INF FAKULTÄT FÜR INFORMATIK

Intelligente Systeme

Neuronale Netze

Prof. Dr. R. Kruse C. Braune C. Moewes

{kruse,cbraune,cmoewes}@iws.cs.uni-magdeburg.de

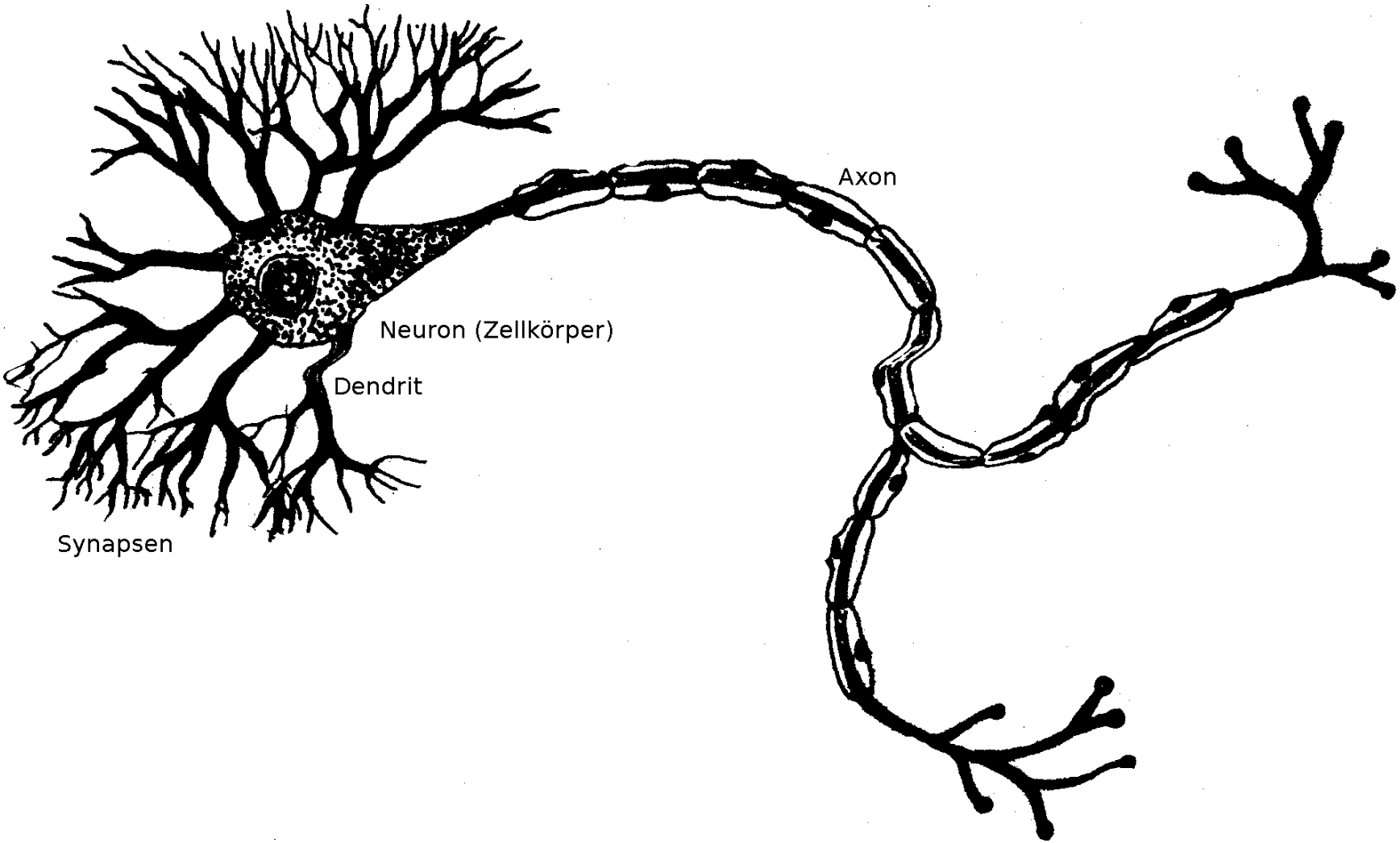
Institut für Wissens- und Sprachverarbeitung
Fakultät für Informatik
Otto-von-Guericke Universität Magdeburg

Übersicht

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	4 CPUs, $2 \cdot 10^9$ Transistoren	10^{11} Neuronen
Speicherkapazität	$8 \cdot 10^9$ Bytes RAM, 10^{12} Bytes Festspeicher	10^{11} Neuronen, 10^{14} Synapsen
Verarbeitungs-geschwindigkeit	10^{-8} s	10^{-3} s
Bandbreite	$3.4 \cdot 10^{10}$ bits/s	10^{14} bits/s
Neuronale Updates pro Sekunde	10^5	10^{14}

Konventionelle Rechner vs. Gehirn

- beachte: Hirnschaltzeit von 10^{-3} s recht langsam
- aber: Updates erfolgen parallel
- dagegen: serielle Simulation auf Rechner mehrere 100 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
- daher sinnvoll, Vorteile natürlicher NN künstlich nachzuahmen

Künstliche Neuronale Netze (NN)

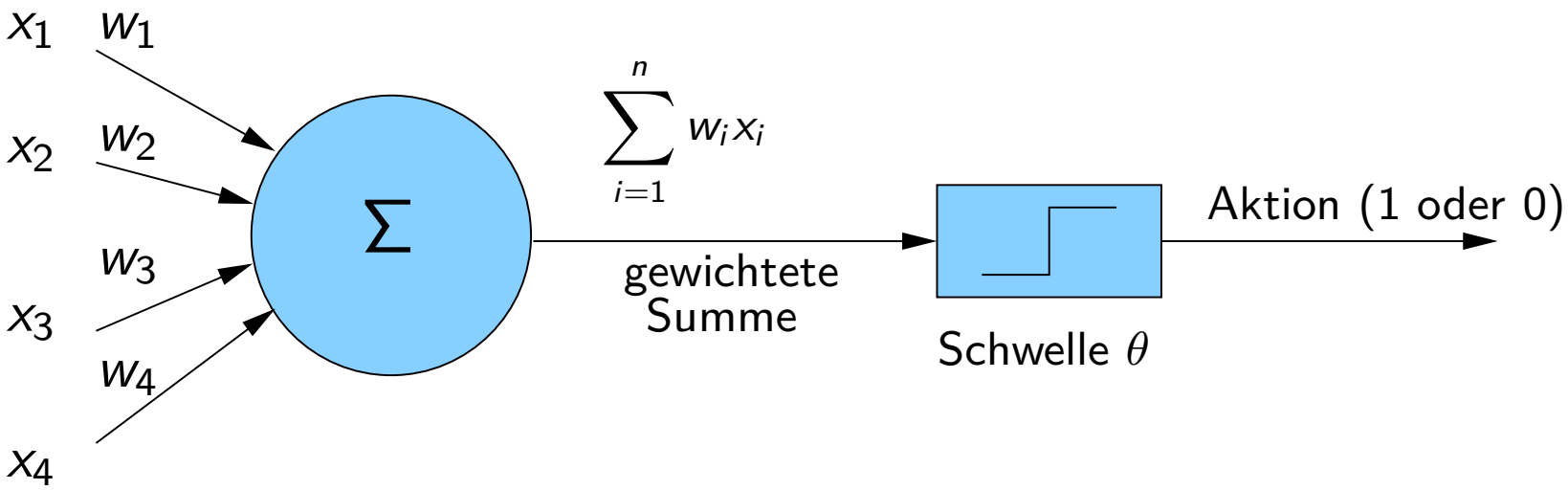
- **mathematisch:** Methode, Funktionen zu repräsentieren
  - Netzwerke von einfachen Berechungselementen (vergleichbar mit logischen Schaltkreisen)
  - Lernen von Beispielen
  - *Sichtweise in dieser Vorlesung!*
- **biologisch:** stark vereinfachtes Modell des Gehirns und seiner Funktionsweise
  - Konnektionismus
  - *Nicht in dieser Vorlesung!*

Übersicht

Perzeptrons

- Implementierung eines S-R-Agenten: verschiedene Möglichkeiten
- S-R-Agent repräsentiert eine Funktion
- im Folgenden: Funktion = Perzeptron
- auch Schwellenwert-Element genannt
- engl. *threshold logic unit (TLU)*

Ein Perzeptron



formale Definition:

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

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

Perzeptrons: Vereinfachte Schreibweise

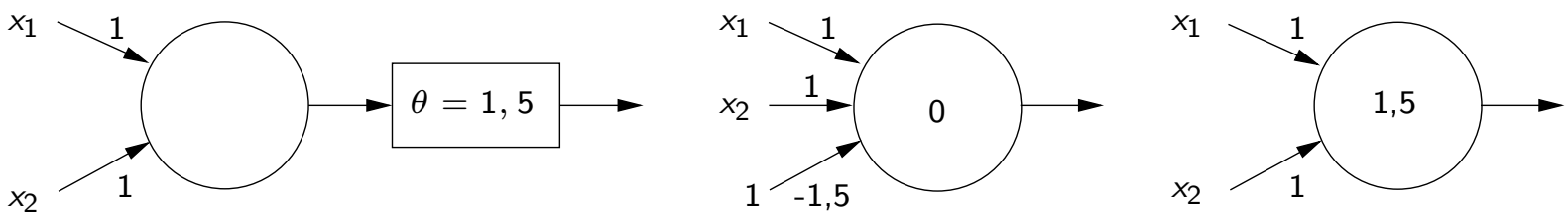
seien $x_{n+1} \stackrel{\text{def}}{=} 1$ und $w_{n+1} \stackrel{\text{def}}{=} -\theta$

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

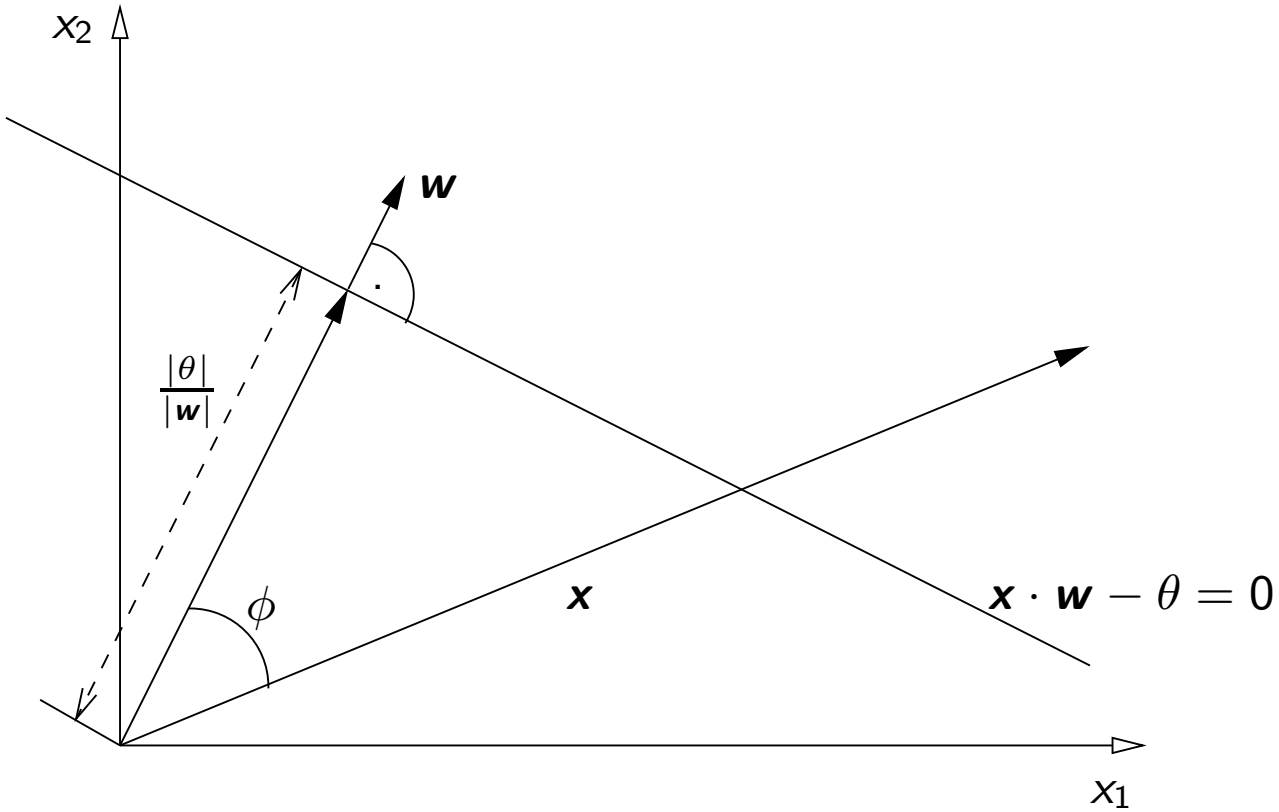
Eingabevektor $\boldsymbol{x} = (x_1, \ldots, x_n)$

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

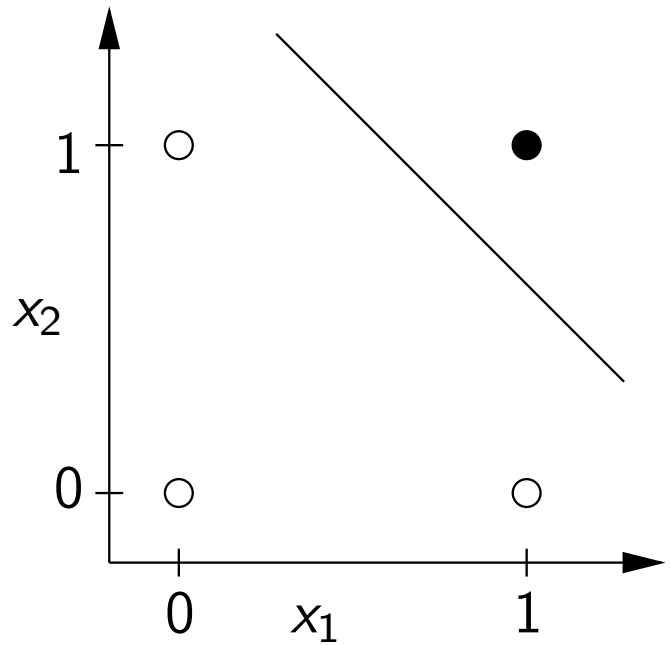


- θ ist negativ falls Ursprung auf Ebenenseite, in die $\boldsymbol{w}$ zeigt
- ansonsten ist θ positiv
- $\boldsymbol{w} \cdot \boldsymbol{x} - \theta > 0$ falls x auf Ebenenseite, in deren Richtung $\boldsymbol{w}$ zeigt

Geometrische Interpretation

- Perzeptron repräsentiert eine linear separable Funktion
- logisches `AND` ist linear separabel
- somit durch Perzeptron darstellbar

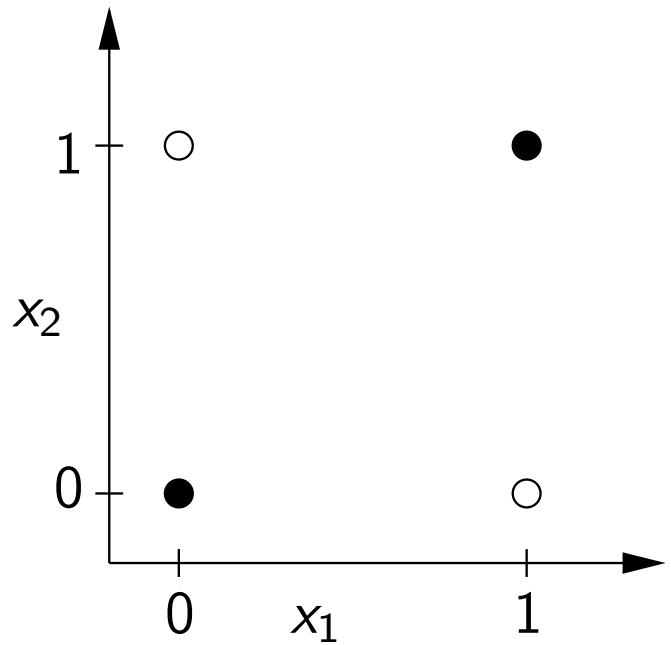
x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1



Biimplikationsproblem

- Biimplikation (Äquivalenz) ist nicht linear separabel
- also existiert kein Perzeptron dafür

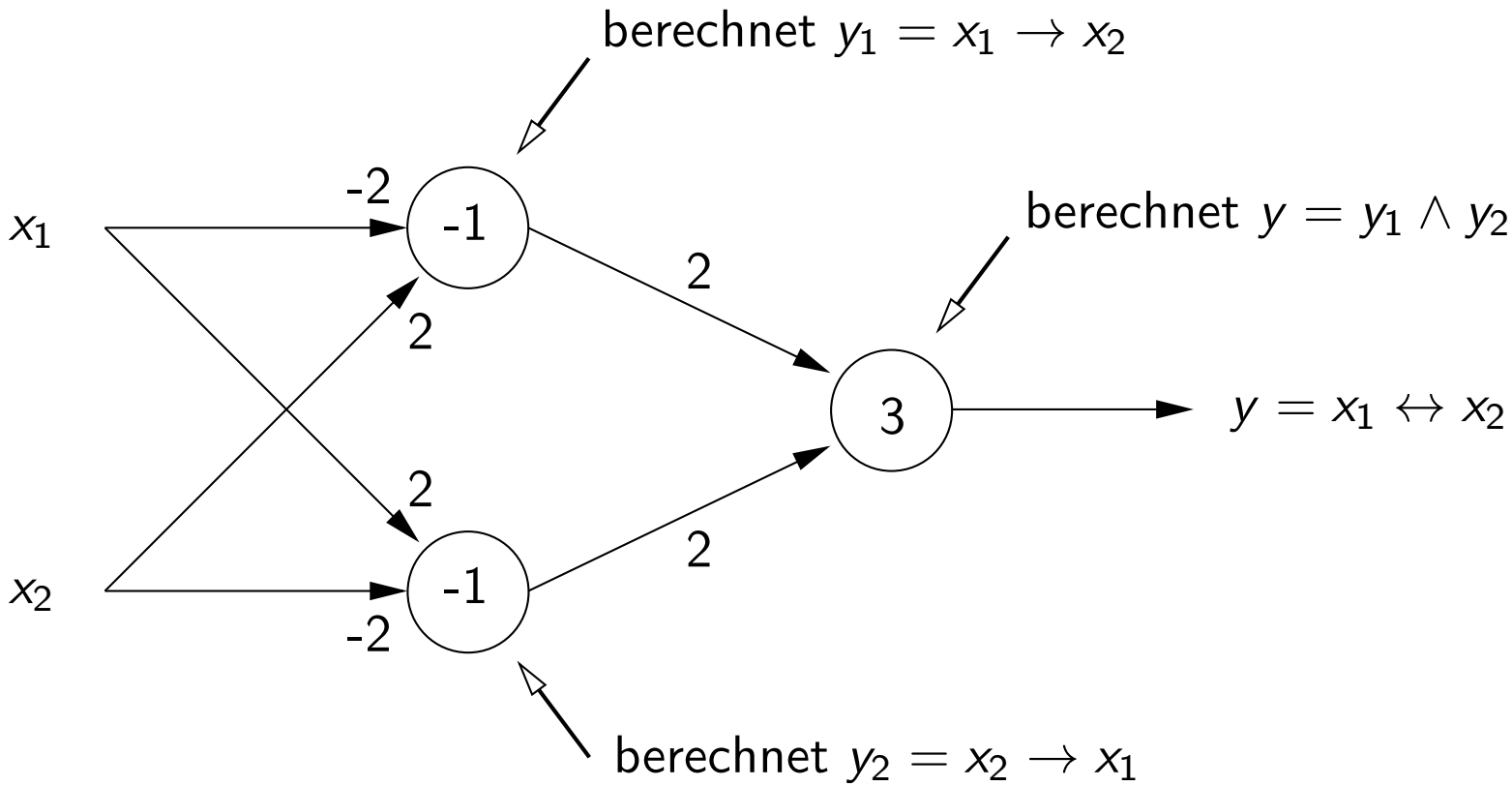
x_1	x_2	$x_1 \leftrightarrow x_2$
0	0	1
1	0	0
0	1	0
1	1	1



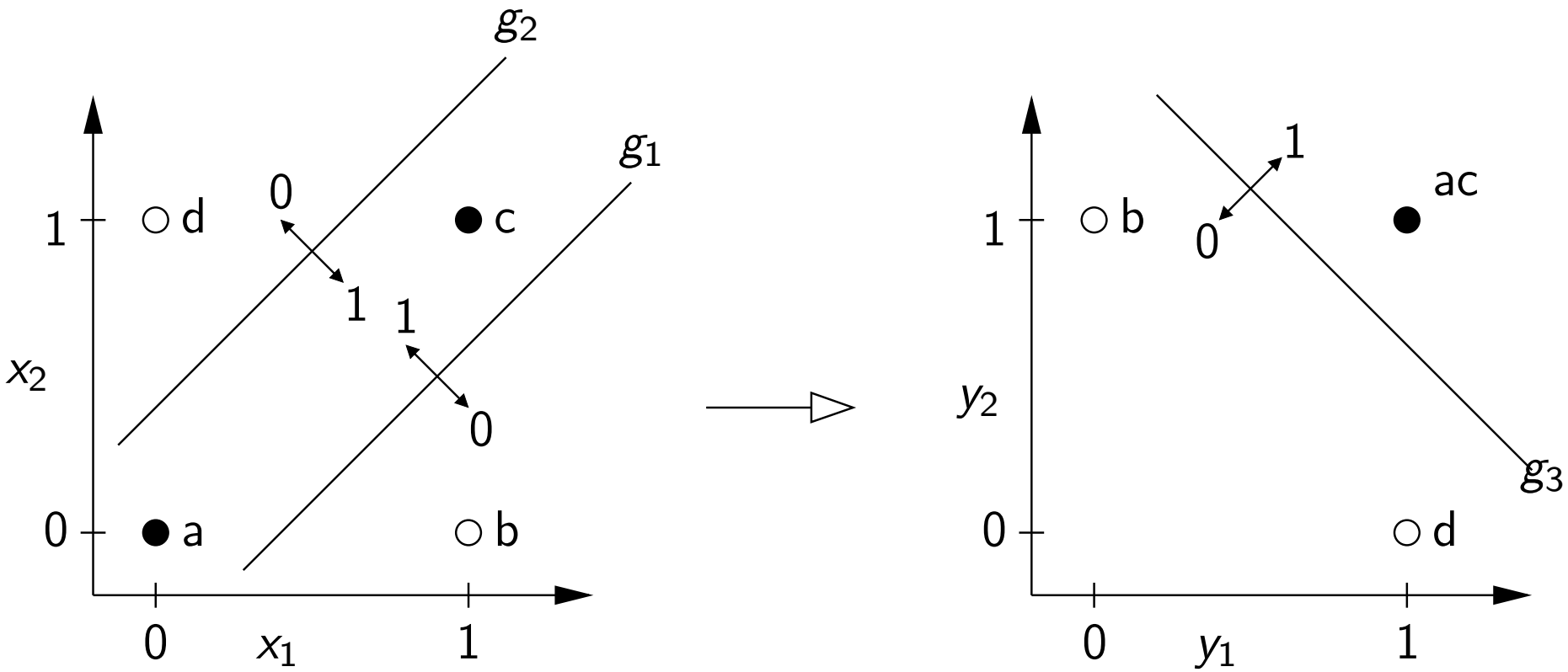
- es gibt keine Trenngerade, die Lösungsraum aufteilt

Biimplikationsproblem

- Lösung: Zusammenschalten mehrerer Schwellenwertelemente



Biimplikationsproblem



Geometrische Deutung des Zusammenschaltens mehrerer Schwellenwertelemente zur Berechnung der Biimplikation

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: Elemente $\boldsymbol{x}_i$ aus L stimmen mit d_i und 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

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

Gradientenabstiegsverfahren

- typisches Optimierungsverfahren
- möglich: Minimieren von ε für alle $(\boldsymbol{x}_i, d_i) \in L$ gleichzeitig
- hier: Minimieren von ε der Reihe nach (inkrementell)

Gradientenabstiegsverfahren

- *Idee:* Bestimmung der Abhängigkeit von ε zu $\boldsymbol{w}$ durch Berechnung partieller Ableitungen

$$\nabla_{\boldsymbol{w}} \varepsilon = \frac{\partial \varepsilon}{\partial \boldsymbol{w}} = \left[\frac{\partial \varepsilon}{\partial w_1}, \ldots, \frac{\partial \varepsilon}{\partial w_{n+1}} \right]$$

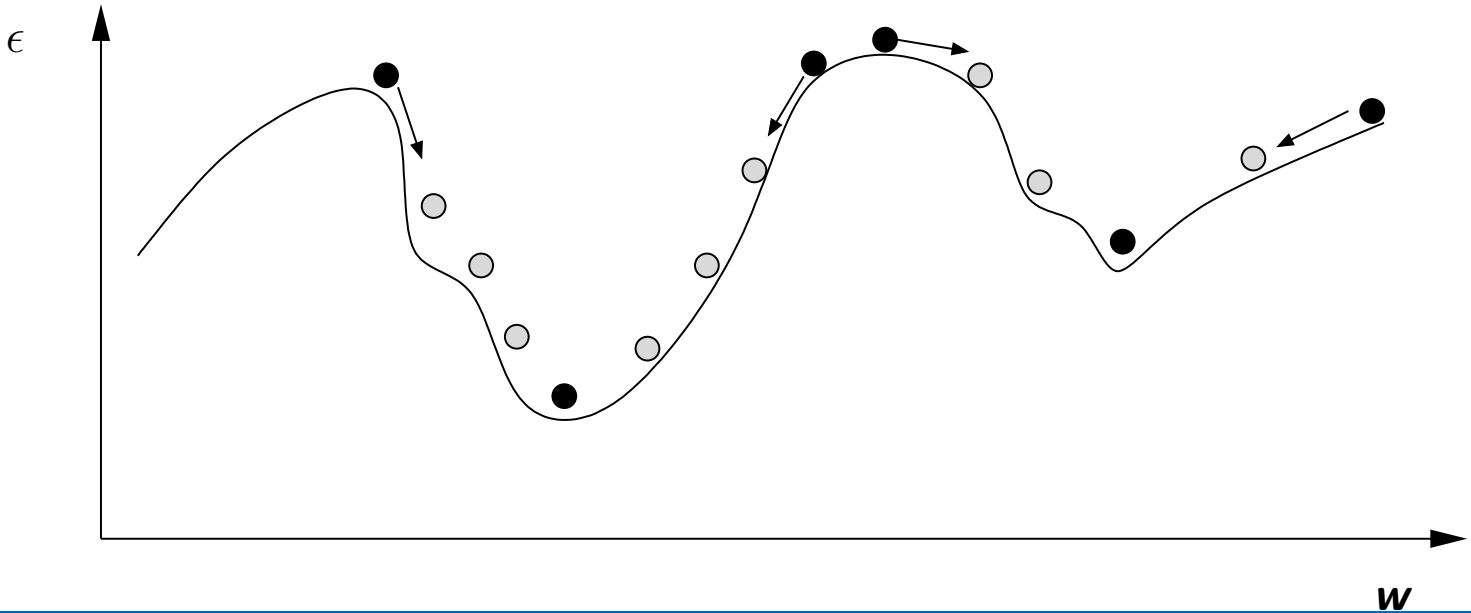
- anschließend: Berechnung neuer Gewichte

$$\boldsymbol{w}^{(\text{neu})} \stackrel{\text{def}}{=} \boldsymbol{w}^{(\text{alt})} + \eta \cdot \nabla_{\boldsymbol{w}} \varepsilon$$

- Lernrate η legt „Schrittweite“ des Abstiegs fest
- Abstieg in entgegengesetzter Richtung des Gradienten $\nabla_{\boldsymbol{w}} \varepsilon$

Gradientenabstieg – Anschauung

- zu jedem $\boldsymbol{w}$ gehört ein Fehlerwert
- „Fehlergebirge“ über dem durch $\boldsymbol{w}$'s aufgespannten Raum
- Gradientenabstieg: von gegebenem Punkt in „Gebirge“ talwärts wandern bis Talsohle erreicht
- Hängenbleiben in lokalen Minima: je nach „Fehlergebirge“, Startpunkt, und η



Gradientenabstieg für Perzeptron

Fehler $\varepsilon = (d - f)^2$

Gradient

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial \boldsymbol{w}} &= \left[\frac{\partial \varepsilon}{\partial w_1}, \ldots, \frac{\partial \varepsilon}{\partial w_{n+1}}\right] \\
&= \frac{\partial \varepsilon}{\partial s} \cdot \frac{\partial s}{\partial \boldsymbol{w}} && (\text{da } s = \boldsymbol{x} \cdot \boldsymbol{w}) \\
&= \frac{\partial \varepsilon}{\partial s} \cdot \boldsymbol{x} && (\text{da } \frac{\partial s}{\partial \boldsymbol{w}} s = \boldsymbol{x}) \\
&= -2(d - f) \cdot \frac{\partial f}{\partial s} \cdot \boldsymbol{x}
\end{aligned}$$

Problem: ε nicht stetig differenzierbar wegen θ

Übersicht

Delta-Regel

Fehlerkorrekturverfahren:

- Änderung von $\boldsymbol{w}$ wird nur bei Fehler vorgenommen
- Einführen einer Lernrate $\eta > 0$
- sei

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

Algorithmus:

1. Initialisierung von $\boldsymbol{w}$ (zufällig oder gesetzt)
2. $\varepsilon \leftarrow 0$
3. führe folgende Schritte für jedes Element $(\boldsymbol{x}, d) \in L$ durch:

Delta-Regel – Algorithmus

3. (führe folgende Schritte für jedes Element $(\boldsymbol{x}, d) \in L$ durch:)

- $\varepsilon \leftarrow \varepsilon + (d - f)^2 = \varepsilon + (d - f_{\boldsymbol{w}}(\boldsymbol{x}))^2$
- bestimme für $i = 1, \ldots, n+1$

$$\Delta_{(\boldsymbol{x},d)} w_i = \begin{cases} 0 & \text{falls } d = f_{\boldsymbol{w}}(\boldsymbol{x}) \\ \eta \cdot x_i & \text{falls } f_{\boldsymbol{w}}(\boldsymbol{x}) = 0 \text{ und } d = 1 \\ -\eta \cdot x_i & \text{falls } f_{\boldsymbol{w}}(\boldsymbol{x}) = 1 \text{ und } d = 0. \end{cases}$$

- **beachte:** $+\Delta\theta = -\Delta_{(\Delta,d)} w_{n+1}, \quad x_{n+1} = 1, \quad \theta = -w_{n+1}$
- bestimme neue Gewichte

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta_{(\boldsymbol{x},d)}(\boldsymbol{w})$$

4. terminiere falls ε minimal, sonst weiter mit Schritt 2

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*

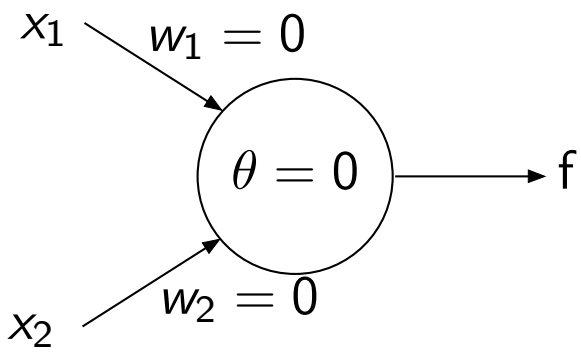
Algorithmus terminiert für linear separable Lernstichproben

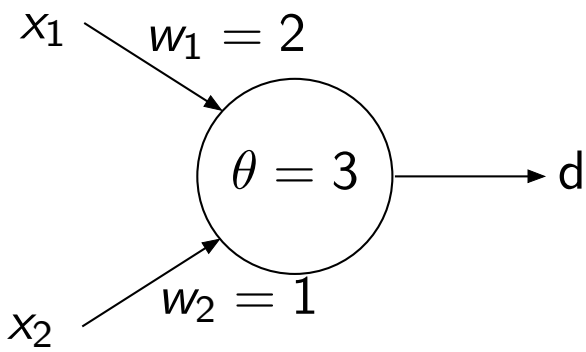
Delta-Regel – Beispiel AND

- Lernvorgang liefert

Initialisierung	Nach dem Lernen

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





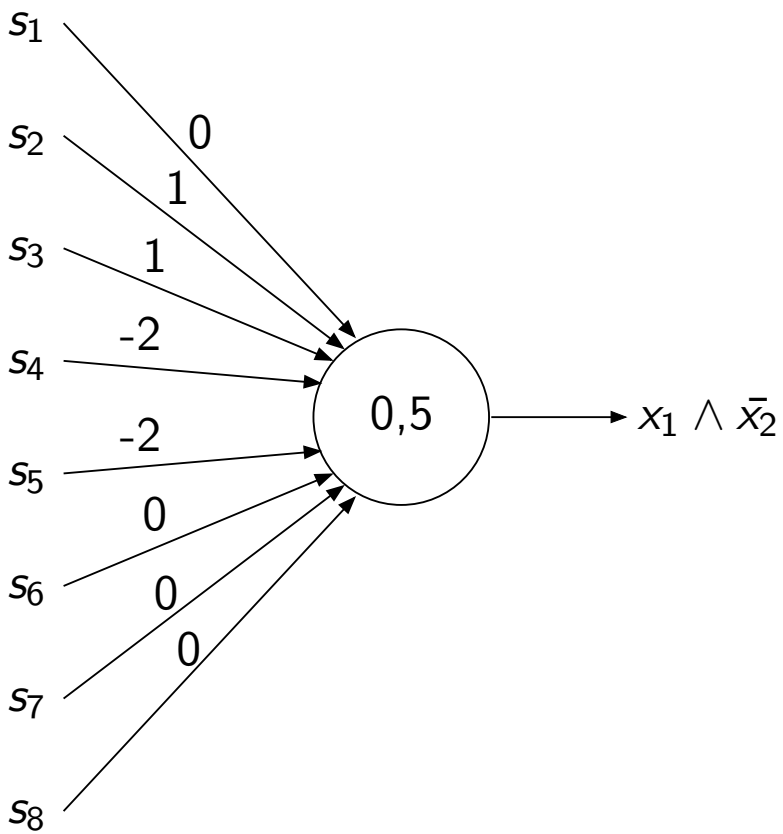
- Rechnung → Übung, Hinweis: Tabelle mit folgenden Werten

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

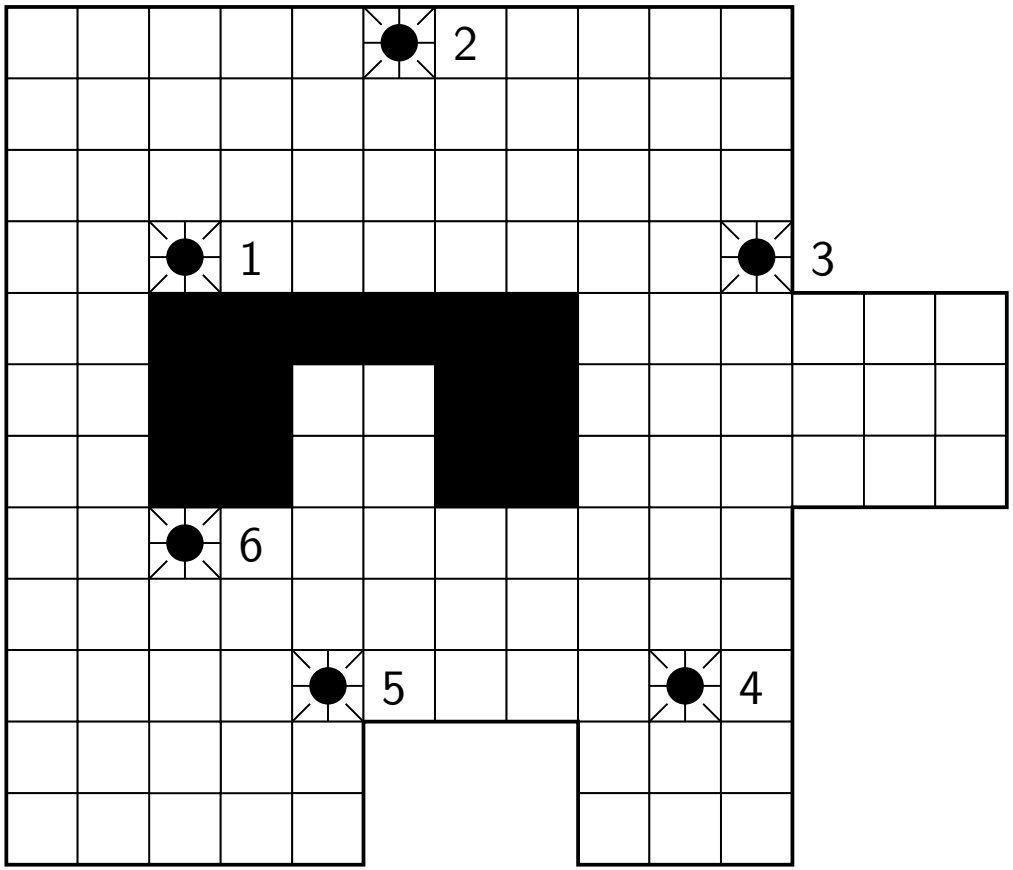
Perzeptrons in der *Grid World*

- `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

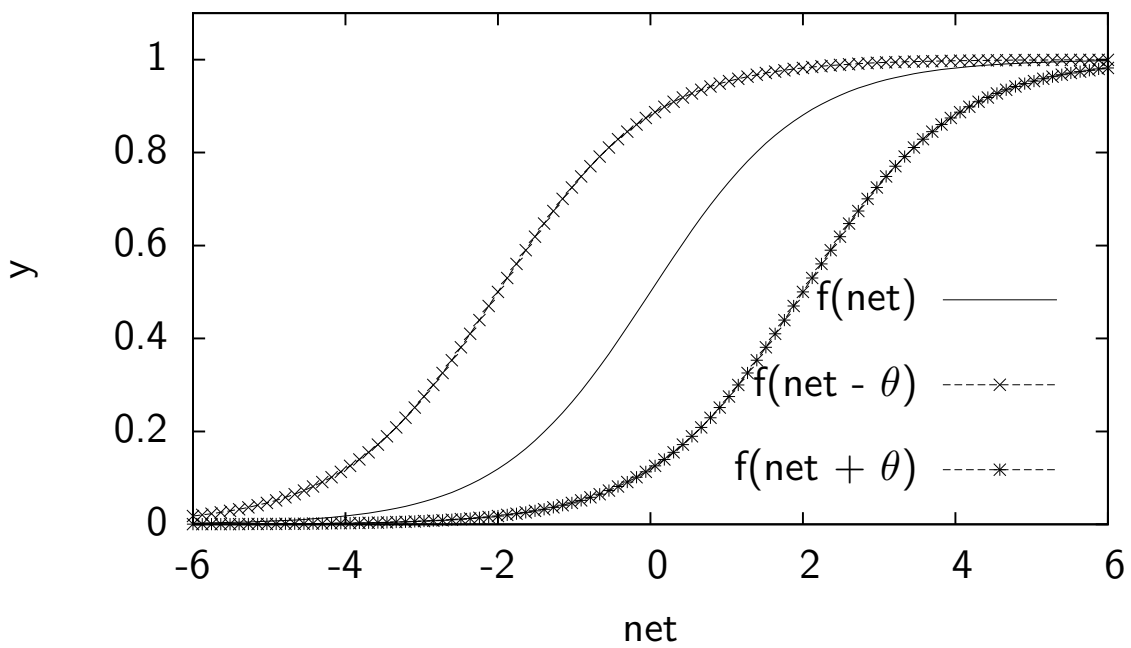
Nr.	Sensordaten	$x_1 \wedge \overline{x_2}$ (`move east`)
1	$(0,0,0,0,1,1,0,0)$	0
2	$(1,1,1,0,0,0,0,0)$	1
3	$(0,0,1,0,0,0,0,0)$	0
4	$(0,0,0,0,0,0,0,0)$	0
5	$(0,0,0,0,0,1,0,0)$	0
6	$(0,1,1,0,0,0,0,0)$	1

Verallgemeinerte Delta-Regel

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

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

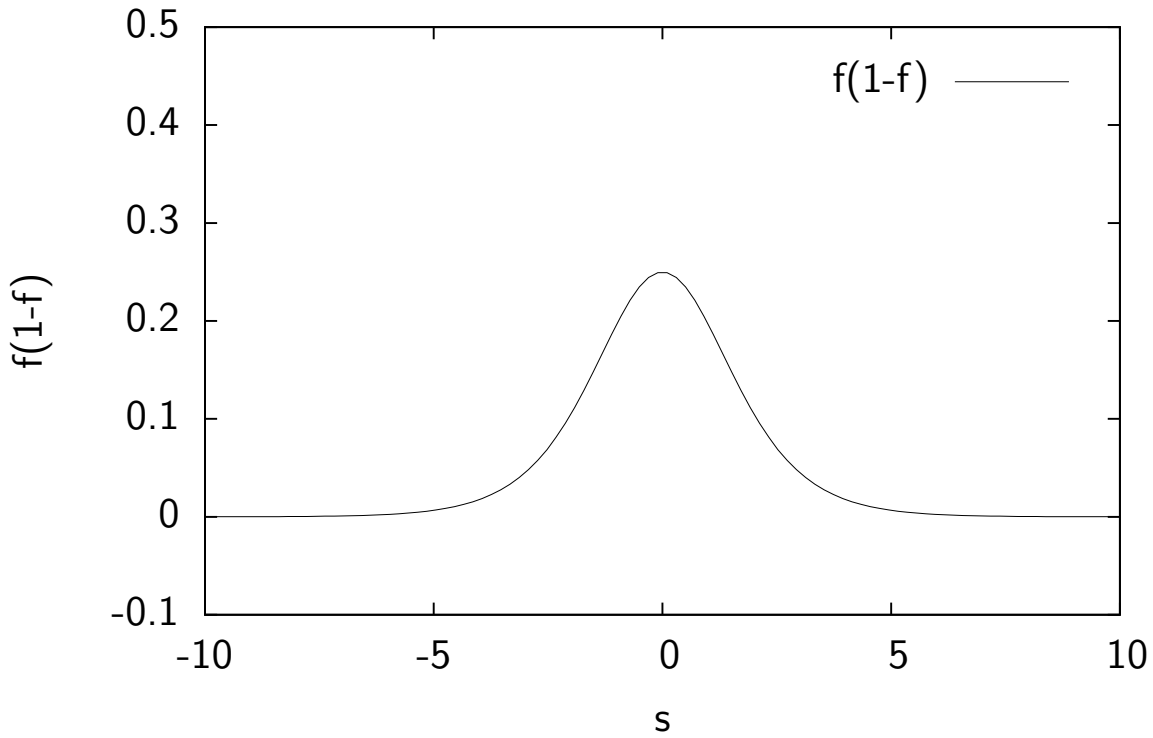
Einfluss des Bias-Wertes θ



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

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

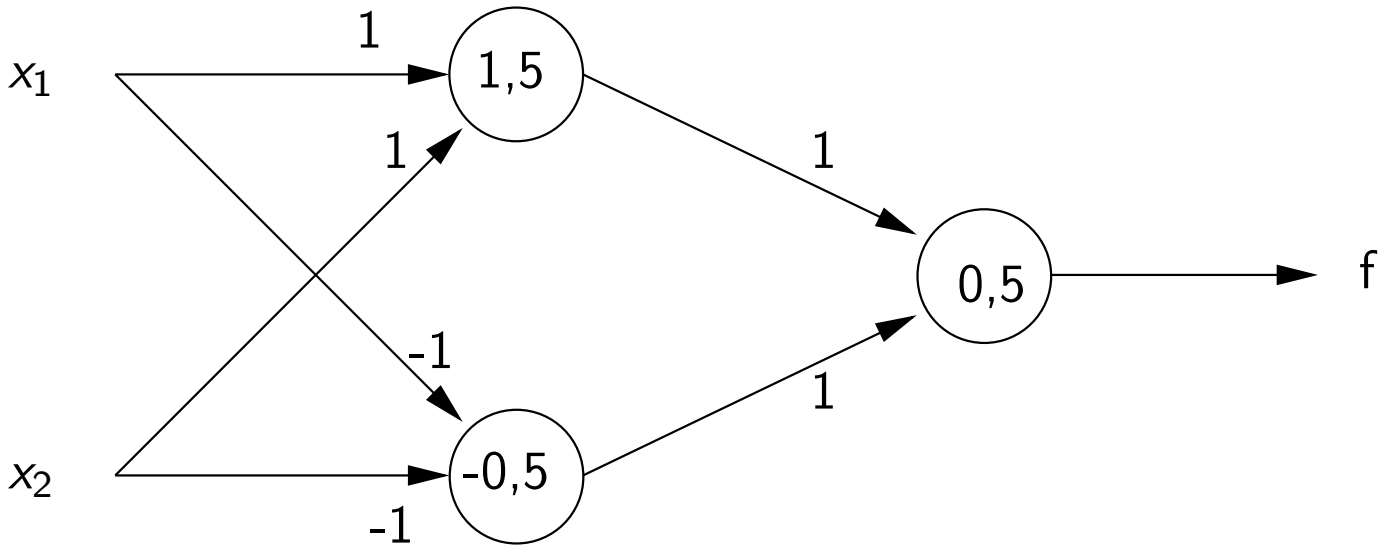
- „variable“ Lernrate η um Änderungsstärke einzustellen
- weitere Verbesserungen des Lernverfahrens sind möglich
- Lernen $\rightarrow$ Übungsaufgabe, Hinweis zur Tabelle:

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

Übersicht

Mehrschichtige Perzeptrons

- falls Problem nicht linear separabel, dann mehrschichtige NNs
- Beispiel: Funktion *gleiche Parität*

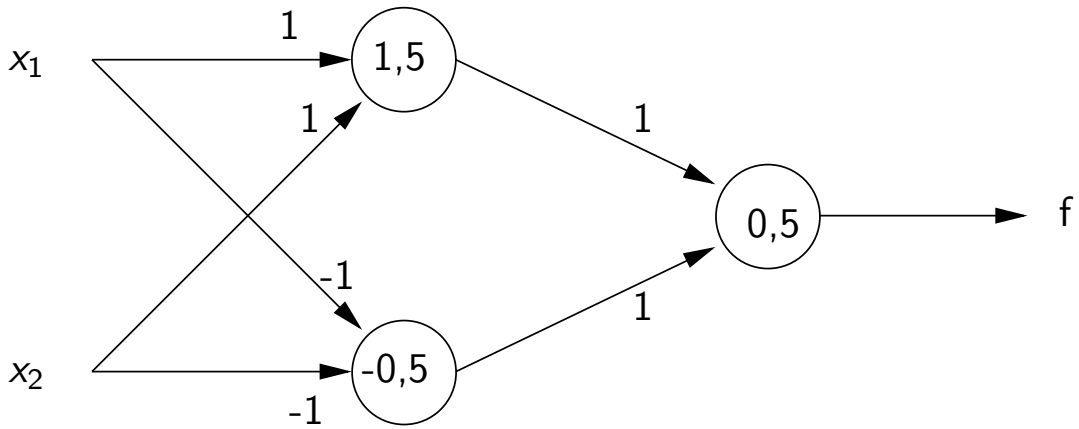


- zweischichtiges NN
- dreischichtiges NN falls Eingabeschicht mitzählt
- Neuronen in mittlerer Schicht: verborgene/innere Neuronen

Propagation am Beispiel

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

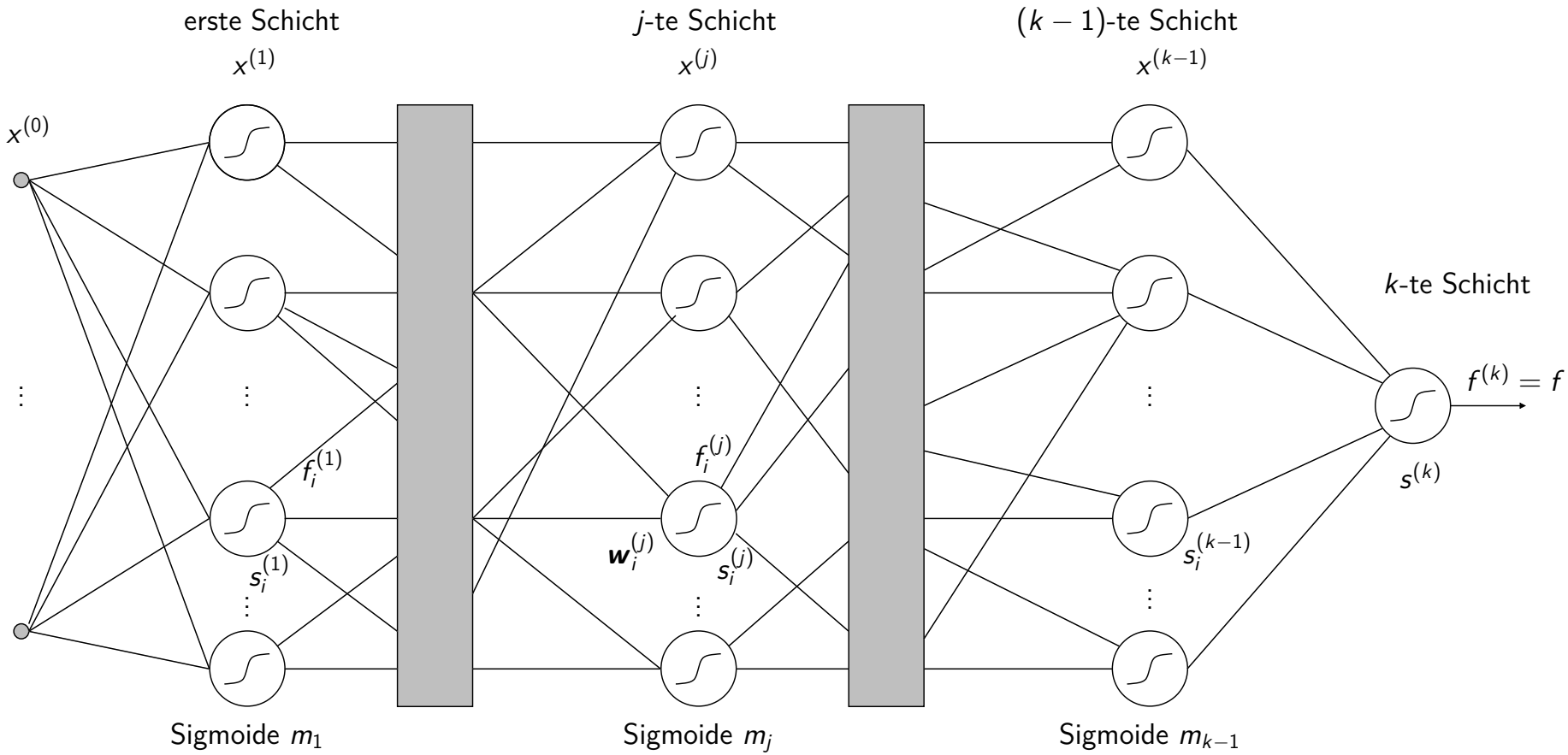
Eingabe		Neuron I		Neuron II		Neuron III		Ausgabe
x_1	x_2	Eingabe	Ausgabe	Eingabe	Ausgabe	Eingabe	Ausgabe	
0	0	0	0	0	1	1	1	1
0	1	1	0	-1	0	0	0	0
1	0	1	0	-1	0	0	0	0
1	1	2	1	-2	0	1	1	1



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

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

- somit rekursives Gleichungssystem zur Berechnung der δ_i^j

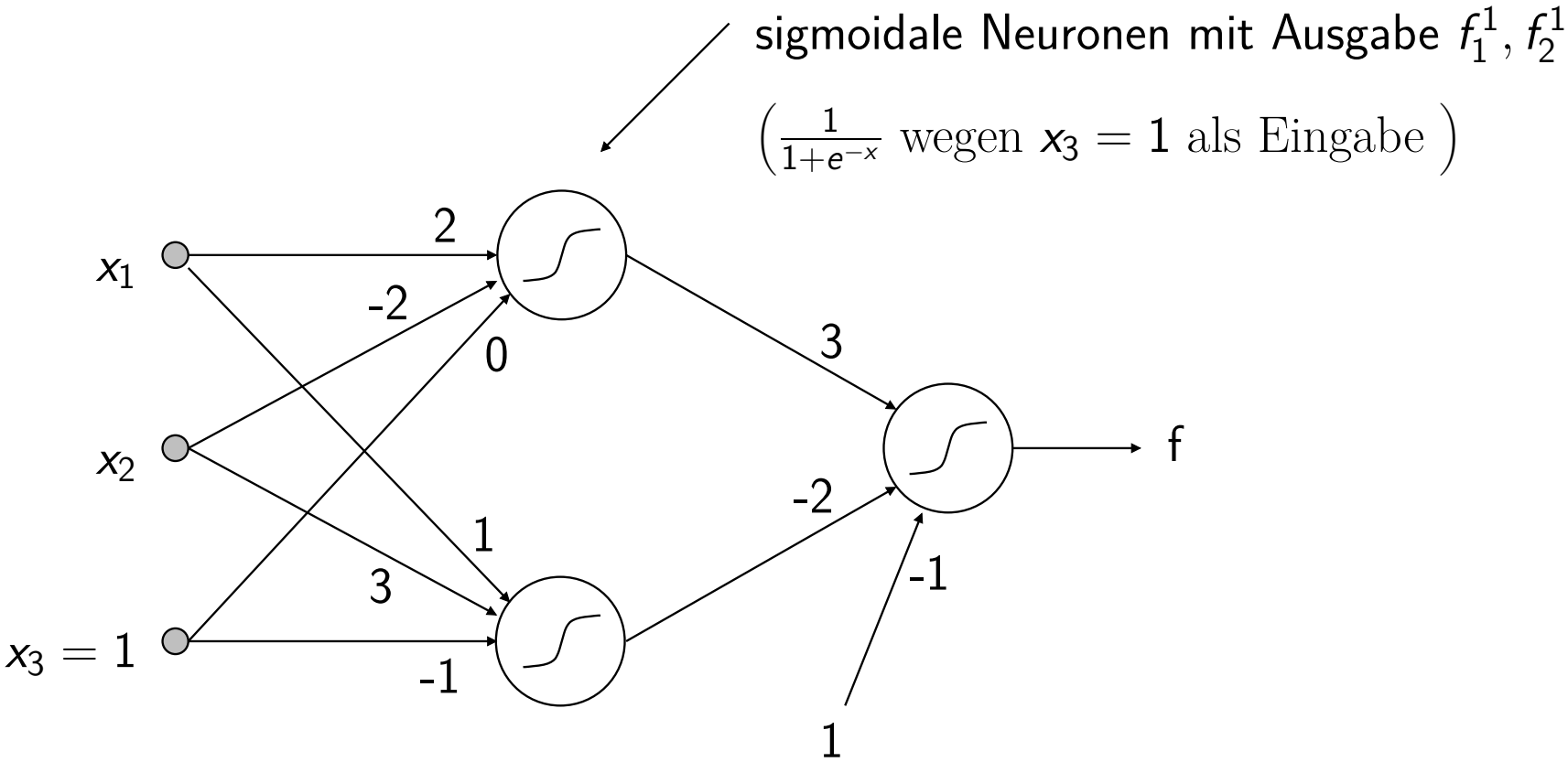
$$\delta_i^j = f_i^j (1 - f_i^j) \sum_{l=1}^{m_{j+1}} \delta_l^{j+1} \cdot w_{il}^{j+1} \text{ und } \delta^k = (d - f) f (1 - f)$$

- 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:

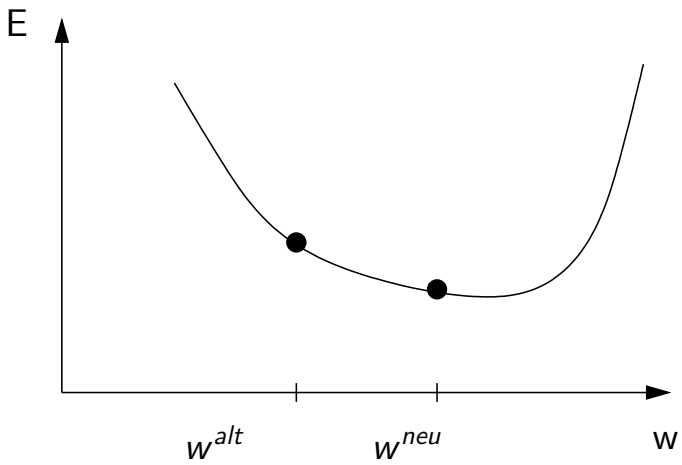
$$\delta^{(2)} = -0.148, \quad d_1^{(1)} = -0.047, \quad d_2^{(1)} = 0,074$$

- neue Gewichte mit $\eta = 1$:

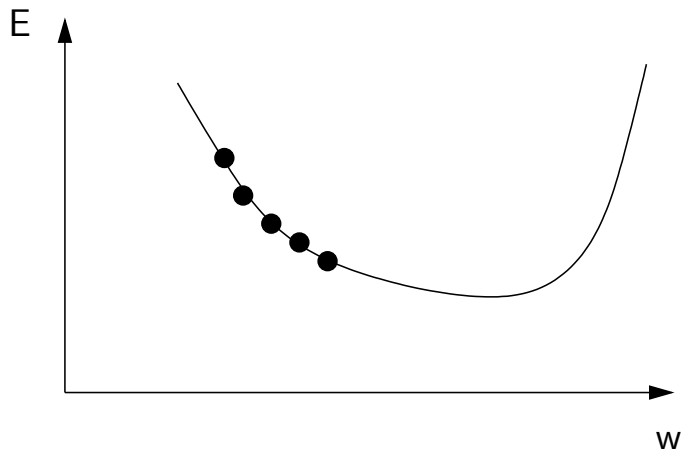
$$\boldsymbol{w}_1^{(1)} = (1.935, -2, -0.047)$$
$$\boldsymbol{w}_2^{(1)} = (1.074, 3, -0.926)$$
$$\boldsymbol{w}^{(2)} = (2.870, -2.074, -1.148)$$

Effekt der Lernrate

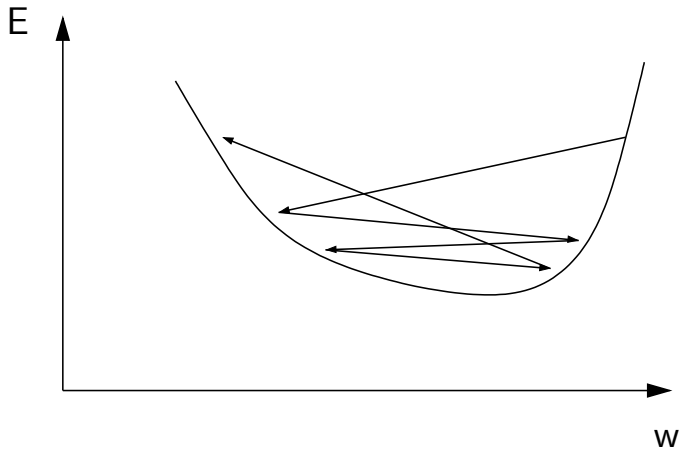
gewünschtes Verhalten:



zu klein: Lernen verläuft zu langsam

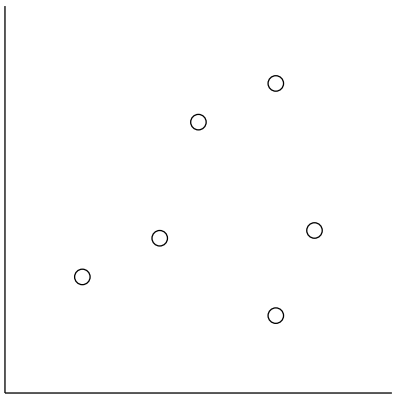


zu groß: Pingpong-Effekt

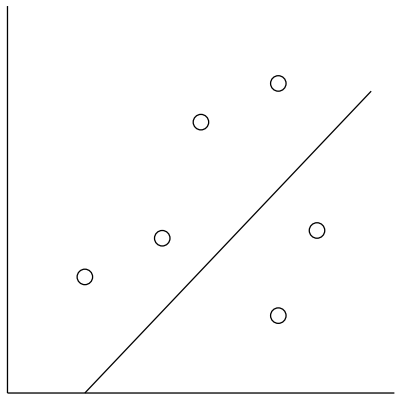


Eigenschaften von Neuronalen Netzen

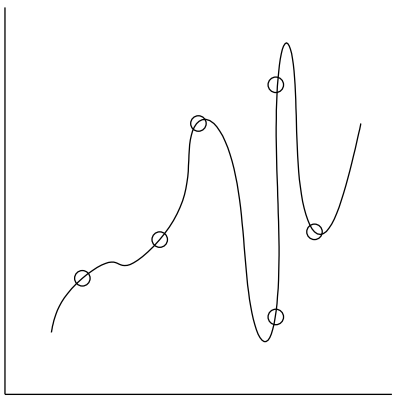
- NN können *generalisieren*, d.h. Vektoren klassifizieren, die $\notin L$
- möglich: Maß der Generalisierungsgüte
- Analogie: Funktionsapproximation



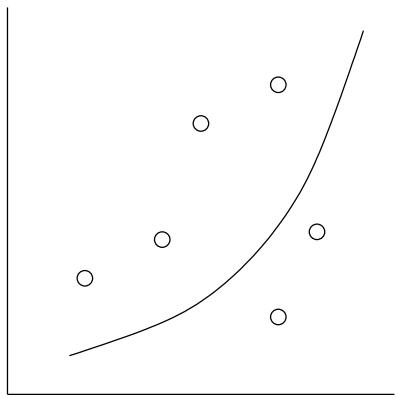
Datensatz



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*)
  1. Lernstichprobe in n disjunkte Mengen teilen
  2. jeweils $n-1$ Mengen fürs Training und 1 Menge für Validierung
  3. 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

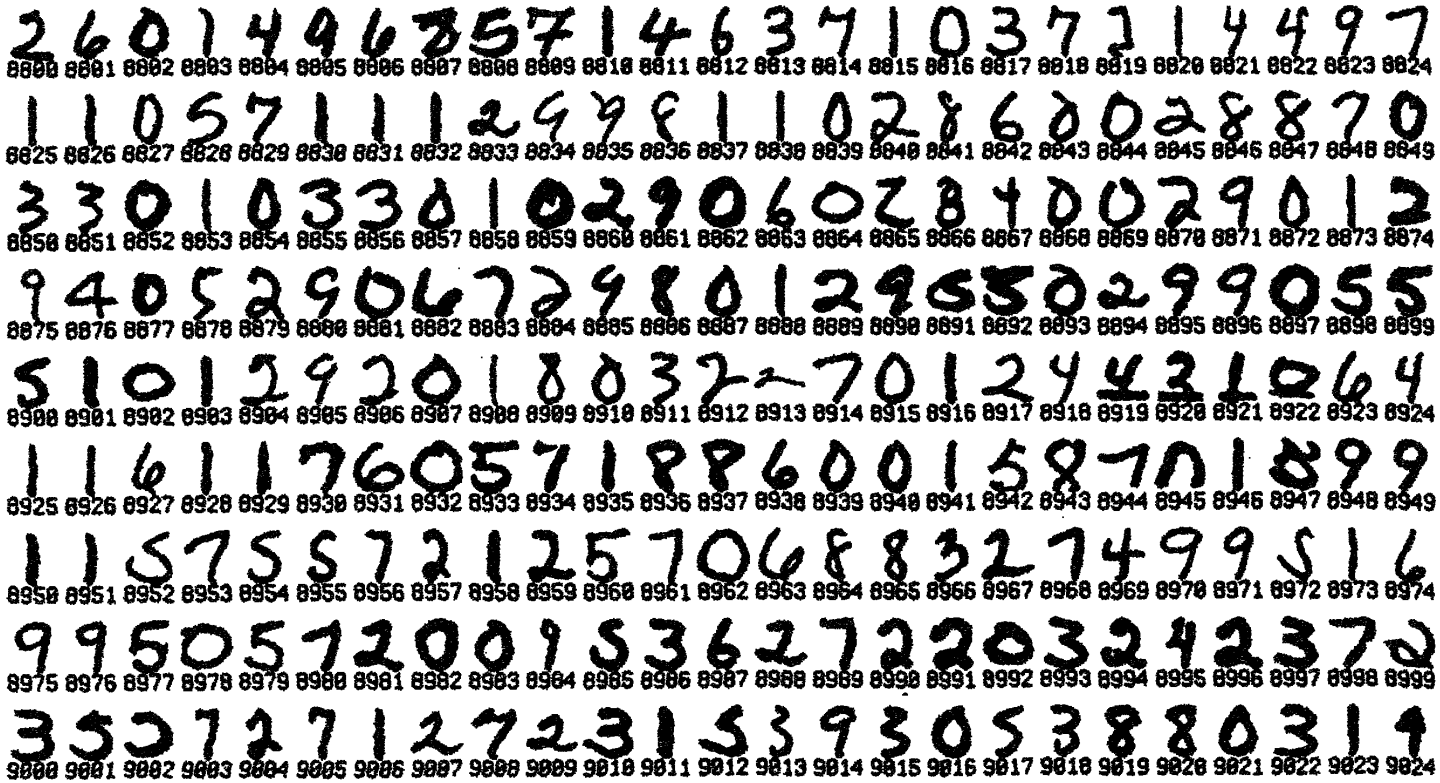


Abbildung: USPS-Datensatz, Originaldaten (segmentierte Ziffern) mit Labels, entnommen aus [Vapnik, 1998]

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

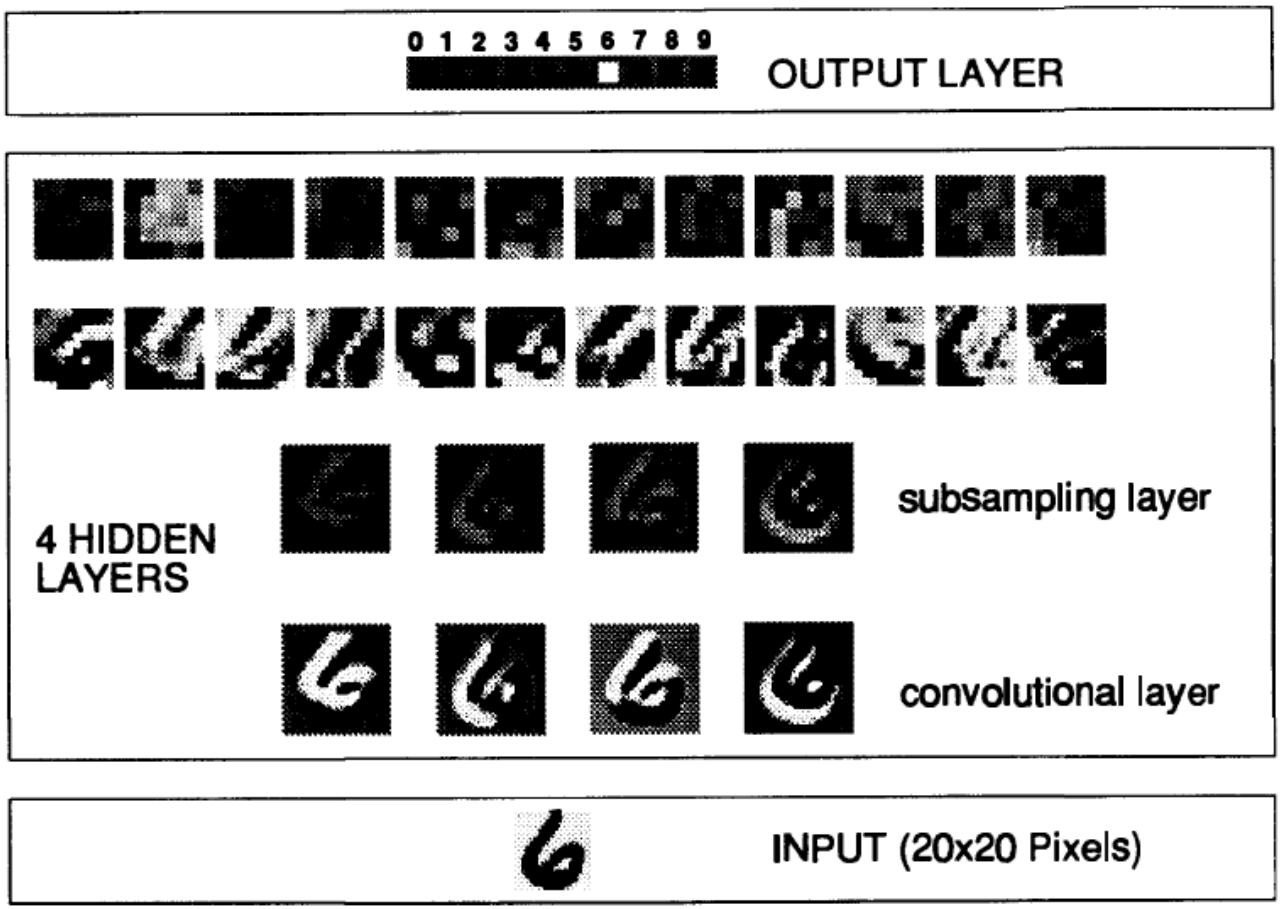


Abbildung: Ziffererkennung im USPS-Datensatz mit mehrschichtigem Perzeptron, nach [Matan et al., 1992]

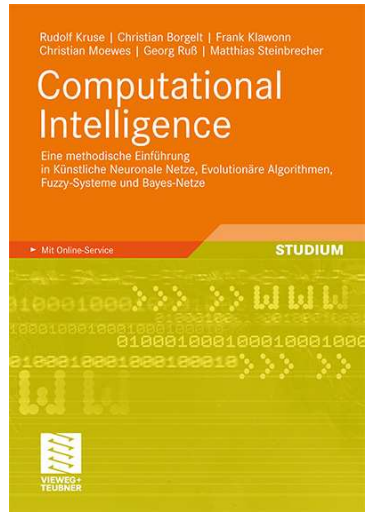
USPS-Datensatz, Erkennungsleistung

„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, Zahlen nach [Vapnik, 1998]

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., 2011]

Literatur zur Lehrveranstaltung

Kruse, R., Borgelt, C., Klawonn, F., Moewes, C., Ruß, G., and Steinbrecher, M. (2011).
Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze.
Vieweg+Teubner-Verlag, 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.

Vapnik, V. (1998).
Statistical Learning Theory.
John Wiley & Sons Ltd, New York, NY, USA.