Computational Intelligence

Winter Term 2020/21

Prof. Dr. Günter Rudolph

Lehrstuhl für Algorithm Engineering (LS 11)

Fakultät für Informatik

TU Dortmund

- Introduction to Artificial Neural Networks
  - McCulloch Pitts Neuron (MCP)
  - Minsky / Papert Perceptron (MPP)

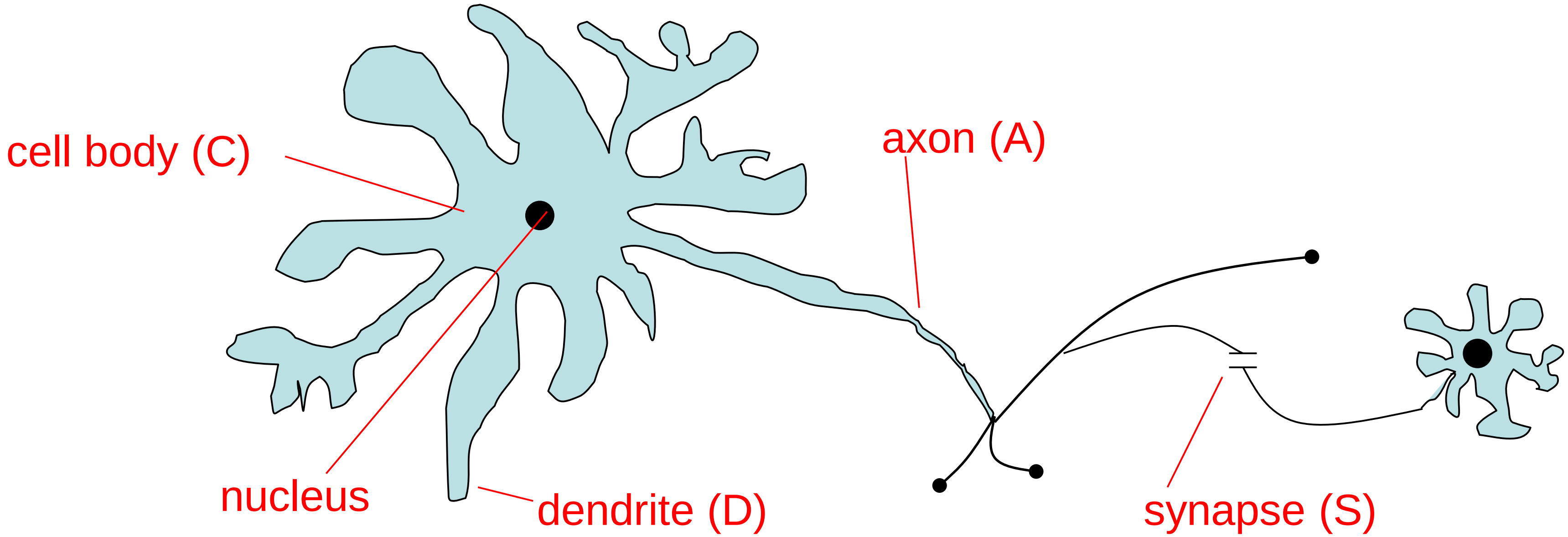
Biological Prototype

- Neuron
  - Information gathering (D)
  - Information processing (C)
  - Information propagation (A / S)

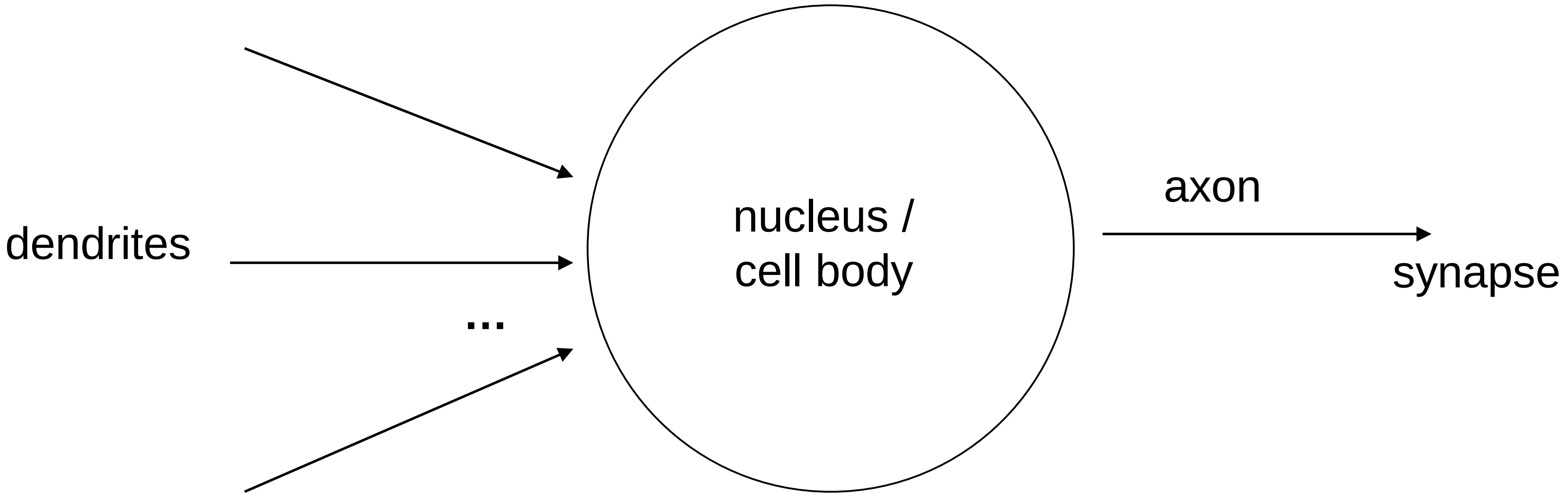
human being: 10^{12} neurons

electricity in mV range

speed: 120 m / s



Abstraction

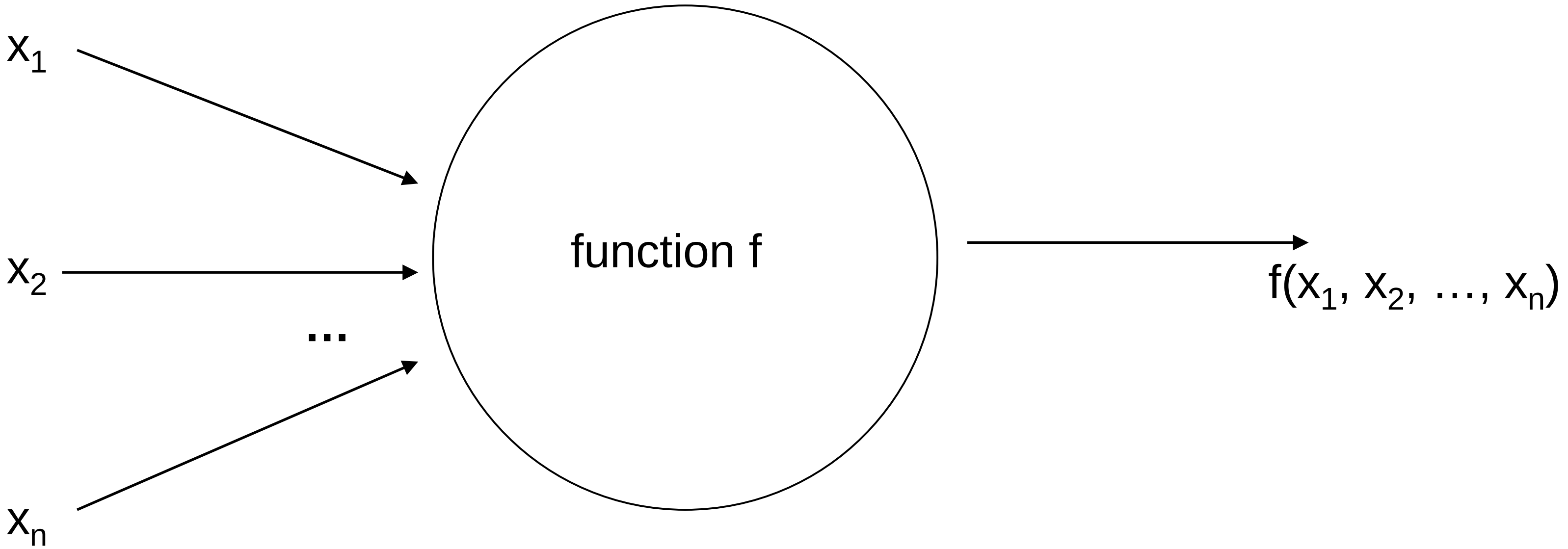


signal input

signal processing

signal output

Model



McCulloch-Pitts-Neuron 1943:

$x_i \in \{ 0, 1 \} =: B$

$f: B^n \rightarrow B$

1943: Warren McCulloch / Walter Pitts

- description of neurological networks
  → modell: McCulloch-Pitts-Neuron (MCP)

- basic idea:

  - neuron is either active or inactive
  - skills result from ***connecting*** neurons

- considered static networks
  (i.e. connections had been constructed and not learnt)

McCulloch-Pitts-Neuron

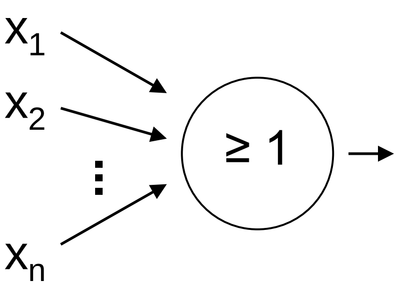
n binary input signals $x_1, \ldots, x_n$

threshold $\theta > 0$

$$f(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geq \theta \\ 0 & \text{else} \end{cases}$$

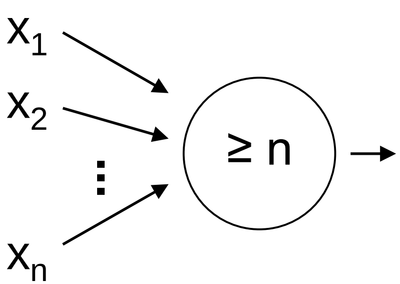
$\Rightarrow$ can be realized:

boolean OR



$\theta = 1$

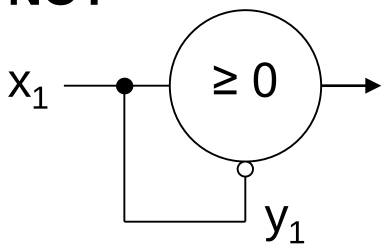
boolean AND



$\theta = n$

McCulloch-Pitts-Neuron

NOT



n binary input signals $x_1, \ldots, x_n$

threshold $\theta > 0$

in addition: m binary inhibitory signals $y_1, \ldots, y_m$

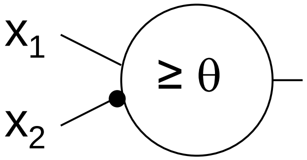
$$\tilde{f}(x_1, \ldots, x_n; y_1, \ldots, y_m) = f(x_1, \ldots, x_n) \cdot \prod_{j=1}^{m} (1 - y_j)$$

- if at least one $y_j = 1$, then output = 0
- otherwise:

  - sum of inputs ≥ threshold, then output = 1
    else output = 0

Assumption:

inputs also available in inverted form, i.e. ∃ inverted inputs.



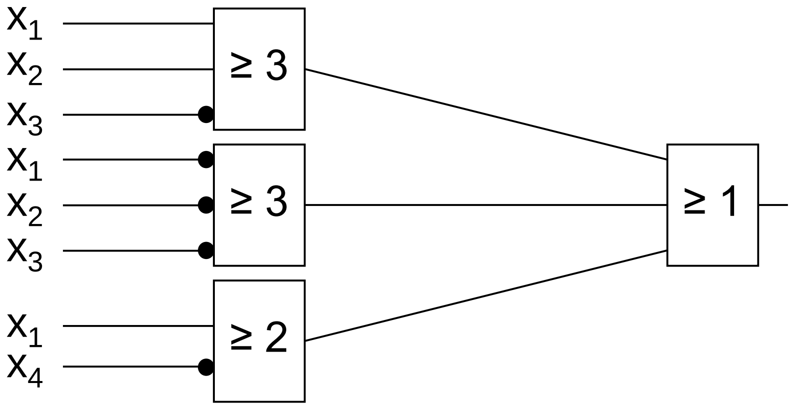
$\Rightarrow x_1 + \bar{x}_2 \geq \theta$

Theorem:

Every logical function $F: B^n \to B$ can be simulated with a two-layered McCulloch/Pitts net.

Example:

$$F(x) = x_1 x_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_4$$



Proof: (by construction)

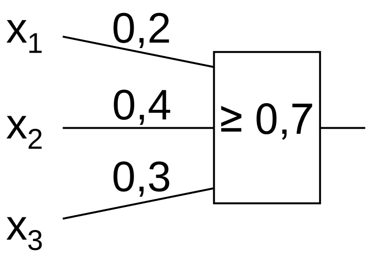
Every boolean function F can be transformed in disjunctive normal form

$\Rightarrow$ 2 layers (AND - OR)

1. Every clause gets a decoding neuron with $\theta = n$
   $\Rightarrow$ output = 1 only if clause satisfied (AND gate)
2. All outputs of decoding neurons
   are inputs of a neuron with $\theta = 1$ (OR gate)

q.e.d.

Generalization: inputs with weights



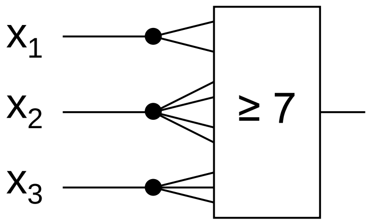
fires 1 if

$$0{,}2\,x_1 + 0{,}4\,x_2 + 0{,}3\,x_3 \geq 0{,}7 \quad \Big|\cdot 10$$

$$2\,x_1 + 4\,x_2 + 3\,x_3 \geq 7$$

$\Downarrow$

duplicate inputs!



$\Rightarrow$ equivalent!

Theorem:

Weighted and unweighted MCP-nets are equivalent for weights $\in \mathbb{Q}^+$.

Proof:

„⇒“ Let $\sum_{i=1}^{n} \frac{a_i}{b_i} x_i \geq \frac{a_0}{b_0}$ with $a_i, b_i \in \mathbb{N}$

Multiplication with $\prod_{i=0}^{n} b_i$ yields inequality with coefficients in N

Duplicate input x_i, such that we get $a_i\, b_1\, b_2 \cdots b_{i-1}\, b_{i+1} \cdots b_n$ inputs.

Threshold $\theta = a_0\, b_1 \cdots b_n$

„⇐“

Set all weights to 1. **q.e.d.**

Conclusion for MCP nets:

+ feed-forward: able to compute any Boolean function

+ recursive: able to simulate DFA (**d**eterministic **f**inite **a**utomaton)

– very similar to conventional logical circuits

– difficult to construct

– no good learning algorithm available

Perceptron (Rosenblatt 1958)

→ complex model → reduced by Minsky & Papert to what is „necessary“

→ Minsky-Papert perceptron (MPP), 1969 → essential difference: $x \in [0,1] \subset \mathbb{R}$

What can a <u>single</u> MPP do?

$$w_1 x_1 + w_2 x_2 \geq \theta$$

Y → 1

N → 0

isolation of x_2 yields:

$$x_2 \geq \frac{\theta}{w_2} - \frac{w_1}{w_2} x_1$$

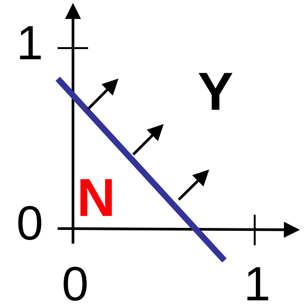
Y → 1

N → 0

Example:

$$0,9\, x_1 + 0,8\, x_2 \geq 0,6$$

$$\Leftrightarrow \quad x_2 \geq \frac{3}{4} - \frac{9}{8} x_1$$

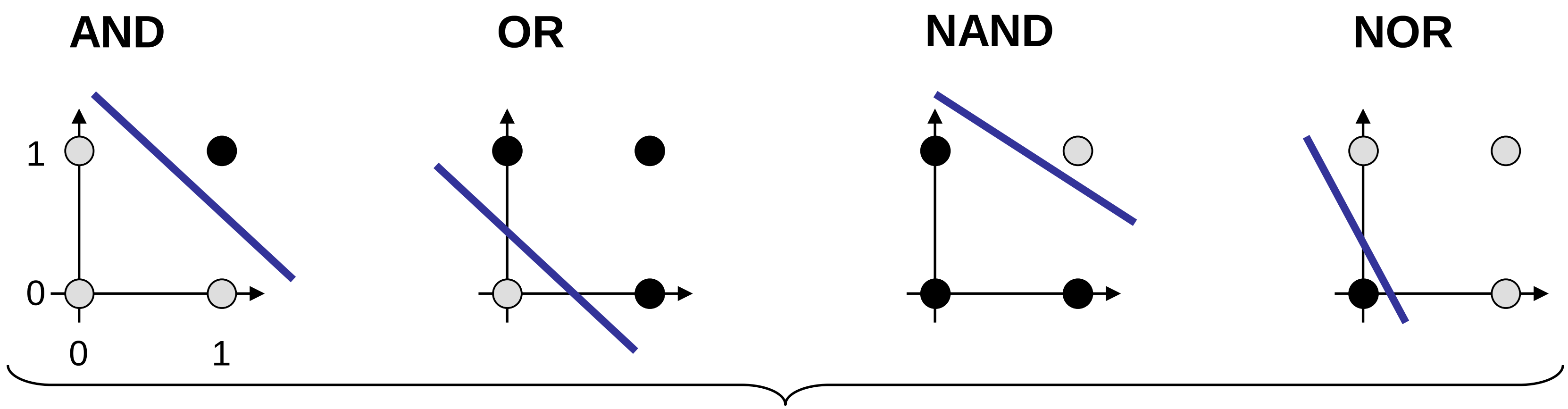


separating line

<u>separates</u> $\mathbb{R}^2$

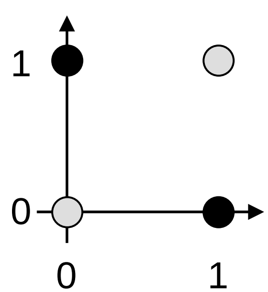
in 2 classes

○ = 0 ● = 1



→ MPP at least as powerful as MCP neuron!

XOR



?

x_1	x_2	xor
0	0	0
0	1	1
1	0	1
1	1	0

$w_1 x_1 + w_2 x_2 \geq \theta$

$\Rightarrow 0 < \theta$

$\Rightarrow w_2 \geq \theta$

$\Rightarrow w_1 \geq \theta$

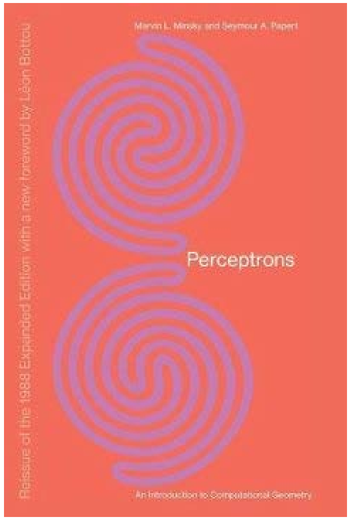
$\Rightarrow w_1 + w_2 < \theta$

$w_1, w_2 \geq \theta > 0$

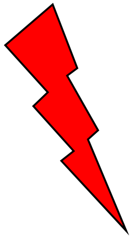
$\Rightarrow w_1 + w_2 \geq 2\theta$

contradiction!

1969: Marvin Minsky / Seymor Papert



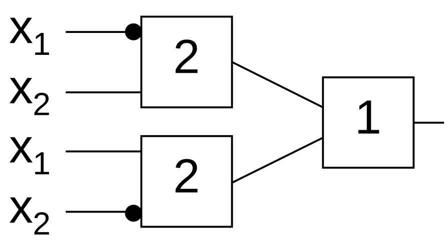
- book *Perceptrons* → analysis math. properties of perceptrons
- disillusioning result:
  perceptions fail to solve a number of trivial problems!
  - XOR Problem
  - Parity Problem
  - Connectivity Problem
- "conclusion": all artificial neurons have this kind of weakness!
  ⇒ research in this field is a scientific dead end!



- consequence: research funding for ANN cut down extremely (~ 15 years)

how to leave the „dead end“:

1. <u>Multilayer</u> Perceptrons:

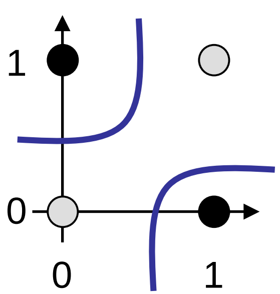


$\Rightarrow$ realizes XOR

2. <u>Nonlinear</u> separating functions:

XOR

$g(x_1, x_2) = 2x_1 + 2x_2 - 4x_1x_2 - 1$ with $\theta = 0$



$g(0,0) = -1$
$g(0,1) = +1$
$g(1,0) = +1$
$g(1,1) = -1$

How to obtain weights w_i and threshold θ ?

<u>as yet:</u> by construction

example: NAND-gate

x_1	x_2	NAND
0	0	1
0	1	1
1	0	1
1	1	0

$\Rightarrow 0 \geq \theta$

$\Rightarrow w_2 \geq \theta$

$\Rightarrow w_1 \geq \theta$

$\Rightarrow w_1 + w_2 < \theta$

requires solution of a system of linear inequalities ($\in$ P)

(e.g.: $w_1 = w_2 = -2$, $\theta = -3$)

<u>now:</u> by „learning“ / training

Perceptron Learning

Assumption: test examples with correct I/O behavior available

Principle:

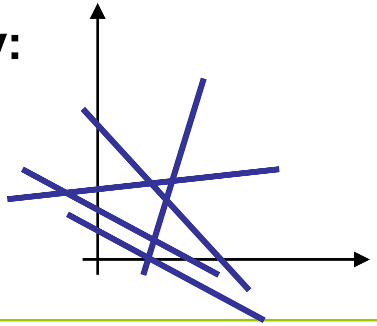
(1) choose initial weights in arbitrary manner

(2) feed in test pattern

(3) if output of perceptron wrong, then change weights

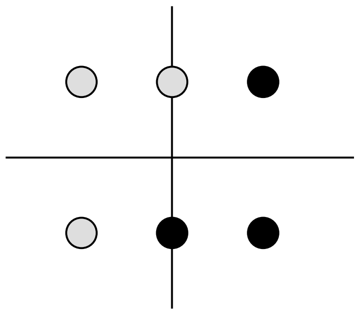
(4) goto (2) until correct output for all test paterns

graphically:



→ translation and rotation of separating lines

Example



$$P = \left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\} \quad \bullet$$

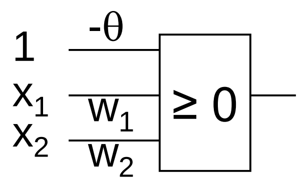
$$N = \left\{ \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} \quad \circ$$

threshold as a weight: $w = (\theta, w_1, w_2)'$

$$\Downarrow$$

$$P = \left\{ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} \right\}$$

$$N = \left\{ \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \right\}$$



$$w_1x_1+w_2x_2 \geq \theta \Leftrightarrow w_1x_1+w_2x_2 \underbrace{- \theta}_{w_0} \cdot \underbrace{1}_{x_0} \geq 0$$

$\Rightarrow$ separating hyperplane:

$H(w) = \{ x : h(x;w) = 0 \}$

where

$h(x;w) = w'x = w_0x_0+w_1x_1+ \ldots + w_nx_n$

$\Rightarrow$ origin $0 \in H(w)$ since $h(0;w) = 0$

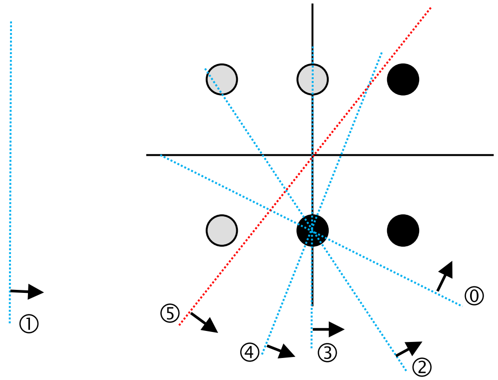
Perceptron Learning

P: set of positive examples → output 1
N: set of negative examples → output 0
threshold θ integrated in weights

1. choose w_0 at random, $t = 0$
2. choose arbitrary $x \in P \cup N$
3. if $x \in P$ and $w_t'x > 0$ then **goto** 2
   if $x \in N$ and $w_t'x \le 0$ then **goto** 2
   } I/O correct!
4. if $x \in P$ and $w_t'x \le 0$ then
   $w_{t+1} = w_t + x$; t++; **goto** 2
   } let $w'x \le 0$, should be > 0!
   $(w+x)'x = w'x + x'x > w' x$
5. if $x \in N$ and $w_t'x > 0$ then
   $w_{t+1} = w_t - x$; t++; **goto** 2
   } let $w'x > 0$, should be ≤ 0!
   $(w-x)'x = w'x - x'x < w' x$
6. stop? If I/O correct for all examples!

remark: if separating $H(w^*)$ exists, then
algorithm converges, is finite (but in worst case: exponential runtime)

Example



suppose initial vector of weights is

$w^{(0)} = (1, ½, 1)'$

```
> w = SPL(m,c(1,0.5,1))
[1] 1.0 0.5 1.0
[1] 2.0 0.5 0.0
[1] 1.0 1.5 1.0
[1] 0.0 2.5 0.0
[1] -1.0  2.5 -1.0
[1]  0.0  2.5 -2.0
```

```
SPL <- function(m,w) {
  print(w)
  repeat {
    OK <- TRUE
    for (i in 1:nrow(m)) {
      x <- m[i,]
      s <- x[1]*w[1]+x[2]*w[2]+x[3]*w[3]
      if (s <= 0) {
        OK <- FALSE
        w <- w + x
        print(w) # show every change
      }
    }
    if (OK) break;
  }
  return(w)
}
```

```
m <- matrix(  # only positive examples
  c(c( 1,1,1),c( 1,1,-1),c( 1,0,-1),
    c(-1,1,1),c(-1,1,-1),c(-1,0,-1)),
  nrow=6,byrow=TRUE)
```

Acceleration of Perceptron Learning

Assumption: $x \in \{ 0, 1 \}^n \Rightarrow ||x|| = \sum_{i=1}^{n} |x_i| \geq 1$ for all $x \neq (0, ..., 0)'$

Let $B = P \cup \{ -x : x \in N \}$ (only positive examples)

If classification incorrect, then $w'x < 0$.

Consequently, size of error is just $\delta = -w'x > 0$.

$\Rightarrow w_{t+1} = w_t + (\delta + \varepsilon)\, x$ for $\varepsilon > 0$ (small) corrects error in a ***single*** step, since

$$
\begin{aligned}
w'_{t+1}x &= (w_t + (\delta + \varepsilon)\, x)'\, x \\
&= \underbrace{w'_t\, x} + (\delta + \varepsilon)\, x'x \\
&= -\delta + \delta\, ||x||^2 + \varepsilon\, ||x||^2 \\
&= \underbrace{\delta\, (||x||^2 - 1)}_{\geq 0} + \underbrace{\varepsilon\, ||x||^2}_{> 0} > 0 \quad ☑
\end{aligned}
$$

Generalization:

<u>Assumption:</u> $x \in \mathbb{R}^n \quad \Rightarrow \|x\| > 0$ for all $x \neq (0, ..., 0)'$

as before: $w_{t+1} = w_t + (\delta + \varepsilon)\, x$ for $\varepsilon > 0$ (small) and $\delta = - w'_t\, x > 0$

$$\Rightarrow \quad w'_{t+1}x = \underbrace{\delta\,(\|x\|^2 - 1)}_{<\,0\ \text{possible!}} + \underbrace{\varepsilon\,\|x\|^2}_{>\,0}$$

<u>Claim:</u> Scaling of data does not alter classification task (if threshold 0)!

Let $\ell = \min\{\, \|x\| : x \in B \,\} > 0$

Set $\hat{x} = \dfrac{x}{\ell} \quad \Rightarrow$ set of scaled examples $\hat{B}$

$\Rightarrow \|\hat{x}\| \geq 1 \quad \Rightarrow \quad \|\hat{x}\|^2 - 1 \geq 0 \quad \Rightarrow \quad w'_{t+1}\,\hat{x} > 0$ ☑

Theorem:
Let X = P ∪ N with P ∩ N = ∅ be training patterns (P: positive; N: negative examples).
Suppose training patterns are embedded in $\mathbb{R}^{n+1}$ with threshold 0 and origin 0 ∉ X.

If separating hyperplane H(w) exists,
then scaling of data does not alter classification task!

Proof:
Suppose ∃ x ∈ P ∪ N with ||x|| < 1 and let ℓ = min{ ||x|| : x ∈ P ∪ N } > 0.

Set $\hat{x} = \frac{1}{\ell} x$ so that $\hat{P} = \{\frac{x}{\ell} : x \in P\}$ and $\hat{N} = \{\frac{x}{\ell} : x \in N\}$.

Suppose $\exists w$ with $\forall \hat{x} \in \hat{P} : w'\hat{x} > 0$ and $\forall \hat{x} \in \hat{N} : w'\hat{x} \leq 0$.

Then holds:

$$w'\hat{x} > 0 \Leftrightarrow w'\frac{x}{\ell} > 0 \Leftrightarrow w'x > 0$$

$$w'\hat{x} \leq 0 \Leftrightarrow w'\frac{x}{\ell} \leq 0 \Leftrightarrow w'x \leq 0$$

q.e.d.

There exist numerous variants of Perceptron Learning Methods.

Theorem: (Duda & Hart 1973)

If rule for correcting weights is $w_{t+1} = w_t + \gamma_t x$ (i.e., if $w'_t x < 0$) and

1. $\forall t \geq 0 : \gamma_t \geq 0$
2. $$\sum_{t=0}^{\infty} \gamma_t = \infty$$
3. $$\lim_{m \to \infty} \frac{\sum_{t=0}^{m} \gamma_t^2}{\left(\sum_{t=0}^{m} \gamma_t \right)^2} = 0$$

then $w_t \to w^*$ for $t \to \infty$ with $\forall x: x'w^* > 0$. ■

e.g.: $\gamma_t = \gamma > 0$ or $\gamma_t = \gamma / (t+1)$ for $\gamma > 0$

as yet: *Online Learning*

→ Update of weights after each training pattern (if necessary)

now: *Batch Learning*

→ Update of weights only after test of all training patterns

→ Update rule:

$$w_{t+1} = w_t + \gamma \sum_{\substack{w'_t x < 0 \\ x \in B}} x \qquad (\gamma > 0)$$

vague assessment in literature:

- advantage : „usually faster“
- disadvantage : „needs more memory“ ← just a single vector!

find weights by means of optimization

Let $F(w) = \{ x \in B : w'x < 0 \}$ be the set of patterns incorrectly classified by weight w.

Objective function: $f(w) = -\sum_{x \in F(w)} w'x \rightarrow \min!$

Optimum: $f(w) = 0$ iff $F(w)$ is empty

Possible approach: *gradient method*

$$w_{t+1} = w_t - \gamma \nabla f(w_t) \qquad (\gamma > 0)$$

converges to a local minimum (dep. on w_0)

Gradient method

$w_{t+1} = w_t - \gamma \nabla f(w_t)$

Gradient points in direction of steepest ascent of function f(·)

Gradient $\nabla f(w) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \ldots, \frac{\partial f(w)}{\partial w_n} \right)$

Caution:
Indices i of w_i here denote components of vector w; they are **not** the iteration counters!

$$\frac{\partial f(w)}{\partial w_i} = -\frac{\partial}{\partial w_i} \sum_{x \in F(w)} w'x = -\frac{\partial}{\partial w_i} \sum_{x \in F(w)} \sum_{j=1}^{n} w_j \cdot x_j$$

$$= - \sum_{x \in F(w)} \underbrace{\frac{\partial}{\partial w_i} \left(\sum_{j=1}^{n} w_j \cdot x_j \right)}_{x_i} = - \sum_{x \in F(w)} x_i$$

Gradient method

thus:

gradient $$\nabla f(w) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \ldots, \frac{\partial f(w)}{\partial w_n} \right)'$$

$$= \left(-\sum_{x \in F(w)} x_1, -\sum_{x \in F(w)} x_2, \ldots, -\sum_{x \in F(w)} x_n \right)'$$

$$= -\sum_{x \in F(w)} x$$

$$\Rightarrow \quad w_{t+1} = w_t + \gamma \sum_{x \in F(w_t)} x$$

gradient method ⇔ batch learning

How difficult is it

(a) to find a separating hyperplane, provided it exists?

(b) to decide, that there is no separating hyperplane?

Let $B = P \cup \{ -x : x \in N \}$ (only positive examples), $w_i \in R$, $\theta \in R$, $|B| = m$

For every example $x_i \in B$ should hold:

$x_{i1} w_1 + x_{i2} w_2 + \ldots + x_{in} w_n \geq \theta$ → trivial solution $w_i = \theta = 0$ to be excluded!

Therefore additionally: $\eta \in R$

$x_{i1} w_1 + x_{i2} w_2 + \ldots + x_{in} w_n - \theta - \eta \geq 0$

Idea: η maximize → if $\eta^* > 0$, then solution found

Matrix notation:

$$A = \begin{pmatrix} x'_1 & -1 & -1 \\ x'_2 & -1 & -1 \\ \vdots & \vdots & \vdots \\ x'_m & -1 & -1 \end{pmatrix} \qquad z = \begin{pmatrix} w \\ \theta \\ \eta \end{pmatrix}$$

Linear Programming Problem:

$f(z_1, z_2, \ldots, z_n, z_{n+1}, z_{n+2}) = z_{n+2} \quad \rightarrow \quad \max!$

s.t. $Az \geq 0$

} calculated by e.g. Kamarkar-algorithm in **polynomial time**

If $z_{n+2} = \eta > 0$, then weights and threshold are given by z.

Otherwise separating hyperplane does not exist!