INF FAKULTÄT FÜR INFORMATIK

Intelligente Systeme

Neuronale Netze

Prof. Dr. R. Kruse **C. Braune**

{rudolf.kruse,christian.braune}@ovgu.de

Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Übersicht

1. **Einleitung**
   - Natürliche (biologische) Neuronen
   - Konventionelle Rechner vs. Gehirn
   - Künstliche Neuronale Netze

2. **Schwellenwert-Elemente**

3. **Mehrschichtige Perzeptrons**

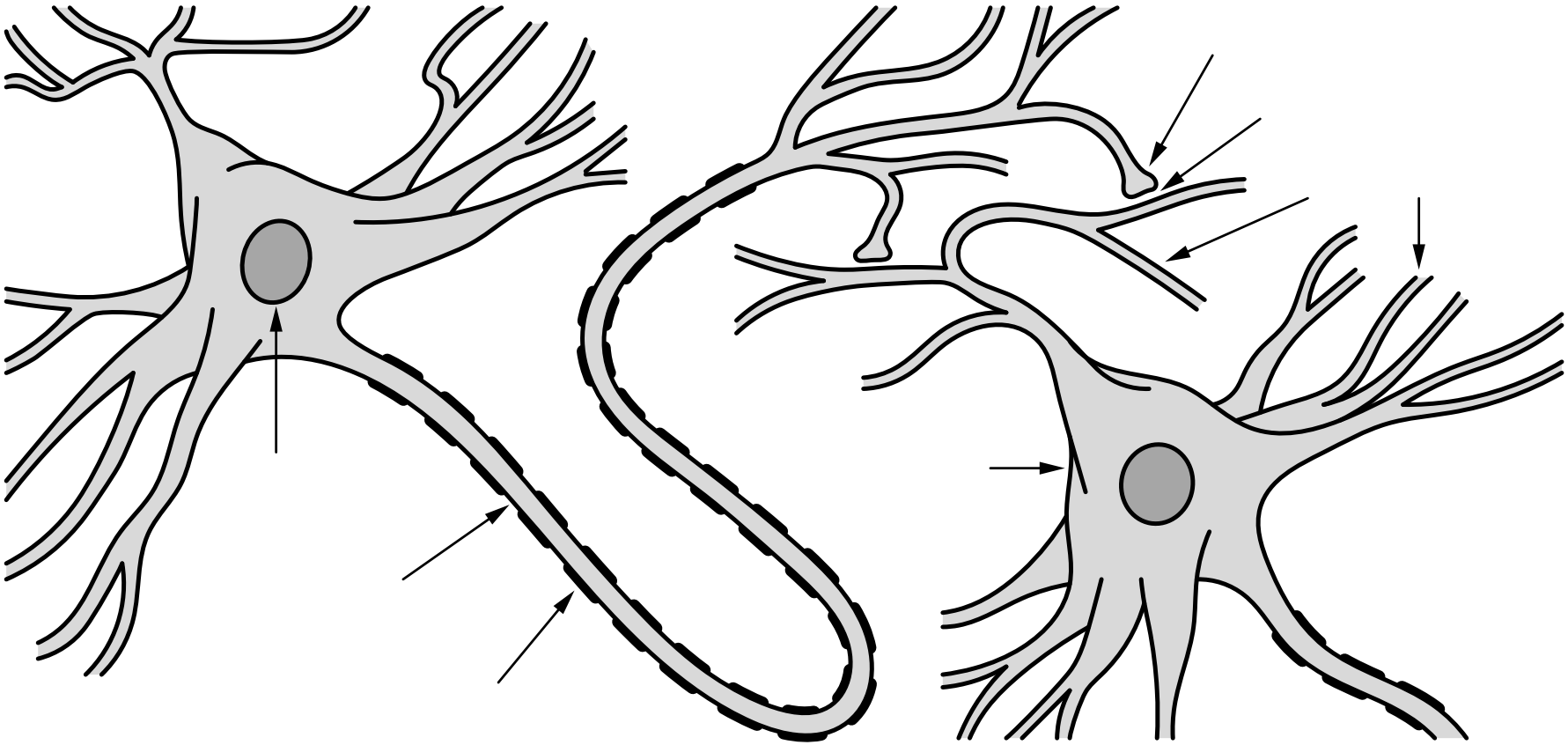
Neuronale Netze

Bisher „Intelligente Systeme von oben“:
Modellierung eines intelligenten Agenten durch algorithmische Realisierung bestimmter Aspekte rationalen Handeln

Jetzt „Intelligente Systeme von unten“:
Grobe Nachbildung der Struktur und der Verarbeitungsmechanismen des Gehirns

Viele Prozessoren (Neuronen) und Verbindungen (Synapsen), die parallel und lokal Informationen verarbeiten

Natürliche (biologische) Neuronen



Konventionelle Rechner vs. Gehirn

	Computer	Gehirn
Verarbeitungs-einheiten	CPU (SPARC M7), 10^{10} Transistoren nVidia Tesla P100 (GPU, 2016) $1.5 \cdot 10^{10}$ Transistoren	10^{11} Neuronen
Speicherkapazität	$32 \cdot 10^9$ Bytes RAM, 10^{13} Bytes Festspeicher	10^{11} Neuronen, 10^{15} Synapsen
Verarbeitungs-geschwindigkeit	10^{-8} s	120m/s
Bandbreite	10^{12} bits/s	10^{14} bits/s
Neuronale Updates pro Sekunde	10^6	10^{14}

IBM entwickelte 2014 TrueNorth Prozessoren: $5.4 \cdot 10^9$ Transistoren simulieren 10^6 Neuronen und $256 \cdot 10^6$ Synapsen.

Konventionelle Rechner vs. Gehirn

Beachte: Hirnschaltzeit von 10^{-3} s recht langsam

Aber: Updates erfolgen parallel

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
- exzellente Energieeffizienz

Daher sinnvoll, Vorteile natürlicher NN künstlich nachzuahmen, z.B. im EU Human Brain Project: 1 Milliarde EUR Fördergelder im Zeitraum 2013-2023 für „brain inspired computing“

Übersicht

1. Einleitung

2. **Schwellenwert-Elemente**
   - Perzeptron
   - Geometrische Interpretation
   - Biimplikationsproblem
   - Lernverfahren

3. Mehrschichtige Perzeptrons

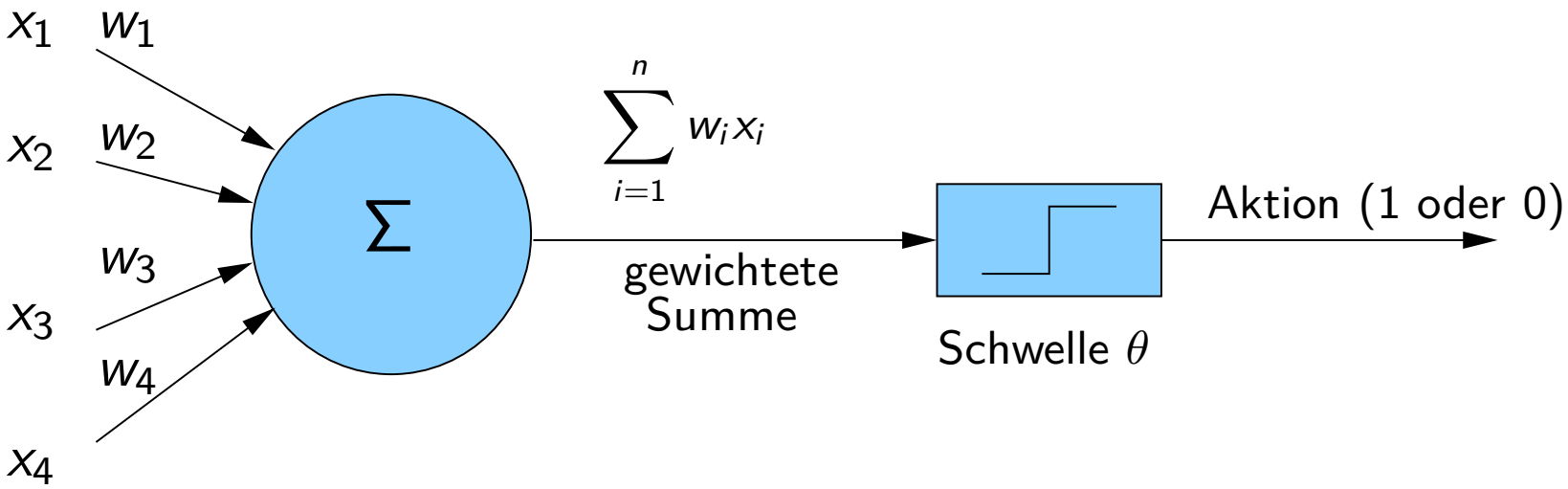
Perzeptron

Implementierung eines S-R-Agenten: verschiedene Möglichkeiten

S-R-Agent repräsentiert eine Funktion

Im Folgenden werden Perzeptren auch engl. *threshold logic unit (TLU)* genannt

Ein Perzeptron



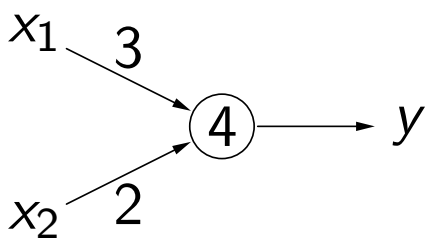
Formale Definition:

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

$$\boldsymbol{x} \stackrel{\text{def}}{=} (x_1, \ldots, x_n)$$

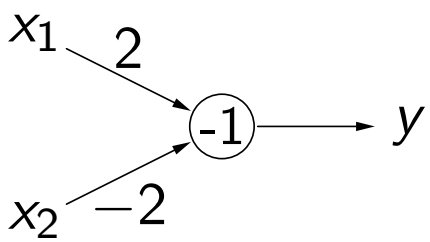
Schwellenwertelemente: Beispiele

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

Perzeptron: Vereinfachte Schreibweise

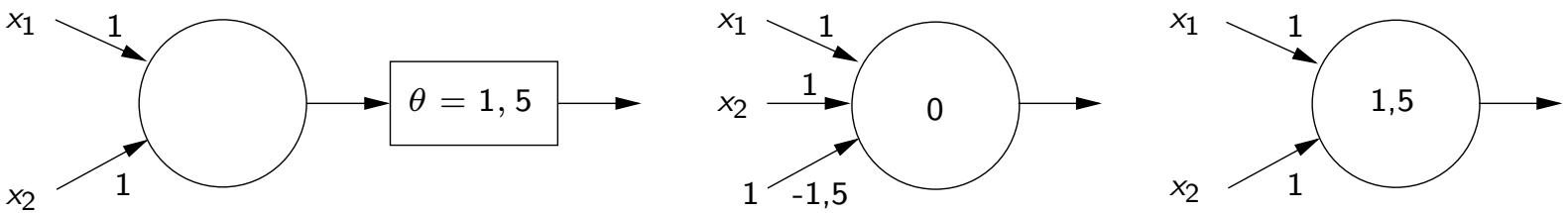
Seien $x_{n+1} \stackrel{\text{def}}{=} 1$ und $w_{n+1} \stackrel{\text{def}}{=} -\theta$

So sind $\boldsymbol{x} = (x_1, \ldots, x_n, 1)$ und $\boldsymbol{w} = (w_1, \ldots, w_n, -\theta)$

Vereinfachte Schreibweise:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{falls } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0, \\ 0 & \text{sonst.} \end{cases}$$

Beispiele äquivalenter Darstellungen:



Geometrische Interpretation

Gewichtsvektor $\boldsymbol{w} = (w_1, \ldots, w_n)$

Schwellenwert θ

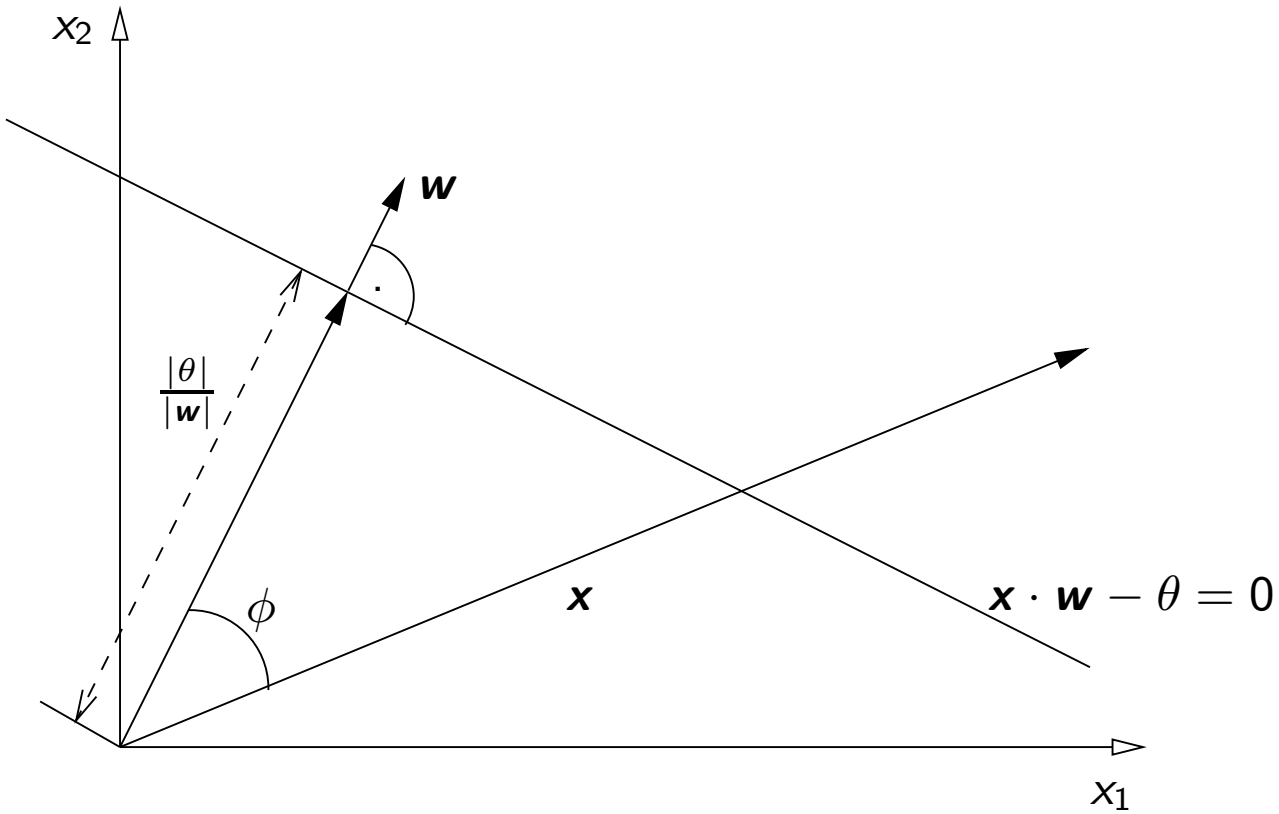
Eingabevektor $\boldsymbol{x} = (x_1, \ldots, x_n)$

Ausgabewert

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

Trennende Hyperebene $\boldsymbol{w} \cdot \boldsymbol{x} - \theta = 0$

Geometrische Interpretation



θ ist negativ, falls Ursprung auf Ebenenseite, in die $\boldsymbol{w}$ zeigt

Ansonsten ist θ positiv

$\boldsymbol{w} \cdot \boldsymbol{x} - \theta > 0$, falls x auf Ebenenseite, in deren Richtung $\boldsymbol{w}$ zeigt

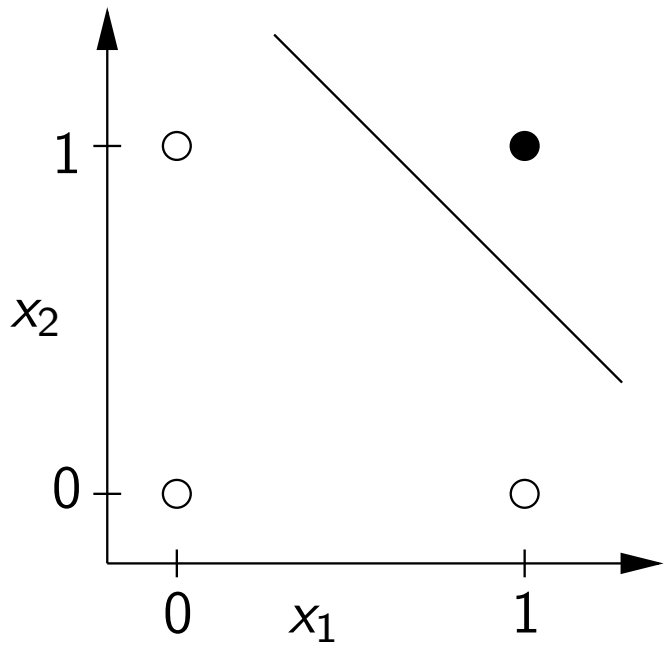
Geometrische Interpretation

Perzeptron repräsentiert eine linear separable Funktion

Logisches `AND` ist linear separabel

Somit durch Perzeptron darstellbar

x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1

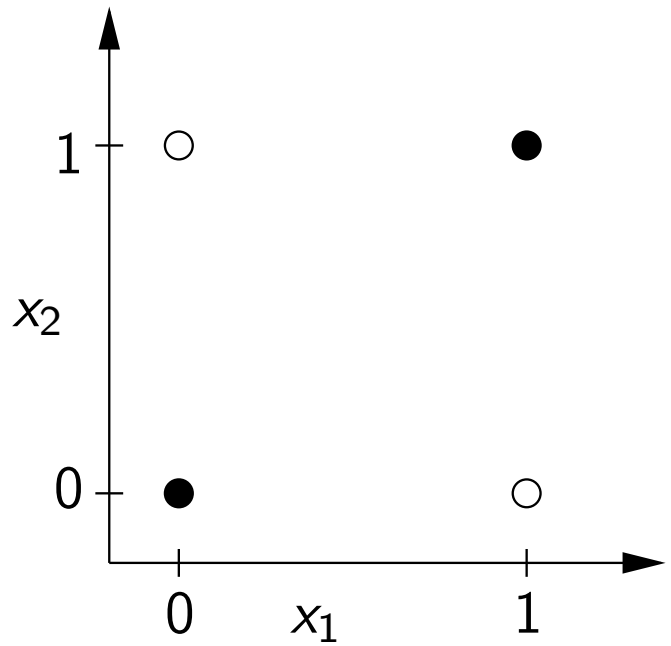


Biimplikationsproblem

Biimplikation (Äquivalenz) ist nicht linear separabel

Also existiert kein Perzeptron dafür

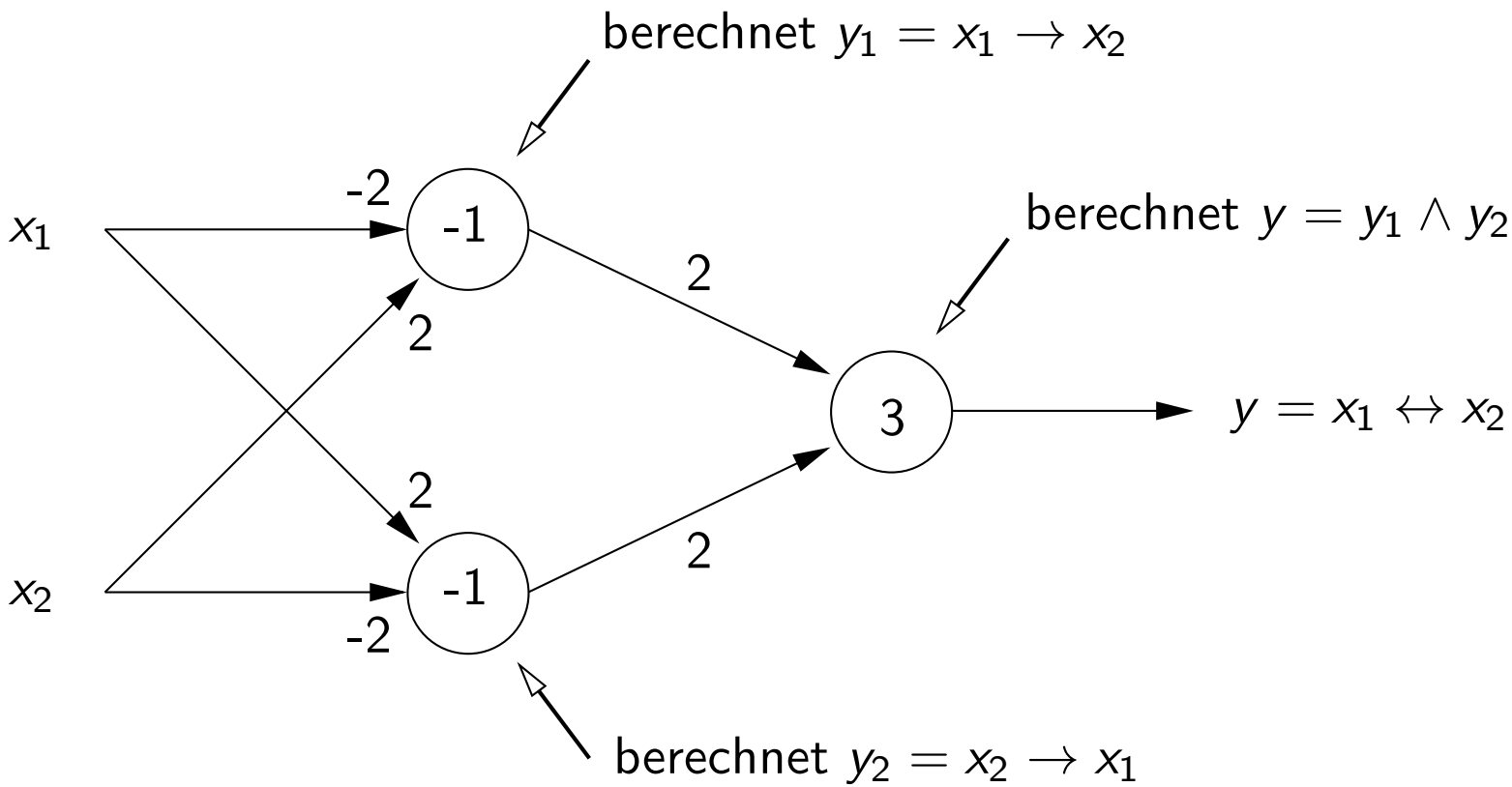
x_1	x_2	$x_1 \leftrightarrow x_2$
0	0	1
1	0	0
0	1	0
1	1	1



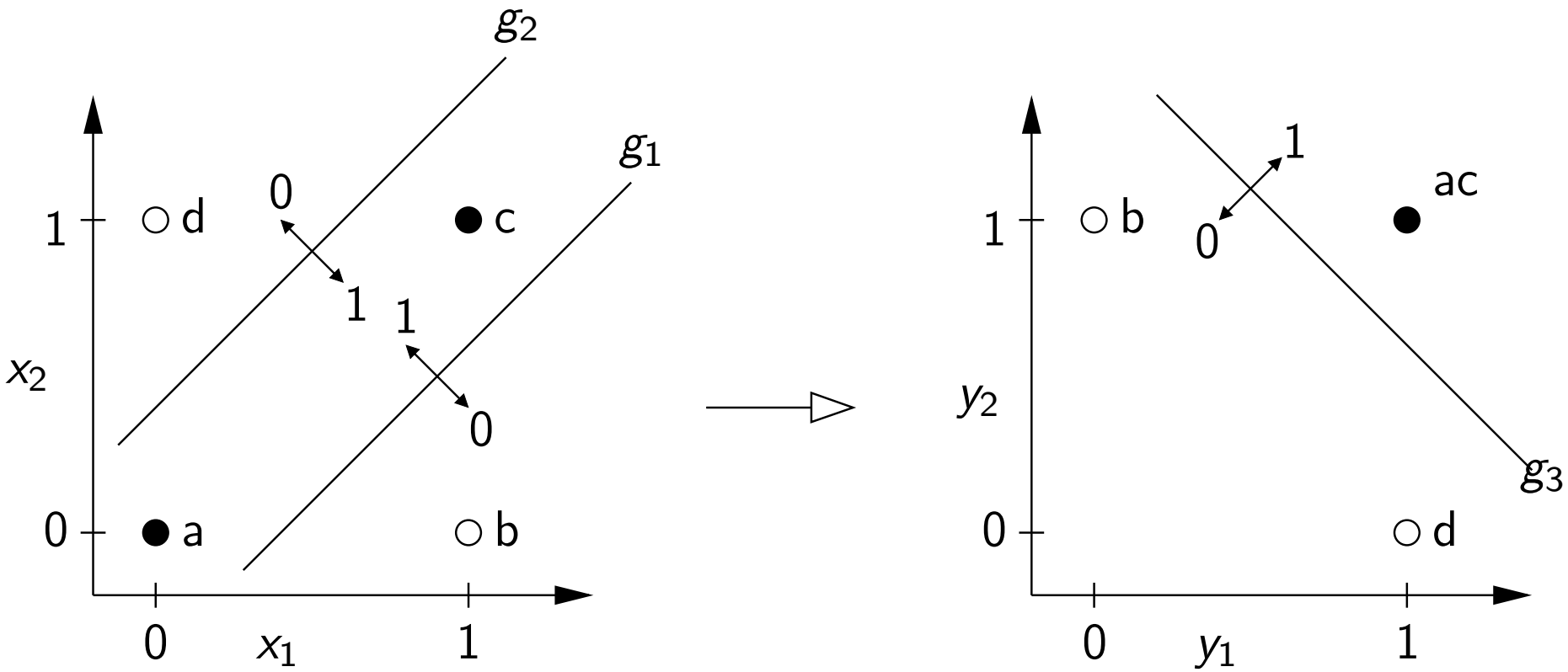
Es gibt keine Trenngerade, die den Lösungsraum wie gewünscht aufteilt

Biimplikationsproblem

Lösung: Zusammenschalten mehrerer Schwellenwertelemente



Biimplikationsproblem



Geometrische Deutung des Zusammenschaltens mehrerer Schwellenwertelemente zur Berechnung der Biimplikation

Trainieren von Schwellenwertelementen

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

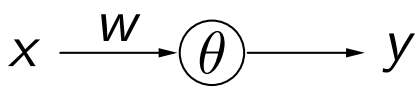
- Es ist keine automatische Methode (Visualisierung und Begutachtung ist nötig).
- Nicht möglich für mehr als drei Eingabevariablen.

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.

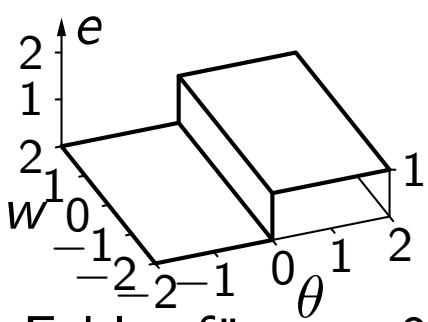
Trainieren von Schwellenwertelementen

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

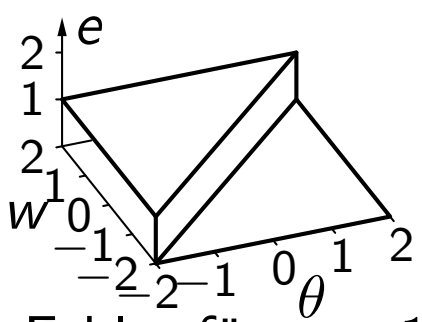


x	y
0	1
1	0

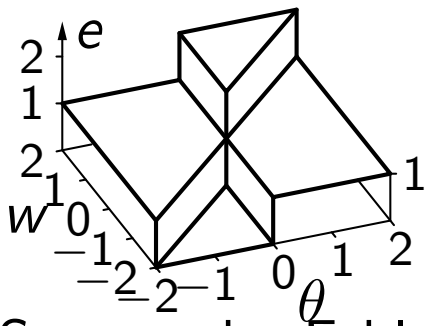
Ausgabefehler als eine Funktion von Gewicht und Schwellenwert.



Fehler für $x = 0$



Fehler für $x = 1$



Summe der Fehler

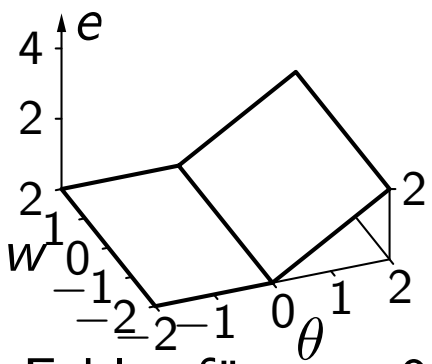
Trainieren von Schwellenwertelementen

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

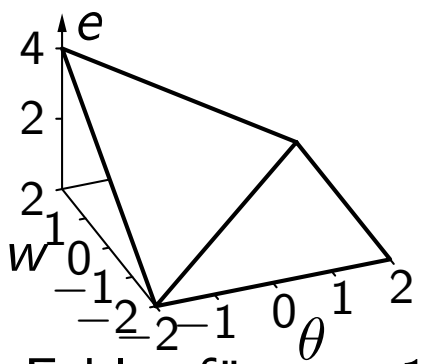
Lösung: Falls die berechnete Ausgabe falsch ist, dann berücksichtige, wie weit θ überschritten (für $x = 0$) oder unterschritten ist (für $x = 1$).

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

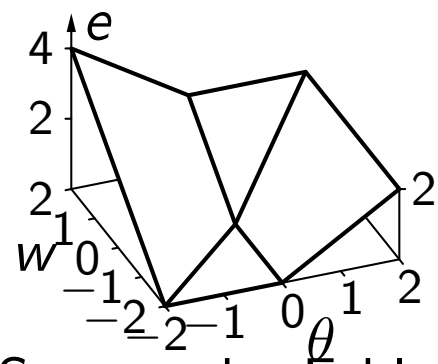
Modifizierter Ausgabefehler als Funktion von w und θ.



Fehler für $x = 0$



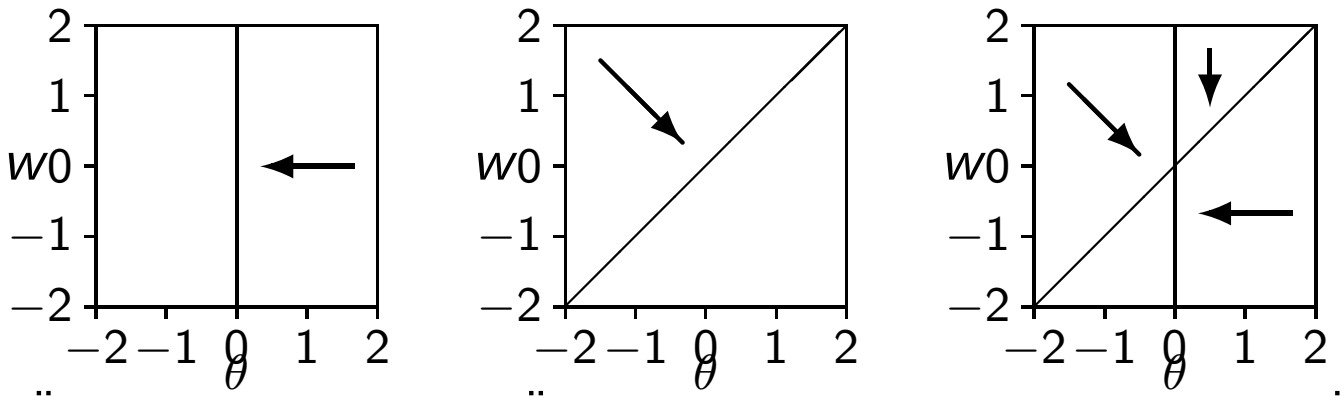
Fehler für $x = 1$



Summe der Fehler

Trainieren von Schwellenwertelementen

Schema der resultierenden Richtungen der Parameteränderungen.



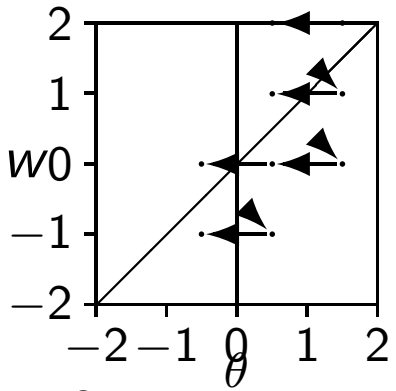
Änderungen für $x = 0$ Änderungen für $x = 1$ Summe der Änderungen

Beginne an zufälligem Punkt.

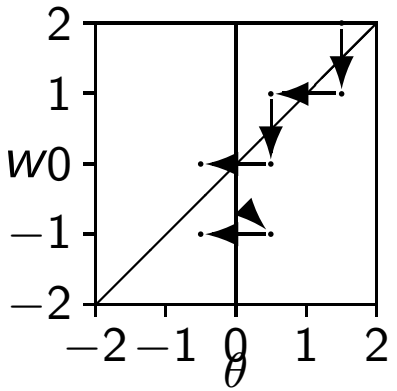
Passe Parameter iterativ an,
entsprechend der zugehörigen Richtung am aktuellen Punkt.

Trainieren von Schwellenwertelementen

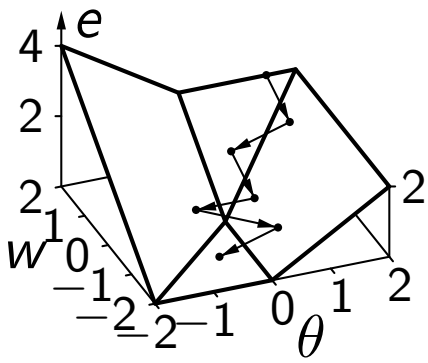
Beispieltrainingsprozedur: Online- und Batch-Training.



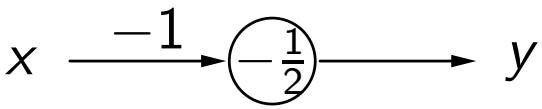
Online-Lernen

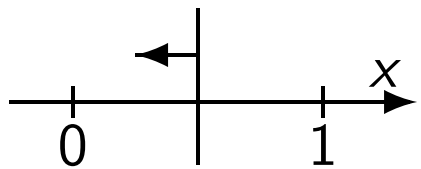


Batch-Lernen



Batch-Lernen





Trainieren von Schwellenwertelementen: Δ-Regel

Formale Trainingsregel: Sei $\boldsymbol{x} = (x_1, \ldots, x_n)$ 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 $\boldsymbol{w} = (w_1, \ldots, w_n)$ wie folgt angepasst, um den Fehler zu reduzieren:

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

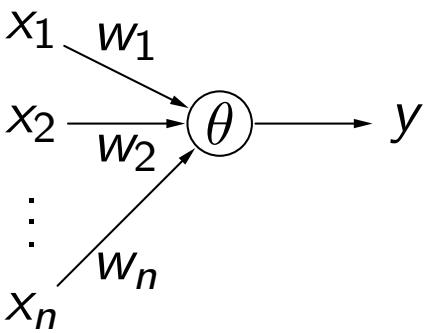
wobei η ein Parameter ist, der **Lernrate** genannt wird. Er bestimmt die Größenordnung der Gewichtsänderungen. Diese Vorgehensweise nennt sich **Delta-Regel** oder **Widrow–Hoff–Procedure**

Online-Training: Passe Parameter nach jedem Trainingsmuster an.

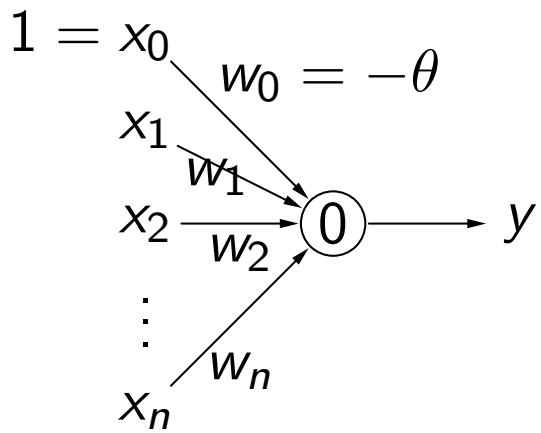
Batch-Training: Passe Parameter am Ende jeder **Epoche** an, d.h. nach dem Durchlaufen aller Trainingsbeispiele.

Trainieren von Schwellenwertelementen: Delta-Regel

Ändern des Schwellenwerts in ein Gewicht:



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



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

Trainieren von Schwellenwertelementen: Δ-Regel

```
procedure online_training (var w, var θ, L, η); var y, e;
begin                                          (* vars: Ausgabe, Fehlersumme *)
  repeat
    e := 0;                                    (* initialisiere Fehlersumme *)
    for all (x, o) ∈ L do begin                (* durchlaufe Trainingsmuster*)
      if (wx ≥ 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 *)
        w := w + η(o − y)x;                    (* 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: Δ-Regel

```
procedure batch_training (var w, var θ, L, η); var y, e, θ_c, w_c;
begin                                          (* vars: Ausgabe, Fehlersumme *)
  repeat                                       (* summierte Änderungen *)
    e := 0; θ_c := 0; w_c := 0;                (* Initialisierungen *)
    for all (x, o) ∈ L do begin                (* durchlaufe Trainingsbeispiele*)
      if (wx ≥ 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*)
        w_c := w_c + η(o - y)x;                (* Schwellenwert und Gewichten *)
        e   := e   +  |o - y|;                 (* summiere Fehler*)
      end; end;
    θ := θ + θ_c;                              (* passe Schwellenwert*)
    w := w + w_c;                              (* und Gewichte an *)
  until (e ≤ 0);                               (* wiederhole Berechnungen *)
end;                                           (* bis der Fehler verschwindet*)
```

Trainieren von Schwellenwertelementen: Online

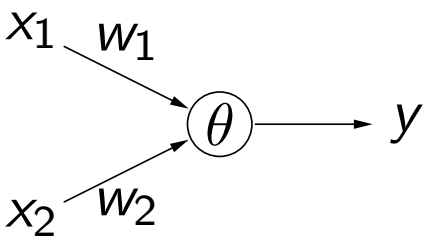
Epoche	x	o	$\boldsymbol{xw}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	−1.5	0	1	−1	0	0.5	2
	1	0	1.5	1	−1	1	−1	1.5	1
2	0	1	−1.5	0	1	−1	0	0.5	1
	1	0	0.5	1	−1	1	−1	1.5	0
3	0	1	−1.5	0	1	−1	0	0.5	0
	1	0	0.5	0	0	0	0	0.5	0
4	0	1	−0.5	0	1	−1	0	−0.5	0
	1	0	0.5	1	−1	1	−1	0.5	−1
5	0	1	−0.5	0	1	−1	0	−0.5	−1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

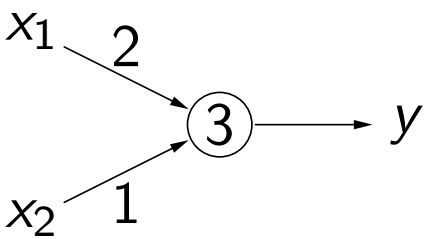
Epoche	x	o	$\boldsymbol{xw}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	0.5	1
3	0	1	-0.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	0.5	0
4	0	1	-0.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	-0.5	0
5	0	1	0.5	1	0	0	0		
	1	0	0.5	1	-1	1	-1	0.5	-1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

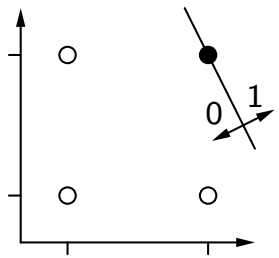
Trainieren von Schwellenwertelementen: Konjunktion

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: Konj.

Epoche	x_1	x_2	o	$\boldsymbol{xw}$	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	$\boldsymbol{xw}$	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 $L = \{(\boldsymbol{x}_1, o_1), \ldots (\boldsymbol{x}_m, o_m)\}$ eine Menge von Trainingsmustern, jedes bestehend aus einem Eingabevektor $\boldsymbol{x}_i \in \mathbb{R}^n$ und einer gewünschten Ausgabe $o_i \in \{0, 1\}$. Sei weiterhin $L_0 = \{(\boldsymbol{x}, o) \in L \mid o = 0\}$ und $L_1 = \{(\boldsymbol{x}, o) \in L \mid o = 1\}$. Falls L_0 und L_1 linear separabel sind, d.h., falls $\boldsymbol{w} \in \mathbb{R}^n$ und $\theta \in \mathbb{R}$ existieren, so dass

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

dann terminieren sowohl Online- als auch Batch-Training.

Für nicht linear separable Probleme terminiert der Algorithmus nicht.

Lernverfahren für Perzeptrons

Problemstellung:

Gegeben: Lernstichprobe $L = \{(\boldsymbol{x}_1, d_1), \ldots, (\boldsymbol{x}_N, d_N)\}$

- Merkmale $\boldsymbol{x}_i \in \{0,1\}^n$ mit zugehörigen Aktionen d_i, $1 \leq i \leq N$
- L auch Trainingsmenge genannt
- L gegeben durch Lehrer/Orakel
- Daher: überwachtes Lernen, engl. *supervised learning*

Gesucht: Funktion $f : \{0,1\}^n \to \{0,1\}$, die zu L „passt“

Trainieren einzelner Perzeptrons

Im Training: $\boldsymbol{w}$ und θ werden angepasst

Ziel: Für Elemente $\boldsymbol{x}_i$ aus L stimmen d_i und die Perzeptron-Ausgabe

$$f_i = f_{(w_1,\ldots,w_n,\theta)}(\boldsymbol{x}_i)$$

möglichst gut überein

Lösung: Minimierung des quadratischen Fehlers

Also, Lernen = Optimierungsaufgabe

$$\varepsilon = \sum_{i=1}^{N}(d_i - f_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 = \sum_{i=1}^{N}(d_i - f_i)^2 \stackrel{!}{=} min$$

Delta-Regel – Anmerkungen

Berechnung der Δ-Werte lässt sich auch schreiben als

$$\eta \cdot (d - f) \cdot x_i$$

Bzw. in Vektorschreibweise

$$\Delta_{(\boldsymbol{x},d)} \boldsymbol{w} = \begin{cases} 0 & \text{falls } d = f \\ \eta \cdot (d - f) \cdot \boldsymbol{x} & \text{sonst.} \end{cases}$$

Oft Abbruchbedingung: $\varepsilon <$ vorgegebene Schranke

Ein Durchgang heißt *Lernperiode*

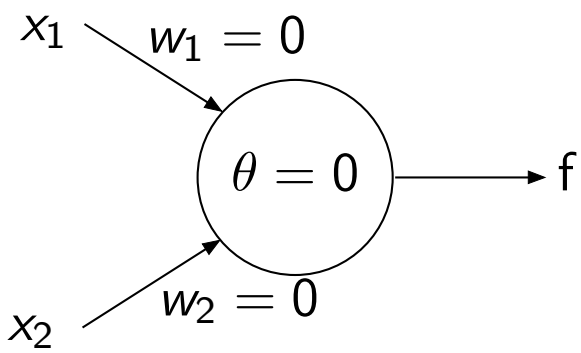
Algorithmus terminiert für linear separable Lernstichproben

Delta-Regel – Beispiel AND

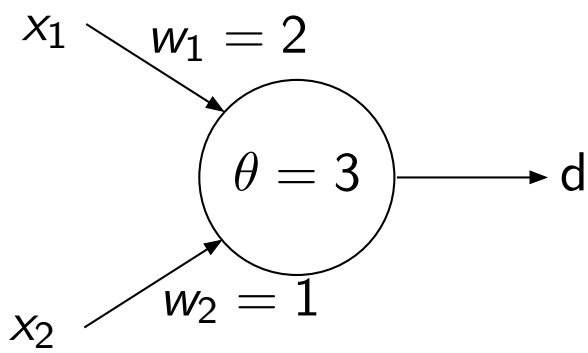
Lernvorgang liefert

x_1	x_2	d
0	0	0
1	0	0
0	1	0
1	1	1

Initialisierung



Nach dem Lernen



Rechnung → Übung, Hinweis: Tabelle mit folgenden Werten

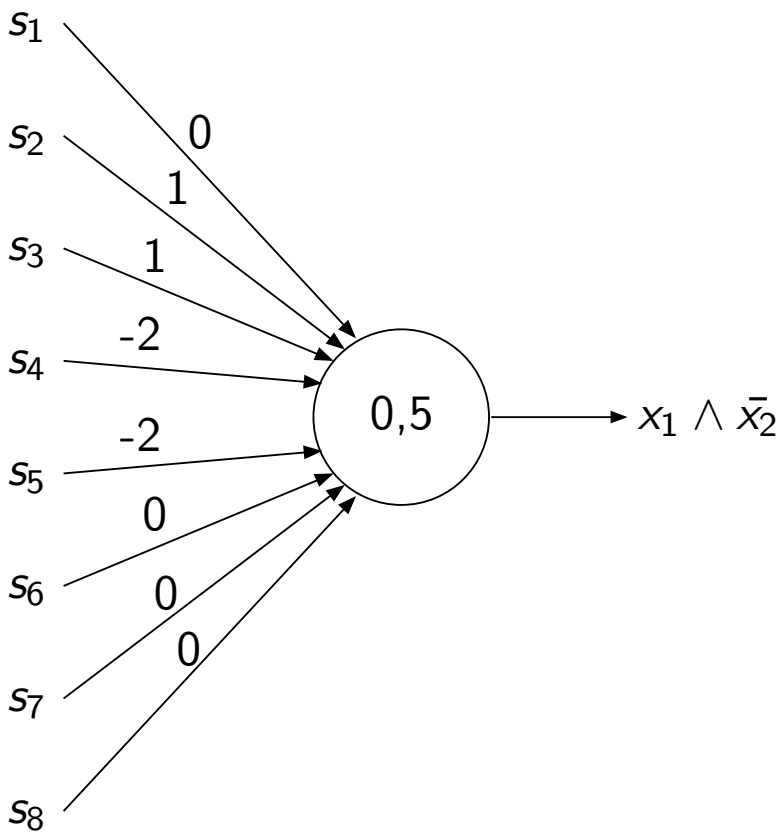
Epoche	x_1	x_2	d	$\boldsymbol{xw}$	f	ε	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2

Eine Epoche = ein Durchlauf aller Lernbeispiele

Perzeptrons in der *Grid World*

$$\texttt{move east} \leftarrow x_1 = 1 \wedge x_2 = 0 \leftarrow x_1 \wedge \overline{x_2} = (s_2 \vee s_3) \wedge \overline{s_4} \wedge \overline{s_5}$$

Folgendes Perzeptron liefert 1 $\Longleftrightarrow$ `move east`

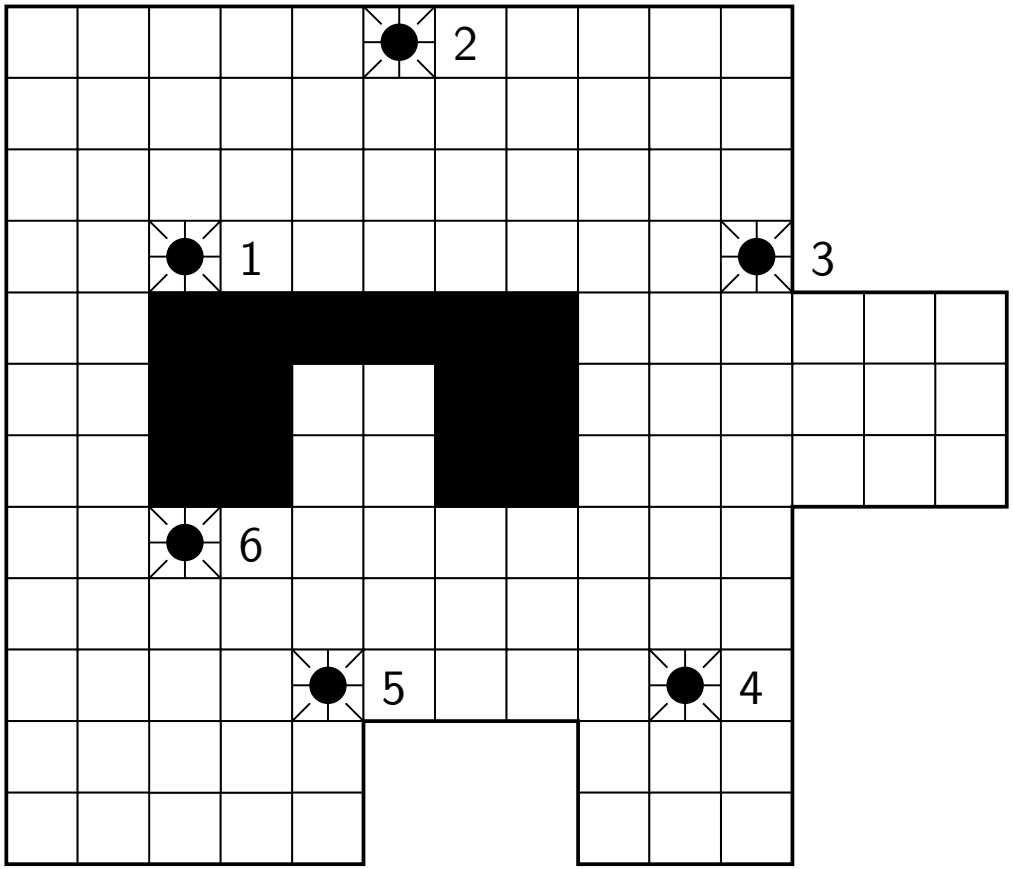


Lerndatensatz – „nach Osten gehen“

Möglich: Gewichte von Beispielen lernen

hier: nicht lohnenswert, da Funktion sehr einfach

Aufgabe: suche $\boldsymbol{x}$ für die 6 markierten Positionen



Lerndatensatz – „nach Osten gehen“

Hier L sieht wie folgt aus

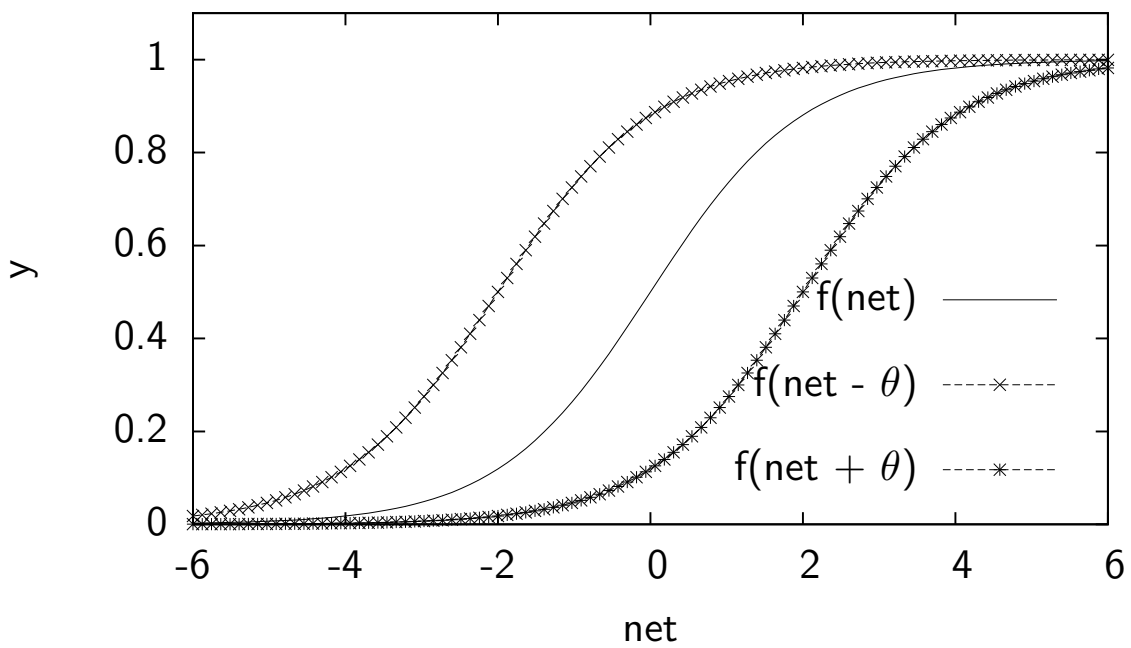
Nr.	Sensordaten	$x_1 \wedge \overline{x_2}$ (`move east`)
1	$(0, 0, 0, 0, 1, 1, 0, 0)$	0
2	$(1, 1, 1, 0, 0, 0, 0, 0)$	1
3	$(0, 0, 1, 0, 0, 0, 0, 0)$	0
4	$(0, 0, 0, 0, 0, 0, 0, 0)$	0
5	$(0, 0, 0, 0, 0, 1, 0, 0)$	0
6	$(0, 1, 1, 0, 0, 0, 0, 0)$	1

Verallgemeinerte Delta-Regel

Idee: ersetze f durch s-förmige (sigmoide), differenzierbare Funktion wie z.B. logistische Funktion

$$f(s) = \frac{1}{1 + e^{-s}} \quad \text{und somit} \quad \frac{\partial f}{\partial s} = f(1 - f)$$

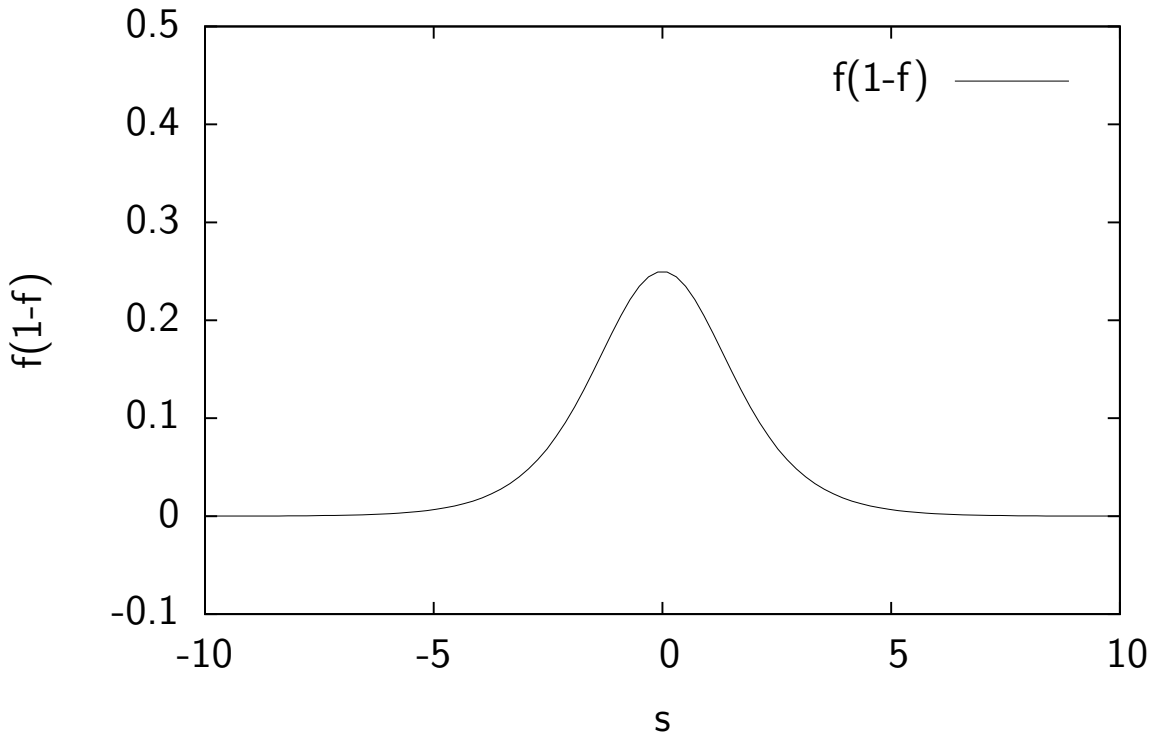
Einfluss des Schwellenwerts θ



Ableitung der logistischen Funktion

$$
\begin{aligned}
\frac{\partial f(s)}{\partial s} &= \frac{\partial}{\partial s}\frac{1}{1+e^{-s}} = \frac{\partial}{\partial s}(1+e^{-s})^{-1} = \overbrace{-(1+e^{-s})^{-2}}^{\text{äußere Ableitung}} \cdot \overbrace{(-e^{-s})}^{\text{innere Ableitung}} \\
&= \frac{e^{-s}}{1+e^{-s})^2} = \frac{1+e^{-s}-1}{(1-e^{-s})^2} \\
&= \frac{1+e^{-s}}{(1+e^{-s})^2} - \frac{1}{(1+e^{-s})^2} \\
&= f - f^2 = f(1-f) \\
\frac{\partial f(s)}{\partial s}|_{s=0} &= \frac{1}{1+1}(1-\frac{1}{1+1}) = \frac{1}{4}
\end{aligned}
$$

Graph von $f(1-f)$



Wirkung:

- Nahe an der Hyperebene durch $\boldsymbol{w}$ ist Änderung stark
- Weiter davon entfernt ist sie gering (bei gleichem Fehler)

Verallgemeinerte Delta-Regel

Fehlergradienten somit

$$\frac{\partial \varepsilon}{\partial \boldsymbol{w}} = -2(d - f)f(1 - f) \cdot \boldsymbol{x}$$

Wie bei alter Delta-Regel, neue Schätzung für w

$$\boldsymbol{w}^{(\text{neu})} \leftarrow \boldsymbol{w}^{(\text{alt})} + \eta \cdot (d - f)f(1 - f) \cdot \boldsymbol{x}$$

„Variable“ Lernrate η, um Änderungsstärke einzustellen

Weitere Verbesserungen des Lernverfahrens sind möglich

Lernen $\rightarrow$ Übungsaufgabe, Hinweis zur Tabelle:

Epoche	x_1	x_2	d	$\boldsymbol{xw}$	f	ε	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2

Übersicht

1. Einleitung

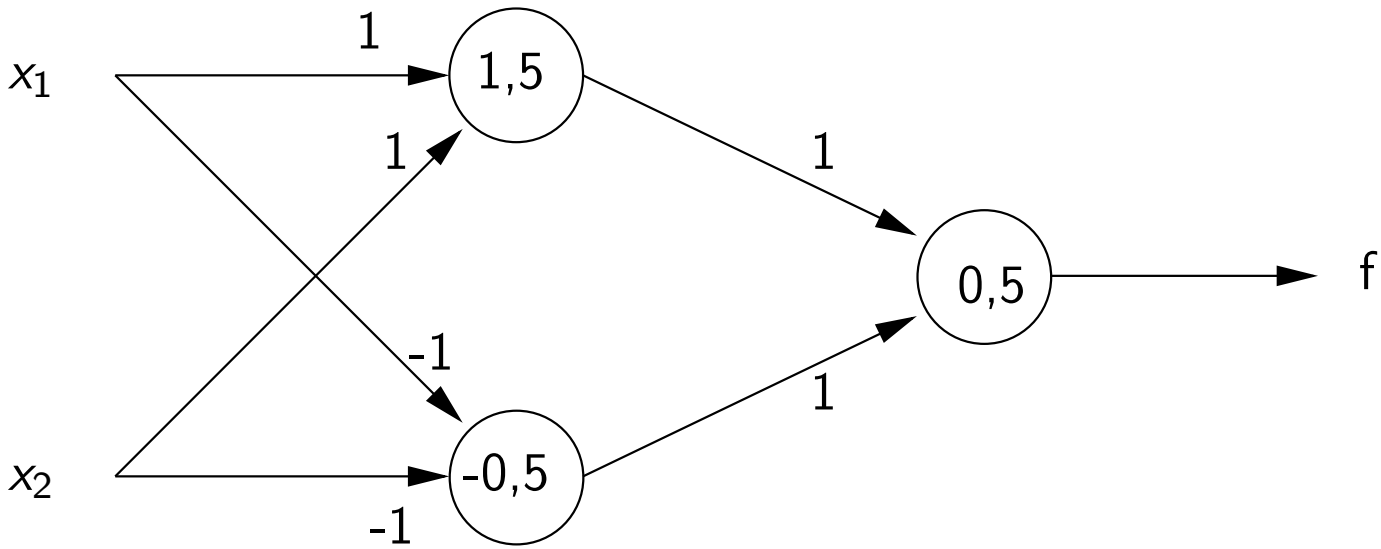
2. Schwellenwert-Elemente

3. **Mehrschichtige Perzeptrons**
   - Backpropagation
   - Beispiel
   - Bemerkungen

Mehrschichtige Perzeptrons

Falls Problem nicht linear separabel, dann mehrschichtige NNs

Beispiel: Funktion *gleiche Parität*



Zweischichtiges NN

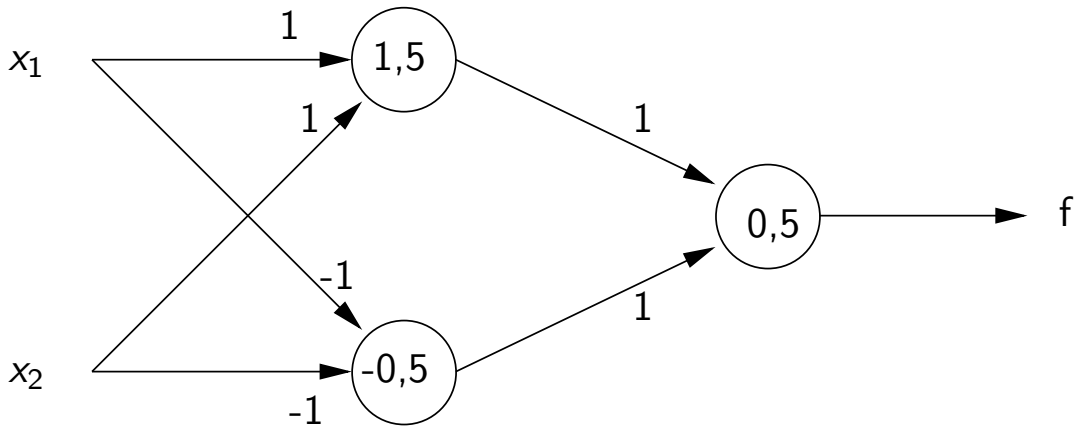
Dreischichtiges NN falls Eingabeschicht mitzählt

Neuronen in mittlerer Schicht: verborgene/innere Neuronen

Propagation am Beispiel

$$f(x_1, x_2) = (x_1 \wedge x_2) \vee (\overline{x_1} \wedge \overline{x_2})$$

Eingabe		Neuron I		Neuron II		Neuron III		Ausgabe
x_1	x_2	Eingabe	Ausgabe	Eingabe	Ausgabe	Eingabe	Ausgabe	
0	0	0	0	0	1	1	1	1
0	1	1	0	-1	0	0	0	0
1	0	1	0	-1	0	0	0	0
1	1	2	1	-2	0	1	1	1



Mehrschichtige Perzeptrons – Definition

k-schichtiges Perzeptron besteht aus k Schichten, wobei in Schicht j Neuronen m_j mit Sigmoiden Ausgabe x^j liefern

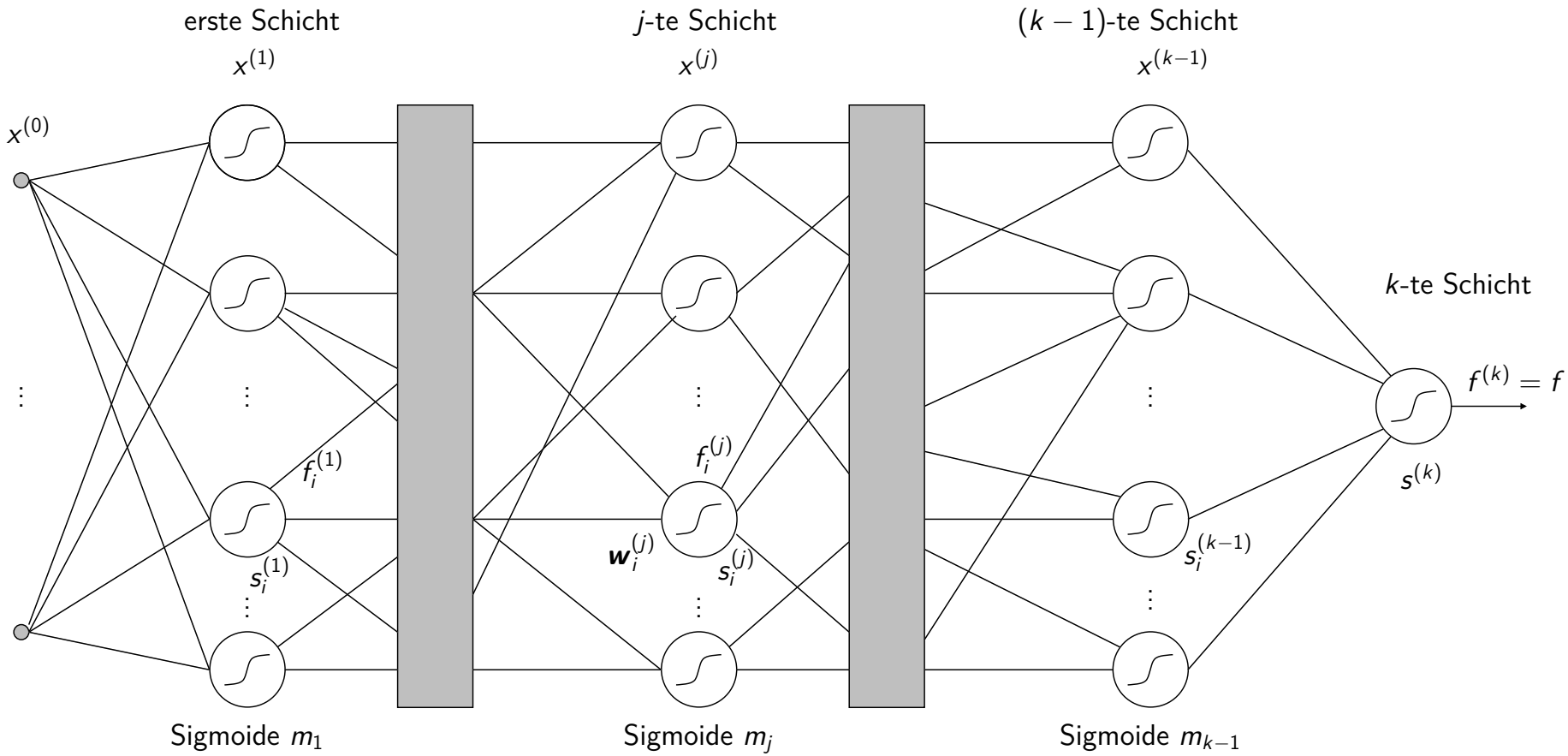
$x^{(0)}$ ist Eingabe

f ist Ausgabe

$\boldsymbol{w}_i^j$ ist Gewichtsvektor des i-ten Sigmoids in Schicht j, wobei letzte Komponente jeweils θ

Aktivierung des Neurons = Summe $s_i^j = \boldsymbol{x}^{j-1} \cdot \boldsymbol{w}_i^j$

Mehrschichtige Perzeptrons



Backpropagation

Spezielles Trainingsverfahren

Fehler wird durch Netz schichtweise zurückgeschickt

Idee: analog zu Gradientenabstieg, berechne Fehlergradient

Herleitung:
Fehler für Ausgabeschicht $\varepsilon = (d - f)^2$
Gradient:

$$\frac{\partial \varepsilon}{\partial \boldsymbol{w}_i^j} = \left[\frac{\partial \varepsilon}{\partial w_{1i}^j}, \ldots, \frac{\partial \varepsilon}{\partial w_{li}^j}, \ldots, \frac{\partial \varepsilon}{\partial w_{m_{j-1}+1,i}^j}, \right]$$

wobei w_{li}^j die l-te Komponenten von $\boldsymbol{w}_i^j$

Backpropagation – Herleitung

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial \boldsymbol{w}_i^j} &= \frac{\partial \varepsilon}{\partial s_i^j} \cdot \frac{\partial s_i^j}{\partial \boldsymbol{w}_i^j} && \left(\varepsilon \text{ hängt von } \boldsymbol{w}_i^j \text{ nur über } s_i^j \text{ ab}\right) \\
&= \frac{\partial \varepsilon}{\partial s_i^j} \cdot \boldsymbol{x}^{j-1} && \left(s_i^j = \boldsymbol{x}^{j-1} \cdot \boldsymbol{w}_i^j \text{ und somit } \frac{\partial s_i^j}{\partial \boldsymbol{w}_i^j}\right) \\
&= -2(d-f)\frac{\partial f}{\partial s_i^j} \cdot \boldsymbol{x}^{j-1} && \left(\frac{\partial \varepsilon}{\partial s_i^j} = \frac{\partial (d-f)^2}{\partial s_i^j} = -2(d-f)\frac{\partial f}{\partial s_i^j}\right) \\
&= -2(\delta_i^j \boldsymbol{x}^{j-1}) && \left(\delta_i^j := (d-f)\frac{\partial f}{\partial s_i^j} = -\frac{1}{2}\frac{\partial \varepsilon}{\partial s_i^j}\right)
\end{aligned}
$$

Berechnung neuer Gewichte $(\boldsymbol{w}_i^j)^{(\text{neu})} \leftarrow (\boldsymbol{w}_i^j)^{(\text{alt})} + \eta_i^j \cdot \delta_i^j \boldsymbol{x}^{j-1}$

Lernrate η_i^j meistens ein Wert für gesamtes Netz

Backpropagation – Herleitung

Gewichtsänderungen in der Ausgabeschicht:

$$\delta^k = (d - f)\frac{\partial f}{\partial s^k} = (d - f)f(1 - f)$$

$$\boldsymbol{w}^{k,\text{neu}} \leftarrow \boldsymbol{w}^{k,\text{alt}} + \eta^k (d - f)f(1 - f)\boldsymbol{x}^{k-1}$$

Anmerkungen:

- Gilt bei logistischer Funktion mit Bias 0 (wegen zusätzlicher Eingabe θ)
- Nur ein Wert, daher keine Indizes
- Sonst wie bei verallgemeinerter Delta-Regel

Backpropagation – Herleitung

Gewichtsänderungen in verborgener Schicht:

$$\delta_i^j = (d - f) \sum_{l=1}^{m_{j+1}} \frac{\partial f}{\partial s_l^{j+1}} \cdot \frac{\partial s_l^{j+1}}{\partial s_i^j} = \sum_{l=1}^{m_{j+1}} \delta_l^{j+1} \frac{\partial s_l^{j+1}}{\partial s_i^j}$$

Summand muss noch berechnet werden

Betrachte hierzu zunächst:

$$s_l^{j+1} = \boldsymbol{x}^j \cdot \boldsymbol{w}_l^{j+1} = \sum_{\gamma=1}^{m_{j+1}} x_\gamma^j \cdot w_{\gamma l}^{j+1} = \sum_{\gamma=1}^{m_{j+1}} f_\gamma^j \cdot w_{\gamma l}^{j+1}$$

(γ ist Index über einzelnen Vektorelemente)

Backpropagation – Herleitung

$$\begin{aligned}
\frac{\partial s_l^{j+1}}{\partial s_i^j} &= \frac{\partial \left[\sum_{\gamma=1}^{m_{j+1}} f_\gamma^j w_{\gamma l}^{j+1}\right]}{\partial s_i^j} \\
&= \sum_{\gamma=1}^{m_{j+1}} w_{\gamma l}^{j+1} \frac{\partial f_\gamma^j}{\partial s_i^j} \qquad \left(\text{da } \frac{\partial f_{\gamma j}}{\partial s_i^j} = 0 \text{ für } \gamma \neq i,\right. \\
&= w_{il}^{j+1} f_i^j (1 - f_i^j) \qquad \left.\text{sonst } \frac{\partial f_\gamma^j}{\partial s_\gamma^j = f_\gamma^j (1 - f_\gamma^j)}\right)
\end{aligned}$$

Somit rekursives Gleichungssystem zur Berechnung der δ_i^j

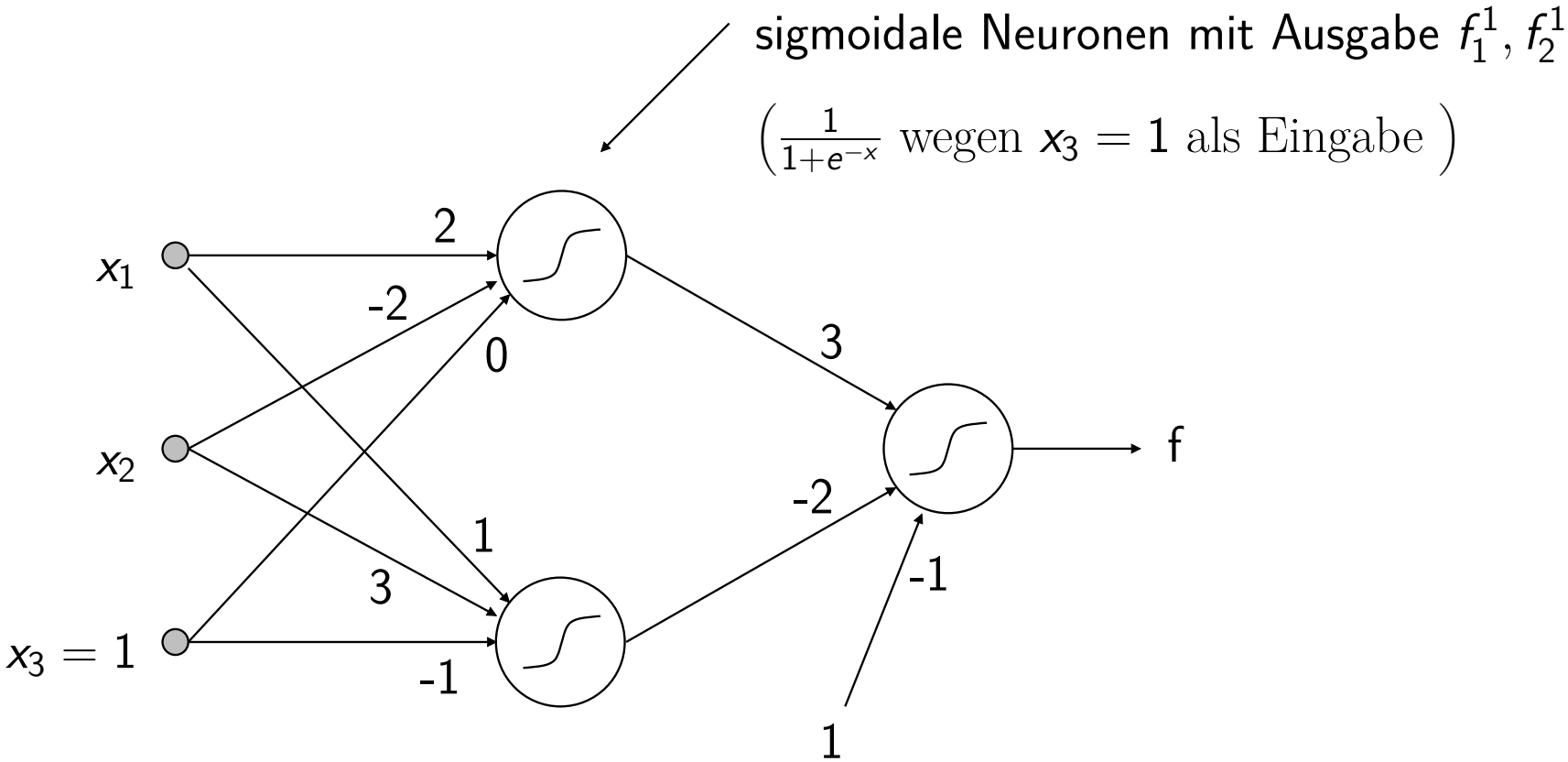
$$\delta_i^j = f_i^j (1 - f_i^j) \sum_{l=1}^{m_{j+1}} \delta_l^{j+1} \cdot w_{il}^{j+1} \text{ und } \delta^k = (d - f) f (1 - f)$$

Schließlich: Berechnung der neuen Gewichte

$$\boldsymbol{w}_i^{j(\text{neu})} \leftarrow \boldsymbol{w}_i^{j(\text{alt})} + \eta \cdot \delta_i^j \cdot f_i^{j-1}$$

Backpropagation – Beispiel

Netz mit zufällig generierten Gewichten:



Backpropagation – Beispiel

Lernstichprobe:
$\{((1,0,1),0), \quad ((0,1,1),0), \quad ((0,0,1),1), \quad ((1,1,1),1)\}$

Propagation: $(1,0,1) \rightarrow f_1^1 = 0.881, \quad f_2^1 = 0.5, \quad f = 0.655$

Backpropagation — Fehlersignale:

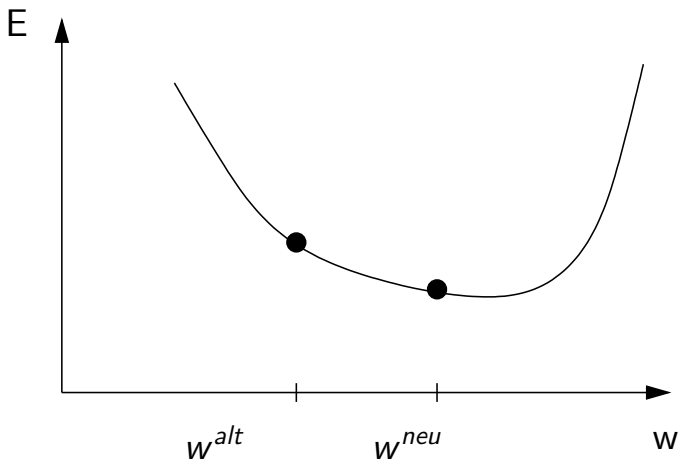
$$\delta^{(2)} = -0.148, \quad \delta_1^{(1)} = -0.047, \quad \delta_2^{(1)} = 0,074$$

Neue Gewichte mit $\eta = 1$:

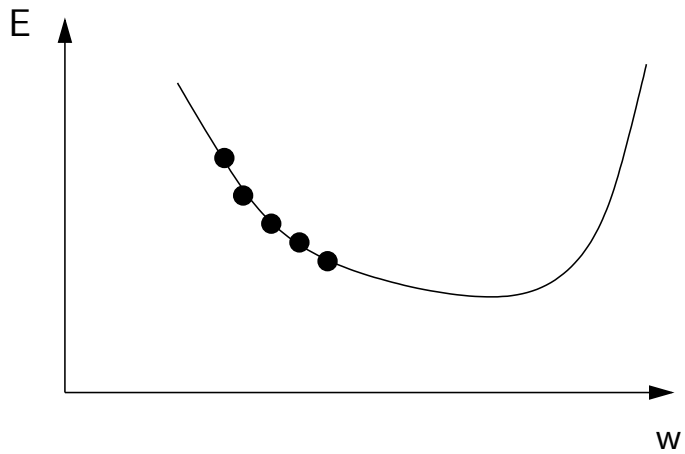
$$\boldsymbol{w}_1^{(1)} = (1.935, -2, -0.047)$$
$$\boldsymbol{w}_2^{(1)} = (1.074, 3, -0.926)$$
$$\boldsymbol{w}^{(2)} = (2.870, -2.074, -1.148)$$

Effekt der Lernrate

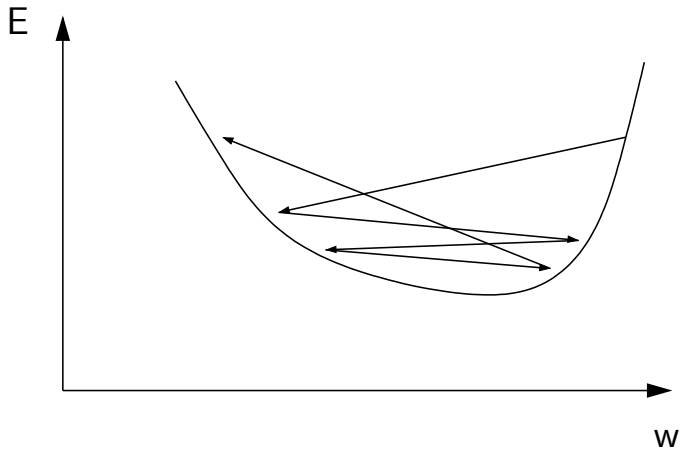
gewünschtes Verhalten:



zu klein: Lernen verläuft zu langsam



zu groß: Pingpong-Effekt

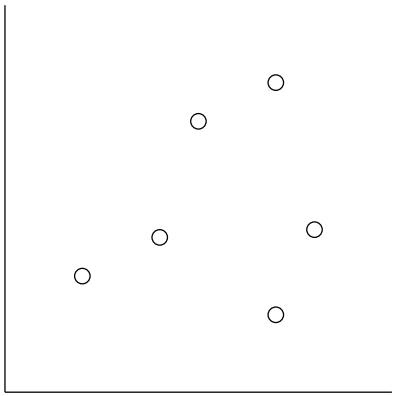


Eigenschaften von Neuronalen Netzen

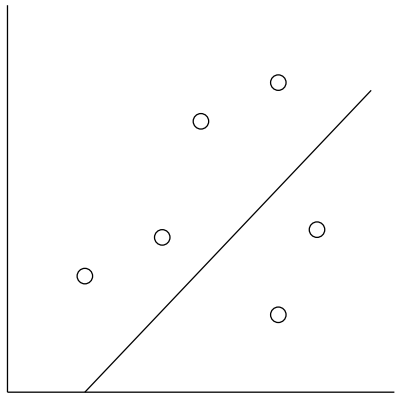
NN können *generalisieren*, d.h. Vektoren klassifizieren, die $\notin L$

Möglich: Maß der Generalisierungsgüte

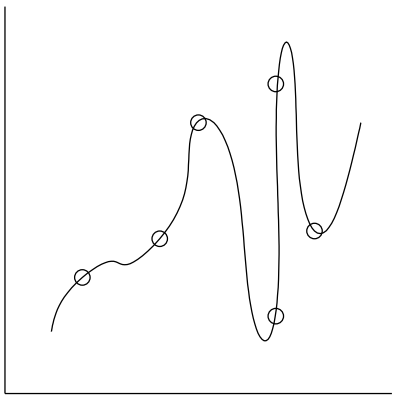
Analogie: Funktionsapproximation



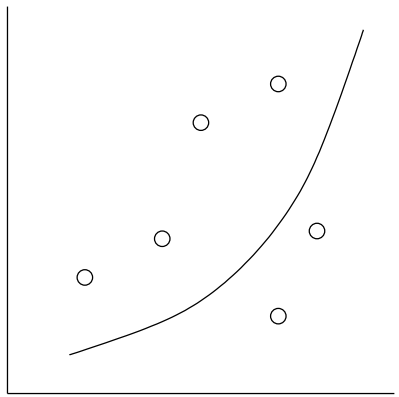
Datensatz



zu einfach,
schlechte Güte



zu kompliziert,
kein Fehler,
Daten auswendig gelernt,
schlechte Generalisierung,
(overfitting)



Mittelweg
Occam's razor

Vorgehensweise beim Lernen

Lernstichprobe = Trainingsmenge + Validierungsmenge

- Trainingsmenge (ca. 2/3 der Daten) zum Lernen
- Validierungsmenge (ca. 1/3 der Daten) für Güteschätzung

Kreuzvalidierung (*cross validation*)

- Lernstichprobe in n disjunkte Mengen teilen
- Jeweils $n - 1$ Mengen fürs Training und 1 Menge für Validierung
- Ermittlung der n *out-of-sample errors*

Anwendungen von NNs:

- Gesichtserkennung, Börsenprognosen, ...
- Data Mining (siehe z.B. unseren Artikel in *Spektrum der Wissenschaften*, Nov. 2002, S. 80 ff.)

USPS-Datensatz, Originaldaten

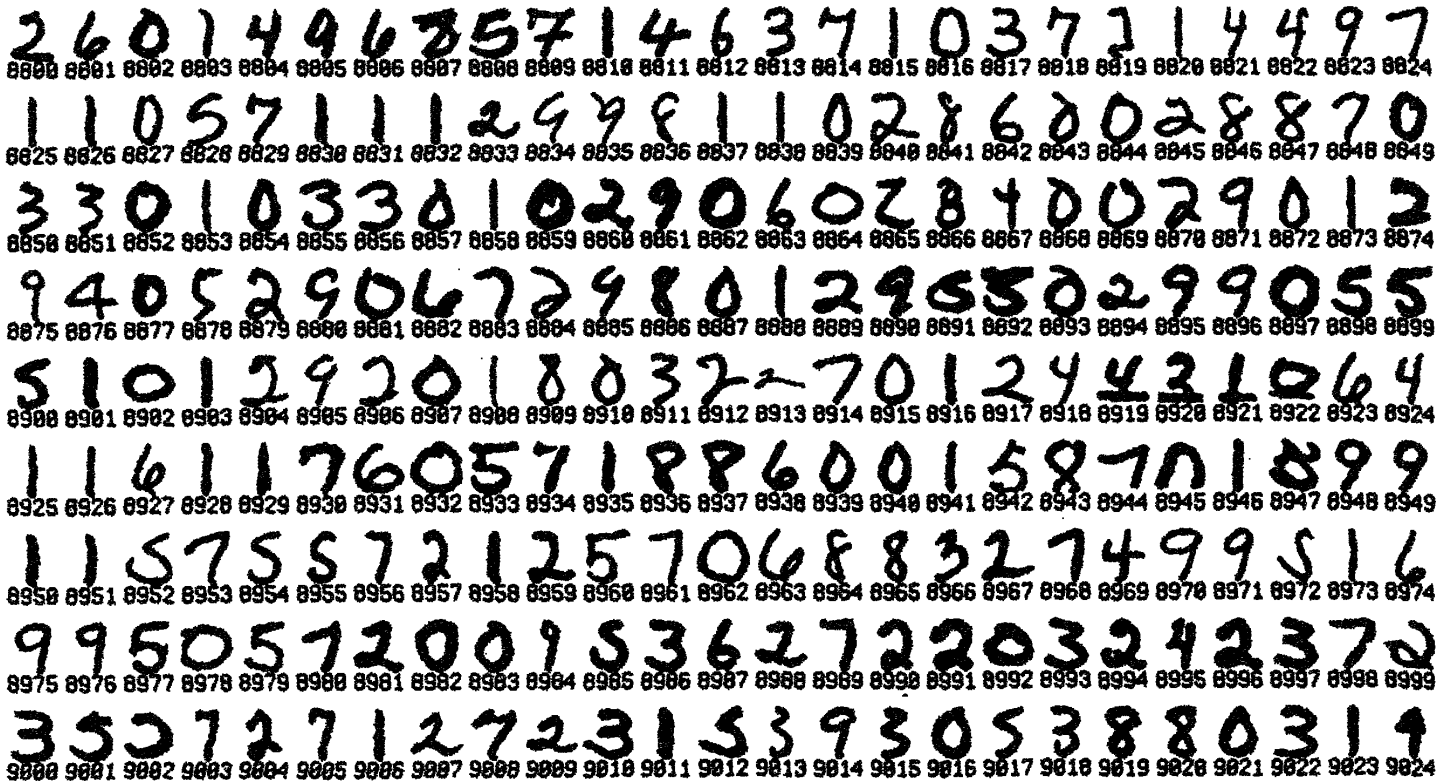


Abbildung: USPS-Datensatz, Originaldaten (segmentierte Ziffern) mit Labels

USPS-Datensatz

Problemstellung: Erkennung von Handschrift

- Datensatz: United States Postal Service (USPS)
- Automatisch gescannte Postleitzahl-Ziffern
- Schwarz-Weiß-Bilder der Ziffern
- 16x16 Pixel, (nach Vorverarbeitung)
- Lerndatensatz: 7291 Beispiele
- Testdatensatz: 2007 Beispiele

Zeile im Datensatz: Ziffer

Codierung: Graustufenwerte von 0 (weiß) bis 2 (schwarz)

D.h. Tabelle mit 7291 Zeilen und $256 + 1$ Spalten

(nach [LeCun et al., 1990])

USPS-Datensatz, Beispielcodierung

Ziffer	p_1	p_2	p_3	p_4	...	p_{255}	p_{256}
6	0	0	0	0	...	0	0
5	0	0	0	0.187	...	0.172	0
4	0	0	0	0	...	0	0
2	0	0	0	0	...	0	0.144
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

USPS-Datensatz, Klassifikation

- Trainieren eines MLP auf dem Trainingsdatensatz
  - 256 Eingabeneuronen
  - Mehrere versteckte Schichten zur Merkmalsbestimmung
  - 10 Ausgabeneuronen (eins pro Klasse/Ziffer)
  - Insgesamt: 6465 Neuronen, ca. 150.000 Verbindungen, 3658 freie Parameter
  - Propagieren der Trainingsmuster durch das Netz
  - Bestimmung des Fehlers
  - Rückpropagation des Fehlers, Anpassung der Netzparameter
- Testen des oben trainierten MLP
  - Propagation des Testdatensatzes
  - Bestimmung der Fehlklassifikationsrate

USPS-Datensatz, Beispielnetz

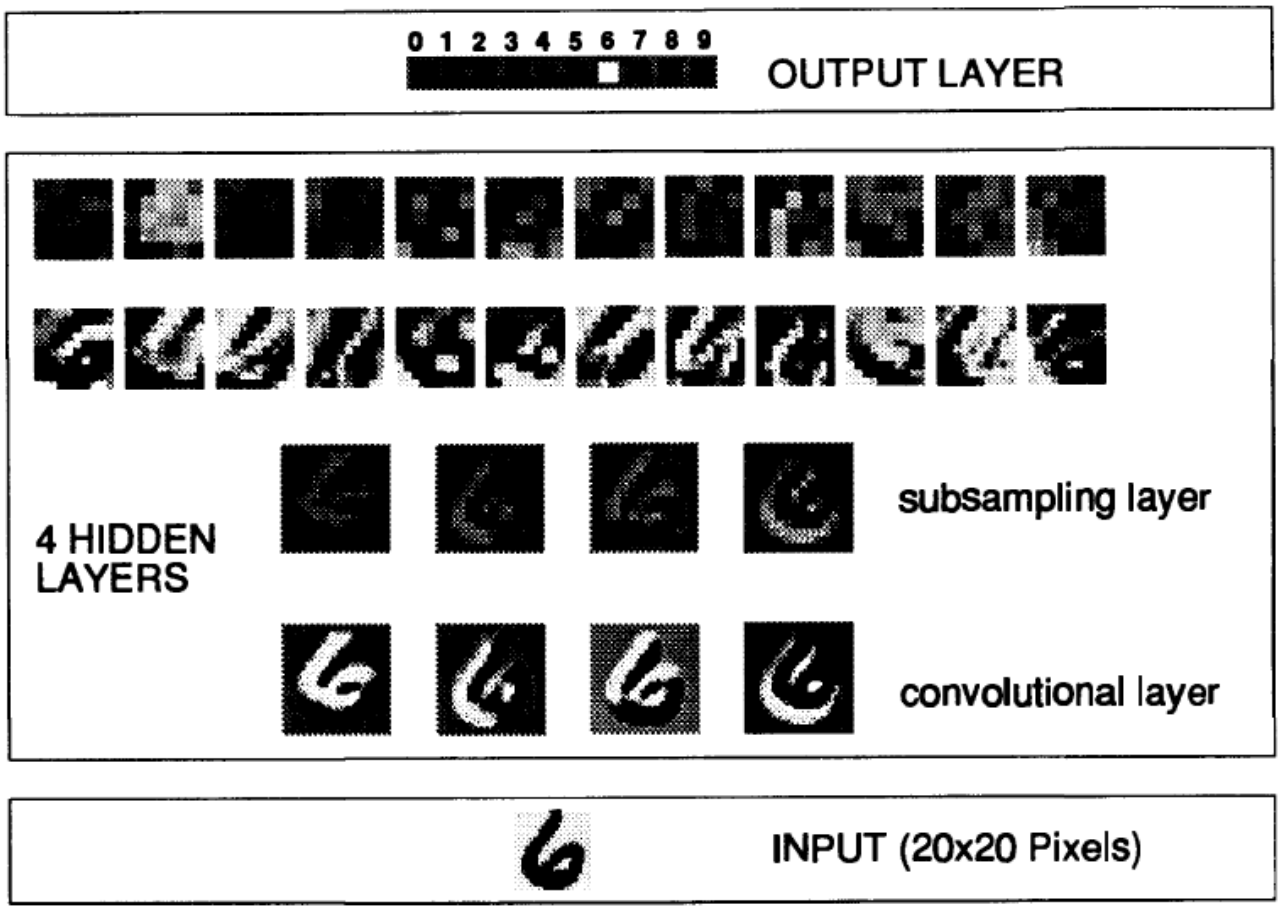


Abbildung: Ziffererkennung im USPS-Datensatz mit mehrschichtigem Perzeptron, nach [Matan et al., 1992]

USPS-Datensatz, Erkennungsleistung

„Erkenner“	Fehlklassifikation
Entscheidungsbaum C4.5	16,2%
Bestes zweischichtiges Netz	5,9%
Fünfschichtiges Netz	5,1%
Support-Vektor-Maschine	>4,0%
Mensch	2,5%

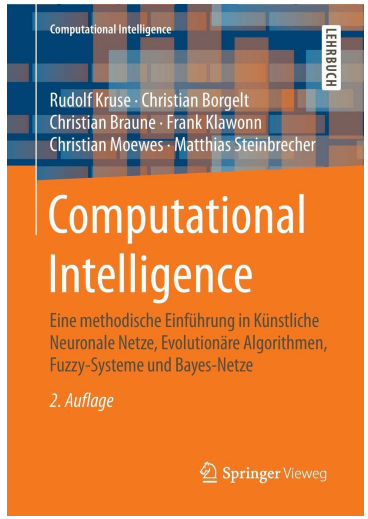
Tabelle: Erkennungsleistung verschiedener Klassifikatoren auf USPS-Datensatz

Weitere Details...

Im Sommersemester: Vorlesung *Neuronale Netze*

Detailliertere Betrachtung der hier gezeigten Prinzipien

Außerdem: weitergehende Themen und Netztypen



Weitere Informationen in unserem aktuellen Buch [Kruse et al., 2015]

Literatur zur Lehrveranstaltung

Kruse, R., Borgelt, C., Braune, C., Moewes, C., and Steinbrecher, M. (2015).
Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze.
Springer Vieweg, Wiesbaden.

LeCun, Y., Matan, O., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., and Baird, H. S. (1990).
Handwritten zip code recognition with multilayer networks.
In *Proc. of the International Conference on Pattern Recognition*, volume II, pages 35–40, Atlantic City. IEEE Press.

Matan, O., Bromley, J., Burges, C., Denker, J., Jackel, L., LeCun, Y., Pednault, E., Satterfield, W., Stenard, C., and Thompson, T. (1992).
Reading handwritten digits: A zip code recognition system.
IEEE Computer, 25(7):59–63.