Neuronale Netze

Prof. Dr. Rudolf Kruse

Computational Intelligence
Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
rudolf.kruse@ovgu.de

Aktuelle Motivation: Alphago

2016: Alphago besiegt Lee Sedol 4:1

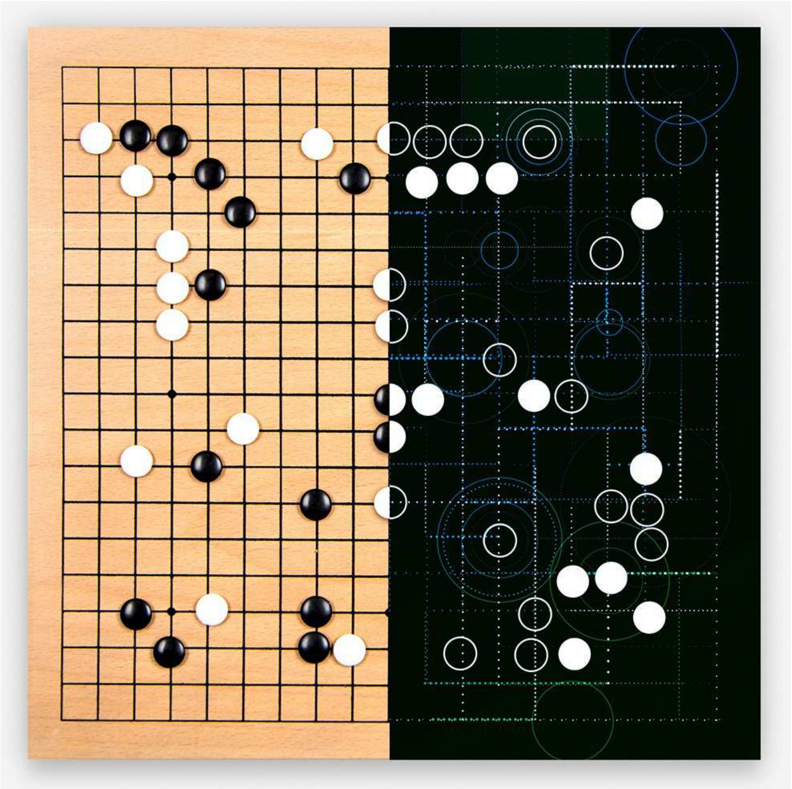
Es nutzt dabei tiefe Neuronale Netze:

Policy-Networks zur Vorauswahl möglicher Züge

Value-Networks zur Bewertung von Spielpositionen

Monte-Carlo Baumsuche zum Finden der besten Züge

Reinforcement Learning zur Verbesserung der Neuronalen Netze



Motivation: Warum (künstliche) neuronale Netze?

(Neuro-)Biologie / (Neuro-)Physiologie / Psychologie:

- Ausnutzung der Ähnlichkeit zu echten (biologischen) neuronalen Netzen
- Modellierung zum Verständnis Arbeitsweise von Nerven und Gehirn durch Simulation

Informatik / Ingenieurwissenschaften / Wirtschaft

- Nachahmen der menschlichen Wahrnehmung und Verarbeitung
- Lösen von Lern-/Anpassungsproblemen sowie Vorhersage- und Optimierungsproblemen

Physik / Chemie

- Nutzung neuronaler Netze, um physikalische Phänomene zu beschreiben
- Spezialfall: Spin-Glas (Legierungen von magnetischen und nicht-magnetischen Metallen)

Konventionelle Rechner vs. Gehirn

	Computer	Gehirn
Verarbeitungseinheiten	1 CPU, 10^{10} Transistoren	10^{11} Neuronen
Speicherkapazität	10^{11} Bytes RAM, 10^{13} Bytes Festspeicher	10^{11} Neuronen, 10^{14} Synapsen
Verarbeitungsgeschwindigkeit	10^{-8} Sekunden	10^{-3} Sekunden
Bandbreite	$10^{12}\frac{bits}{s}$	$10^{14}\frac{bits}{s}$
Neuronale Updates pro Sekunde	10^{6}	10^{14}

Konventionelle Rechner vs. Gehirn

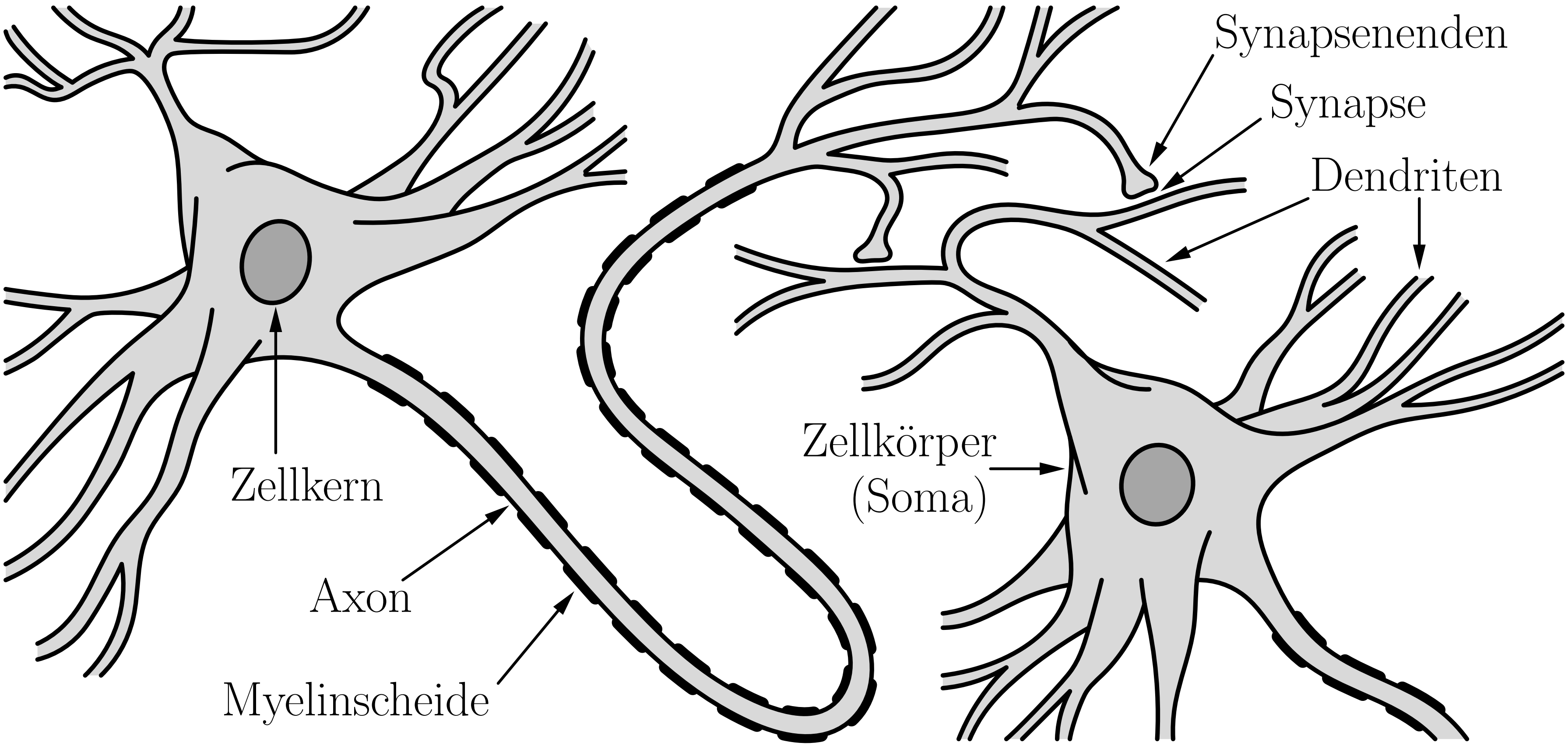
Beachte: die Hirnschaltzeit ist mit 10^{-3} s recht langsam, aber Updates erfolgen parallel. Dagegen braucht die serielle Simulation auf einem Rechner mehrere hundert Zyklen für ein Update.

Vorteile neuronaler Netze:

- Hohe Verarbeitungsgeschwindigkeit durch massive Parallelität
- Funktionstüchtigkeit selbst bei Ausfall von Teilen des Netzes (Fehlertoleranz)
- Langsamer Funktionsausfall bei fortschreitenden Ausfällen von Neuronen (*graceful degradation*)
- Gut geeignet für induktives Lernen

Es erscheint daher sinnvoll, diese Vorteile natürlicher neuronaler Netze künstlich nachzuahmen.

Struktur eines prototypischen biologischen Neurons



(Stark) vereinfachte Beschreibung neuronaler Informationsverarbeitung

Das Axonende gibt Chemikalien ab, **Neurotransmitter** genannt.

Diese bewirken an der Membran des Empfängerdendriten die Veränderung der Polarisierung.
(Das Innere ist typischerweise 70mV negativer als die Außenseite.)

Abnahme in der Potentialdifferenz: **anregende** Synapse
Zunahme in der Potentialdifferenz: **hemmende** Synapse

Wenn genügend anregende Information vorhanden ist, wird das Axon depolarisiert.

Das resultierende **Aktionspotential** pflanzt sich entlang des Axons fort.
(Die Geschwindigkeit hängt von der Bedeckung mit Myelin ab.)

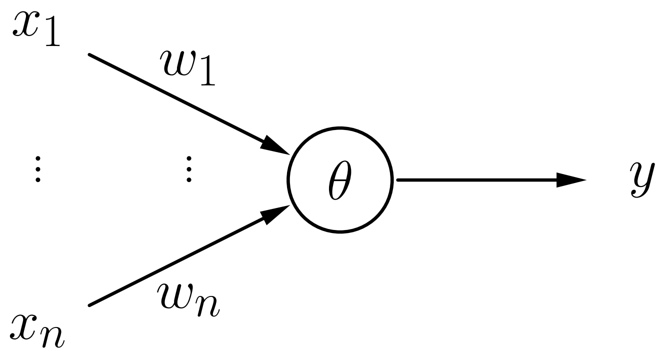
Wenn das Aktionspotential die Synapsenenden erreicht,
löst es die Abgabe von Neurotransmittern aus.

Schwellenwertelemente

Schwellenwertelemente

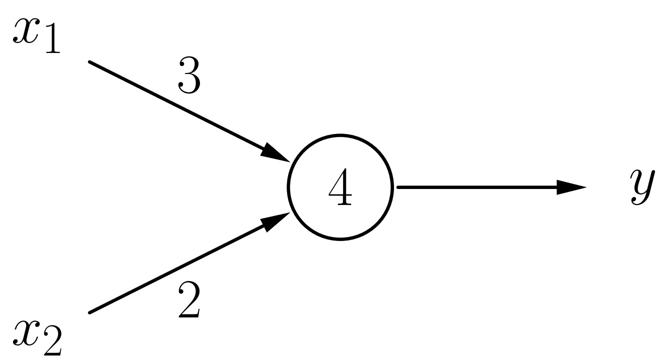
Ein **Schwellenwertelement** (Threshold Logic Unit, TLU) ist eine Verarbeitungseinheit für Zahlen mit n Eingängen $x_1, \ldots, x_n$ und einem Ausgang y. Das Element hat einen **Schwellenwert** θ und jeder Eingang x_i ist mit einem **Gewicht** w_i versehen. Ein Schwellenwertelement berechnet die Funktion

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



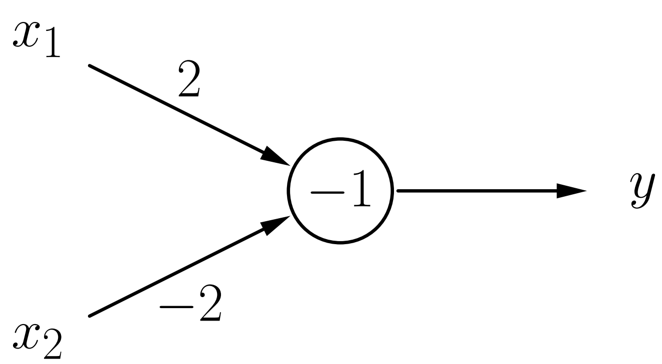
Schwellenwertelemente: Beispiele

Schwellenwertelement für die Konjunktion $x_1 \wedge x_2$.



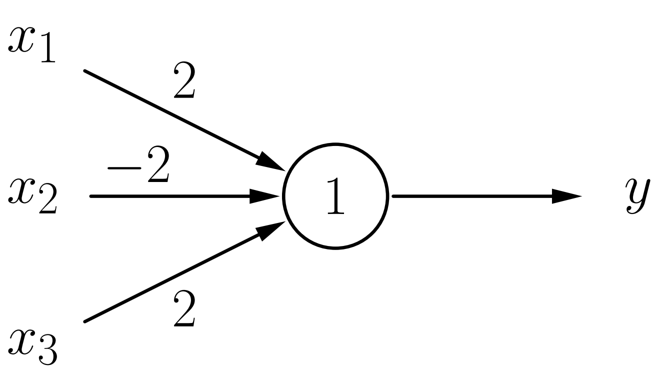
x_1	x_2	$3x_1 + 2x_2$	y
0	0	0	0
1	0	3	0
0	1	2	0
1	1	5	1

Schwellenwertelement für die Implikation $x_2 \rightarrow x_1$.



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

Schwellenwertelement für $(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$.



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

Rückblick: Geradendarstellungen

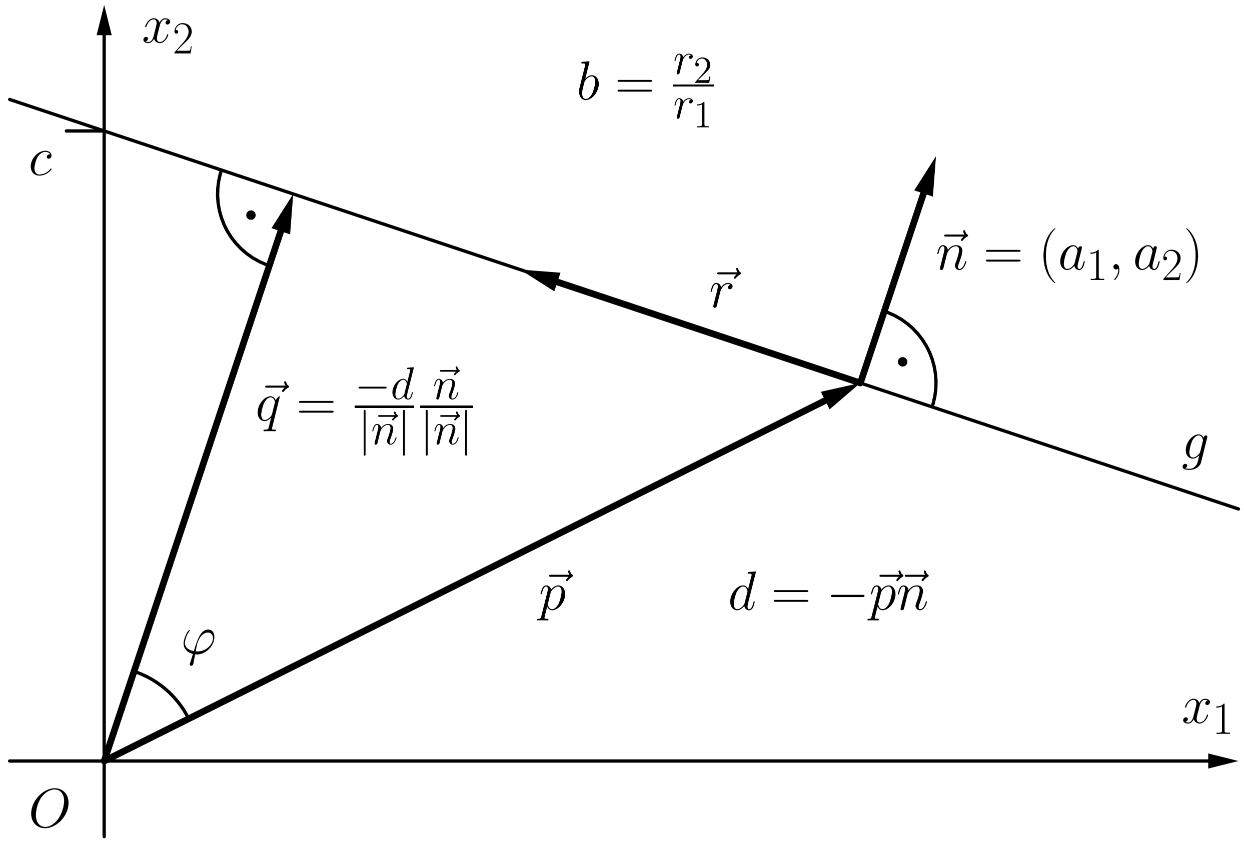
Geraden werden typischerweise in einer der folgenden Formen dargestellt:

$$
\begin{array}{lrcl}
\text{Explizite Form:} & g & \equiv & x_2 = bx_1 + c \\
\text{Implizite Form:} & g & \equiv & a_1x_1 + a_2x_2 + d = 0 \\
\text{Punkt-Richtungs-Form:} & g & \equiv & \vec{x} = \vec{p} + k\vec{r} \\
\text{Normalform} & g & \equiv & (\vec{x} - \vec{p})\vec{n} = 0
\end{array}
$$

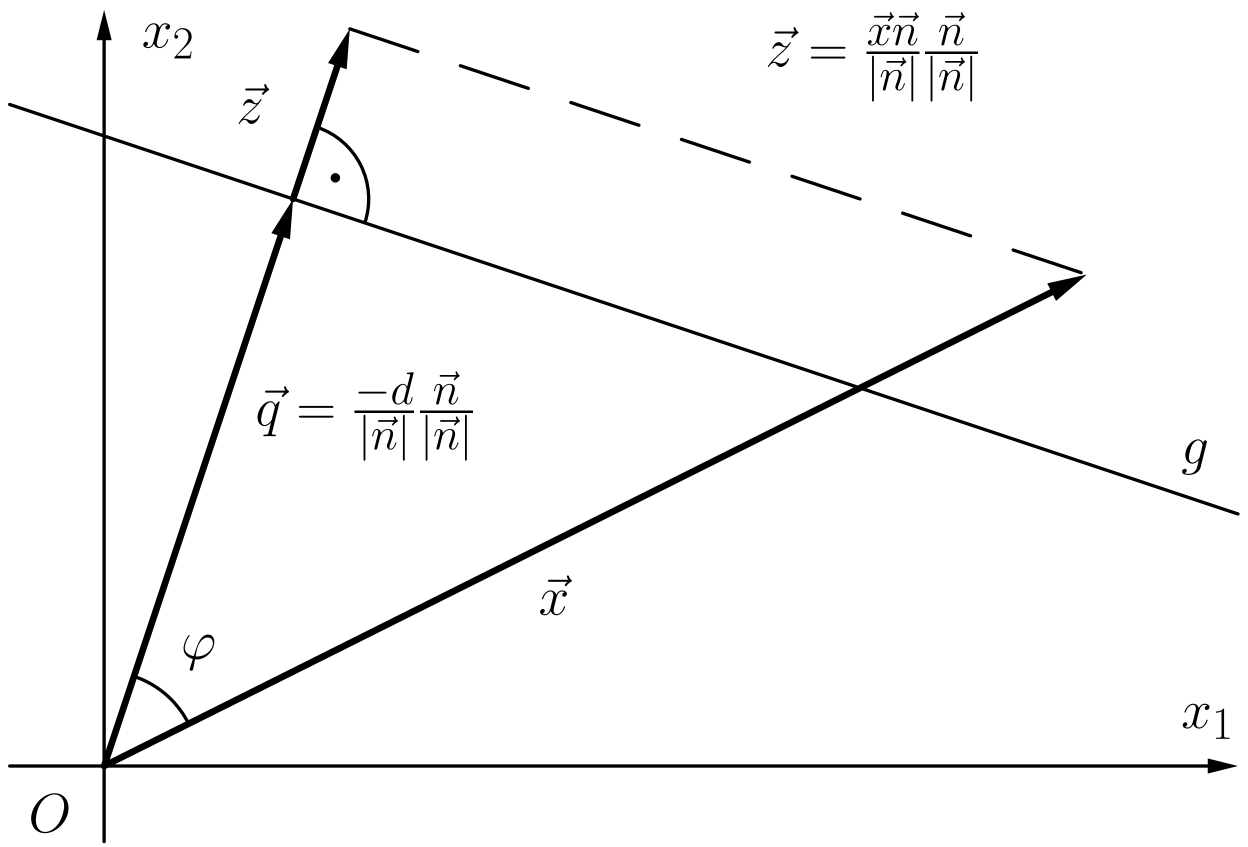
mit den Parametern

b : Anstieg der Geraden
c : Abschnitt der x_2-Achse
$\vec{p}$: Vektor zu einem Punkt auf der Gerade (Ortsvektor)
$\vec{r}$: Richtungsvektor der Gerade
$\vec{n}$: Normalenvektor der Gerade

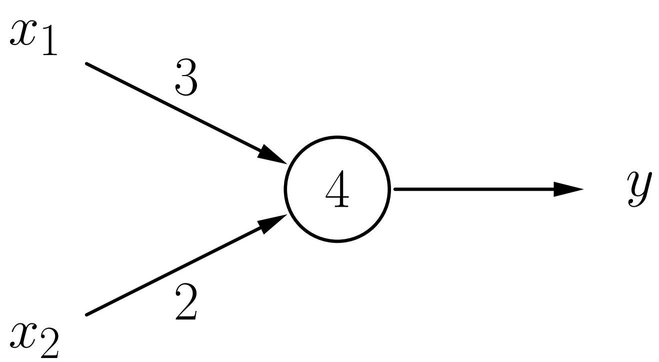
Eine Gerade und ihre definierenden Eigenschaften.

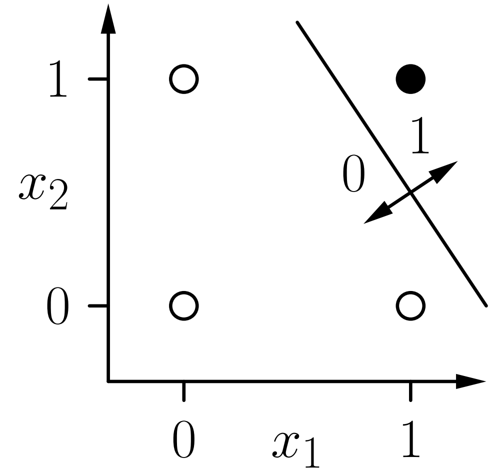


Bestimmung, auf welcher Seite ein Punkt $\vec{x}$ liegt.

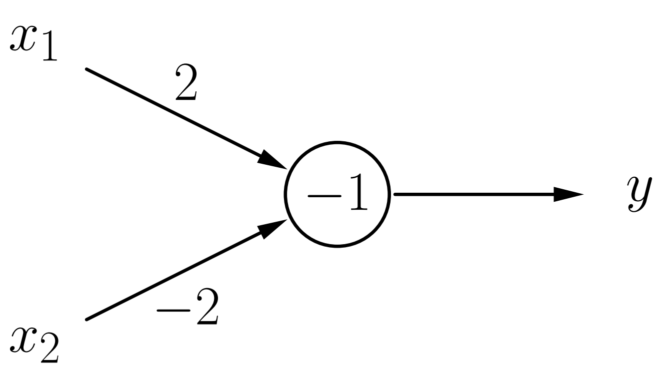


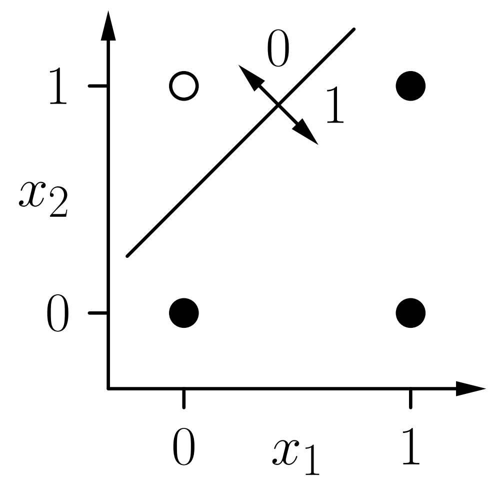
Schwellenwertelement für $x_1 \wedge x_2$.



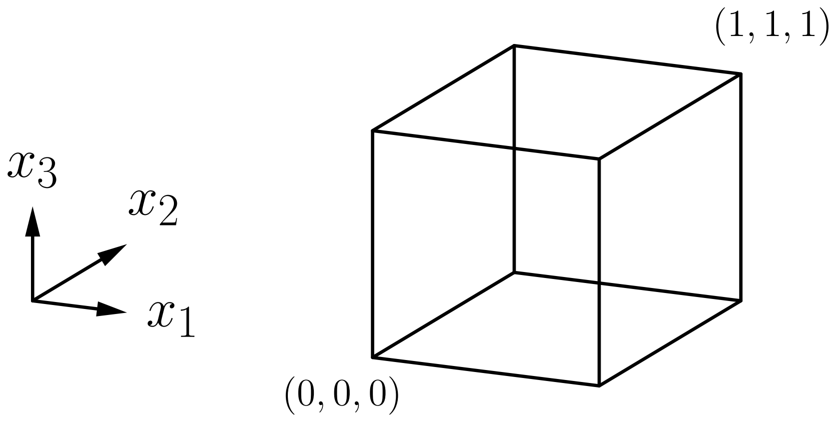


Ein Schwellenwertelement für $x_2 \rightarrow x_1$.

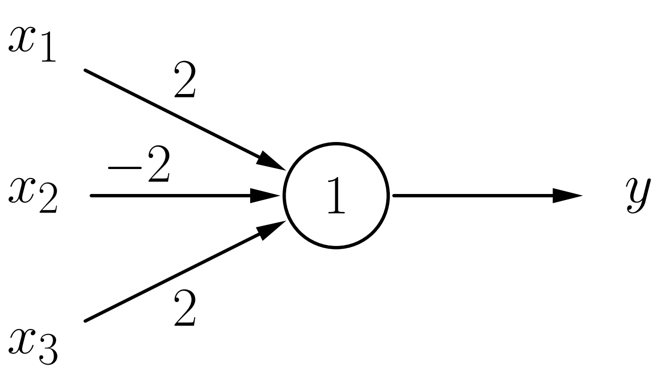


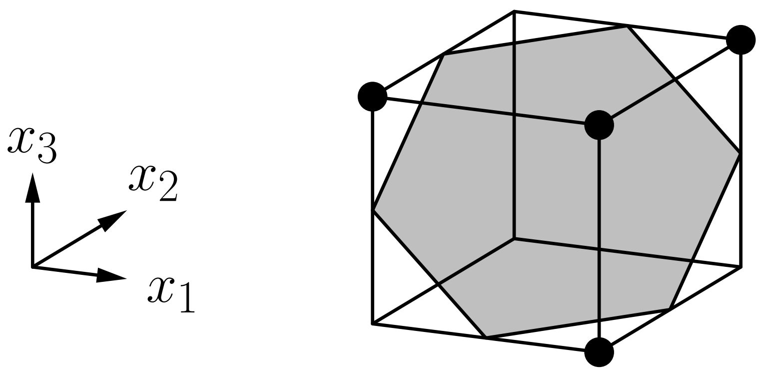


Darstellung 3-dimensionaler Boolescher Funktionen:



Schwellenwertelement für $(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$**.**



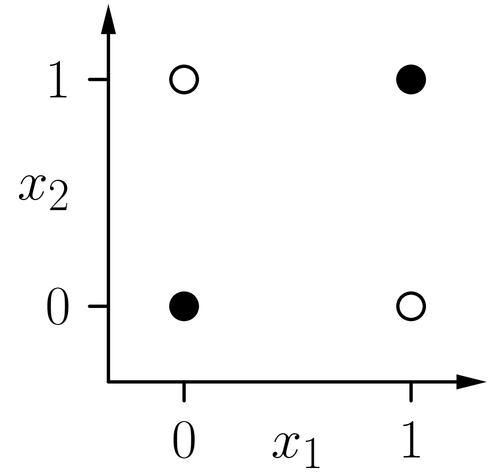


Zwei Punktmengen in einem n-dimensionalen Raum heißen linear separabel, wenn sie durch eine (n-1)-dimensionale Hyperebene getrennt werden können. Die Punkte der einen Menge dürfen dabei auch auf der Hyperebene liegen.

Eine Boolesche Funktion heißt linear separabel, falls die Menge der Urbilder von 0 und die Menge der Urbilder von 1 linear separabel sind.

Das Biimplikationsproblem $x_1 \leftrightarrow x_2$: Es gibt keine Trenngerade.

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



Formaler Beweis durch *reductio ad absurdum*:

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

(2) und (3): $w_1 + w_2 < 2\theta$. Mit (4): $2\theta > \theta$, oder $\theta > 0$. Widerspruch zu (1).

Schwellenwertelemente: Grenzen

Vergleich zwischen absoluter Anzahl und der Anzahl linear separabler Boolescher Funktionen.
([Widner 1960] zitiert in [Zell 1994])

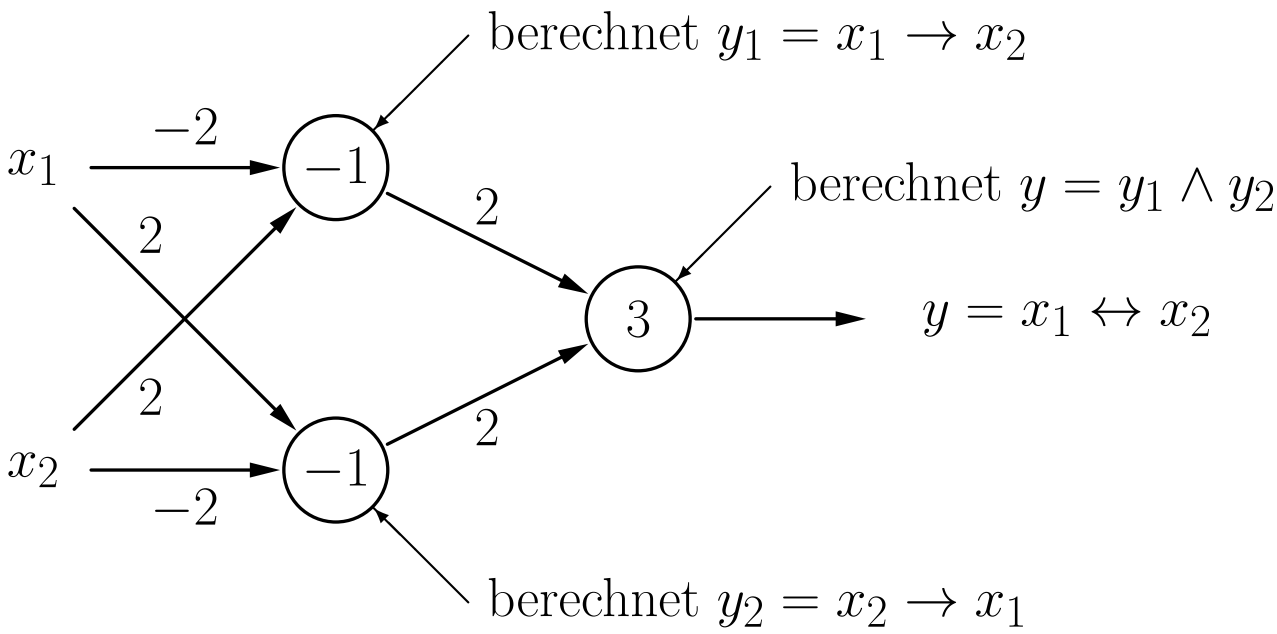
Eingaben	Boolesche Funktionen	linear separable Funktionen
1	4	4
2	16	14
3	256	104
4	65536	1774
5	$4.3 \cdot 10^{9}$	94572
6	$1.8 \cdot 10^{19}$	$5.0 \cdot 10^{6}$

Für viele Eingaben kann ein SWE fast keine Funktion berechnen.

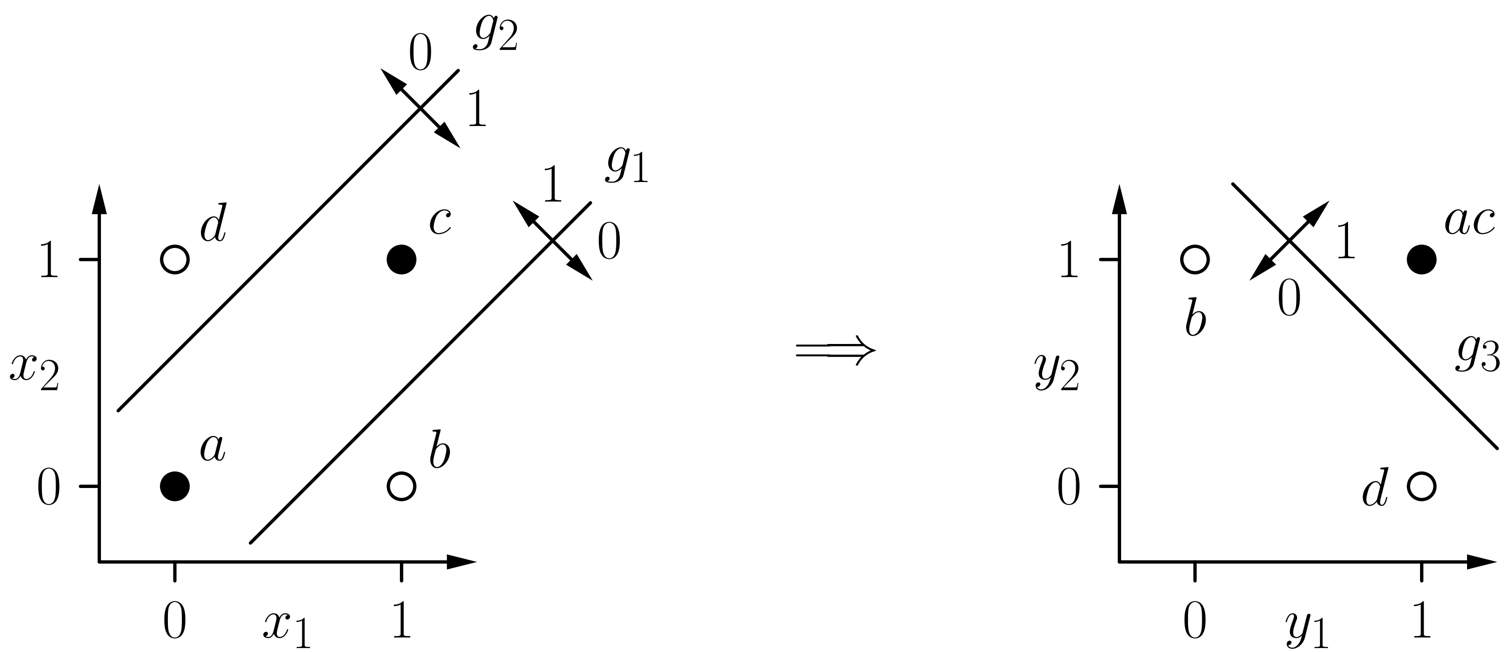
Netze aus Schwellenwertelementen sind notwendig, um die Berechnungsfähigkeiten zu erweitern.

Biimplikationsproblem, Lösung durch ein Netzwerk.

Idee: logische Zerlegung $\quad x_1 \leftrightarrow x_2 \quad \equiv \quad (x_1 \rightarrow x_2) \land (x_2 \rightarrow x_1)$



Lösung des Biimplikationsproblems: Geometrische Interpretation



Die erste Schicht berechnet neue Boolesche Koordinaten für die Punkte.

Nach der Koordinatentransformation ist das Problem linear separabel.

Darstellung beliebiger Boolescher Funktionen

Sei $y = f(x_1, \ldots, x_n)$ eine Boolesche Funktion mit n Variablen.

(i) Stelle $f(x_1, \ldots, x_n)$ in disjunktiver Normalform dar. D.h. bestimme $D_f = K_1 \vee \ldots \vee K_m$, wobei alle K_j Konjunktionen von n Literalen sind, d.h., $K_j = l_{j1} \wedge \ldots \wedge l_{jn}$ mit $l_{ji} = x_i$ (positives Literal) oder $l_{ji} = \neg x_i$ (negatives Literal).

(ii) Lege ein Neuron für jede Konjunktion K_j der disjunktiven Normalform an (mit n Eingängen — ein Eingang pro Variable), wobei

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

(iii) Lege ein Ausgabeneuron an (mit m Eingängen — ein Eingang für jedes Neuron, das in Schritt (ii) angelegt wurde), wobei

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

Trainieren von Schwellenwertelementen

Trainieren von Schwellenwertelementen

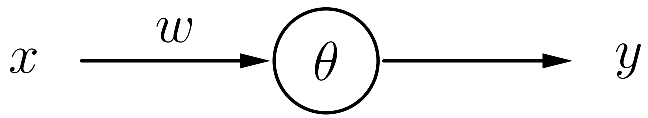
Die geometrische Interpretation bietet eine Möglichkeit, SWE mit 2 und 3 Eingängen zu konstruieren, aber:

- Es ist keine automatische Methode (Visualisierung und Begutachtung ist nötig).
- Nicht möglich für mehr als drei Eingabevariablen.

Grundlegende Idee des automatischen Trainings:

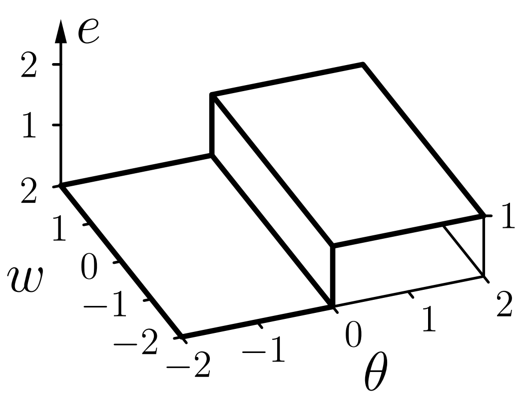
- Beginne mit zufälligen Werten für Gewichte und Schwellenwert.
- Bestimme den Ausgabefehler für eine Menge von Trainingsbeispielen.
- Der Fehler ist eine Funktion der Gewichte und des Schwellenwerts: $e = e(w_1, \ldots, w_n, \theta)$.
- Passe Gewichte und Schwellenwert so an, dass der Fehler kleiner wird.
- Wiederhole diese Anpassung, bis der Fehler verschwindet.

Schwellenwertelement mit einer Eingabe für die Negation $\neg x$.

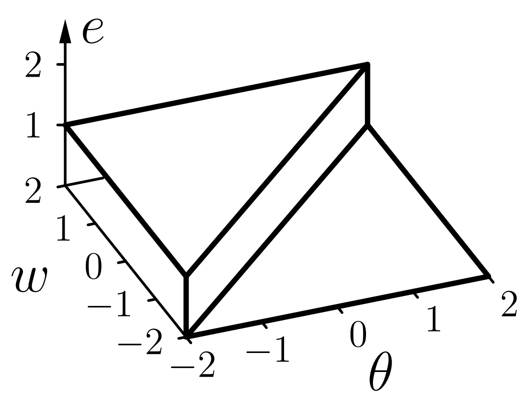


x	y
0	1
1	0

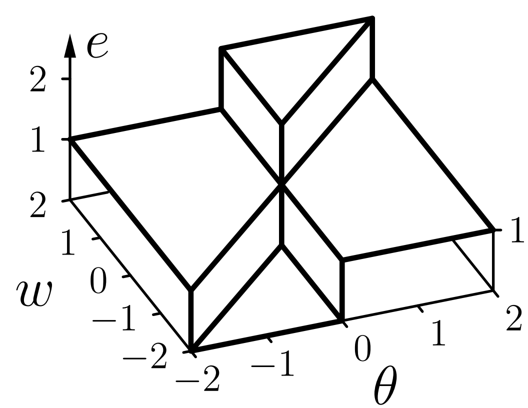
Ausgabefehler als eine Funktion von Gewicht und Schwellenwert.



Fehler für $x = 0$



Fehler für $x = 1$



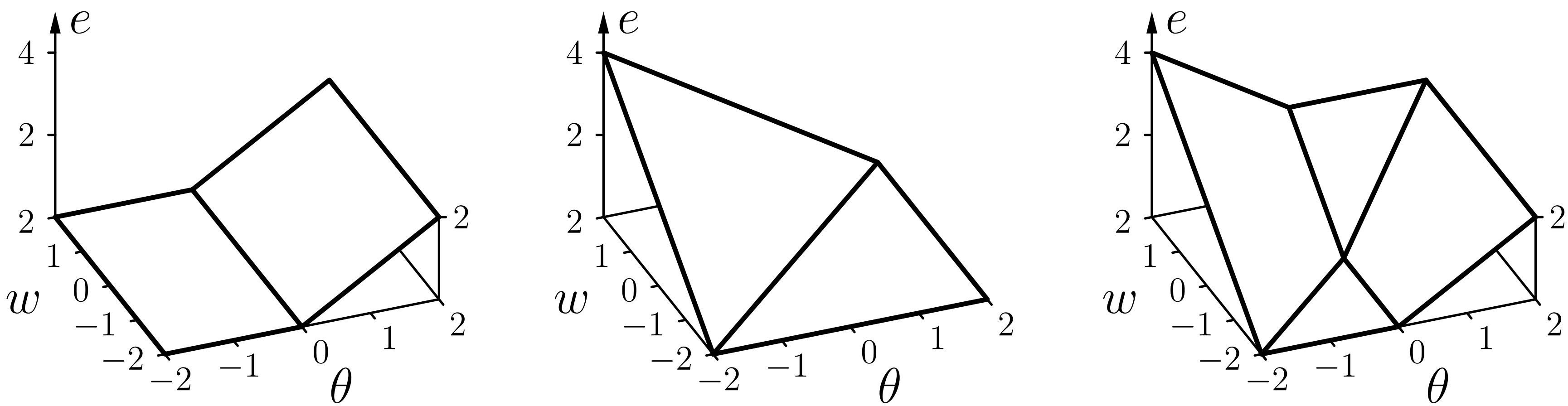
Summe der Fehler

Die Fehlerfunktion kann nicht direkt verwendet werden,
da sie aus Plateaus besteht.

Lösung: Falls die berechnete Ausgabe falsch ist, dann berücksichtige,
wie weit θ überschritten (für $x = 0$) oder unterschritten ist (für $x = 1$).

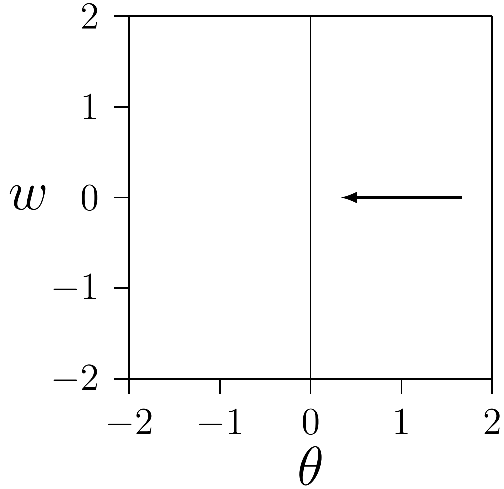
anschaulich: Berechnung ist „umso falscher“,
je weiter θ überschritten (für $x = 0$) bzw. unterschritten ist (für $x = 1$).

Modifizierter Ausgabefehler als Funktion von $\vec{w}$ und θ.

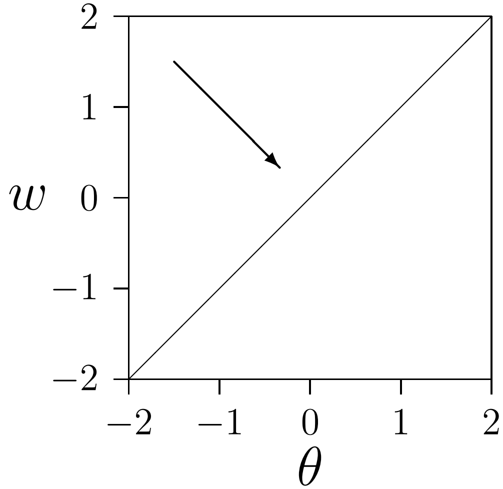


Fehler für $x = 0$ Fehler für $x = 1$ Summe der Fehler

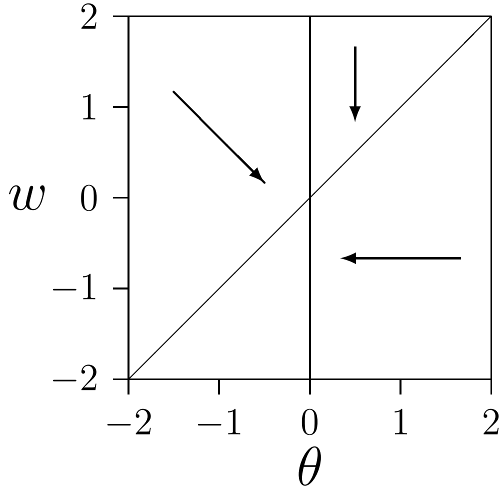
Schema der resultierenden Richtungen der Parameteränderungen.



Änderungen für $x = 0$



Änderungen für $x = 1$

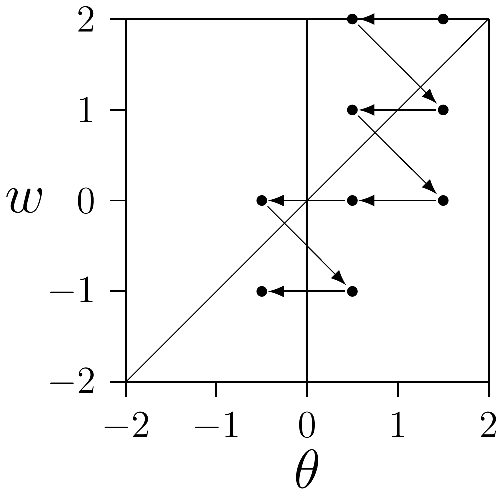


Summe der Änderungen

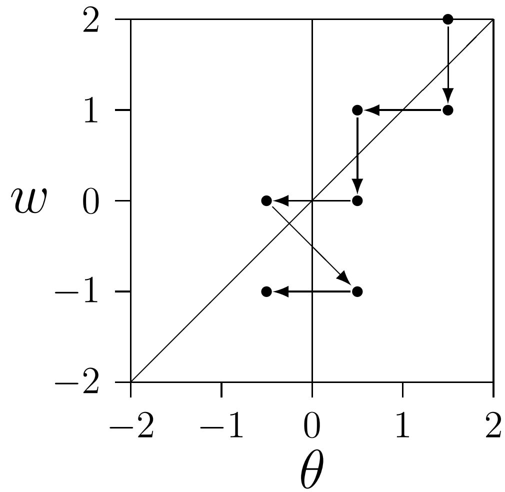
Beginne an zufälligem Punkt.

Passe Parameter iterativ an,
entsprechend der zugehörigen Richtung am aktuellen Punkt.

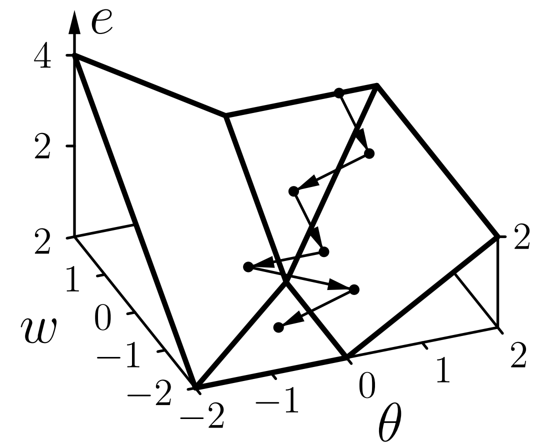
Beispieltrainingsprozedur: Online- und Batch-Training.



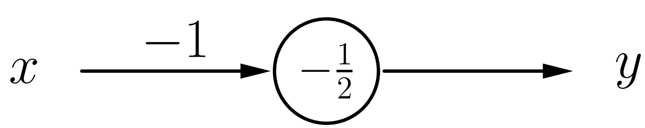
Online-Lernen

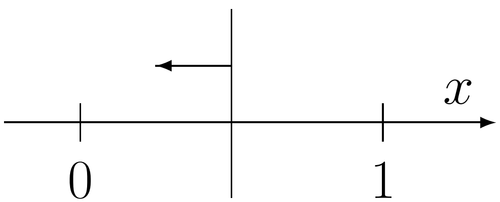


Batch-Lernen



Batch-Lernen





Trainieren von Schwellenwertelementen: Delta-Regel

Formale Trainingsregel: Sei $\vec{x} = (x_1, \ldots, x_n)$ ein Eingabevektor eines Schwellenwertelements, o die gewünschte Ausgabe für diesen Eingabevektor, und y die momentane Ausgabe des Schwellenwertelements. Wenn $y \neq o$, dann werden Schwellenwert θ und Gewichtsvektor $\vec{w} = (w_1, \ldots, w_n)$ wie folgt angepasst, um den Fehler zu reduzieren:

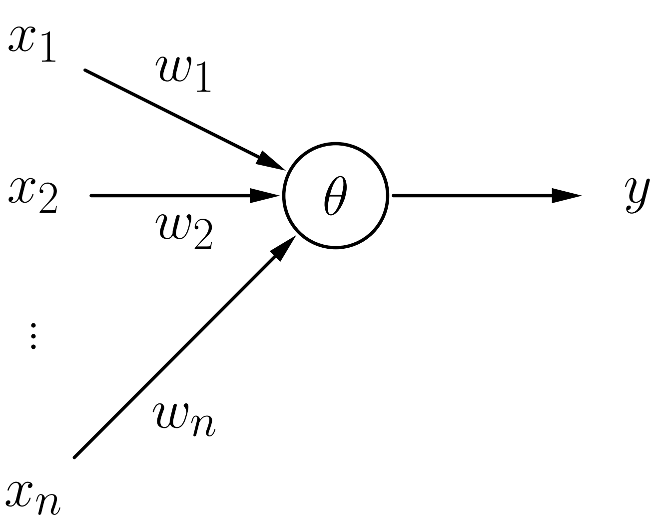
$$\begin{aligned} \theta^{(\text{neu})} &= \theta^{(\text{alt})} + \Delta\theta \quad \text{wobei} \quad \Delta\theta = -\eta(o-y), \\ \forall i \in \{1, \ldots, n\}: \quad w_i^{(\text{neu})} &= w_i^{(\text{alt})} + \Delta w_i \quad \text{wobei} \quad \Delta w_i = \eta(o-y)x_i, \end{aligned}$$

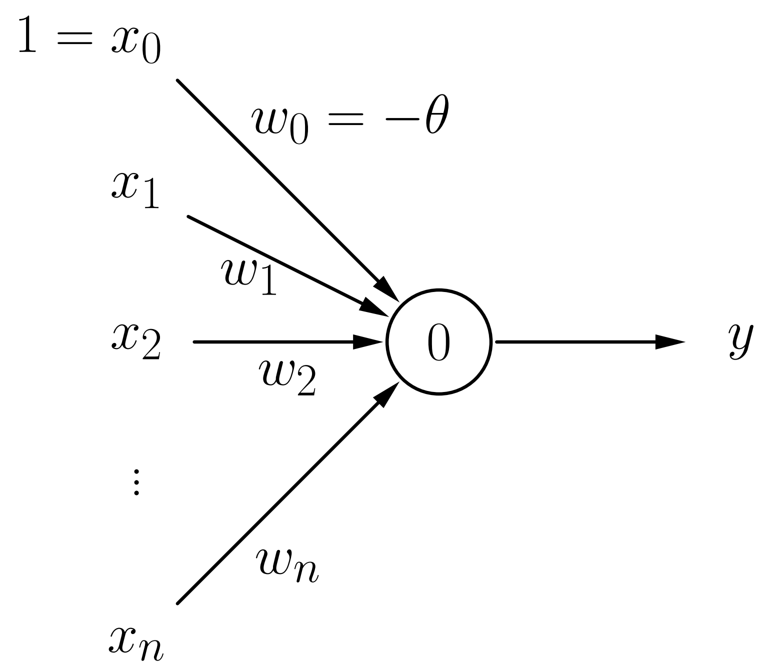
wobei η ein Parameter ist, der **Lernrate** genannt wird. Er bestimmt die Größenordnung der Gewichtsänderungen. Diese Vorgehensweise nennt sich **Delta-Regel** oder **Widrow–Hoff–Procedure** [Widrow and Hoff 1960].

Online-Training: Passe Parameter nach jedem Trainingsmuster an.

Batch-Training: Passe Parameter am Ende jeder **Epoche** an,
d.h. nach dem Durchlaufen aller Trainingsbeispiele.

Ändern des Schwellenwerts in ein Gewicht:





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

Trainieren von Schwellenwertelementen: Delta-Regel

```
procedure online_training (var w⃗, var θ, L, η);
var y, e;                                   (* Ausgabe, Fehlersumme *)
begin
  repeat
    e := 0;                                 (* initialisiere Fehlersumme *)
    for all (x⃗, o) ∈ L do begin             (* durchlaufe Trainingsmuster*)
      if (w⃗x⃗ ≥ θ) then  y := 1;              (* berechne Ausgabe*)
                 else  y := 0;              (* des Schwellenwertelements *)
      if (y ≠ o) then begin                 (* Falls Ausgabe falsch *)
        θ  := θ − η(o − y);                 (* passe Schwellenwert *)
        w⃗  := w⃗ + η(o − y)x⃗;                (* und Gewichte an *)
        e  := e + |o − y|;                  (* summiere die Fehler*)
      end;
    end;
  until (e ≤ 0);                            (* wiederhole die Berechnungen*)
end;                                        (* bis der Fehler verschwindet*)
```

Trainieren von Schwellenwertelementen: Delta-Regel

```
procedure batch_training (var w⃗, var θ, L, η);
var y, e,                                        (* Ausgabe, Fehlersumme *)
    θ_c, w⃗_c;                                    (* summierte Änderungen *)
begin
  repeat
    e := 0; θ_c := 0; w⃗_c := 0⃗;                  (* Initialisierungen *)
    for all (x⃗, o) ∈ L do begin                  (* durchlaufe Trainingsbeispiele*)
      if (w⃗x⃗ ≥ θ) then  y := 1;                  (* berechne Ausgabe *)
                  else  y := 0;                  (* des Schwellenwertelements *)
      if (y ≠ o) then begin                      (* Falls Ausgabe falsch*)
        θ_c  := θ_c  − η(o − y);                 (* summiere die Änderungen von*)
        w⃗_c := w⃗_c + η(o − y)x⃗;                  (* Schwellenwert und Gewichten *)
        e   := e  +  |o − y|;                    (* summiere Fehler*)
      end;
    end;
    θ  := θ  + θ_c;                              (* passe Schwellenwert*)
    w⃗ := w⃗ + w⃗_c;                                (* und Gewichte an *)
  until (e ≤ 0);                                 (* wiederhole Berechnungen *)
end;                                             (* bis der Fehler verschwindet*)
```

Trainieren von Schwellenwertelementen: Online

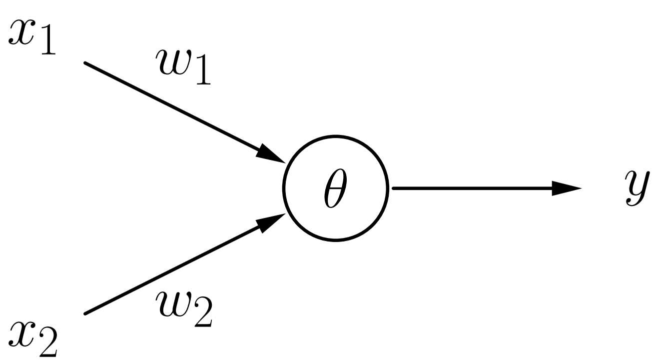
Epoche	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0	0.5	2
	1	0	1.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0	0.5	1
	1	0	0.5	1	-1	1	-1	1.5	0
3	0	1	-1.5	0	1	-1	0	0.5	0
	1	0	0.5	0	0	0	0	0.5	0
4	0	1	-0.5	0	1	-1	0	-0.5	0
	1	0	0.5	1	-1	1	-1	0.5	-1
5	0	1	-0.5	0	1	-1	0	-0.5	-1
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

Trainieren von Schwellenwertelementen: Batch

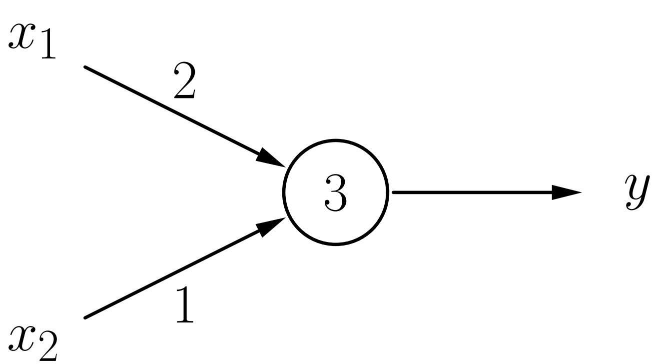
Epoche	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	0.5	1
3	0	1	-0.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	0.5	0
4	0	1	-0.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	-0.5	0
5	0	1	0.5	1	0	0	0		
	1	0	0.5	1	-1	1	-1	0.5	-1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

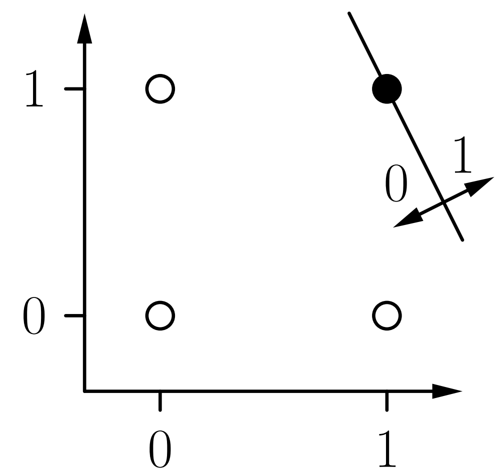
Trainieren von Schwellenwertelementen: Konjunktion

Schwellenwertelement mit zwei Eingängen für die Konjunktion.



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





Trainieren von Schwellenwertelementen: Konjunktion

Epoche	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

Trainieren von Schwellenwertelementen: Biimplikation

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

Trainieren von Schwellenwertelementen: Konvergenz

Konvergenztheorem: Sei $L = \{(\vec{x}_1, o_1), \ldots (\vec{x}_m, o_m)\}$ eine Menge von Trainingsmustern, jedes bestehend aus einem Eingabevektor $\vec{x}_i \in \mathbb{R}^n$ und einer gewünschten Ausgabe $o_i \in \{0, 1\}$. Sei weiterhin $L_0 = \{(\vec{x}, o) \in L \mid o = 0\}$ und $L_1 = \{(\vec{x}, o) \in L \mid o = 1\}$. Falls L_0 und L_1 linear separabel sind, d.h., falls $\vec{w} \in \mathbb{R}^n$ und $\theta \in \mathbb{R}$ existieren, so dass

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

dann terminieren sowohl Online- als auch Batch-Training.

Für nicht linear separable Probleme terminiert der Algorithmus nicht.

Trainieren von Netzwerken aus Schwellenwertelementen

Einzelne Schwellenwertelemente haben starke Einschränkungen:
Sie können nur linear separable Funktionen berechnen.

Netzwerke aus Schwellenwertelementen können beliebige Boolesche Funktionen berechnen.

Das Trainieren einzelner Schwellenwertelemente mit der Delta-Regel ist schnell und findet garantiert eine Lösung, falls eine existiert.

Netzwerke aus Schwellenwertelementen können nicht mit der Delta-Regel trainiert werden.