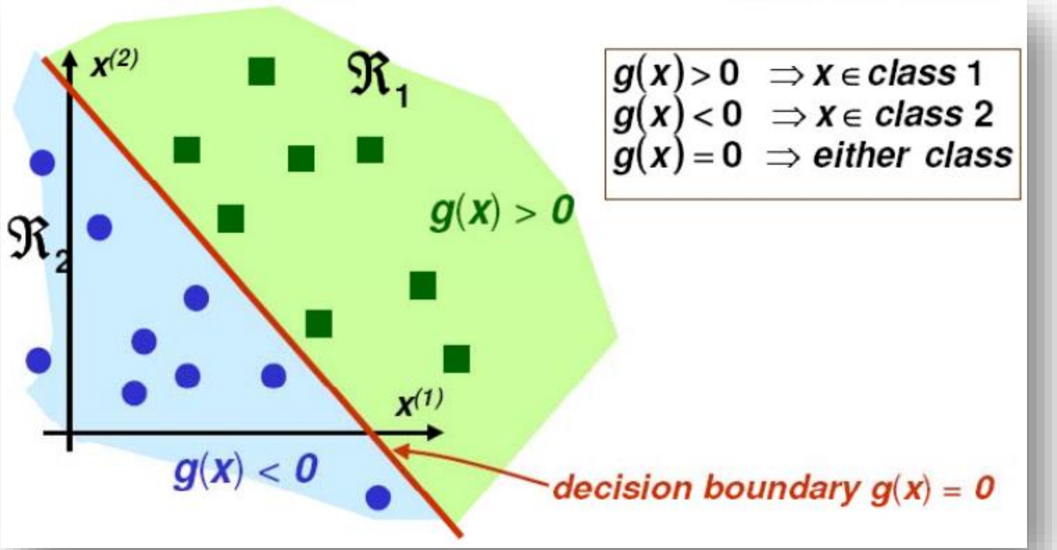


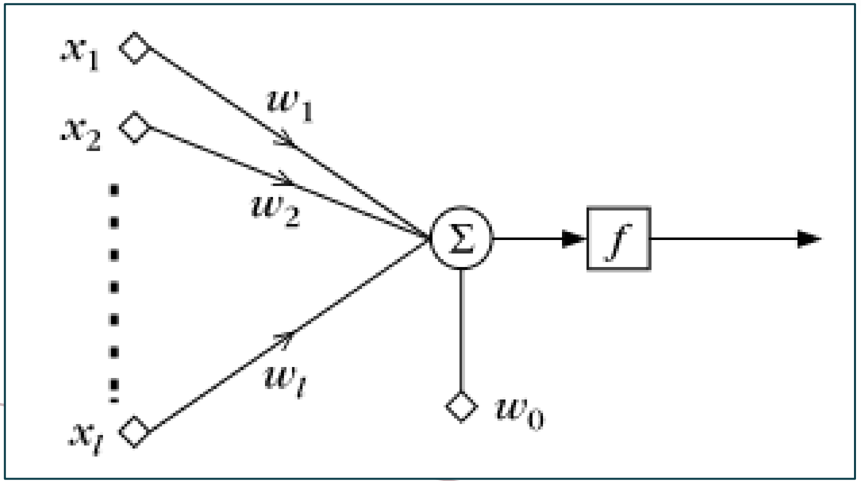
Demo
Class A
Class B
Feature 1
Feature 2
10
8
6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6



Perceptron - Batch
Decision Value
3
2
1
0
-1
-2
-3
-5
0
5
Feature 2
-5
0
5
10
Feature 1

Perceptron





$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$$

$$\boldsymbol{w}^{*T} \boldsymbol{x} > 0 \quad \forall \boldsymbol{x} \in \omega_1$$

$$\boldsymbol{w}^{*T} \boldsymbol{x} < 0 \quad \forall \boldsymbol{x} \in \omega_2$$

Cost function

$$J(\boldsymbol{w}) = \sum_{\boldsymbol{x} \in Y} (\delta_x \boldsymbol{w}^T \boldsymbol{x})$$

$\delta_x = -1$ if $x \in \omega_1$

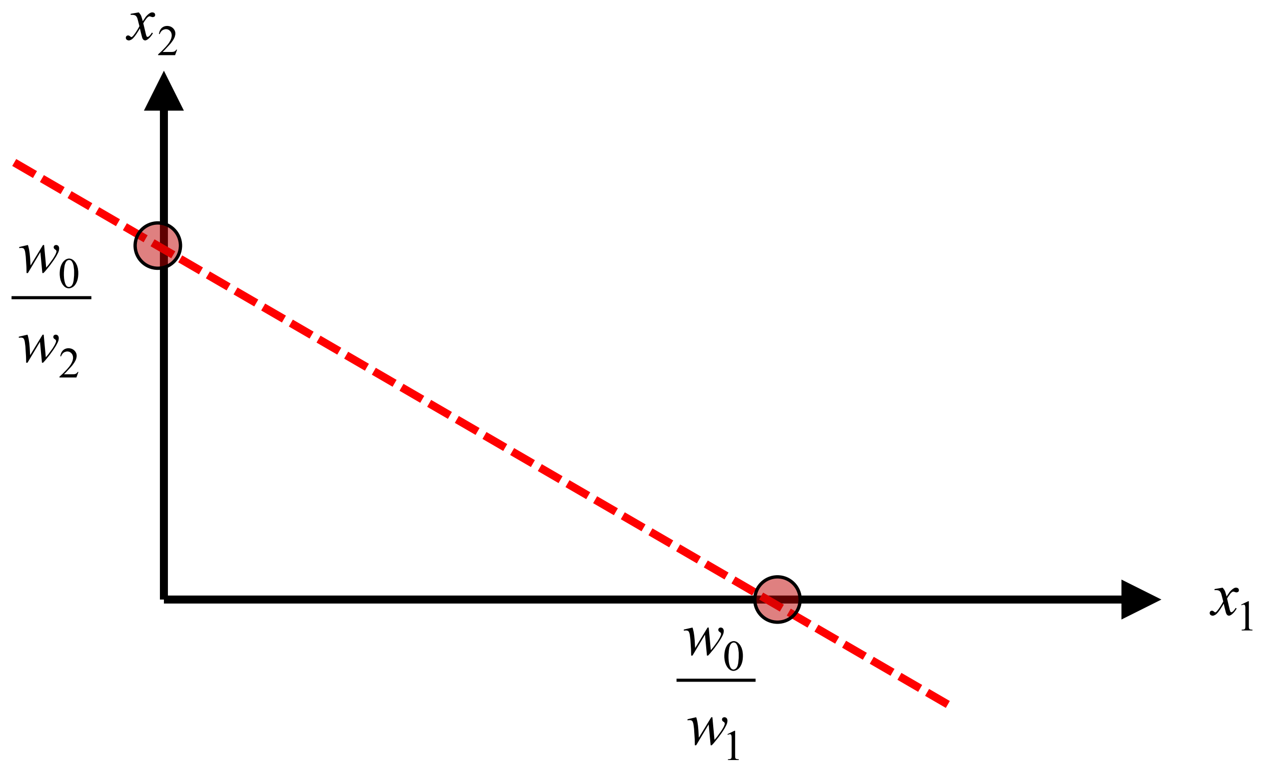
$\delta_x = +1$ if $x \in \omega_2$

Perceptron

$$g(x) = w^T x + w_0 = 0$$

Ex.

$$w_1 x_1 + w_2 x_2 - w_0 = 0$$

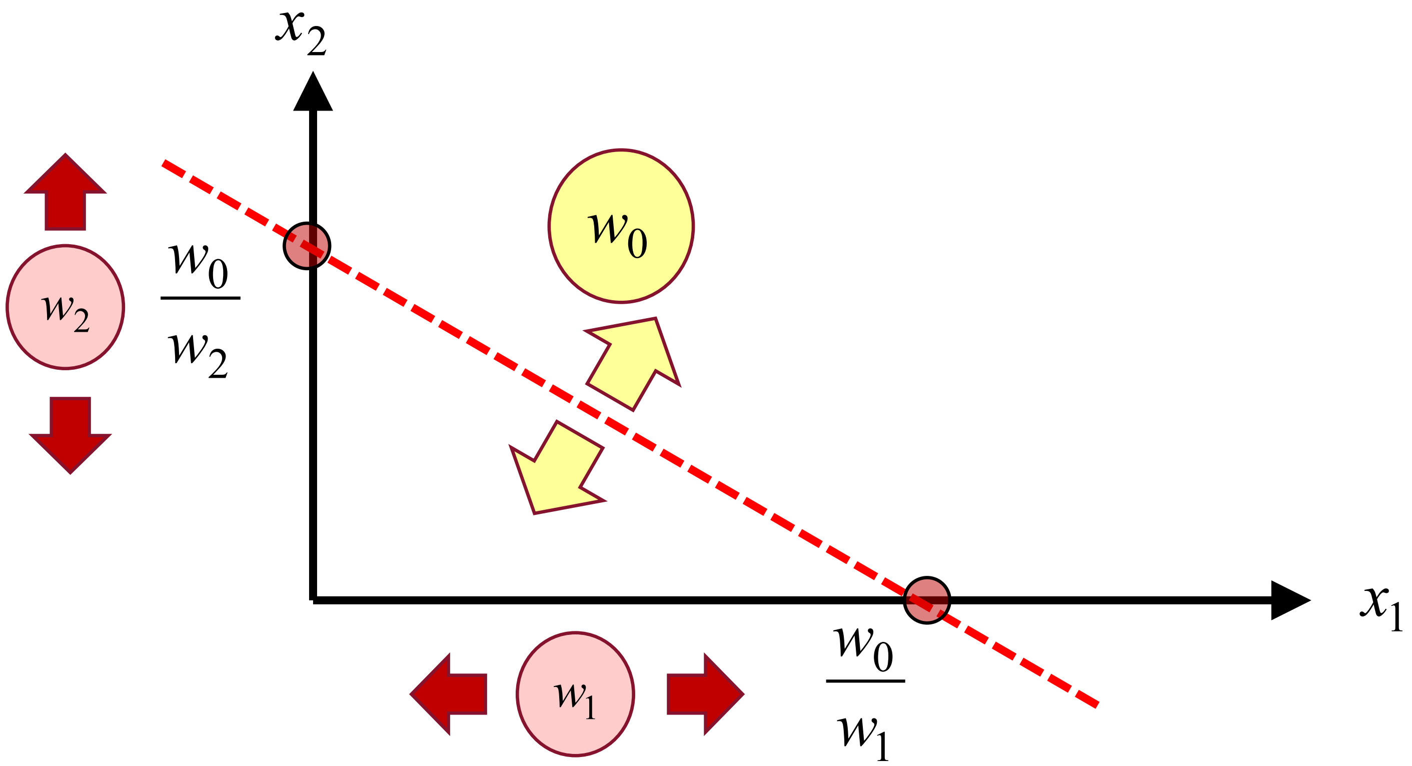


Perceptron

$$g(x) = w^T x + w_0 = 0$$

Ex.

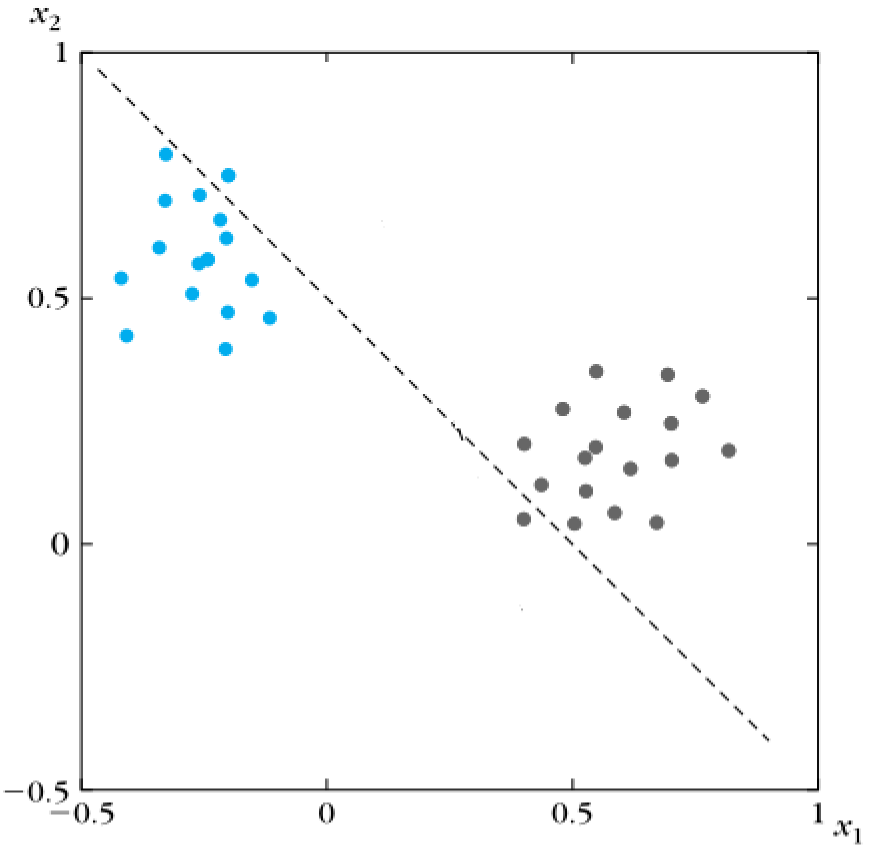
$$w_1 x_1 + w_2 x_2 - w_0 = 0$$



Perceptron

<u>Example</u> กำหนดเส้นแบ่งเริ่มต้น (เส้นประ)

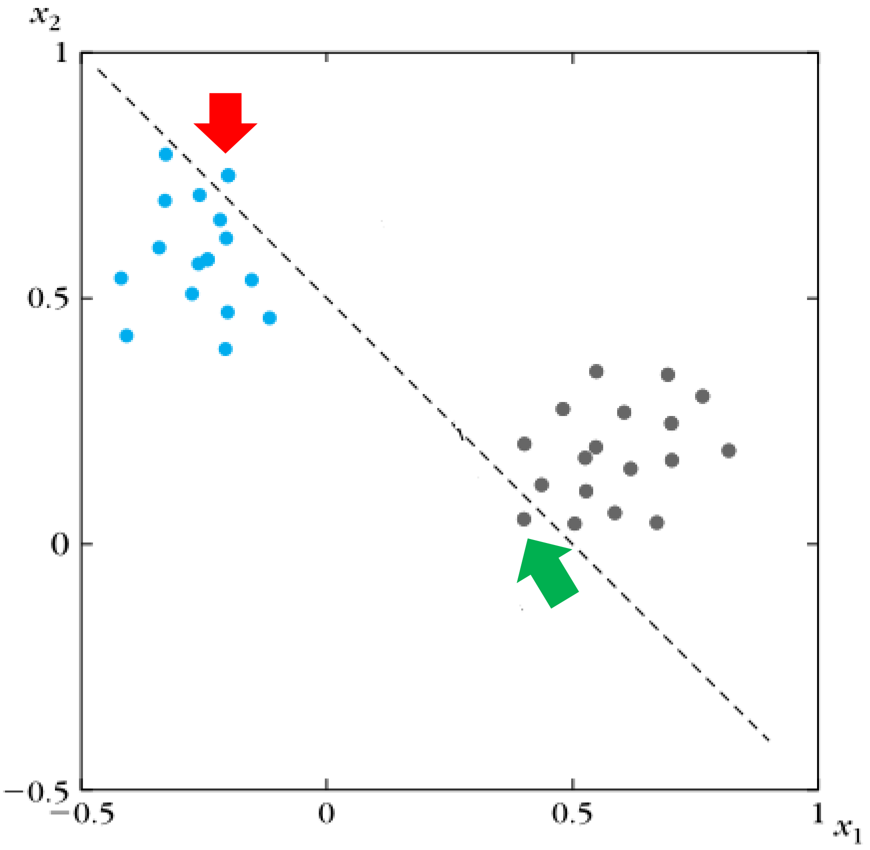
$$g(x) = x_1 + x_2 - 0.5 = 0$$



$$w(t) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix}$$

Perceptron

<u>Example</u> กำหนดเส้นแบ่งเริ่มต้น (เส้นประ)



$$g(x) = x_1 + x_2 - 0.5 = 0$$

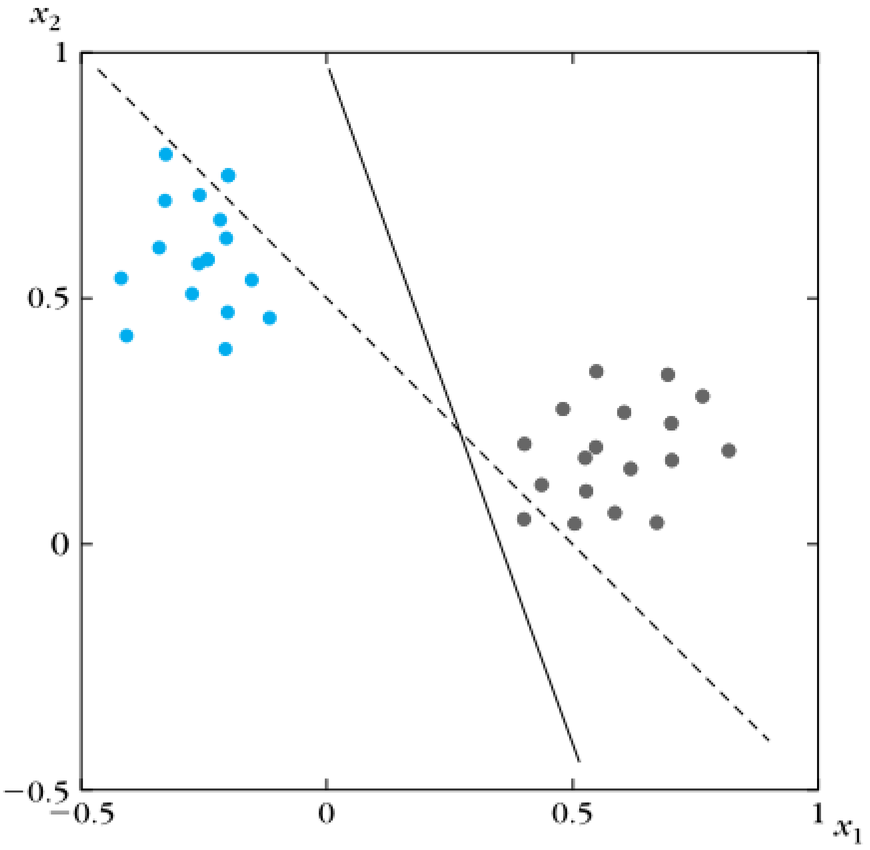
$$w(t) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix}$$

0.7 คือ Parameter สำหรับการกำหนดระยะการลู่เข้า

$$w(t+1) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix} - 0.7(-1)\begin{bmatrix} 0.4 \\ 0.05 \\ 1 \end{bmatrix} - 0.7(+1)\begin{bmatrix} -0.2 \\ 0.75 \\ 1 \end{bmatrix}$$

Perceptron

<u>Example</u> กำหนดเส้นแบ่งเริ่มต้น (เส้นประ)



$$g(x) = x_1 + x_2 - 0.5 = 0$$

$$w(t) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix}$$

$$w(t+1) = \begin{bmatrix} 1.42 \\ 0.51 \\ -0.5 \end{bmatrix}$$

Perceptron

แล้วเรารู้ได้อย่างไรว่าจะ (+) หรือ (–) weight

$$w(t+1)=\begin{bmatrix}1\\1\\-0.5\end{bmatrix}-0.7(-1)\begin{bmatrix}0.4\\0.05\\1\end{bmatrix}-0.7(+1)\begin{bmatrix}-0.2\\0.75\\1\end{bmatrix}$$

ทำไมเครื่องหมายตรงนี้ต้องเป็นเครื่องหมายลบ

$$w(t+1)=\begin{bmatrix}1\\1\\-0.5\end{bmatrix}\boxed{-0.7(-1)\begin{bmatrix}0.4\\0.05\\1\end{bmatrix}-0.7(+1)\begin{bmatrix}-0.2\\0.75\\1\end{bmatrix}}$$

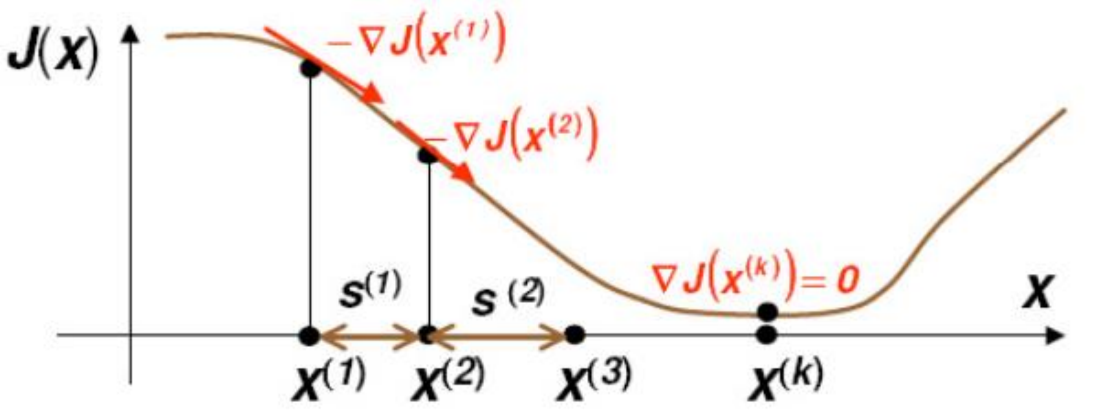
(1) The Cost Function $J(\underline{w})=\sum_{\underline{x}\in Y}(\delta_x \underline{w}^T \underline{x})$

where Y is the subset of the vectors wrongly classified by w.

When Y =(empty set) a solution is achieved and $J(\underline{w})=0$

$$\delta_x=-1 \text{ if } \underline{x}\in Y \text{ and } \underline{x}\in\omega_1$$
$$\delta_x=+1 \text{ if } \underline{x}\in Y \text{ and } \underline{x}\in\omega_2$$

Perceptron



(2) The algorithm to minimize this cost function

The philosophy of the gradient descent is adopted.

(3) This iterative process will search for minimum cost function by

$$\boxed{\underline{w}(t+1) = \underline{w}(t) - \rho_t \sum_{\underline{x} \in Y} \delta_x \underline{x}}$$

This is called the Perceptron Algorithm

Gradient Descent for minimizing any function $J(x)$

set $k = 1$ and $x^{(1)}$ to some initial guess for the weight vector

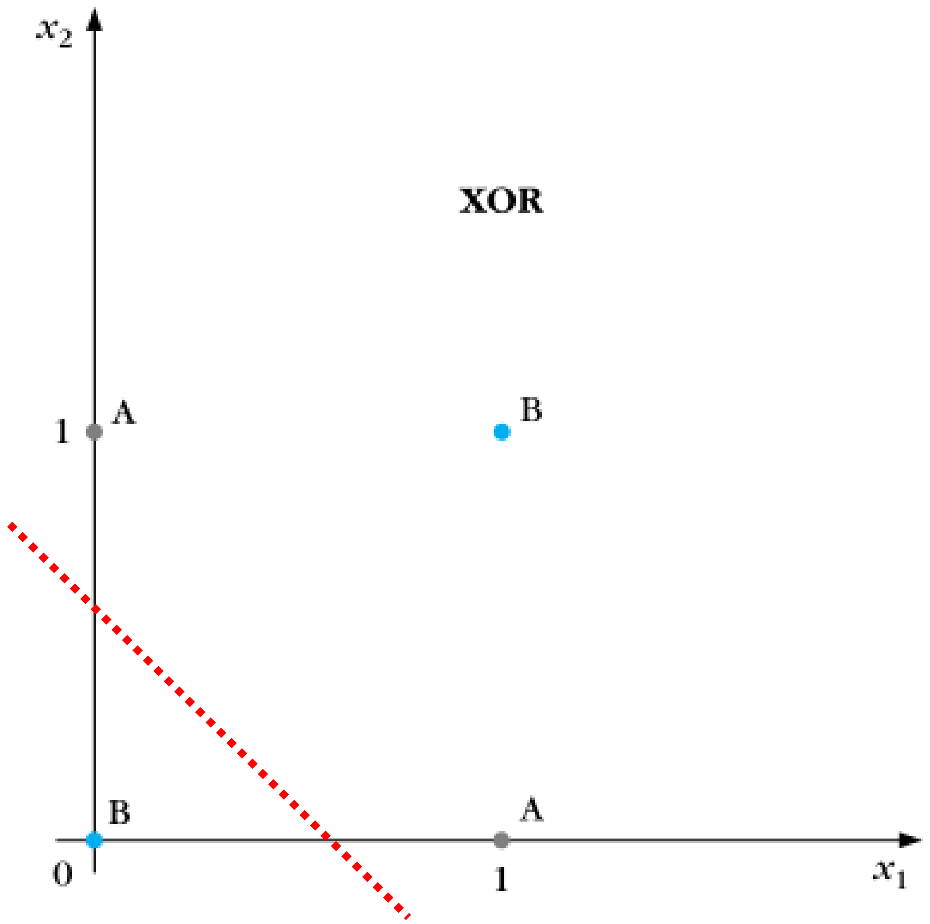
while $\eta^{(k)} \left| \nabla J\left(x^{(k)}\right) \right| > \varepsilon$

choose learning rate $\eta^{(k)}$

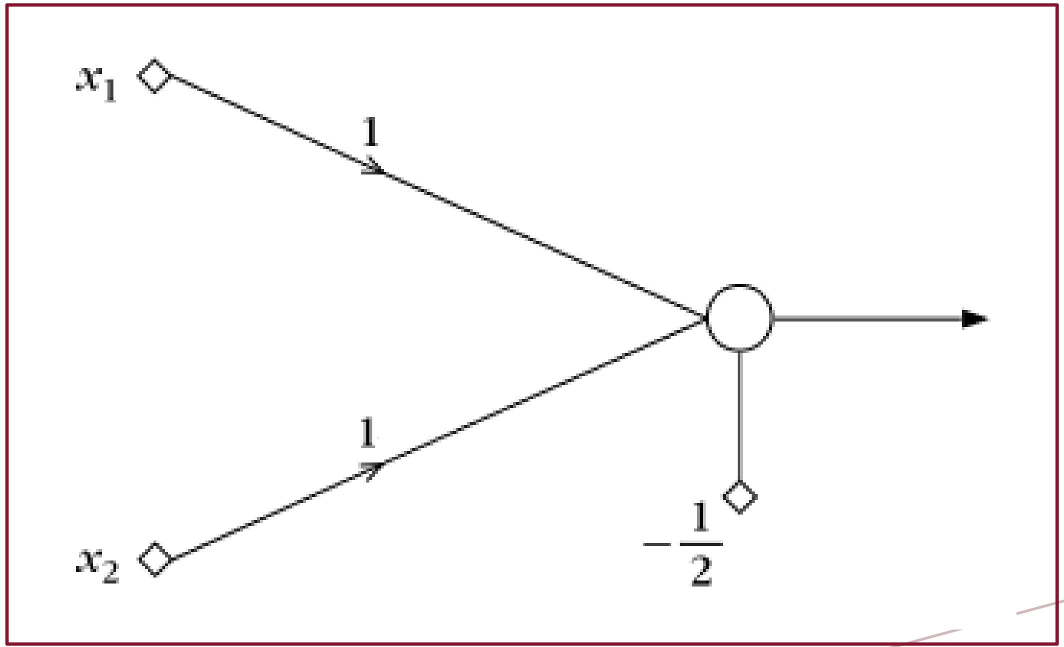
$x^{(k+1)} = x^{(k)} - \eta^{(k)} \nabla J(x)$ (update rule)

$k = k + 1$

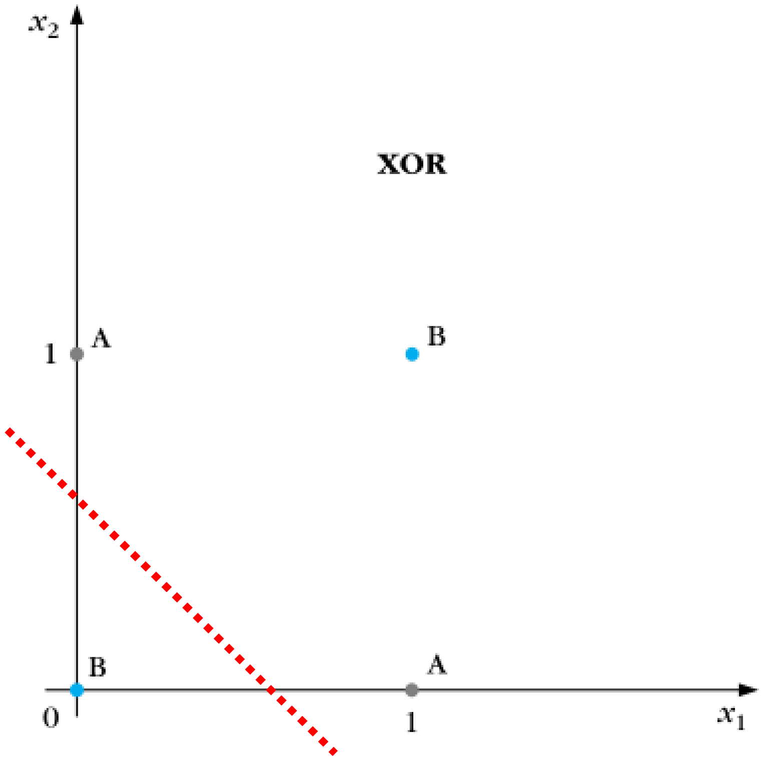
Non-linear : XOR Problem



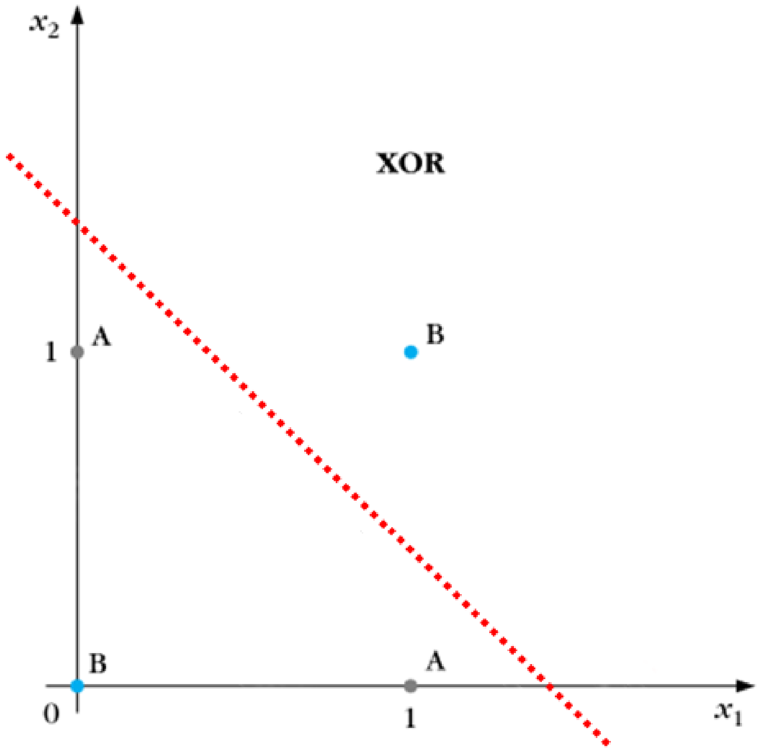
x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B



Non-linear : XOR Problem

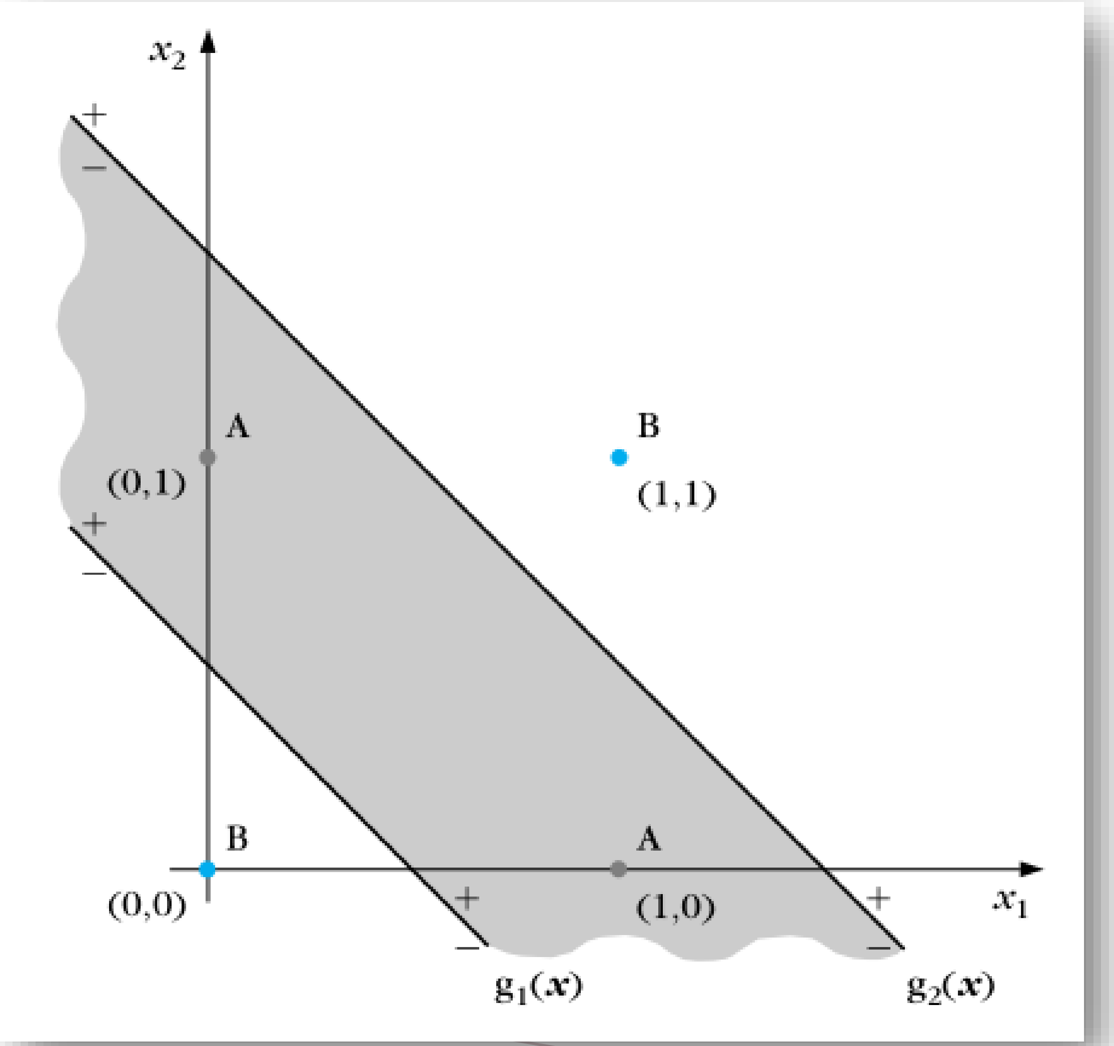


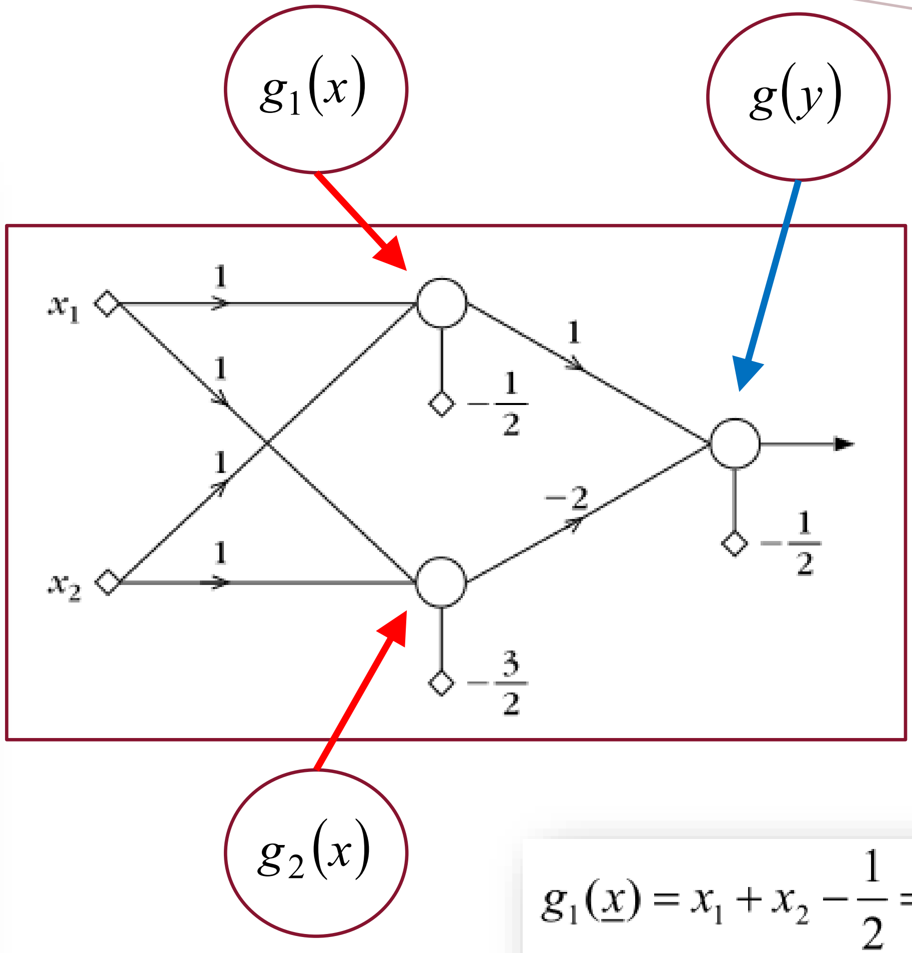
$$g_1(\underline{x}) = x_1 + x_2 - \frac{1}{2} = 0$$



$$g_2(\underline{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

Non-linear : XOR Problem



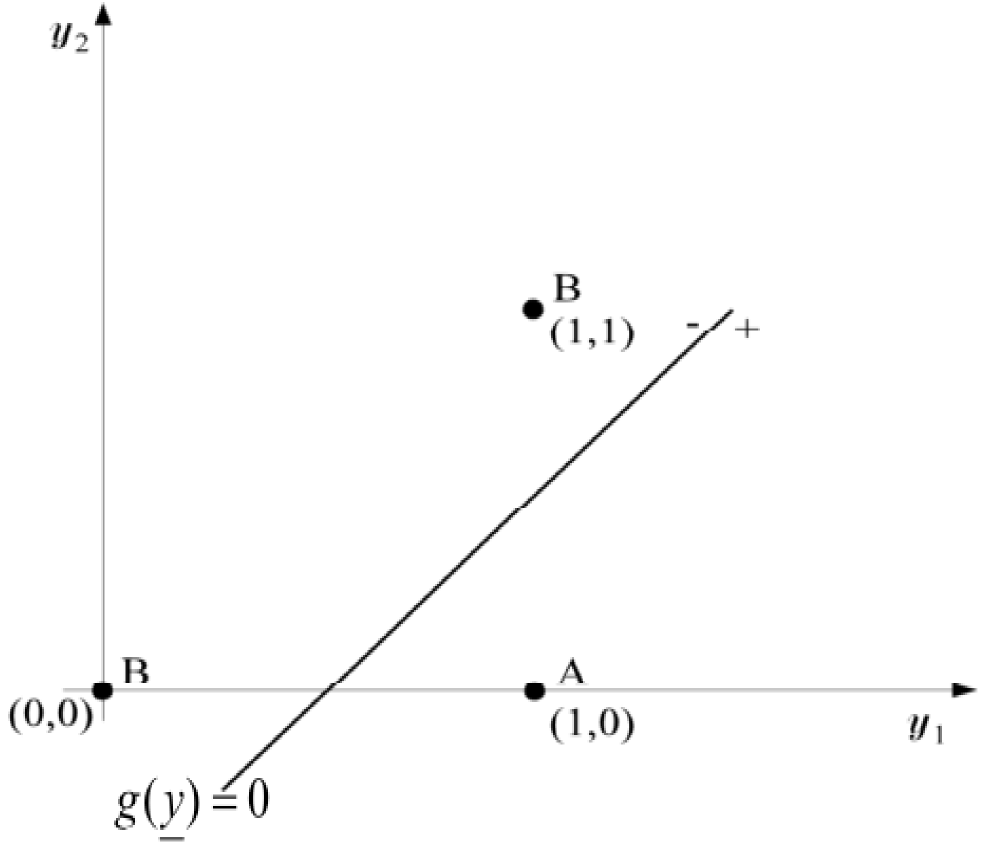


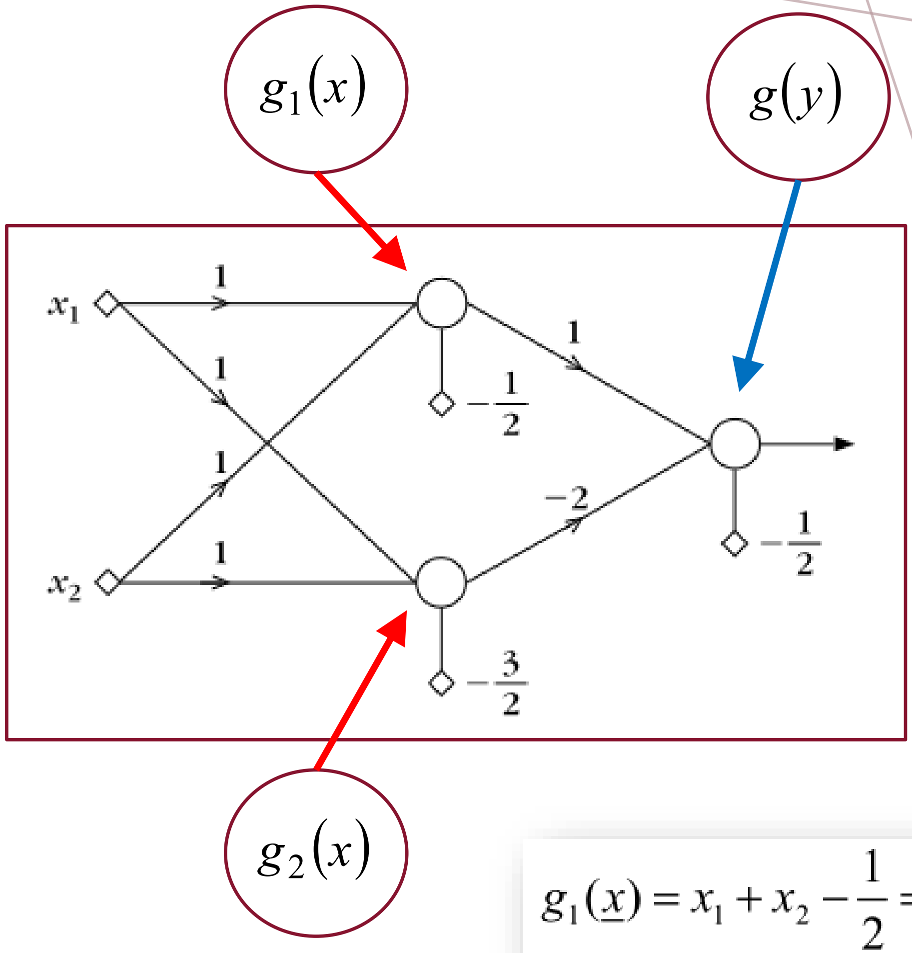
$$g_1(\underline{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

$$g_2(\underline{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

$$g(\underline{y}) = y_1 - 2y_2 - \frac{1}{2} = 0$$

Non-linear : XOR Problem





$$g_1(\underline{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

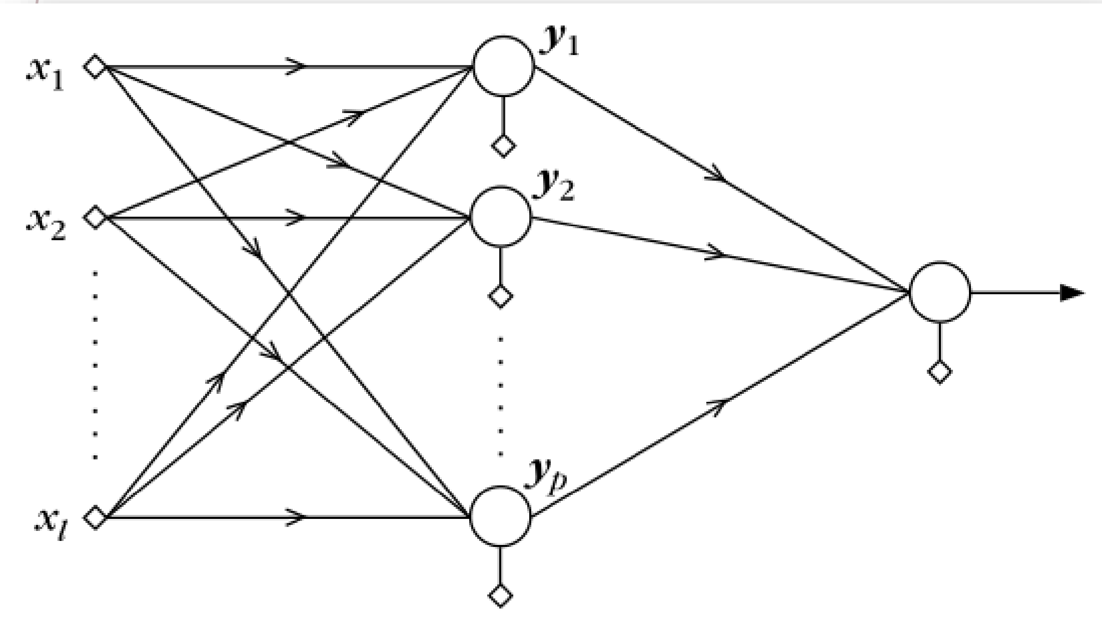
$$g_2(\underline{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

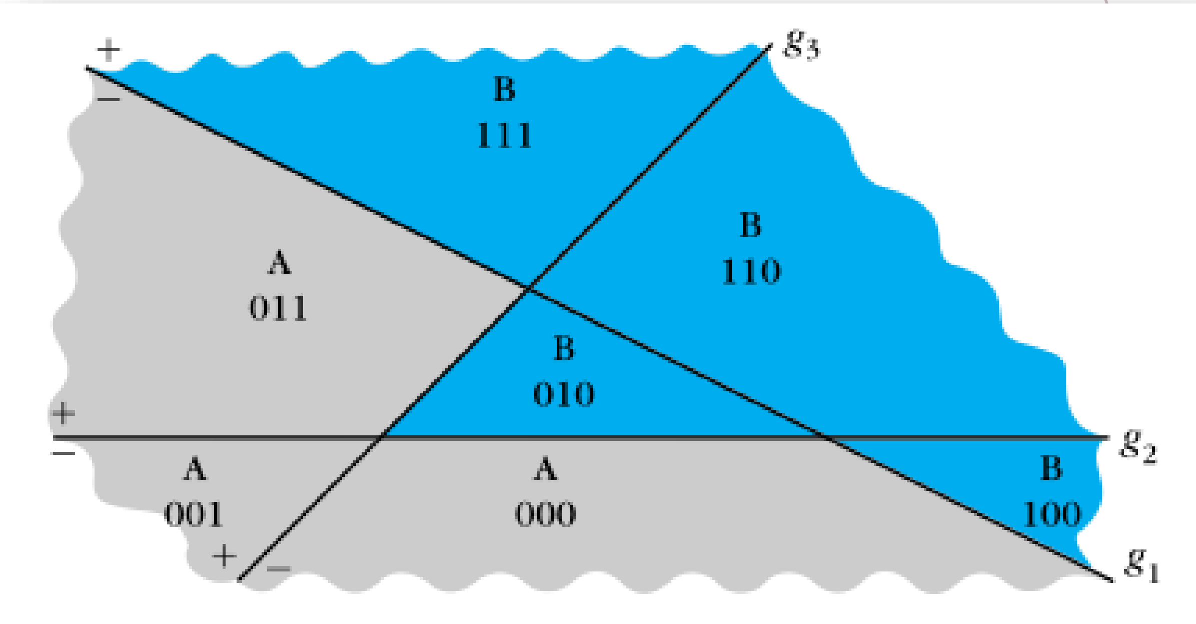
$$g(\underline{y}) = y_1 - 2y_2 - \frac{1}{2} = 0$$

ค่าของ Feature X_1 และ X_2 ถูก Map ให้มาอยู่ใน Plane ใหม่

<u>note</u> (1,0) และ (0,1) จะให้ค่า Y_1 และ Y_2 ค่าเดียวกัน

Neural Network

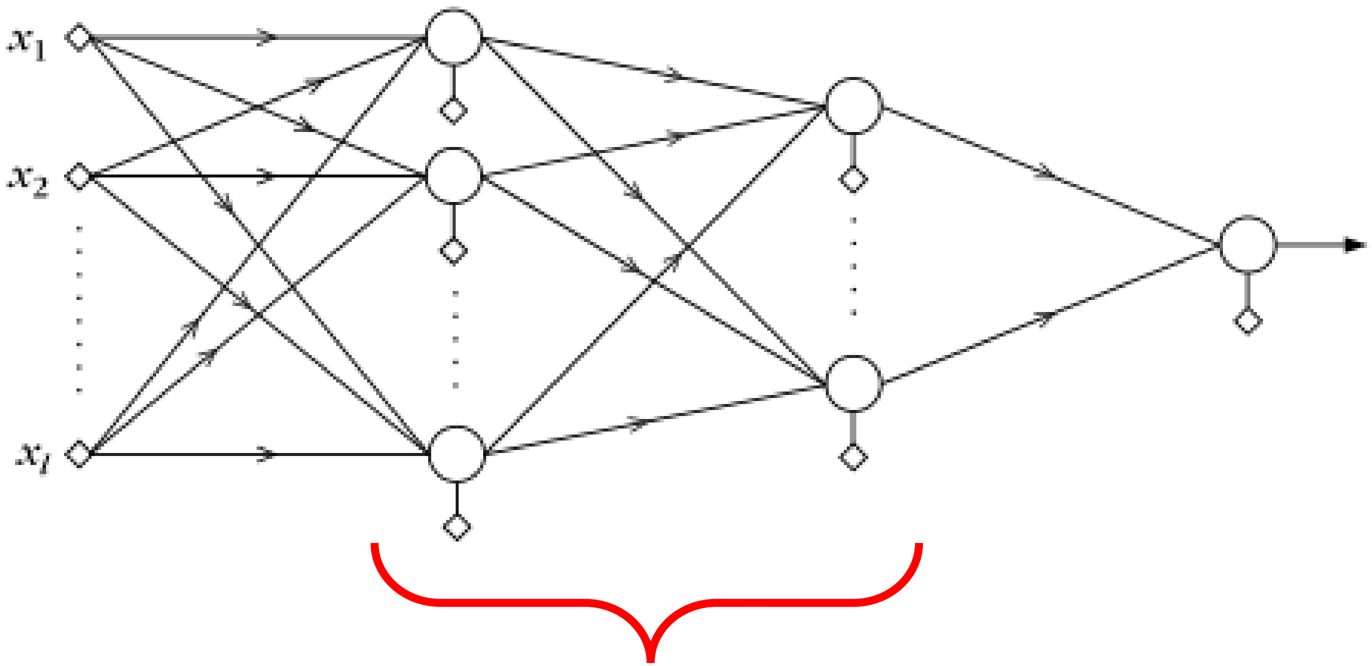




เพิ่มจำนวน Node ในแต่ละ Layer

Neural Network

เพิ่มจำนวน Layer ของ Network



Two Hidden Layer

Neural Network

Polynomial (Order 2)

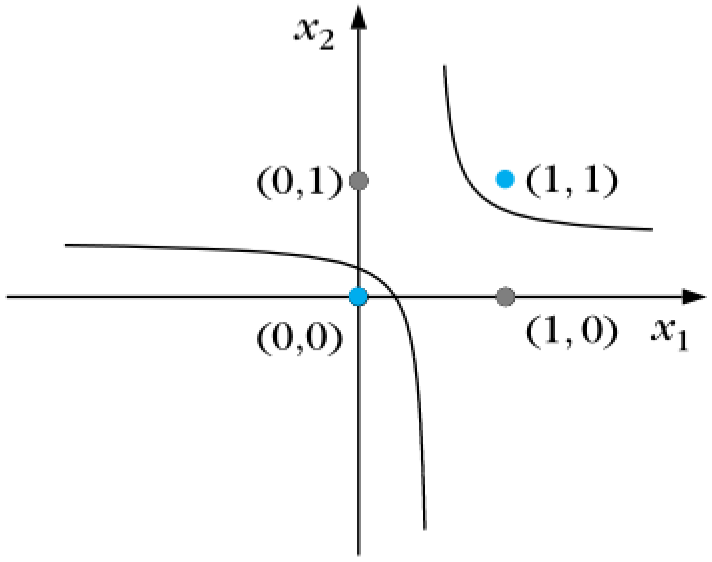
$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{l} w_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^{l} w_{im} x_i x_m + \sum_{i=1}^{l} w_{ii} x_i^2$$

Radial Basis

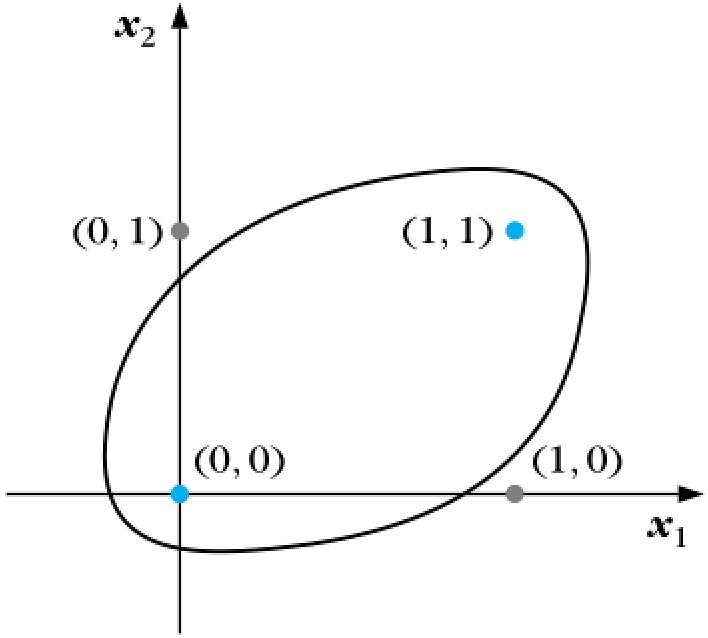
$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{k} w_i \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{c}_i)^T (\boldsymbol{x} - \boldsymbol{c}_i)}{2\sigma_i^2} \right)$$

เราสามารถเปลี่ยน Linear function ให้เป็น Non-linear ด้วยการใส่ Kernel function ต่าง ๆ ได้

Neural Network



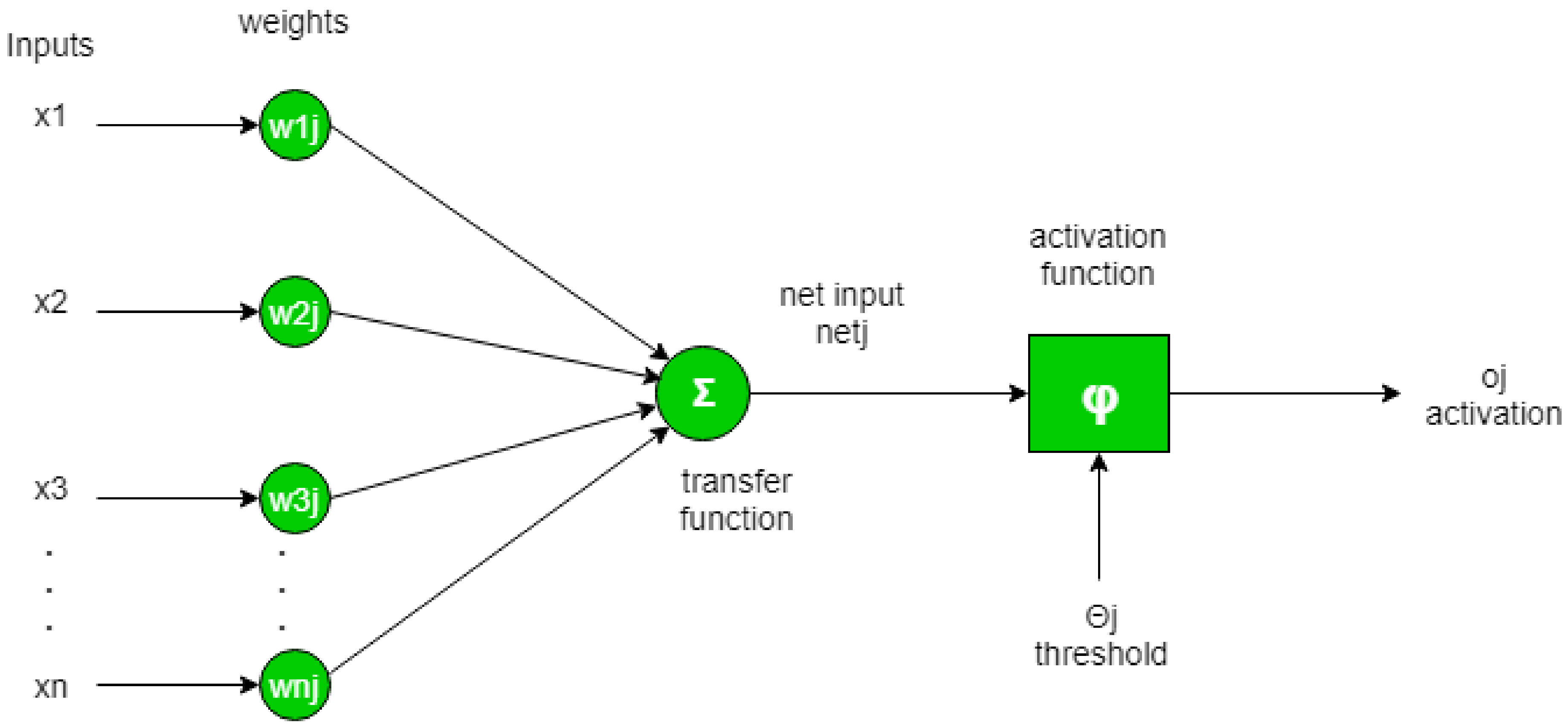
Polynomial
(Order 2)



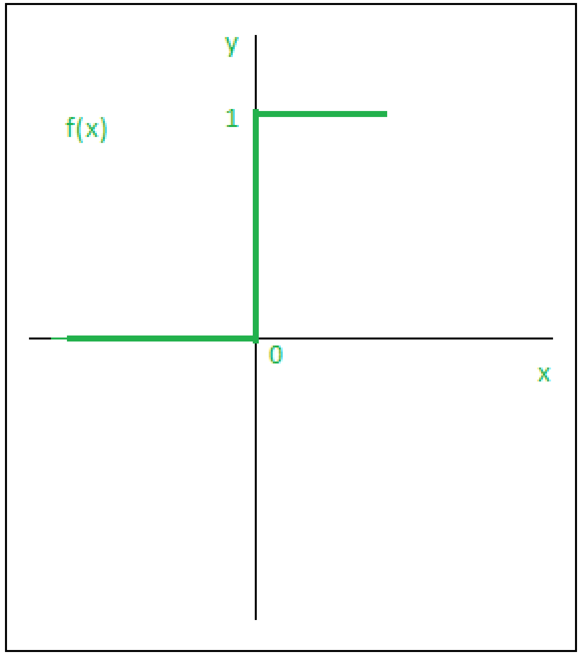
Radial Basis

โจทย์ XOR ที่สามารถแยกข้อมูลได้หลายวิธีการ

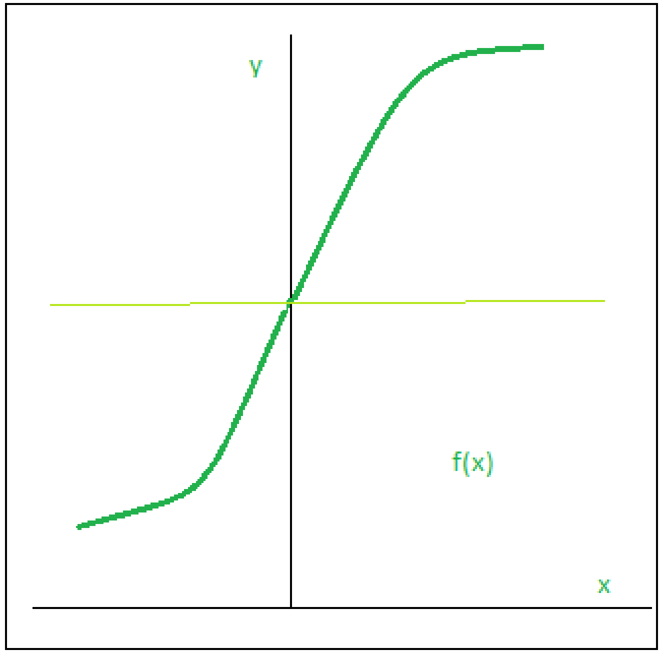
Neural Network : Activation Function



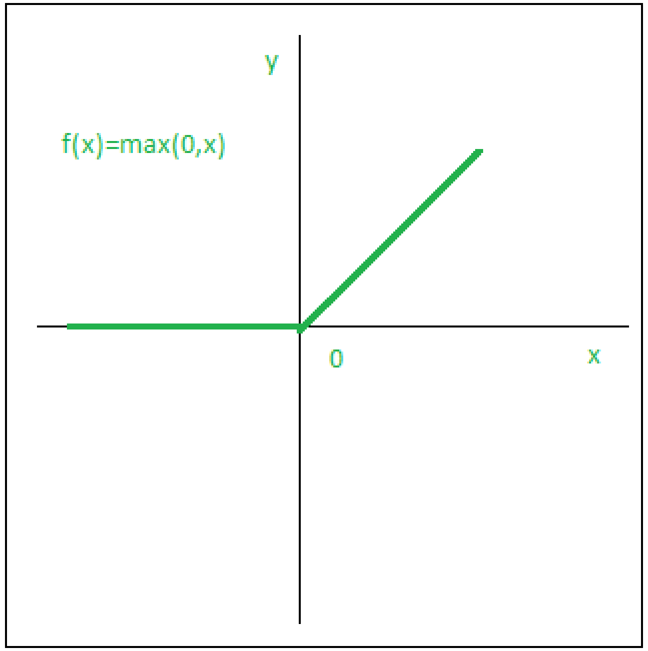
Neural Network : Activation Function



Step Function



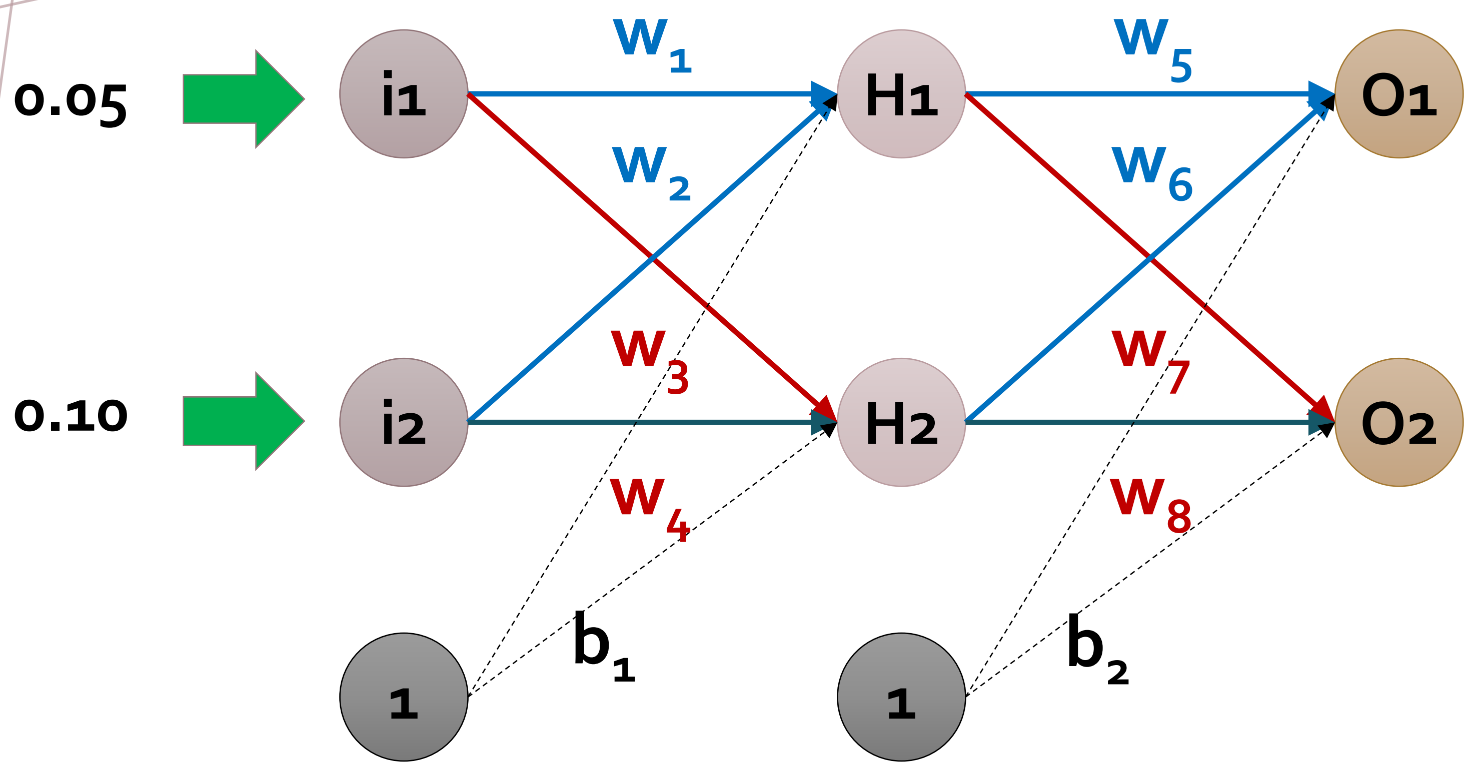
Sigmoid Function



Rectified linear unit (ReLU)

Neural Network

http://playground.tensorflow.org



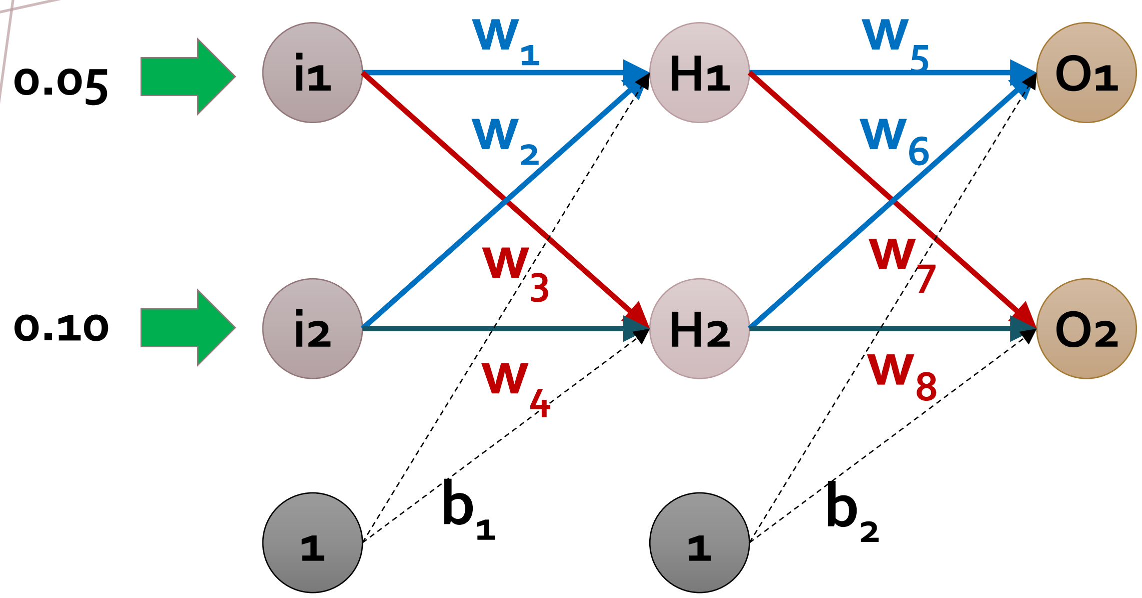
$w_1 = 0.15$ | $w_5 = 0.40$
$w_2 = 0.20$ | $w_6 = 0.45$
$w_3 = 0.25$ | $w_7 = 0.50$
$w_4 = 0.30$ | $w_8 = 0.55$
$b_1 = 0.35$ | $b_2 = 0.60$

Forward

Calculate total net input for H1, H2

$$NetH_1 = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$NetH_2 = w_3 * i_1 + w_4 * i_2 + b_1 * 1$$



$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

$$NetH_1 = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

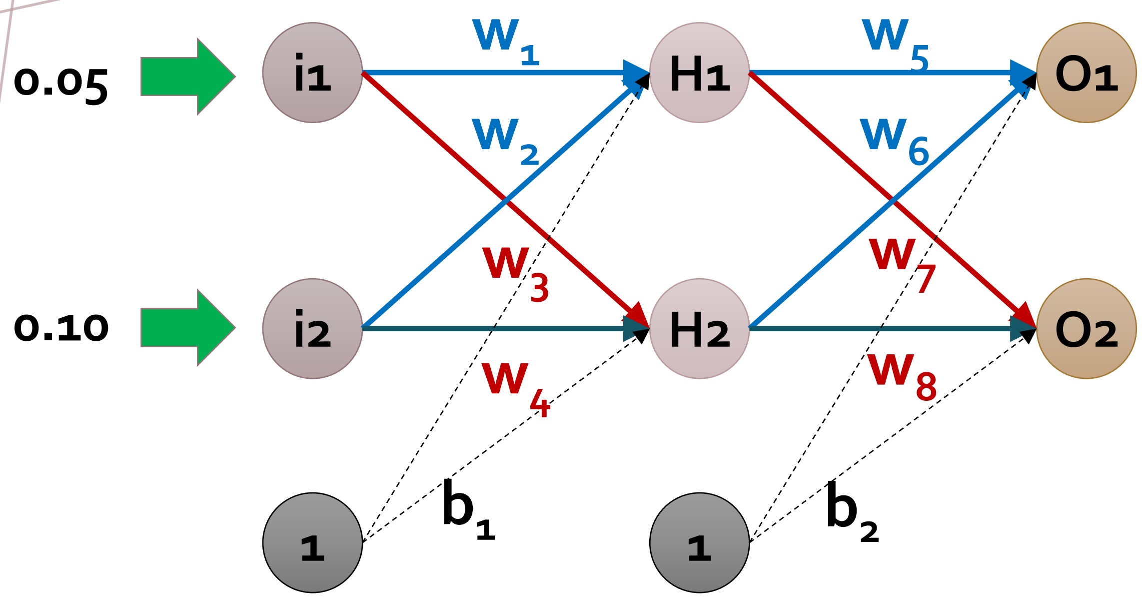
$$NetH_2 = w_3 * i_1 + w_4 * i_2 + b_1 * 1$$

Forward

Calculate total net input for H1, H2

$$NetH_1 = 0.15 * 0.05 + 0.20 * 0.10 + 0.35 * 1$$

$$NetH_2 = 0.25 * 0.05 + 0.30 * 0.10 + 0.35 * 1$$



$w_1 = 0.15$ | $w_5 = 0.40$
$w_2 = 0.20$ | $w_6 = 0.45$
$w_3 = 0.25$ | $w_7 = 0.50$
$w_4 = 0.30$ | $w_8 = 0.55$
$b_1 = 0.35$ | $b_2 = 0.60$

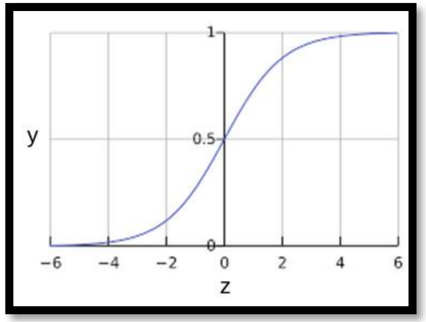
$$NetH_1 = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$NetH_2 = w_3 * i_1 + w_4 * i_2 + b_1 * 1$$

Forward

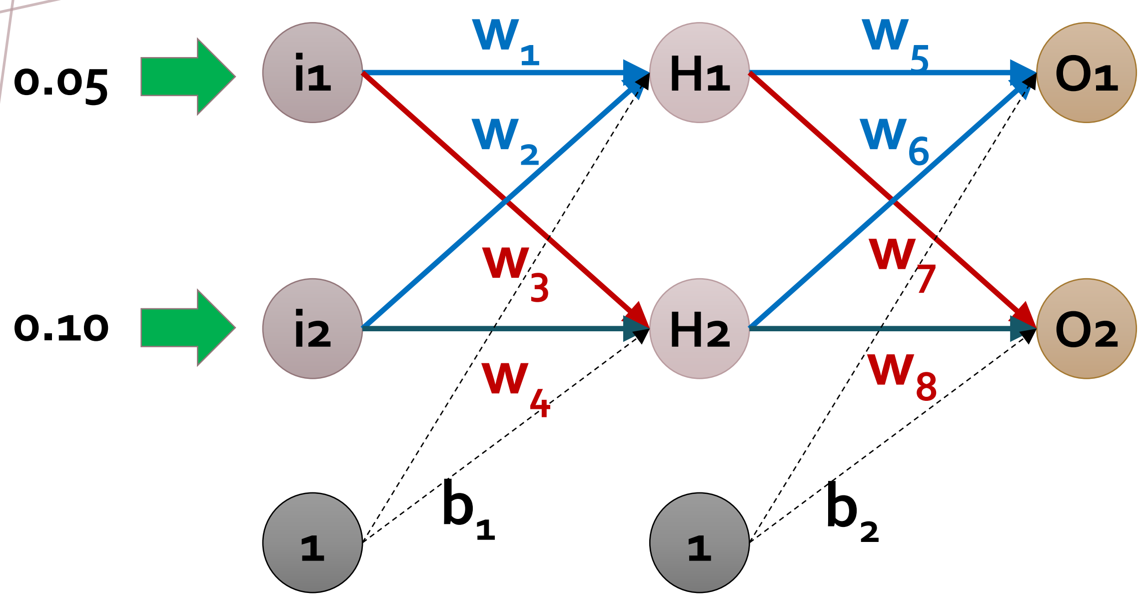
Calculate output for H1, H2

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

$$NetH_1 = 0.3775$$
$$NetH_2 = 0.3925$$



$$OutH_1 = 1/(1 + e^{-0.3775})$$
$$OutH_2 = 1/(1 + e^{-0.3925})$$



$w_1 = 0.15$ | $w_5 = 0.40$

$w_2 = 0.20$ | $w_6 = 0.45$

$w_3 = 0.25$ | $w_7 = 0.50$

$w_4 = 0.30$ | $w_8 = 0.55$

$b_1 = 0.35$ | $b_2 = 0.60$

$$NetH_1 = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$NetH_2 = w_3 * i_1 + w_4 * i_2 + b_1 * 1$$

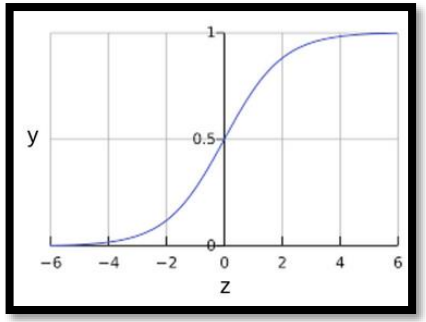
Forward

Calculate output for H1, H2

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

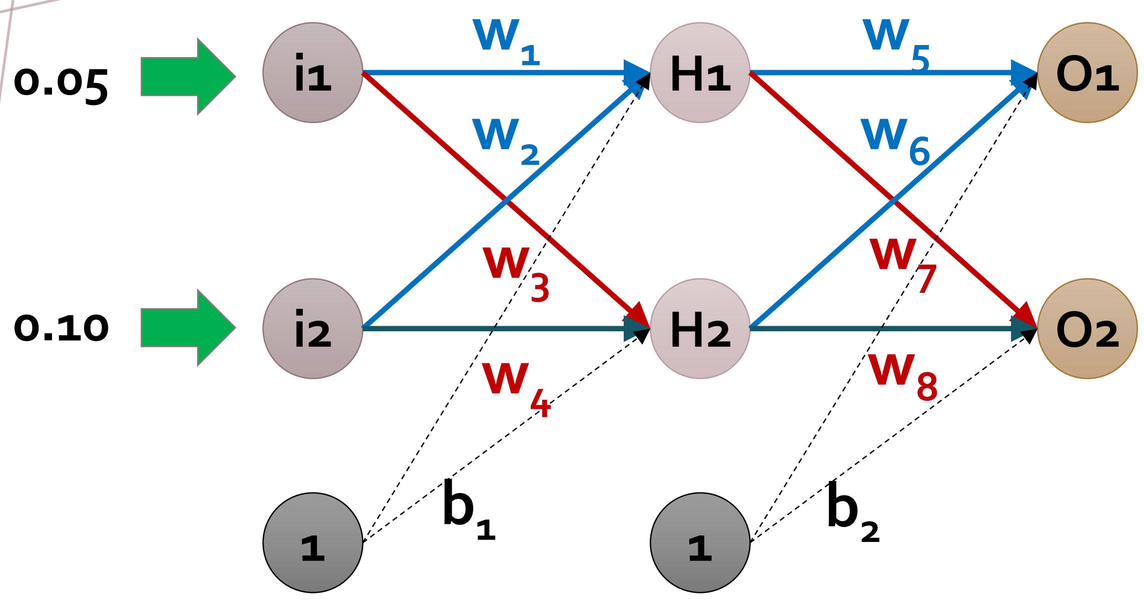
$$NetH_1 = 0.3775$$

$$NetH_2 = 0.3925$$



$$OutH_1 = 0.59326$$

$$OutH_2 = 0.59688$$



$w_1 = 0.15$

$w_2 = 0.20$

$w_3 = 0.25$

$w_4 = 0.30$

$b_1 = 0.35$

$w_5 = 0.40$

$w_6 = 0.45$

$w_7 = 0.50$

$w_8 = 0.55$

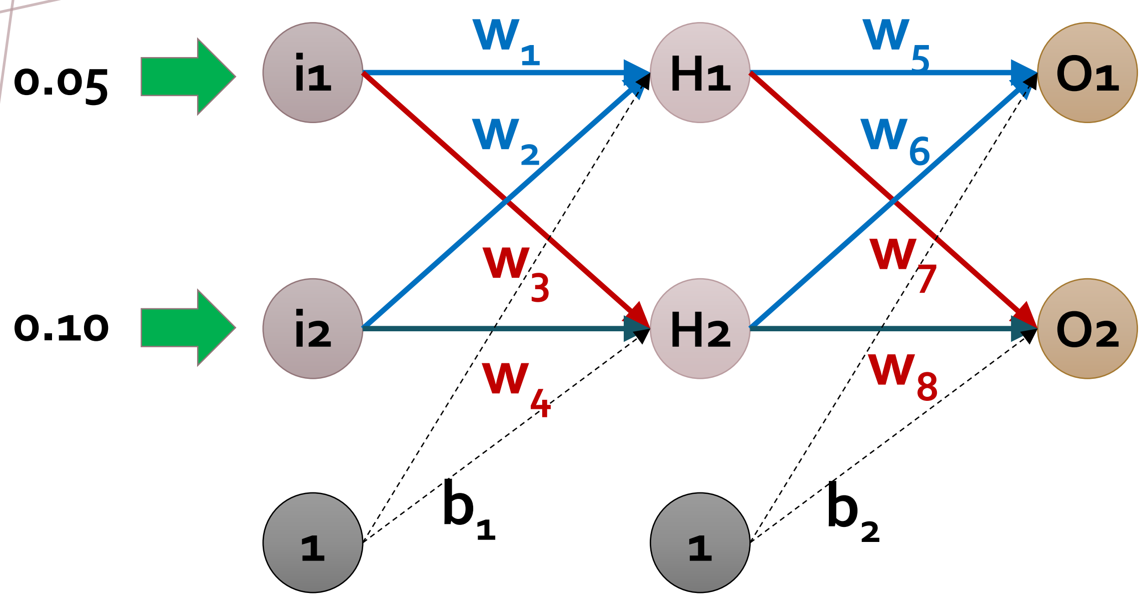
$b_2 = 0.60$

Forward

Calculate output for H1, H2

$$OutH_1 = 0.59326$$

$$OutH_2 = 0.59688$$



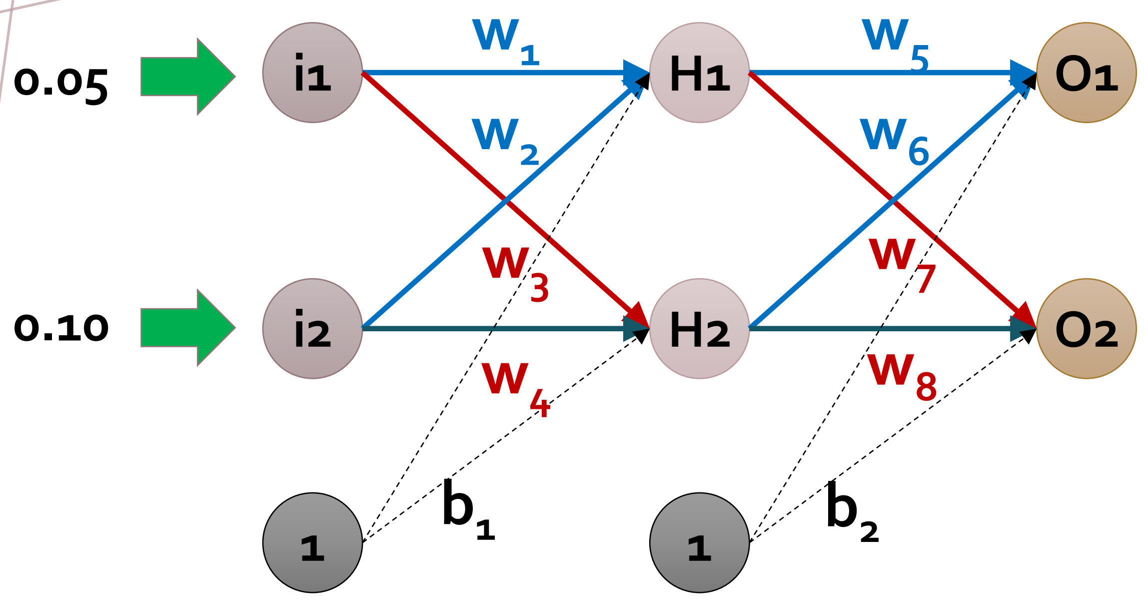
$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

Forward

Calculate total net input for O1, O2

$$OutH_1 = 0.59326 \qquad NetO_1 = 0.40 * 0.59326 + 0.45 * 0.59688 + 0.60 * 1$$

$$OutH_2 = 0.59688 \qquad NetO_2 = 0.50 * 0.59326 + 0.55 * 0.59688 + 0.35 * 1$$



$w_1 = 0.15$

$w_2 = 0.20$

$w_3 = 0.25$

$w_4 = 0.30$

$b_1 = 0.35$

$w_5 = 0.40$

$w_6 = 0.45$

$w_7 = 0.50$

$w_8 = 0.55$

$b_2 = 0.60$

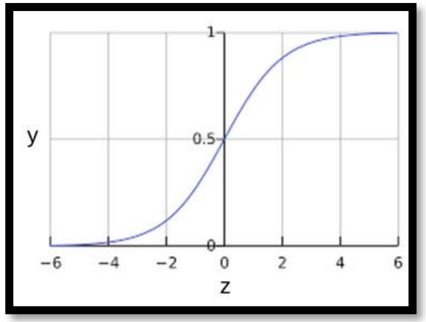
Forward

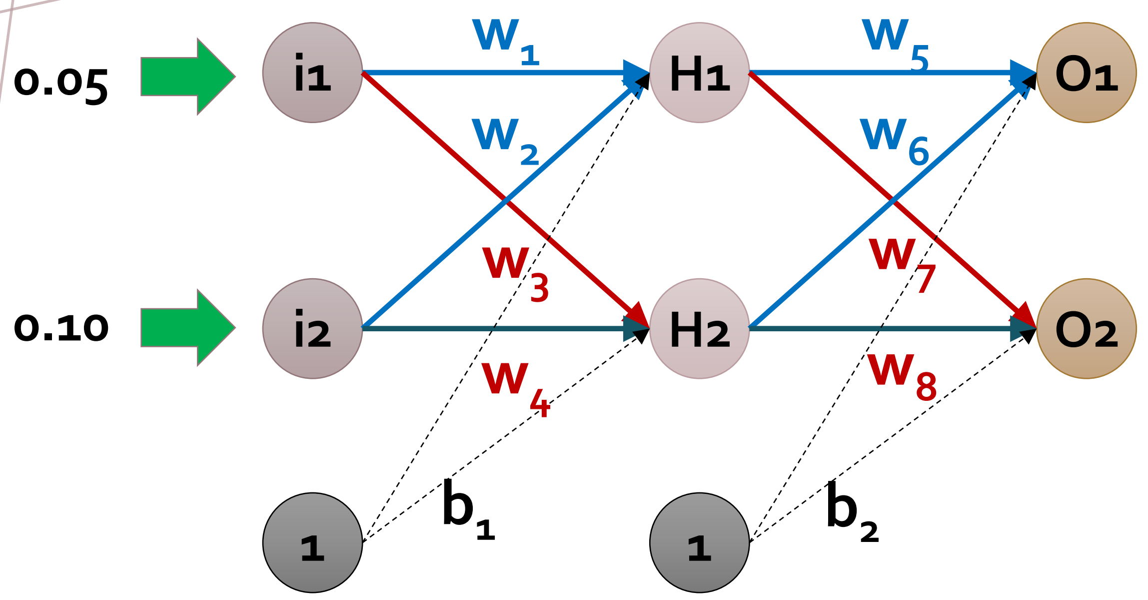
Calculate output for O1, O2

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

$$NetO_1 = 1.105900$$

$$NetO_2 = 0.974914$$





$w_1 = 0.15$

$w_2 = 0.20$

$w_3 = 0.25$

$w_4 = 0.30$

$b_1 = 0.35$

$w_5 = 0.40$

$w_6 = 0.45$

$w_7 = 0.50$

$w_8 = 0.55$

$b_2 = 0.60$

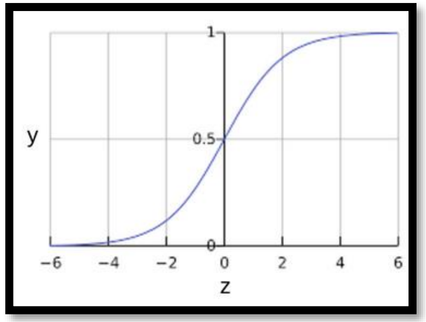
Forward

Calculate output for O1, O2

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

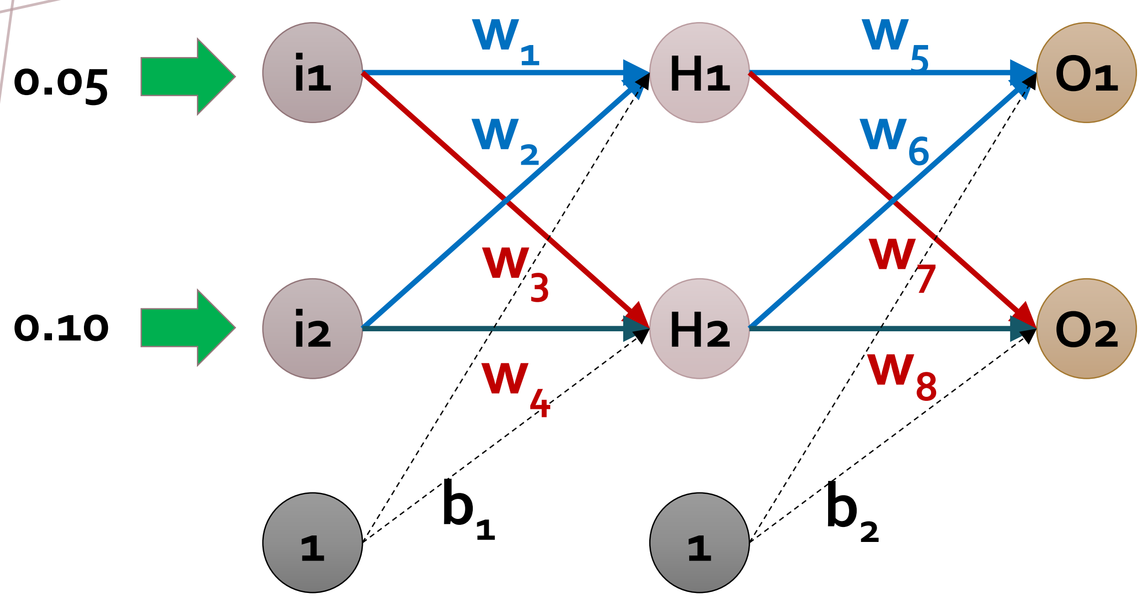
$$NetO_1 = 1.105900$$

$$NetO_2 = 0.974914$$



$$OutO_1 = 0.75136$$

$$OutO_2 = 0.77292$$



T1 = 0.01

T2 = 0.99

Target

$w_1 = 0.15$ | $w_5 = 0.40$

$w_2 = 0.20$ | $w_6 = 0.45$

$w_3 = 0.25$ | $w_7 = 0.50$

$w_4 = 0.30$ | $w_8 = 0.55$

$b_1 = 0.35$ | $b_2 = 0.60$

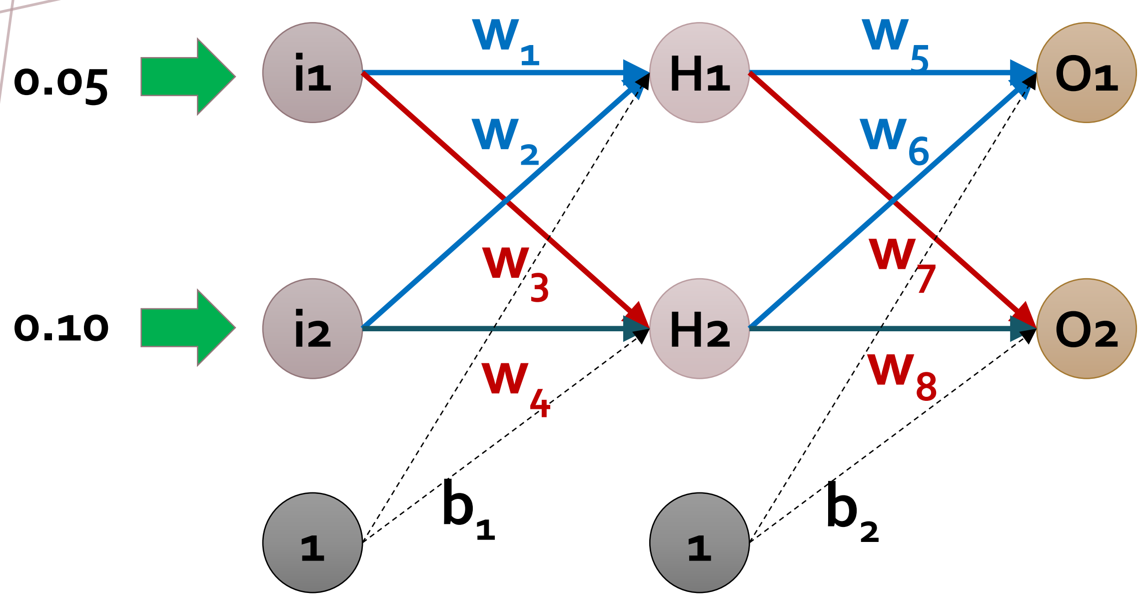
Calculating total error

[Squared error function]

$$E_{Total} = \sum \frac{1}{2}(\text{target} - \text{output})^2$$

$$OutO_1 = 0.75136$$

$$OutO_2 = 0.77292$$



T1 = 0.01

T2 = 0.99

Target

$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

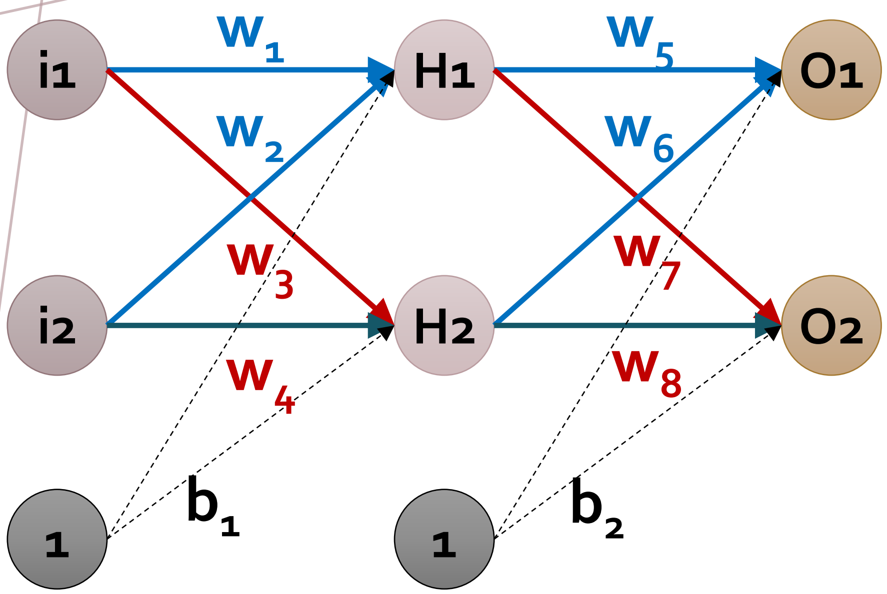
Calculating total error

[Squared error function]

$$E_{Total} = \sum \frac{1}{2}(\text{target} - \text{output})^2$$

$$OutO_1 = 0.75136 \qquad OutO_2 = 0.77292$$

$$\left.\begin{aligned} E_{O1} &= \frac{1}{2}(0.01 - 0.75136)^2 = 0.2748 \\ E_{O2} &= \frac{1}{2}(0.99 - 0.77292)^2 = 0.0235 \end{aligned}\right\} \quad E_{Total} = 0.2748 + 0.0235 = 0.2983$$



T1 = 0.01

T2 = 0.99

<u>Target</u>

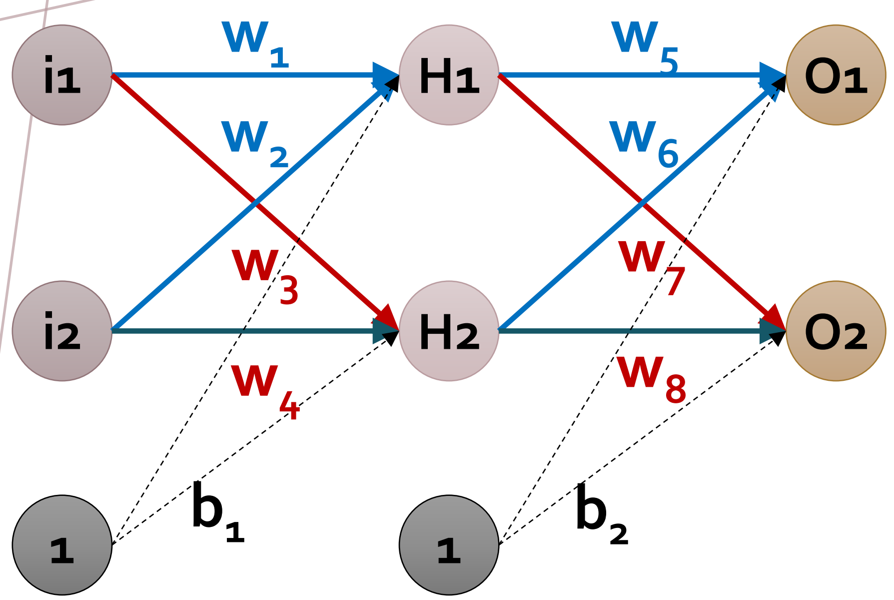
$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

Backward Pass

$$OutO_1 = 0.75136$$

$$OutO_2 = 0.77292$$

$$E_{Total} = 0.2748 + 0.0235 = 0.2983$$



T1 = 0.01

T2 = 0.99

Target

$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

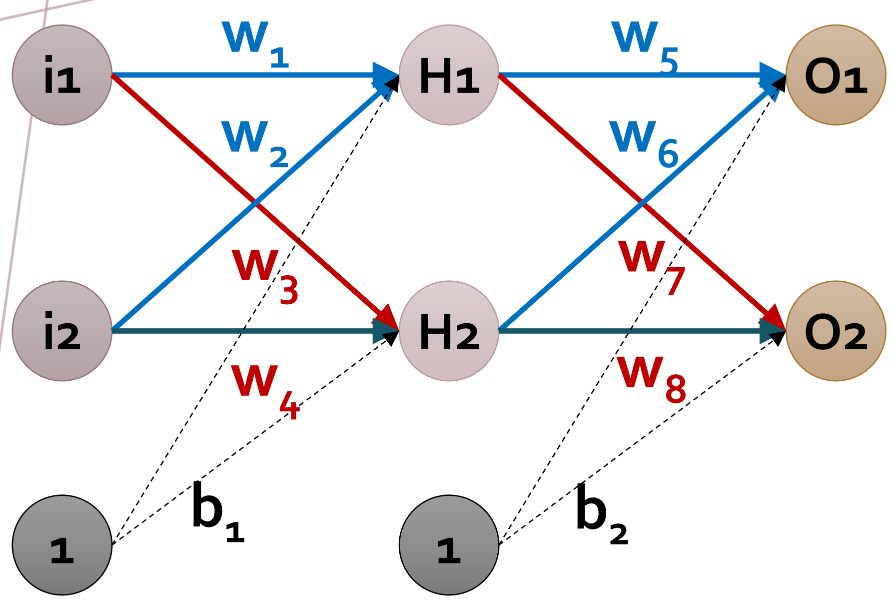
Backward Pass

How to find new weights?

$$OutO_1 = 0.75136$$

$$OutO_2 = 0.77292$$

$$E_{Total} = 0.2748 + 0.0235 = 0.2983$$



T1 = 0.01

T2 = 0.99

Target

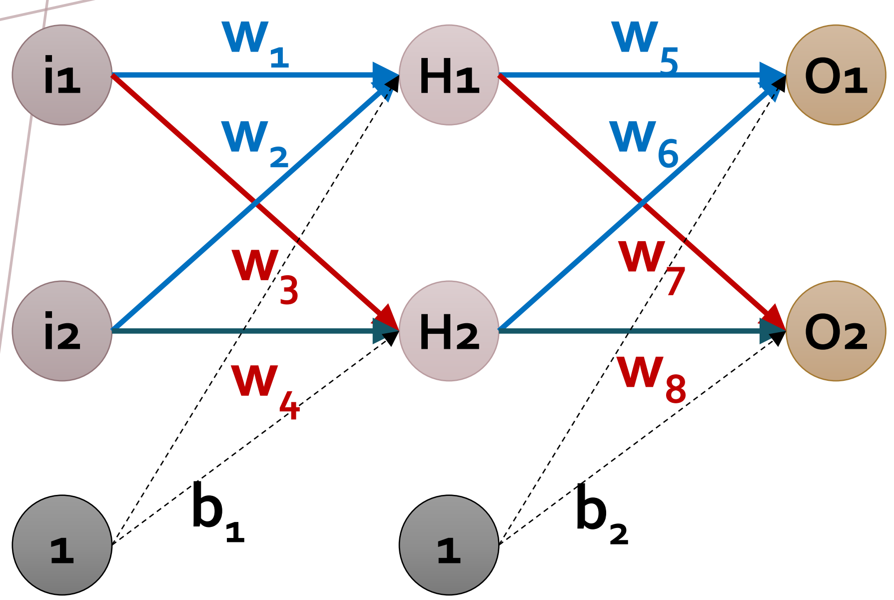
$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

Backward Pass

Consider w_5. We want to know how much change in w_5 affect the total error.

Applying the chain rule we know that…

$$\frac{\partial E_{Total}}{\partial \mathrm{w}_5} = \frac{\partial E_{Total}}{\partial \mathrm{Out}_{O1}} * \frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}} * \frac{\partial \mathrm{Net}_{O1}}{\partial \mathrm{w}_5}$$



T1 = 0.01

T2 = 0.99

Target

$w_1 = 0.15$ | $w_5 = 0.40$

$w_2 = 0.20$ | $w_6 = 0.45$

$w_3 = 0.25$ | $w_7 = 0.50$

$w_4 = 0.30$ | $w_8 = 0.55$

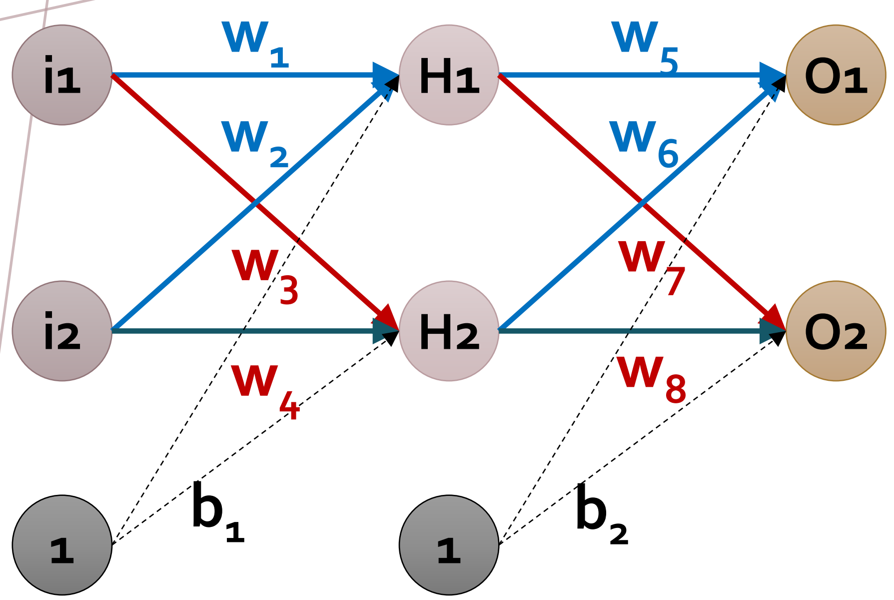
$b_1 = 0.35$ | $b_2 = 0.60$

Backward Pass

Consider w_5. We want to know how much change in w_5 affect the total error.

Applying the <u>chain rule</u> we know that…

$$\frac{\partial E_{Total}}{\partial \mathrm{w}_5} = \boxed{\frac{\partial E_{Total}}{\partial \mathrm{Out}_{O1}}} * \frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}} * \frac{\partial \mathrm{Net}_{O1}}{\partial \mathrm{w}_5}$$



T1 = 0.01

T2 = 0.99

Target

$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

Backward Pass

How much does the total error change with respect to the output?

$$E_{Total} = \frac{1}{2}(\text{target}_{O1} - \text{output}_{O1})^2 + \frac{1}{2}(\text{target}_{O2} - \text{output}_{O2})^2$$

$$\frac{\partial E_{Total}}{\partial \text{output}_{O1}} = 2 * \frac{1}{2}(\text{target}_{O1} - \text{output}_{O1})^{2-1} * (-1) + 0$$

Backward Pass

How much does the total error change with respect to the output?

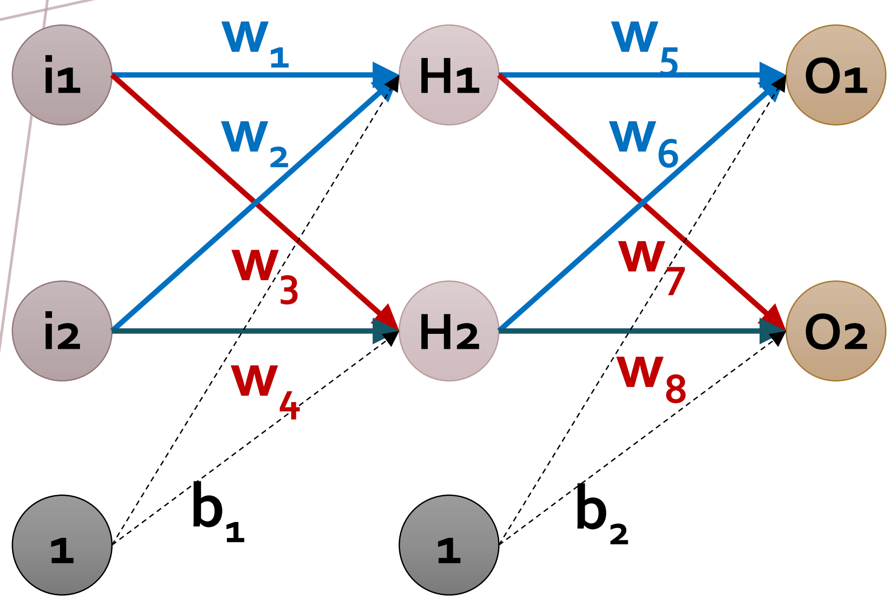
$$E_{Total} = \frac{1}{2}(\text{target}_{O1} - \text{output}_{O1})^2 + \frac{1}{2}(\text{target}_{O2} - \text{output}_{O2})^2$$

$$\frac{\partial E_{Total}}{\partial \text{output}_{O1}} = 2 * \frac{1}{2}(\text{target}_{O1} - \text{output}_{O1})^{2-1} * (-1) + 0$$

$$\frac{\partial E_{Total}}{\partial \text{output}_{O1}} = -(\text{target}_{O1} - \text{output}_{O1})$$

$$\boxed{\frac{\partial E_{Total}}{\partial \text{output}_{O1}} = \text{output}_{O1} - \text{target}_{O1}}$$

$$\boxed{\frac{\partial E_{Total}}{\partial \text{output}_{O2}} = \text{output}_{O2} - \text{target}_{O2}}$$



T1 = 0.01

T2 = 0.99

Target

$w_1 = 0.15$	$w_5 = 0.40$
$w_2 = 0.20$	$w_6 = 0.45$
$w_3 = 0.25$	$w_7 = 0.50$
$w_4 = 0.30$	$w_8 = 0.55$
$b_1 = 0.35$	$b_2 = 0.60$

Backward Pass

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial \mathrm{Out}_{O1}} * \boxed{\frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}}} * \frac{\partial \mathrm{Net}_{O1}}{\partial w_5}$$

Backward Pass

how much does the output of $O1$ change with respect to its total net input?

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial \mathrm{Out}_{O1}} * \boxed{\frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}}} * \frac{\partial \mathrm{Net}_{O1}}{\partial w_5}$$

$$\mathrm{Out}_{O1} = \frac{1}{1 + e^{-Net_{O1}}}$$

$$\frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}} = \mathrm{Out}_{O1}(1 - \mathrm{Out}_{O1})$$

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

$$\frac{d}{dx} f(x) = \frac{e^x \cdot (1+e^x) - e^x \cdot e^x}{(1+e^x)^2}$$

$$\frac{d}{dx} f(x) = \frac{e^x}{(1+e^x)^2} = f(x)(1-f(x))$$

Backward Pass

how much does the total net input of $O1$ change with respect to w_5 ?

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial Out_{O1}} * \frac{\partial Out_{O1}}{\partial Net_{O1}} * \boxed{\frac{\partial Net_{O1}}{\partial w_5}}$$

$$Net_{O1} = w_5 * OutH1 + w_6 * OutH2 + b_2 * 1$$

$$\frac{\partial Net_{O1}}{\partial w_5} = 1 * OutH1 * (w_5)^{1-1} + 0 + 0$$

$$\frac{\partial Net_{O1}}{\partial w_5} = OutH1$$

Backward Pass

how much does the total net input of $O1$ change with respect to w_5 ?

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial \mathrm{Out}_{O1}} * \frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}} * \frac{\partial \mathrm{Net}_{O1}}{\partial w_5}$$

$$\frac{\partial E_{Total}}{\partial \mathrm{output}_{O1}} = \mathrm{output}_{O1} - \mathrm{target}_{O1}$$

$$\frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}} = \mathrm{Out}_{O1}(1 - \mathrm{Out}_{O1})$$

$$\frac{\partial \mathrm{Net}_{O1}}{\partial w_5} = OutH1$$

Backward Pass

how much does the total net input of $O1$ change with respect to w_5 ?

$$\frac{\partial E_{Total}}{\partial \mathrm{w}_5} = \frac{\partial E_{Total}}{\partial \mathrm{Out}_{O1}} * \frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}} * \frac{\partial \mathrm{Net}_{O1}}{\partial \mathrm{w}_5}$$

$$\frac{\partial E_{Total}}{\partial \mathrm{output}_{O1}} = \mathrm{output}_{O1} - \mathrm{target}_{O1}$$

$$\frac{\partial \mathrm{Out}_{O1}}{\partial \mathrm{Net}_{O1}} = \mathrm{Out}_{O1}(1 - \mathrm{Out}_{O1})$$

$$\frac{\partial \mathrm{Net}_{O1}}{\partial \mathrm{w}_5} = OutH1$$

$$\frac{\partial E_{Total}}{\partial \mathrm{w}_5} = 0.08216$$

What's next ?

Backward Pass

how much does the total net input of $O1$ change with respect to w_5 ?

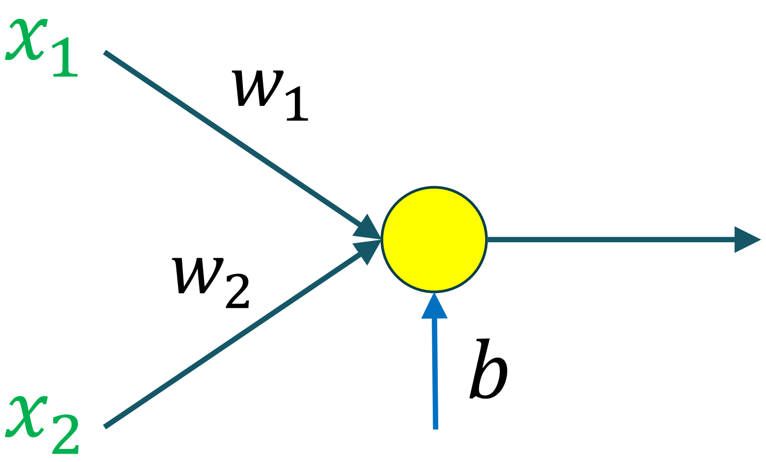
$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial Out_{O1}} * \frac{\partial Out_{O1}}{\partial Net_{O1}} * \frac{\partial Net_{O1}}{\partial w_5}$$

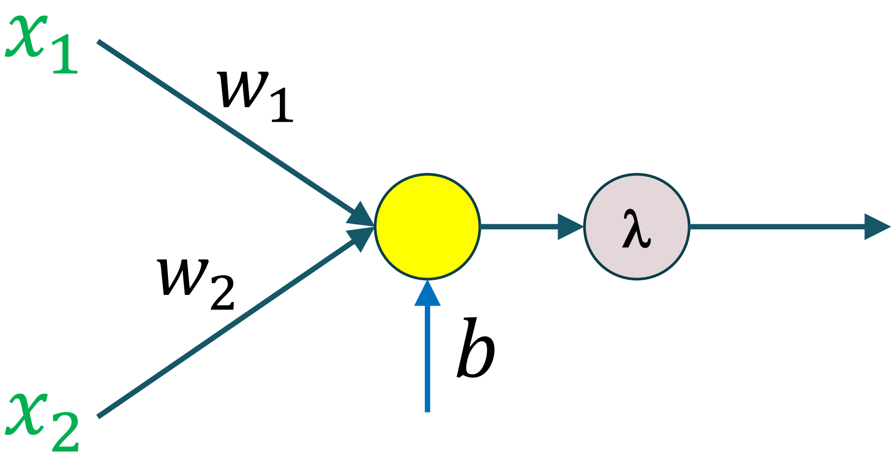
$$\boxed{\frac{\partial E_{Total}}{\partial w_5} = 0.08216}$$

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate)

$$\acute{w}_5 = w_5 - \eta \frac{\partial E_{Total}}{\partial w_5}$$

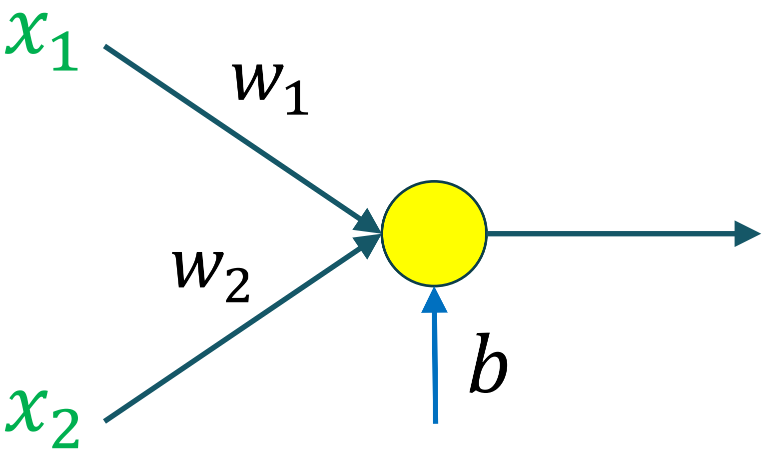
Keep Doing this Process until…?



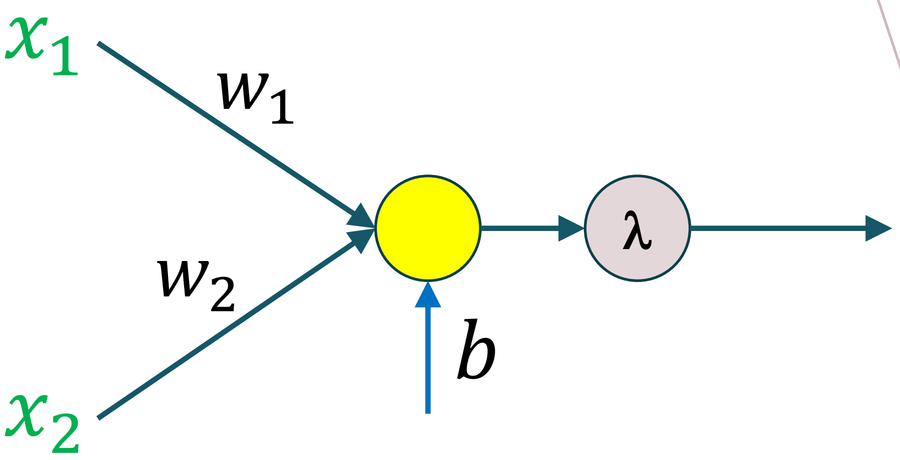


$$\acute{y} = w_1 x_1 + w_2 x_2 + b$$

$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

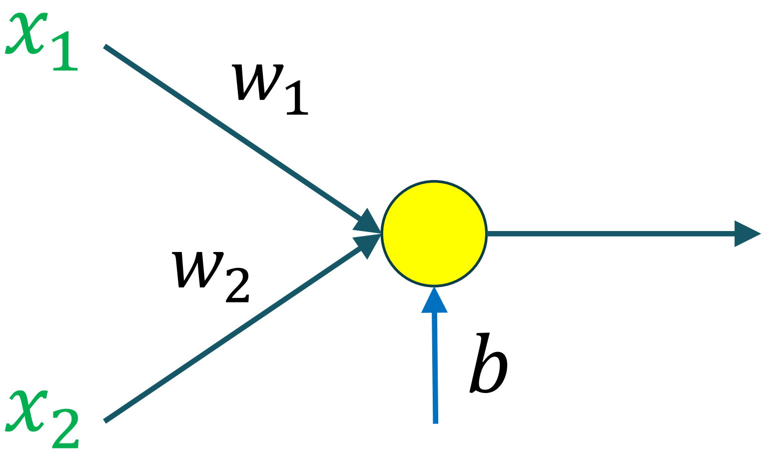


$$\acute{y} = w_1 x_1 + w_2 x_2 + b$$



$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

MSE $J = \frac{1}{N} \sum_{(x,y)\epsilon D} (y - \acute{y})^2$

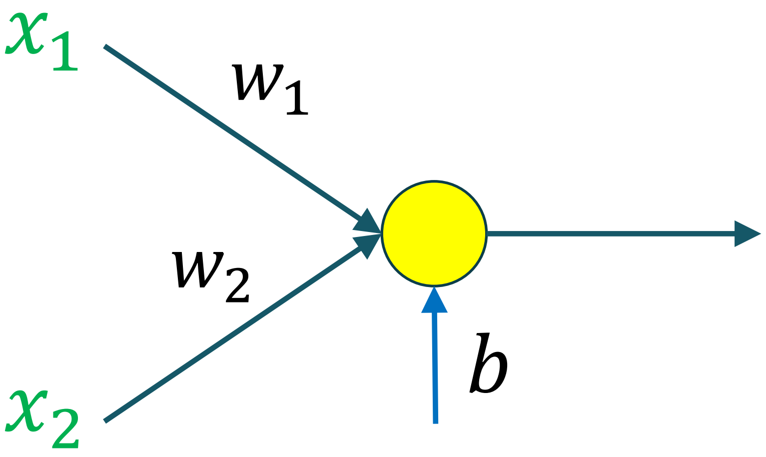


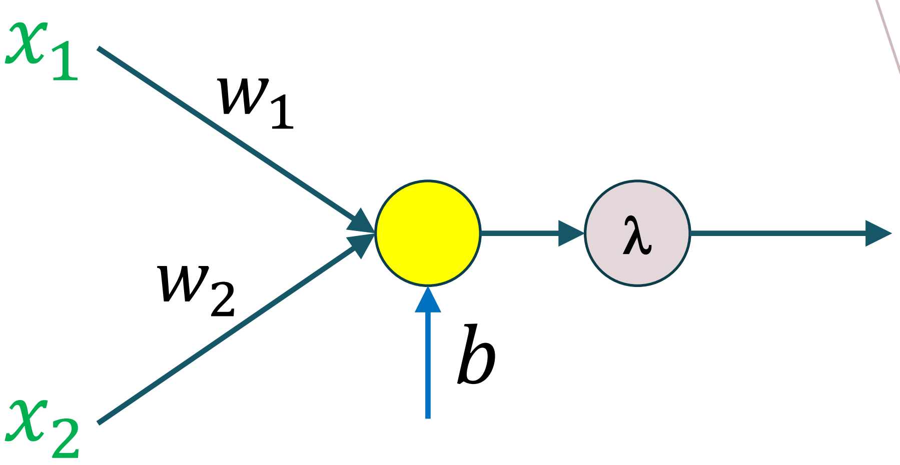
$$\acute{y} = w_1 x_1 + w_2 x_2 + b$$

MSE

$$J = \frac{1}{N} \sum_{(x,y) \epsilon D} (y - \acute{y})^2$$

$$J = \frac{1}{N} \sum_{(x,y) \epsilon D} (y - w^T x)^2$$





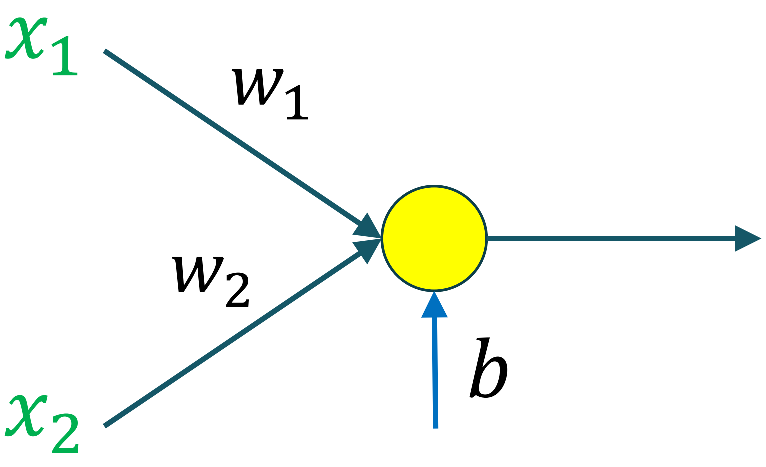
$$\acute{y} = w_1 x_1 + w_2 x_2 + b$$

$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

MSE $J = \dfrac{1}{N} \sum_{(x,y)\epsilon D} (y - \acute{y})^2$

Wait!!!

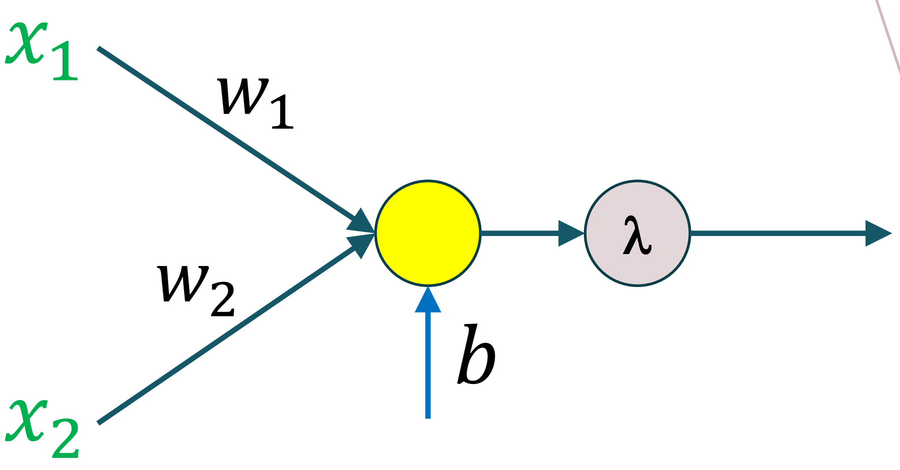
Our prediction function is Non-linear



$$\acute{y} = w_1 x_1 + w_2 x_2 + b$$

Wait!!!

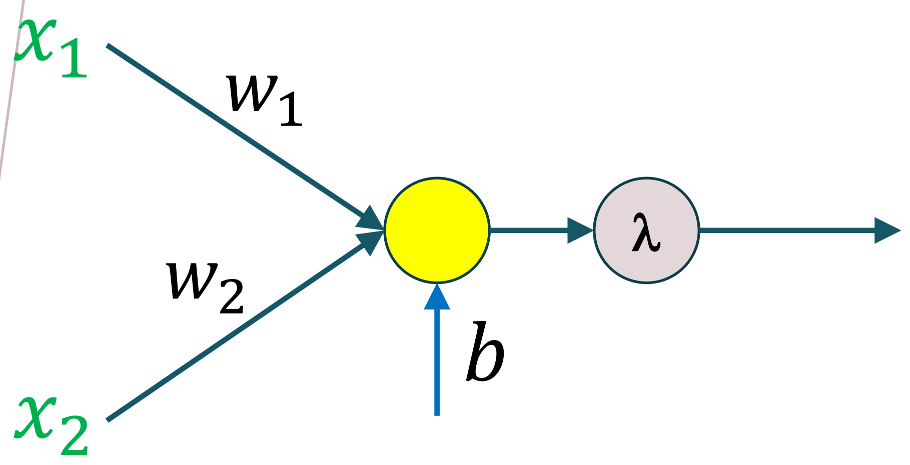
Our prediction function is Non-linear



$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

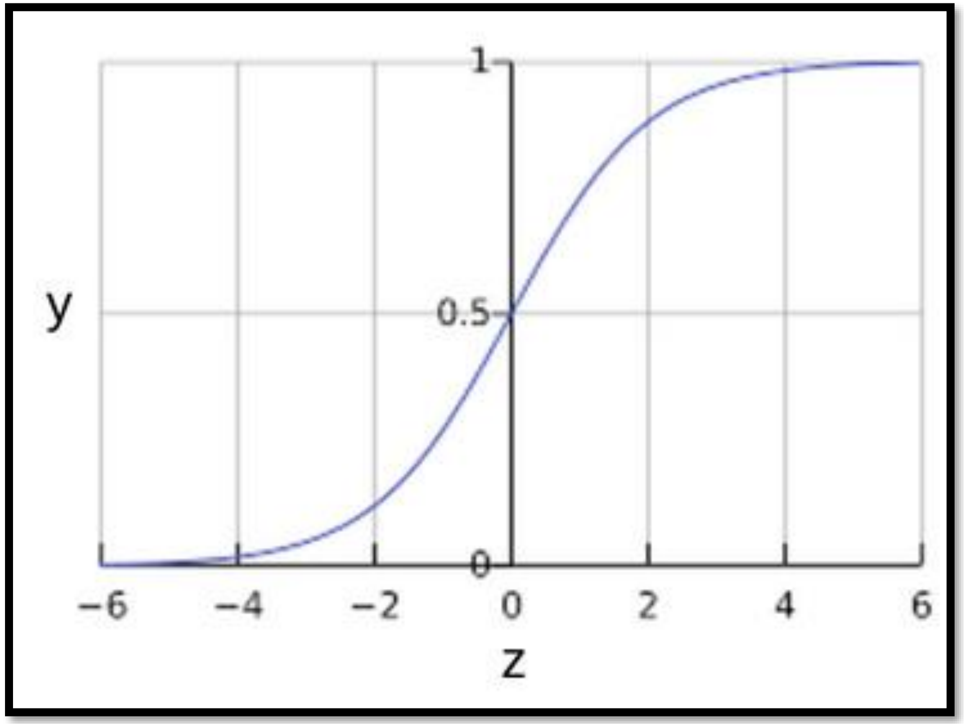
MSE $J = \frac{1}{N} \sum_{(x,y)\epsilon D} (y - \acute{y})^2$

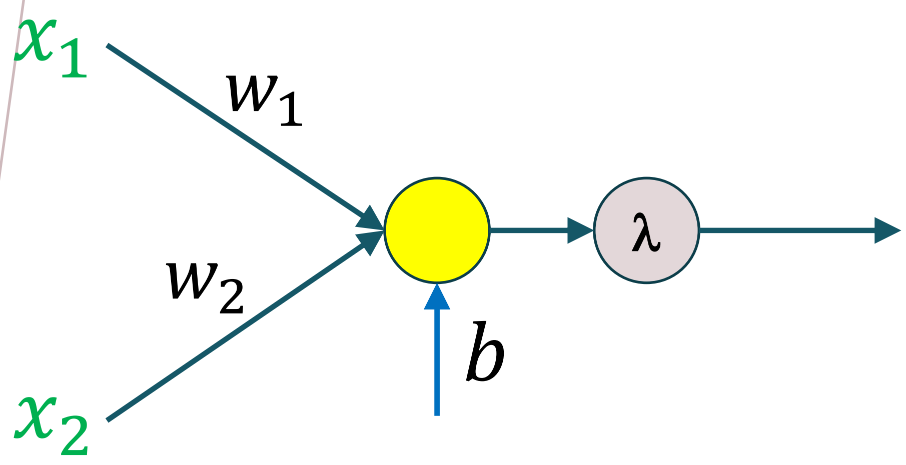
Squaring this prediction as we do in MSE results in a non-convex function with many local minimums. If our cost function has many local minimums, gradient descent may not find the optimal global minimum.

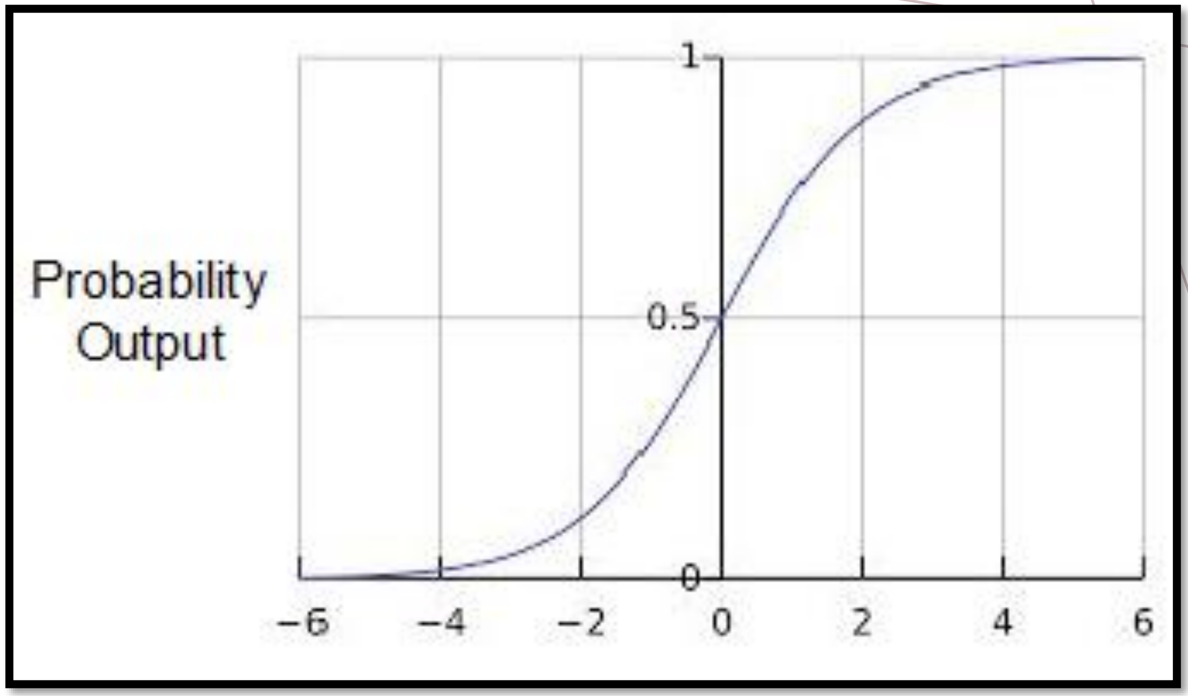


$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

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

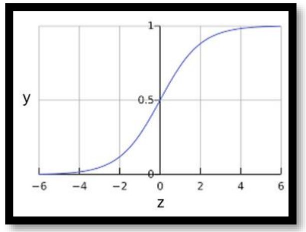






$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

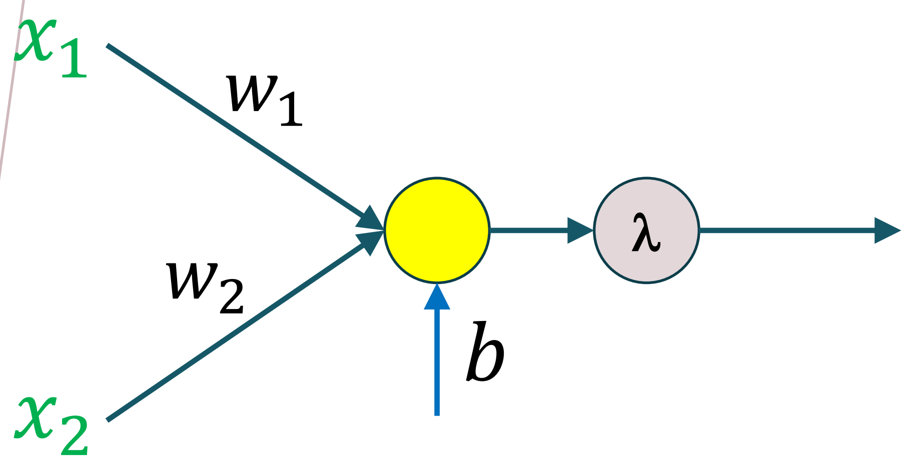
$$z = (w_0 + w_1 x_1 + w_2 x_2)$$



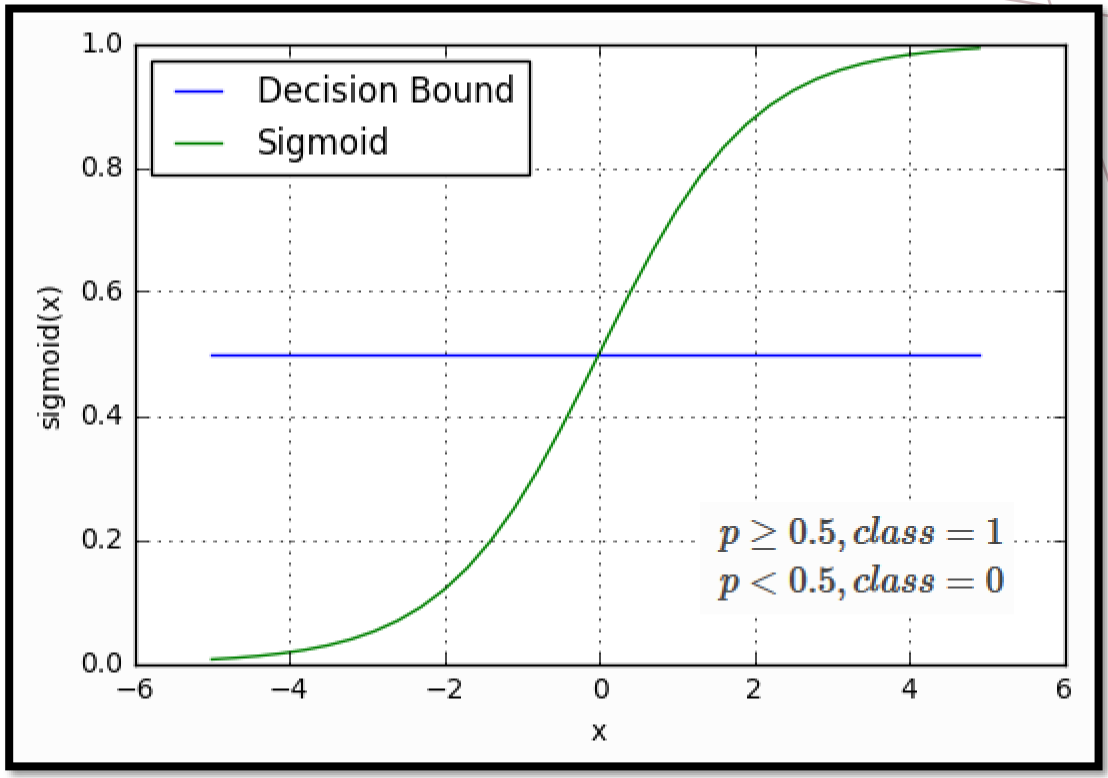
$$y(z) = \frac{1}{1 + e^{-z}}$$

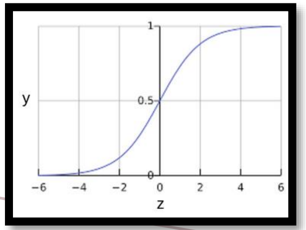
$$z = \log\left(\frac{y}{1-y}\right)$$

"Log-odds"

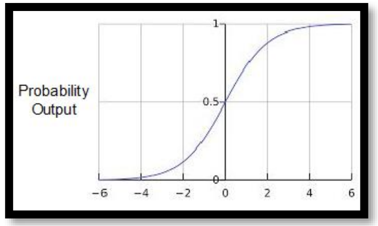


$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$





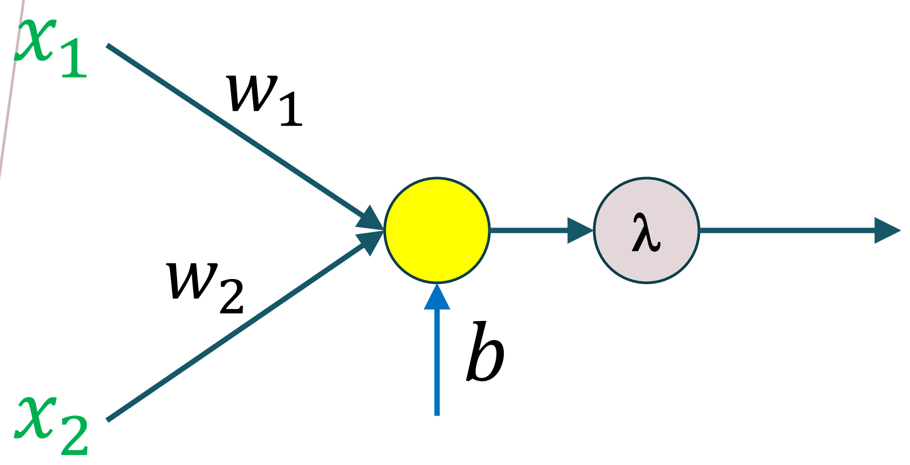
$$y(z) = \frac{1}{1 + e^{-z}}$$



$$z = \log\left(\frac{y}{1 - y}\right)$$

Neural Network : Kernel

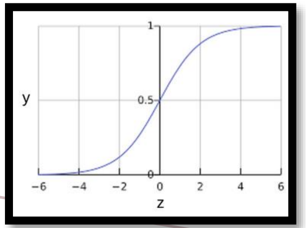
Cross-entropy (Log Loss)



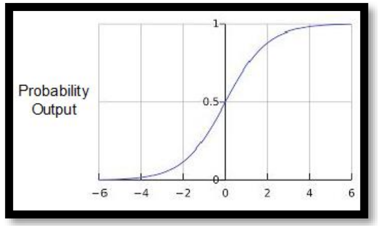
$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

$$J = \sum_{(x,y)\epsilon D} -y\log(\acute{y}) - (1-y)\log(1-\acute{y})$$

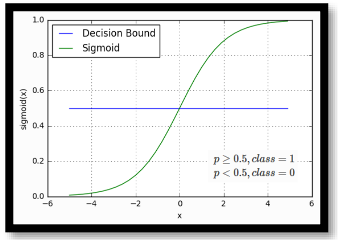
The equation for Log Loss is closely related to [Shannon's Entropy measure from Information Theory](). It is also the negative logarithm of the [likelihood function](), assuming a [Bernoulli distribution]() of *y*. Indeed, minimizing the loss function yields a maximum likelihood estimate.

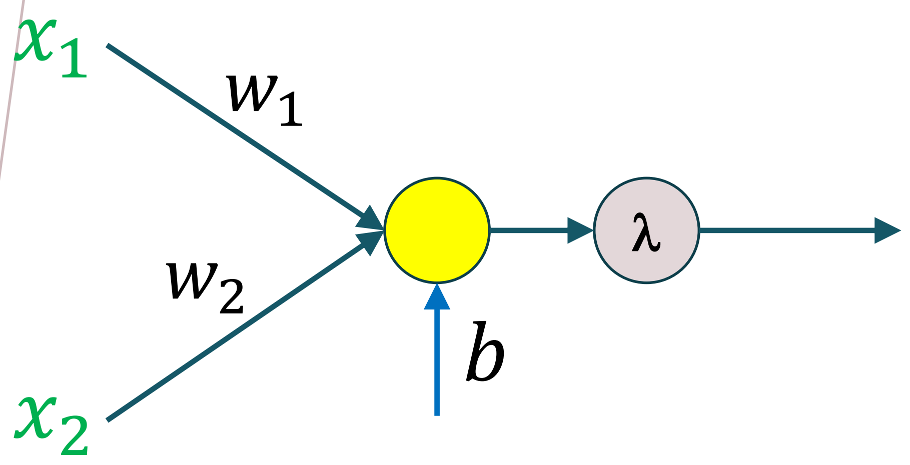


$$y(z) = \frac{1}{1+e^{-z}}$$



$$z = \log\left(\frac{y}{1-y}\right)$$





$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

Imagine this, prediction function as a Posterior Prob.

$$P(\acute{y} = 1|x) = \lambda(w^T x)$$

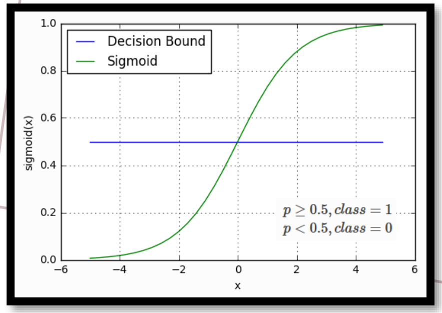
$$P(\acute{y} = 0|x) = 1 - \lambda(w^T x)$$

Likelihood

Posterior Prob.

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$$

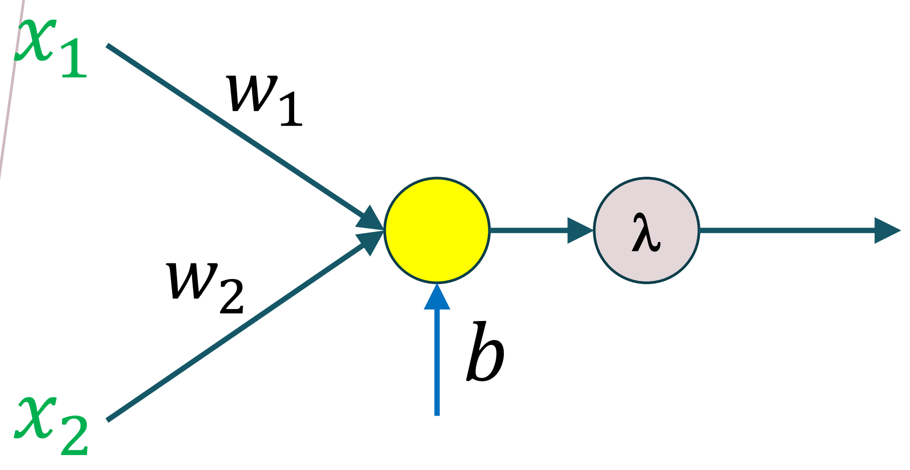
Prior Prob.



Cross-entropy (Log Loss)

$$J = \sum_{(x,y)\epsilon D} -y\log(\acute{y}) - (1-y)\log(1-\acute{y})$$

Neural Network : Kernel



$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

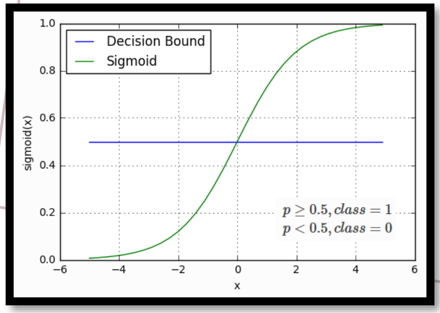
Imagine this, prediction function as a Posterior Prob.

$$P(\acute{y} = 1|x) = \lambda(w^T x)$$

$$P(\acute{y} = 0|x) = 1 - \lambda(w^T x)$$

Why **Bernoulli distribution**?

The probability distribution of any single [experiment]() that asks a [yes–no question]()



Cross-entropy (Log Loss)

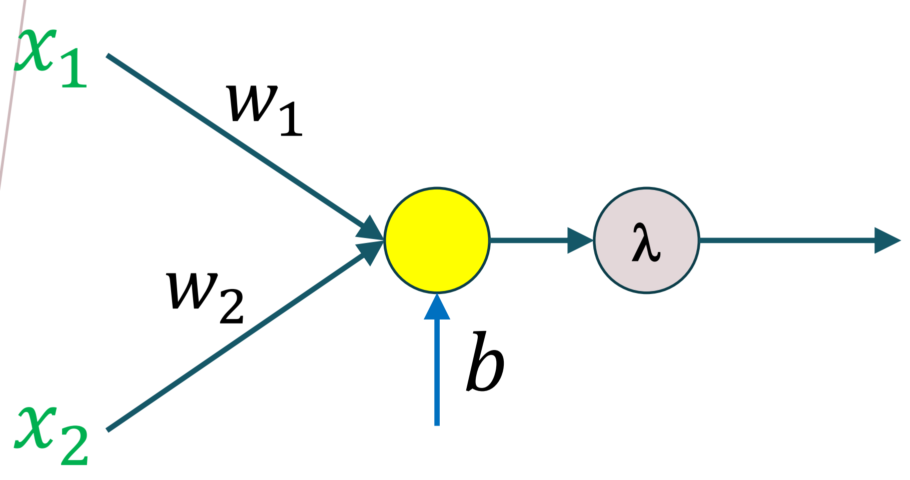
$$J = \sum_{(x,y)\epsilon D} -y\log(\acute{y}) - (1-y)\log(1-\acute{y})$$

Neural Network : Kernel

Imagine this, prediction function as a Posterior Prob

$P(\acute{y} = 1 | x) = \lambda(w^T x)$

$P(\acute{y} = 0 | x) = 1 - \lambda(w^T x)$



$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b)$$

Why **Bernoulli distribution**?

The probability distribution of any single experiment that asks a yes–no question

If X is random variable (R.V.)

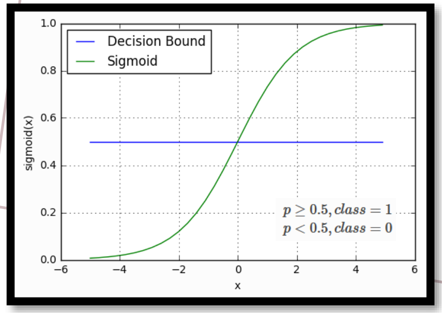
$$Pr(X = 1) = p = 1 - Pr(X = 0) = 1 - q$$

The probability mass function (PMF), over possible outcomes k is

$$f(p;k) = \begin{cases} p & \text{if } k = 1 \\ 1 - p & \text{if } k = 0 \end{cases}$$

$$f(p;k) = p^k (1-p)^{1-k}$$

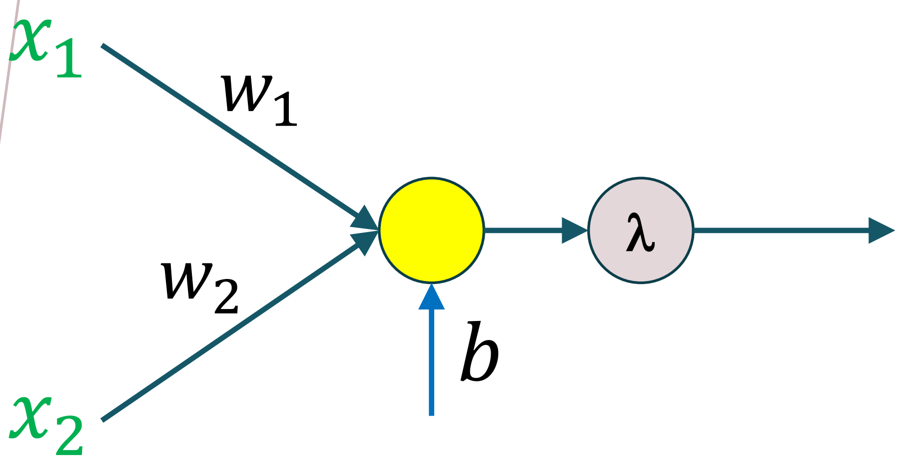
for $k \in \{0,1\}$



Cross-entropy (Log Loss)

$$J = \sum_{(x,y)\epsilon D} -y \log(\acute{y}) - (1-y)\log(1-\acute{y})$$

Neural Network : Kernel



$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

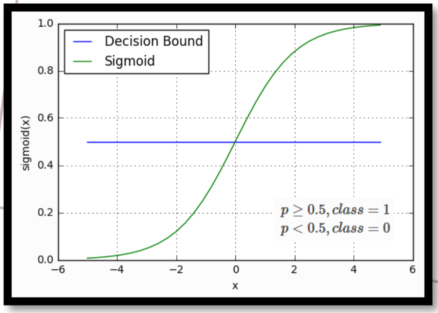
Imagine this, prediction function as a Posterior Prob.

$$P(\acute{y} = 1|x) = \lambda(w^T x)$$

$$P(\acute{y} = 0|x) = 1 - \lambda(w^T x)$$

Write this more compactly as…

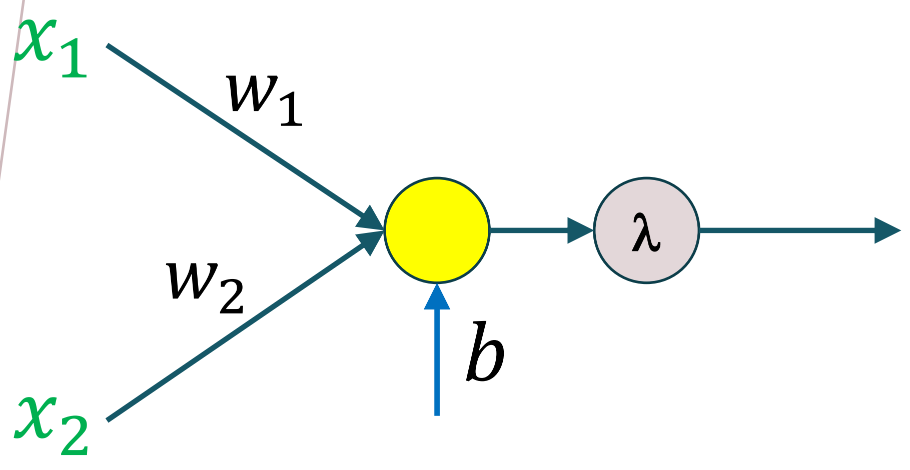
$$P(\acute{y}|x) = \left(\lambda(w^T x)\right)^{\acute{y}} \left(1 - \lambda(w^T x)\right)^{1-\acute{y}}$$



Cross-entropy (Log Loss)

$$J = \sum_{(x,y)\epsilon D} -y \log(\acute{y}) - (1-y)\log(1-\acute{y})$$

Neural Network : Kernel



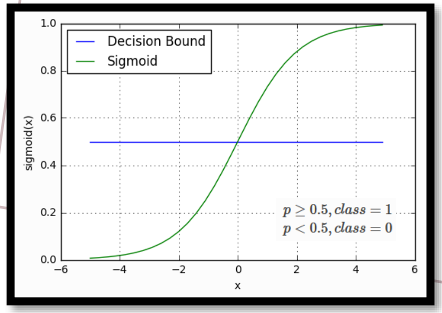
$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

Imagine this, prediction function as a Posterior Prob.

$$P(\acute{y}|x) = \left(\lambda(w^T x)\right)^{\acute{y}} \left(1 - \lambda(w^T x)\right)^{1-\acute{y}}$$

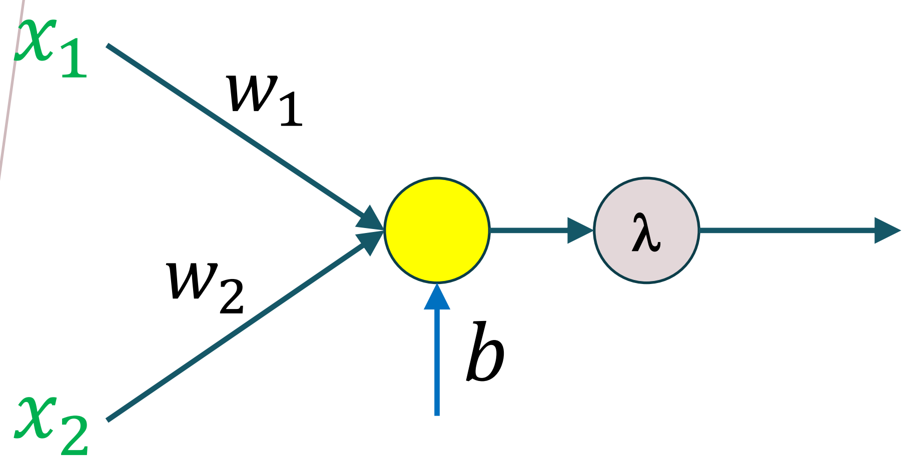
Then, the likelihood (assuming data independence) is…

$$P(x|y) = \prod_{i}^{N} \left(\lambda(w^T x)\right)^{y} \left(1 - \lambda(w^T x)\right)^{1-y}$$



Cross-entropy (Log Loss)

$$J = \sum_{(x,y)\epsilon D} -y \log(\acute{y}) - (1-y)\log(1-\acute{y})$$



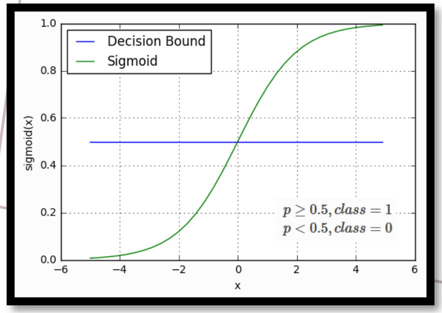
$$\acute{y} = \lambda(w_1 x_1 + w_2 x_2 + b\ )$$

Then, the likelihood (assuming data independence) is…

$$P(x|y) = \prod_{i}^{N} \left(\lambda(w^T x)\right)^{y} \left(1 - \lambda(w^T x)\right)^{1-y}$$

Finally, the negative log likelihood is…

$$J(w) = \sum_{i}^{N} -y \log\left(\lambda(w^T x)\right) - (1 - y) \log\left(\left(\lambda(w^T x)\right)\right)$$



Cross-entropy (Log Loss)

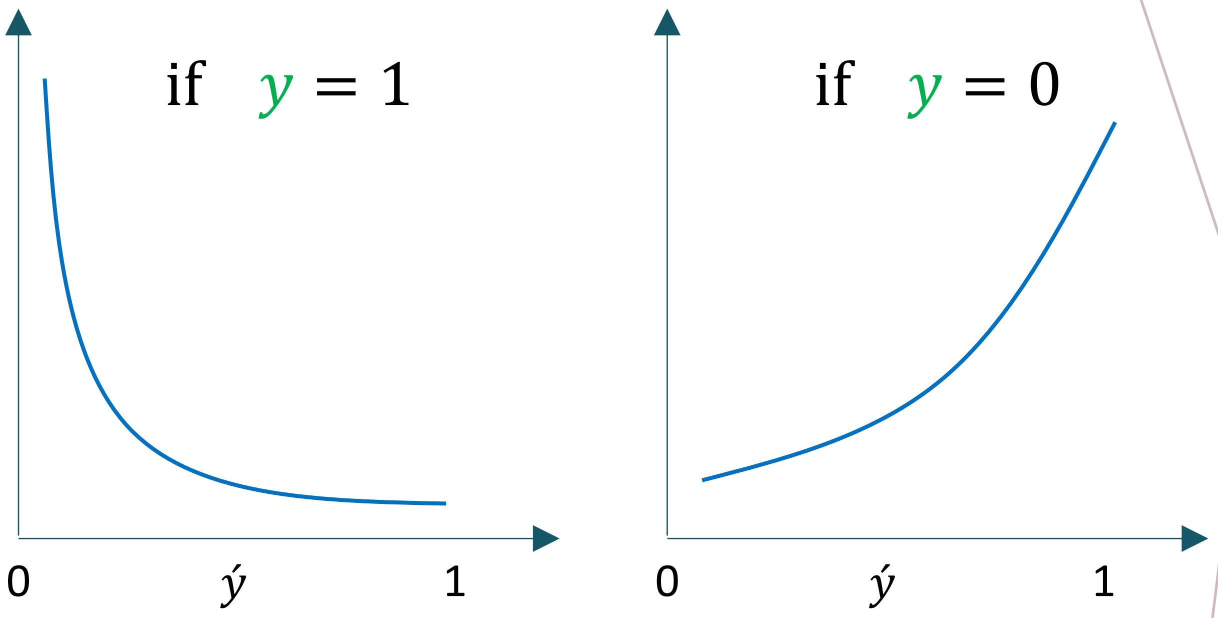
$$J = \sum_{(x,y)\epsilon D} -y \log(\acute{y}) - (1 - y) \log(1 - \acute{y})$$

Cross-entropy

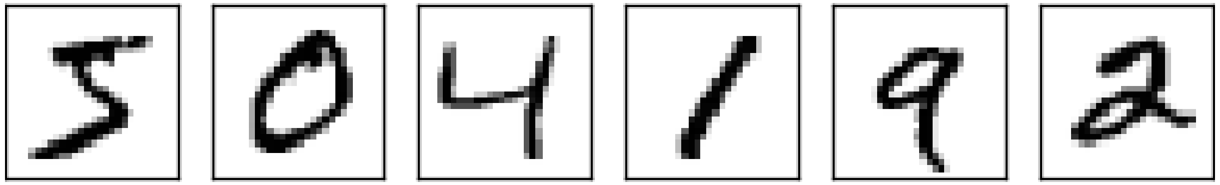
$$J(w) = \sum_{(x,y)\epsilon D} \text{Cost}(y, \acute{y})$$

$$\text{Cost}(y, \acute{y}) = -\log(\acute{y}) \qquad \text{if} \quad y = 1$$

$$\text{Cost}(y, \acute{y}) = -\log(1 - \acute{y}) \qquad \text{if} \quad y = 0$$



Workshop



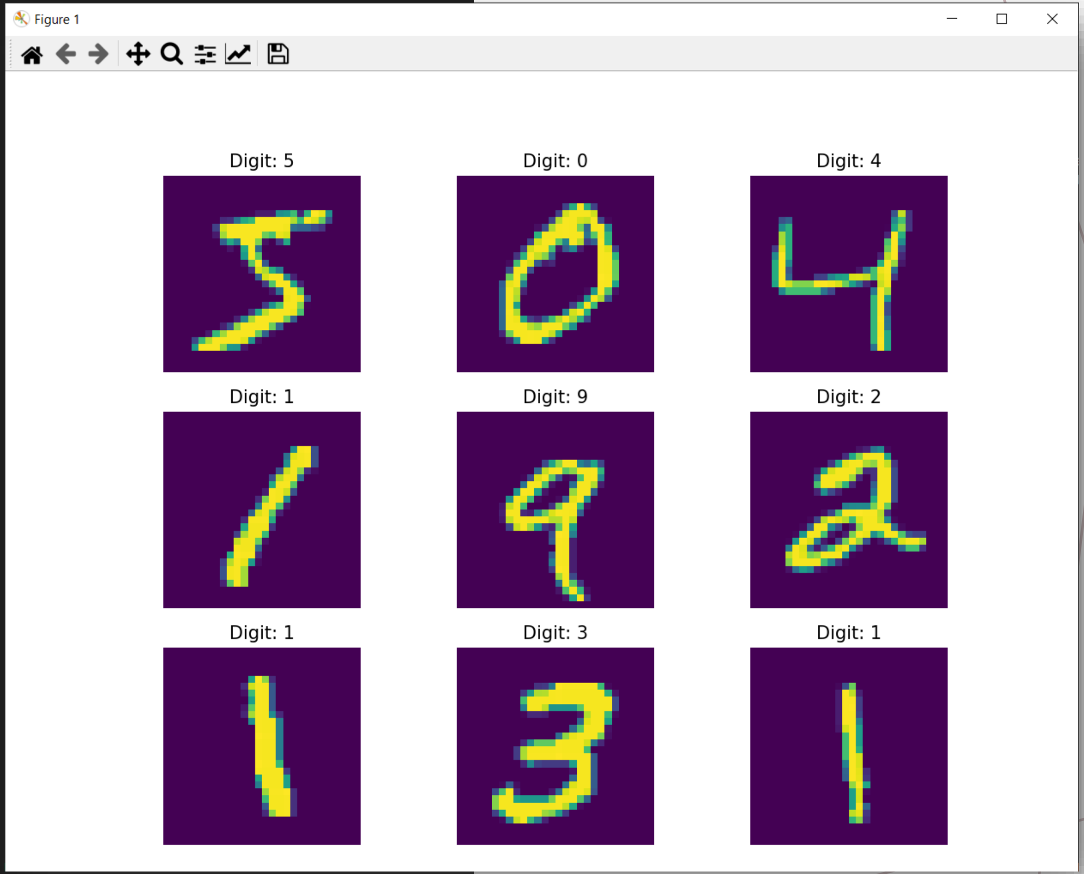
- We would like to create a model to classify all handwritten digits (0 – 9)
- Training dataset contain 3,000 images for each class.
- Input layer contains 784 = 28 x 28 neurons.

TensorFlow is an open-source software library for high performance numerical computation.
Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

Originally developed by researchers and engineers from the **Google Brain** *team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.*

1.) Load data

```python
import tensorflow as tf
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(train_images, train_labels) , (test_images, test_labels) = mnist.load_data()

# Printing the shapes
print("train_images shape: ", train_images.shape)
print("train_labels shape: ", train_labels.shape)
print("test_images shape: ", test_images.shape)
print("test_labels shape: ", test_labels.shape)

# Displaying first 9 images of dataset
fig = plt.figure(figsize=(10,10))

nrows=3
ncols=3
for i in range(9):
    fig.add_subplot(nrows, ncols, i+1)
    plt.imshow(train_images[i])
    plt.title("Digit: {}".format(train_labels[i]))
    plt.axis(False)
plt.show()

```

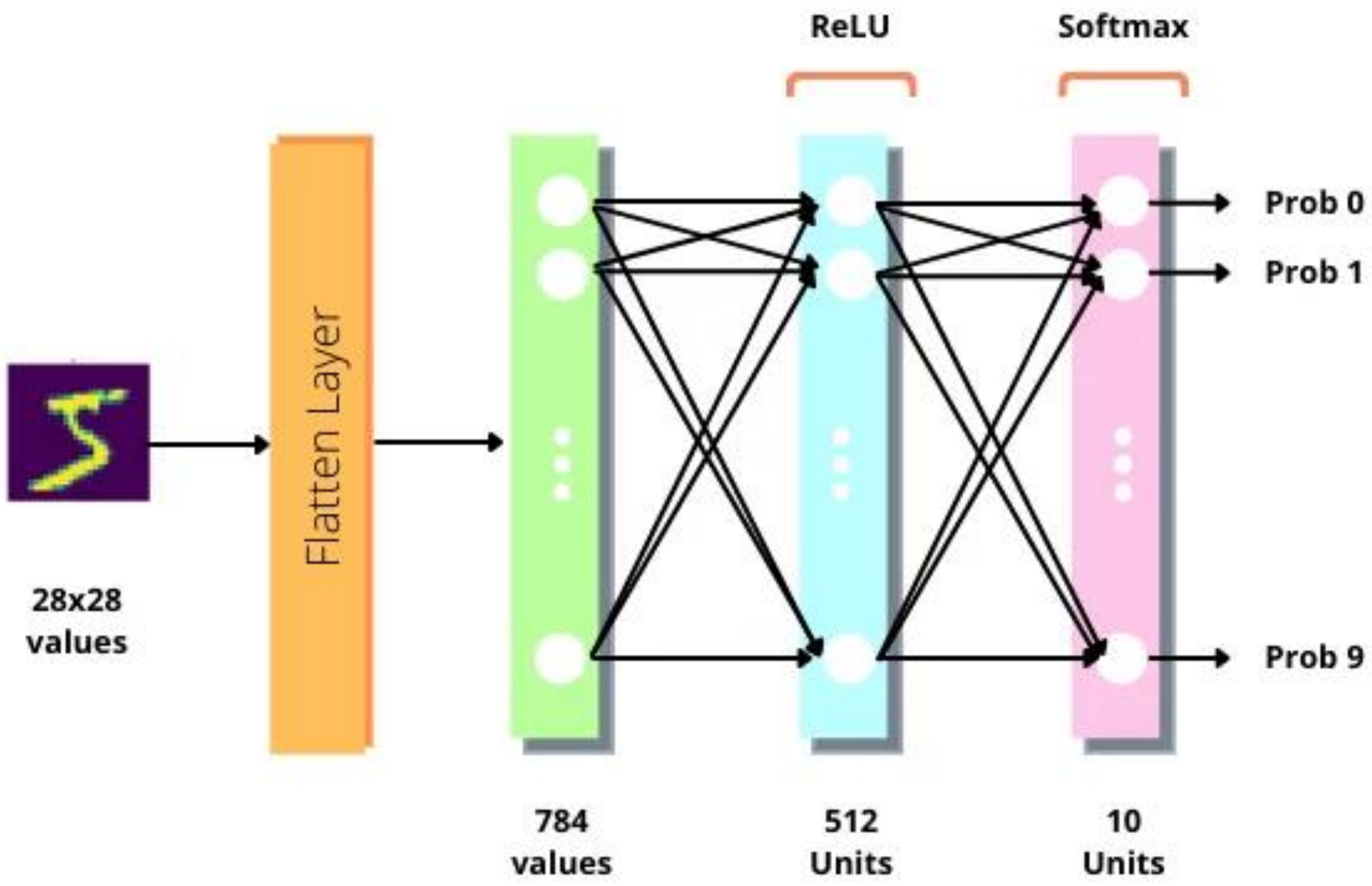


2.) Preprocessing the Data

```python
# Converting image pixel values to 0 - 1
train_images = train_images / 255
test_images = test_images / 255

print("First Label before conversion:")
print(train_labels[0])

# Converting labels to one-hot encoded vectors
train_labels = tf.keras.utils.to_categorical(train_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)

print("First Label after conversion:")
print(train_labels[0])

```

```
First Label before conversion:
5
First Label after conversion:
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

3.) Build Neural Network Model



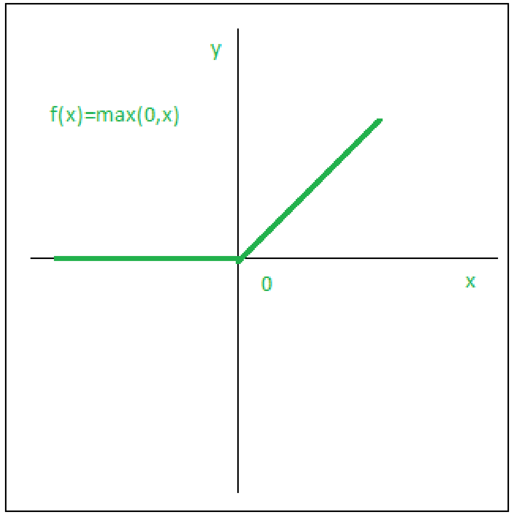
Flatten Layer

Convert 2D array ➔ 1D array

Number of Input Node

784 Node ➔ 28 x 28 pixels

Activation Function



Rectified linear unit (ReLU)

3.) Build Neural Network Model

```python

# === Part 3 === #
# Using Sequential() to build layers one after another
model = tf.keras.Sequential([

    # Flatten Layer that converts images to 1D array
    tf.keras.layers.Flatten(),

    # Hidden Layer with 512 units and relu activation
    tf.keras.layers.Dense(units=512, activation='relu'),

    # Output Layer with 10 units for 10 classes and softmax activation
    tf.keras.layers.Dense(units=10, activation='softmax')
])

```

4.) Compiling the Model

Three attributes given to the model during the models compile step

Loss Function: This tells our model how to find the error between the actual label and the label predicted by the model.

Optimizer: This tells our model how to update weights/parameters of the model by looking at the data and loss function value.

Metrics (Optional): It contains a list of metrics used to monitor the train and test steps.

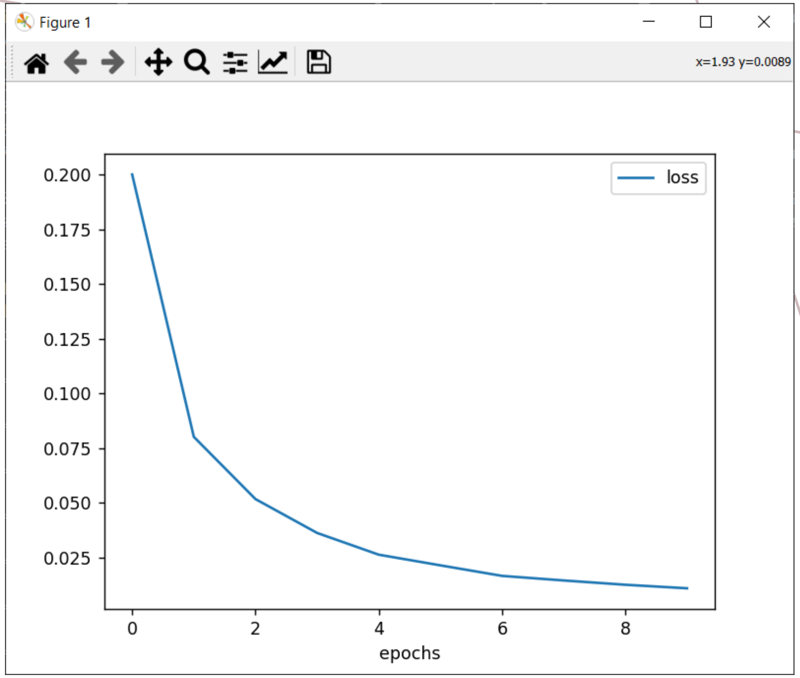
4.) Compiling the Model

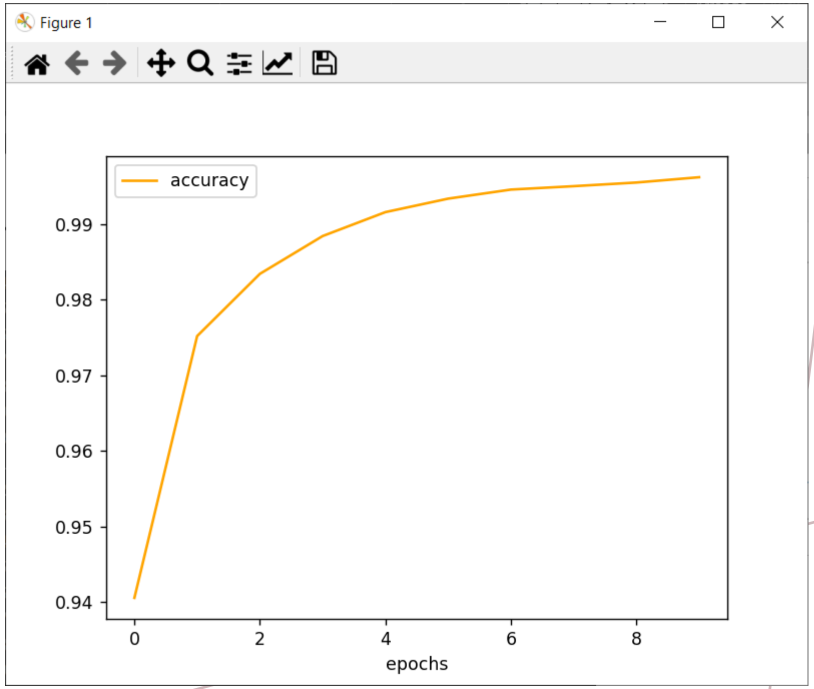
```
# === Part 4 === #
model.compile(
    loss = 'categorical_crossentropy',
    optimizer = 'adam',
    metrics = ['accuracy']
)

```

ADAM (Adaptive Moment Estimation) : Algorithm for optimization technique for gradient descent

5.) Training a Neural Network

```python

# === Part 5 === #
history = model.fit(
    x = train_images,
    y = train_labels,
    epochs = 10
)

# Showing plot for loss
plt.plot(history.history['loss'])
plt.xlabel('epochs')
plt.legend(['loss'])
plt.show()

# Showing plot for accuracy
plt.plot(history.history['accuracy'], color='orange')
plt.xlabel('epochs')
plt.legend(['accuracy'])
plt.show()

```





6.) Evaluating a neural network

```python

# === Part 6 === #
# Call evaluate to find the accuracy on test images
test_loss, test_accuracy = model.evaluate(
    x = test_images,
    y = test_labels
)

print("Test Loss: %.4f"%test_loss)
print("Test Accuracy: %.4f"%test_accuracy)

```

7.) Inference and Prediction

```python
# === Part 7 === #
predicted_probabilities = model.predict(test_images)
predicted_classes = tf.argmax(predicted_probabilities, axis=-1).numpy()

index=11

# Showing image
plt.imshow(test_images[index])

# Printing Probabilities
print("Probabilities predicted for image at index", index)
print(predicted_probabilities[index])

print()

# Printing Predicted Class
print("Probabilities class for image at index", index)
print(predicted_classes[index])

```