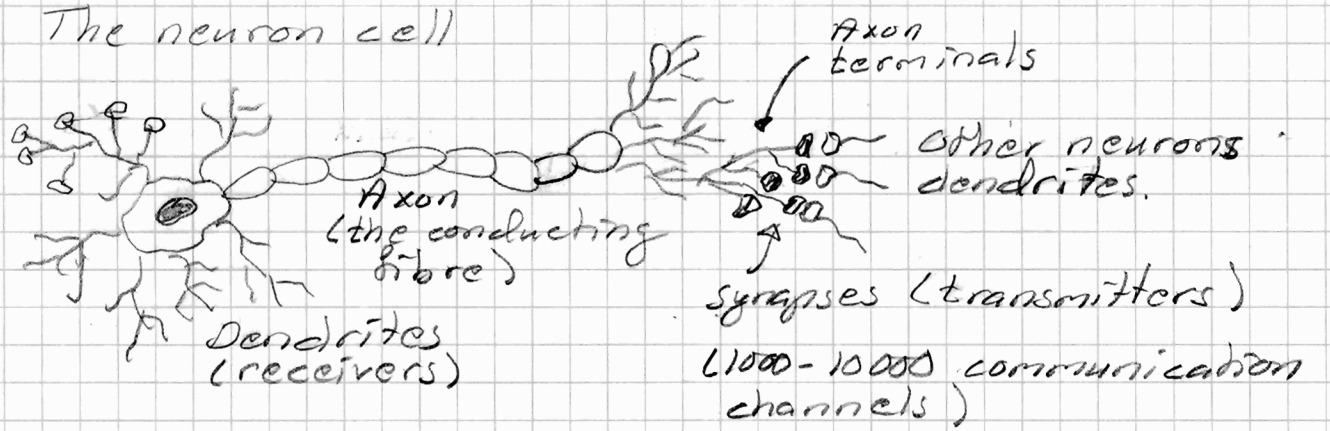
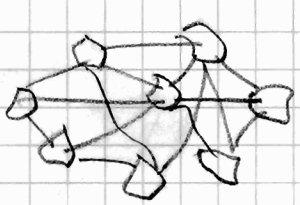
The neuron cell



Approx: 100 billion neuron cells (10^{11})

Brain composed of many interconnected parts

A highly complex non-linear and parallel computer



Structural constituents: "Neurons" (beståndsdelar)

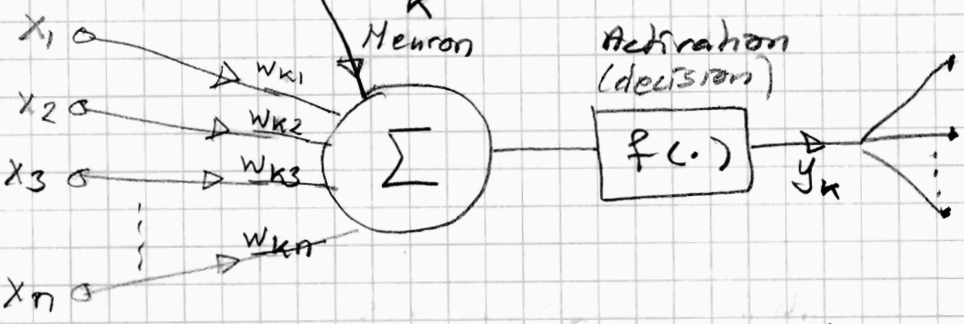
Artificial neural network (6 properties)

1) Nonlinearity:
   Interconnection of nonlinear neurons
2) Input - output mapping:
   Learning with a "teacher"
3) Adaptivity:
   Can adapt the free parameters (w_i) (based on evidence)
4) Evidential response:
   Decision with a measure of confidence
5) Fault tolerance:
   Graceful degradation
6) (VLSI) implementability:
   Very-large-scale-integration

Treshold

b_k (bias) modeled as: $x_0 = +1$ $w_{k0} = b_k$

Model



$w > 0$ excitatory synapse

$w < 0$ inhibitory synapse

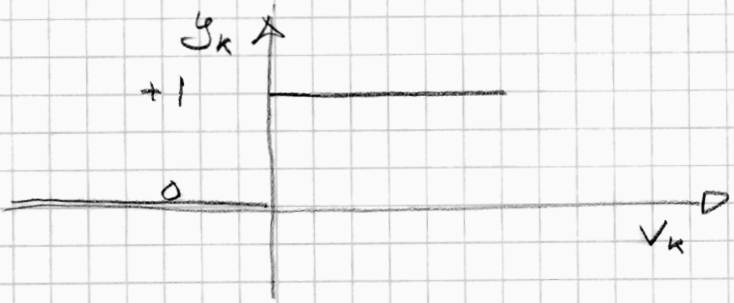
$$v_k = w_{k1}x_1 + w_{k2}x_2 + \cdots + w_{kn}x_n + \underbrace{w_{k0}}_{b_k}\underbrace{x_0}_{+1}$$

w_{kj} = connection wheight from neuron j to neuron k

Threshold function (activation function)

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

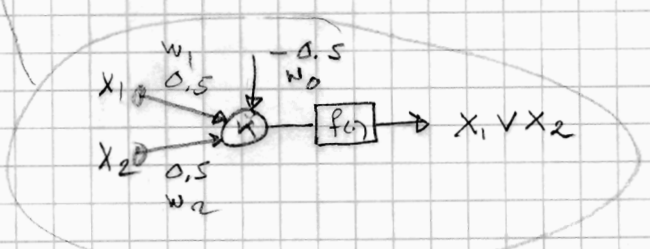
McCulloch-Pitts



Example

Choose $w_0 = -0.5$, $w_1 = 0.5$, $w_2 = 0.5$

x_1	x_2	$x_1 \vee x_2$		
1	1	1	$v_k = -0.5 \cdot 1 + 0.5 \cdot 1 + 0.5 \cdot 1 = 0.5$	1
1	0	1	$v_k = -0.5 \cdot 1 + 0.5 \cdot 1 + 0.5 \cdot 0 = 0$	1
0	1	1	$v_k = -0.5 \cdot 1 + 0.5 \cdot 0 + 0.5 \cdot 1 = 0$	1
0	0	0	$v_k = -0.5 \cdot 1 + 0.5 \cdot 0 + 0.5 \cdot 0 = -0.5$	0



Hebbian learning

If neuron j repeatedly or persistantly fires at neuron k (ie j fires 1 when output $k = 1$) then the strength of w_{kj} increases and vice versa.

How do the neuron learn
Define D = desired output A = actual output $A, D \in \{0, 1\}$

Definition of learning rule:

If $D > A$ $\quad w_{kj} := w_{kj} + x_j$

If $D < A$ $\quad w_{kj} := w_{kj} - x_j$

If $D = A$ do nothing

Example (cont)

x_1	x_2	D
1	1	1
1	0	1
0	1	1
0	0	0

Start with $w_0 = w_1 = w_2 = 0$

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = (w_0, w_1, w_2) \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

we change weights here

$$(0,0,0) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = (0,0,0,0) \xrightarrow{f} \begin{matrix} (1,1,1,1) \\ (1,1,1,0) \end{matrix} \begin{matrix} A \\ D \end{matrix} \quad A > D$$

$-x_j$

we change weights here

$$(0-1, 0-0, 0-0) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = (-1,-1,-1,-1) \xrightarrow{f} \begin{matrix} (0,0,0,0) \\ (1,1,1,0) \end{matrix} \begin{matrix} A \\ D \end{matrix} \quad A < D$$

x_j

we change weights here

$$(0+1, 0+1, 0+1) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = (0,1,1) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = (2,1,1,0) \xrightarrow{f} \begin{matrix} (1,1,1,1) \\ (1,1,1,0) \end{matrix} \begin{matrix} A \\ D \end{matrix} \quad A > D$$

$-x_j$

$$(1-1, 1-0, 1-0) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = (-1,1,1) \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} = (1,0,0,-1) \xrightarrow{f} \begin{matrix} (1,1,1,0) \\ (1,1,1,0) \end{matrix} \begin{matrix} A \\ D \end{matrix} \quad A = D$$

Hence this example shows that it is possible to find weights that make a neuron equivalent to $x_1 \vee x_2$ (obs 0.5 and 1 the same)

Excercise

Show that there is a neuron that match $x_1 \wedge x_2$

The logical table XOR can not be matched by any singel neuro.

X_1	X_2	X_1 XOR X_2
1	1	0
1	0	1
0	1	1
0	0	0

Proof One finds the system

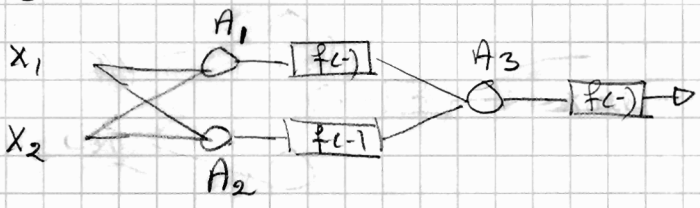
1) $w_0 + w_1 + w_2 < 0$
2) $w_0 + w_1 \geq 0$
3) $w_0 + w_2 \geq 0$
4) $w_0 < 0$

1+4) $2w_0 + w_1 + w_2 < 0$

2+3) $2w_0 + w_1 + w_2 \geq 0$

Motsägelse !

But



X_1	X_2	A_1	A_2	A_3
1	1	0	0	0
1	0	1	0	1
0	1	0	1	1
0	0	0	0	0

Single neurons

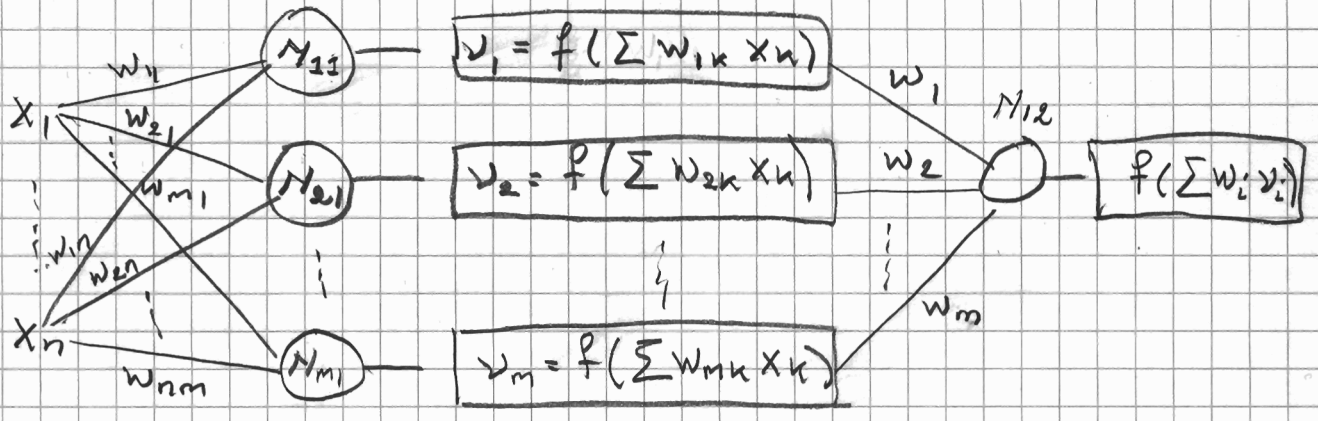
$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} \quad \begin{pmatrix} 0.5 \\ 1 \\ -1 \end{pmatrix} \quad \begin{pmatrix} 0.5 \\ -1 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0.5 \\ 1 \\ 1 \end{pmatrix}$$

$A_3 = A_1 \vee A_2$ Network w. th neurons

A_1 and A_2 play the role of x_1 and x_2 but for A_3.

The (A_1, A_2)-layer is called: A hidden layer

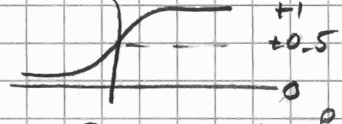
A general 1-hidden layer network



The function f

So far our activation function f has been a simple step function. It is possible to maintain the treshold property with a continuous function

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



is common, its derivative is easy to find $f'(x) = -f(x)(1-f(x))$
This is important when updating the weights w_{ik} (Backpropagation)

Kolmogorov theorem (theoretical interest)

Any continuous function $f: R^n \to [0,1]^n$ may be written

$$f(x_1, \ldots, x_n) = \sum_{k=1}^{2n+1} g\left(\sum_{i=1}^{n} \lambda_i h_k(x_i)\right)$$

g and h_k are functions of one variable.
g do not depend on f

With aid of kolmogorovs theorem it has been shown that

1) Every boolean function can be represented exactly by a network with two layers (XOR)

2) Every bounded continuous function can be approximated by a network with two layers

3) Any function can be approximated by a network with three layers.

In case 2) and 3) the activation function ~~for~~ the hidden layers must be the sigmoid and for the output linear.