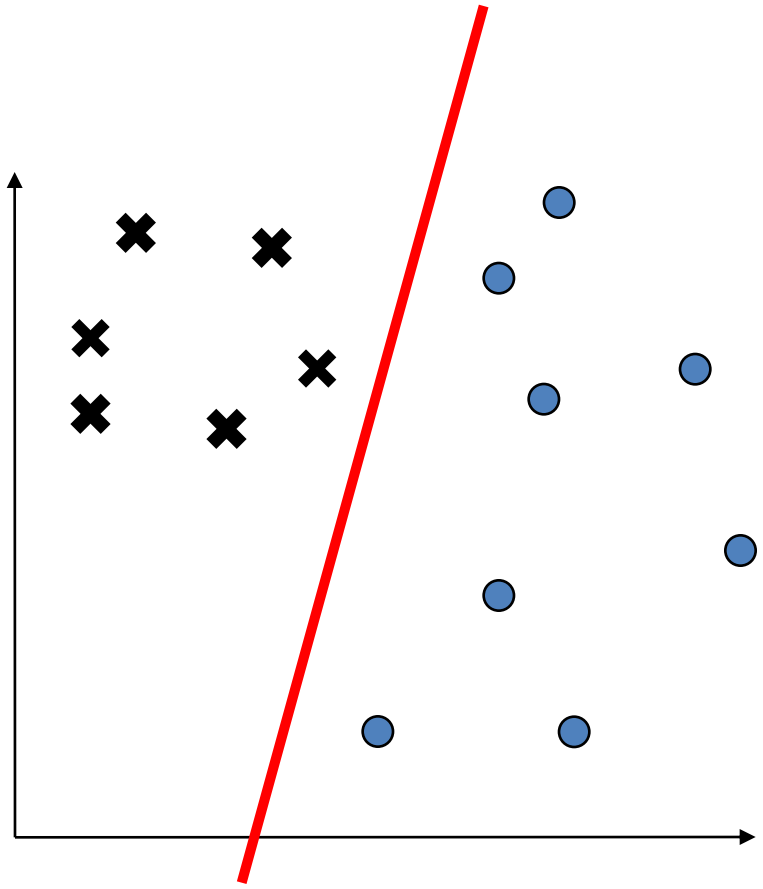
Lecture -13
Machine Learning

Artificial Neural Network - Perceptron

A (Linear) Decision Boundary



Represented by:

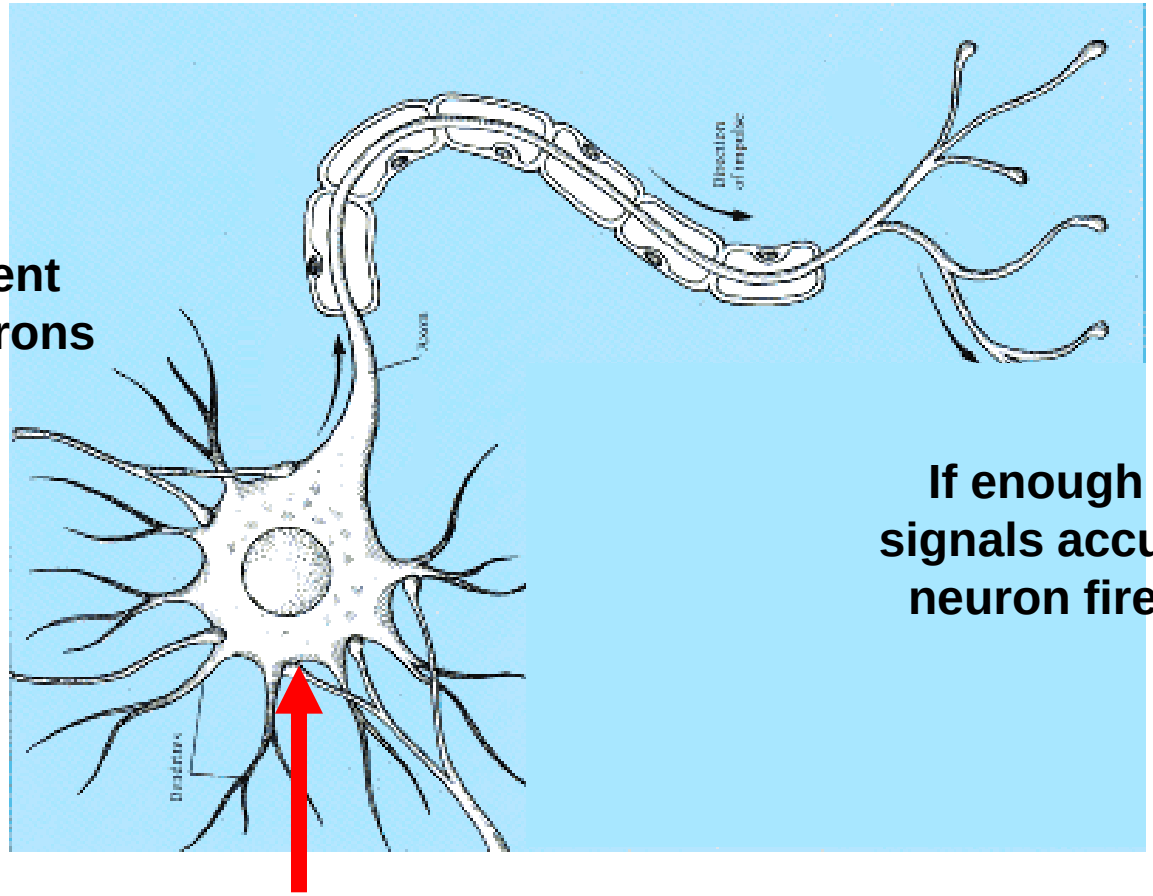
One artificial neuron called a "Perceptron"

Low accuracy (mostly)

Low space complexity

Low time complexity

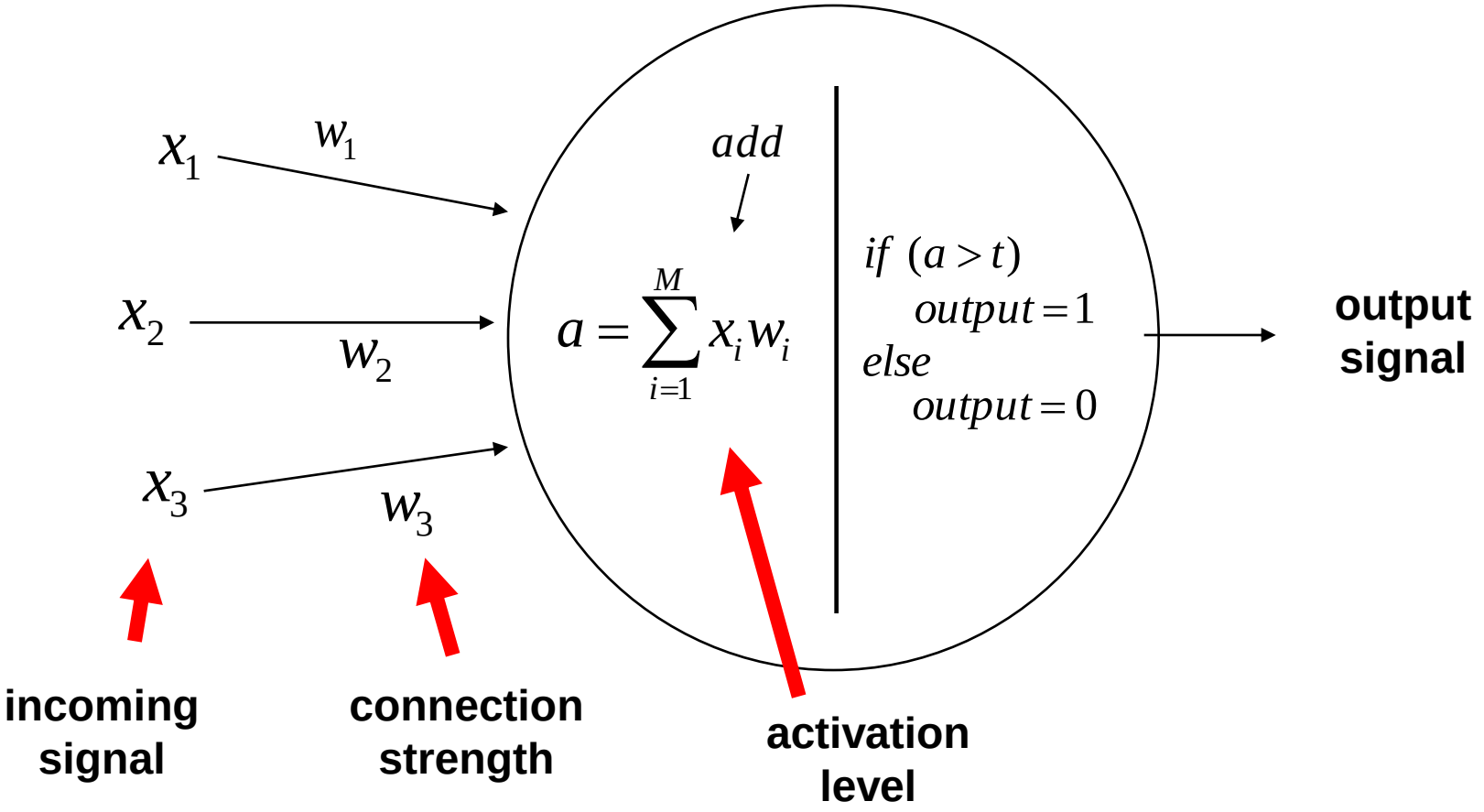
Input signals sent from other neurons



If enough sufficient signals accumulate, the neuron fires a signal.

Connection strengths determine how the signals are accumulated

- input signals ‘x’ and weights ‘w’ are multiplied
- weights correspond to connection strengths
- signals are added up – if they are enough, FIRE!



Calculation…

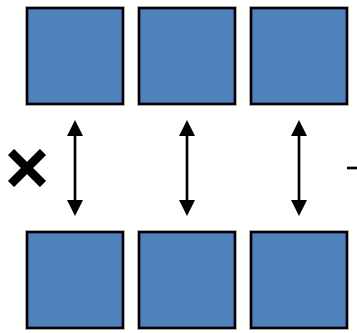
$$a = \sum_{i=1}^{M} x_i w_i$$

Sum notation

(just like a loop from 1 to M)

`double[] x =`

`double[] w =`



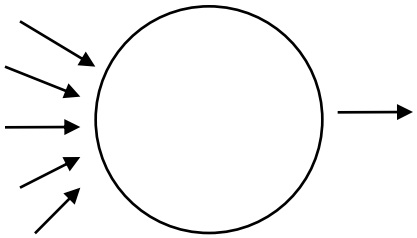
Multiple corresponding elements and add them up

(activation)

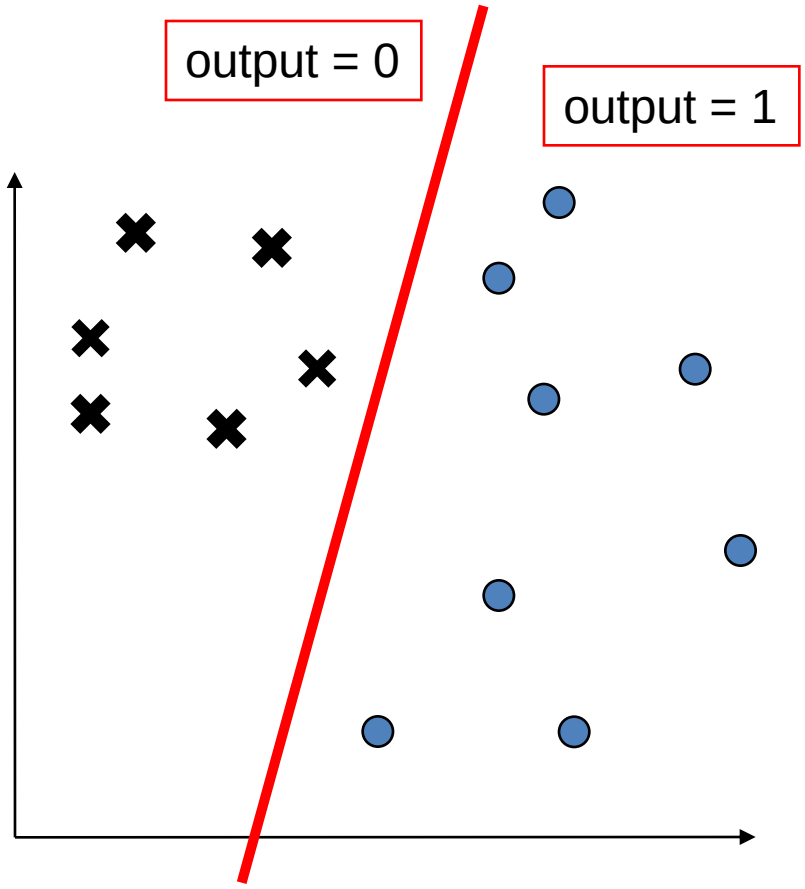
if (activation > threshold) FIRE !

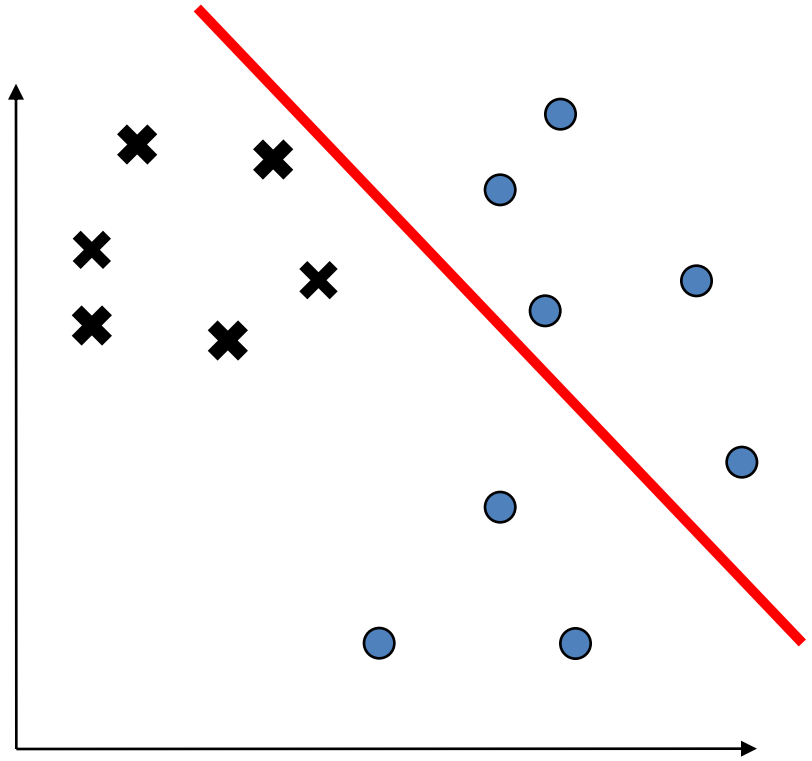
Perceptron Decision Rule

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$



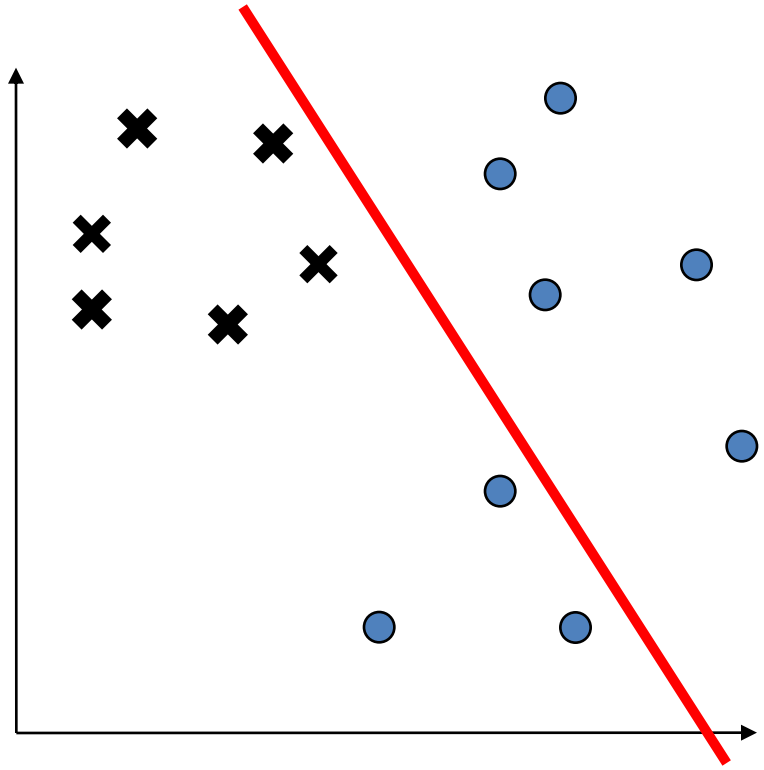
$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$





Is this a good decision boundary?

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

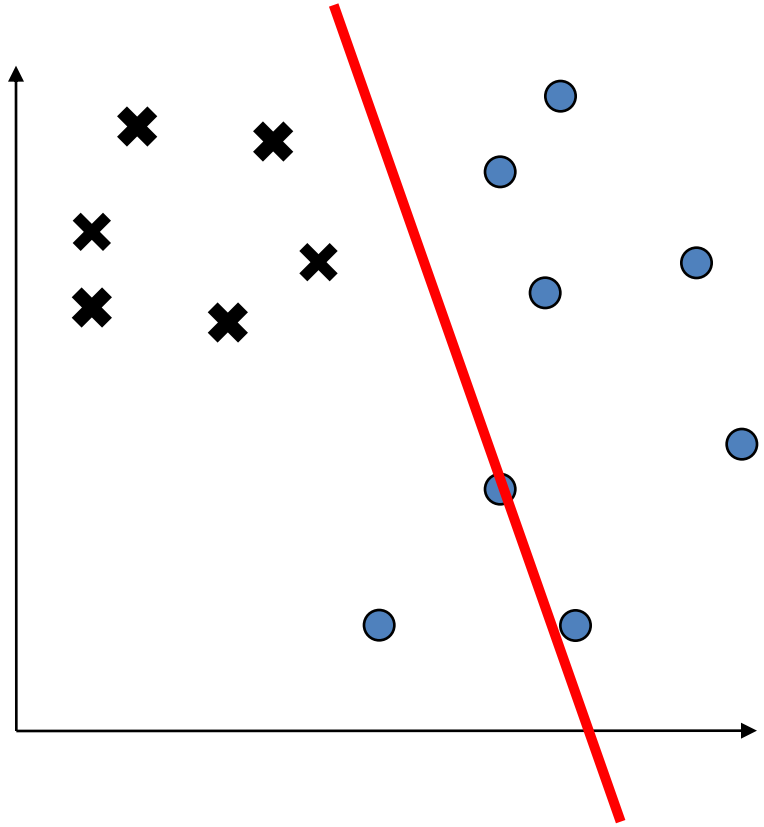


$w_1 = 1.0$

$w_2 = 0.2$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

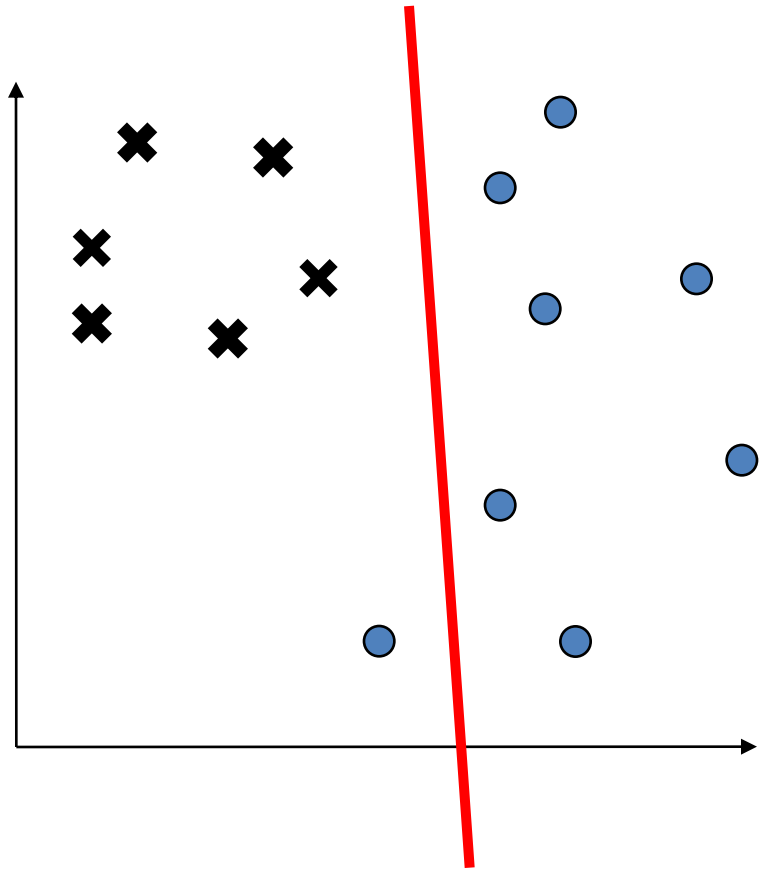


$w_1 = 2.1$

$w_2 = 0.2$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

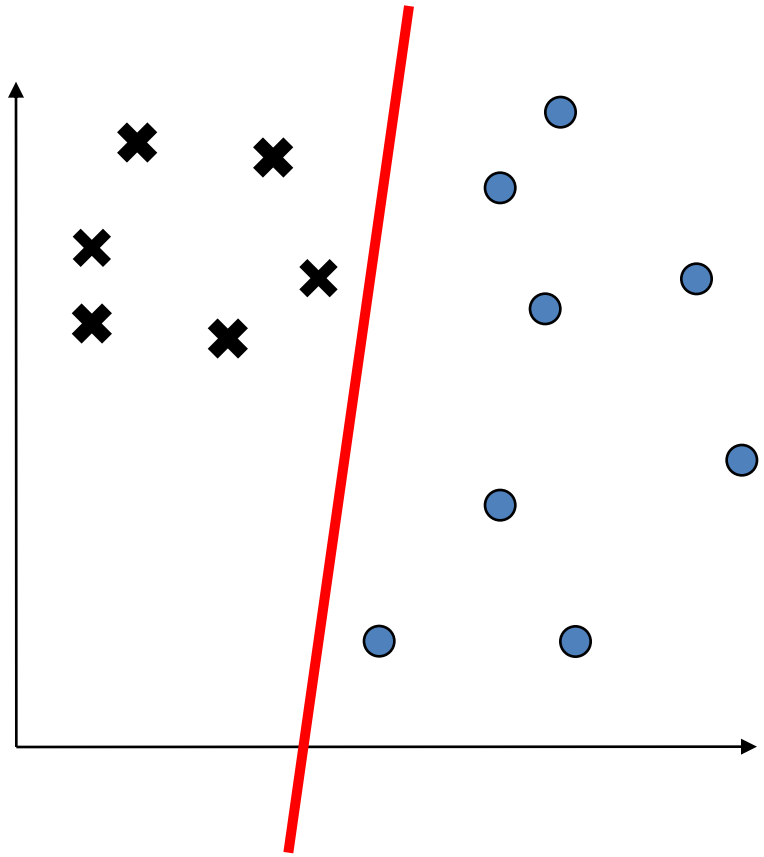


$w_1 = 1.9$

$w_2 = 0.02$

$t = 0.05$

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$



$w_1 = -0.8$

$w_2 = 0.03$

$t = 0.05$

Changing the weights/threshold makes the decision boundary move.

Pointless / impossible to do it by hand – only ok for simple 2-D case.

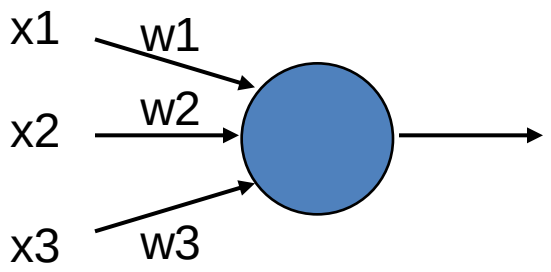
We need an algorithm….

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q1. What is the activation, ***a***, of the neuron?

Q2. Does the neuron fire?

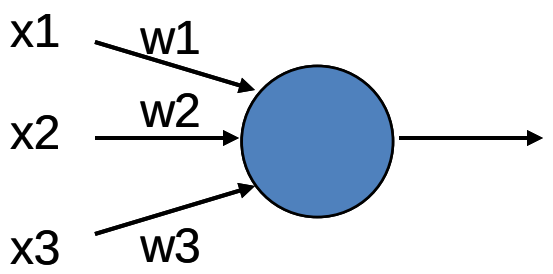
Q3. What if we set threshold at 0.5 and weight #3 to zero?

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q1. What is the activation, *a*, of the neuron?

$$a = \sum_{i=1}^{M} x_i w_i = (1.0 \times 0.2) + (0.5 \times 0.5) + (2.0 \times 0.5) = 1.45$$

Q2. Does the neuron fire?

if (activation > threshold) output=1 else output=0

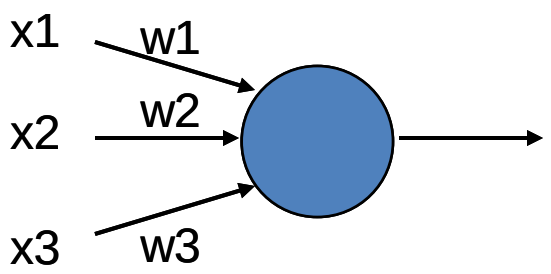
…. So yes, it fires.

$$x = [\ 1.0,\ 0.5,\ 2.0\ ]$$

$$w = [\ 0.2,\ 0.5,\ 0.5\ ]$$

$$t = 1.0$$

$$a = \sum_{i=1}^{M} x_i w_i$$



Q3. What if we set threshold at 0.5 and weight #3 to zero?

$$a = \sum_{i=1}^{M} x_i w_i = (1.0 \times 0.2) + (0.5 \times 0.5) + (2.0 \times 0.0) = 0.45$$

if (activation > threshold) output=1 else output=0

…. So no, it does not fire..

We can rearrange the decision rule….

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) - t > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) + (-1 \times t) > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) + (x_0 \times w_0) > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

$$\text{if} \quad \left(\sum_{i=0}^{M} x_i w_i \right) > 0 \quad \text{then } output = 1, \text{ else } output = 0$$

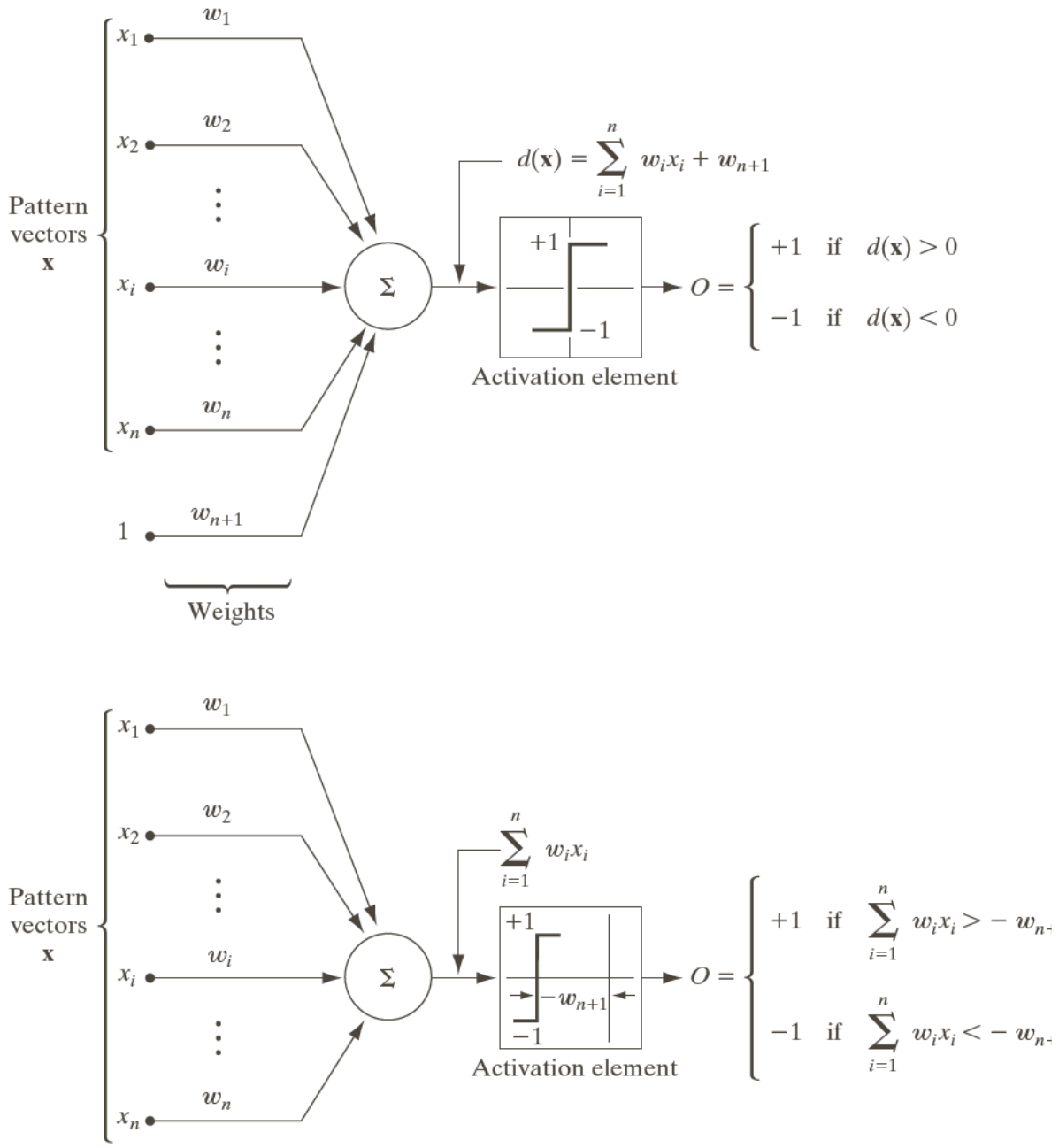
We now treat the threshold like any other weight with a permanent input of -1

Perceptron Learning Algorithm

```
initialise weights (w)
Repeat until all points are correctly classified
        Repeat for each point
                Calculate margin (yiwXi) for point i)
                If margin > 0, point is correclty classified
                Else change the weights to increase margin such
                that Δw = ηyiXi and wnew = wold + Δw
        end
end
```

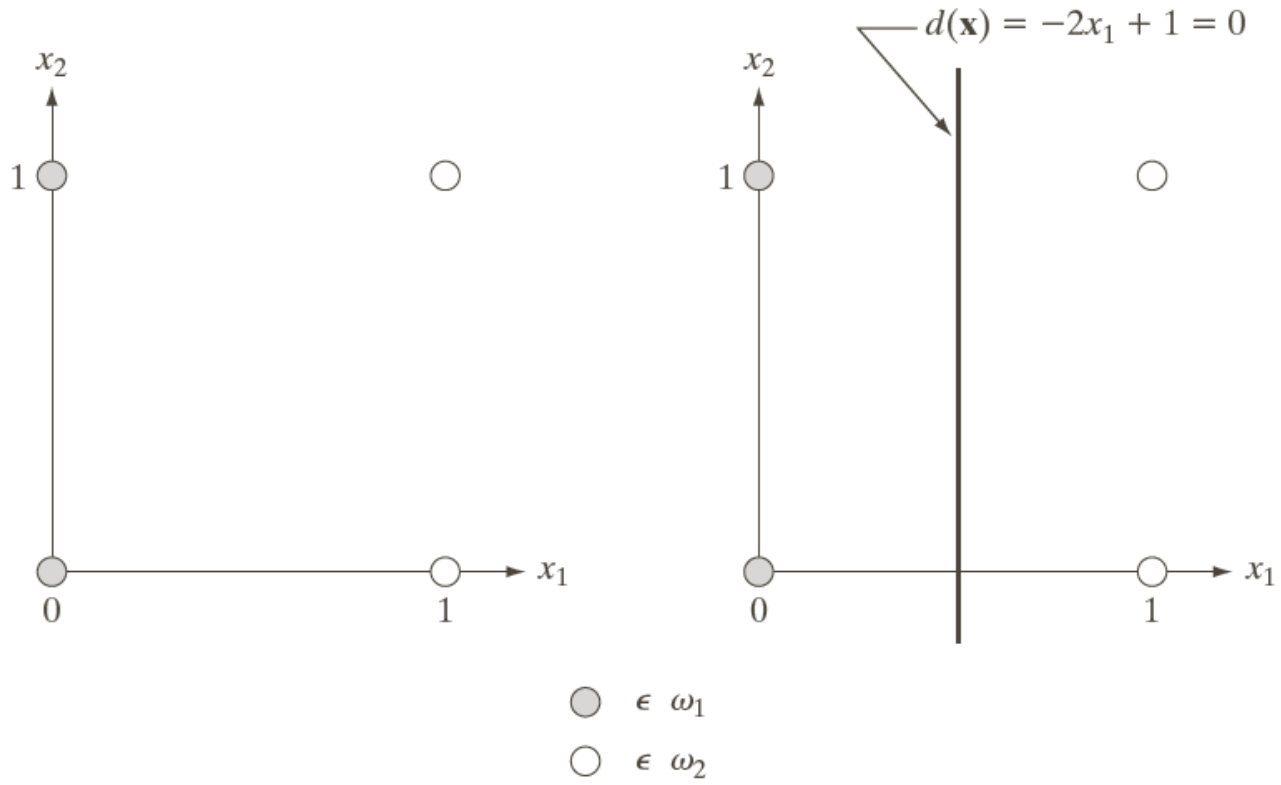
Perceptron convergence theorem:

If the data is linearly separable, then application of the Perceptron learning rule will find a separating decision boundary, within a finite number of iterations



a
b **FIGURE 12.14** Two equivalent representations of the perceptron model for two pattern classes.

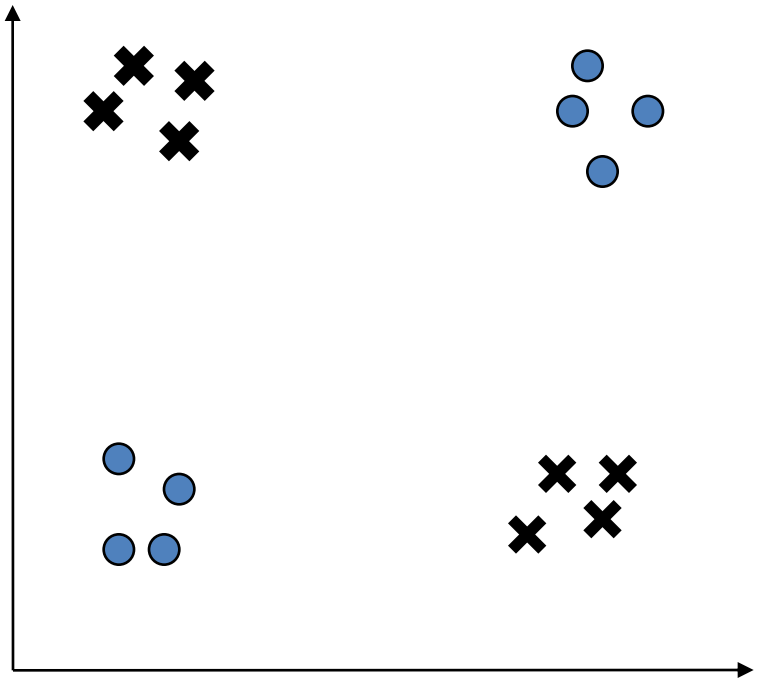
Decision Boundary Using Perceptron



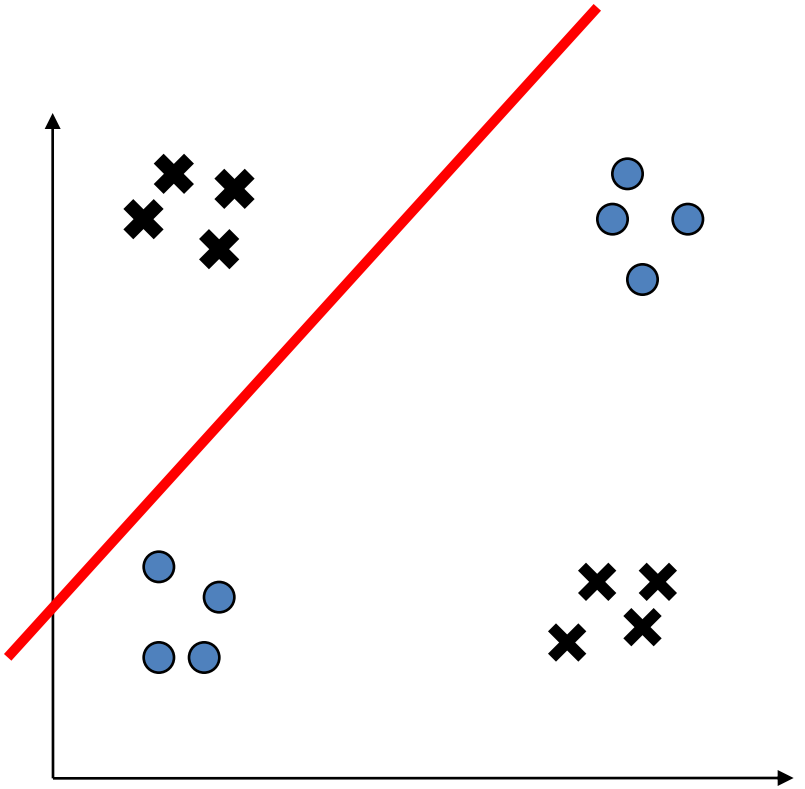
a b

FIGURE 12.15 (a) Patterns belonging to two classes. (b) Decision boundary determined by training.

Can a Perceptron solve this problem?



Can a Perceptron solve this problem? ….. NO.



Perceptrons only solve

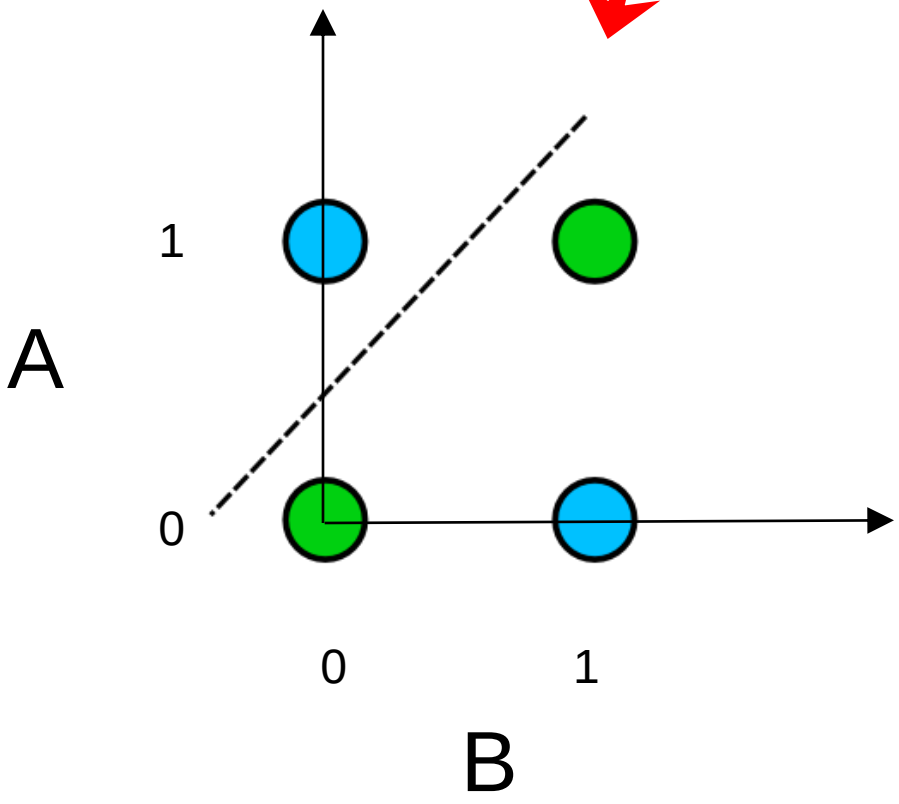
LINEARLY SEPARABLE

problems

With a perceptron…
the decision boundary is
LINEAR

Exclusive-OR gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



Multilayer Neural Network

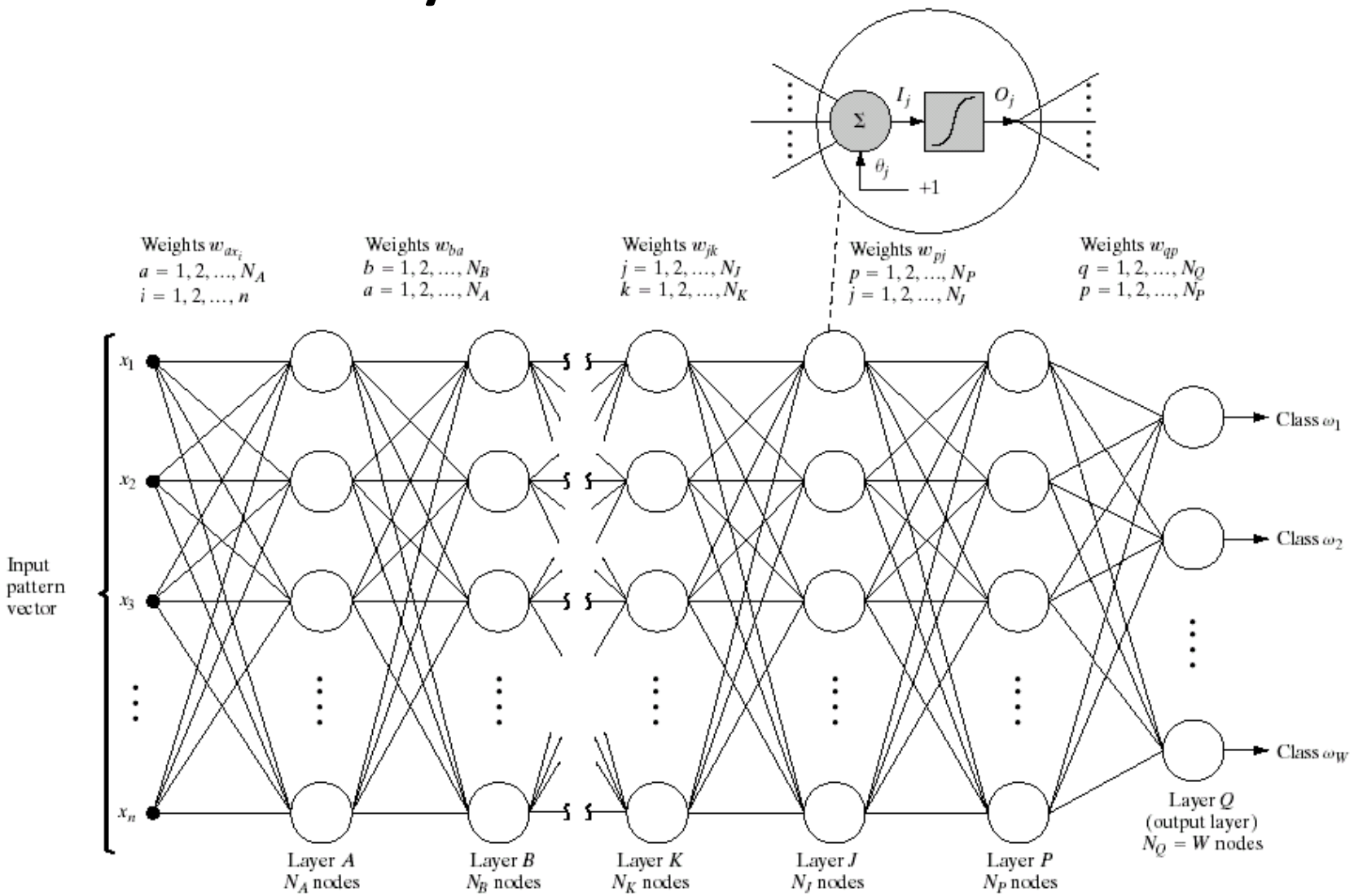


FIGURE 12.16 Multilayer feedforward neural network model. The blowup shows the basic structure of each neuron element throughout the network. The offset, θ_j, is treated as just another weight.

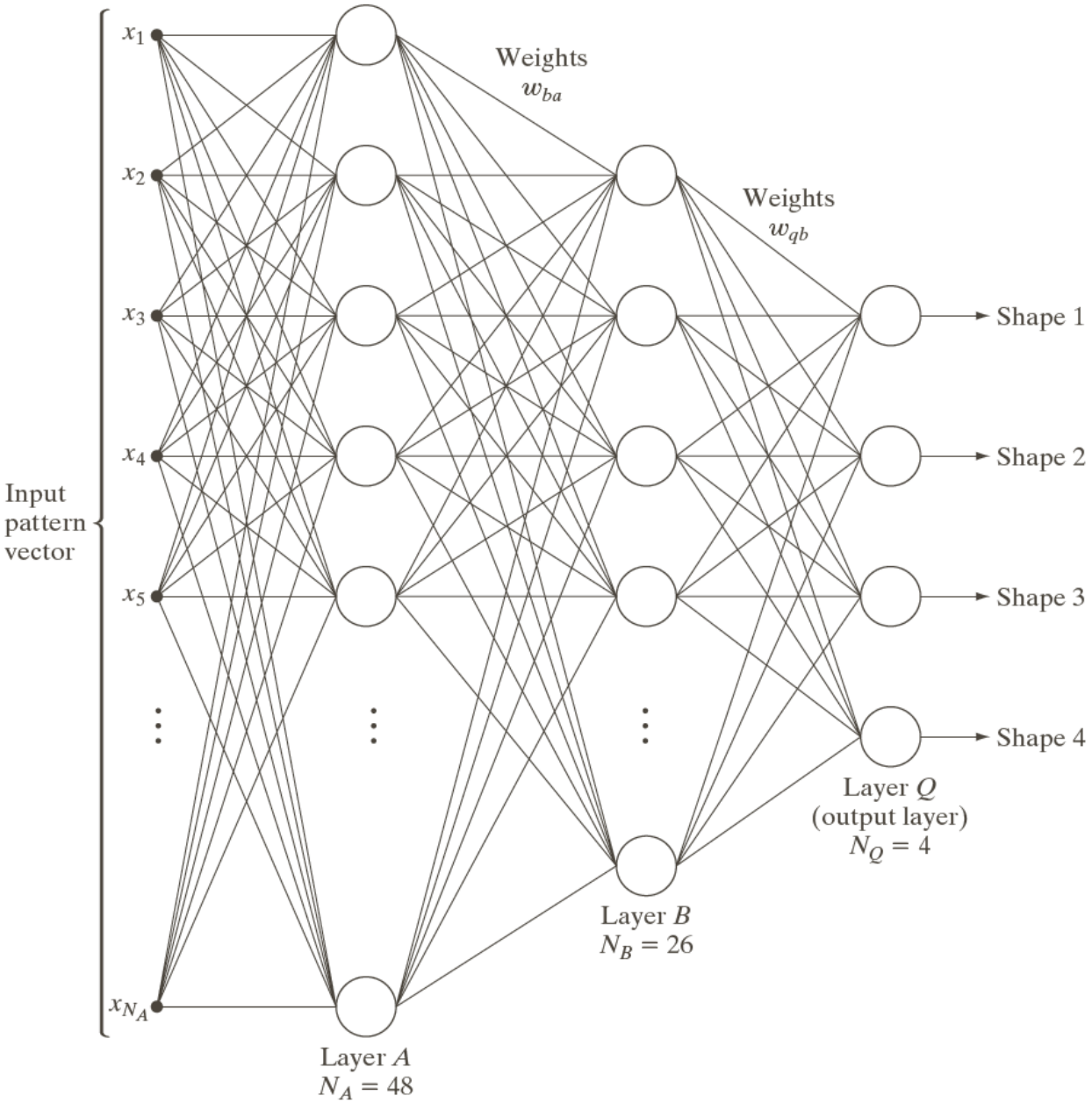
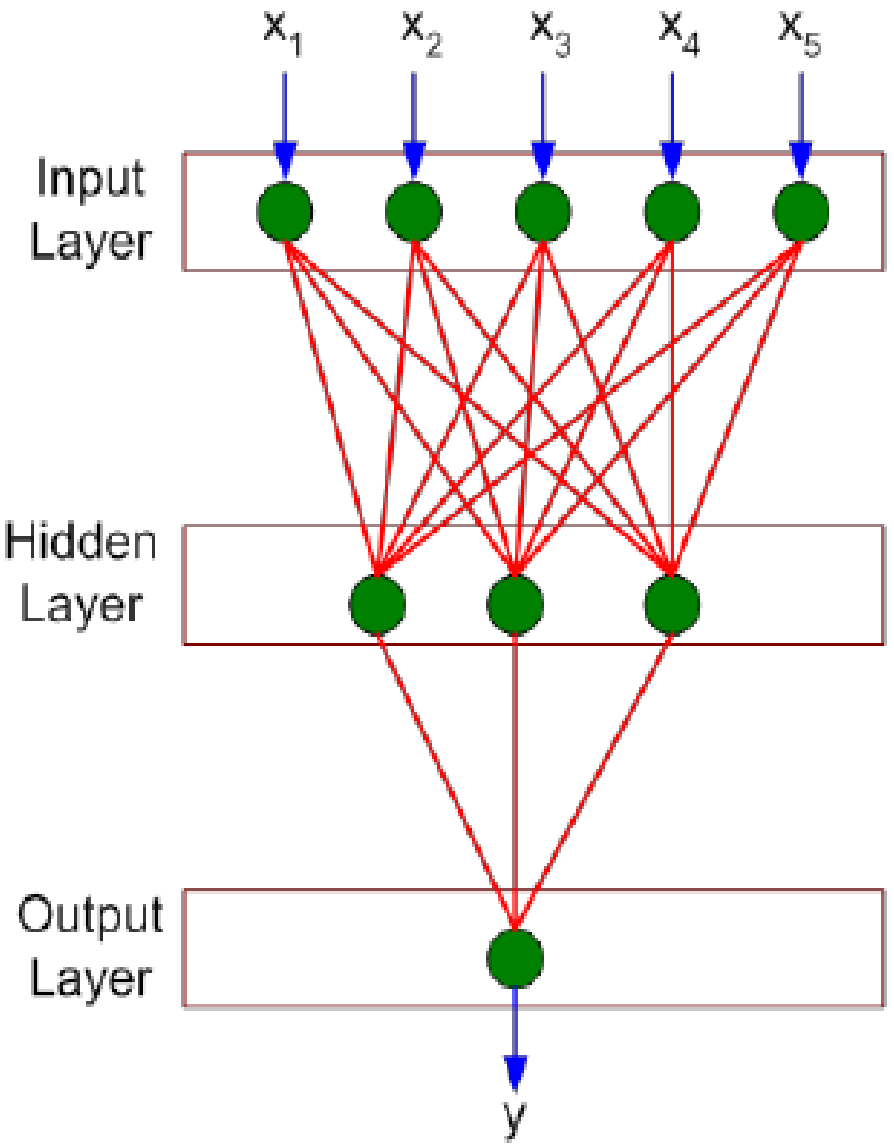


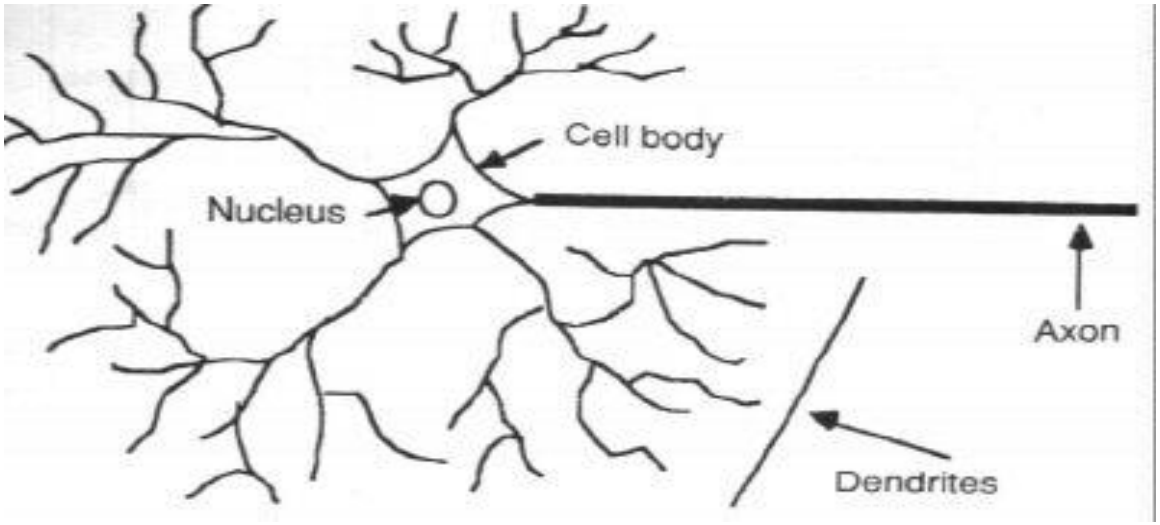
FIGURE 12.19 Three-layer neural network used to recognize the shapes in Fig. 12.18. (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

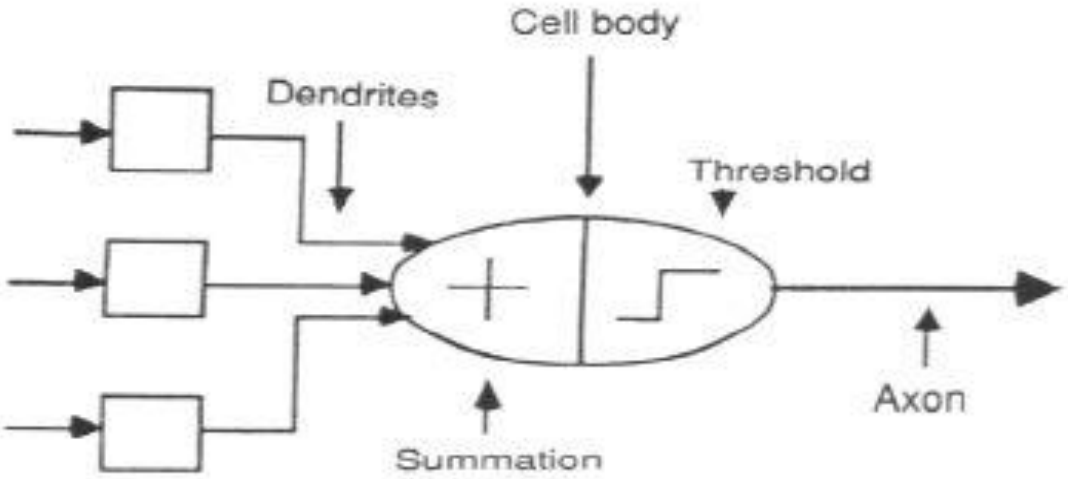
Artificial Neural Networks (ANN)

- Neural computing requires a number of neurons, to be connected together into a neural network.
- A neural network consists of:
  - layers
  - links between layers
- The links are weighted.
- There are three kinds of layers:
  1. input layer
  2. Hidden layer
  3. output layer



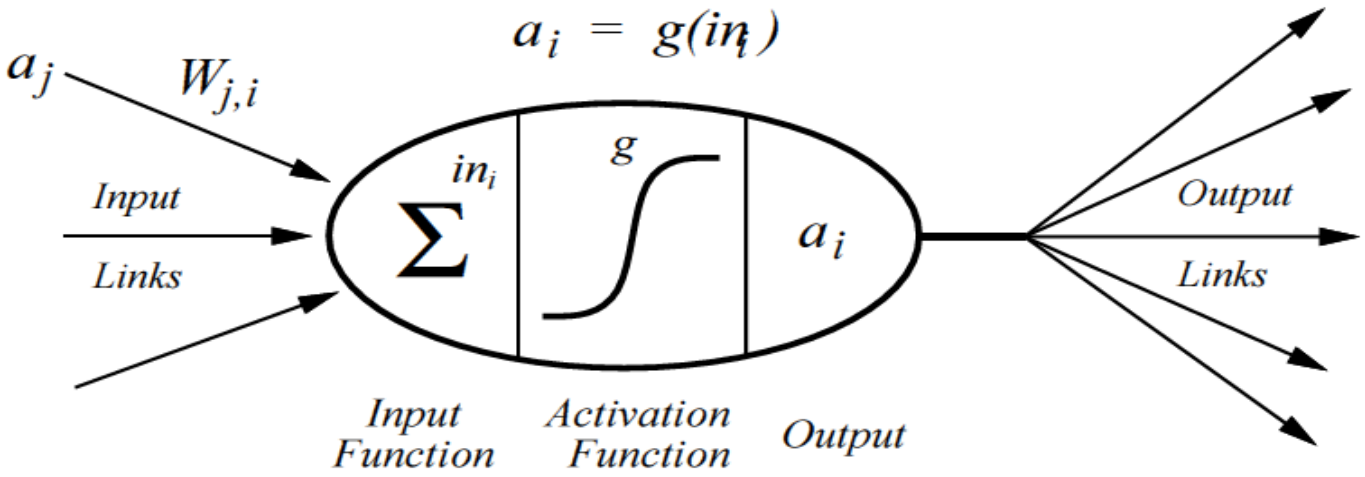
From Human Neurones to Artificial Neurones





A simple neuron

- At each neuron, every input has an associated weight which modifies the strength of each input.
- The neuron simply adds together all the inputs and calculates an output to be passed on.

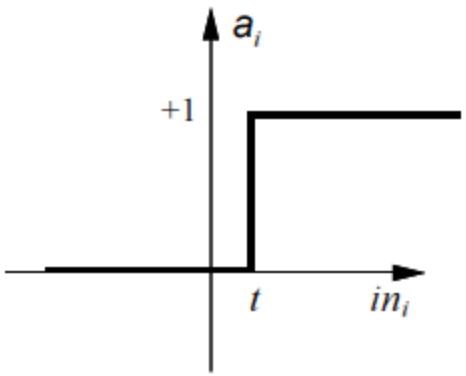


Activation function

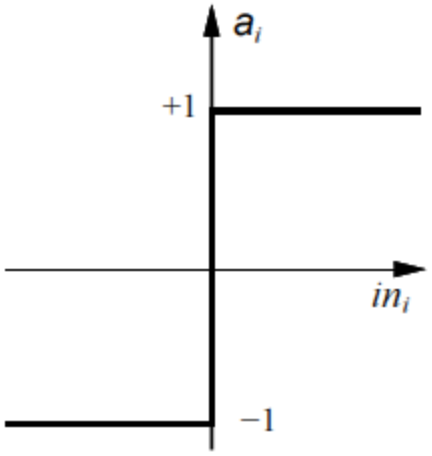
The **activation function** g calculates the output a_i (from the inputs) which will be transferred to other units via output-links:

$$a_i := g(in_i)$$

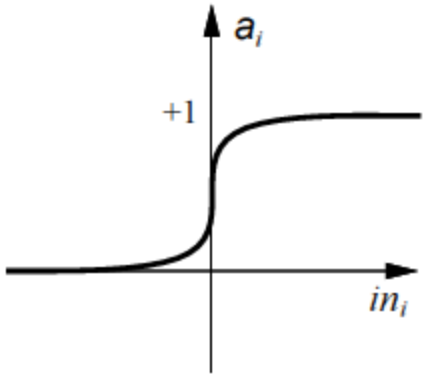
Examples:



(a) Step function



(b) Sign function



(c) Sigmoid function

MultiLayer Perceptron (MLP)

Motivation

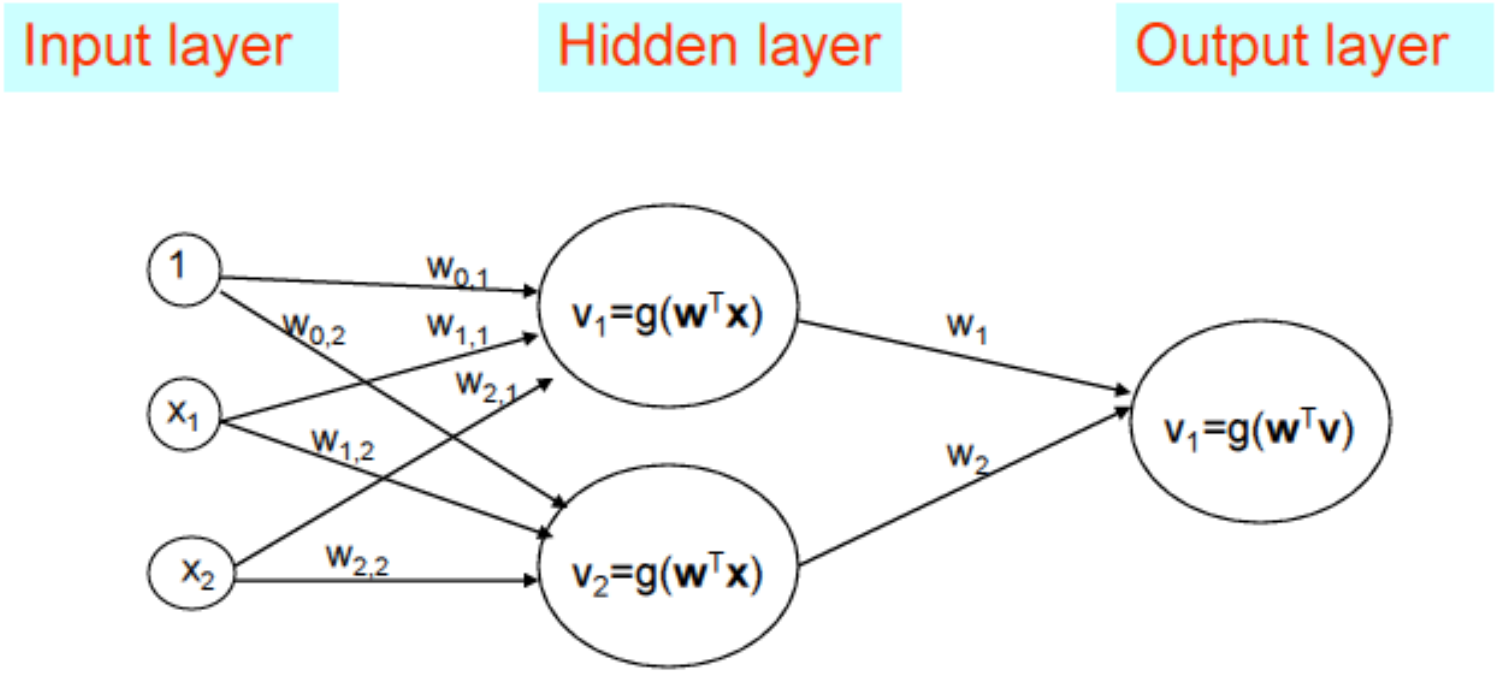
- Perceptrons are limited because they can only solve problems that are linearly separable
- We would like to build more complicated learning machines to model our data
- One way to do this is to build a multiple layers of perceptrons

Brief History

- 1985 Ackley, Hinton and Sejnowski propose the Boltzmann machine
  - This was a multi-layer step perceptron
  - More powerful than perceptron
  - Successful application NETtalk
- 1986 Rummelhart, Hinton and Williams invent Multi-Layer Perceptron (MLP) with backpropagation
  - Dominant neural net architecture for 10 years

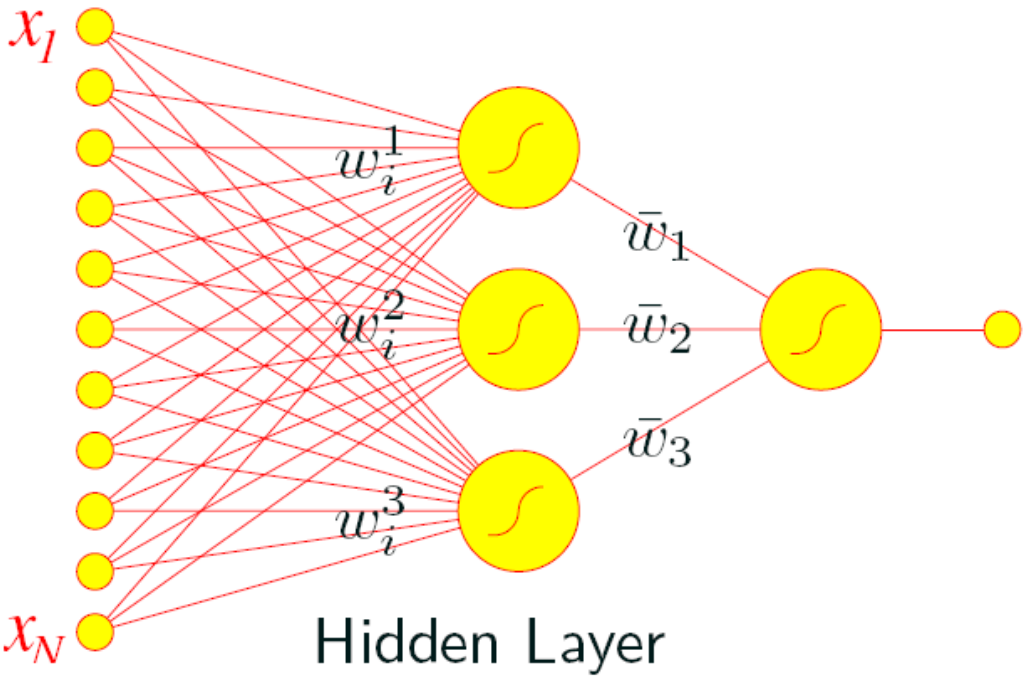
Multi layer networks

- So far we discussed networks with one layer.
- But these networks can be extended to combine several layers, increasing the set of functions that can be represented using a NN



MLP

- E.g. A multi-layer perceptron (MLP)



$$z = f(\boldsymbol{x}|\boldsymbol{w}) = g\left(\sum_i \bar{w}_i \, g\left(\boldsymbol{x}^\mathsf{T}\boldsymbol{w}^i\right)\right)$$

- Note that $\boldsymbol{w}$ in $f(\boldsymbol{x}|\boldsymbol{w})$ includes all weights. $\boldsymbol{w}$ is no longer the same dimension as the inputs $\boldsymbol{x}$

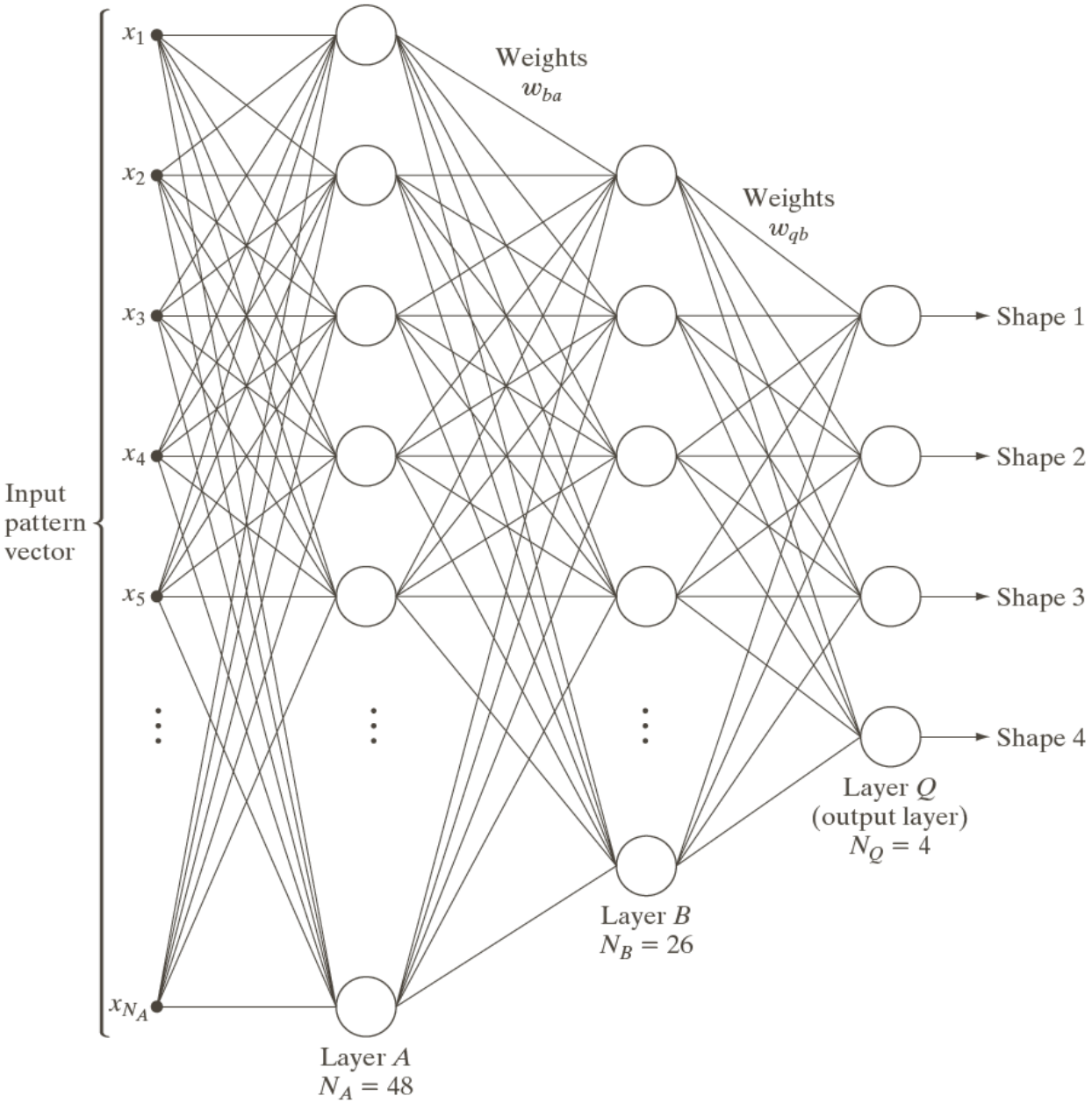


FIGURE 12.19 Three-layer neural network used to recognize the shapes in Fig. 12.18. (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

Multilayer Neural Network

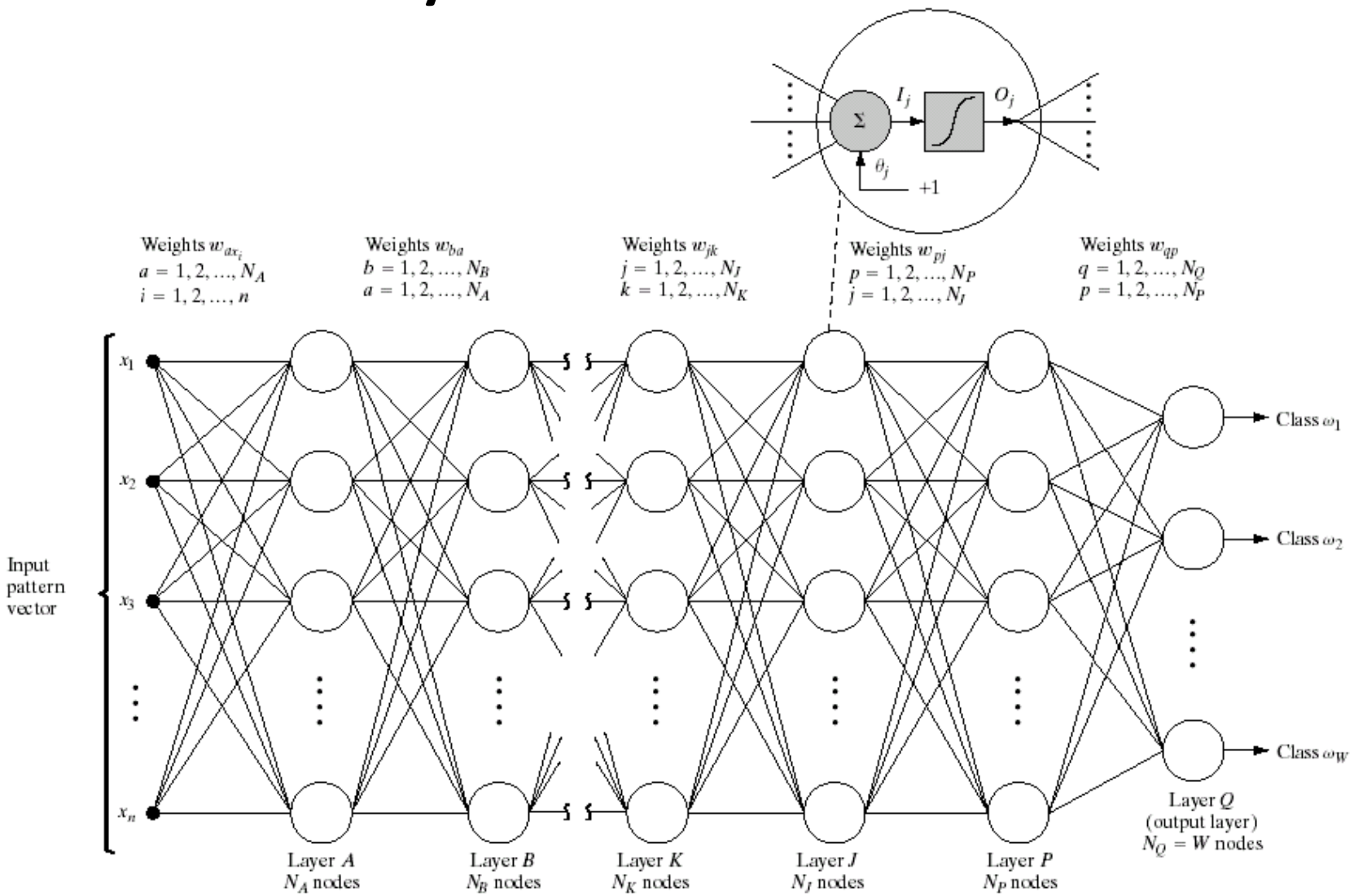


FIGURE 12.16 Multilayer feedforward neural network model. The blowup shows the basic structure of each neuron element throughout the network. The offset, θ_j, is treated as just another weight.

Sigmoid Response Functions

- To obtain a more powerful learning machine we have to use non-linear response functions
- The two most used functions are the sigmoidal (squashing) functions:
- A logistic function

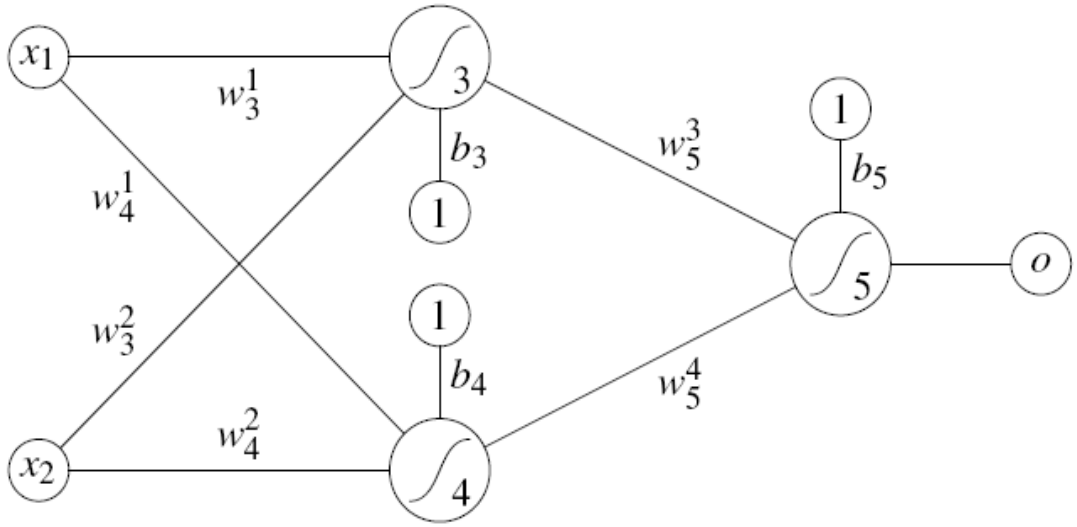
$$g(V) = \frac{1}{1 + \mathrm{e}^{-(V)}}$$

- A tanh function

$$g(V) = \tanh(V)$$

MLP

- A diagram for an neural network such as an MLP



- Stands for the function $(o = f(\boldsymbol{x}|\boldsymbol{w}))$

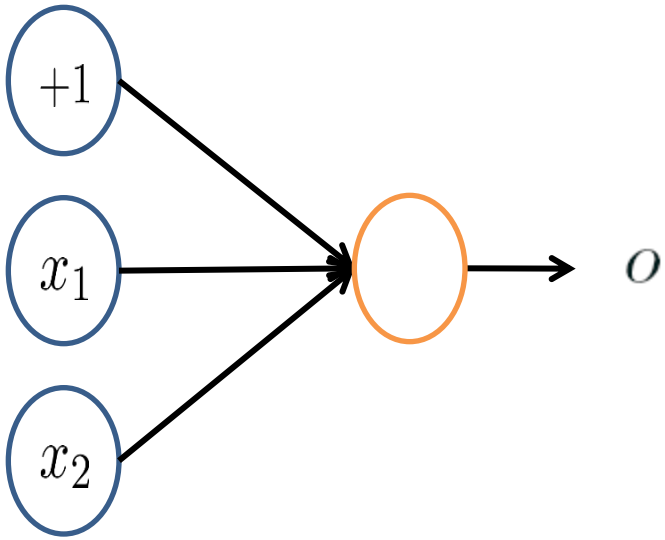
$$o = g\big(w_5^3\, g(w_3^1\, x_1 + w_3^2\, x_2 + b_3) + w_5^4\, g(w_4^1\, x_1 + w_4^2\, x_2 + b_4) + b_5\big)$$

where, for example, $g(V) = \frac{1}{1+\mathrm{e}^{-V}}$

Simple example: AND

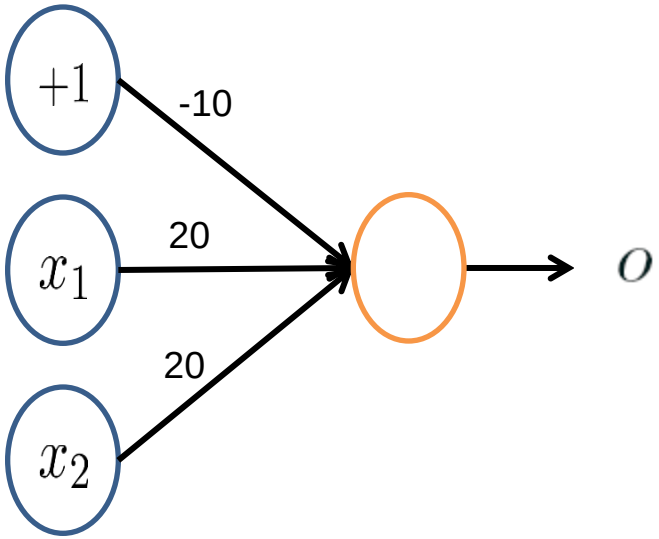
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



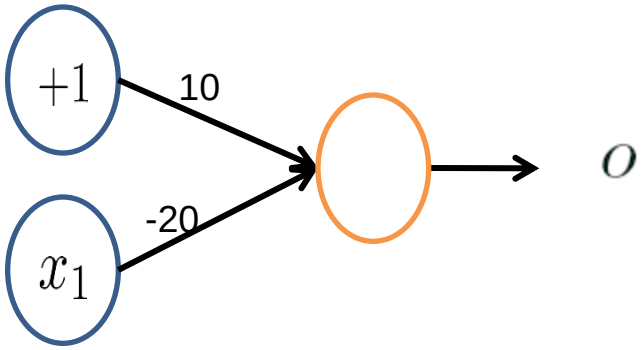
x_1	x_2	o
0	0	
0	1	
1	0	
1	1	

Example: OR function



x_1	x_2	o
0	0	
0	1	
1	0	
1	1	

Negation:

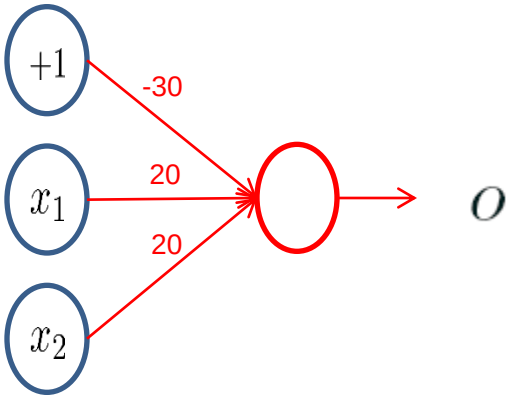


x_1	o
0	
1	

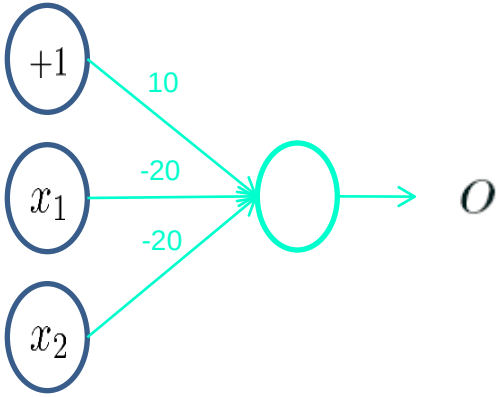
$$o = g(10 - 20x_1)$$

$$(\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$$

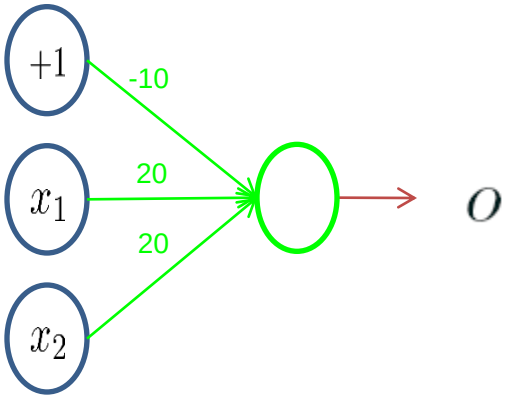
Putting it together: x_1 XNOR x_2



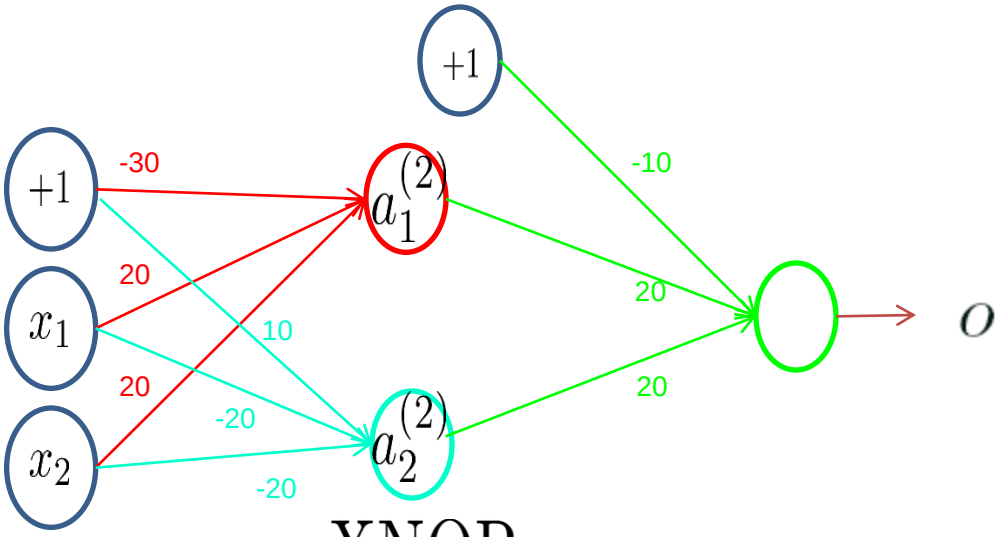
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



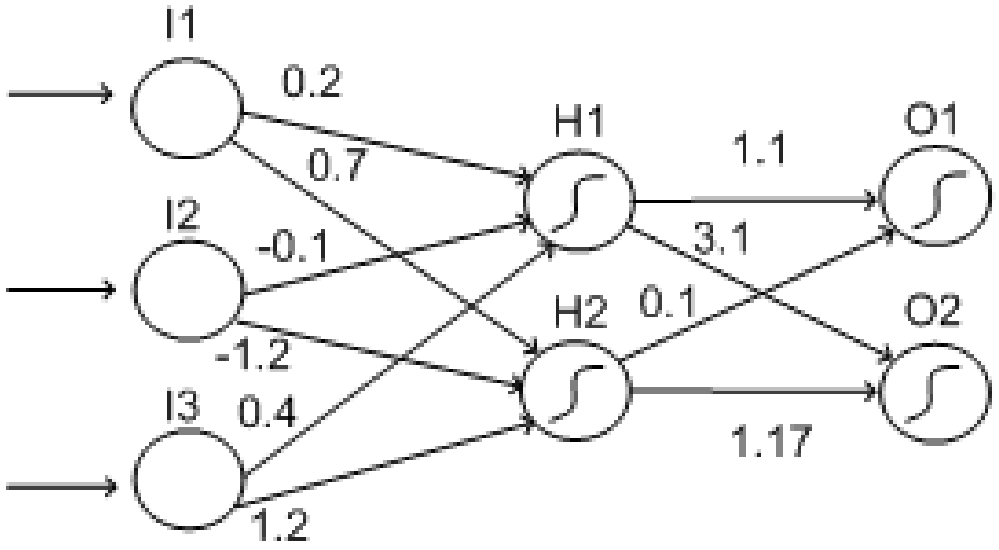
x_1 OR x_2



x_1 XNOR x_2

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	o
0	0			
0	1			
1	0			
1	1			

Example of multilayer Neural Network



- Suppose input values are 10, 30, 20
- The weighted sum coming into H1

$$S_{H1} = (0.2 * 10) + (-0.1 * 30) + (0.4 * 20)$$
$$= 2 -3 + 8 = 7.$$

- The σ function is applied to S_{H1}:

$$\sigma(S_{H1}) = 1/(1+e^{-7}) = 1/(1+0.000912) = 0.999$$

- Similarly, the weighted sum coming into H2:

$$S_{H2} = (0.7 * 10) + (-1.2 * 30) + (1.2 * 20)$$
$$= 7 - 36 + 24 = -5$$

- σ applied to S_{H2}:

$$\sigma(S_{H2}) = 1/(1+e^{5}) = 1/(1+148.4) = 0.0067$$

- Now the weighted sum to output unit O1 :

  $S_{O1} = (1.1 * 0.999) + (0.1*0.0067) = 1.0996$

- The weighted sum to output unit O2:

  $S_{O2} = (3.1 * 0.999) + (1.17*0.0067) = 3.1047$

- The output sigmoid unit in O1:

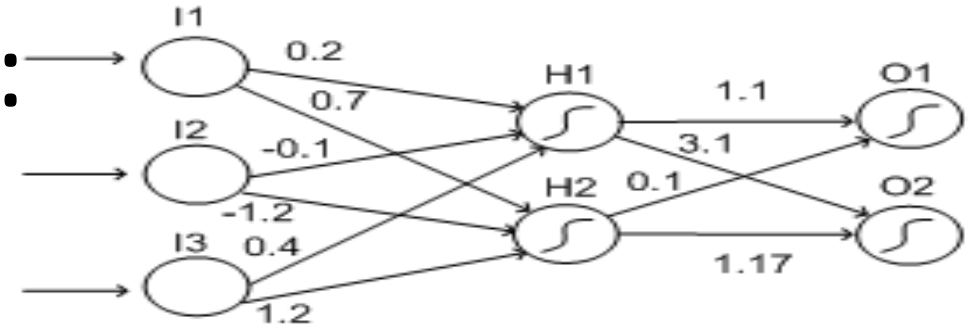
  $\sigma(S_{O1}) = 1/(1+e^{-1.0996}) = 1/(1+0.333) = 0.750$

- The output from the network for O2:

  $\sigma(S_{O2}) = 1/(1+e^{-3.1047}) = 1/(1+0.045) = 0.957$

- **The input triple (10,30,20) would be categorised with O2, because this has the larger output.**

A Worked Example:



Input units		Hidden units			Output units		
Unit	Output	Unit	Weighted Sum Input	Output	Unit	Weighted Sum Input	Output
I1	10	H1	7	0.999	O1	1.0996	0.750
I2	30	H2	-5	0.0067	O2	3.1047	0.957
I3	20						

- Propagated the values (10,30,20) through the network
- Suppose now that the target categorization for the example was the one associated with O1(using a learning rate of $\eta = 0.1$)
- the target output for O1 was 1, and the target output for O2 was 0
- t1(E) = 1; t2(E) = 0; o1(E) = 0.750; o2(E) = 0.957
- error values for the output units O1 and O2
  - $\delta O1 = o1(E)(1 - o1(E))(t1(E) - o1(E)) = 0.750(1-0.750)(1-0.750) = 0.0469$
  - $\delta O2 = o2(E)(1 - o2(E))(t2(E) - o2(E)) = 0.957(1-0.957)(0-0.957) = -0.0394$

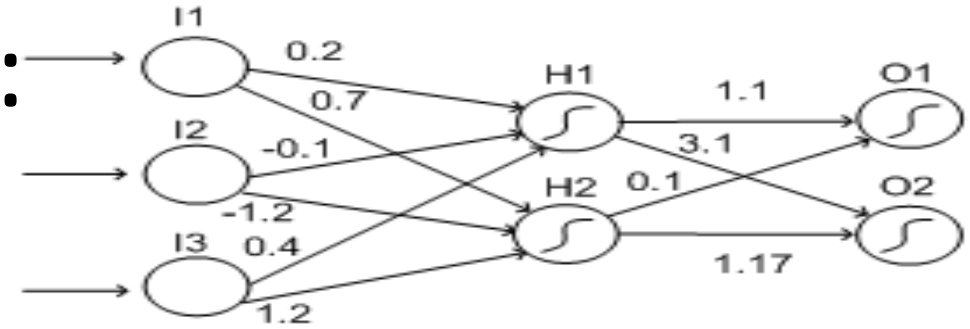
A Worked Example:

- To propagate this information backwards to the hidden nodes H1 and H2
  - Multiply the error term for O1 by the weight from H1 to O1, then add this to the multiplication of the error term for O2 and the weight between H1 and O2, (1.1*0.0469) + (3.1*-0.0394) = -0.0706

$$\delta_{H_k} = h_k(E)(1 - h_k(E)) \sum_{i \in outputs} w_{ki}\delta_{O_i}$$

  - δ_{H1} = -0.0706*(0.999 * (1-0.999)) = -0.0000705
  - Similarly for H2: (0.1*0.0469)+(1.17*-0.0394) = -0.0414
  - δ_{H2} -0.0414 * (0.067 * (1-0.067)) = -0.00259

A Worked Example:



Input unit	Hidden unit	η	δ_H	x_i	$\Delta = \eta * \delta_H * x_i$	Old weight	New weight
I1	H1	0.1	-0.0000705	10	-0.0000705	0.2	0.1999295
I1	H2	0.1	-0.00259	10	-0.00259	0.7	0.69741
I2	H1	0.1	-0.0000705	30	-0.0002115	-0.1	-0.1002115
I2	H2	0.1	-0.00259	30	-0.00777	-1.2	-1.20777
I3	H1	0.1	-0.0000705	20	-0.000141	0.4	0.39999
I3	H2	0.1	-0.00259	20	-0.00518	1.2	1.1948

Hidden unit	Output unit	η	δ_O	$h_i(E)$	$\Delta = \eta * \delta_O * h_i(E)$	Old weight	New weight
H1	O1	0.1	0.0469	0.999	0.000469	1.1	1.100469
H1	O2	0.1	-0.0394	0.999	-0.00394	3.1	3.0961
H2	O1	0.1	0.0469	0.0067	0.00314	0.1	0.10314
H2	O2	0.1	-0.0394	0.0067	-0.0000264	1.17	1.16998

Acknowledgements

- Introduction to Machine Learning, Alphaydin
- Statistical Pattern Recognition: A Review – A.K Jain et al., PAMI (22) 2000
- Pattern Recognition and Analysis Course – A.K. Jain, MSU
- *Pattern Classification"* by Duda et al., John Wiley & Sons.
- http://www.doc.ic.ac.uk/~sgc/teaching/pre2012/v231/lecture13.html
- Some Material adopted from Dr. Adam Prugel-Bennett Dr. Andrew Ng and Dr. Aman ullah's Slides

Material in these slides has been taken from, the following resources