

Intelligente Systeme

Neuronale Netze

Prof. Dr. R. Kruse

C. Moewes

G. Ruß

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

Institut für Wissens- und Sprachverarbeitung

Fakultät für Informatik

Otto-von-Guericke Universität Magdeburg

Übersicht

1. Einleitung

Natürliche (biologische) Neuronen

Konventionelle Rechner vs. Gehirn

Künstliche Neuronale Netze

2. Schwellenwert-Elemente

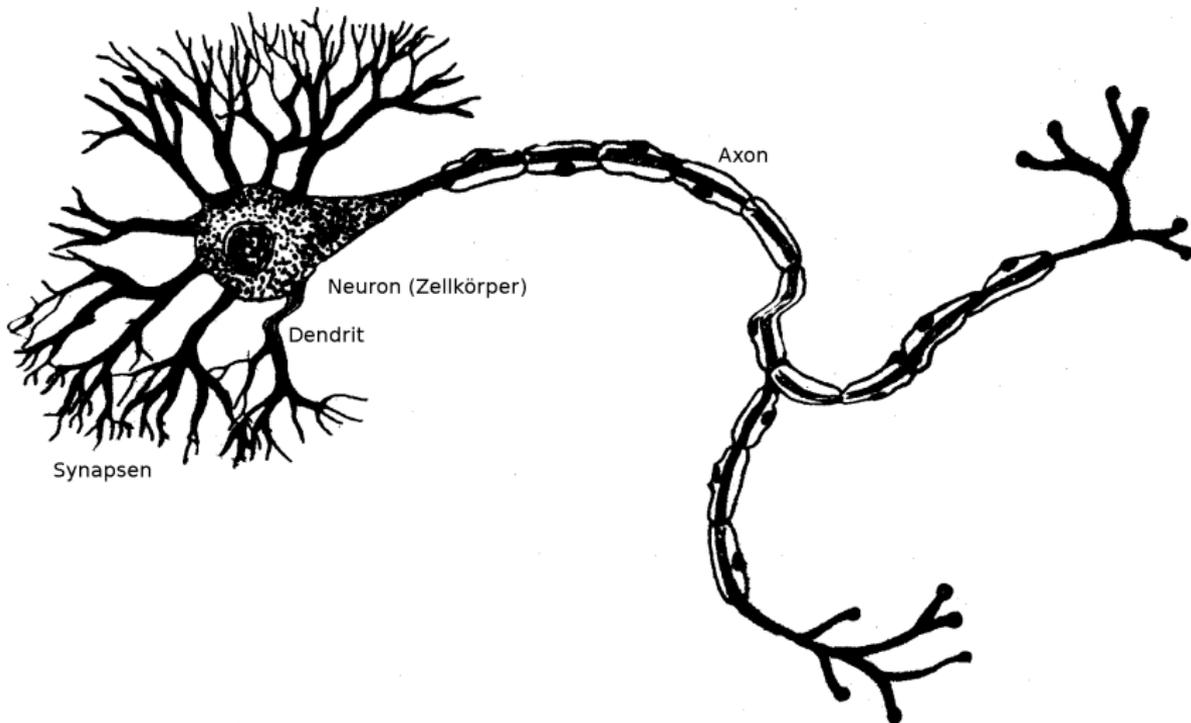
3. Delta-Regel

4. Mehrschichtige Perzeptrons

Neuronale Netze

- ▶ bisher KI von “oben”:
Modellierung eines intelligenten Agenten durch algorithmische Realisierung bestimmter Aspekte rationalen Handelns
- ▶ jetzt KI 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
Verarbeitungseinheiten	1 CPU, 10^9 Transistoren	10^{11} Neuronen
Speicherkapazität	10^9 Bytes RAM, 10^{10} Bytes Festspeicher	10^{11} Neuronen, 10^{14} Synapsen
Verarbeitungsgeschwindigkeit	10^{-8} s	10^{-3} s
Bandbreite	10^9 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 Berechnungselementen (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

1. Einleitung

2. Schwellenwert-Elemente

Perzeptron

Geometrische Interpretation

Biimplikationsproblem

Lernverfahren

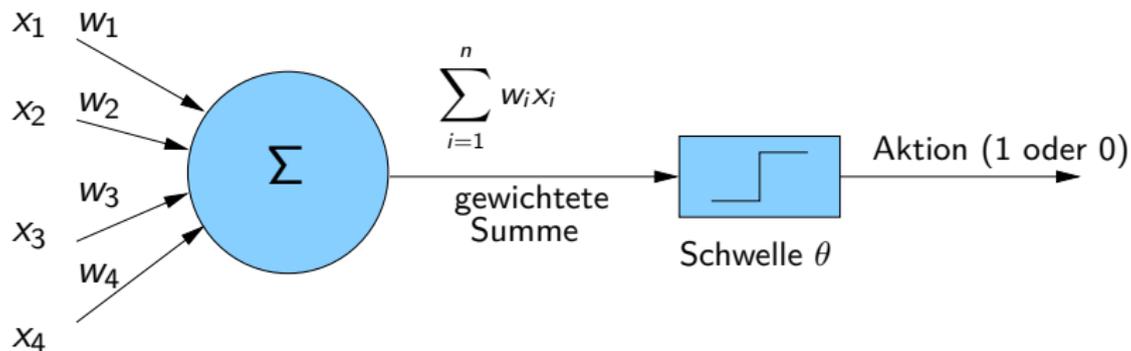
Gradientenabstiegsverfahren

3. Delta-Regel

Perzeptrons

- ▶ Implementation 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:

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

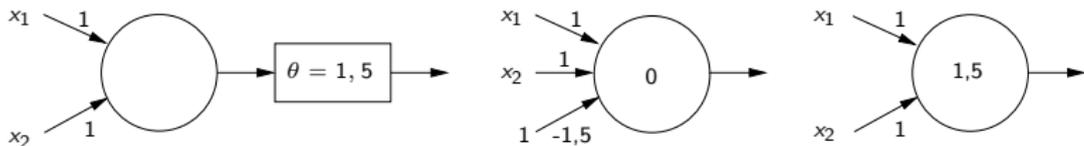
$$\mathbf{x} \stackrel{\text{def}}{=} (x_1, \dots, x_n)$$

Perzeptrons: Vereinfachte Schreibweise

- ▶ seien $x_{n+1} \stackrel{\text{def}}{=} 1$ und $w_{n+1} \stackrel{\text{def}}{=} -\theta$
- ▶ so sind $\mathbf{x} = (x_1, \dots, x_n, 1)$ und $\mathbf{w} = (w_1, \dots, w_n, -\theta)$
- ▶ vereinfachte Schreibweise:

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

- ▶ Beispiele äquivalenter Darstellungen:



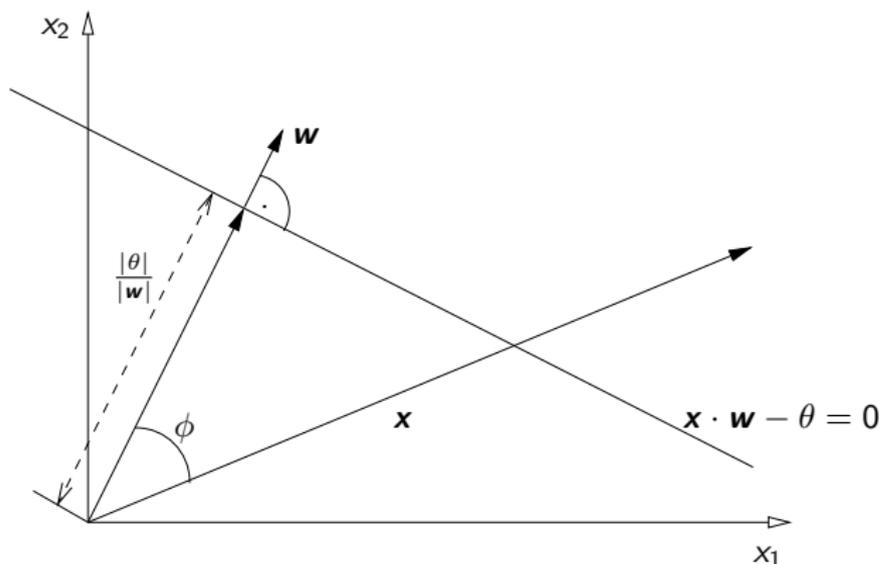
Geometrische Interpretation

- ▶ Gewichtsvektor $\mathbf{w} = (w_1, \dots, w_n)$
- ▶ Schwellenwert θ
- ▶ Eingabevektor $\mathbf{x} = (x_1, \dots, x_n)$
- ▶ Ausgabewert

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

- ▶ trennende Hyperebene $\mathbf{w} \cdot \mathbf{x} - \theta = 0$

Geometrische Interpretation

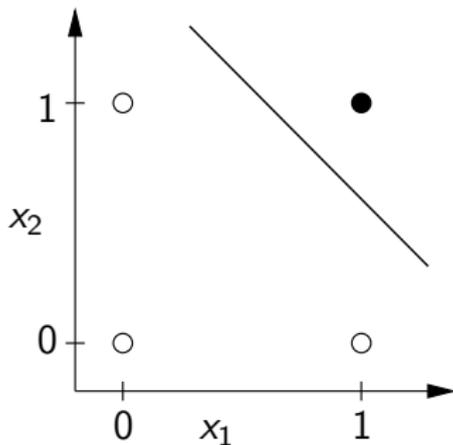


- ▶ θ ist negativ falls Ursprung auf Ebenenseite, in die w zeigt
- ▶ ansonsten ist θ positiv
- ▶ $w \cdot x - \theta > 0$ falls x auf Ebenenseite, in deren Richtung w zeigt

Geometrische Interpretation

- ▶ Perzeptron = Menge aller linear separablen Funktionen
- ▶ logisches AND ist linear separabel
- ▶ somit durch Perzeptron darstellbar

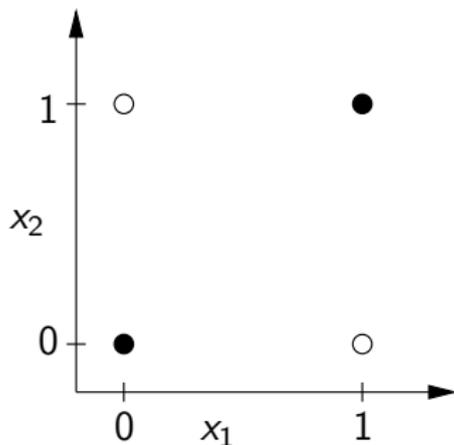
x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1



Biimplikationsproblem

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

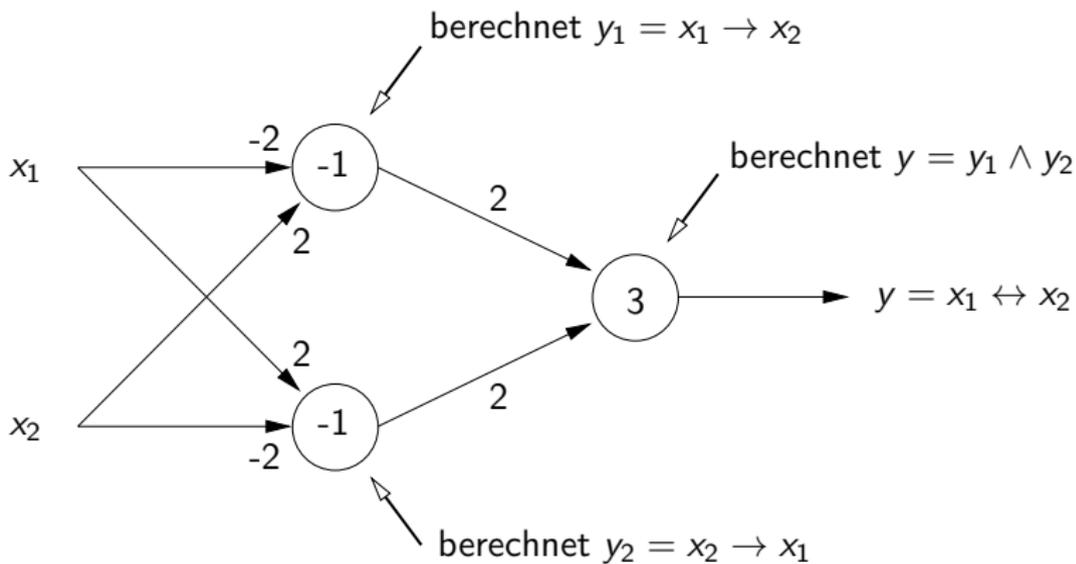
x_1	x_2	$x_1 \leftrightarrow x_2$
0	0	1
1	0	0
0	1	0
1	1	1



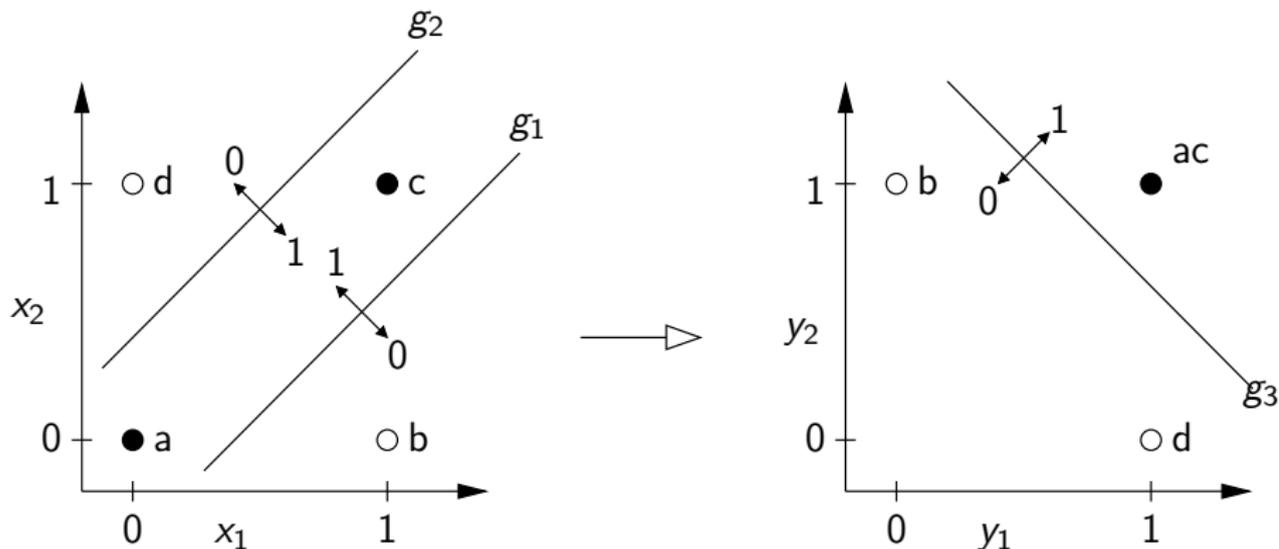
- ▶ es gibt keine Trenngerade, die Lösungsraum aufteilt

Biimplikationsproblem

- Lösung: Zusammenschalten mehrerer Schwellenwertelemente



Biimplikationsproblem



Geometrische Deutung des Zusammenschaltens mehrerer Schwellenwertelemente zur Berechnung der Biimplikation

Lernverfahren für Perzeptrons

Problemstellung:

- ▶ gegeben: Lernstichprobe $L = \{(\mathbf{x}_1, d_1), \dots, (\mathbf{x}_N, d_N)\}$
 - ▶ Merkmale $\mathbf{x}_i \in \{0, 1\}^n$ 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 $f : \{0, 1\}^n \rightarrow \{0, 1\}$, die zu L „passt“

Trainieren einzelner Perzeptrons

- ▶ im Training: \mathbf{w} und θ werden angepasst
- ▶ Ziel: Elemente \mathbf{x}_i aus L stimmen mit d_i und Perzeptron-Ausgabe

$$f_i = f_{(w_1, \dots, w_n, \theta)}(\mathbf{x}_i)$$

(möglichst) gut überein

- ▶ Lösung: Minimierung des quadratischen Fehlers
- ▶ also, Lernen = Optimierungsaufgabe

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

Gradientenabstiegsverfahren

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

Gradientenabstiegsverfahren

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

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

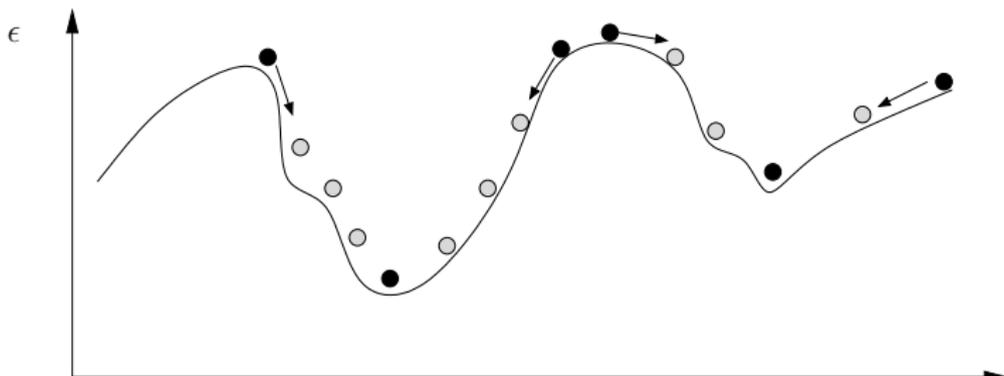
- ▶ anschließend: Berechnung neuer Gewichte

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

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

Gradientenabstieg – Anschauung

- ▶ zu jedem \mathbf{w} gehört ein Fehlerwert
- ▶ „Fehlergebirge“ über dem durch \mathbf{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

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

► Problem: ε nicht stetig differenzierbar wegen θ

Übersicht

1. Einleitung

2. Schwellenwert-Elemente

3. Delta-Regel

Algorithmus

Beispiel

Perzeptrons in der Grid World

Verallgemeinerte Delta-Regel

4. Mehrschichtige Perzeptrons

Delta-Regel

Fehlerkorrekturverfahren:

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

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

Algorithmus:

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

Delta-Regel – Algorithmus

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

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

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

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

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

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

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

- ▶ oft Abbruchbedingung: $\varepsilon <$ vorgegebene Schranke
- ▶ ein Durchgang heißt *Lernperiode*
- ▶ Algorithmus terminiert für linear separable Lernstichproben

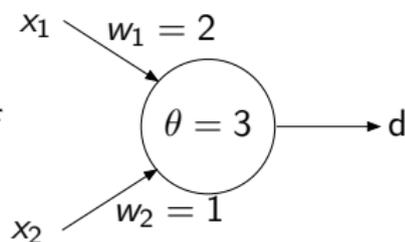
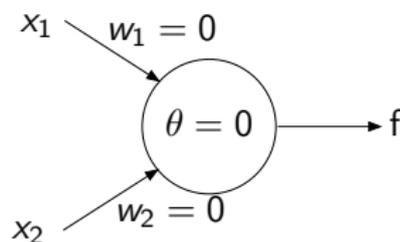
Delta-Regel – Beispiel AND

- Lernvorgang liefert

Initialisierung

Nach dem Lernen

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

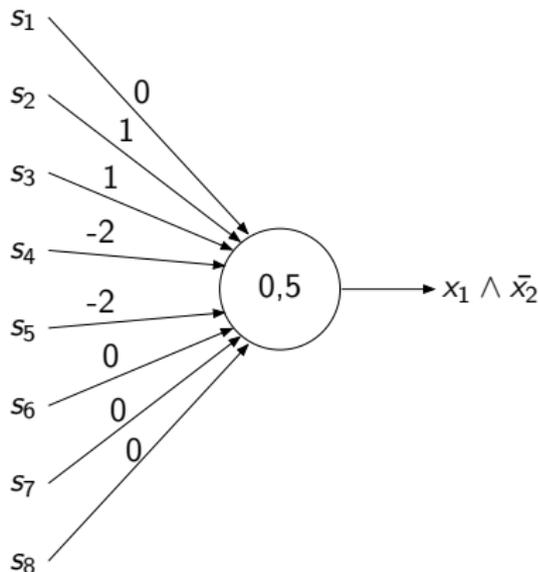


- Rechnung → Übung, Hinweis: Tabelle mit folgenden Werten

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

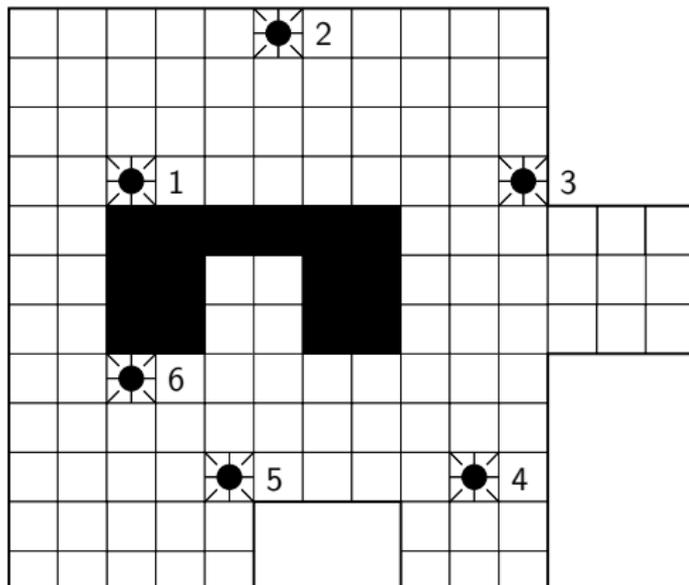
Perzeptrons in der *Grid World*

- ▶ move east $\leftarrow x_1 = 1 \wedge x_2 = 0 \leftarrow x_1 \wedge \bar{x}_2 = (s_2 \vee s_3) \wedge \bar{s}_4 \wedge \bar{s}_5$
- ▶ folgendes Perzeptron liefert $1 \iff$ move east



Lerndatensatz – „nach Osten gehen“

- ▶ möglich: Gewichte von Beispielen lernen
- ▶ hier: nicht lohnenswert, da Funktion sehr einfach
- ▶ Aufgabe: suche x für die 6 markierten Positionen



Lerndatensatz – „nach Osten gehen“

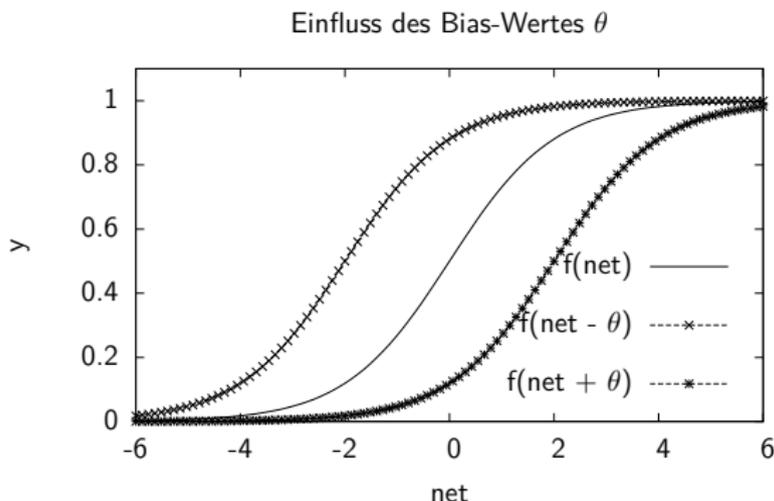
- hier L sieht wie folgt aus

Nr.	Sensordaten	$x_1 \wedge \bar{x}_2$ (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

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



Ableitung der logistischen Funktion

$$\frac{\partial f(s)}{\partial s} = \frac{\partial}{\partial s} \frac{1}{1+e^{-s}} = \frac{\partial}{\partial s} (1 + e^{-s})^{-1} = \overbrace{-(1 + e^{-s})^{-2}}^{\text{äußere Ableitung}} \cdot \overbrace{(-e^{-s})}^{\text{innere Ableitung}}$$

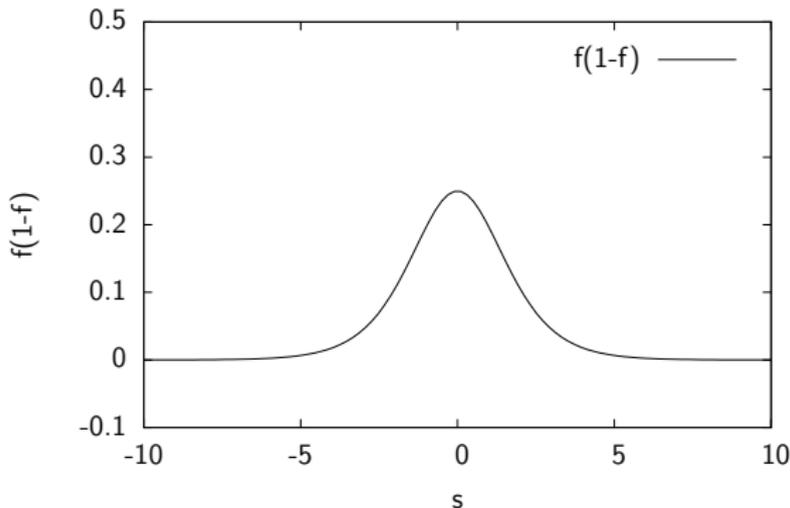
$$= \frac{e^{-s}}{(1+e^{-s})^2} = \frac{1+e^{-s}-1}{(1+e^{-s})^2}$$

$$= \frac{1+e^{-s}}{(1+e^{-s})^2} - \frac{1}{(1+e^{-s})^2}$$

$$= f - f^2 = f(1 - f)$$

$$\left. \frac{\partial f(s)}{\partial s} \right|_{s=0} = \frac{1}{1+1} \left(1 - \frac{1}{1+1} \right) = \frac{1}{4}$$

Graph von $f(1-f)$



Wirkung:

- ▶ nahe an der Hyperebene durch \mathbf{w} ist Änderung stark
- ▶ weiter davon entfernt ist sie gering (bei gleichem Fehler)

Verallgemeinerte Delta-Regel

- ▶ Fehlergradienten somit

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

- ▶ wie bei alter Delta-Regel, neue Schätzung für \mathbf{w}

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

- ▶ „variable“ Lernrate η um Änderungstärke einzustellen
- ▶ weitere Verbesserungen des Lernverfahrens sind möglich
- ▶ Lernen → Übungsaufgabe, Hinweis zur Tabelle:

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

Übersicht

1. Einleitung

2. Schwellenwert-Elemente

3. Delta-Regel

4. Mehrschichtige Perzeptrons

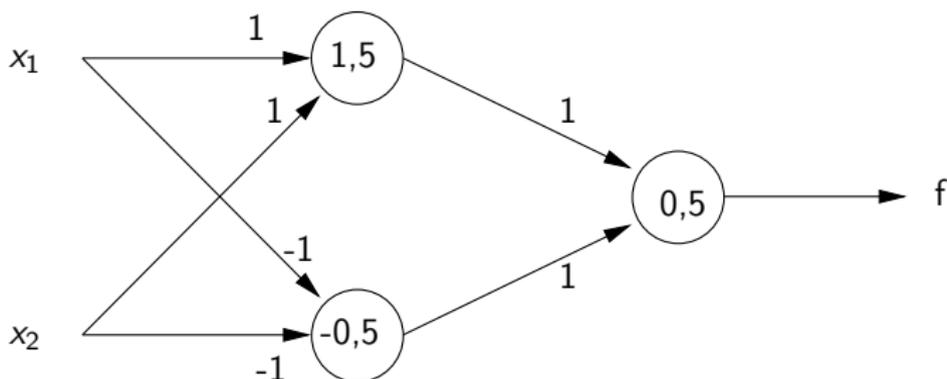
Backpropagation

Beispiel

Bemerkungen

Mehrschichtige Perzeptrons

- ▶ bei nicht separabler Funktionen: mehrschichtige NNs
- ▶ Beispiel: Funktion *gleiche Parität*

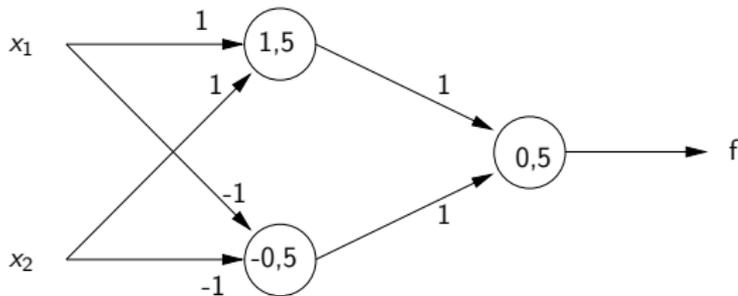


- ▶ zweischichtiges NN
- ▶ dreischichtiges NN falls Eingabeschicht mitzählt
- ▶ Neuronen in mittlerer Schicht: verborgene/innere Neuronen

Propagation am Beispiel

$$f(x_1, x_2) = (x_1 \wedge x_2) \vee (\bar{x}_1 \wedge \bar{x}_2)$$

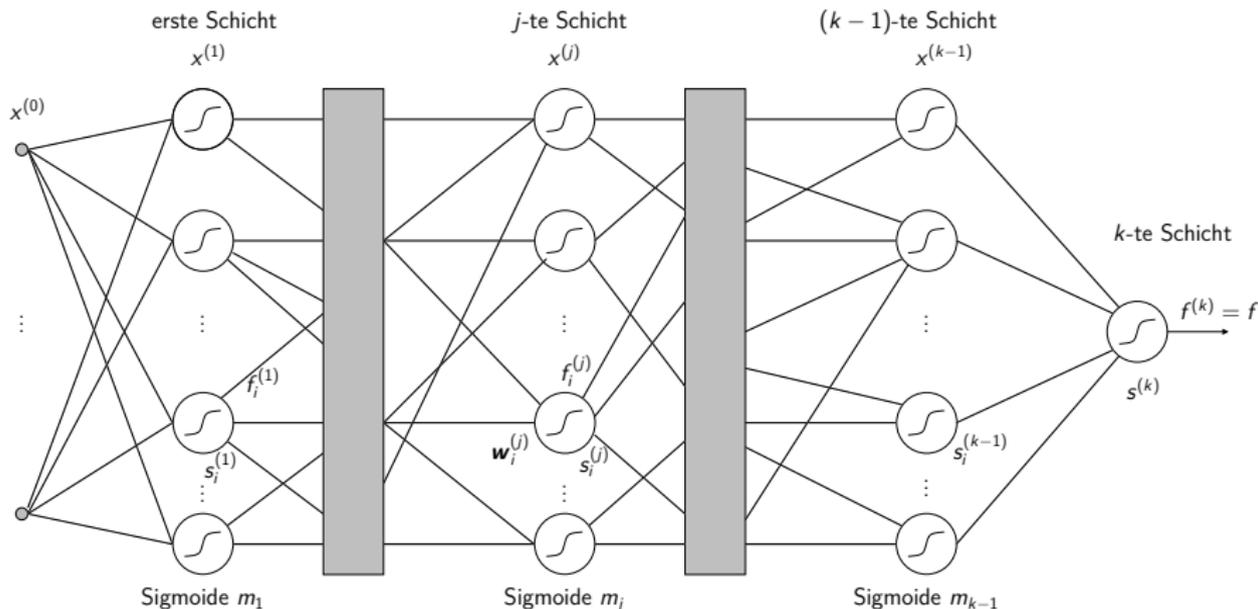
Eingabe		Neuron I		Neuron II		Neuron III		Ausgabe
x_1	x_2	Eingabe	Ausgabe	Eingabe	Ausgabe	Eingabe	Ausgabe	
0	0	0	0	0	1	1	1	1
0	1	1	0	-1	0	0	0	0
1	0	1	0	-1	0	0	0	0
1	1	2	1	-2	0	1	1	1



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
- ▶ \mathbf{w}_i^j ist Gewichtsvektor des i -ten Sigmoids in Schicht j , wobei letzte Komponente jeweils θ
- ▶ *Aktivierung* des Neurons = Summe $s_i^j = \mathbf{x}^{j-1} \cdot \mathbf{w}_i^j$

Mehrschichtige Perzeptrons



Backpropagation

- ▶ spezielles Trainingsverfahren
- ▶ Fehler wird durch Netz schichtweise zurückgeschickt
- ▶ *Idee*: analog zu Gradientenabstieg, berechne Fehlergradient
- ▶ Herleitung
 - ▶ Fehler für Ausgangsschicht $\varepsilon = (d - f)^2$
 - ▶ Gradient:

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

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

Backpropagation – Herleitung

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

- ▶ Berechnung neuer Gewichte $(\mathbf{w}_i^j)^{(\text{neu})} \leftarrow (\mathbf{w}_i^j)^{(\text{alt})} + \eta_i^j \cdot \delta_i^j \mathbf{x}^{j-1}$
- ▶ Lernrate η_i^j meistens ein Wert für gesamtes Netz

Backpropagation – Herleitung

Gewichtsänderungen in der Ausgabeschicht:

$$\delta^k = (d - f) \frac{\partial f}{\partial s^k} = (d - f)f(1 - f)$$
$$\mathbf{w}^{k,\text{neu}} \leftarrow \mathbf{w}^{k,\text{alt}} + \eta^k (d - f)f(1 - f)\mathbf{x}^{k-1}$$

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:

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

(γ ist Index über einzelnen Vektorelemente)

Backpropagation – Herleitung

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

- ▶ somit rekursives Gleichungssystem zur Berechnung der δ_i^j

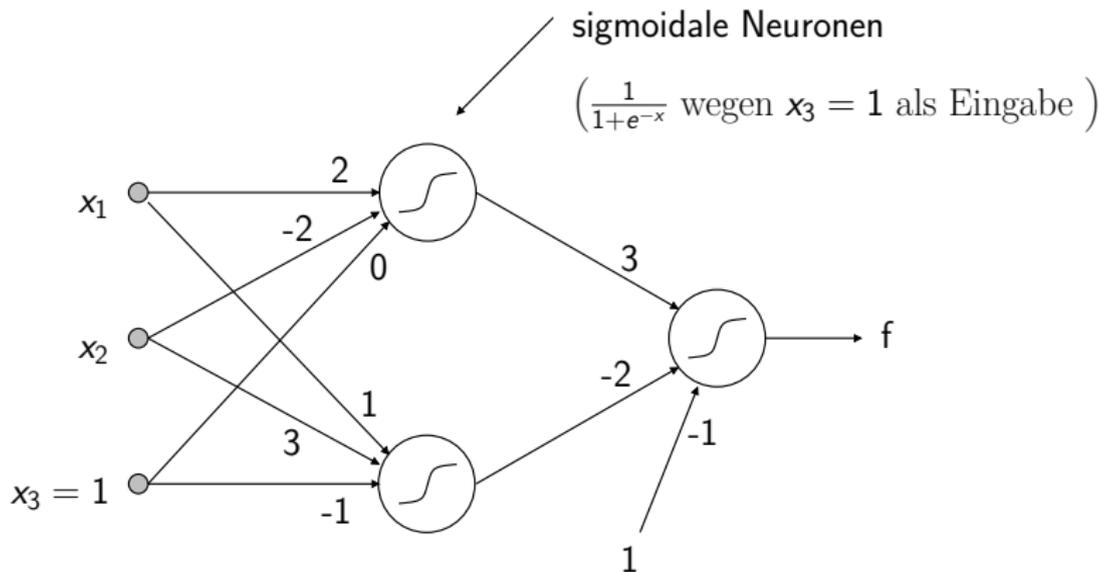
$$\delta_i^j = f_i^j (1 - f_i^j) \sum_{l=1}^{m_{j+1}} \delta_l^{j+1} \cdot w_{il}^{j+1} \quad \text{und } \delta^k = (d - f) f (1 - f)$$

- ▶ schließlich: Berechnung der neuen Gewichte

$$\mathbf{w}_i^{j(\text{neu})} \leftarrow \mathbf{w}_i^{j(\text{alt})} + c_i^j \cdot \delta_i^j - \mathbf{x}^{j-1}$$

Backpropagation – Beispiel

Netz mit zufällig generierten Gewichten:



Backpropagation – Beispiel

- ▶ Lernstichprobe:

$$\{((1, 0, 1), 0), ((0, 1, 1), 0), ((0, 0, 1), 1), ((1, 1, 1), 1)\}$$

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

- ▶ Backpropagation — Fehlersignale:

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

- ▶ neue Gewichte mit $\eta = 1$:

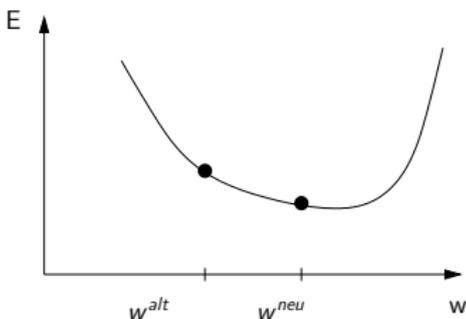
$$\mathbf{w}_1^{(1)} = (1.935, -2, -0.047)$$

$$\mathbf{w}_2^{(1)} = (1.074, 3, -0.926)$$

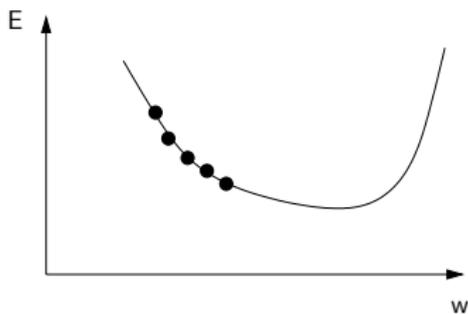
$$\mathbf{w}^{(2)} = (2.870, -2.074, -1.148)$$

Effekt der Lernrate

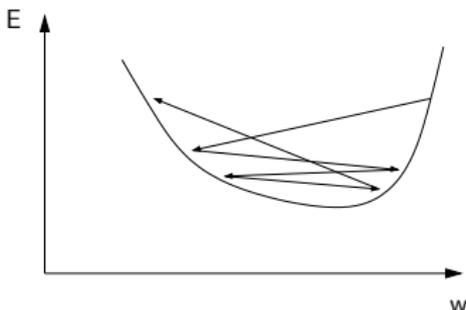
gewünschtes Verhalten:



zu klein: Lernen verläuft zu langsam

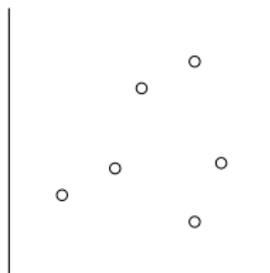


zu groß: Pingpong-Effekt

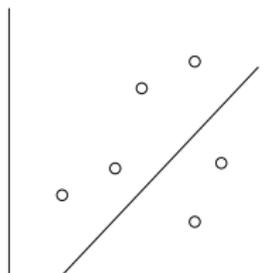


Eigenschaften von Neuronalen Netzen

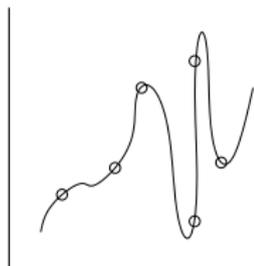
- ▶ NN können *generalisieren*, d.h. Vektoren klassifizieren, die $\notin L$
- ▶ möglich: Maß der Generalisierungsgüte
- ▶ Analogie: Funktionsapproximation



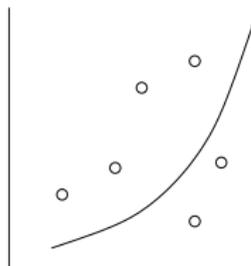
Datensatz



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

2601496857146371037214497
0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 0020 0021 0022 0023 0024
 1105711129981102860028870
0025 0026 0027 0028 0029 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039 0040 0041 0042 0043 0044 0045 0046 0047 0048 0049
 3301033010290602840029012
0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 0060 0061 0062 0063 0064 0065 0066 0067 0068 0069 0070 0071 0072 0073 0074
 9405290672980129550299055
0075 0076 0077 0078 0079 0080 0081 0082 0083 0084 0085 0086 0087 0088 0089 0090 0091 0092 0093 0094 0095 0096 0097 0098 0099
 5101292018032-70124431064
0100 0101 0102 0103 0104 0105 0106 0107 0108 0109 0110 0111 0112 0113 0114 0115 0116 0117 0118 0119 0120 0121 0122 0123 0124
 1161176057188600158701899
0125 0126 0127 0128 0129 0130 0131 0132 0133 0134 0135 0136 0137 0138 0139 0140 0141 0142 0143 0144 0145 0146 0147 0148 0149
 1157557212570688327499516
0150 0151 0152 0153 0154 0155 0156 0157 0158 0159 0160 0161 0162 0163 0164 0165 0166 0167 0168 0169 0170 0171 0172 0173 0174
 9950572009536272203242370
0175 0176 0177 0178 0179 0180 0181 0182 0183 0184 0185 0186 0187 0188 0189 0190 0191 0192 0193 0194 0195 0196 0197 0198 0199
 3507271272315393053880319
0200 0201 0202 0203 0204 0205 0206 0207 0208 0209 0210 0211 0212 0213 0214 0215 0216 0217 0218 0219 0220 0221 0222 0223 0224

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

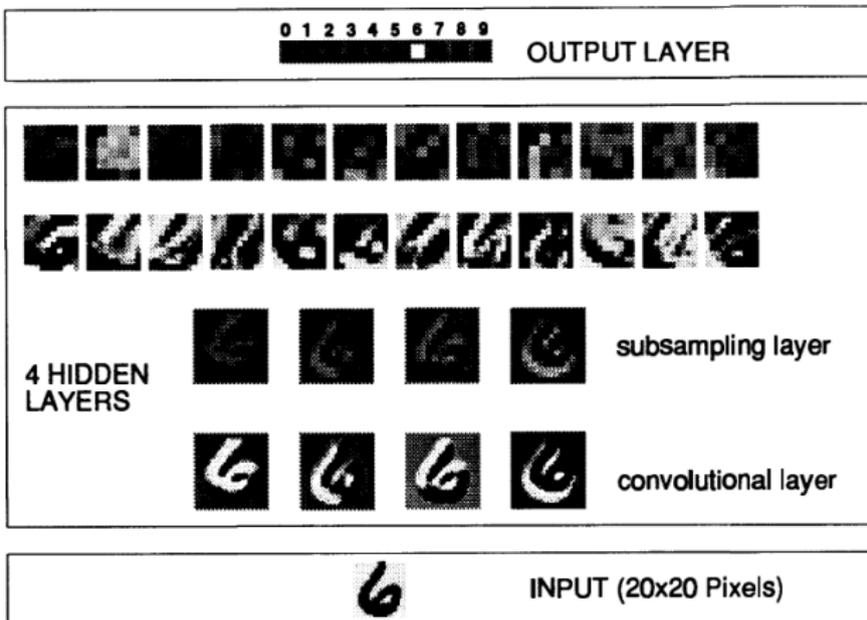


Abbildung: Ziffererkennung im USPS-Datensatz mit mehrschichtigem Perzeptron, nach [Matan et al., 1992]

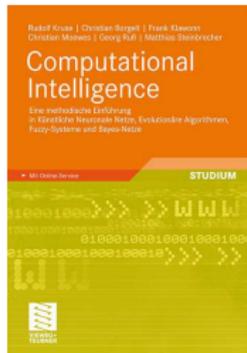
USPS-Datensatz, Erkennungsleistung

“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.