-Artificial Neural Network-

Chapter 3 Perceptron

朝陽科技大學

資訊管理系

李麗華 教授

Outline

- **History**
- **Structure**
- **Learning Process**
- **Recall Process**
- **Solving OR Problem**
- **Solving AND Problem**
- **Solving XOR Problem**

History of Perceptron Model

- In 1957, Rosenblatt and several other researchers developed perceptron, which used the similar network as proposed by McCulloch and the learning rule for training network to solve pattern recognition problem.

(*) But, this model was later criticized by Minsky (1969) who proved that it cannot solve the XOR problem.
In 1986, Rumelhart et al. proposed the Generalized Delta rule and the XOR problem were then solved.

Structure

The network structure includes:

Input layer: input variables with binary type information.

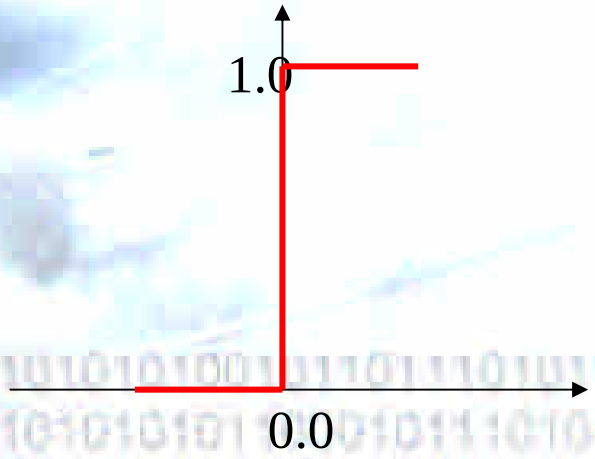
The number of node depends on the problem dimension.

Processing node: uses linear activation function, i.e.,

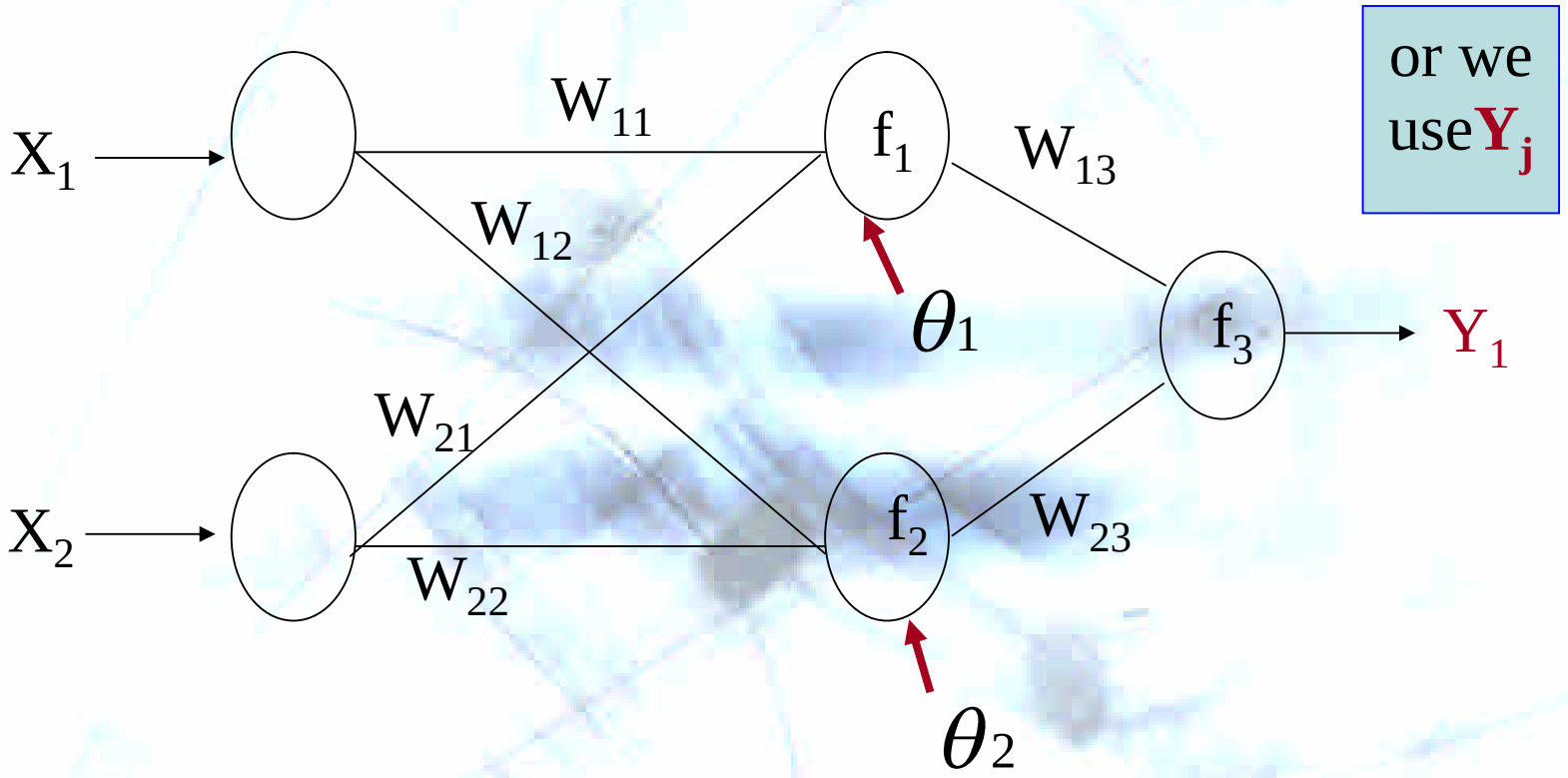
$net_j = I_j^n$, and the Bias θ_j is used.

Output layer: the computed results is generated through transfer function.

Transfer Function: discrete type, i.e., step function.



Perceptron network structure



※The sub-index is designed based on your numbering rule.

The training process

The training steps: (One layer at a time)

1. **Choose the network layer, nodes, and connections.**
2. **Randomly assign weights: W_{ij} & bias: θ_j**
3. **Input training sets X_i (preparing T_j for verification)**
4. **Training computation:**

$$net_j = \sum_i W_{ij} X_i - \theta_j$$

$$Y_j = \begin{cases} 1 & \text{if } net_j > 0 \\ 0 & \text{if } net_j \le 0 \end{cases}$$

The training process

5. Training computation:

If $(T_j - Y_j) \neq 0$

than: $\Delta W_{ij} = \eta (T_j - Y_j) X_i$

$$\Delta \theta_j = -\eta (T_j - Y_j)$$

Update weights and bias :

$$W_{ij} = W_{ij} + \Delta W_{ij}$$

$$new\ \theta_j = \theta_j + \Delta \theta_j$$

6. repeat steps 3 ~step 5 until every input pattern is satisfied as:

$$(T_j - Y_j) == 0$$

The recall process

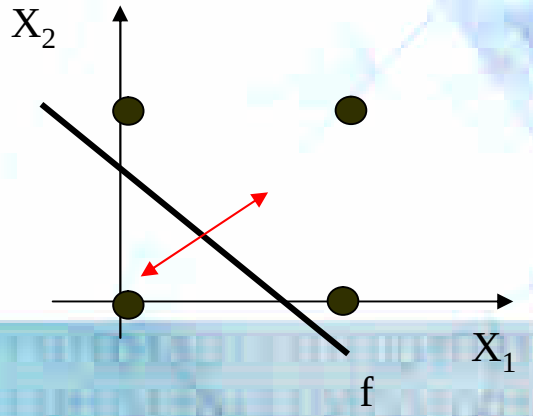
After the network has trained as mentioned above, any input vector X can be send into the Perceptron network to derive the computed output. The ratio of total number of corrected output is treated as the prediction performance of the network.

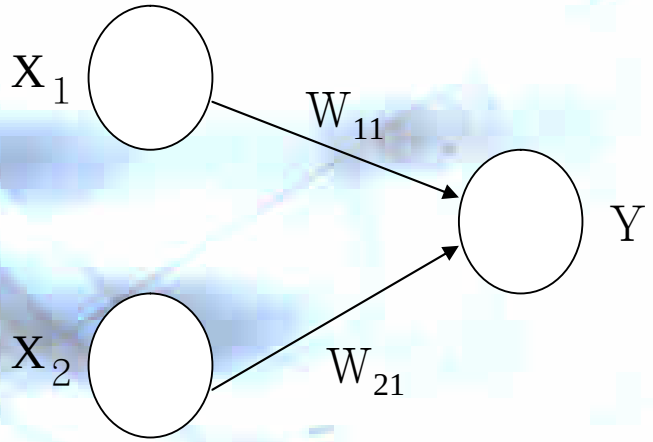
The trained weights, W_{ij}, and the bias, θ_j, is used to derive net_j and, therefore, the output Y_j can be obtained for pattern recognition (or for prediction).

Example: Using Perceptron to Solve the OR problem

- Let the training patterns are used as follow.

X_1	X_2	T
0	0	0
0	1	1
1	0	1
1	1	1

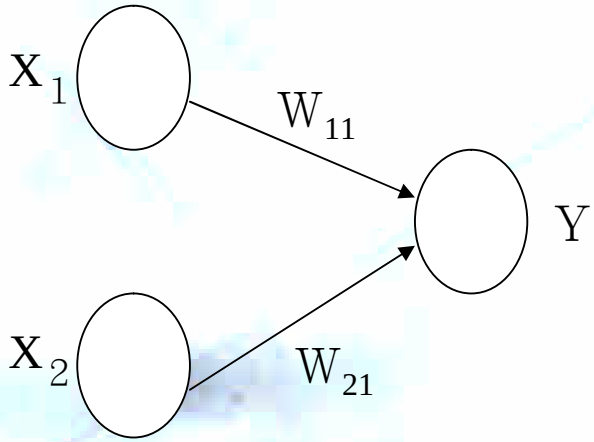




Training process to recognize OR(1)

SOL:

- Let $W_{11}=1$, $W_{21}=0.5$, $\Theta=0.5$, and $\eta=0.1$



- The initial net function is as follows.

  $net = W_{11} \bullet X_1 + W_{21} \bullet X_2 - \Theta$

  $net = 1 \bullet X_{11} + 0.5 \bullet X_{21} - 0.5$ (apply the weights)

- Feed the input pattern into network one by one

  (0,0), net= -0.5, Y=0, (T-Y) = 0 (ˇ) pass

  (0,1), net= 0, Y= 0, (T-Y) =1-Ø=1 (X) update weights

- update weights for pattern (0,1) which is not satisfying the expected output.

  $\Delta W_{11} = (0.1)(1)(0) = 0$, $W_{11} = W_{11+} \Delta W_{11} = 1$

  $\Delta W_{12} = (0.1)(1)(1) = 0.1$, $W_{21} = 0.5 + 0.1 = 0.6$

  $\Delta \Theta = -(0.1)(1) = -0.1$, $\Theta = 0.5 - 0.1 = 0.4$

Training process to recognize OR(2)

- Applying new weights to the net function.

  $net=1 \bullet X_1+0.6 \bullet X_2-0.4$

- Verify the pattern (0,1) to see if it satisfies the expected output.

  (0,1), net= 0.2, Y= 1, (T-Y) =Ø (∨) pass

- Keep feeding the net pattern into network one by one.

  (1,0), net= 0.6, Y=1, (T-Y) =Ø (∨) pass

  (1,1), net= 1.2, Y=1, (T-Y) =Ø (∨) pass

Training process to recognize OR(3)

- Since the first pattern(0,0) has not been testified with the new weights, therefor, we need to feed the first pattern again.

  (0,0), net= -0.4, Y=Ø, (T-Y) =Ø (∨) pass

- Now, all the patterns are satisfied the expected output. Hence, the network is successfully trained for understanding the OR problem(pattern).

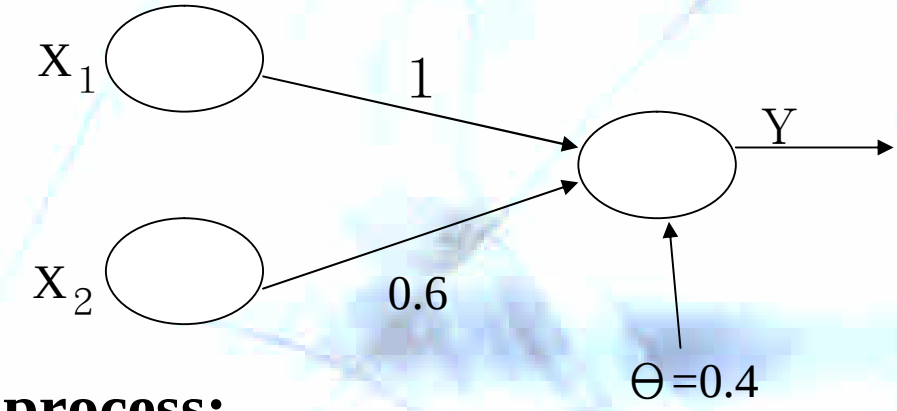
  We now obtained the pattern recognition function for OR pattern based on perceptron model:

  $$\mathbf{net = X_1 + 0.6X_2 - 0.4}$$

(*)This is not the only solution. Other solutions are possible. The various solution is due to the initial weights.

The Trained OR Network & Recall

- The trained network is formed as follow.



- **Recall process:**

  Once the network is trained, we can apply any two element vectors as a pattern and feed the pattern into the network for recognition. For example, we can feed (1,0) into to the network

  (1,0), net= 0.6, **Y=1**

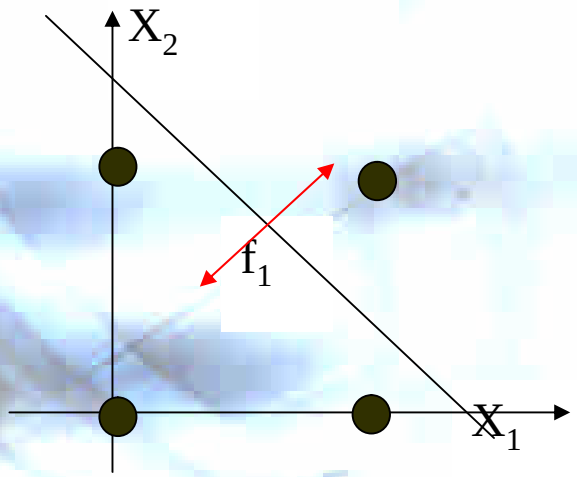
  Therefore, this pattern is recognized as 1.

Now, we say this network is ready for recognizing the OR problem.

Example: Solving the AND problem

- **This is a problem for recognizing the AND pattern**
- Let the training patterns are used as follow

X_1	X_2	T
0	0	0
0	1	0
1	0	0
1	1	1



Training process to recognize AND(1)

- Let W_{11}=0.5, W_{21}=0.5, Θ=1, Let η=0.1
- The initial net function is:

  net $=0.5X_{11}+0.5X_{21}-1$

- Feed the input pattern into network one by one

  (0,0), net=-1, Y=Ø, (T-Y) =Ø (∨) pass

  (0,1), net=-0.5, Y= Ø, (T-Y) =Ø (∨) pass

  (1,0), net=- 0.5, Y= Ø, (T-Y) =Ø (∨) pass

  (1,1), net= Ø, Y= Ø, (T-Y) = 1 (X) update weights

Training process to recognize AND(2)

- update weights for pattern (1,1) which does not satisfying the expected output:

  $\Delta W_{11}=(0,1)(1)(1)= 0.1,$ update $W_{11}=0.6$

  $\Delta W_{21}=(0,1)(1)(1)= 0.1,$ update $W_{21}=0.5+0.1=0.6,$

  $\Delta \Theta=-(0.1)(1)=-0.1,$ update $\Theta=1-0.1=0.9$

- Applying new weights to the net function:

  $net=0.6X_1 + 0.6X_2 - 0.9$

- Verify the pattern (1,1) to see if it satisfies the expected output.

  (1,1), net= 0.3, Y= 1, (T-Y) =Ø ($\vee$) pass

Training process to recognize AND(3)

- Since the previous patterns are not testified with the new weights, feed them again.

  (0,0), net=-0.9, Y=Ø, (T-Y) =Ø (∨) pass

  (0,1), net=-0.3, Y= Ø, (T-Y) =Ø (∨) pass

  (1,0), net=- 0.3, Y= Ø, (T-Y) =Ø (∨) pass

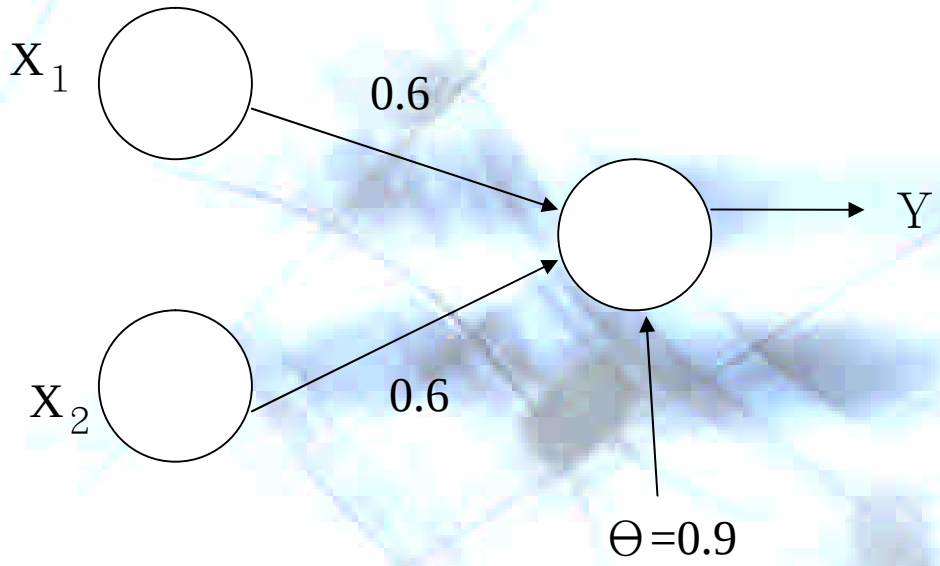
- We can generate the pattern recognition function for OR pattern is:

$$\mathbf{net= 0.6X_1 + 0.6X_2 - 0.9}$$

(*)This is not the only solution. Other solutions are possible. The various solution is due to the initial weights.

The Trained AND Network & Recall

- The trained network is formed as follow:

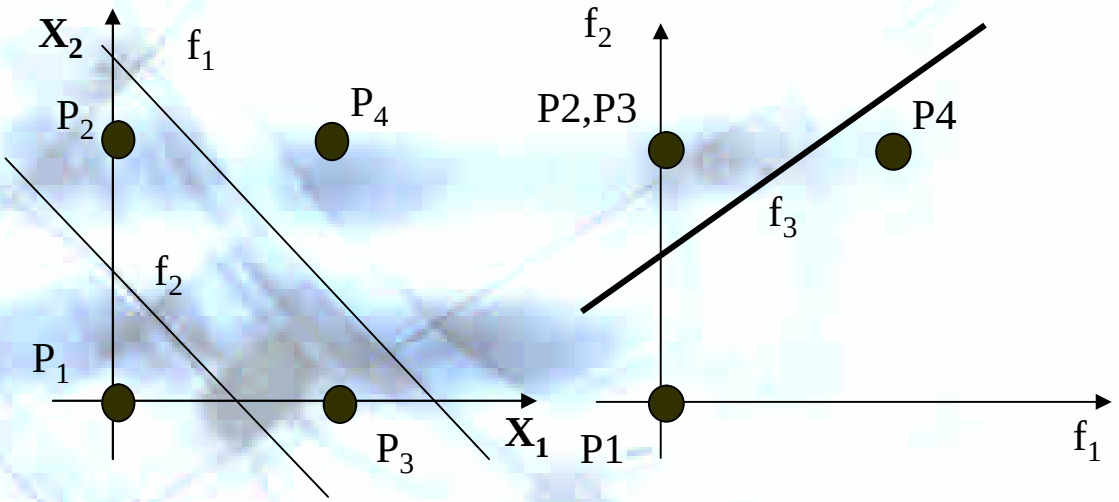


Now you can send any input pattern into this trained AND network.
The computing process will bring you the correct answer.

Example: Solving the XOR problem(1)

- Let the training patterns are used as follow.

X_1	X_2	T
0	0	0
0	1	1
1	0	1
1	1	0

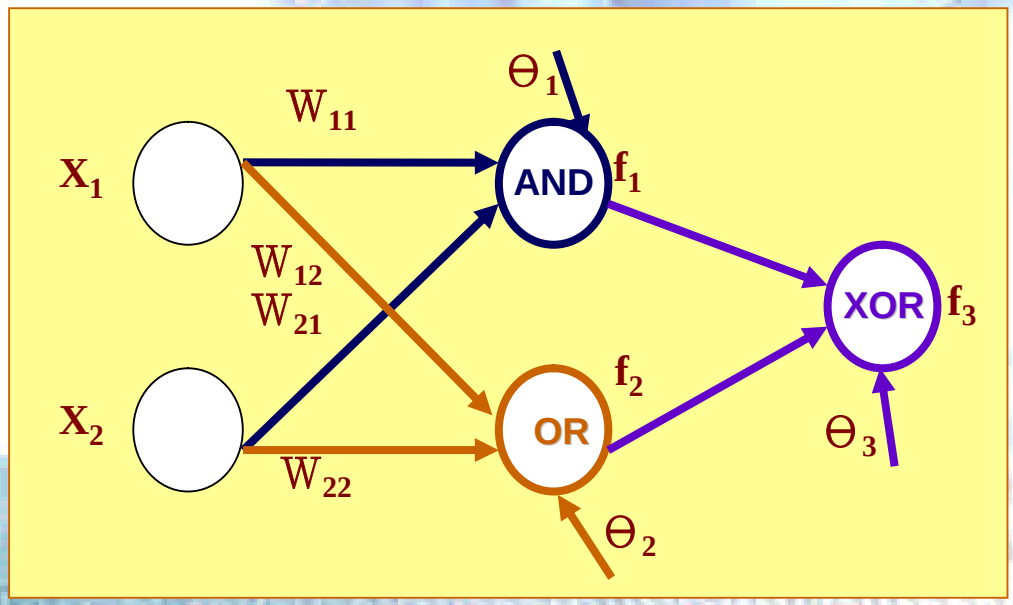


Now, we are run into trouble, because we need at least two lines for recognizing the XOR data patterns. But the network we learned before, i.e., "OR" or "AND" network, cannot recognize this nonlinear problem.

Solving the XOR problem(2)

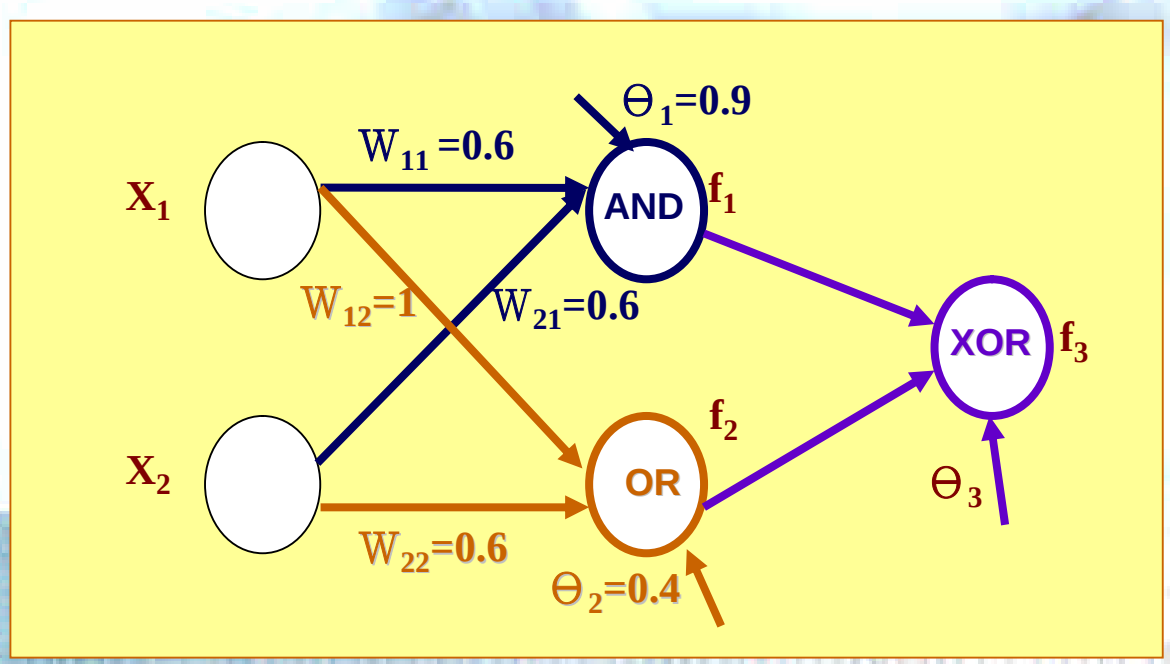
- Minsky did prove the single layer perceptron cannot solve the XOR problem.
- However, perceptron can be extended into multi-layer for solving the higher dimension or multiple function problem.
- Now, we can solve XOR problem by using

  [**AND net(f_1)** + **OR net(f_2)**] ➔ **net(f_1&f_2)** ➔ **y**



Solving the XOR problem(3)

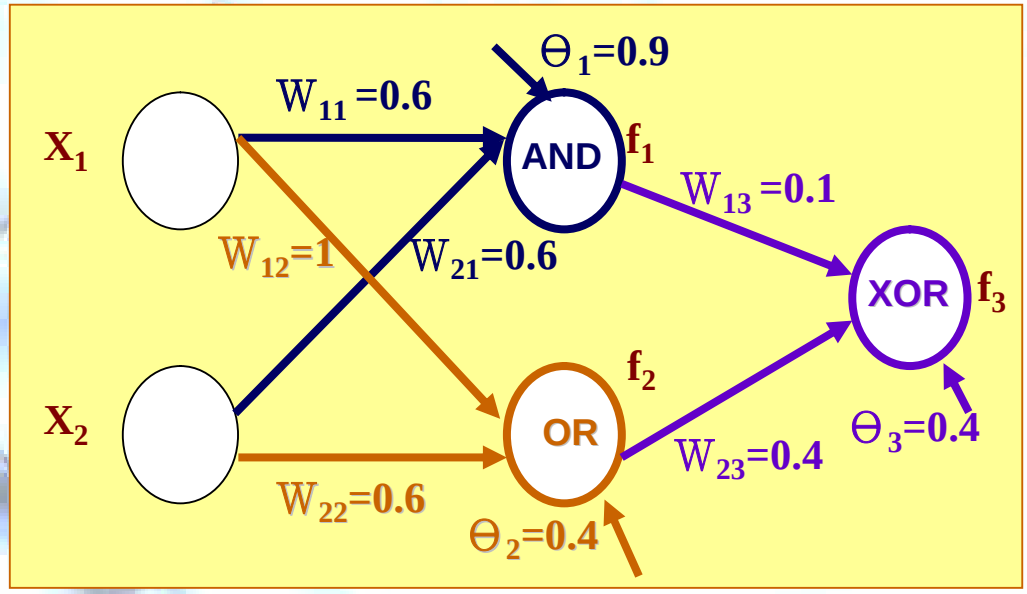
- Let us use the previous weights result from trained AND network (f_1) for W_{11}, W_{21}, and $\Theta 1$. Let us use the previous weights result from trained OR network (f_2) for W_{12}, W_{22}, and Θ_2
- Now, we uses the output of f_1 and f_2 for next layer process.



Solving the XOR problem(4)

- The following pattern is formed.

X_1	X_2	T_1 AND(f_1)	T_2 OR (f_2)	T_3
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



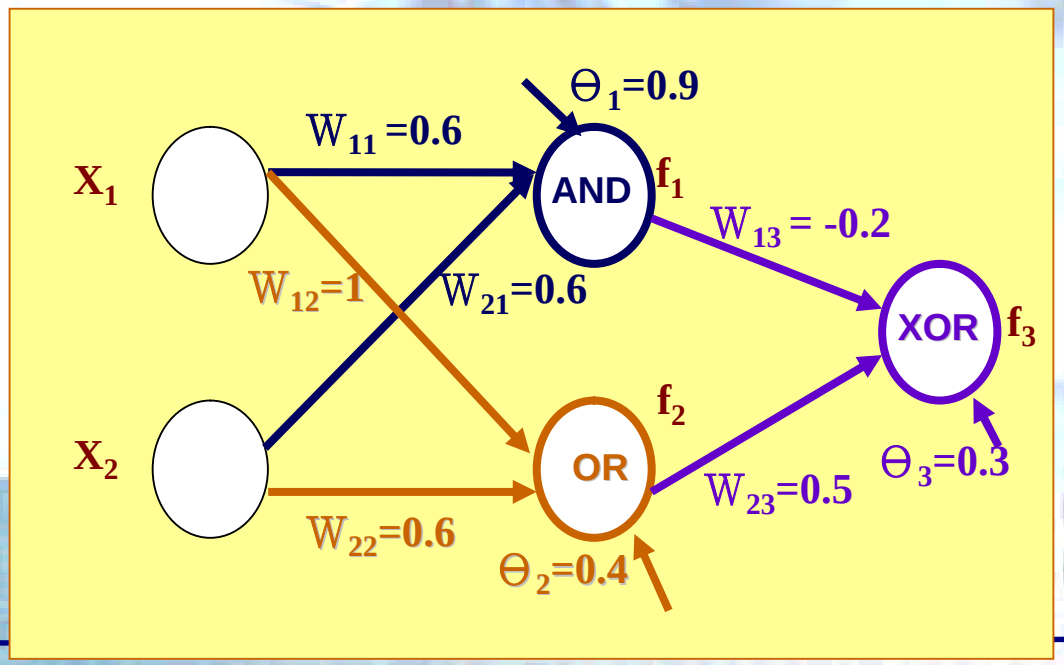
- Let $W_{13}=0.1$, $W_{23}=0.4$, $\Theta_3=0.4$
- The previous trained net function for node f_1 and node f_2 are:

$$f_1 = 0.6X_{11} + 0.6X_{21} - 0.9$$

$$f_2 = 1 \cdot X_{12} + 0.6 \cdot X_{22} - 0.4$$

Solving the XOR problem(5)

- Now we need to feed the input, one by one, for training the network. We do the recall for f1 and f2 separately. This is to satisfying the expected output for f1 using T1 and for f2 using T2.
- Finally, we use f1 and f2 as input pattern to train the node f_3, the result is $\mathbf{f_3=-0.2 \bullet X_{13} + 0.5X_{23} - 0.3}$



Deriving AND(f1) weights using Excel

AND NET (f1)

X1	X2	W11	W21	θ 2	net1	Y1(f1)	T1	T1 - Y1	△W11	△W21	△θ 2	η	
0	0	0.5	0.5	1	-1	0	0	0	0.5	0.5	1	0.1	pass
0	1	0.5	0.5	1	-0.5	0	0	0	0.5	0.5	1	0.1	pass
1	0	0.5	0.5	1	-0.5	0	0	0	0.5	0.5	1	0.1	pass
1	1	0.5	0.5	1	0	0	1	1	0.1	0.1	-0.1	0.1	<=update weight
1	1	0.6	0.6	0.9	0.3	1	1	0	0.6	0.6	0.9	0.1	verify
0	0	0.6	0.6	0.9	-0.9	0	0	0	0.6	0.6	0.9	0.1	pass
0	1	0.6	0.6	0.9	-0.3	0	0	0	0.6	0.6	0.9	0.1	pass
1	0	0.6	0.6	0.9	-0.3	0	0	0	0.6	0.6	0.9	0.1	pass

Deriving OR(f2) weights using Excel

OR NET (f2)

X1	X2	W21	W22	θ 1	net2	Y2(f2)	T2	T2 - Y2	△W12	△W22	△θ 1	η	
0	0	1	0.5	0.5	-0.5	0	0	0	1	0.5	0.5	0.1	pass
0	1	1	0.5	0.5	0	0	1	1	0	0.1	-0.1	0.1	<=update weight
0	1	1	0.6	0.4	0.2	1	1	0	1	0.6	0.4	0.1	verify
1	0	1	0.6	0.4	0.6	1	1	0	1	0.6	0.4	0.1	pass
1	1	1	0.6	0.4	1.2	1	1	0	1	0.6	0.4	0.1	pass

Deriving XOR(f3) weights using Excel

XOR NET (f3)

f1	f2	W13	W23	θ3	net3	Y3(f3)	T3	T3 - Y3	△W13	△W23	△θ3	η	
0	0	0.1	0.4	0.4	-0.4	0	0	0	0.1	0.4	0.4	0.1	pass
0	1	0.1	0.4	0.4	0	0	1	1	0	0.1	-0.1	0.1	<=update weight
1	1	0.1	0.5	0.3	0.3	1	0	-1	-0.1	-0.1	0.1	0.1	<=update weight
0	0	0	0.4	0.4	-0.4	0	0	0	0	0.4	0.4	0.1	pass
0	1	0	0.4	0.4	0	0	1	1	0	0.1	-0.1	0.1	<=update weight
1	1	0	0.5	0.3	0.2	1	0	-1	-0.1	-0.1	0.1	0.1	<=update weight
0	0	-0.1	0.4	0.4	-0.4	0	0	0	-0.1	0.4	0.4	0.1	pass
0	1	-0.1	0.4	0.4	0	0	1	1	0	0.1	-0.1	0.1	<=update weight
1	1	-0.1	0.5	0.3	0.1	1	0	-1	-0.1	-0.1	0.1	0.1	<=update weight
0	0	-0.2	0.4	0.4	-0.4	0	0	0	-0.2	0.4	0.4	0.1	pass
0	1	-0.2	0.4	0.4	0	0	1	1	0	0.1	-0.1	0.1	<=update weight
1	1	-0.2	0.5	0.3	0	0	0	0	-0.2	0.5	0.3	0.1	pass
0	0	-0.2	0.5	0.3	-0.3	0	0	0	-0.2	0.5	0.3	0.1	pass