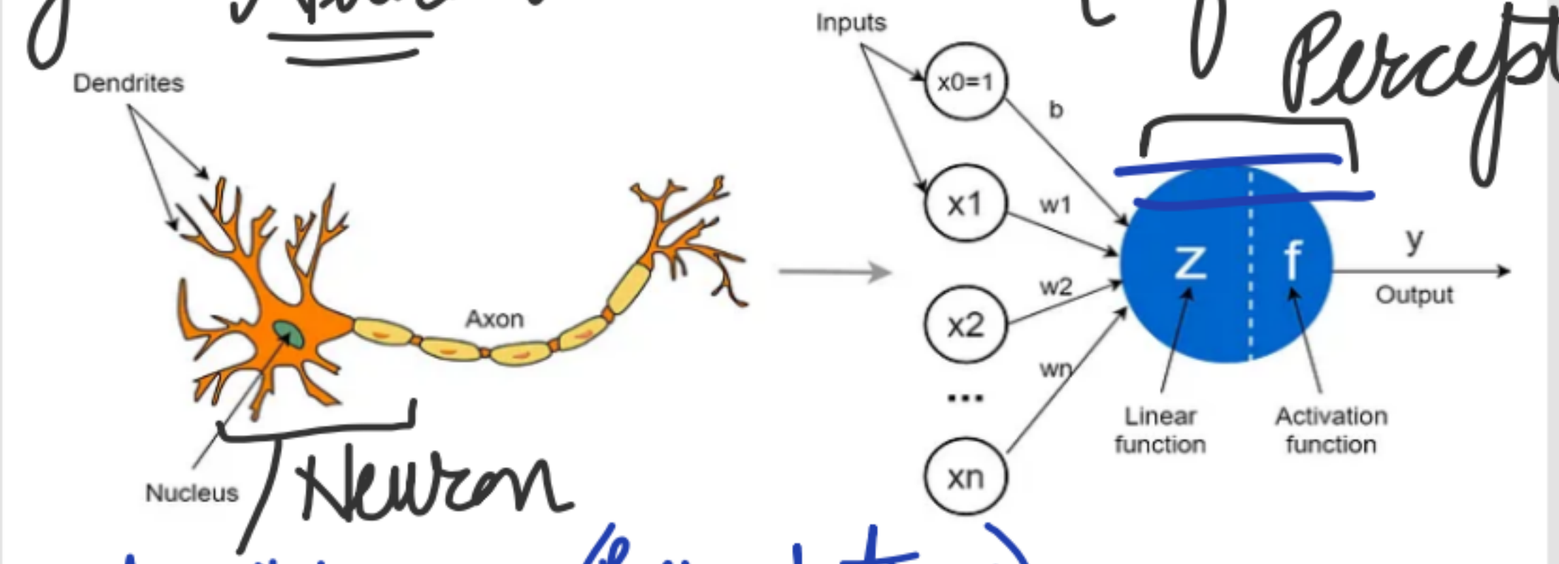
Artificial Neural Network

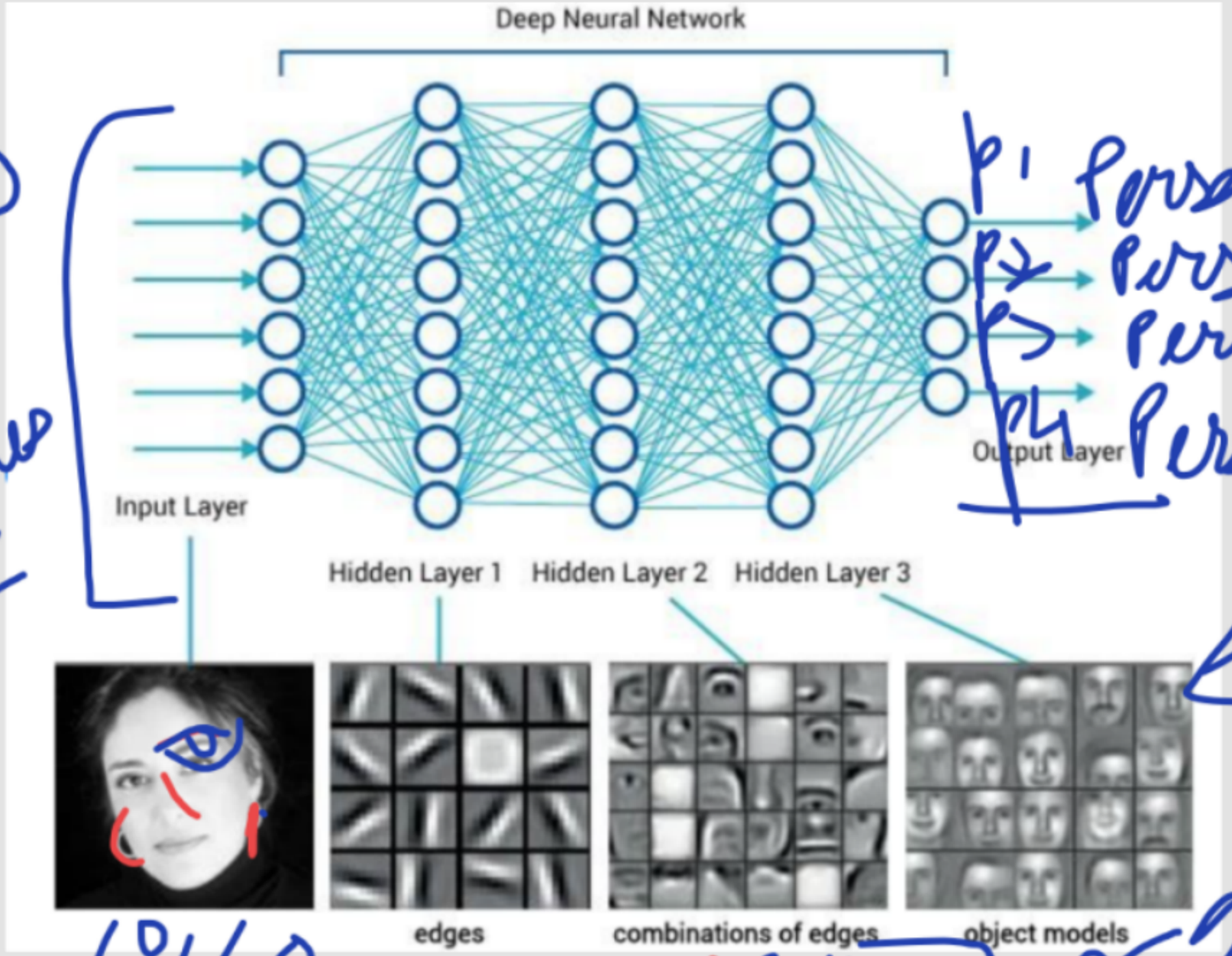
Biological Neuron

Artificial Neuron/ Perceptron



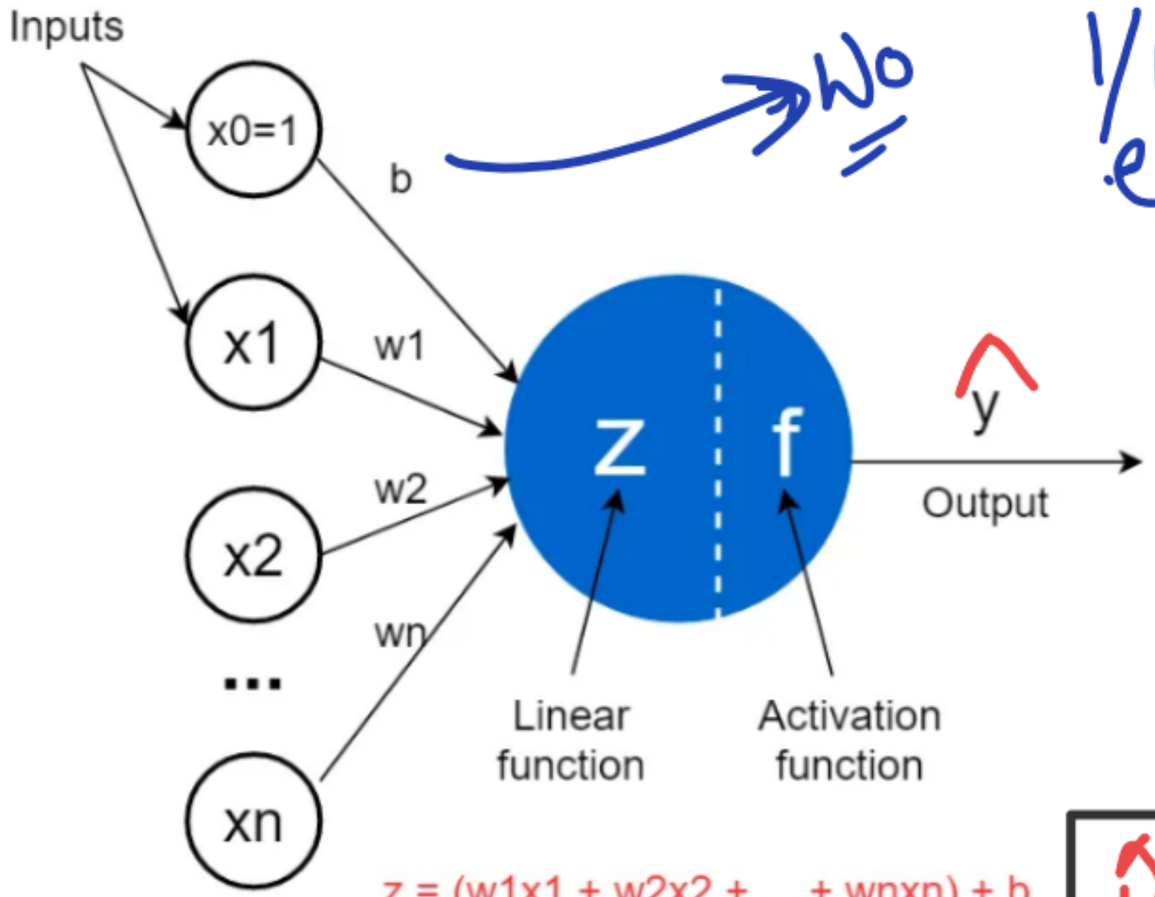
Artificial Neuron (Perceptron) - Mathematical model inspired by biological neuron of Brain

* Dendrite receives electrical signals from the axons of other neurons (Represented in perceptron as numerical values)
* At the synapses between the dendrite and axons, electrical signals are modulated in various amounts (modeled in the perceptron by multiplying each input value by weight)
* An actual neuron fires an output signal only when the total strength of the input signals exceed a certain threshold (modeled using binary thresholded unit)


Deep Neural Network
Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Output Layer
edges
combinations of edges
object models

Perceptron

→ Binary Thresholded neuron
→ Takes real valued I/Ps
→ Calculates linear combination of I/Ps & O/P 1 if linear combination is greater than 0 else 0 or -1



W_0

$$z = \sum_{i=0}^{n} w_i x_i$$

$$= w_0 x_0 + w_1 x_1 + \cdots + w_n$$

$w_0 \to b$, $x_0 \to 1$

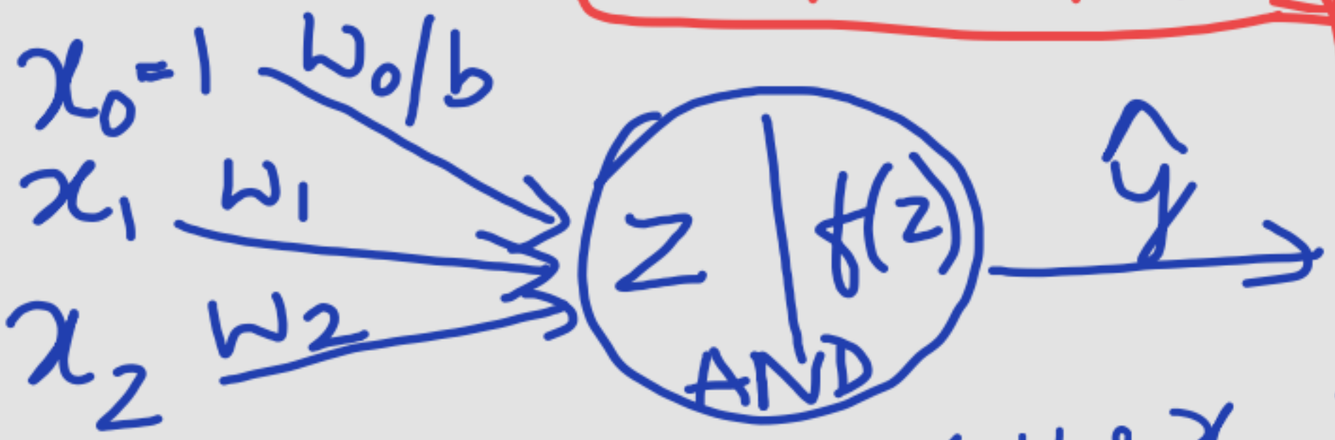
$$\hat{y} = f(z) = \begin{cases} +1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1/0 & \text{otherwise} \end{cases}$$

Implementing Boolean functions using single perceptron

① AND

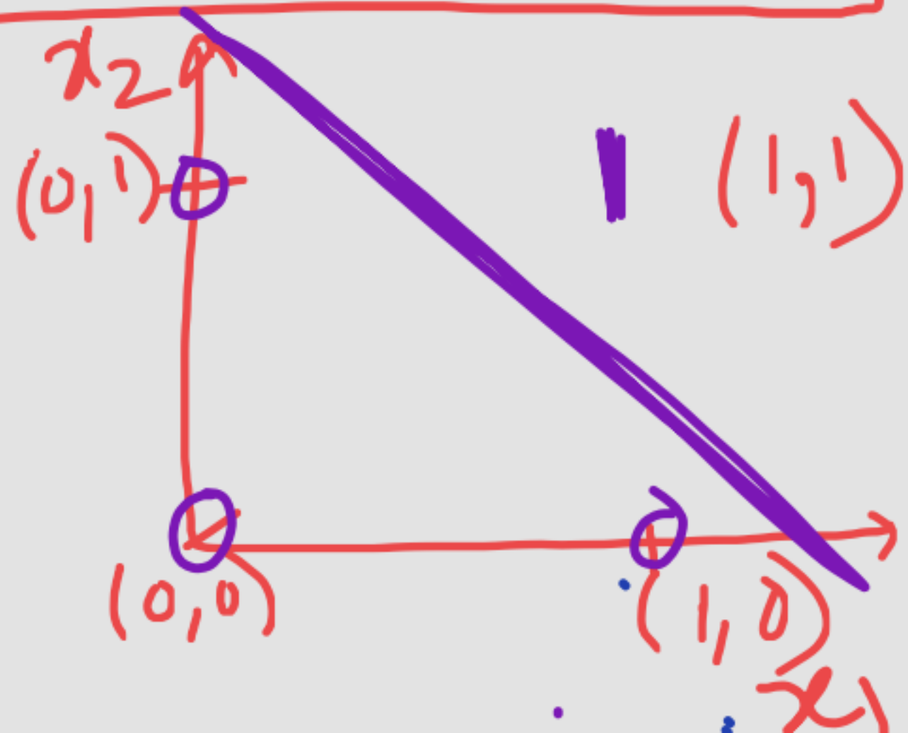
x_1	x_2	y	$z = w_0 + w_1 x_1 + w_2 x_2$	$\hat{y} / f(z)$
0	0	0 ✓	$z = -0.5$	0 ✓
0	1	0 ✓	$z = -0.5 + 0.4$	0 ✓
1	0	0	$z = -0.5 + 0.4$	0 ✓
1	1	1 ✓	$z = 0.3$	1 ✓

$$z = 0.5 + 0.1 x_1 + 0.1 x_2$$



$$\hat{y} = \begin{cases} +1 & \text{if } \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} > 0 \\ 0 & \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} \leq 0 \end{cases}$$

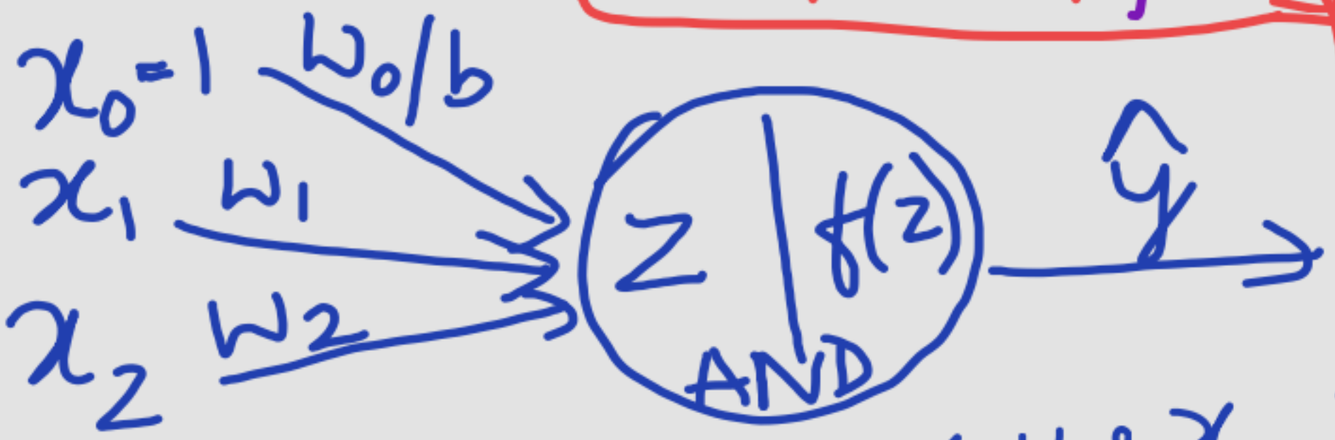
$f(z) =$



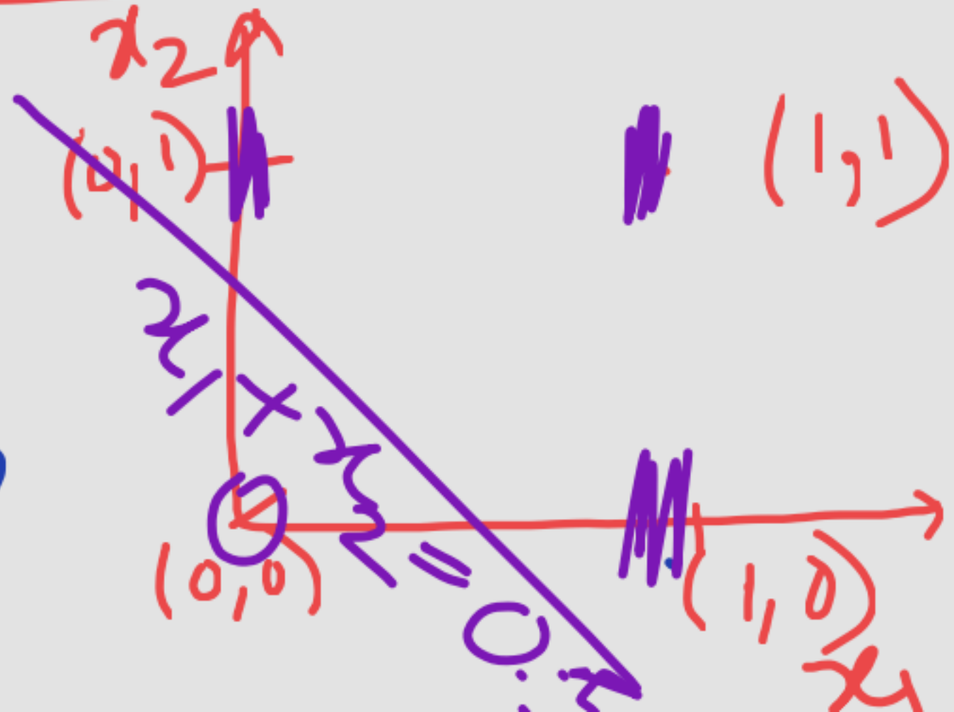
② OR

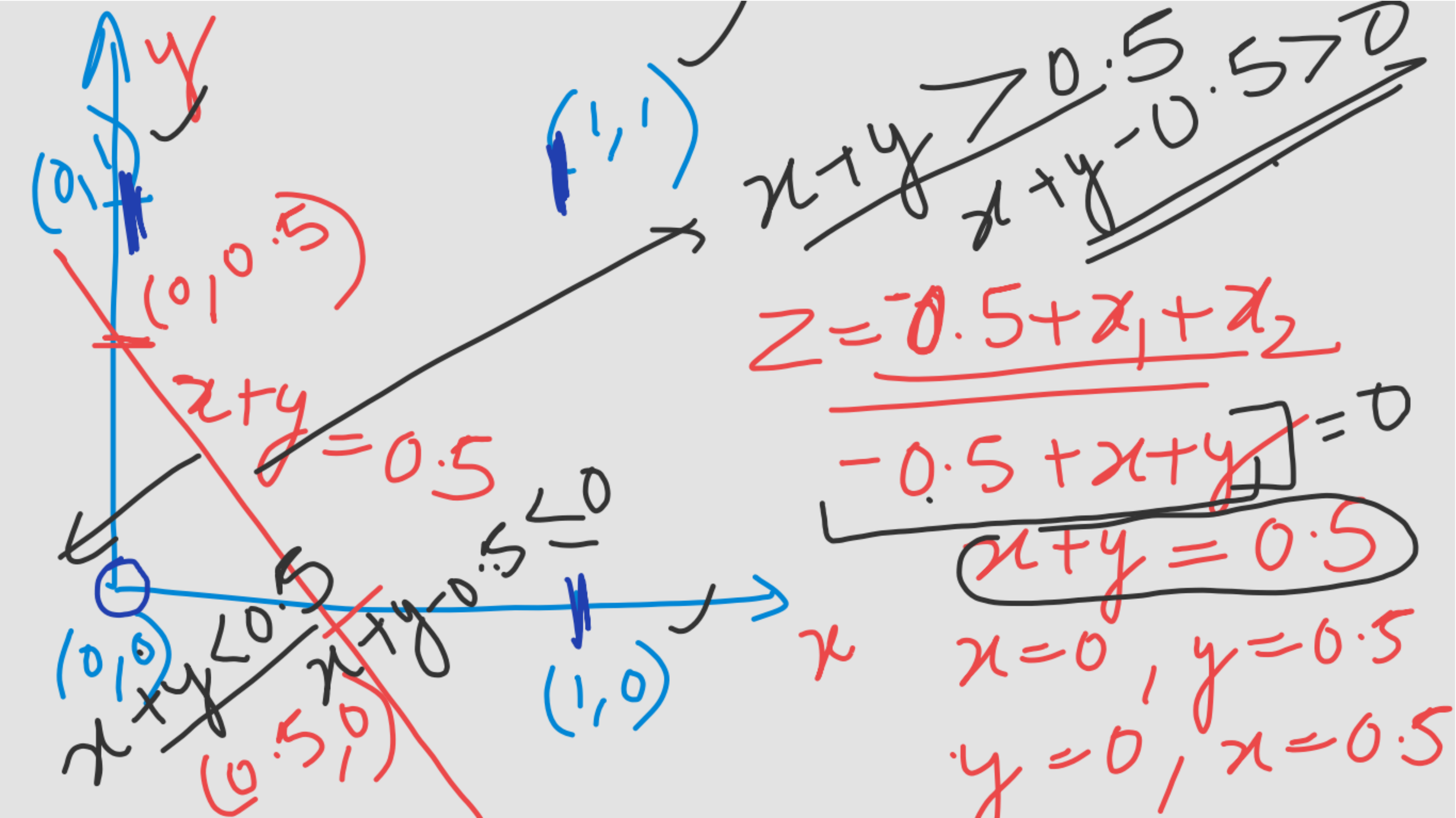
x_1	x_2	y	$z = w_0 + w_1 x_1 + w_2 x_2$	$\hat{y} / f(z)$
0	0	0 ✓	$z = -0.5$	0 ✓
0	1	1 ✓	$z = 0.5$	1 ✓
1	0	1	$z = 0.5$	1 ✓
1	1	1	$z = 1.5$	1 ✓

$z = 0.5 + 1 x_1 + 1 x_2$



$$\hat{y} = f(z) = \begin{cases} +1 & \text{if } \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} > 0 \\ 0 & \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} \leq 0 \end{cases}$$

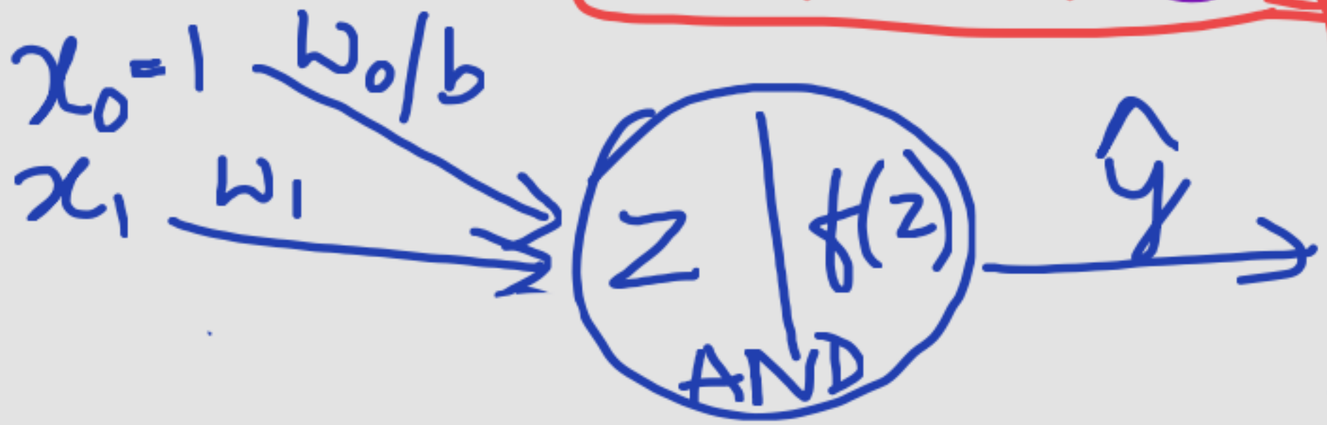



y
x
(0,1)
(1,1)
(0,0.5)
(0.5,0)
(1,0)
(0,0)
x+y=0.5
x+y<0.5
x+y>0.5
x+y-0.5>0
x+y≥0.5
Z=-0.5+x1+x2
-0.5+x+y=0
x+y=0.5
x=0, y=0.5
y=0, x=0.5

③ NOT

x_1	y	$z = w_0 + w_1 x_1$	$g / b(z)$
0	1 ✓	$z = 1$	1 ✓
1	0	$z = -0.5$	0

$z = 1 + -1.5 x_1$



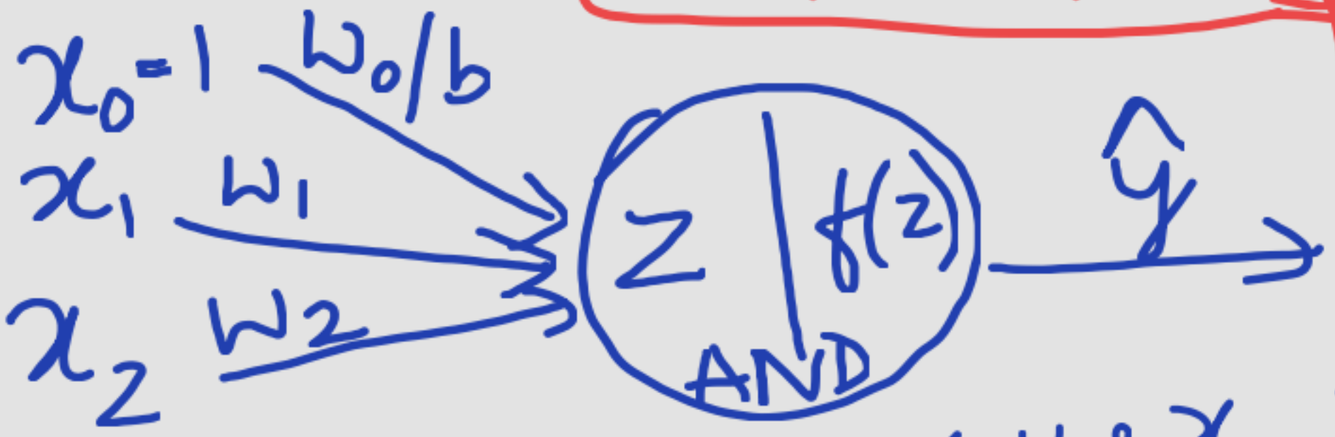
$$\hat{y} = \begin{cases} +1 \text{ if } \underbrace{w_0 + w_1 x_1}_{z} > 0 \\ 0 \quad \underbrace{w_0 + w_1 x_1}_{z} \leq 0 \end{cases}$$

$$f(z) = \begin{cases} \\ \end{cases}$$

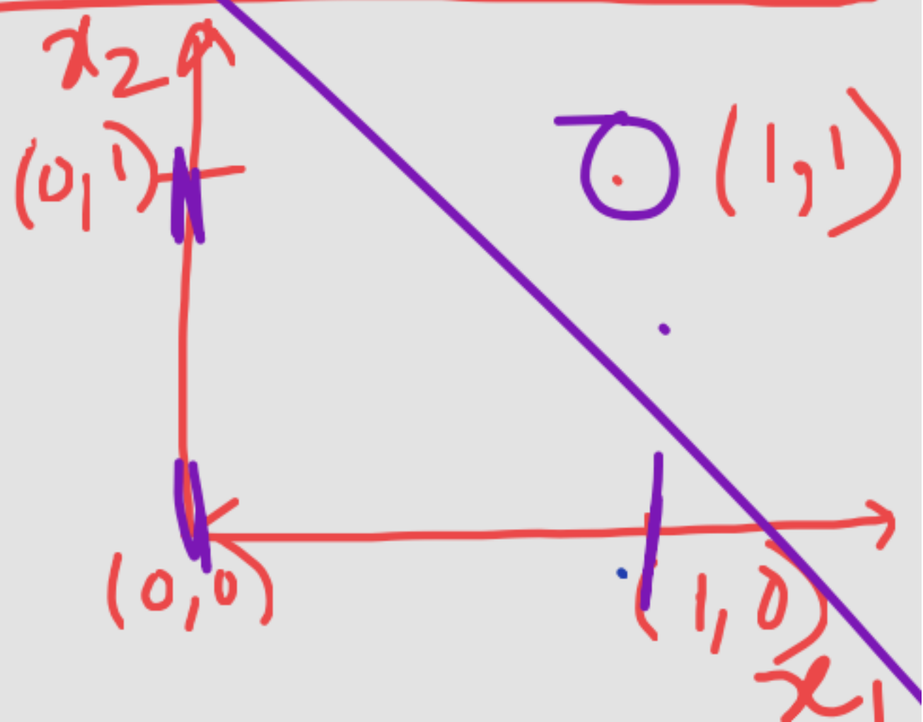
④ NAND

x_1	x_2	y	$z = w_0 + w_1 x_1 + w_2 x_2$	$g / h(z)$
0	0	1 ✓	$z = 0.5$	1
0	1	1	$z = 0.1$	1
1	0	1	$z = 0.1$	1
1	1	0 ✓	$z = -0.3$	0 ✓

$z = 0.5 + (-0.9)x_1 + (-0.4)x_2$



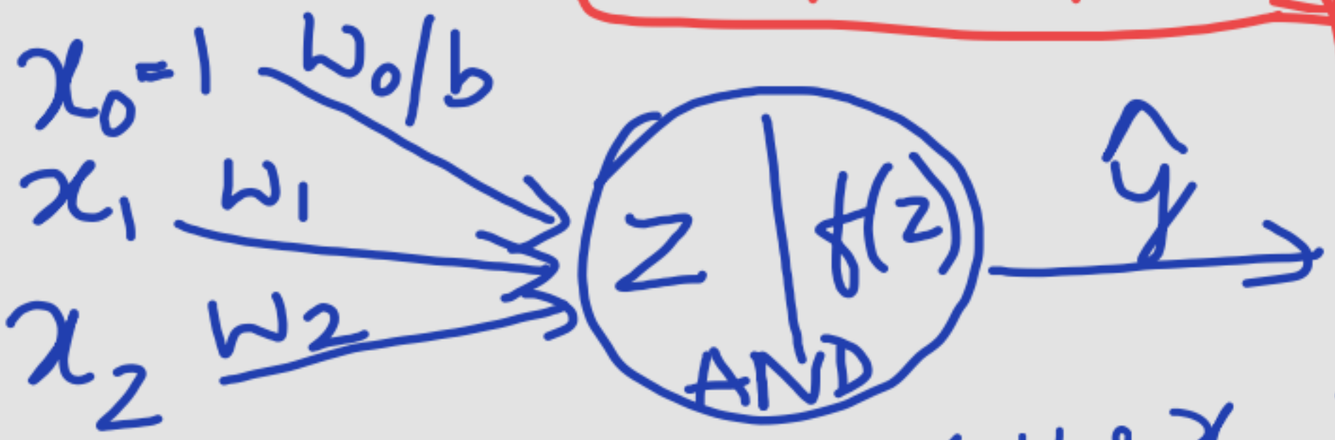
$$\hat{y} = h(z) = \begin{cases} +1 & \text{if } \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} > 0 \\ 0 & \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} \leq 0 \end{cases}$$



⑤ NOR

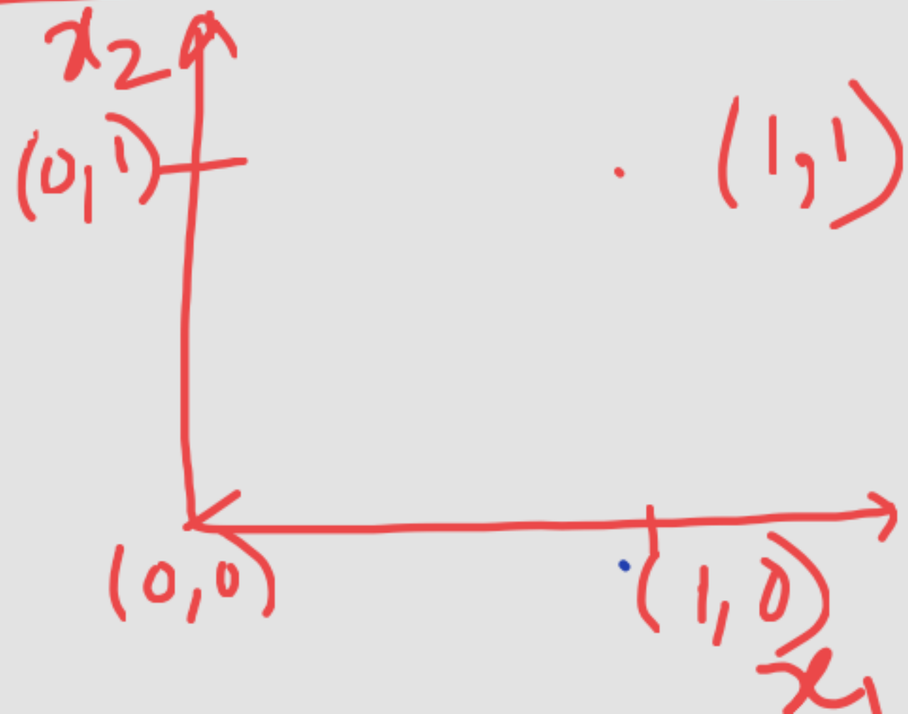
x_1	x_2	y	$z = w_0 + w_1 x_1 + w_2 x_2$	$\hat{y} / f(z)$
0	0			
0	1			
1	0			
1	1			

$z = \_\_ + \_\_ x_1 + \_\_ x_2$



$$\hat{y} = \begin{cases} +1 & \text{if } \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} > 0 \\ 0 & \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} \leq 0 \end{cases}$$

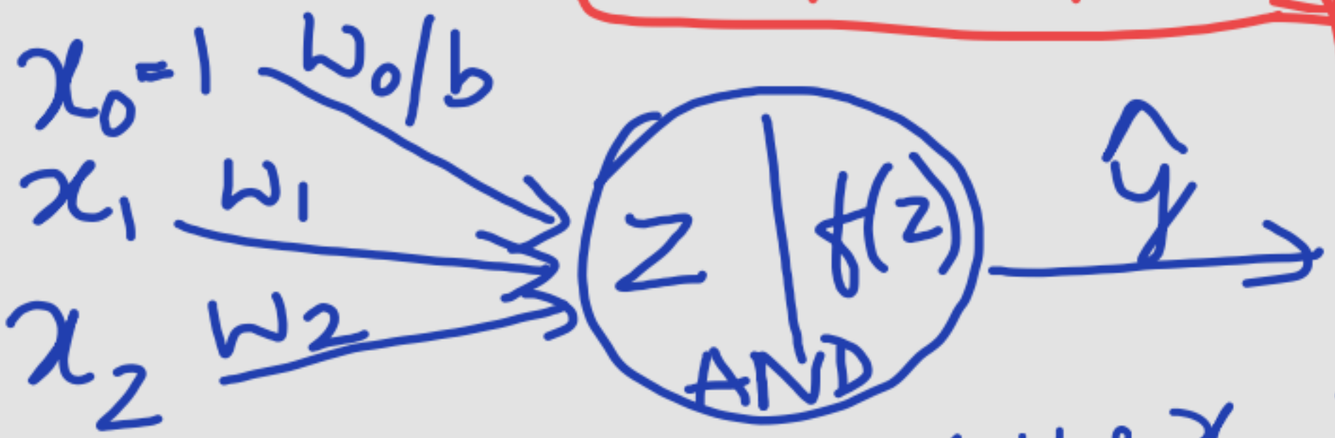
$f(z) =$



(b) XNOR

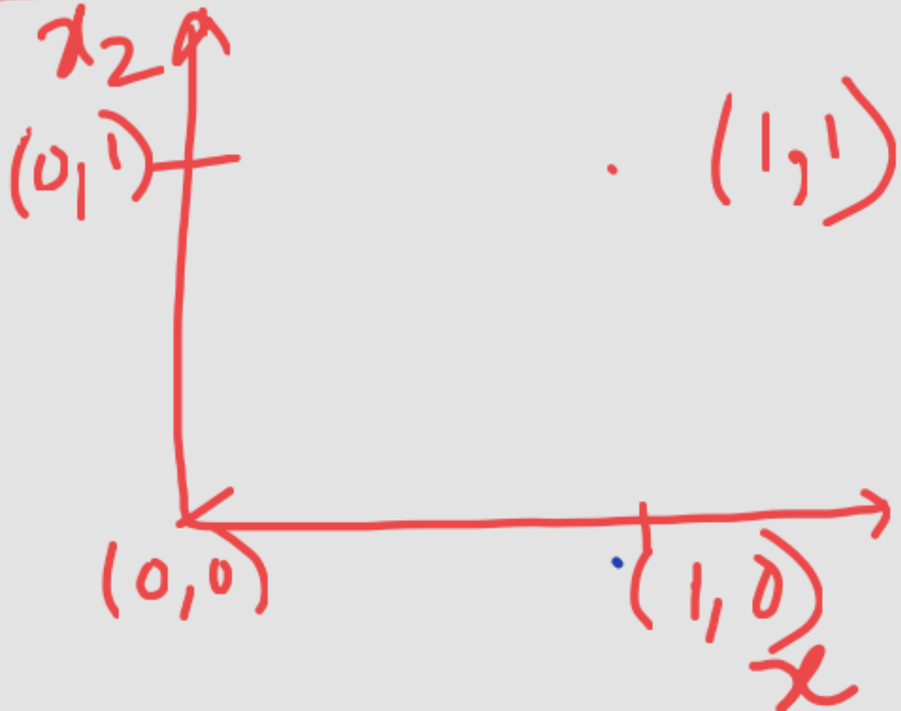
x_1	x_2	y	$z = w_0 + w_1 x_1 + w_2 x_2$	$\hat{y} / f(z)$
0	0			
0	1			
1	0			
1	1			

$z = \_\_ + \_\_ x_1 + \_\_ x_2$



$$\hat{y} = \begin{cases} +1 & \text{if } \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} > 0 \\ 0 & \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} \leq 0 \end{cases}$$

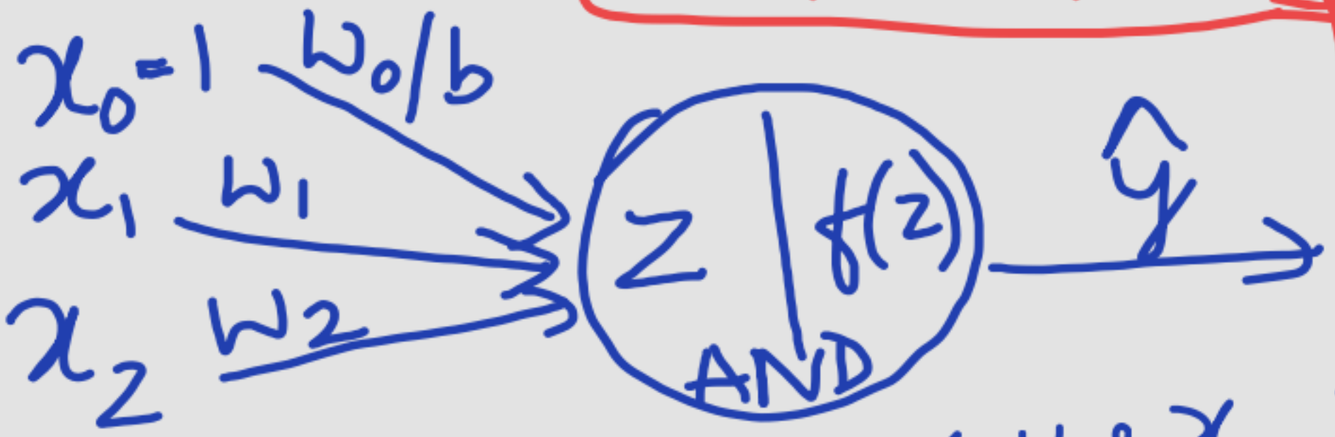
$$f(z) = \begin{cases} \\ \\ \end{cases}$$



(7) X OR

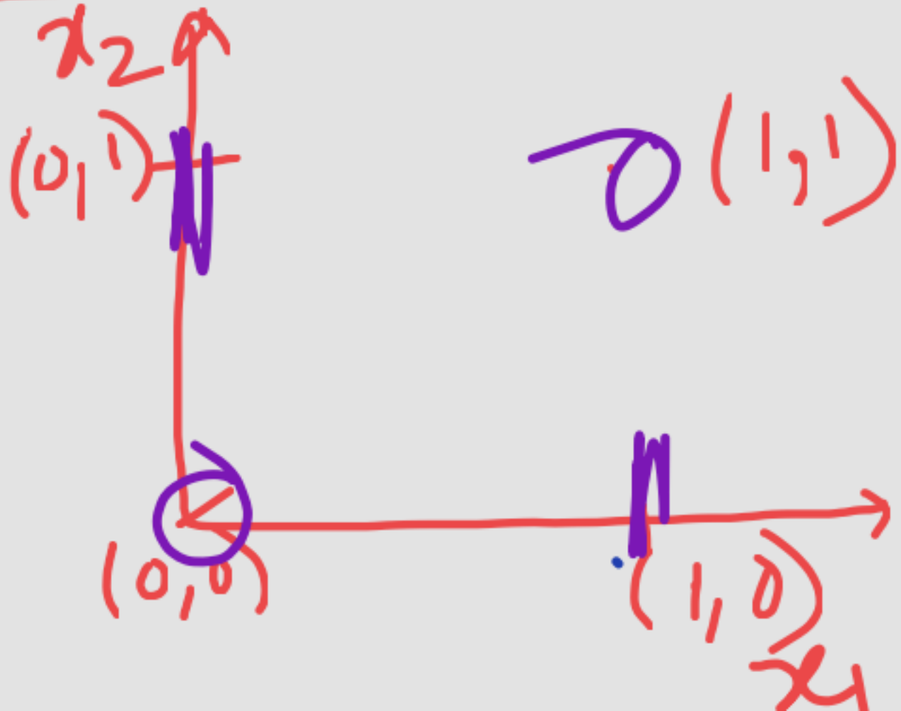
x_1	x_2	y	$z = w_0 + w_1 x_1 + w_2 x_2$	$g / b(z)$
0	0	0		
0	1	1		
1	0	1		
1	1	0		

$z = \_\_ + \_\_ x_1 + \_\_ x_2$



$$\hat{y} = \begin{cases} +1 & \text{if } \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} > 0 \\ 0 & \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{z} \leq 0 \end{cases}$$

$b(z) =$



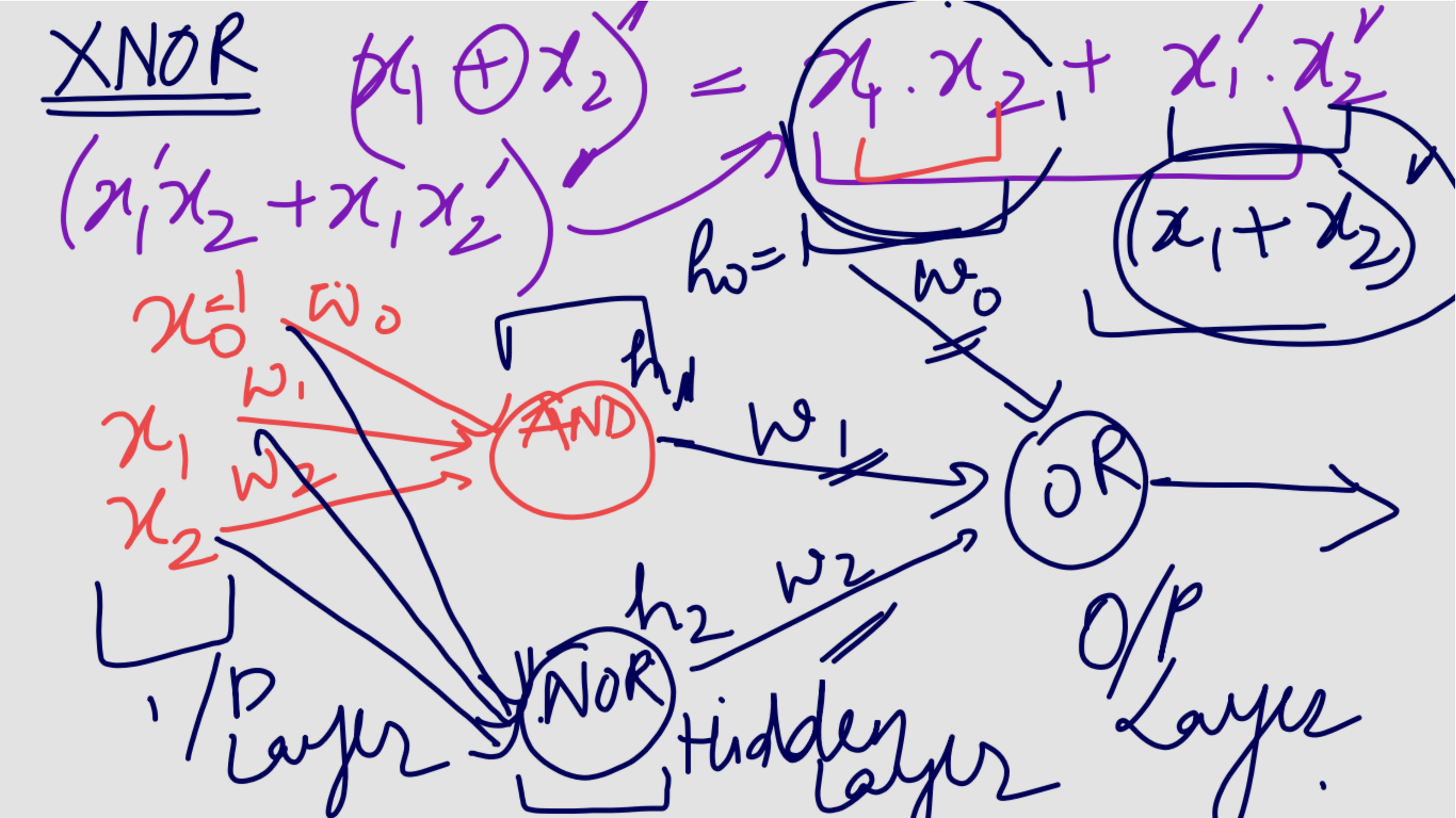
NOTE

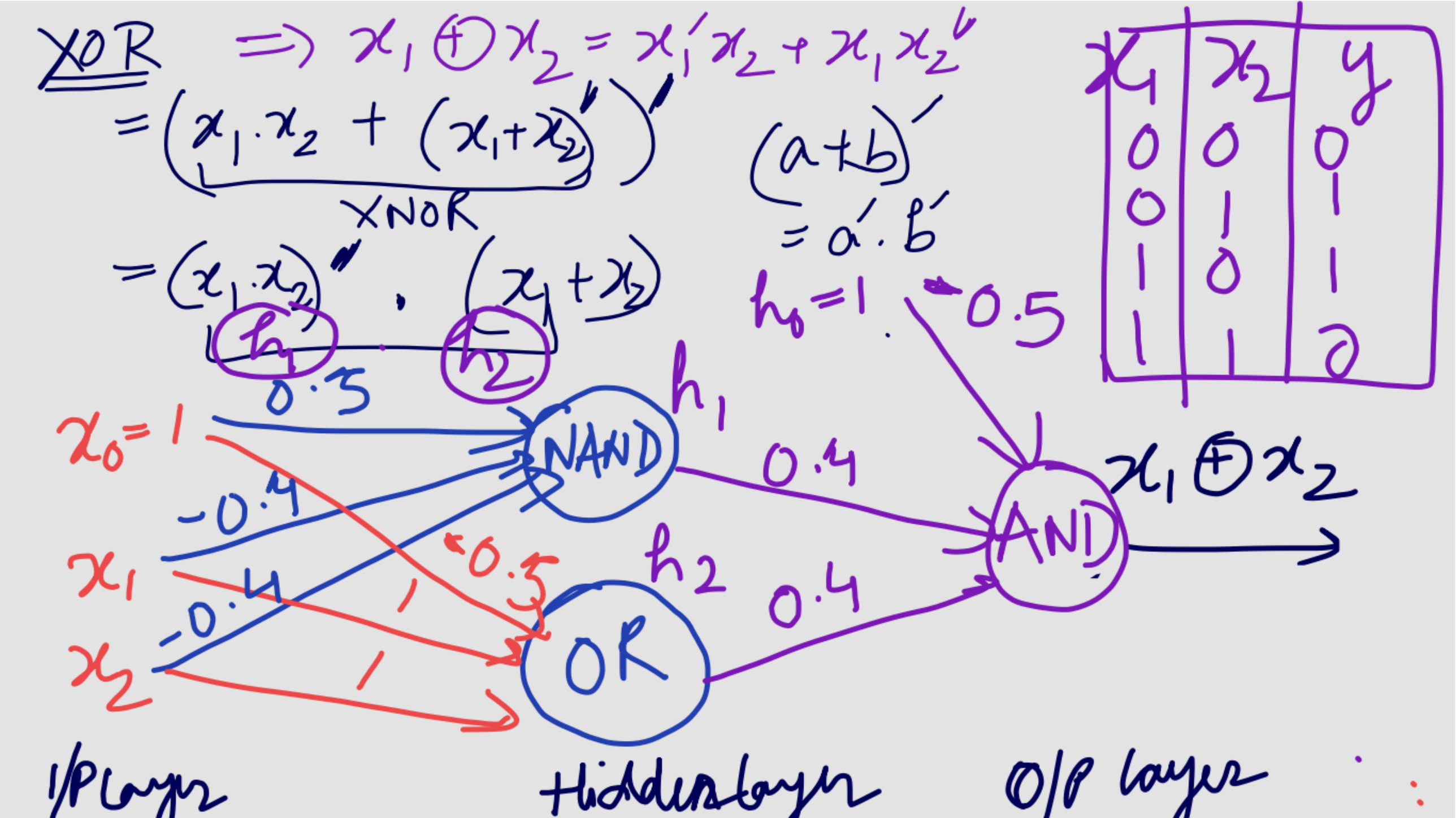
→ If data is linearly Separable, it can be represented by a Single perceptron unit

→ Since in XOR & XNOR datasets, points are not linearly separable, so a multilayer perceptron is required.

OR

Single perceptron can only find decision boundary for data which is linearly separable.


XNOR
$(x_1 \oplus x_2)'$ = $x_1 \cdot x_2 + x_1' \cdot x_2'$
$(x_1'x_2 + x_1x_2')$
$x_1 + x_2$
$h_0 = 1$
w_0
$x_0 = 1$
w_0
w_1
x_1
w_2
x_2
AND
h_1
w_1
OR
h_2
w_2
NOR
I/P Layer
Hidden Layer
O/P Layer


XOR
$\Rightarrow x_1 \oplus x_2 = x_1' x_2 + x_1 x_2'$
$= (x_1 \cdot x_2 + (x_1 + x_2)')'$
XNOR
$= (x_1 \cdot x_2)'' \cdot (x_1 + x_2)$
h_1
h_2
$(a+b)' = a' \cdot b'$
$h_0 = 1$
0.5
$x_0 = 1$
x_1
x_2
NAND
OR
AND
h_1
h_2
0.4
-0.4
-0.4
0.5
1
1
$x_1 \oplus x_2$
x_1 x_2 y
0 0 0
0 1 1
1 0 1
1 1 0
I/P layer
Hidden layer
O/P layer


h_1
h_2
0.5
$x_0 = 1$
NAND
h_1
0.4
$h_0 = 1$
0.5
$y =$
$x_1 \oplus x_2$
-0.4
x_1
-0.4
1
0.5
h_2
0.4
AND
x_2
1
OR
NAND
OR
I/P layer
hidden layer
O/P layer
AND
x_1
x_2
y
h_1
h_2
$0.4 - 0.5 = -0.1$
$= 0$

Algorithms to learn weights that causes perceptron to produce Correct Output

1. Perceptron rule
2. Delta rule → Gradient Descent Rule

Perceptron Training Rule

→ Neuron/Computing unit is thresholded perceptron

Initialize wts to random value

Until Convergence

For every training instance i

For each weight j

$$\Delta w_j = \alpha (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

$$w_j = w_j + \Delta w_j$$

Note: $\hat{y}^{(i)} = \begin{cases} 1 & \sum_{j=0} w_j x_j > 0 \\ -1/0 & \text{otherwise} \end{cases}$

Delta Rule

→ Neuron/computing unit is linear o/p unit

Initialize wts to random value

Until Convergence

For each weight j

$$\Delta w_j = \alpha \sum_i (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

$$w_j = w_j + \Delta w_j$$

Note: $\hat{y}^{(i)} = \sum_{j=0} w_j x_j$

$w_0, w_1, \ldots$

Delta Rule / Gradient Descent

until Convergence

$$\Delta w_j = \alpha \sum_{i=0}^{n} \left(\hat{y}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

Rule

$$w_j = w_j^{\circ} - \Delta w_j$$

$$\hat{y}^{(i)} = \sum_{j=0}^{d} w_j x_j$$

$b = w_0$

$x_0 = 1$

Perceptron Training Rule

→ Modify weights via incremental updates

→ Train on thresholded outputs

→ Driven by binary differences b/w predicted & Corrected outputs.

* Guaranteed to converge after finite number of iterations to hypothesis that perfectly classifies all the training data provided data is linearly separable.

Delta Rule

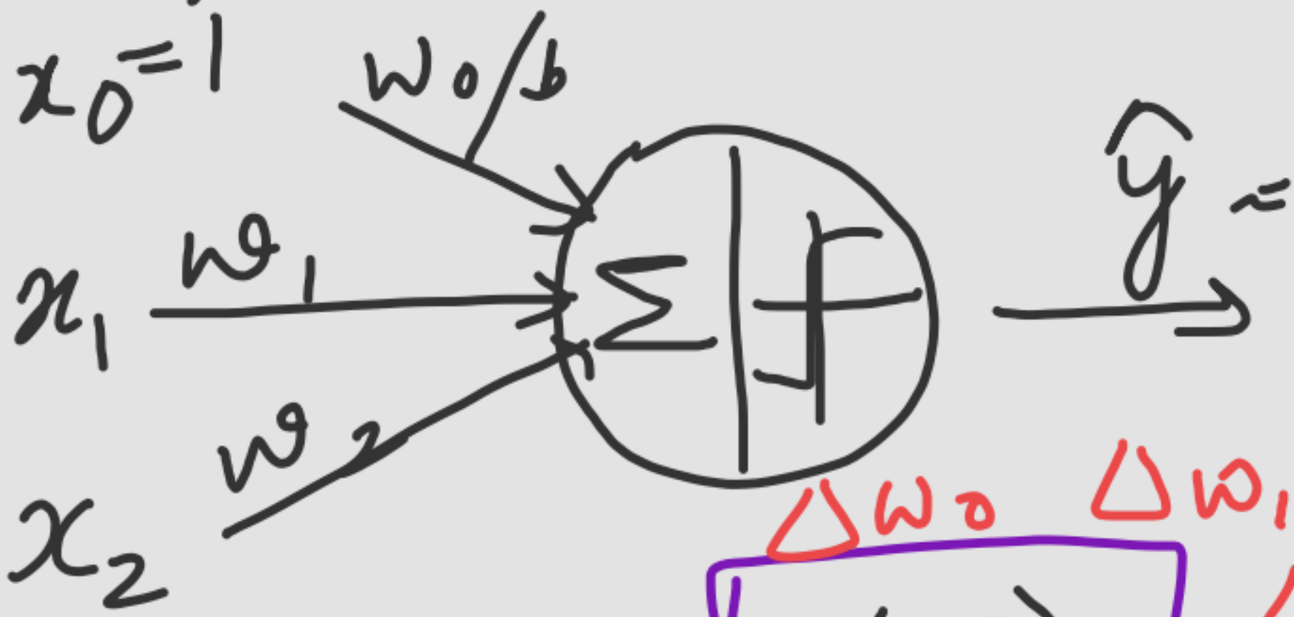
→ Modify weights via gradient descent

→ Train on unthresholded outputs

→ Driven by continuous differences b/w predicted & Corrected outputs

* Converges towards minimum error hypothesis, requiring large amount of time, regardless of whether data is linearly separable or not.

Ques Train the neural N/W using perceptron rule.



$$\hat{y} = \begin{cases} +1 & \text{if } \sum_j w_j x_j > 0 \\ -1/0 & \text{otherwise} \end{cases}$$

$$w_j = w_j + \underbrace{\alpha (y - \hat{y}) x_j}_{\Delta w_j}$$

Δw_0 Δw_1 Δw_2

Note: No change if $\Delta w_j = 0$

$\alpha = 0.5$

OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

x_0	x_1	x_2	w_0	w_1	w_2	y	$\hat{y} = \text{Sign}(\sum w_i x_i)$	Δw_0	Δw_1	Δw_2	Updated wt w_0	w_1	w_2
1	0	0	0	0	0	0	$z=0$, $\hat{y}=0$	0	0	0	$=w_0+\Delta w_0$	$=w_1+\Delta w_1$	$=w_2+\Delta w_2$
											0	0	0
1	0	1	0	0	0	1	$z=0$ $\hat{y}=0$	0.5	0	0.5	0.5	0	0.5

$$\Delta w_0 = \alpha (y - \hat{y}) x_0 \rightarrow 1$$

$0.5\ (1-0)$

$$\Delta w_1 = \alpha (y - \hat{y}) x_1 \rightarrow 0$$

$$\Delta w_2 = \alpha (y - \hat{y}) x_2 \rightarrow 1$$

$0.5\ (1-0)$

$\alpha = 0.5$

x_0	x_1	x_2	w_0	w_1	w_2	y	$\hat{y} = \text{Sign}(\Sigma w_i x_i)$	Δw_0	Δw_1	Δw_2	Updated wt w_0	w_1	w_2
1	0	0	0	0	0	0	$z=0$, $\hat{y}=0$	0	0	0	$=w_0+\Delta w_0$	$=w_1+\Delta w_1$	$=w_2+\Delta w_2$
1	0	1	0	0	0	1	$z=0$ $\hat{y}=0$	0.5	0	0.5	0.5	0	0.5
1	1	0	0.5	0	0.5	1	$z=0.5$, $\hat{y}=1$	—— No chang ——					
1	1	1	0.5	0	0.5	1	$z=1$, $\hat{y}=1$	—— No chang ——					
1	0	0	0.5	0	0.5	0	$z=0.5$, $\hat{y}=1$	-0.5	0	0	0	0	0.5

$$\Delta w_j = \alpha (y - \hat{y}) x_j = 0.5(0-1) \cdot L \ldots$$

x_0	x_1	x_2	w_0	w_1	w_2	y	$\hat{y} = \text{Sign}(w_0x_0 + w_1x_1 + w_2x_2)$	Δw_0	Δw_1	Δw_2
1	0	0	0	0	0	0	Sign(0) = 0	0	0	0
1	0	1	0	0	0	1	Sign(0) = 0	0.5	0	0.5
1	1	0	0.5	0	0.5	1	Sign(0.5) = 1	0	0	0
1	1	1	0.5	0	0.5	1	Sign(1.5) = 1	0	0	0

New Update

w_0	w_1	w_2
0	0	0
0.5	0	0.5

(Change) (Change) (No Change)

Iteration (2)

x_0	x_1	x_2	w_0	w_1	w_2	y	$\hat{y} = \text{Sign}(w_0x_0 + w_1x_1 + w_2x_2)$	Δw_0	Δw_1	Δw_2
1	0	0	0.5	0	0.5	0	Sign(0.5) = 1	-0.5	0	0
1	0	1	0	0	0.5	1	Sign(0.5) = 1	0	0	0
1	1	0	0	0	0.5	1	Sign(0) = 0	0.5	0.5	0
1	1	1	0.5	0.5	0.5	1	Sign(1.5) = 1	0	0	0

New Update

w_0	w_1	w_2
0	0	0.5
0	0	0.5
0.5	0.5	0.5
0.5	0.5	0.5

It (3)

x_0	x_1	x_2	w_0	w_1	w_2	y	$\hat{y} = \text{Sign}(\sum w_i x_i)$	Δw_0	Δw_1	Δw_2
1	0	0	0.5	0.5	0.5	0	Sign(0.5) = 1	-0.5	0	0
1	0	1	0	0.5	0.5	1	Sign(0.5) = 1			
1	1	0				1	= 1			
1	1	1				1	= 1			

UPDATED WT

w_0	w_1	w_2
0	0.5	0.5

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \tag{T4.1}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i \tag{T4.2}$$