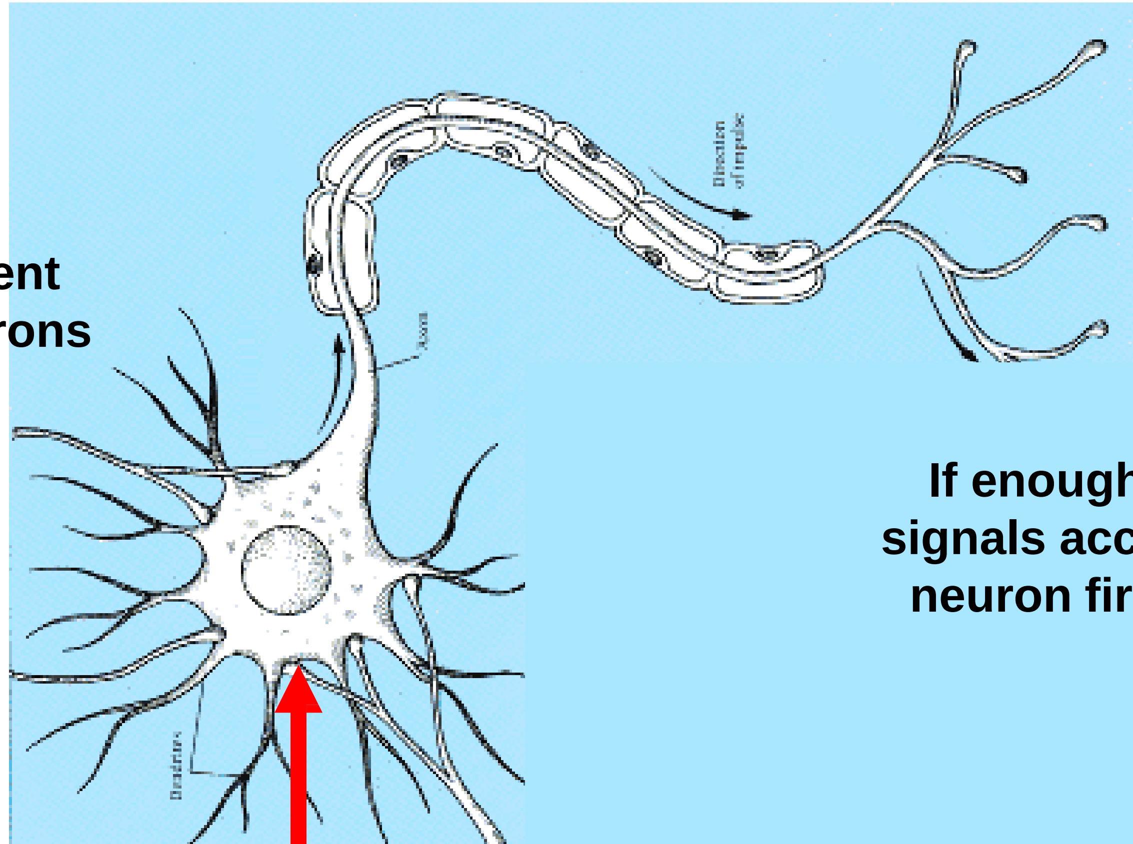
Machine Learning

Lecture # 4
Single & Multilayer Perceptron

Artificial Neural Network - Perceptron

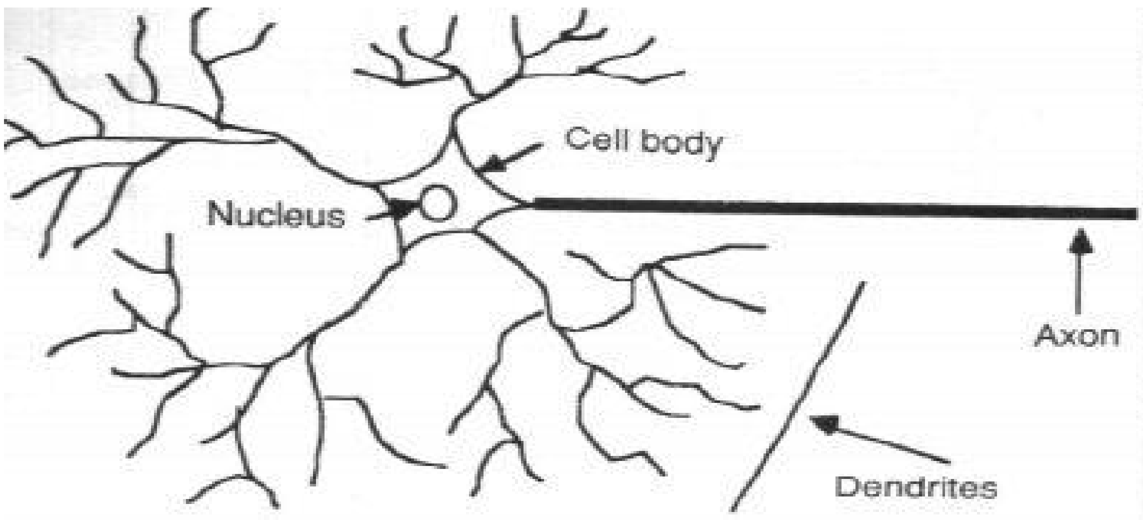
Input signals sent from other neurons

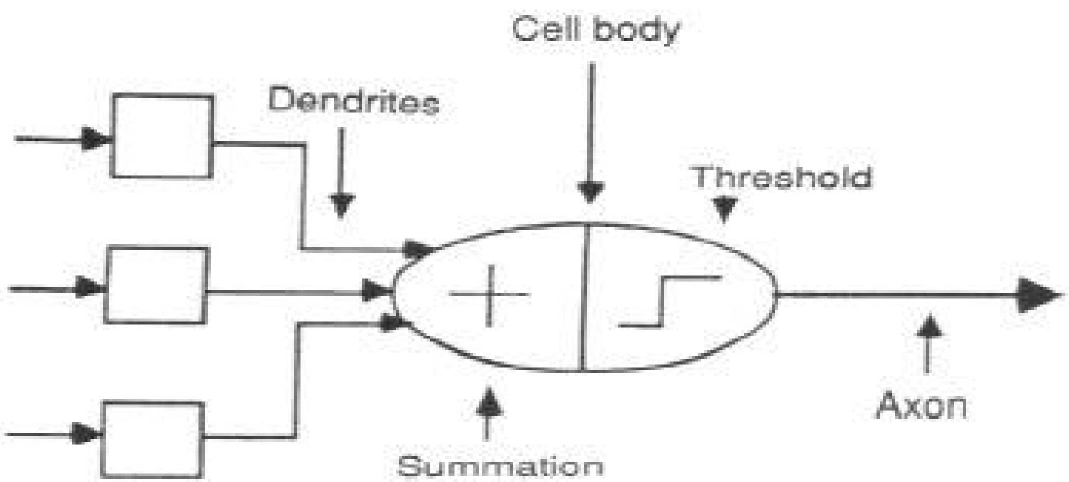


If enough sufficient signals accumulate, the neuron fires a signal.

Connection strengths determine how the signals are accumulated

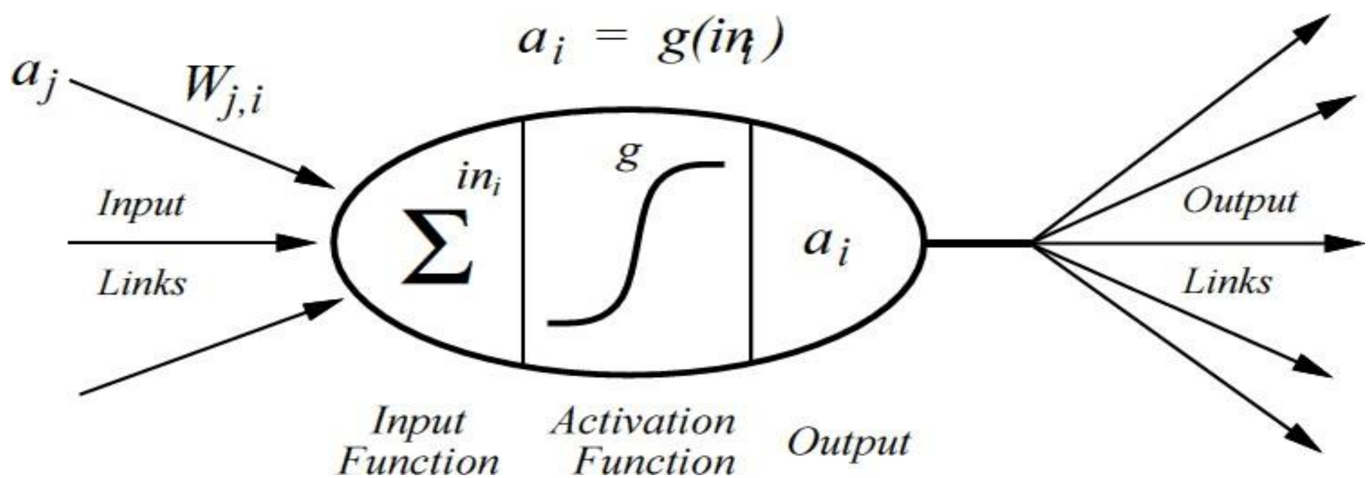
From Human Neurones to Artificial Neurones





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.

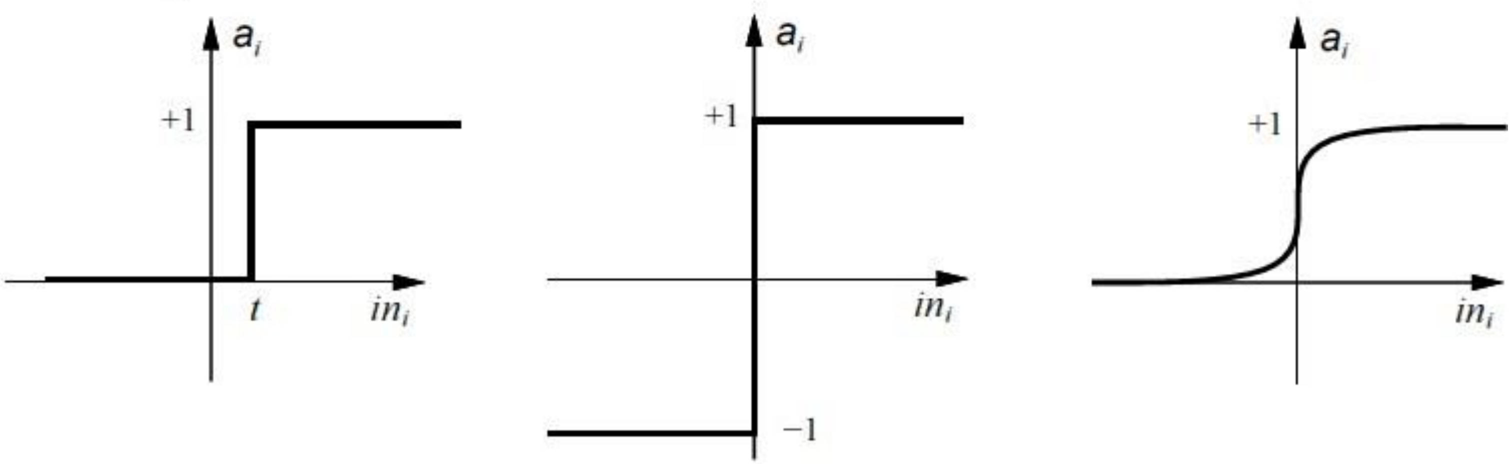


Activation function

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

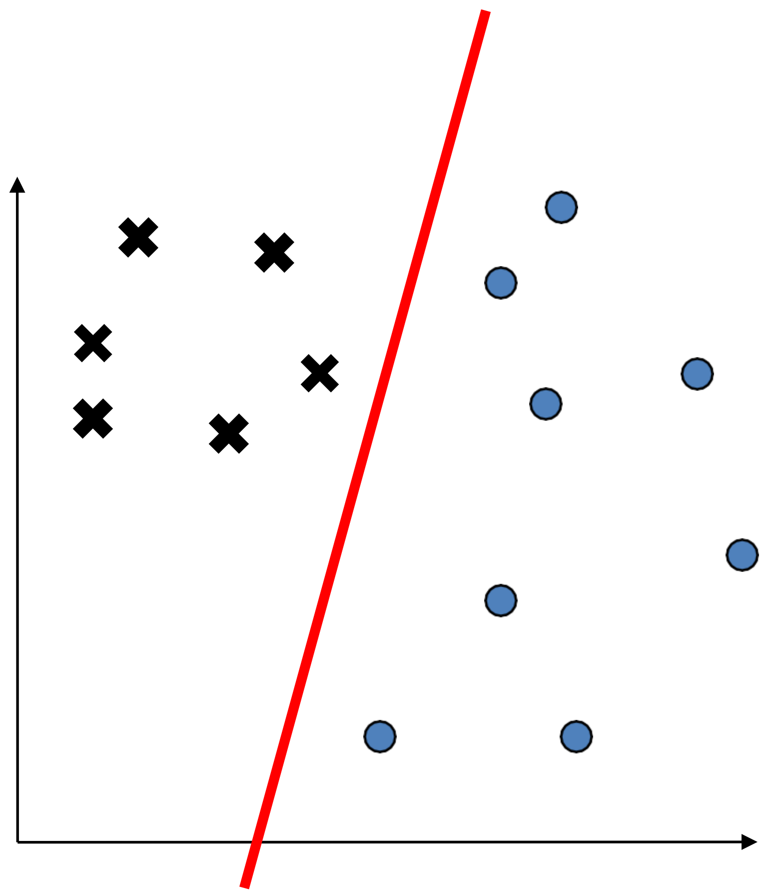
$$a_i := g(in_i)$$

Examples:



(a) Step function (b) Sign function (c) Sigmoid function

A (Linear) Decision Boundary



Represented by:

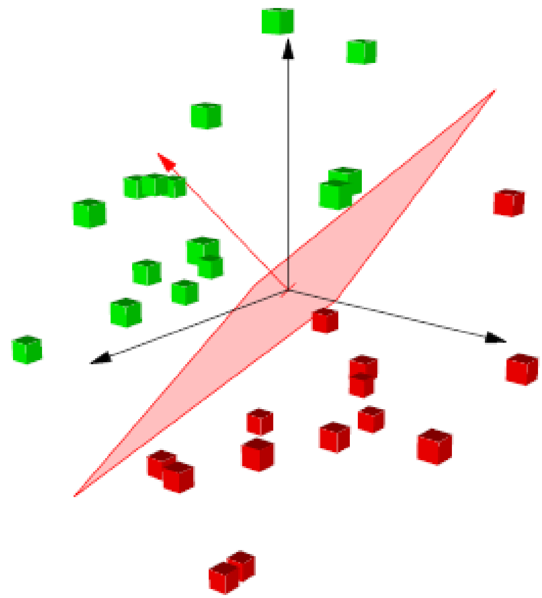
One artificial neuron called a "Perceptron"

Low space complexity

Low time complexity

Perceptron

- The perceptron with a step function performs **classification**
- The perceptron can be ‘visualised’ as a decision boundary in input space



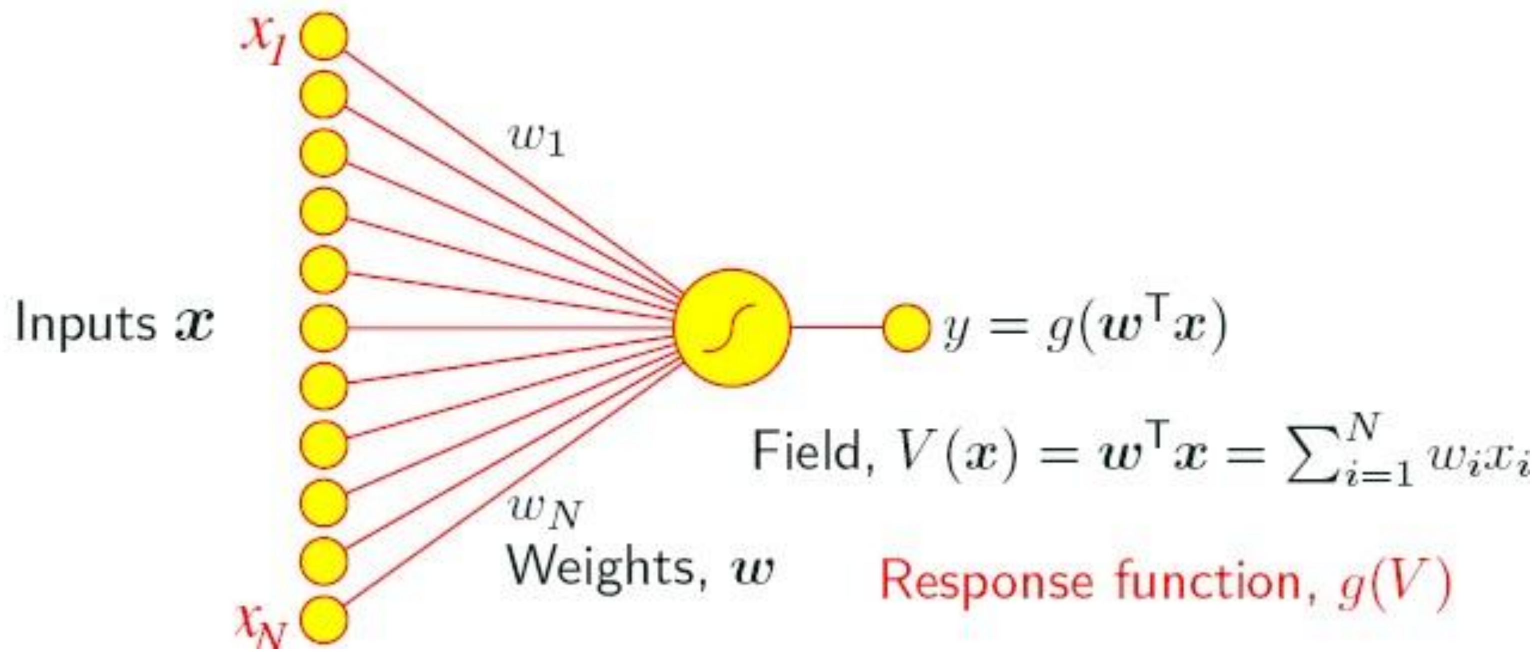
- The perceptron can only separate linear-separable inputs

Perceptron

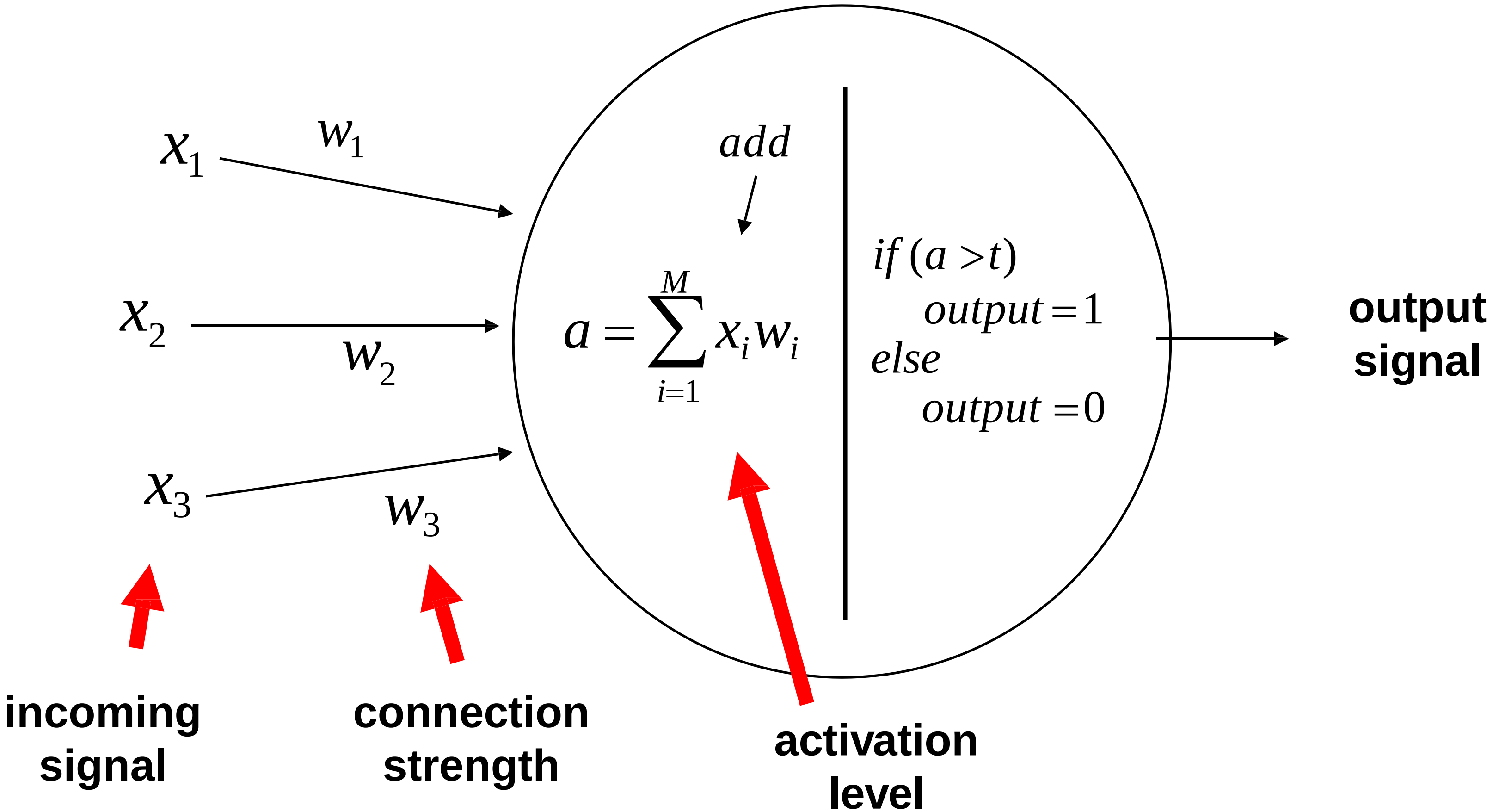
- Given (numeric) input features $x = (x_1, x_2, \ldots, x_n)$
- Prediction given by $f(x; w)$
- w are parameters or "weights" that we train
- The **perceptron** provides the classic example of a **parametric** learning algorithm

Perceptron

- Proposed by Frank Rosenblatt (1958) (Widrow and Hoff proposed **adaline** at same time)
- Schematic representation



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



Calculation…

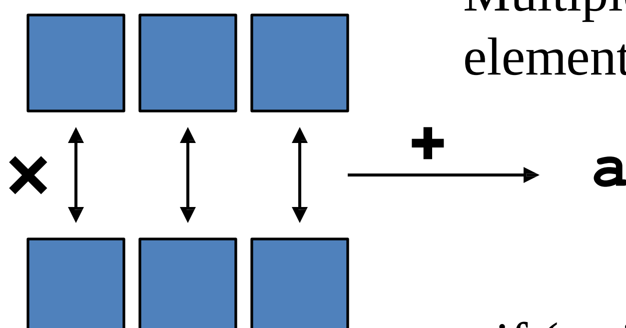
$$a = \sum_{i=1}^{M} x_i w_i$$

Sum notation

(just like a loop from 1 to M)

```
double[] x =

double[] w =
```



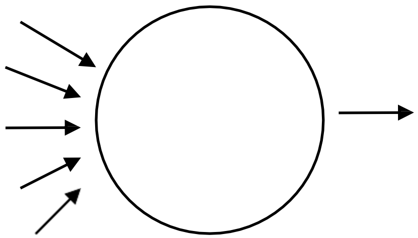
Multiple corresponding elements and add them up

(activation)

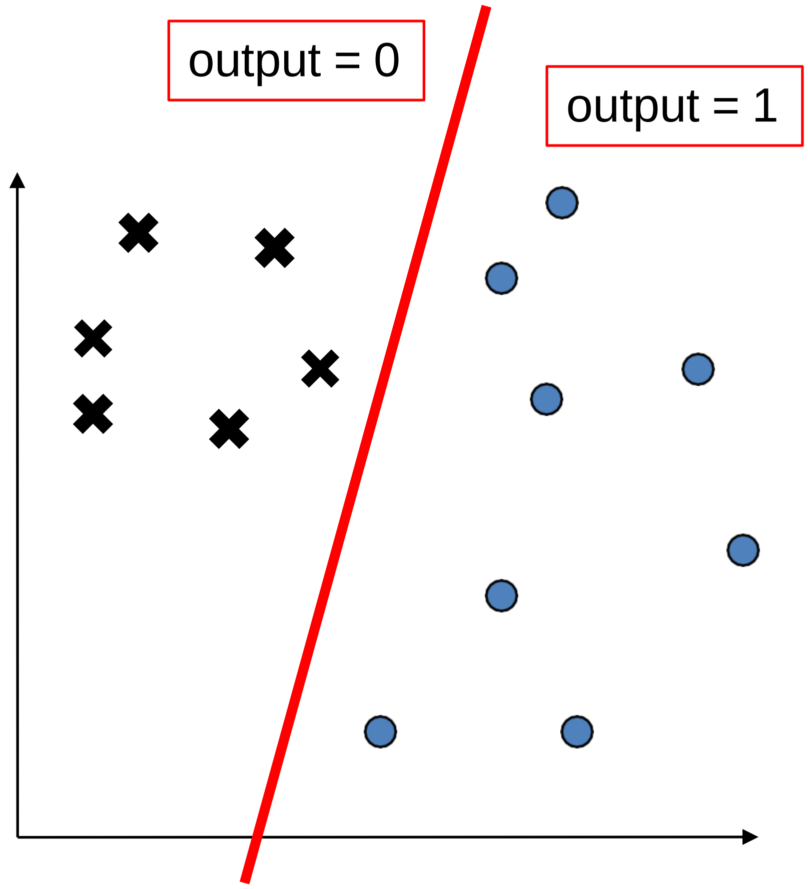
if (activation > threshold) FIRE !

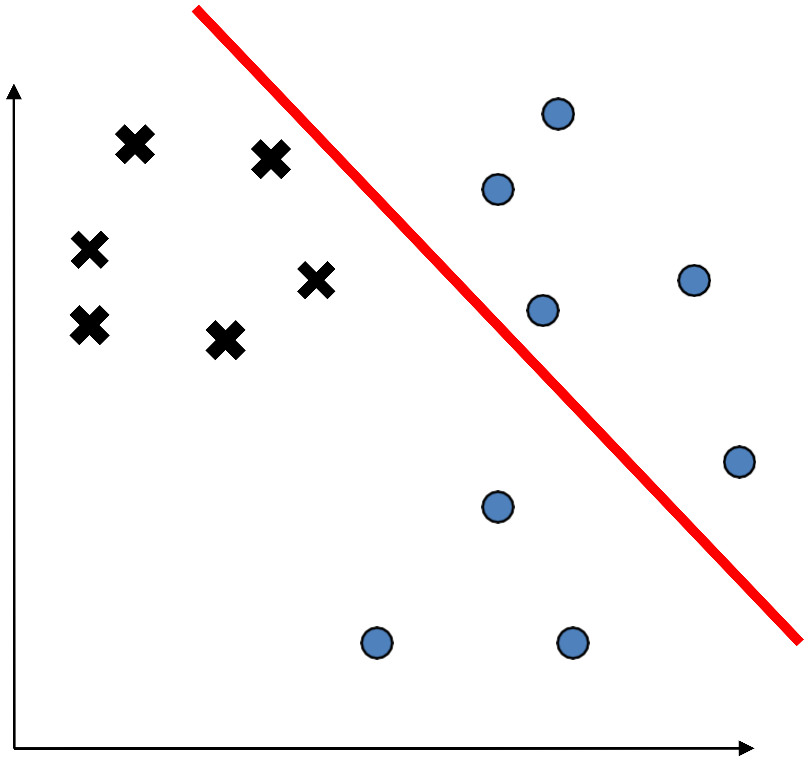
Perceptron Decision Rule

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



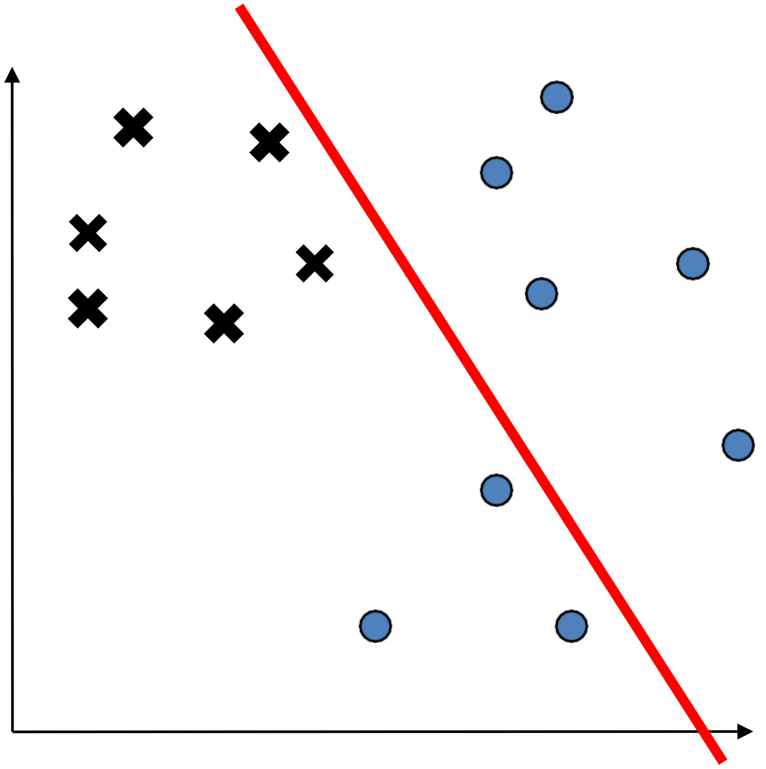
$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \qquad \text{then } output = 1, \text{ else } output = 0$$





Is this a good decision boundary?

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

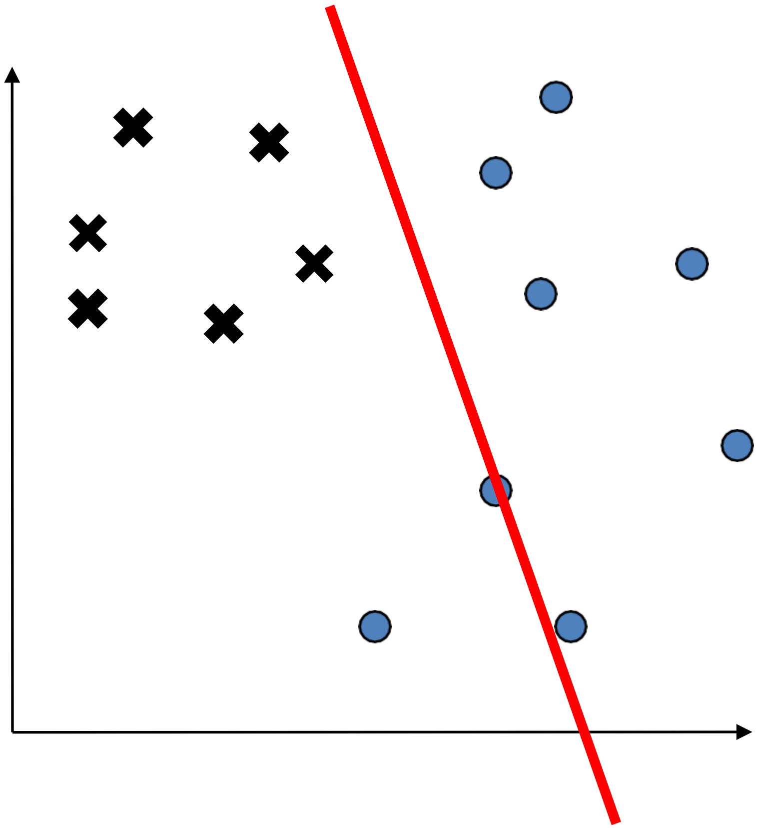


$w_1 = 1.0$

$w_2 = 0.2$

$t = 0.05$

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

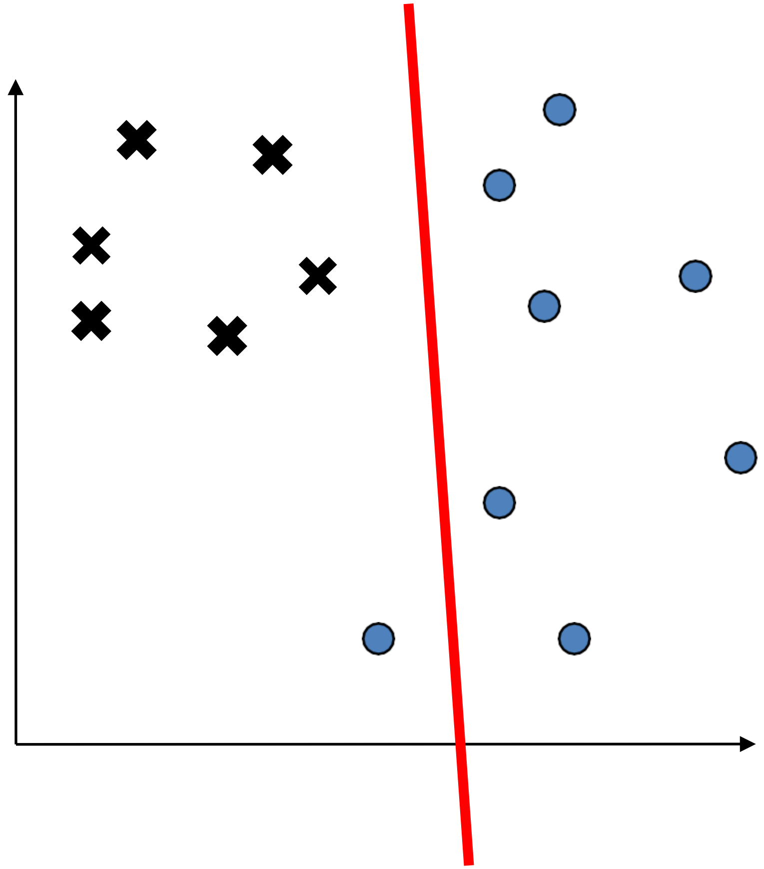


$w_1 = 2.1$

$w_2 = 0.2$

$t = 0.05$

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

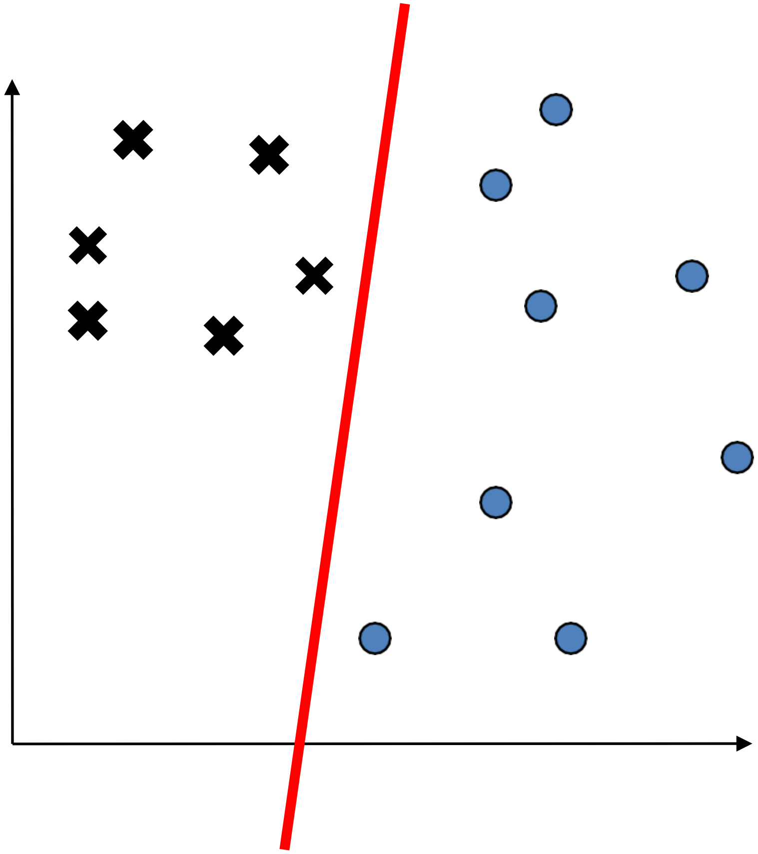


$w_1 = 1.9$

$w_2 = 0.02$

$t = 0.05$

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



$w_1 = -0.8$

$w_2 = 0.03$

$t = 0.05$

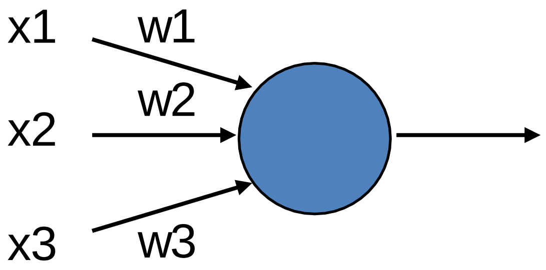
Changing the weights/threshold makes the decision boundary move.

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

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

$$t = 1.0$$

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



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

Q2. Does the neuron fire?

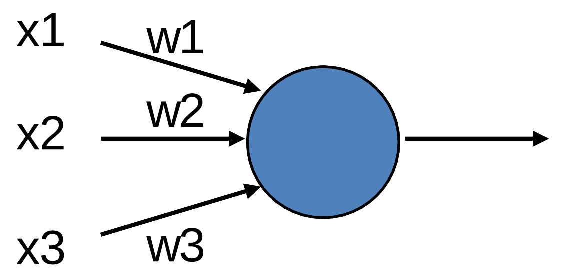
Q3. What if we set threshold at 0.5 and weight #3 to zero?

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

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

$$t = 1.0$$

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



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

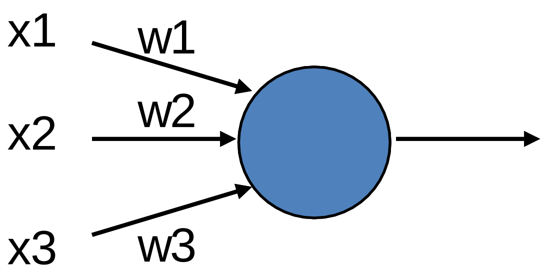
…. So yes, it fires.

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

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

$$t = 1.0$$

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



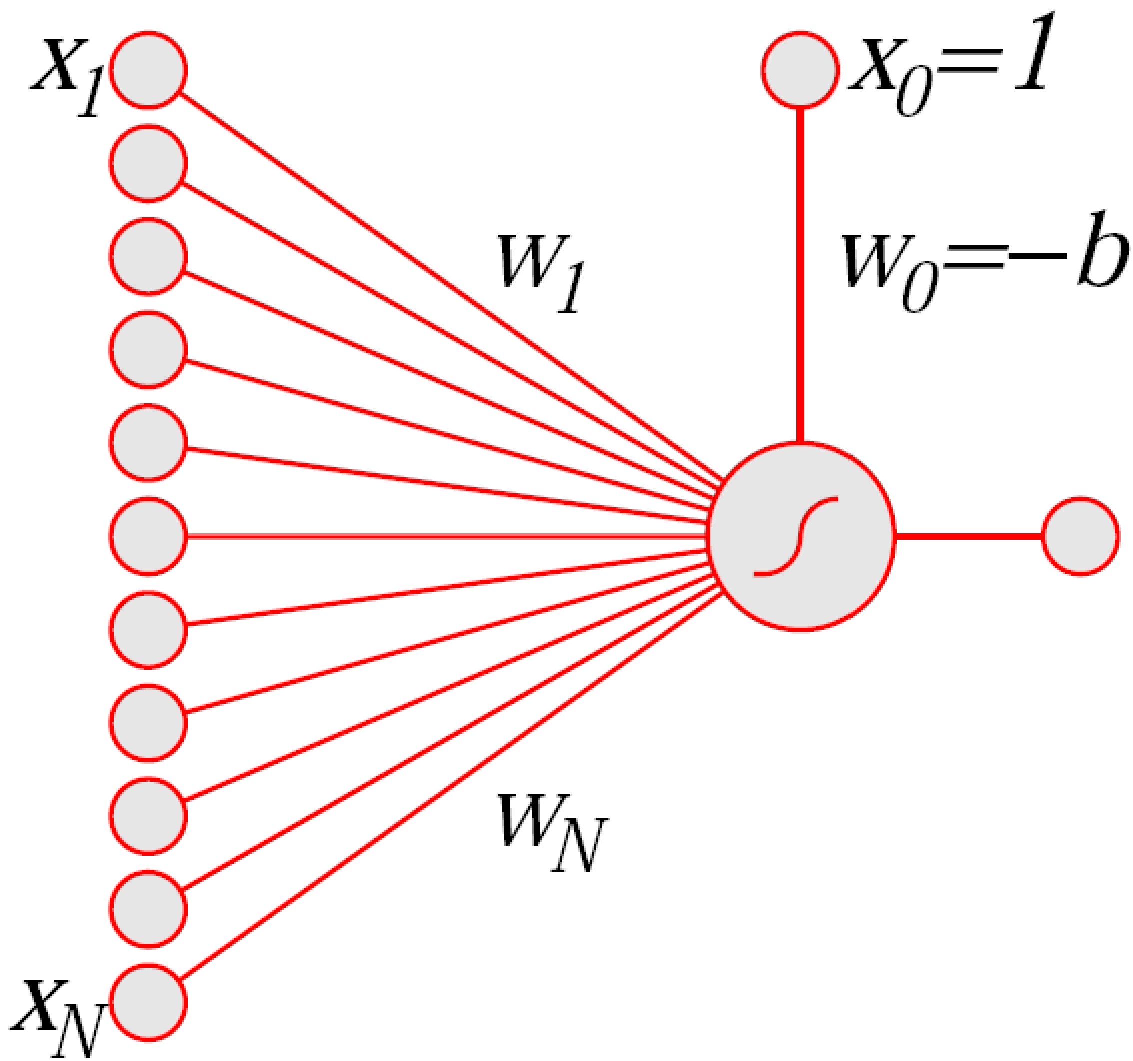
Q3. What if we set threshold at 0.5 and weight #3 to zero?

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

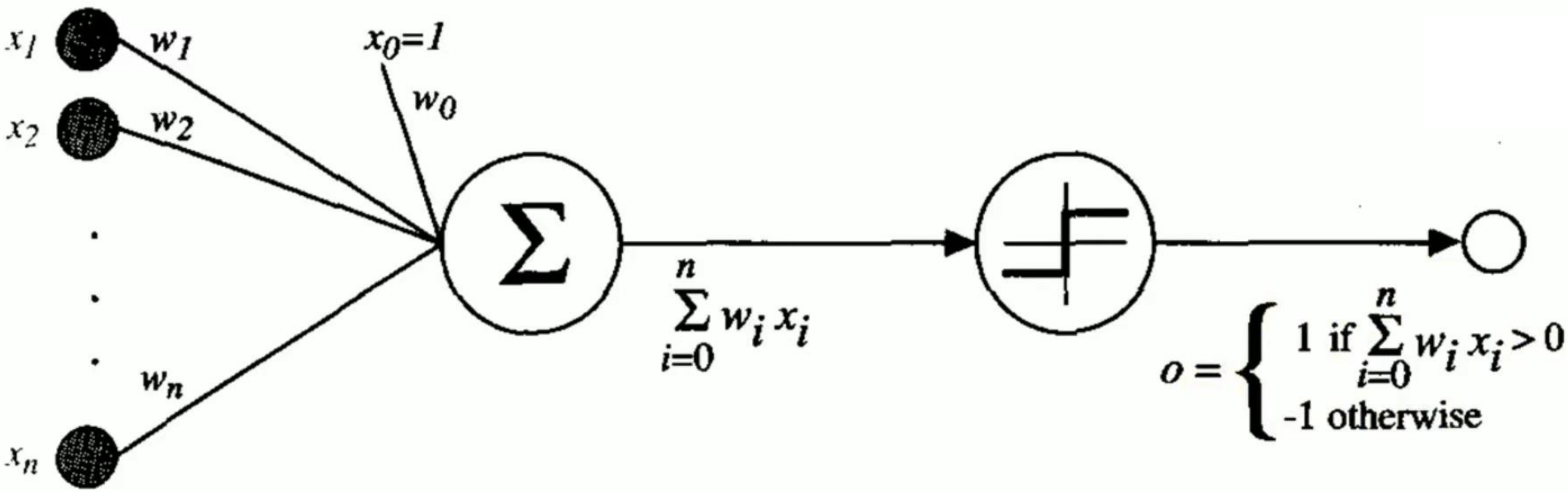
if (activation > threshold) output=1 else output=0

…. So no, it does not fire..

False Input



Perceptron Training Rule



Perceptron Training Rule

- One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the ***perceptron training rule,*** which revises the weight ***wi*** associated with input ***xi*** according to the rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Perceptron Learning Algorithm

Perceptron_training_rule *(X, η)*
initialize **w** *(wi ← an initial (small) random value)*
repeat
    for each training instance (**x**, **tx**) $\in X$
        compute the real output **ox** = *Activation(Summation(***w.x***))*
        if (**tx** $\neq$ **ox**)
            for each wi
                $wi \leftarrow wi + \Delta wi$
                $\Delta wi \leftarrow \eta\,(tx - ox)xi$
            end for
        end if
    end for
until all the training instances in X are correctly classified
return **w**

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

AND Gate-Perceptron Training Rule

w1 = 1.2, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

1. A=0, B=0 and Target = 0
   - $wi.xi = 0*1.2 + 0*0.6 = 0$
   - This is not greater than the threshold of 1, so the output = 0
2. A=0, B=1 and Target = 0
   - $wi.xi = 0*1.2 + 1*0.6 = 0.6$
   - This is not greater than the threshold of 1, so the output = 0

AND Gate-Perceptron Training Rule

w1 = 1.2, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

3. A=1, B=0 and Target = 0
   - *wi.xi = 1*1.2 + 0*0.6 = 1.2*
   - This is greater than the threshold of 1, so the output = 1

$$wi = wi + n(t - o)xi$$

$$w1 = 1.2 + 0.5(0 - 1)1 = 0.7$$

$$w2 = 0.6 + 0.5(0 - 1)0 = 0.6$$

AND Gate-Perceptron Training Rule

w1 = 0.7, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*0.7 + 0*0.6 = 0*
   - This is not greater than the threshold of 1, so the output = 0
2. A=0, B=1 and Target = 0
   - *wi.xi = 0*0.7 + 1*0.6 = 0.6*
   - This is not greater than the threshold of 1, so the output = 0

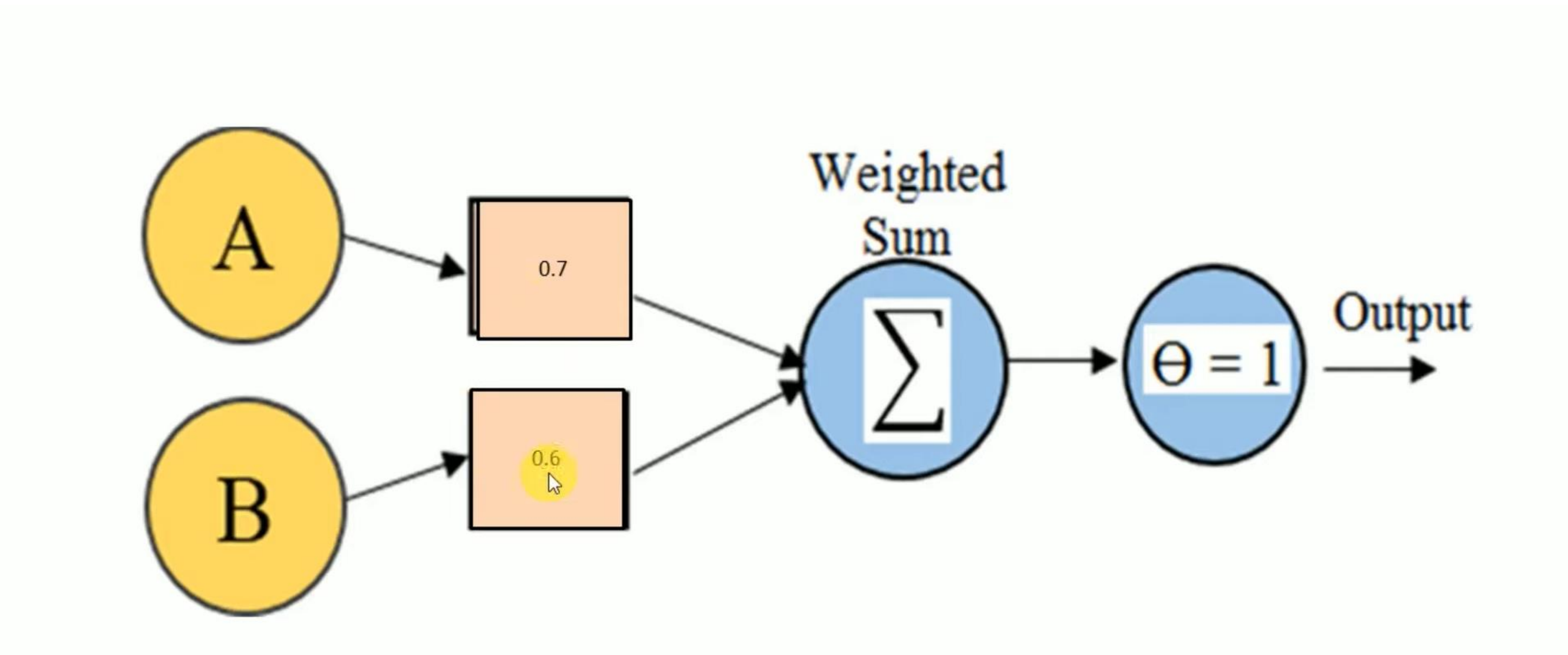
AND Gate-Perceptron Training Rule

w1 = 0.7, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

3. A=1, B=0 and Target = 0
   - *wi.xi = 1*0.7 + 0*0.6 = 0.7*
   - This is not greater than the threshold of 1, so the output = 0
4. A=1, B=1 and Target = 1
   - *wi.xi = 1*0.7 + 1*0.6 = 1.3*
   - This is greater than the threshold of 1, so the output = 1

AND Gate-Perceptron Training Rule



OR Gate-Perceptron Training Rule

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

w1 = 0.6, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*0.6 + 0*0.6 = 0*
   - This is not greater than the threshold of 1, so the output = 0
2. A=0, B=1 and Target = 1
   - *wi.xi = 0*0.6 + 1*0.6 = 0.6*
   - This is not greater than the threshold of 1, so the output = 0

OR Gate-Perceptron Training Rule

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

w1 = 0.6, w2 = 0.6 Threshold = 1 and Learning Rate n = 0.5

2. A=0, B=1 and Target = 1
   - *wi.xi = 0*0.6 + 1*0.6 = 0.6*
   - This is not greater than the threshold of 1, so the output = 0

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)0 = 0.6$$

$$w2 = 0.6 + 0.5(1 - 0)1 = 1.1$$

OR Gate-Perceptron Training Rule

w1 = 0.6, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

1. A=0, B=0 and Target = 0
   - *wi.xi = 0*0.6 + 0*1.1 = 0*
   - This is not greater than the threshold of 1, so the output = 0
2. A=0, B=1 and Target = 1
   - *wi.xi = 0*0.6 + 1*1.1 = 1.1*
   - This is greater than the threshold of 1, so the output = 1

OR Gate-Perceptron Training Rule

w1 = 0.6, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

3. A=1, B=0 and Target = 1
   - $wi.xi = 1*0.6 + 0*1.1 = 0.6$
   - This is not greater than the threshold of 1, so the output = 0

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)1 = 1.1$$

$$w2 = 1.1 + 0.5(1 - 0)0 = 1.1$$

OR Gate-Perceptron Training Rule

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

w1 = 1.1, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5

1. A=0, B=0 and Target = 0
   - $wi.xi = 0*1.1 + 0*1.1 = 0$
   - This is not greater than the threshold of 1, so the output = 0
2. A=0, B=1 and Target = 1
   - $wi.xi = 0*1.1 + 1*1.1 = 1.1$
   - This is greater than the threshold of 1, so the output = 1

OR Gate-Perceptron Training Rule

w1 = 1.1, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5

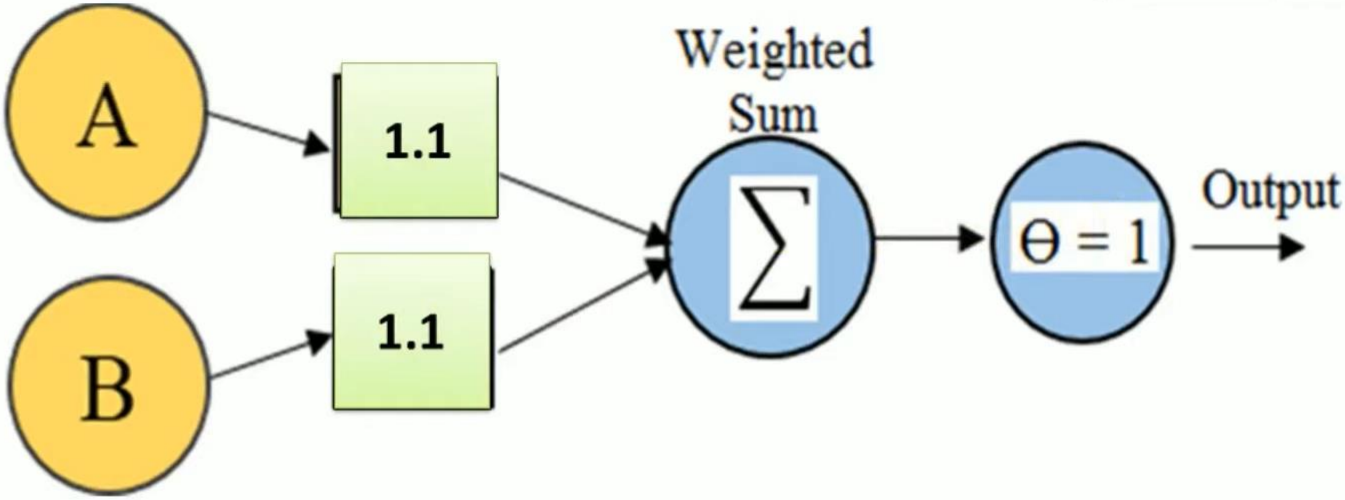
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

3. A=1, B=0 and Target = 1
   - *wi.xi = 1*1.1 + 0*1.1 = 1.1*
   - This is greater than the threshold of 1, so the output = 1
4. A=1, B=1 and Target = 1
   - *wi.xi = 1*1.1 + 1*1.1 = 2.2*
   - This is greater than the threshold of 1, so the output = 1

OR Gate-Perceptron Training Rule

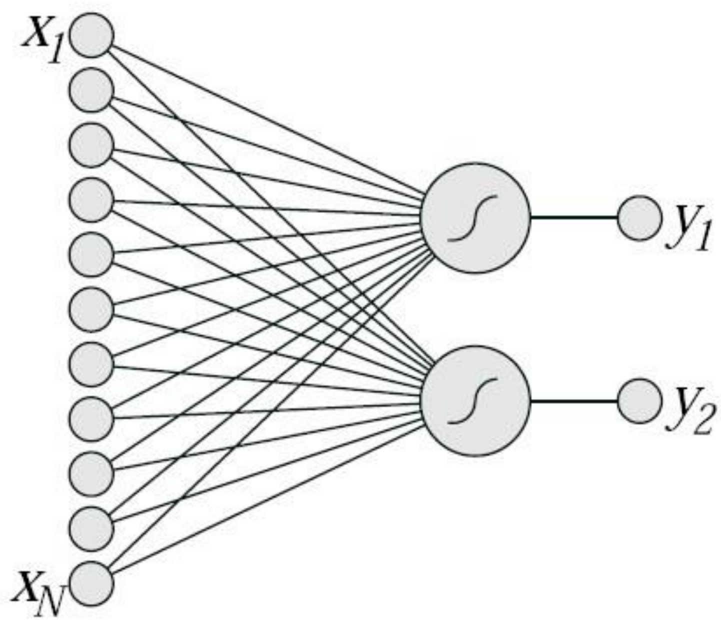
w1 = 1.1, w2 = 1.1 Threshold = 1 and Learning Rate n = 0.5

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1



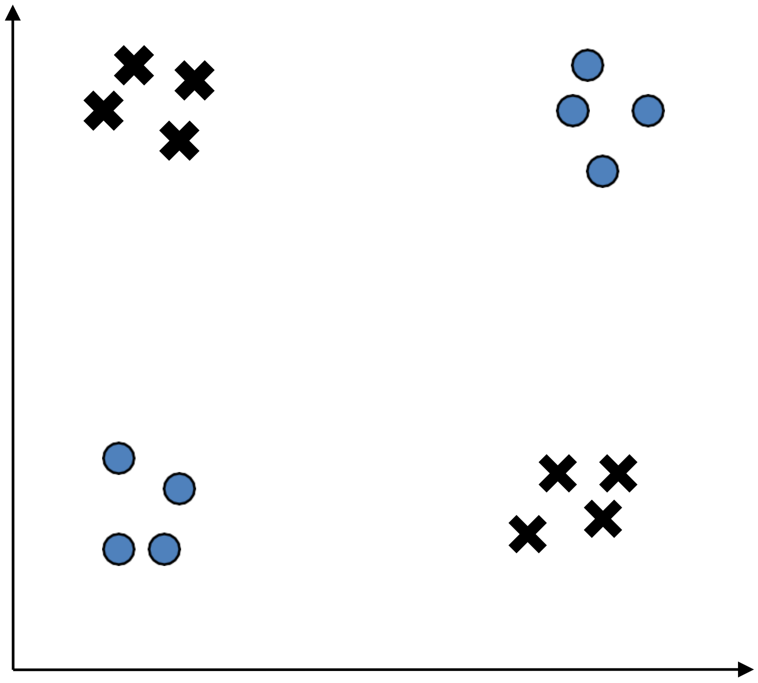
Multiple Outputs

- We can produce more complicated machines by using several perceptrons

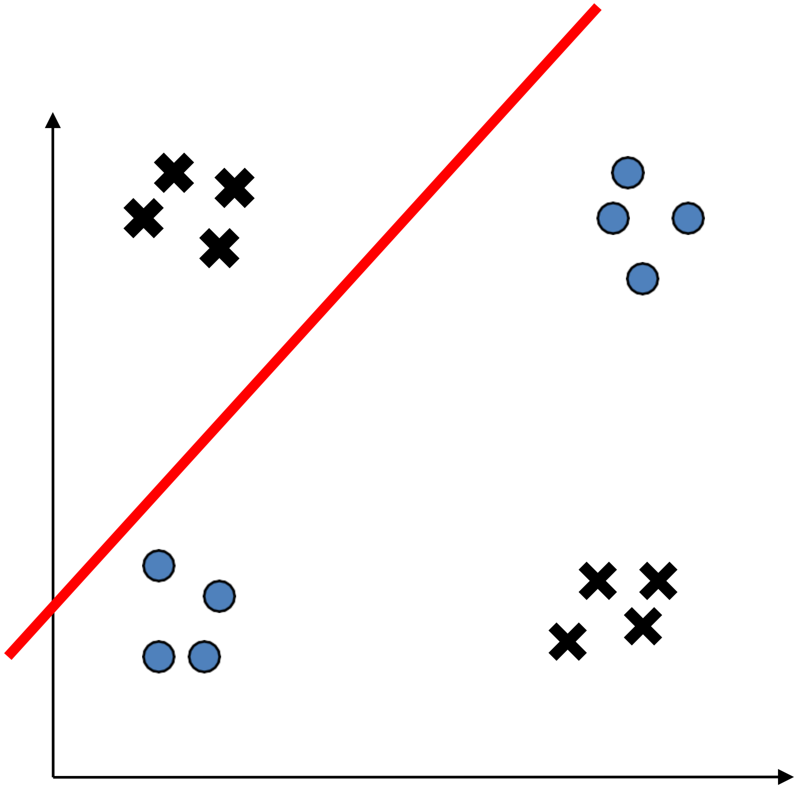


- Treat each perceptron independently
- Consider just single perceptron

Can a Perceptron solve this problem?



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

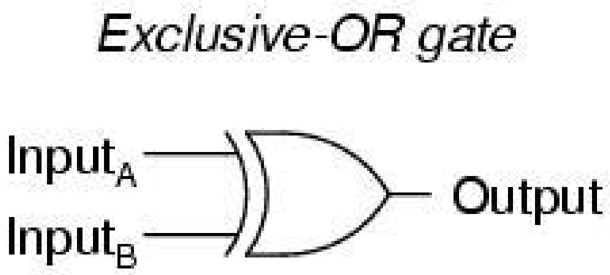


Perceptrons only solve

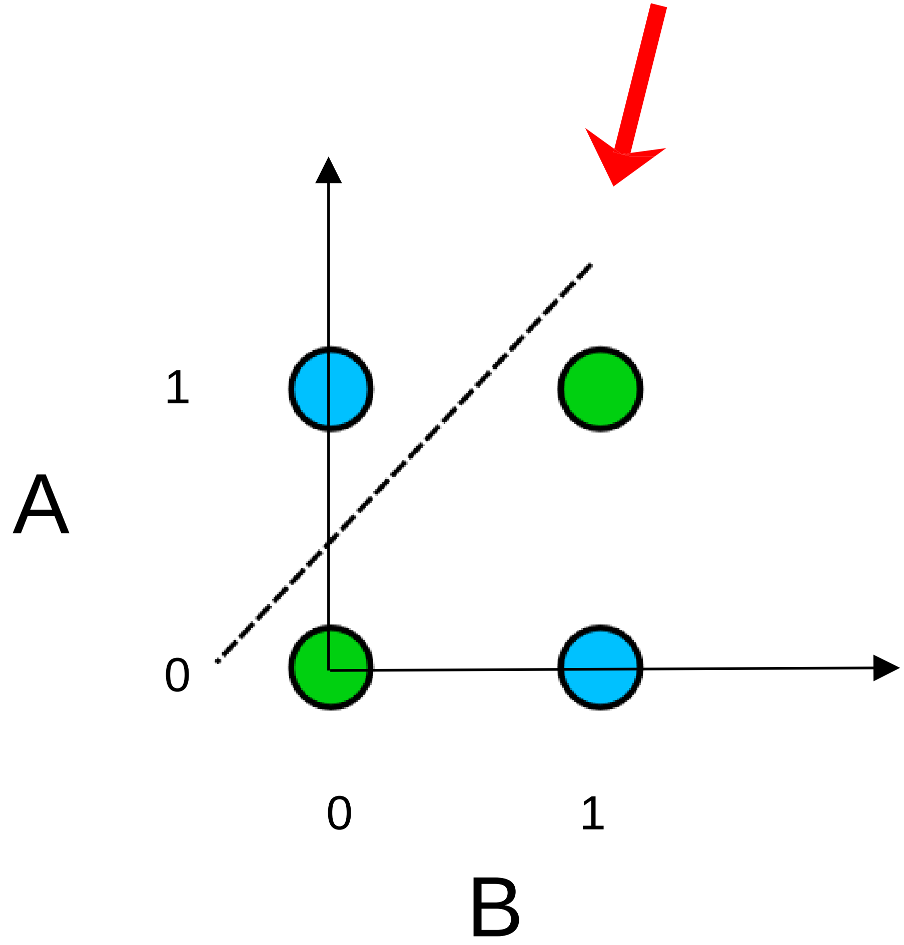
LINEARLY SEPARABLE

problems

With a perceptron…
the decision boundary is
<u>LINEAR</u>



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



MultiLayer Perceptron (MLP)

Motivation

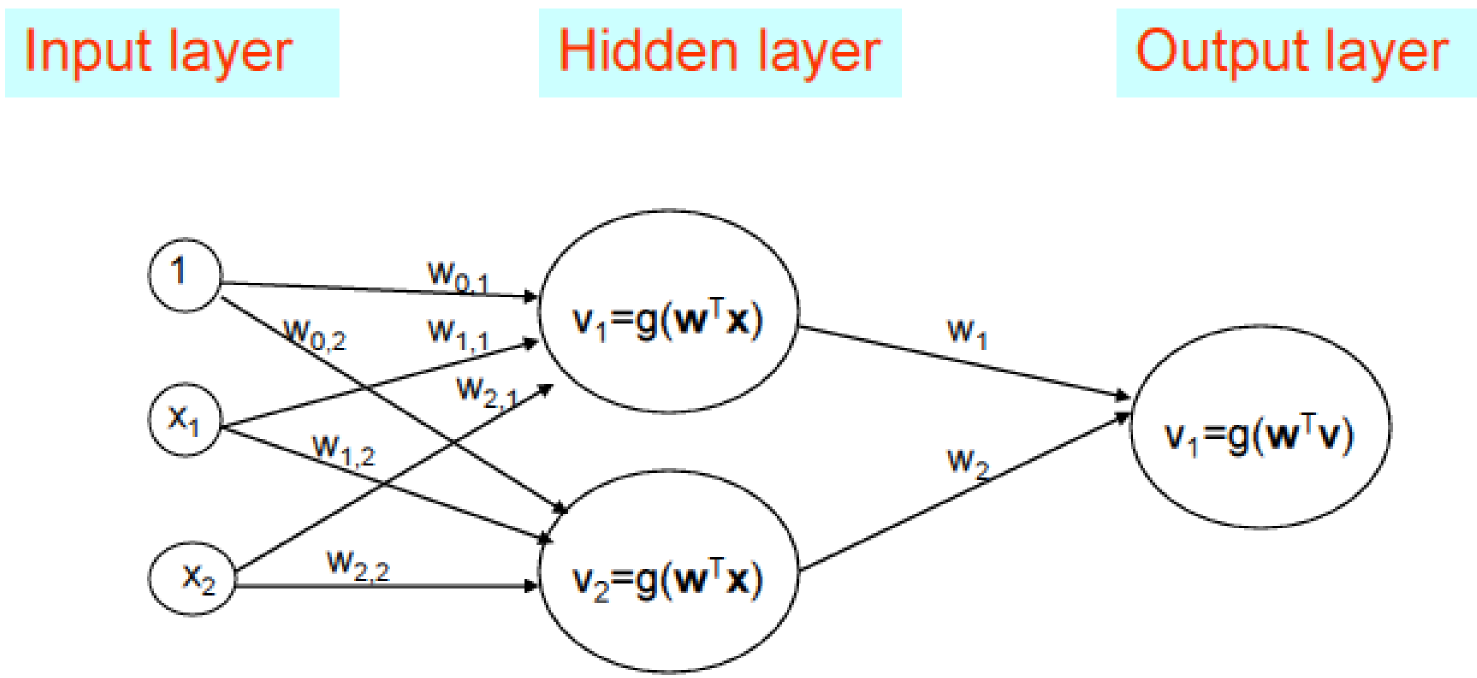
- Perceptrons are limited because they can only solve problems that are linearly separable
- We would like to build more complicated learning machines to model our data
- One way to do this is to build a multiple layers of perceptrons

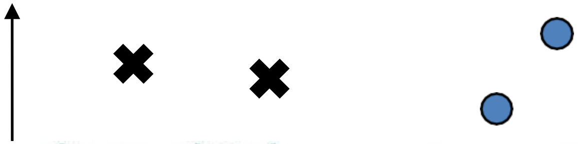
Brief History

- 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

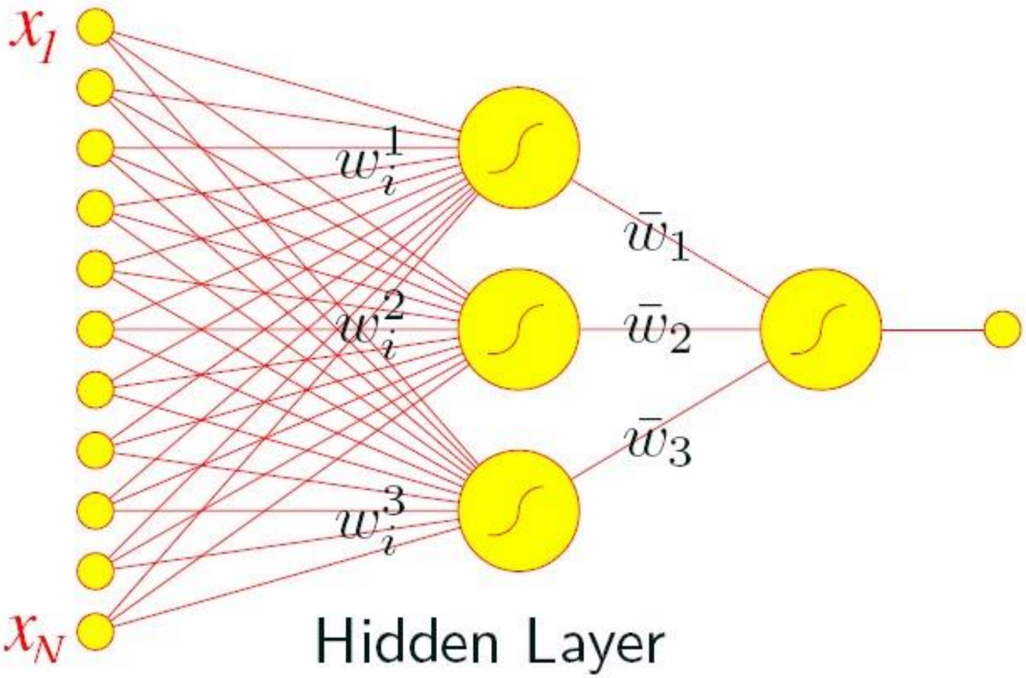
- So far we discussed networks with one layer.
- But these networks can be extended to combine several layers, increasing the set of functions that can be represented using a NN





MLP

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



$$z = f(\boldsymbol{x}|\boldsymbol{w})$$

$$= g\left(\sum_i \bar{w}_i \, g(\boldsymbol{x}^\top \boldsymbol{w}^i)\right)$$

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

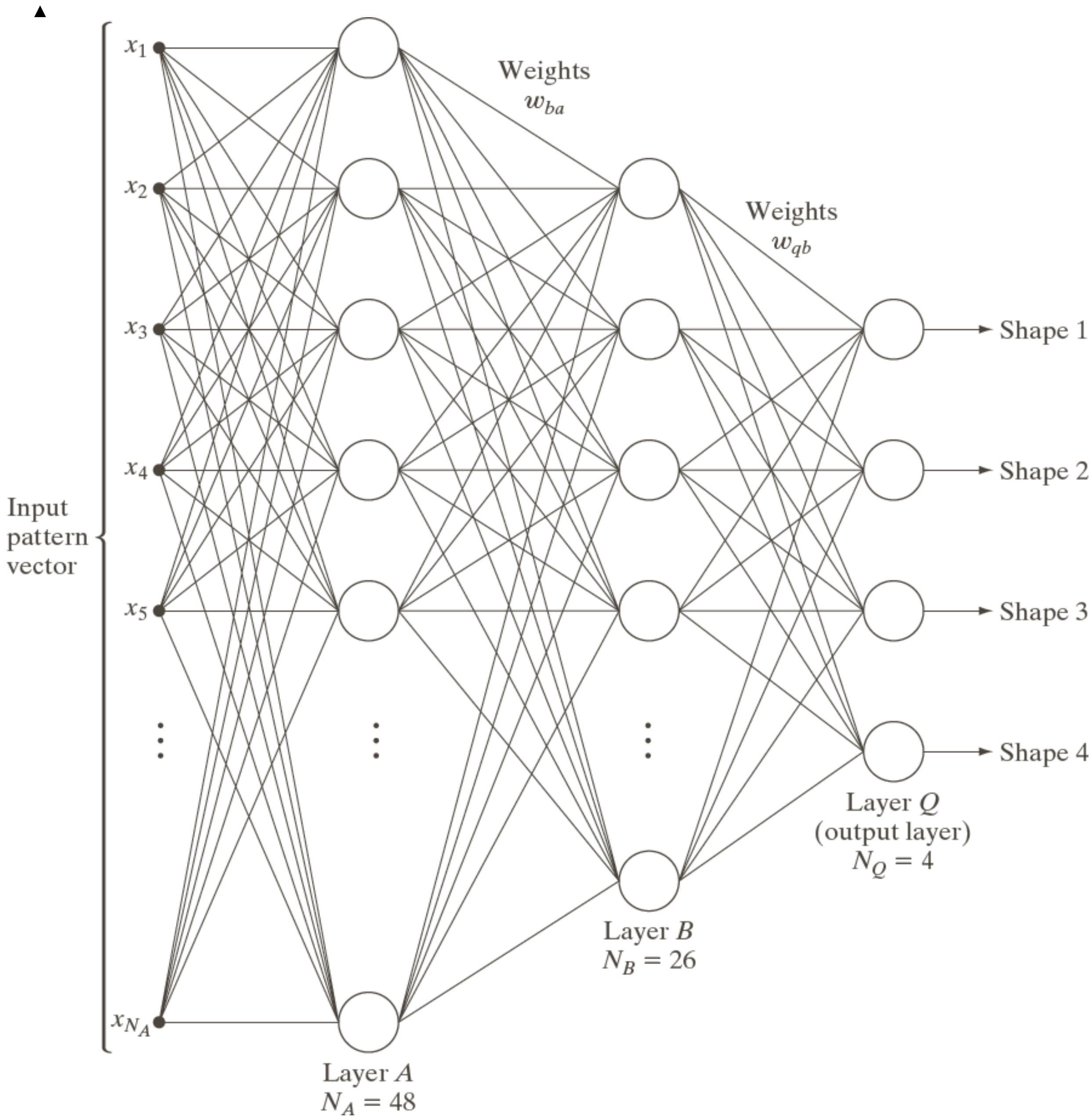


FIGURE 12.19 Three-layer neural network used to recognize the shapes in Fig. 12.18. (Courtesy of Dr. Lalit Gupta, ECE Department, Southern Illinois University.)

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

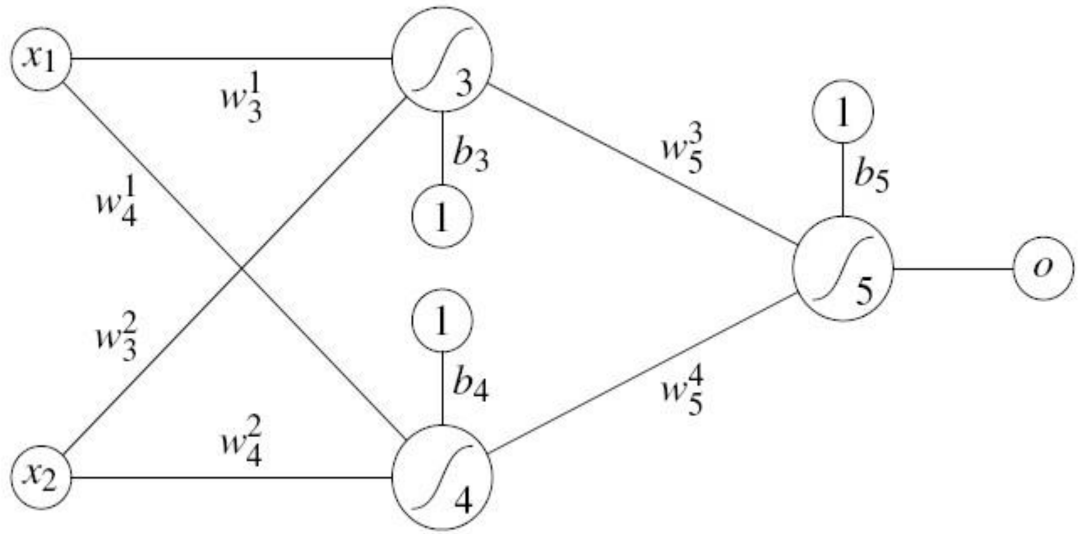
$$g(V) = \frac{1}{1 + \mathrm{e}^{-(V)}}$$

- A tanh function

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

MLP

- A diagram for an neural network such as an MLP

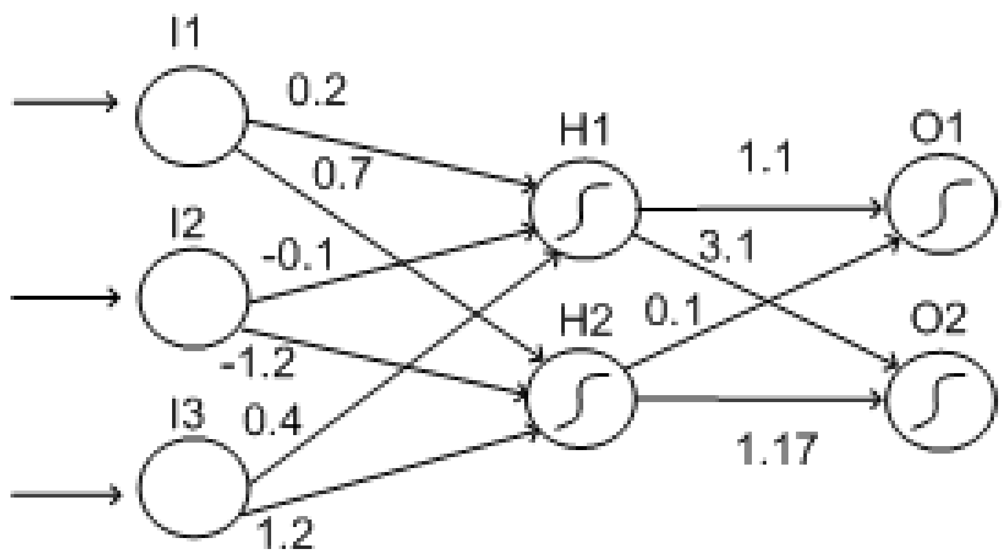


- Stands for the function $(o = f(\boldsymbol{x}|\boldsymbol{w}))$

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

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

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.**

Training Parametric Model

- We have seen how to train single layer step perceptron and linear perceptrons
- But how do we train multilayer non-linear perceptrons?
- More generally how do we train any parametric model $\boldsymbol{f}(\boldsymbol{x}|\boldsymbol{w})$?
- Given data $\mathcal{D} = \left\{(\boldsymbol{x}_k, \boldsymbol{y}_k)\right\}_{k=1}^{P}$ we can seek to minimise the mean squared error

$$E(\boldsymbol{w}|\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x}_k, y_k) \in \mathcal{D}} \big(y_k - f(\boldsymbol{x}_k|\boldsymbol{w})\big)^2$$

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.

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