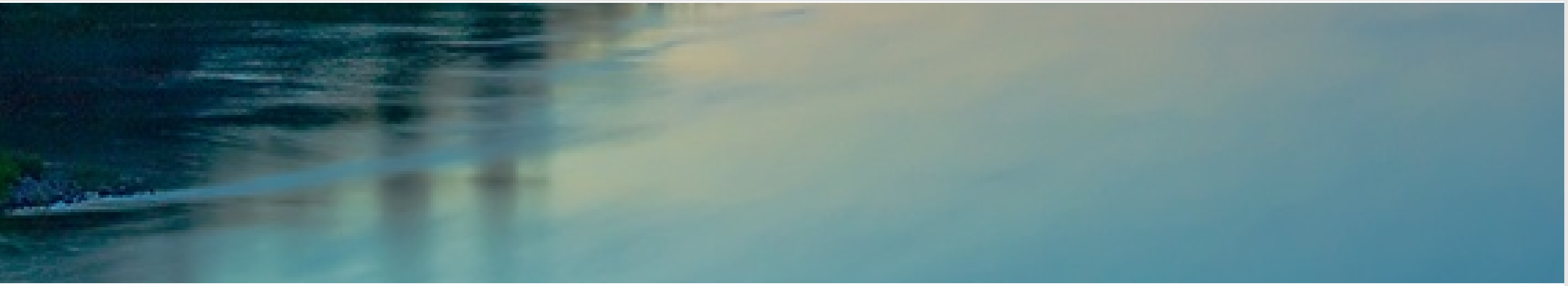


Neural Networks

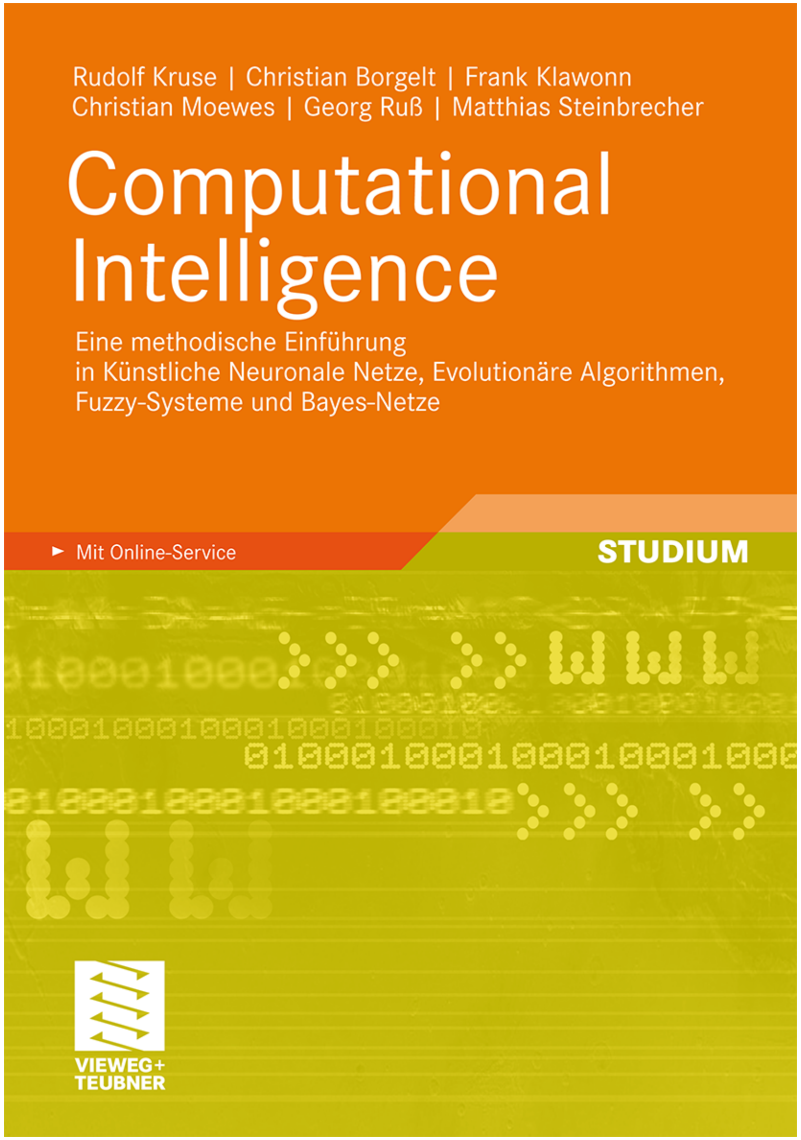
Prof. Dr. Rudolf Kruse

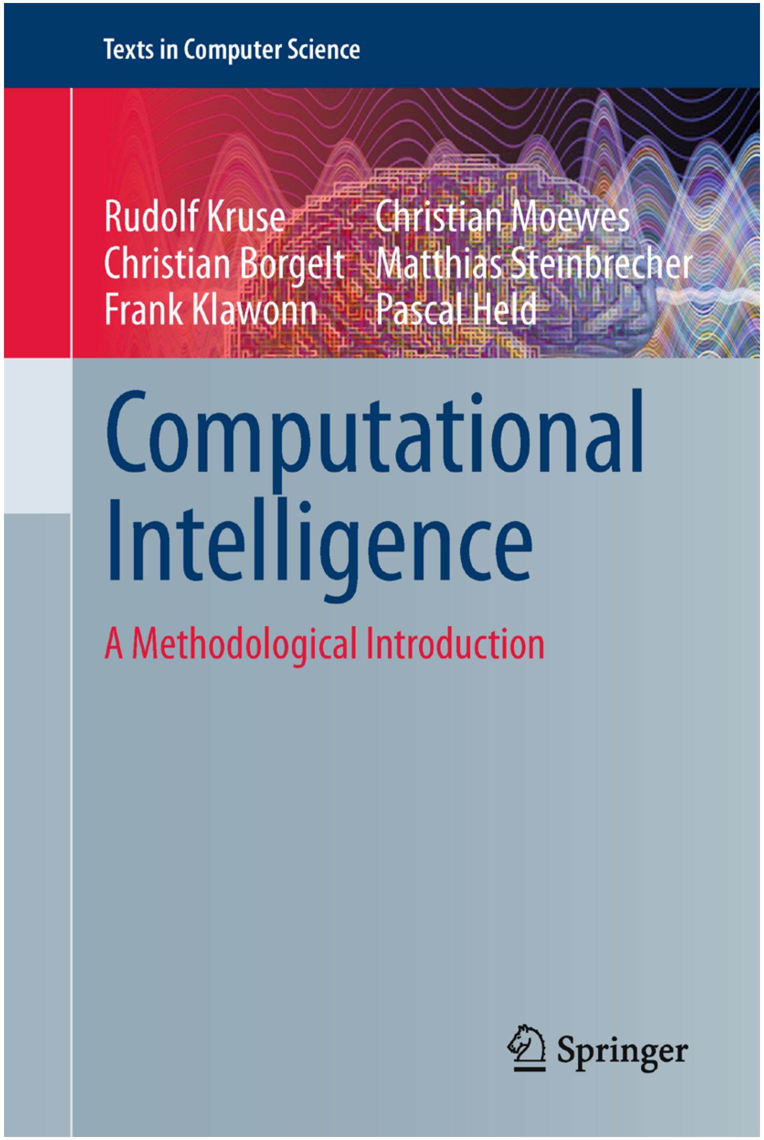
Computational Intelligence Group
Faculty for Computer Science
kruse@iws.cs.uni-magdeburg.de

About me: Rudolf Kruse

- 1979 diploma (Mathematics) degree from the University of Braunschweig, Germany
- 1980 PhD in Mathematics, 1984 the venia legendi in Mathematics from the same university
- 2 years at Fraunhofer Gesellschaft
- 1986 joined the University of Braunschweig as a professor in computer science
- Since 1996 professor at the Department of Computer Science at the University of Magdeburg
- **Research:** data mining, explorative data analysis, fuzzy-systems, neural networks, evolutionary algorithms, bayesian networks
- `mailto:kruse@iws.cs.uni-magdeburg.de`
- Office: G29-008, Telefon: 0391 67-58706
- Consultation: Mi., 11:00–12:00 Uhr

Bücher zur Vorlesung





Literatur zur Lehrveranstaltung

Kruse, Borgelt, Klawonn, Moewes, Steinbrecher und Held. *Computational Intelligence: A Methodological Introduction.* Springer, London, 2013.

Kruse, Borgelt, Klawonn, Moewes, Ruß und Steinbrecher. *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze.* Vieweg+Teubner, Wiesbaden, 2011.

Nauck, Borgelt, Klawonn und Kruse. *Neuro-Fuzzy-Systeme: Von den Grundlagen Neuronaler Netze zu modernen Fuzzy-Systemen.* Vieweg, Wiesbaden, 3. Aufl., 2003.

Rojas. *Theorie der neuronalen Netze: Eine systematische Einführung.* Springer, Berlin, 1993.

Zell. *Simulation neuronaler Netze.* Addison-Wesley, Bonn, 1994.

Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice-Hall, Upper Saddle River, NJ, 1994.

Kriesel. *Ein kleiner Überblick über neuronale Netze.* Manuskript, erhältlich auf http://www.dkriesel.com, 2007.

Motivation: Why (artificial) neural networks?

- **(Neuro-)Biology / (Neuro-)Physiology / Psychology:**
  - exploiting the similarities to real (biological) neural networks
  - modelling and simulation to gain understanding of the operations of nerves and brain

- **Computer Science / engineering / economy**
  - imitating human perception and processing
  - solving problems of learning and tuning as well as prognosis and optimization problems

- **Physics / Chemistry**
  - using neural networks for characterizing physical phenomena
  - special case: spin glass (alloying of magnetic and non-magnetic metals)

Conventional computers vs. The Brain

	Computer	Brain
processing units	1 CPU, 10^9 transistors	10^{11} neurons
storage capacity	10^9 Bytes RAM, 10^{10} Bytes non-volatile memory	10^{11} neurons, 10^{14} synapses
processing speed	10^{-8} sec.	10^{-3} sec.
bandwidth	$10^9 \frac{bits}{s}$	$10^{14} \frac{bits}{s}$
neural updates per sec.	10^5	10^{14}

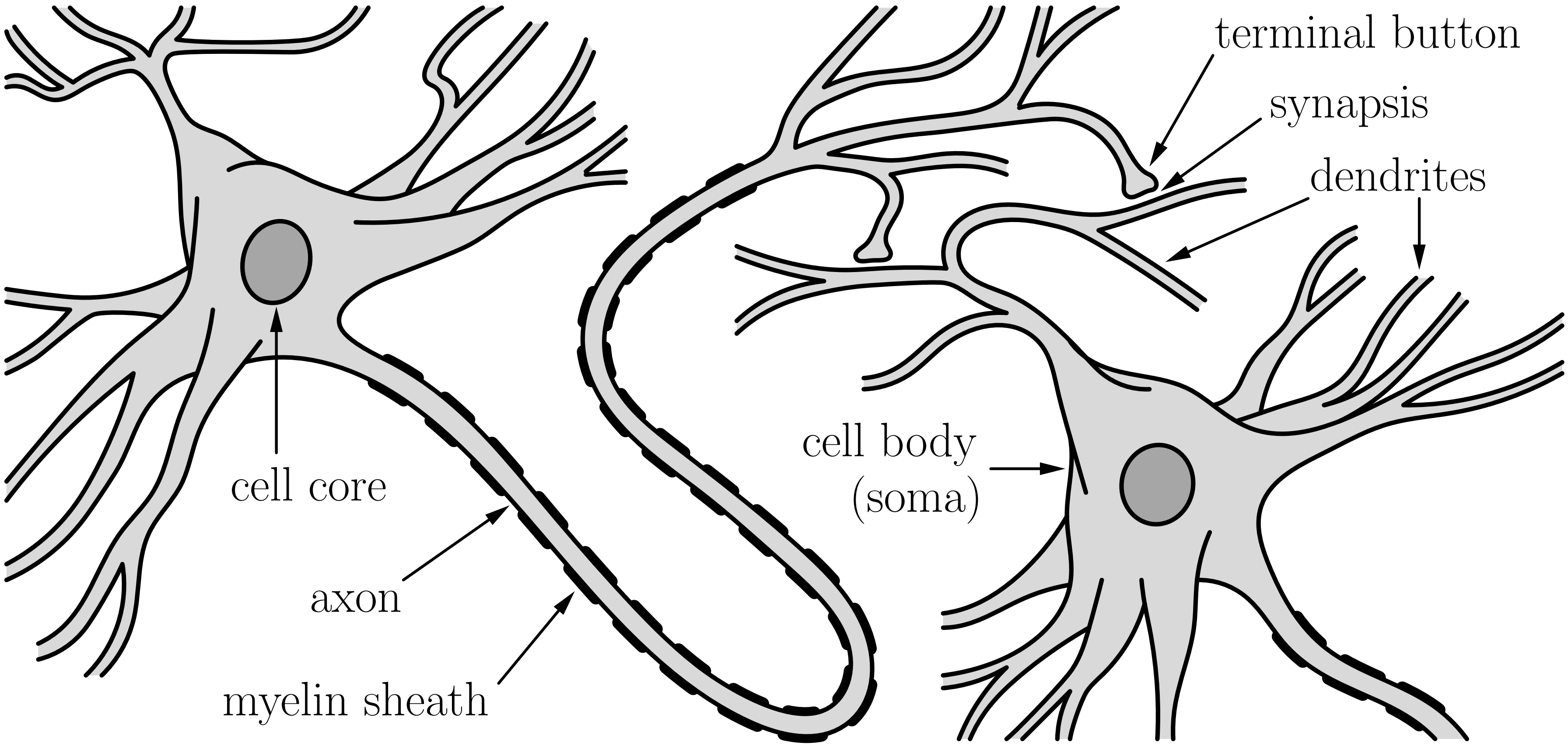
Conventional computers vs. The Brain

- Note: the switching times of the human brain are quite slow, being only 10^{-3} sec., but updates are processed in parallel. In contrast, serial PC simulations take several hundreds of processing cycles for one update.

- Advantages of neural networks:
  - great processing speed by making massively use of parallel processing
  - even after partial failure the network is still in service (fault tolerance)
  - with increasing amount of failing neurons just slow failure of entire system (*graceful degradation*)
  - well-suited for inductive learning

- Thus it seems promising to emulate these advantages by using artificial neural networks.

Structure of a prototypical biological neuron



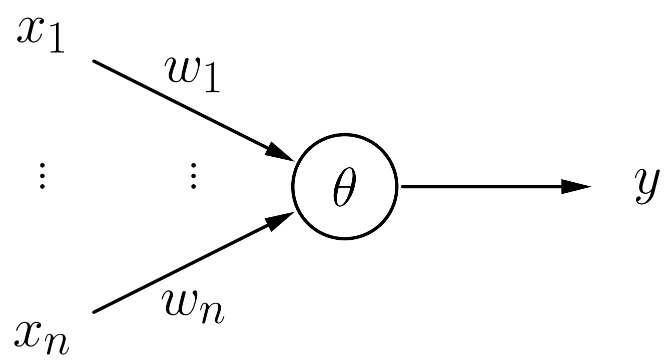
(Very) simplified description of neural information processing

- Axon terminal releases chemicals, called **neurotransmitters**.
- These act on the membrane of the receptor dendrite to change its polarization. (The inside is usually 70mV more negative than the outside.)
- Decrease in potential difference: **excitatory** synapse
  Increase in potential difference: **inhibitory** synapse
- If there is enough net excitatory input, the axon is depolarized.
- The resulting **action potential** travels along the axon. (Speed depends on the degree to which the axon is covered with myelin.)
- When the action potential reaches the terminal buttons, it triggers the release of neurotransmitters.

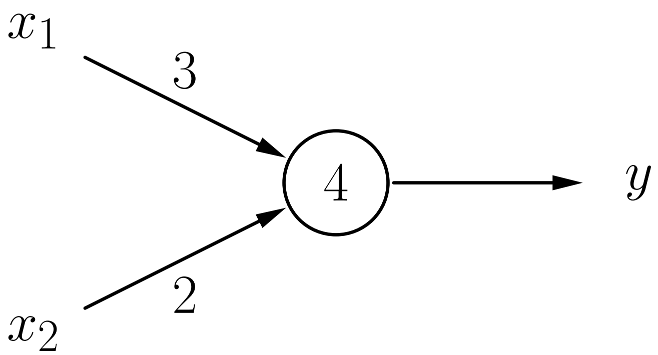
Threshold Logic Units

A **Threshold Logic Unit (TLU)** is a processing unit for numbers with n inputs $x_1, \ldots, x_n$ and one output y. The unit has a **threshold** θ and each input x_i is associated with a **weight** w_i. A threshold logic unit computes the function

$$y = \begin{cases} 1, & \text{if} \quad \vec{x}\vec{w} = \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

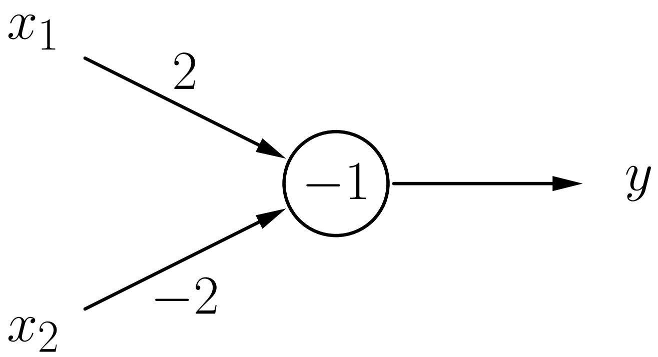


Threshold logic unit for the conjunction $x_1 \wedge x_2$.



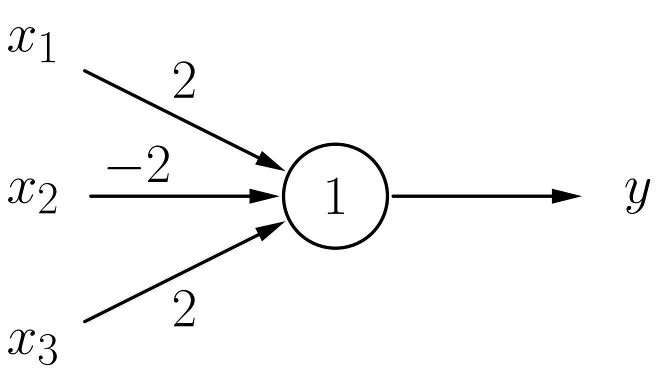
x_1	x_2	$3x_1 + 2x_2$	y
0	0	0	0
1	0	3	0
0	1	2	0
1	1	5	1

Threshold logic unit for the implication $x_2 \rightarrow x_1$.



x_1	x_2	$2x_1 - 2x_2$	y
0	0	0	1
1	0	2	1
0	1	-2	0
1	1	0	1

Threshold logic unit for $(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$**.**



x_1	x_2	x_3	$\sum_i w_i x_i$	y
0	0	0	0	0
1	0	0	2	1
0	1	0	-2	0
1	1	0	0	0
0	0	1	2	1
1	0	1	4	1
0	1	1	0	0
1	1	1	2	1

Review of line representations

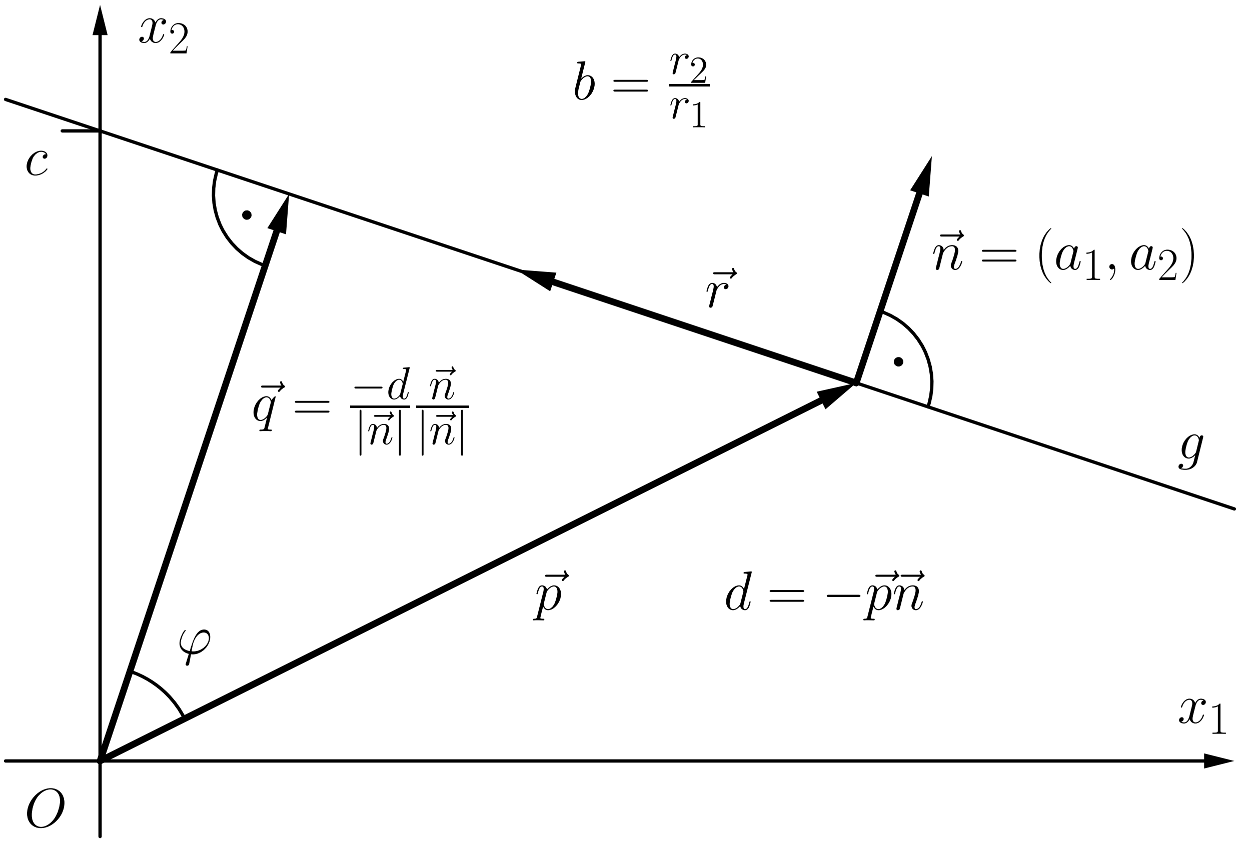
Straight lines are usually represented in one of the following forms:

$$\begin{array}{lrcl}
\text{Explicit Form:} & g & \equiv & x_2 = bx_1 + c \\
\text{Implicit Form:} & g & \equiv & a_1x_1 + a_2x_2 + d = 0 \\
\text{Point-Direction Form:} & g & \equiv & \vec{x} = \vec{p} + k\vec{r} \\
\text{Normal Form:} & g & \equiv & (\vec{x} - \vec{p})\vec{n} = 0
\end{array}$$

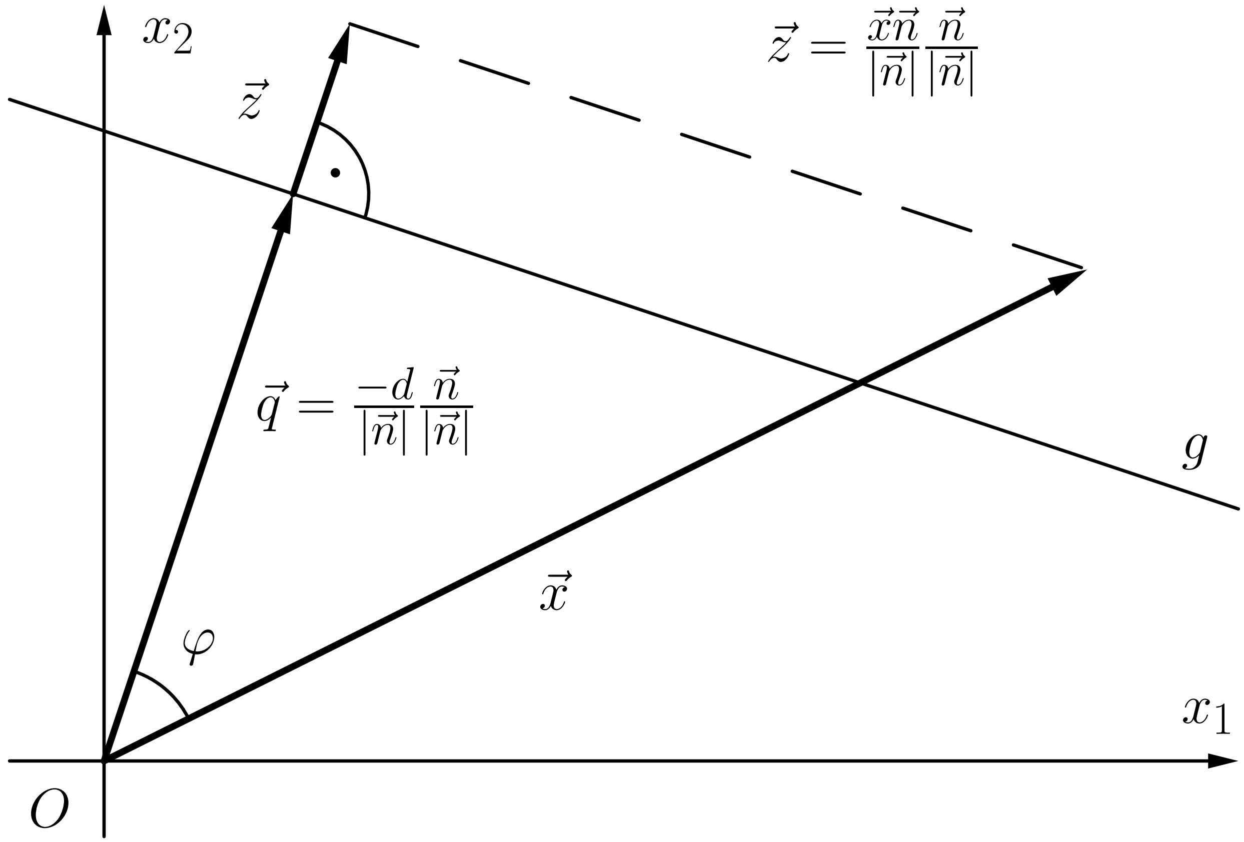
with the parameters:

b : Gradient of the line
c : Section of the x_2 axis
$\vec{p}$: Vector of a point of the line (base vector)
$\vec{r}$: Direction vector of the line
$\vec{n}$: Normal vector of the line

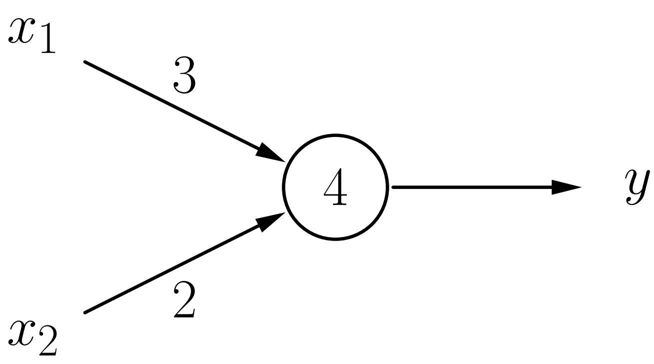
A straight line and its defining parameters.

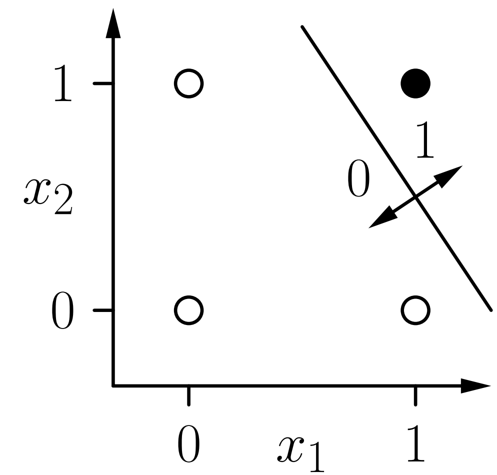


How to determine the side on which a point $\vec{x}$ lies.

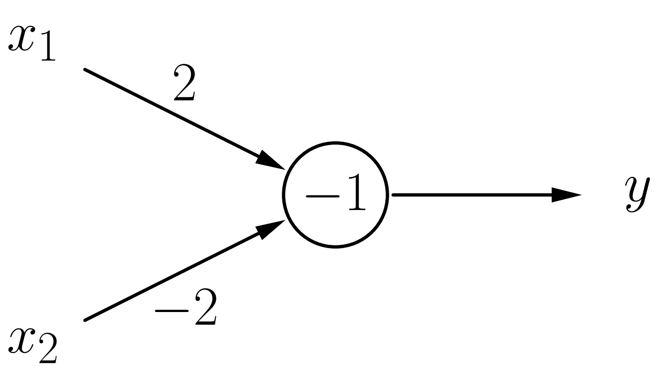


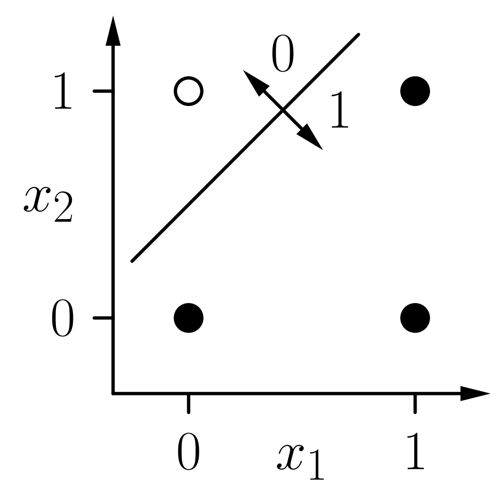
Threshold logic unit for $x_1 \wedge x_2$.



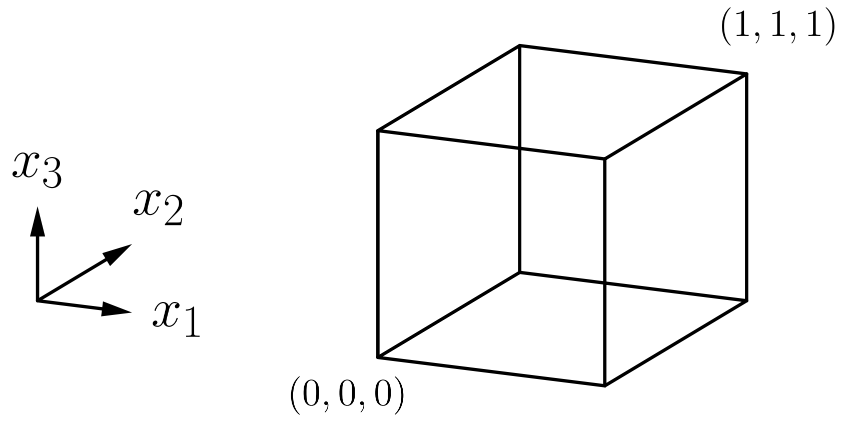


A threshold logic unit for $x_2 \rightarrow x_1$.

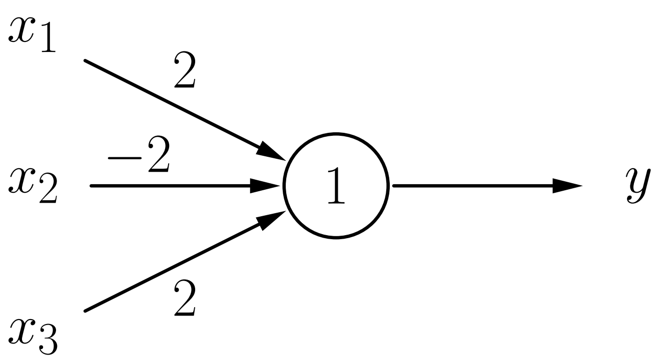


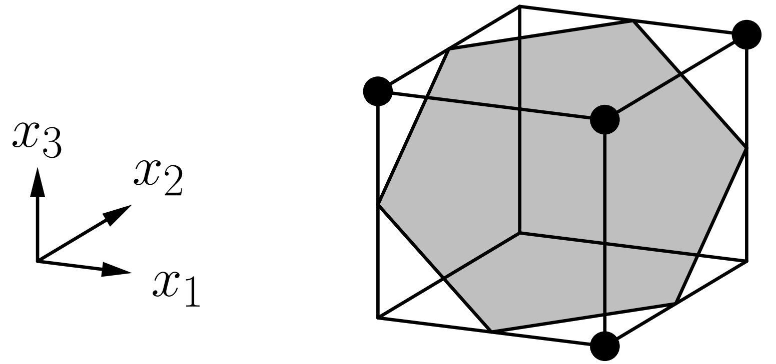


Visualization of 3-dimensional Boolean functions:



Threshold logic unit for $(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$**.**





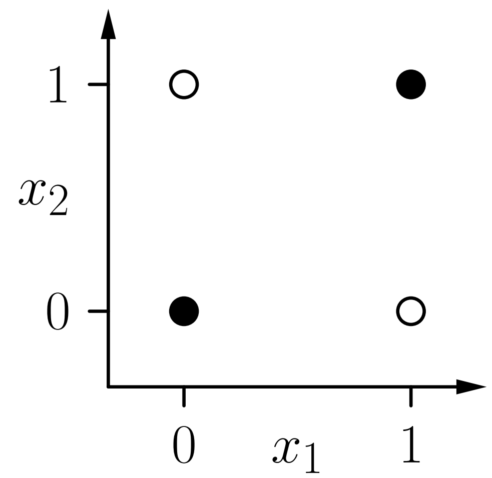
Threshold Logic Units: linear separability

- We call two sets of points in an n-dimensional space linearly separable, if they can separated by an (n-1)-dimensional hyperplane. One of these sets may contain points lying on the hyperplane, too.
- A boolean function is called linearly separable, if the set of fibers of 0 and the set of fibers of 1 are linearly separable.

Threshold Logic Units: Limitations

The biimplication problem $x_1 \leftrightarrow x_2$: There is no separating line.

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



Formal proof by *reductio ad absurdum*:

$$
\begin{array}{llll}
\text{since } (0,0) \mapsto 1: & 0 & \geq \theta, & (1) \\
\text{since } (1,0) \mapsto 0: & w_1 & < \theta, & (2) \\
\text{since } (0,1) \mapsto 0: & w_2 & < \theta, & (3) \\
\text{since } (1,1) \mapsto 1: & w_1 + w_2 & \geq \theta. & (4)
\end{array}
$$

(2) and (3): $w_1 + w_2 < 2\theta$. With (4): $2\theta > \theta$, or $\theta > 0$. Contradiction to (1).

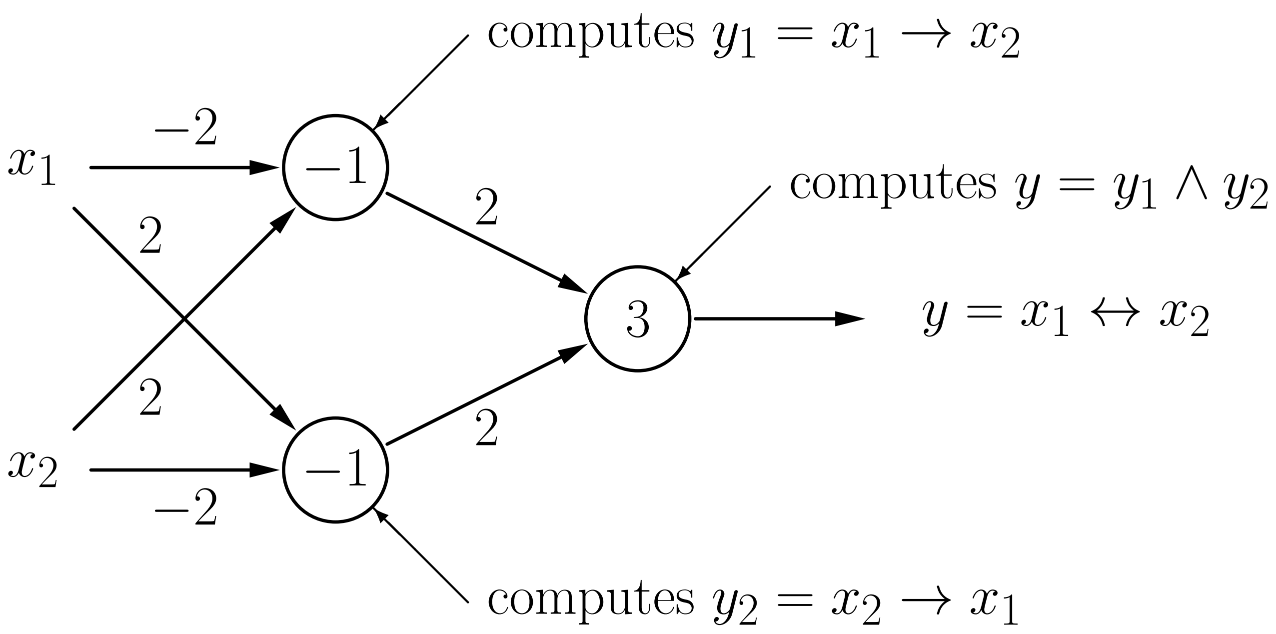
Total number and number of linearly separable Boolean functions.
([Widner 1960] as cited in [Zell 1994])

inputs	Boolean functions	linearly separable functions
1	4	4
2	16	14
3	256	104
4	65536	1774
5	$4.3 \cdot 10^9$	94572
6	$1.8 \cdot 10^{19}$	$5.0 \cdot 10^6$

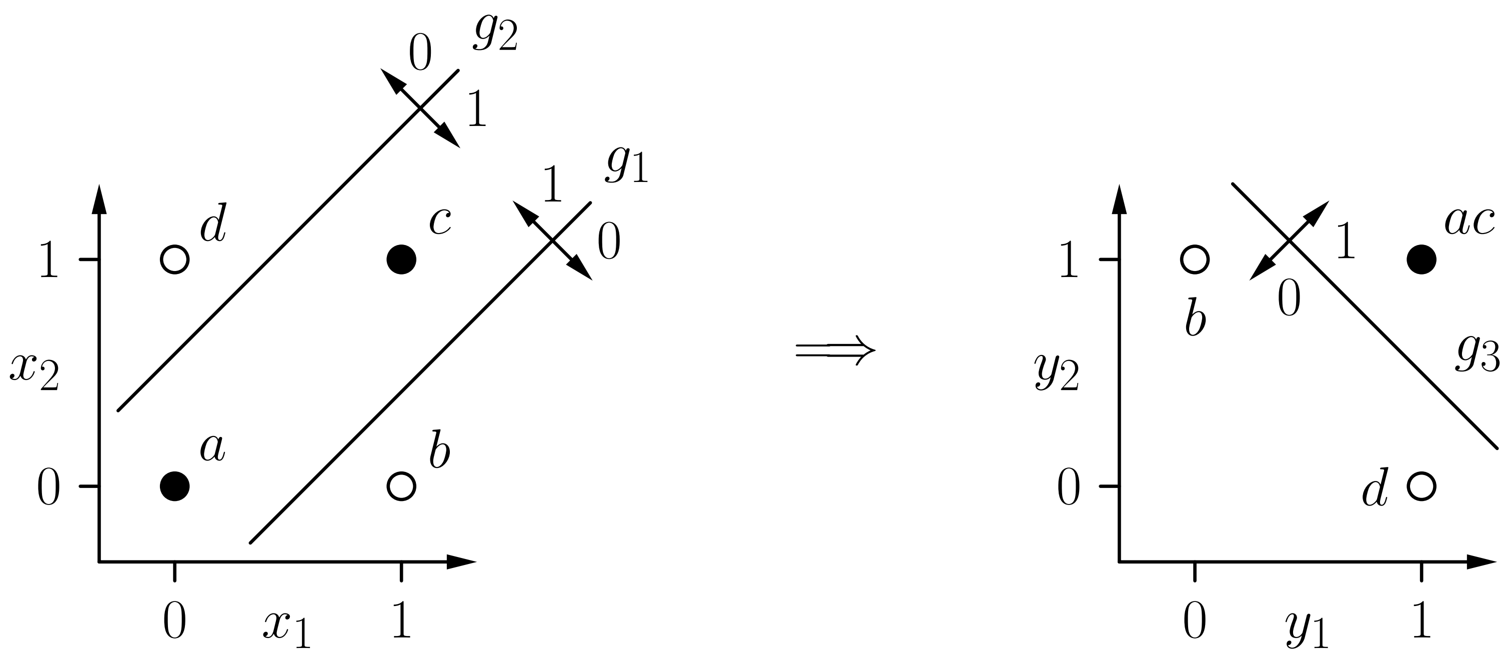
- For many inputs a threshold logic unit can compute almost no functions.
- Networks of threshold logic units are needed to overcome the limitations.

Solving the biimplication problem with a network.

Idea: logical decomposition $\quad x_1 \leftrightarrow x_2 \quad \equiv \quad (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_1)$



Solving the biimplication problem: Geometric interpretation



- The first layer computes new Boolean coordinates for the points.
- After the coordinate transformation the problem is linearly separable.

Representing Arbitrary Boolean Functions

Let $y = f(x_1, \ldots, x_n)$ be a Boolean function of n variables.

(i) Represent $f(x_1, \ldots, x_n)$ in disjunctive normal form. That is, determine $D_f = K_1 \vee \ldots \vee K_m$, where all K_j are conjunctions of n literals, i.e., $K_j = l_{j1} \wedge \ldots \wedge l_{jn}$ with $l_{ji} = x_i$ (positive literal) or $l_{ji} = \neg x_i$ (negative literal).

(ii) Create a neuron for each conjunction K_j of the disjunctive normal form (having n inputs — one input for each variable), where

$$w_{ji} = \begin{cases} 2, & \text{if } l_{ji} = x_i, \\ -2, & \text{if } l_{ji} = \neg x_i, \end{cases} \qquad \text{and} \qquad \theta_j = n - 1 + \frac{1}{2}\sum_{i=1}^{n} w_{ji}.$$

(iii) Create an output neuron (having m inputs — one input for each neuron that was created in step (ii)), where

$$w_{(n+1)k} = 2, \quad k = 1, \ldots, m, \qquad \text{and} \qquad \theta_{n+1} = 1.$$

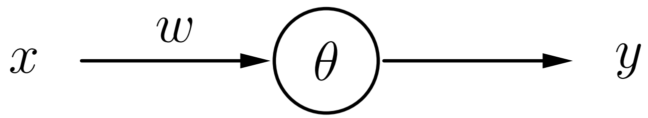
Training Threshold Logic Units

Training Threshold Logic Units

- Geometric interpretation provides a way to construct threshold logic units with 2 and 3 inputs, but:
  - Not an automatic method (human visualization needed).
  - Not feasible for more than 3 inputs.

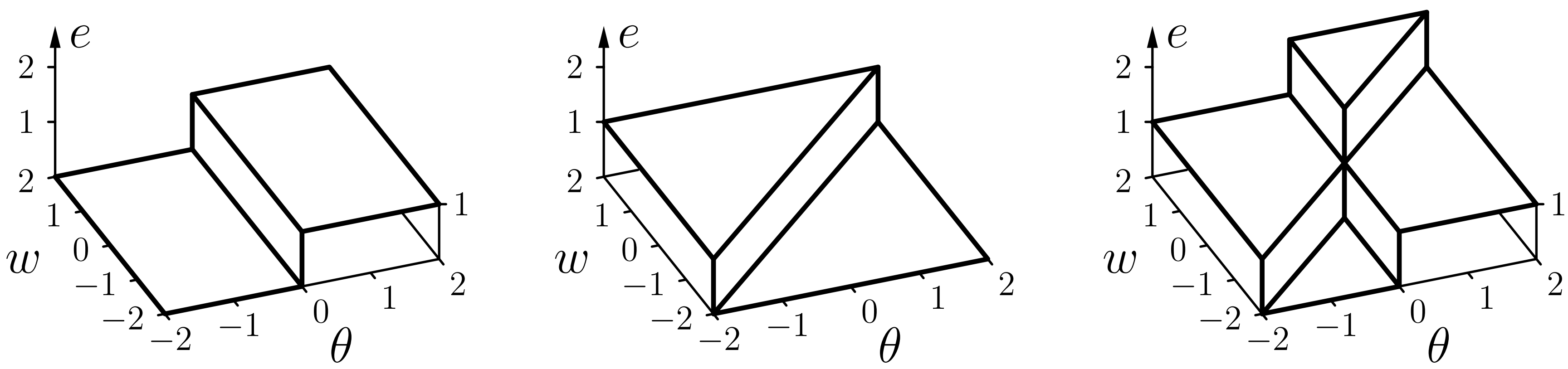
- **General idea of automatic training:**
  - Start with random values for weights and threshold.
  - Determine the error of the output for a set of training patterns.
  - Error is a function of the weights and the threshold: $e = e(w_1, \ldots, w_n, \theta)$.
  - Adapt weights and threshold so that the error gets smaller.
  - Iterate adaptation until the error vanishes.

Single input threshold logic unit for the negation $\neg x$.



x	y
0	1
1	0

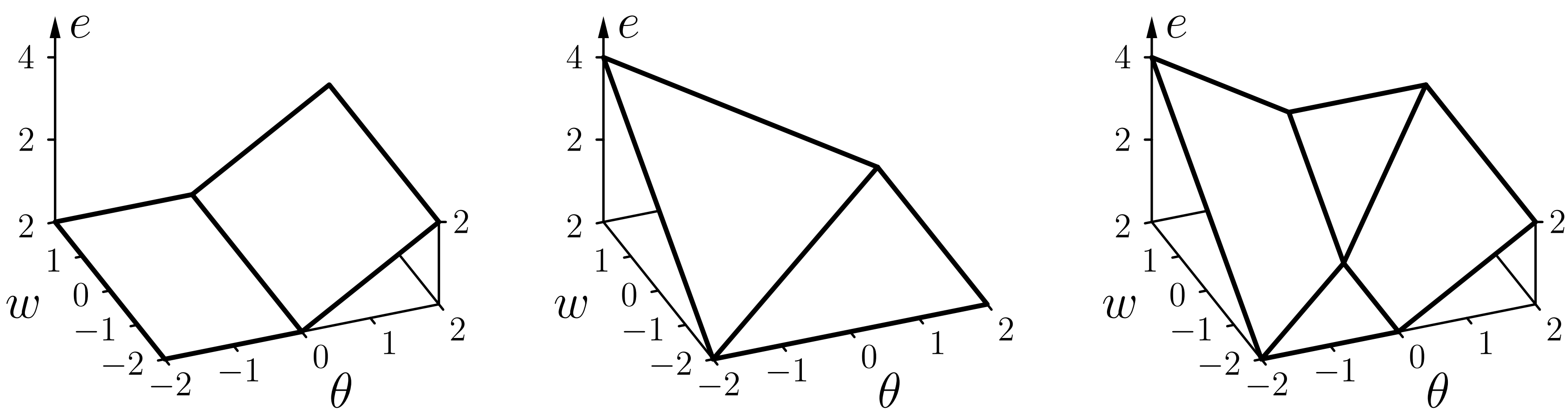
Output error as a function of weight and threshold.



error for $x = 0$ error for $x = 1$ sum of errors

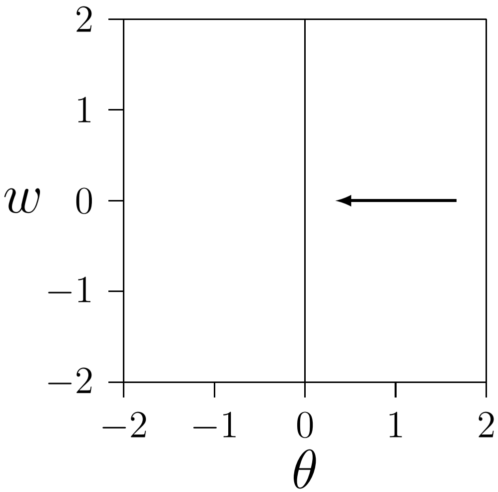
- The error function cannot be used directly, because it consists of plateaus.
- Solution: If the computed output is wrong,
  take into account, how far the weighted sum is from the threshold.

Modified output error as a function of weight and threshold.

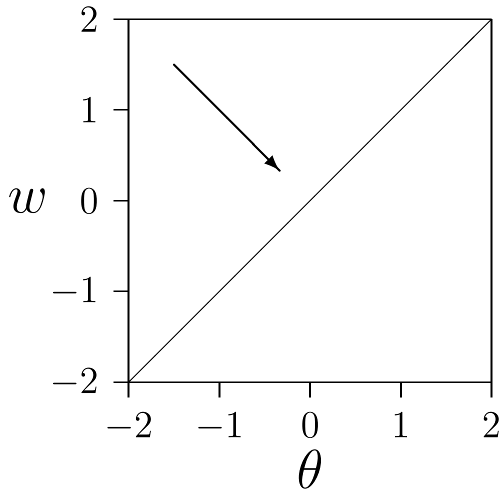


error for $x = 0$ error for $x = 1$ sum of errors

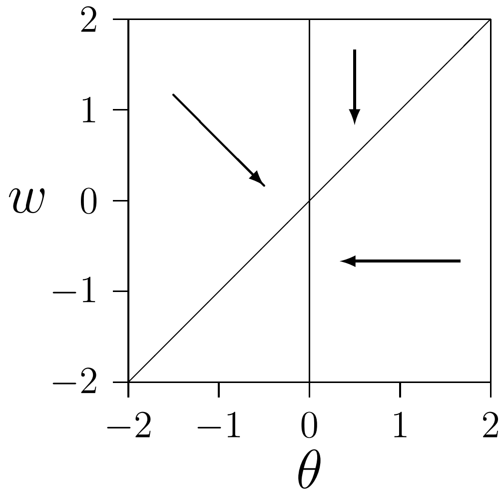
Schemata of resulting directions of parameter changes.



changes for $x = 0$



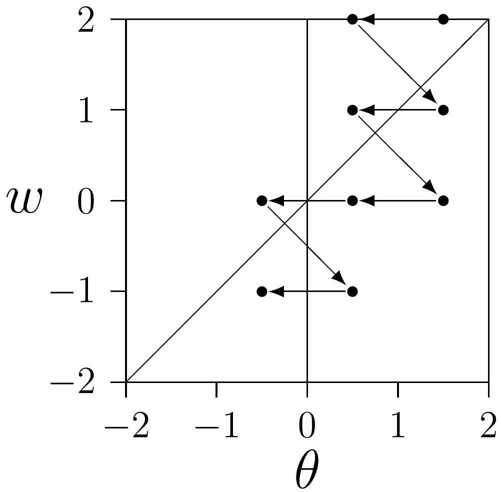
changes for $x = 1$



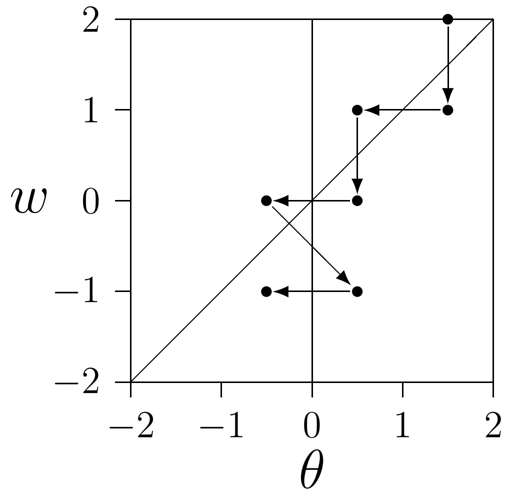
sum of changes

- Start at random point.
- Iteratively adapt parameters according to the direction corresponding to the current point.

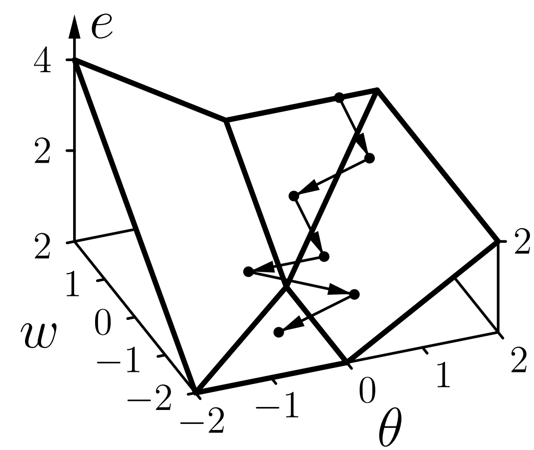
Example training procedure: Online and batch training.



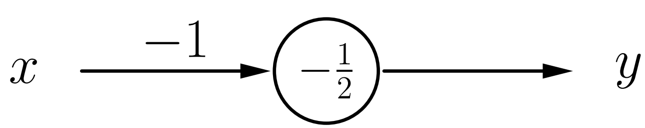
Online-Lernen

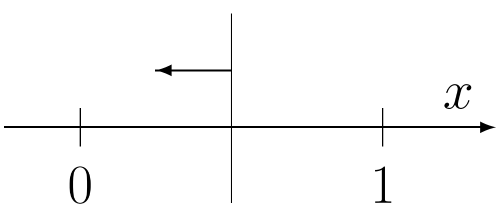


Batch-Lernen



Batch-Lernen





Training Threshold Logic Units: Delta Rule

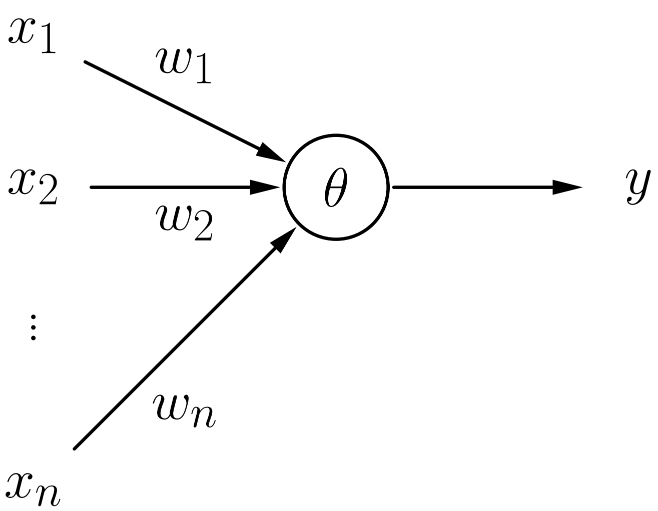
Formal Training Rule: Let $\vec{x} = (x_1, \ldots, x_n)$ be an input vector of a threshold logic unit, o the desired output for this input vector and y the actual output of the threshold logic unit. If $y \neq o$, then the threshold θ and the weight vector $\vec{w} = (w_1, \ldots, w_n)$ are adapted as follows in order to reduce the error:

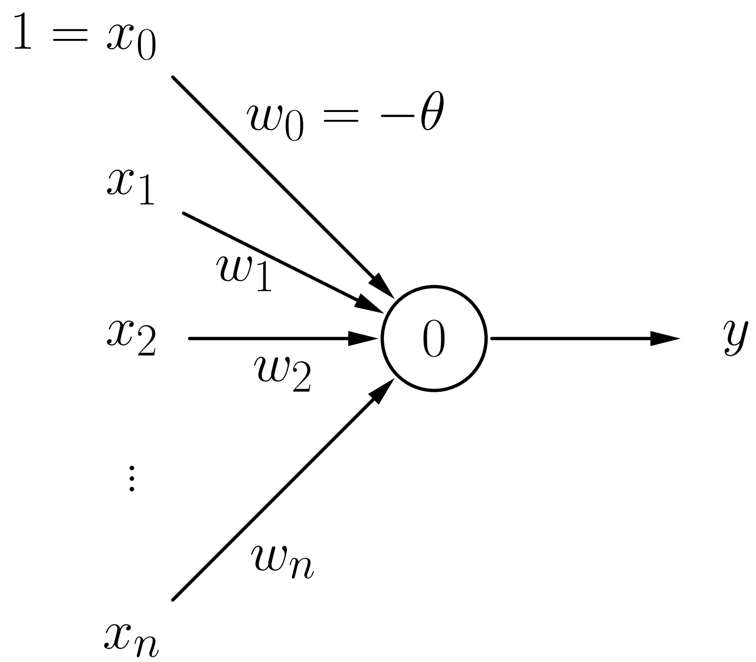
$$\begin{aligned} \theta^{(\text{new})} &= \theta^{(\text{old})} + \Delta\theta \quad \text{with} \quad \Delta\theta = -\eta(o - y), \\ \forall i \in \{1, \ldots, n\}: \quad w_i^{(\text{new})} &= w_i^{(\text{old})} + \Delta w_i \quad \text{with} \quad \Delta w_i = \eta(o - y)x_i, \end{aligned}$$

where η is a parameter that is called **learning rate**. It determines the severity of the weight changes. This procedure is called **Delta Rule** or **Widrow–Hoff Procedure** [Widrow and Hoff 1960].

- **Online Training:** Adapt parameters after each training pattern.
- **Batch Training:** Adapt parameters only at the end of each **epoch**, i.e. after a traversal of all training patterns.

Turning the threshold value into a weight:





$$\sum_{i=1}^{n} w_i x_i \geq \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta \geq 0$$

Training Threshold Logic Units: Delta Rule

```
procedure online_training (var w⃗, var θ, L, η);
var y, e;                                  (* output, sum of errors *)
begin
  repeat
    e := 0;                                (* initialize the error sum *)
    for all (x⃗, o) ∈ L do begin            (* traverse the patterns *)
      if (w⃗x⃗ ≥ θ) then  y := 1;           (* compute the output *)
                 else  y := 0;            (* of the threshold logic unit *)
      if (y ≠ o) then begin               (* if the output is wrong *)
        θ  := θ  − η(o − y);              (* adapt the threshold *)
        w⃗ := w⃗ + η(o − y)x⃗;              (* and the weights *)
        e  := e  +  |o − y|;              (* sum the errors *)
      end;
    end;
  until (e ≤ 0);                          (* repeat the computations *)
end;                                      (* until the error vanishes *)
```

Training Threshold Logic Units: Delta Rule

```
procedure batch_training (var w⃗, var θ, L, η);
var y, e,                                   (* output, sum of errors *)
    θ_c, w⃗_c;                               (* summed changes *)
begin
  repeat
    e := 0; θ_c := 0; w⃗_c := 0⃗;             (* initializations *)
    for all (x⃗, o) ∈ L do begin             (* traverse the patterns *)
      if (w⃗x⃗ ≥ θ) then  y := 1;             (* compute the output *)
                  else  y := 0;             (* of the threshold logic unit *)
      if (y ≠ o) then begin                 (* if the output is wrong *)
        θ_c  := θ_c − η(o − y);             (* sum the changes of the *)
        w⃗_c := w⃗_c + η(o − y)x⃗;             (* threshold and the weights *)
        e    := e  +  |o − y|;              (* sum the errors *)
      end;
    end;
    θ  := θ  + θ_c;                         (* adapt the threshold *)
    w⃗ := w⃗ + w⃗_c;                           (* and the weights *)
  until (e ≤ 0);                            (* repeat the computations *)
end;                                        (* until the error vanishes *)
```

Training Threshold Logic Units: Online

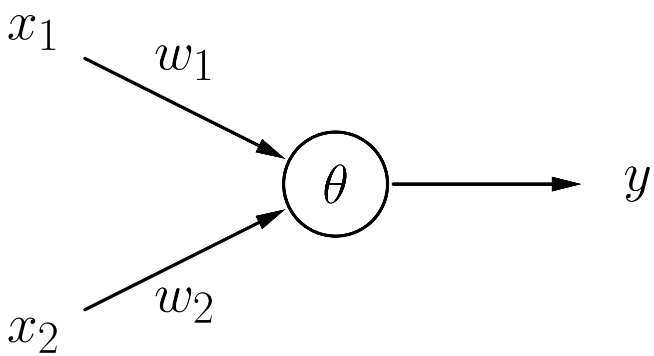
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0	0.5	2
	1	0	1.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0	0.5	1
	1	0	0.5	1	-1	1	-1	1.5	0
3	0	1	-1.5	0	1	-1	0	0.5	0
	1	0	0.5	0	0	0	0	0.5	0
4	0	1	-0.5	0	1	-1	0	-0.5	0
	1	0	0.5	1	-1	1	-1	0.5	-1
5	0	1	-0.5	0	1	-1	0	-0.5	-1
	1	0	-0.5	0	0	0	0	-0.5	-1
6	0	1	0.5	1	0	0	0	-0.5	-1
	1	0	-0.5	0	0	0	0	-0.5	-1

Training Threshold Logic Units: Batch

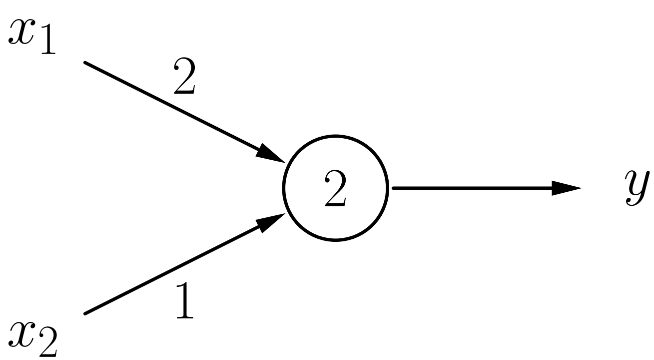
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	0.5	1
3	0	1	-0.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	0.5	0
4	0	1	-0.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	-0.5	0
5	0	1	0.5	1	0	0	0		
	1	0	0.5	1	-1	1	-1	0.5	-1
6	0	1	-0.5	0	1	-1	0		
	1	0	-1.5	0	0	0	0	-0.5	-1
7	0	1	0.5	1	0	0	0		
	1	0	-0.5	0	0	0	0	-0.5	-1

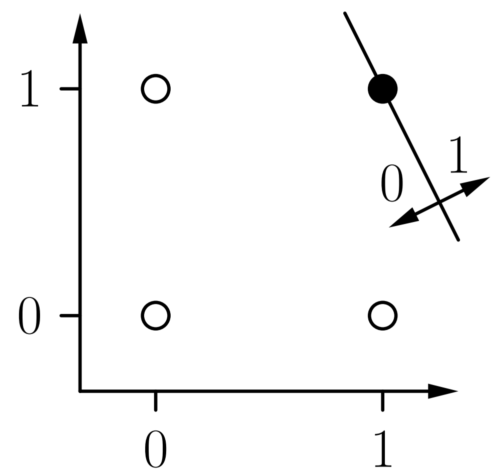
Training Threshold Logic Units: Conjunction

Threshold logic unit with two inputs for the conjunction.



x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1





Training Threshold Logic Units: Conjunction

epoch	x_1	x_2	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2
										0	0	0
1	0	0	0	0	1	-1	1	0	0	1	0	0
	0	1	0	-1	0	0	0	0	0	1	0	0
	1	0	0	-1	0	0	0	0	0	1	0	0
	1	1	1	-1	0	1	-1	1	1	0	1	1
2	0	0	0	0	1	-1	1	0	0	1	1	1
	0	1	0	0	1	-1	1	0	-1	2	1	0
	1	0	0	-1	0	0	0	0	0	2	1	0
	1	1	1	-1	0	1	-1	1	1	1	2	1
3	0	0	0	-1	0	0	0	0	0	1	2	1
	0	1	0	0	1	-1	1	0	-1	2	2	0
	1	0	0	0	1	-1	1	-1	0	3	1	0
	1	1	1	-2	0	1	-1	1	1	2	2	1
4	0	0	0	-2	0	0	0	0	0	2	2	1
	0	1	0	-1	0	0	0	0	0	2	2	1
	1	0	0	0	1	-1	1	-1	0	3	1	1
	1	1	1	-1	0	1	-1	1	1	2	2	2
5	0	0	0	-2	0	0	0	0	0	2	2	2
	0	1	0	0	1	-1	1	0	-1	3	2	1
	1	0	0	-1	0	0	0	0	0	3	2	1
	1	1	1	0	1	0	0	0	0	3	2	1
6	0	0	0	-3	0	0	0	0	0	3	2	1
	0	1	0	-2	0	0	0	0	0	3	2	1
	1	0	0	-1	0	0	0	0	0	3	2	1
	1	1	1	0	1	0	0	0	0	3	2	1

Training Threshold Logic Units: Biimplication

epoch	x_1	x_2	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2
										0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0
	0	1	0	0	1	-1	1	0	-1	1	0	-1
	1	0	0	-1	0	0	0	0	0	1	0	-1
	1	1	1	-2	0	1	-1	1	1	0	1	0
2	0	0	1	0	1	0	0	0	0	0	1	0
	0	1	0	0	1	-1	1	0	-1	1	1	-1
	1	0	0	0	1	-1	1	-1	0	2	0	-1
	1	1	1	-3	0	1	-1	1	1	1	1	0
3	0	0	1	0	1	0	0	0	0	0	1	0
	0	1	0	0	1	-1	1	0	-1	1	1	-1
	1	0	0	0	1	-1	1	-1	0	2	0	-1
	1	1	1	-3	0	1	-1	1	1	1	1	0

Convergence Theorem: Let $L = \{(\vec{x}_1, o_1), \ldots (\vec{x}_m, o_m)\}$ be a set of training patterns, each consisting of an input vector $\vec{x}_i \in \mathbb{R}^n$ and a desired output $o_i \in \{0, 1\}$. Furthermore, let $L_0 = \{(\vec{x}, o) \in L \mid o = 0\}$ and $L_1 = \{(\vec{x}, o) \in L \mid o = 1\}$. If L_0 and L_1 are linearly separable, i.e., if $\vec{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ exist, such that

$$\begin{aligned} \forall(\vec{x}, 0) \in L_0 : \quad & \vec{w}\vec{x} < \theta \qquad \text{and} \\ \forall(\vec{x}, 1) \in L_1 : \quad & \vec{w}\vec{x} \geq \theta, \end{aligned}$$

then online as well as batch training terminate.

- The algorithms terminate only when the error vanishes.
- Therefore the resulting threshold and weights must solve the problem.
- For not linearly separable problems the algorithms do not terminate.

Training Networks of Threshold Logic Units

- Single threshold logic units have strong limitations:
  They can only compute linearly separable functions.

- Networks of threshold logic units can compute arbitrary Boolean functions.

- Training single threshold logic units with the delta rule is fast
  and guaranteed to find a solution if one exists.

- Networks of threshold logic units cannot be trained, because

  - there are no desired values for the neurons of the first layer,

  - the problem can usually be solved with different functions
    computed by the neurons of the first layer.

- When this situation became clear,
  neural networks were seen as a "research dead end".