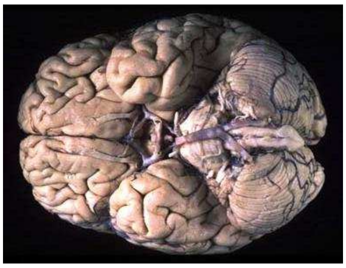
(Artificial) Neural Networks

The real thing

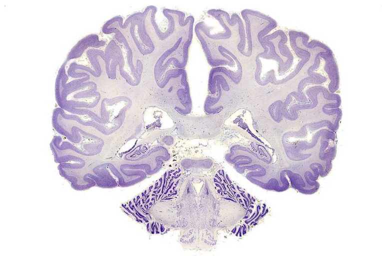
- **Brain with vessels:**



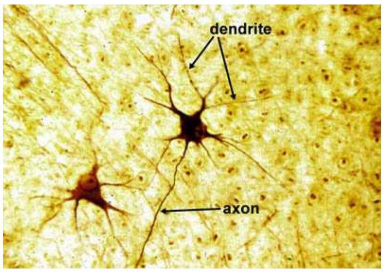
- **Motor and sensory neural pathways:**

Neuronal Tracts and Cells

- **Microscopic cross section brain:**



- **Individual neuron (nerve cell):**

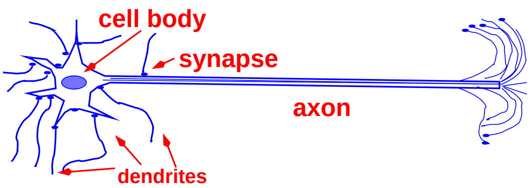


Computational Properties of the Brain

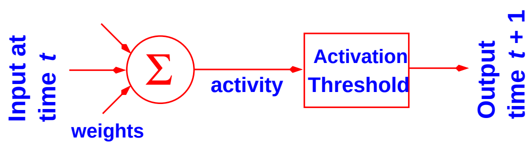
- *Content addressability:* finding information by activating relevant units (in parallel)
- *Graceful degradation:* reduction in the number of known features yields a *gradual* decrease in quality of the response
- *Default assignment:* assuming certain properties in the absence of information, using analogies
- *Spontaneous generalisation*: abstraction from specific characteristics
- *Robustness*: the brain may still be functioning reasonably well despite considerable damage

Artificial Neuron

- **Schema biological neuron:**



- **Schema artificial neuron:**



  - Σ: summation of weighed inputs
  - *Activation threshold:* produce only output when activity is above a threshold

Some Useful Maths Notations

- Vector:

$$\mathbf{v} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- The transpose of a vector $\mathbf{v}$: $\mathbf{v}^T$:

$$\mathbf{v}^T = [x_1\ x_2\ \cdots\ x_n]$$

if

$$\mathbf{v} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- Inner product of two vectors $\mathbf{v}$ and $\mathbf{w}$:

$$\begin{aligned} \mathbf{v}^T\mathbf{w} &= [x_1\ x_2\ \cdots\ x_n] \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \\ &= x_1y_1 + x_2y_2 + \cdots + x_ny_n \end{aligned}$$

Note: $\mathbf{v}^T\mathbf{w} = 0$ if $\mathbf{v} \perp \mathbf{w}$

Some Maths Notations (continued)

- $p \times n$ Matrix:

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ m_{p,1} & m_{p,2} & \cdots & m_{p,n} \end{bmatrix}$$

- Product of matrix and vector:

$$M\mathbf{v} = \begin{bmatrix} m_{1,1}x_1 + m_{1,2}x_2 + \cdots + m_{1,n}x_n \\ m_{2,1}x_1 + m_{2,2}x_2 + \cdots + m_{2,n}x_n \\ \vdots \\ m_{p,1}x_1 + m_{p,2}x_2 + \cdots + m_{p,n}x_n \end{bmatrix}$$

- Partial differentiation:

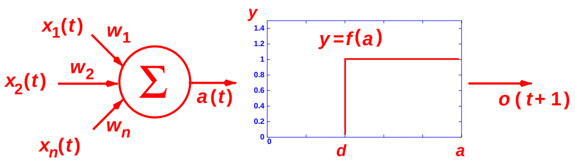
$$f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

with for example $f(x, y) = x^2 + 2xy + y^2$, then:

$$\frac{\partial f}{\partial x}(x, y) = 2x + 2y$$

$$\frac{\partial f}{\partial y}(x, y) = 2x + 2y$$

Artificial Neuron of McCulloch and Pitts



- *Threshold function*: $f : \mathbb{Z} \to \mathbb{Z}$, with $y = f(a)$
- *Activity* at time t:

$$\begin{aligned} a(t) &= w_1 \cdot x_1(t) + w_2 \cdot x_2(t) + \cdots + w_n \cdot x_n(t) \\ &= \textstyle\sum_{i=1}^{n} w_i \cdot x_i(t) \\ &= \mathbf{w}^T\mathbf{x}(t) \end{aligned}$$

where $\mathbf{w}^T = [w_1 w_2 \cdots w_n]$, and

$$\mathbf{x}(t)^T = [x_1(t)x_2(t)\cdots x_n(t)]$$

are (the transposes of) vectors

- $\boxed{o(t+1) = f(a(t)) = f(\mathbf{w}^T\mathbf{x}(t))}$

Example of Activation Function

(ignoring time)

$$o = f(\mathbf{w}^T\mathbf{x})$$

with

$$f(a) = \begin{cases} 1 & \text{if } a \geq d \\ 0 & \text{otherwise} \end{cases}$$

for a given threshold value $d \in \mathbb{Z}$.

Modelling of **logical AND** with $\mathbf{w}^T = [1\,1]$ and $\mathbf{x}^T = [x_1\ x_2]$:

x_1	x_2	$x_1 \wedge x_2$	$\mathbf{w}^T\mathbf{x}$
1	1	1	$[1\ 1]\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2$
1	0	0	$[1\ 1]\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$
0	1	0	$[1\ 1]\begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$
0	0	0	$[1\ 1]\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$

Conclusion: **choose** $d = 2$ (*linearly separable*)

Example of Activation Function

(ignoring time)

$$o = f(\mathbf{w}^T \mathbf{x})$$

with

$$f(a) = \begin{cases} 1 & \text{if } a \geq d \\ 0 & \text{otherwise} \end{cases}$$

for a given threshold value $d \in \mathbb{Z}$.

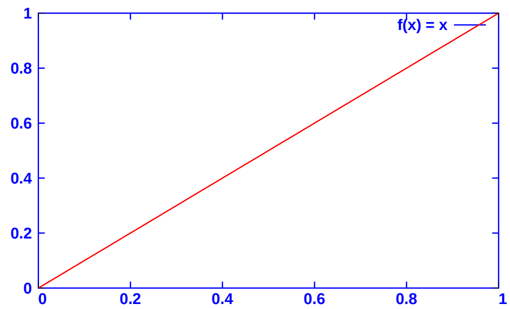
Modelling of **logical OR** with $\mathbf{w}^T = [1\ 1]$ and $\mathbf{x}^T = [x_1\ x_2]$:

x_1	x_2	$x_1 \vee x_2$	$\mathbf{w}^T \mathbf{x}$
1	1	1	$[1\ 1]\begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2$
1	0	1	$[1\ 1]\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1$
0	1	1	$[1\ 1]\begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$
0	0	0	$[1\ 1]\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$

Conclusion: **choose $d = 1$**

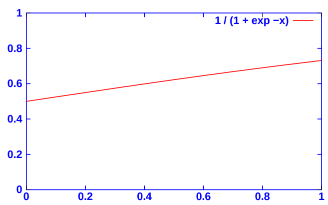
Generalisation of the McCulloch and Pitts' Neuron

- Continuous (instead of discrete) input and output values, i.e. $\mathbf{x} \in \mathbb{R}^n$ and $o(t+1) \in \mathbb{R}$
- Activation function: $f : \mathbb{R} \to \mathbb{R}$
- Typical example: $f : \mathbb{R} \to [1, 0]$, with $f(a) = a$ (identity; what does it do?)



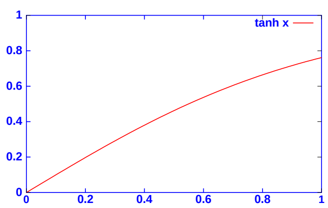
Other Continuous Activation Functions

- Logistic sigmoid function: $f : \mathbb{R} \to [0, 1]$:



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

- Hyperbolic tangent:

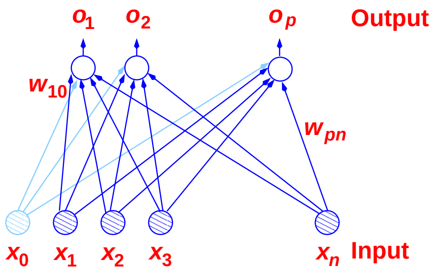


$$f(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

The Learning Problem

- Normally, the weights are not known, and must be learnt from a dataset with problem cases:
  - *Supervised learning* – target output known for each case – aim: to learn the relationship between input pattern and output as well as possible;
  - *Unsupervised learning* – target output unknown – aim: to discover correlations and similarities among the input patterns.
- If the activation function is *differentiable*, we can analyse the behaviour of the learning strategy

Perceptron

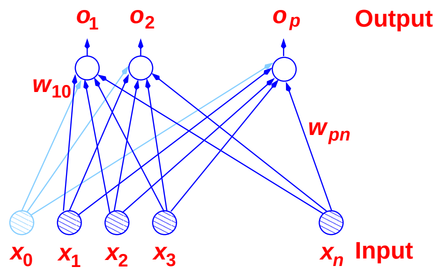


- Perceptron = one-layer feedforward neural network
- One activation function: $f : \mathbb{R} \to \mathbb{R}$
- Input vector $\mathbf{x}^T = [x_0 \cdots x_n]^T$ is transformed into output vector $\mathbf{o}^T = [o_1 \cdots o_p]^T$: $o_i = f(a_i)$, $i = 1, \ldots, p$, where

$$a_i = \sum_{k=0}^{n} w_{ik} x_k$$

with f activation function, and w_{ik} weights

Perceptron: Weight Matrix



Note that:

$$a_i = \sum_{k=0}^{n} w_{ik} x_k$$

for $i = 1, \ldots, p$, and $x_0 = 1$ (used to learn threshold) can also be written compactly as:

$$\begin{aligned} \mathbf{a} &= \begin{bmatrix} w_{10} & w_{11} & w_{12} & \cdots & w_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{p0} & w_{p1} & w_{p2} & \cdots & w_{pn} \end{bmatrix} \mathbf{x} \\ &= W\mathbf{x} \end{aligned}$$

with W the *weight matrix*

Perceptron Learning

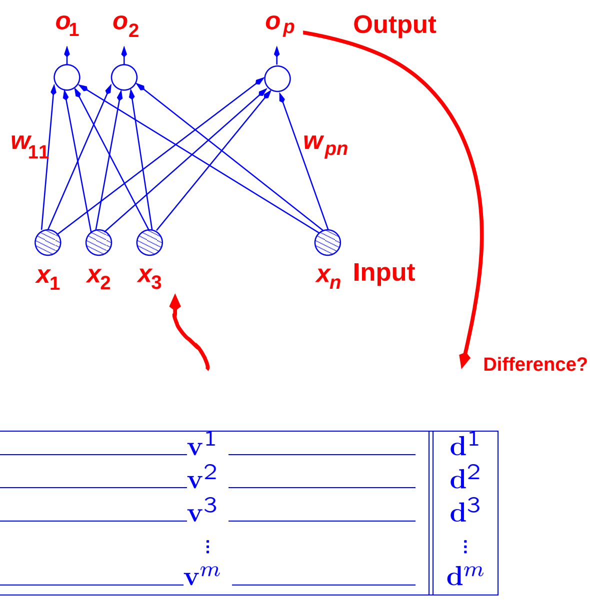
General approach:

- Given a dataset T (*training set*) consisting of m *training examples*:

$$T = \{(\mathbf{v}^q, \mathbf{d}^q) \mid q = 1, \ldots, m\}$$

  where:
  - $\mathbf{v}^q$: input vector
  - $\mathbf{d}^q$: target pattern, i.e. pattern that must be learnt
- Perceptron output vector $\mathbf{o}^q$ for input vector $\mathbf{v}^q$, $q = 1, \ldots, m$
- Goal: to find weight matrix W, such that output vector $\mathbf{o}^q$ is closest to target pattern $\mathbf{v}^q$, for each example q with $1 \leq q \leq m$
- Result: optimal weight matrix $W^\star$

Perceptron Learning: Schematic



Dataset

When is Output Closest to Target Patterns?

- *Error measure* $E(W)$, with W weight matrix; measure of how close $\mathbf{o}^q$ to $\mathbf{d}^q$, for $q = 1, \ldots, m$

- Possible definition – *sum-of-squares error*:

$$E(W) = \sum_{q=1}^{m} E^q(W)$$

where $E^q(W)$ is the error between output $\mathbf{o}^q$ and target $\mathbf{d}^q$, i.e.

$$E^q(W) = \frac{1}{2} \sum_{i=1}^{p} (o_i^q - d_i^q)^2$$

(Note that $E^q(W) = 0$ if $\mathbf{o}^q = \mathbf{d}^q$)

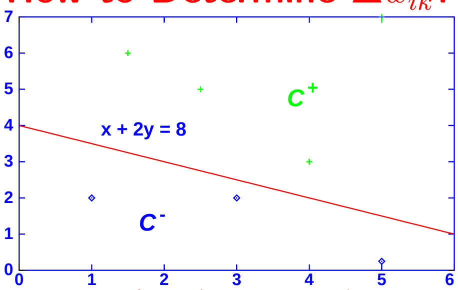
Incremental Perceptron Learning Algorithm

for $r \leftarrow 1, 2, \ldots$ **do** // iteration
  for $q \leftarrow 1$ **to** m **do** // example
    for $i \leftarrow 1$ **to** p **do** // output
      for $k \leftarrow 1$ **to** n **do** // input
        $w_{ik}^{(r+1)} \leftarrow w_{ik}^{(r)} + \Delta w_{ik}^{(r)}$
      od
    od
  od
until $\Delta W = \mathbf{O}$

Δw_{ik}: change in the direction of the minimum of $E(W)$, i.e. make $E(W)$ as small as possible

From now on, we assume that we only have a single output node, i.e. $\mathbf{o} = [o]$, *and* W *is a vector* $\mathbf{w}$

How to Determine Δw_{ik}?



Patterns: *negative class examples:*

$$C^- = \{(1, 2), (3, 2), (5, 0.25)\}$$

and *positive class examples:*

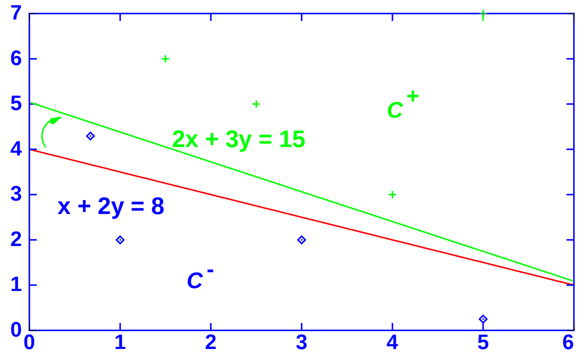
$$C^+ = \{(1.5, 6), (2.5, 5), (4, 3), (5, 7)\}$$

are clearly separated from each other by the decision line $x + 2y = 8$.
Note that $-8 + x + 2y = [-8, 1, 2][1\, x\, y]^T = \mathbf{w}^T\mathbf{x}$

Examples:

- For $(1, 2) \in C^-$: $-8 + 1 + 4 = -3 < 0$
- For $(1.5, 6) \in C^+$: $-8 + 1.5 + 12 = 5.5 > 0$

How to Determine Δw_{ik}?



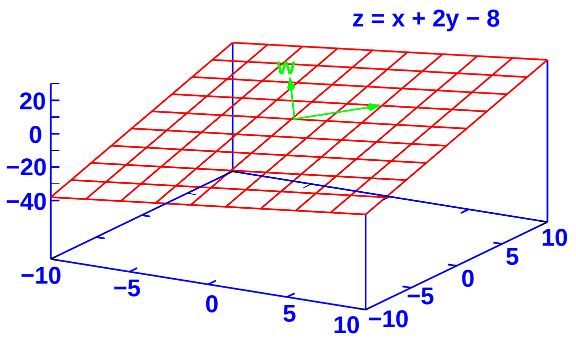
- New negative pattern: $(x, y) = (0.8, 4.3) \in C^-$, but $x + 2y - 8 > 0$ (*misclassified* by $\mathbf{w}$)

- Solution: *modify* weight vector $\mathbf{w}$, e.g.

$$\mathbf{w} = [-8, 1, 2] \Rightarrow \mathbf{w}' = [-15, 2, 3]$$

i.e. the decision line is now $2x + 3y = 15$

Geometry of Decision Hyperplane



$$\mathbf{w}^T\mathbf{x} = [-8, 1, 2][1\ x\ y]^T = 0$$

- is the intersection of the plane $z = x + 2y - 8$ with the $z = 0$ plane
- modifying $\mathbf{w}$ $\Rightarrow$ *moving* (possibly tilting) the hyperplane

How to Determine Δw_{ik}?

Remarks:

- For the decision hyperplane it holds that $\mathbf{w}^T\mathbf{x} = 0$, i.e. $\mathbf{w} \perp \mathbf{x}$ ($\mathbf{w}$ and $\mathbf{x}$ orthogonal)
- Suppose that for an input vector $\mathbf{x} \in C^+$ it holds that $\mathbf{w}^T\mathbf{x} < 0$, then $\mathbf{w}$ should be moved to the positive site of the decision hyperplane:

$$\mathbf{w}' = \mathbf{w} + c \cdot \mathbf{x}$$

  for small $c \in \mathbb{R}_0^+$

- Suppose that for an input vector $\mathbf{x} \in C^-$ it holds that $\mathbf{w}^T\mathbf{x} > 0$, then $\mathbf{w}$ should be moved to the negative site of the decision hyperplane:

$$\mathbf{w}' = \mathbf{w} - c \cdot \mathbf{x}$$

  for small $c \in \mathbb{R}_0^+$

Example

Positive and negative examples:

$$C^+ = \{(1,1), (1,-1), (0,-1)\}$$
$$C^- = \{(-1,-1), (-1,1), (0,1)\}$$

Fill up with 1's, yielding instances of $\mathbf{x}$, i.e. the training set T:

$$T = \{(1,1,1), (1,1,-1), (1,0,-1), (1,-1,-1), (1,-1,1), (1,0,1)\}$$

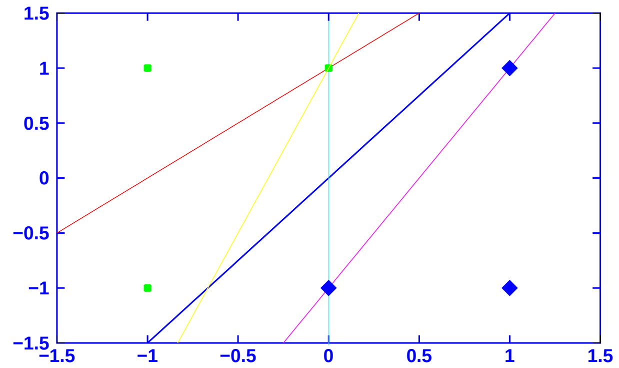
Fixed increment rule:

$$\mathbf{w}^{(r+1)} = \begin{cases} \mathbf{w}^{(r)} + c\mathbf{x}^{(r)} & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} \le 0 \text{ and } \mathbf{x}^{(r)} \in C^+ \\ \mathbf{w}^{(r)} - c\mathbf{x}^{(r)} & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} \ge 0 \text{ and } \mathbf{x}^{(r)} \in C^- \\ \mathbf{w}^{(r)} & \text{otherwise} \end{cases}$$

Example $c = 1$ (continued)

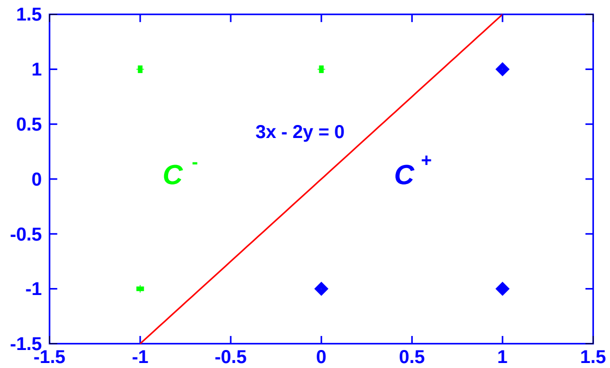
Pattern	Weight	$\mathbf{w}^T\mathbf{x}$	Update	New
		Iteration 1		
$[1,1,1]$	$[0,1,0]$	1	No	$[0,1,0]$
$[1,1,-1]$	$[0,1,0]$	1	No	$[0,1,0]$
$[1,0,-1]$	$[0,1,0]$	0	Yes	$[1,1,-1]$
$[1,-1,-1]$	$[1,1,-1]$	1	Yes	$[0,2,0]$
$[1,-1,1]$	$[0,2,0]$	-2	No	$[0,2,0]$
$[1,0,1]$	$[0,2,0]$	0	Yes	$[-1,2,-1]$
		Iteration 2		
$[1,1,1]$	$[-1,2,-1]$	0	Yes	$[0,3,0]$
$[1,1,-1]$	$[0,3,0]$	3	No	$[0,3,0]$
$[1,0,-1]$	$[0,3,0]$	0	Yes	$[1,3,-1]$
$[1,-1,-1]$	$[1,3,-1]$	-1	No	$[1,3,-1]$
$[1,-1,1]$	$[1,3,-1]$	-3	No	$[1,3,-1]$
$[1,0,1]$	$[1,3,-1]$	0	Yes	$[0,3,-2]$
		Iteration 3		
$[1,1,1]$	$[0,3,-2]$	1	No	$[0,3,-2]$
$[1,1,-1]$	$[0,3,-2]$	5	No	$[0,3,-2]$
$[1,0,-1]$	$[0,3,-2]$	2	No	$[0,3,-2]$
$[1,-1,-1]$	$[0,3,-2]$	-1	No	$[0,3,-2]$
$[1,-1,1]$	$[0,3,-2]$	-5	No	$[0,3,-2]$
$[1,0,1]$	$[0,3,-1]$	-2	No	$[0,3,-2]$

Various Lines



- $0 \cdot 1 + 1x + 0y = x = 0$
- $1 + 1x - 1y = 0$
- $-1 + 2x - 1y = 0$
- $1 + 3x - 1y = 0$
- $0 + 3x - 2y = 0$

Resulting Decision Line



Positive and negative examples:

$$C^+ = \{(1,1),(1,-1),(0,-1)\}$$
$$C^- = \{(-1,-1),(-1,1),(0,1)\}$$

Resulting weight: $\mathbf{w}^T = [0, 3, -2]$, i.e. decision line $3x - 2y = 0$

Analysis of Fixed Increment Rule

$$\mathbf{w}^{(r+1)} = \begin{cases} \mathbf{w}^{(r)} + c\mathbf{x}^{(r)} & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} \le 0 \text{ and } \mathbf{x}^{(r)} \in C^+ \\ \mathbf{w}^{(r)} - c\mathbf{x}^{(r)} & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} \ge 0 \text{ and } \mathbf{x}^{(r)} \in C^- \\ \mathbf{w}^{(r)} & \text{otherwise} \end{cases}$$

Simplification: if $\mathbf{x} \in C^-$, then replace $\mathbf{x}$ by $-\mathbf{x}$, and merge C^+ and C^-

Example: If

$$T = \{(1,1,1),(1,1,-1),(1,0,-1),(1,-1,-1),(1,-1,1),(1,0,1)\}$$

then

$$T' = \{(1,1,1),(1,1,-1),(1,0,-1),(-1,1,1),(-1,1,-1),(-1,0,-1)\}$$

The fixed increment rule then becomes:

$$\mathbf{w}^{(r+1)} = \begin{cases} \mathbf{w}^{(r)} + c\mathbf{x}^{(r)} & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} \le 0 \\ \mathbf{w}^{(r)} & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} > 0 \end{cases}$$

Analysis of Fixed Increment Rule

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} + \Delta\mathbf{w}$$

where:

(1) the change in $\mathbf{w}$ should change the mean square error $E(\mathbf{w})$ as fast as possible

(2) $c \in \mathbb{R}_0^+$

It is known that (1) is true when $\Delta\mathbf{w} = -c\nabla E$, where

$$\nabla E = \begin{bmatrix} \partial E/\partial w_1 \\ \vdots \\ \partial E/\partial w_n \end{bmatrix}$$

Let $e_r(\mathbf{w}^{(r)}, \mathbf{x}^{(r)}) = \frac{1}{2}(|\mathbf{w}^{(r)T}\mathbf{x}^{(r)}| - \mathbf{w}^{(r)T}\mathbf{x}^{(r)})$, for iteration r

Note that $e_r(\mathbf{w}^{(r)}, \mathbf{x}^{(r)})$ has a minimum when $\mathbf{w}^{(r)T}\mathbf{x}^{(r)} \ge 0$.

Fixed Increment Rule (continued)

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - c\nabla E$$

then:

$$\frac{\partial e_r(\mathbf{w}^{(r)T}\mathbf{x}^{(r)})}{\partial \mathbf{w}^{(r)}} = \frac{1}{2}(\mathbf{x}^{(r)}\,Q(\mathbf{w}^{(r)T}\mathbf{x}^{(r)}) - \mathbf{x}^{(r)})$$

where

$$Q(\mathbf{w}^{(r)T}\mathbf{x}^{(r)}) = \begin{cases} 1 & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} > 0 \\ -1 & \text{if } \mathbf{w}^{(r)T}\mathbf{x}^{(r)} \leq 0 \end{cases}$$

Resulting rule:

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} + \frac{c}{2}\cdot(\mathbf{x}^{(r)} - \mathbf{x}^{(r)}Q(\mathbf{w}^{(r)T}\mathbf{x}^{(r)}))$$

Multilayer Feedforward Neural Networks

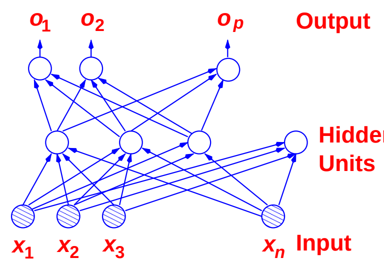
Limitation of one-layer feedforward neural network:

Modelling of **logical XOR** with $\mathbf{w}^T = [1\,1]$ and $\mathbf{x}^T = [x_1\, x_2]$:

x_1	x_2	$x_1 \otimes x_2$	$\mathbf{w}^T\,\mathbf{x}$
1	1	0	$[1\ 1]\begin{bmatrix}1\\1\end{bmatrix} = 2$
1	0	1	$[1\ 1]\begin{bmatrix}1\\0\end{bmatrix} = 1$
0	1	1	$[1\ 1]\begin{bmatrix}0\\1\end{bmatrix} = 1$
0	0	0	$[1\ 1]\begin{bmatrix}0\\0\end{bmatrix} = 0$

Conclusion: *not linearly separable* (i.e. results cannot be separated by a decision line); solution: *multilayer* network

Multilayer Feedforward Neural Networks

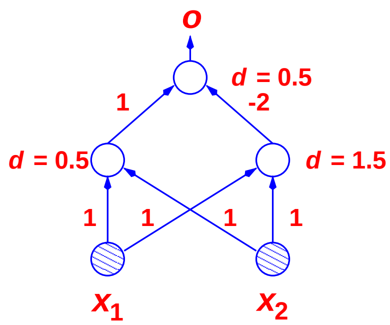


- L layers, $l = 0, \ldots, L$, with $l = 0$: input layer; $l = L$: output layer
- Output o_i, $i = 1, \ldots, n_l$, of *every* layer:

$$o_i = f_i(a_i) = f_i\left(\sum_{k=0}^{n_{l-1}} w_{ik} o_k\right)$$

  with n_l number of units in layer l
- *Universality property:* ML networks universal, *nonlinear* discriminant functions

Solution of XOR problem



d are thresholds of the activation function

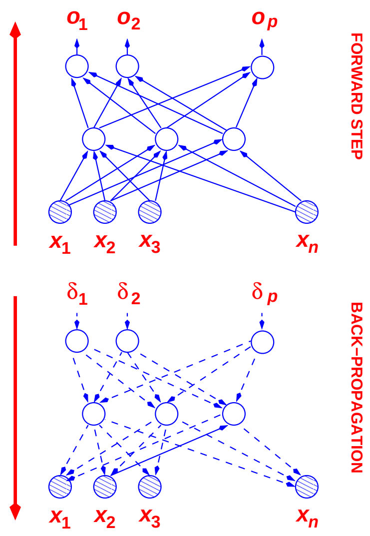
Example:

- Let $x_1 = x_2 = 1$, then hidden layer $a_1 = a_2 = 1 \times 1 + 1 \times 1 = 2$
- $o_1 = 1$, as $2 > 0.5$; $o_2 = 1$, as $2 > 1.5$
- Output layer: $a = 1 \times 1 + 1 \times -2 = -1$
- $o = 0$ as $-1 < 0.5$

Back-propagation Learning

- A multilayer neural network cannot be trained by only comparing outputs $\mathbf{o}^q$ to targets $\mathbf{d}^q$
- **Solution:** *back-propagation*:
  - propagate input from the input layer to (final) output layer
  - error vector (= difference target vector and output vector) fed into the network
  - iterate until converged to a solution within satisfactory bounds

Back-propagation



$\delta_i^{l-1} = f'(a_i)\sum_{j=1}^{n_l} w_{ji}\delta_j^l$, for layer $l-1$, and δ_j^L the difference between output o_j^L and target d_j