

Computational Intelligence

Winter Term 2016/17

Prof. Dr. Günter Rudolph

Lehrstuhl für Algorithm Engineering (LS 11)

Fakultät für Informatik

TU Dortmund

- ▶ **Organization (Lectures / Tutorials)**
- ▶ **Overview CI**
- ▶ **Introduction to ANN**
 - ◆ **McCulloch Pitts Neuron (MCP)**
 - ◆ **Minsky / Papert Perceptron (MPP)**

Who are you?

either

studying “*Automation and Robotics*” (Master of Science)
Module “Optimization”

or

studying “*Informatik*”

- BSc-Modul “Einführung in die Computational Intelligence”
- Hauptdiplom-Wahlvorlesung (SPG 6 & 7)

or ... let me know!

Who am I ?

Günter Rudolph

Fakultät für Informatik, LS 11

Guenter.Rudolph@tu-dortmund.de
OH-14, R. 232

← best way to contact me

← if you want to see me

office hours:

Tuesday, 10:30–11:30am

and by appointment

Lectures Wednesday 10:15-11:45 OH12, R. E.003, weekly

Tutorials either Thursday 14:15-15:45 OH16, R. 2.05, bi-weekly

or Friday 14:15-15:45 OH14, R. 1.04, bi-weekly

Tutor Vanessa Volz, MSc, LS 11

Information

<http://ls11-www.cs.tu-dortmund.de/people/rudolph/teaching/lectures/CI/WS2016-17/lecture.jsp>

Slides see web page

Literature see web page

Exams

Effective since winter term 2016/17: written exam (not oral)

- Informatik, Diplom: Leistungsnachweis → Übungsschein
- Informatik, Diplom: Fachprüfung → written exam (90 min)
- Informatik, Bachelor: Module → written exam (90 min)
- Automation & Robotics, Master: Module → written exam (90 min)

mandatory for registration to written exam: **must pass tutorial**

Knowledge about

- mathematics,
- programming,
- logic

is helpful.

But what if something is unknown to me?

- covered in the lecture
- pointers to literature

... and don't hesitate to ask!

What is CI ?

⇒ umbrella term for computational methods inspired by nature

- artificial neural networks
- evolutionary algorithms
- fuzzy systems
- swarm intelligence
- artificial immune systems
- growth processes in trees
- ...

backbone

new developments

- term „computational intelligence“ made popular by John Bezdek (FL, USA)
- originally intended as a demarcation line
 - ⇒ establish border between artificial and computational intelligence
- nowadays: blurring border

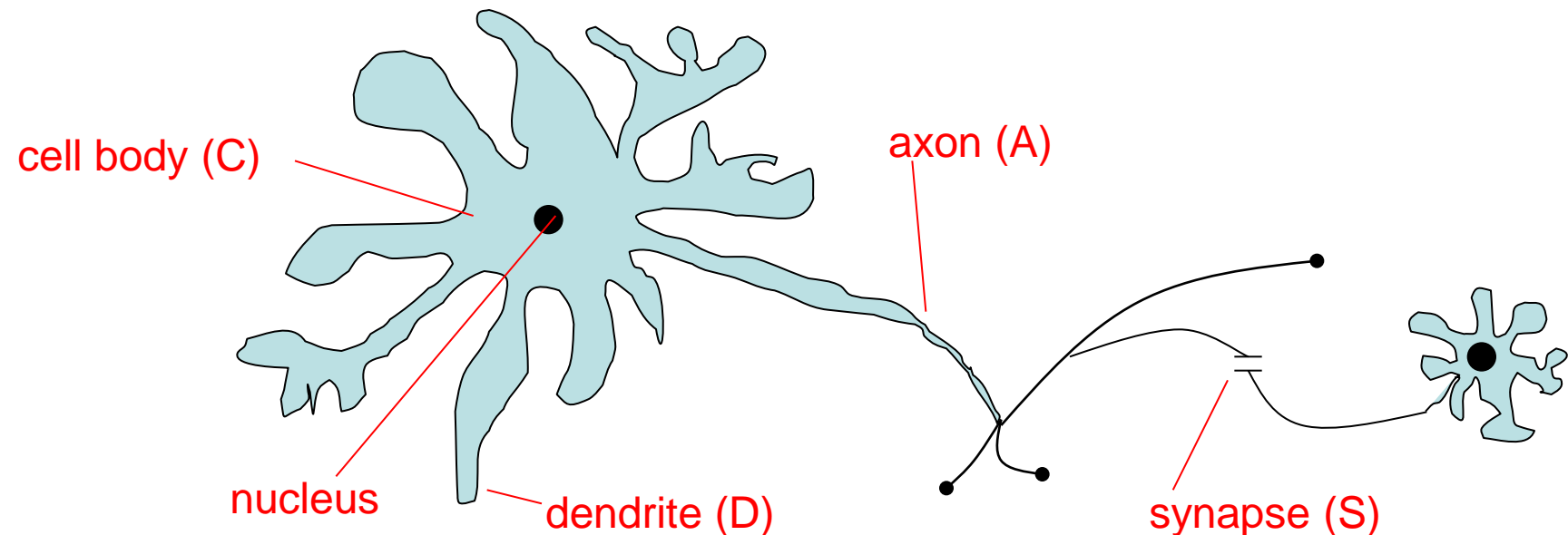
our goals:

1. know what CI methods are good for!
2. know when refrain from CI methods!
3. know why they work at all!
4. know how to apply and adjust CI methods to your problem!

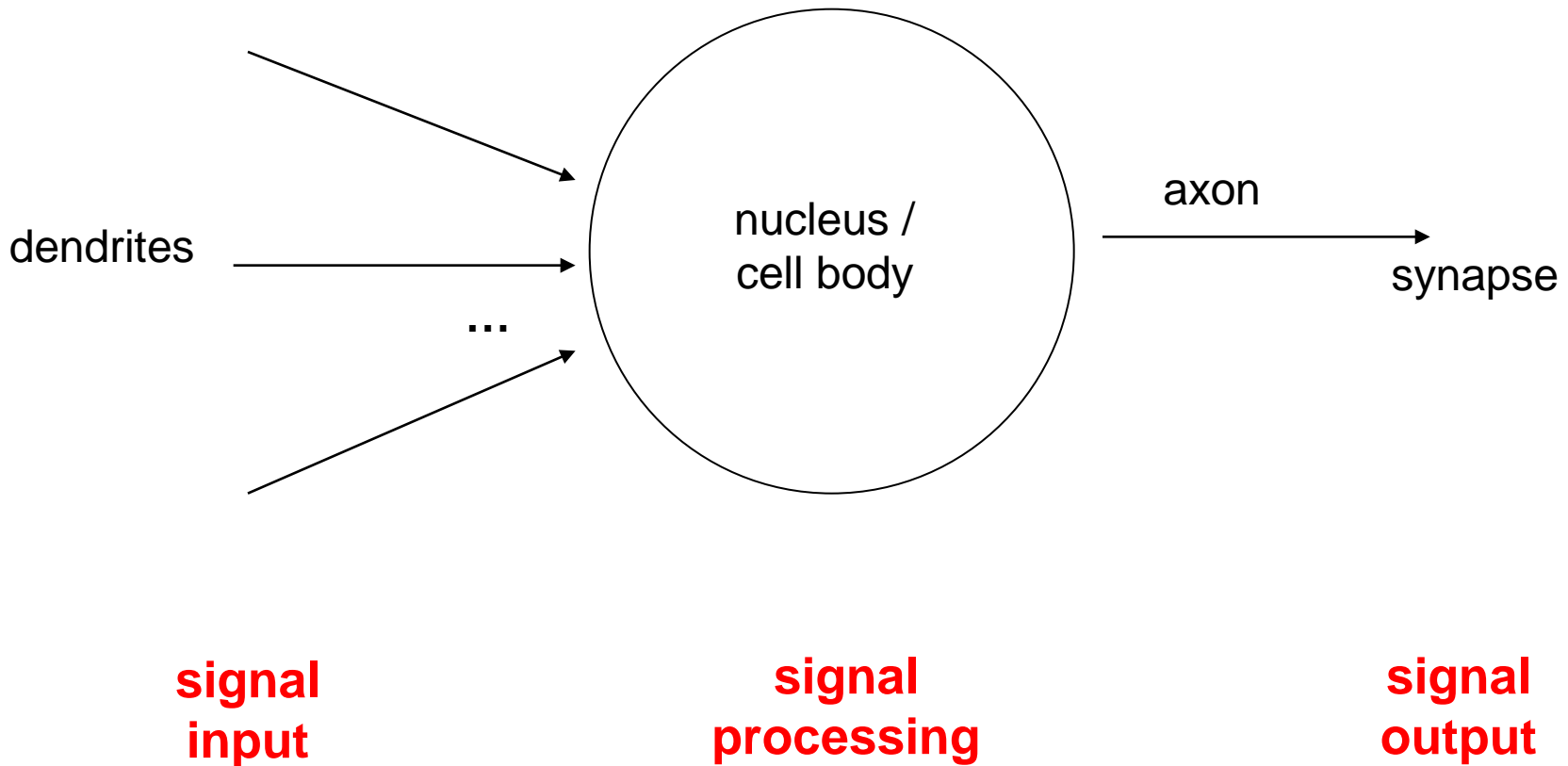
Biological Prototype

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

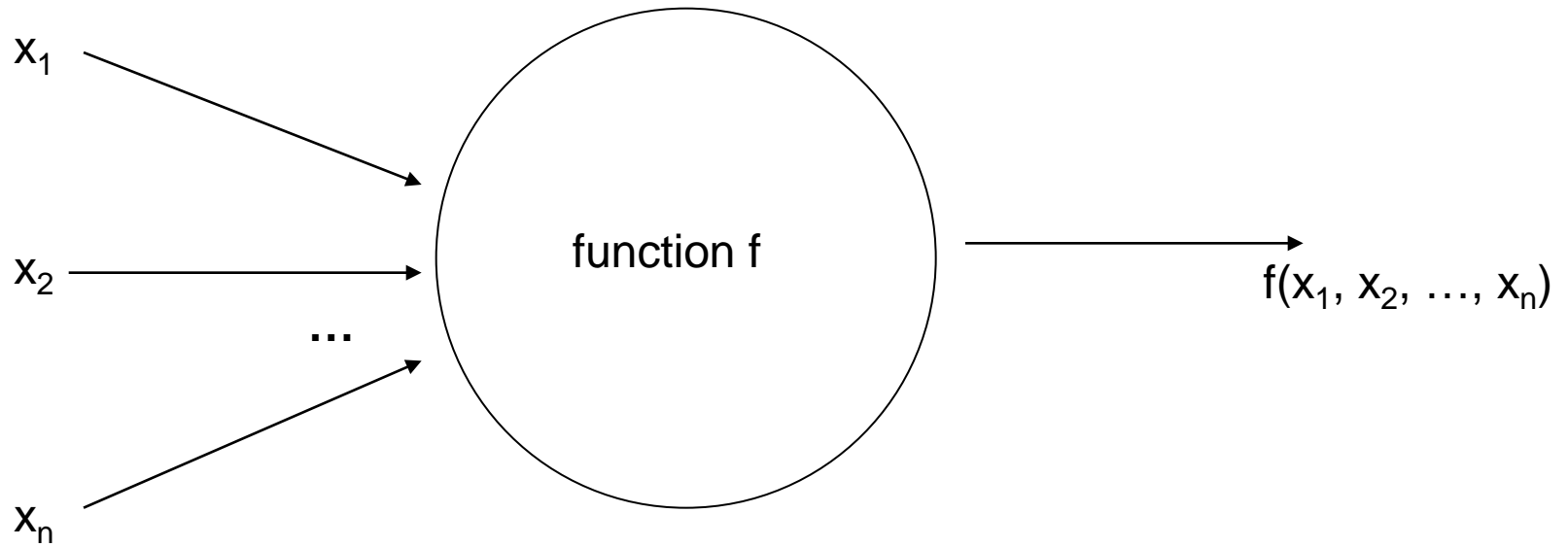
human being: 10^{12} neurons
electricity in mV range
speed: 120 m / s



Abstraction



Model



McCulloch-Pitts-Neuron 1943:

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

$$f: \mathbb{B}^n \rightarrow \mathbb{B}$$

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)

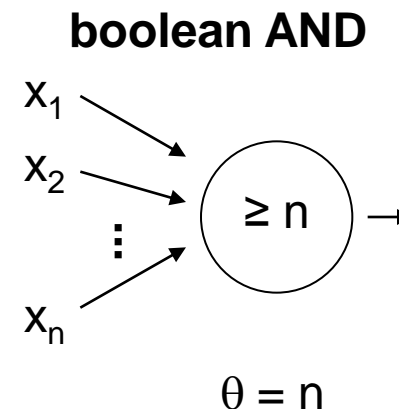
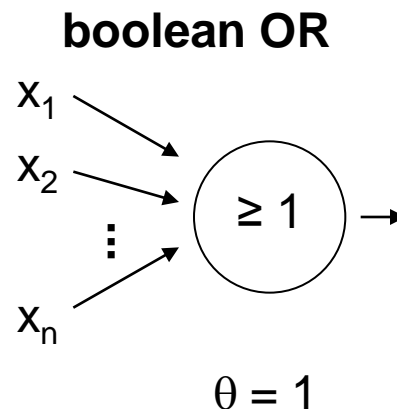
McCulloch-Pitts-Neuron

n binary input signals x_1, \dots, x_n

threshold $\theta > 0$

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

⇒ can be realized:



McCulloch-Pitts-Neuron

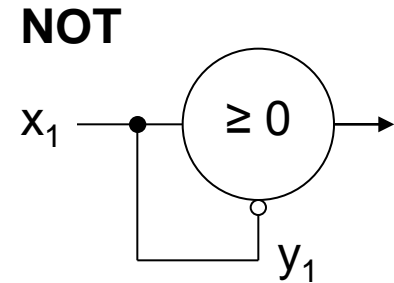
n binary input signals x_1, \dots, x_n

threshold $\theta > 0$

in addition: m binary inhibitory signals y_1, \dots, y_m

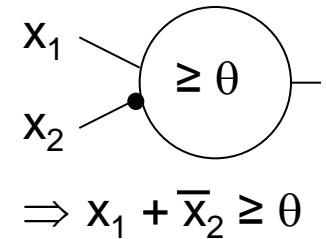
$$\tilde{f}(x_1, \dots, x_n; y_1, \dots, y_m) = f(x_1, \dots, x_n) \cdot \prod_{j=1}^m (1 - y_j)$$

- if at least one $y_j = 1$, then output = 0
- otherwise:
 - sum of inputs \geq threshold, then output = 1
 - else output = 0



Assumption:

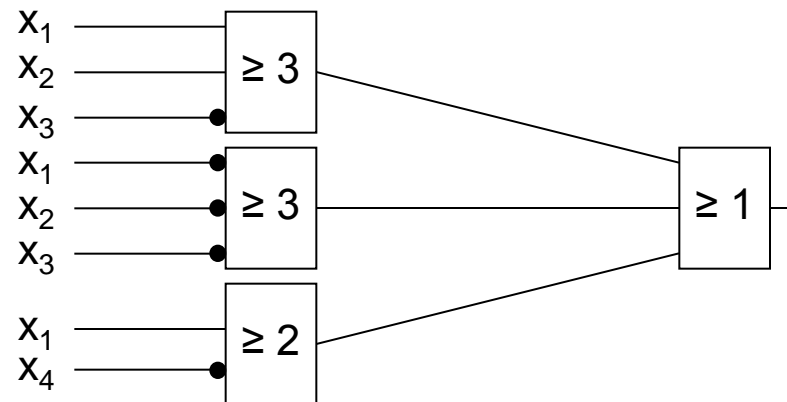
inputs also available in inverted form, i.e. \exists inverted inputs.



Theorem:

Every logical function $F: \mathbb{B}^n \rightarrow \mathbb{B}$ 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)

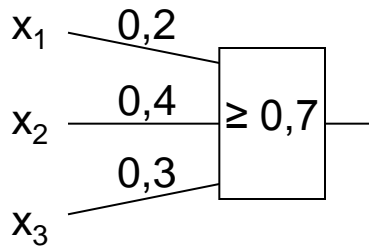
Every boolean function F can be transformed in disjunctive normal form

\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



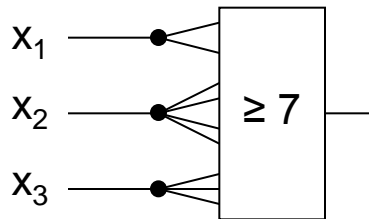
fires 1 if

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

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

⇓

duplicate inputs!



⇒ equivalent!

Theorem:

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

Proof:

„ \Rightarrow “ 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 $\prod_{i=1}^n b_i$ yields inequality with coefficients in \mathbb{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 $\theta = a_0 b_1 \cdots b_n$

„ \Leftarrow “

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
- 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 single MPP do?

$$w_1 x_1 + w_2 x_2 \geq \theta \begin{cases} \text{Y} & \rightarrow 1 \\ \text{N} & \rightarrow 0 \end{cases}$$

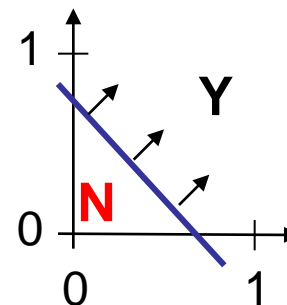
isolation of x_2 yields:

$$x_2 \geq \frac{\theta}{w_2} - \frac{w_1}{w_2} x_1 \begin{cases} \text{Y} & \rightarrow 1 \\ \text{N} & \rightarrow 0 \end{cases}$$

Example:

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

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

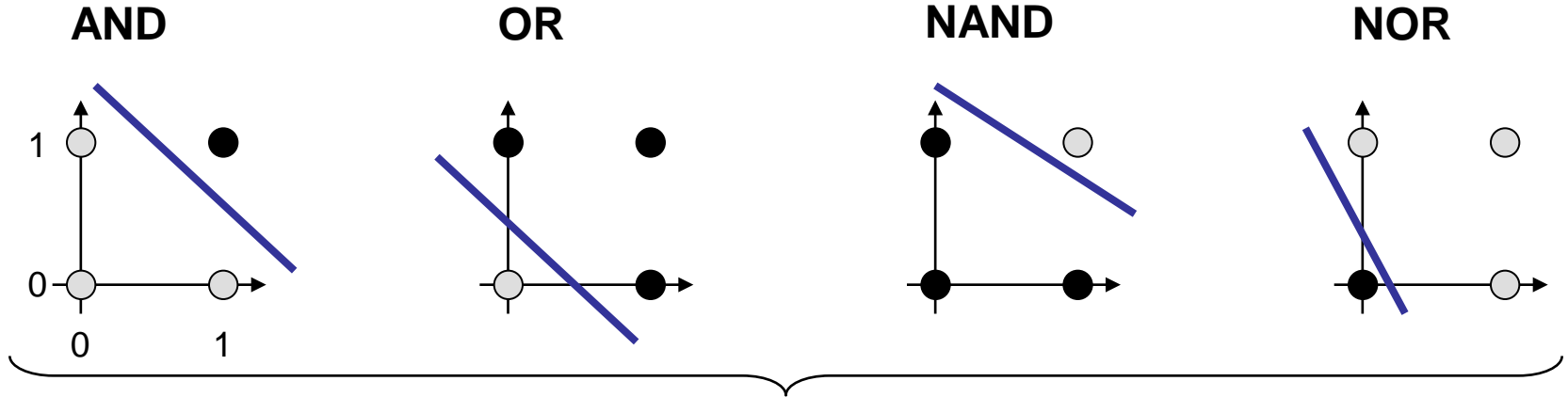


separating line

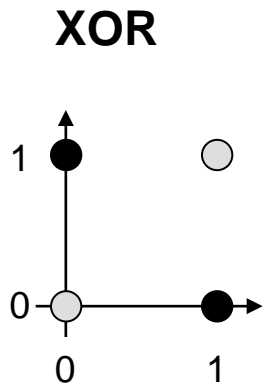
separates \mathbb{R}^2

in 2 classes

○ = 0 ● = 1



→ MPP at least as powerful as MCP neuron!



?

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

$\Rightarrow 0 < \theta$

$\Rightarrow w_2 \geq \theta$

$\Rightarrow w_1 \geq \theta$

$\Rightarrow w_1 + w_2 < \theta$

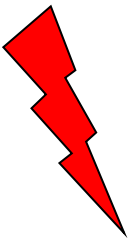
$w_1, w_2 \geq \theta > 0$
 $\Rightarrow w_1 + w_2 \geq 2\theta$

contradiction!

$w_1 x_1 + w_2 x_2 \geq \theta$

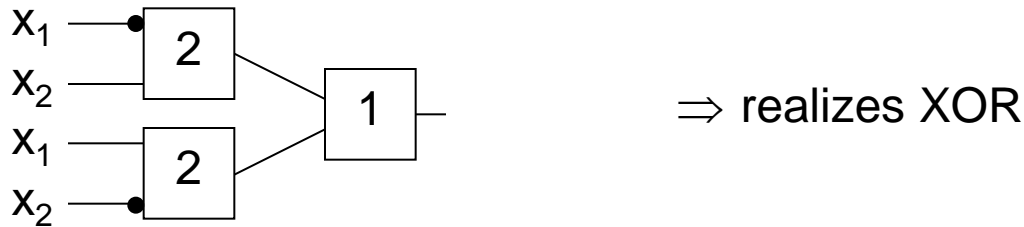
1969: Marvin Minsky / Seymour Papert

- 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“:

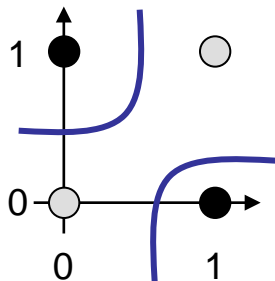
1. Multilayer Perceptrons:



2. Nonlinear separating functions:

XOR

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



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

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

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

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

How to obtain weights w_i and threshold θ ?

as yet: 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.: $w_1 = w_2 = -2, \theta = -3$)

now: by „learning“ / training

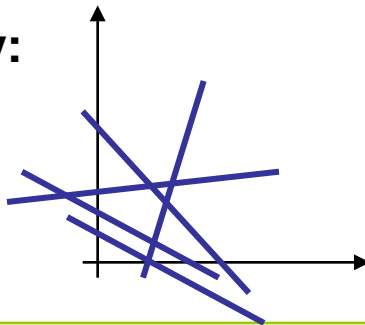
Perceptron Learning

Assumption: test examples with correct I/O behavior available

Principle:

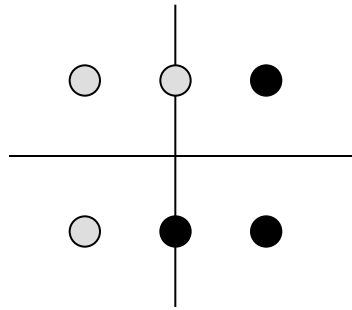
- (1) choose initial weights in arbitrary manner
- (2) feed in test pattern
- (3) if output of perceptron wrong, then change weights
- (4) goto (2) until correct output for all test patterns

graphically:



→ translation and rotation of separating lines

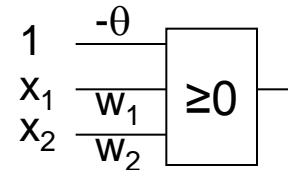
Example



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

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

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



⇓

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

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

suppose initial vector of weights is

$$w^{(0)} = (1, -1, 1)'$$

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 $w_t'x \leq 0$ then **goto 2**
4. if $x \in P$ and $w_t'x \leq 0$ then
 $w_{t+1} = w_t + x$; $t++$; **goto 2**
5. if $x \in N$ and $w_t'x > 0$ then
 $w_{t+1} = w_t - x$; $t++$; **goto 2**
6. stop? If I/O correct for all examples!

} I/O correct!

} let $w'x \leq 0$, should be > 0 !

$$(w+x)'x = w'x + x'x > w'x$$

} let $w'x > 0$, should be ≤ 0 !

$$(w-x)'x = w'x - x'x < w'x$$

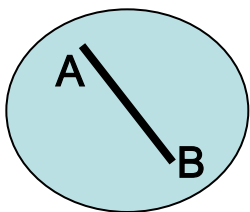
remark: algorithm converges, is finite, worst case: exponential runtime

We know what a single MPP can do.

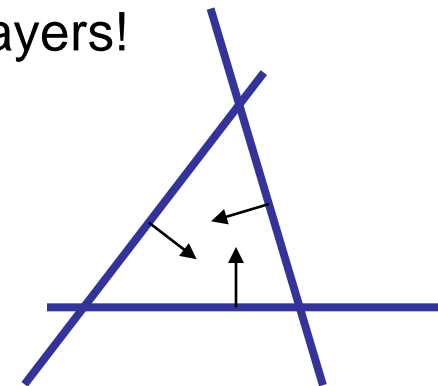
What can be achieved with many MPPs?

Single MPP \Rightarrow separates plane in two half planes

Many MPPs in 2 layers \Rightarrow can identify convex sets



1. How? \Rightarrow 2 layers!
 \Leftarrow 2. Convex?



$\forall a, b \in X:$

$\lambda a + (1-\lambda) b \in X$

for $\lambda \in (0,1)$

Single MPP	⇒ separates plane in two half planes
Many MPPs in 2 layers	⇒ can identify convex sets
Many MPPs in 3 layers	⇒ can identify arbitrary sets
Many MPPs in > 3 layers	⇒ not really necessary!

arbitrary sets:

1. partitioning of nonconvex set in several convex sets
2. two-layered subnet for each convex set
3. feed outputs of two-layered subnets in OR gate (third layer)