
3. Neuronale Netze

Prof. Dr. Rudolf Kruse

University of Magdeburg

Faculty of Computer Science

Magdeburg, Germany

rudolf.kruse@cs.uni-magdeburg.de

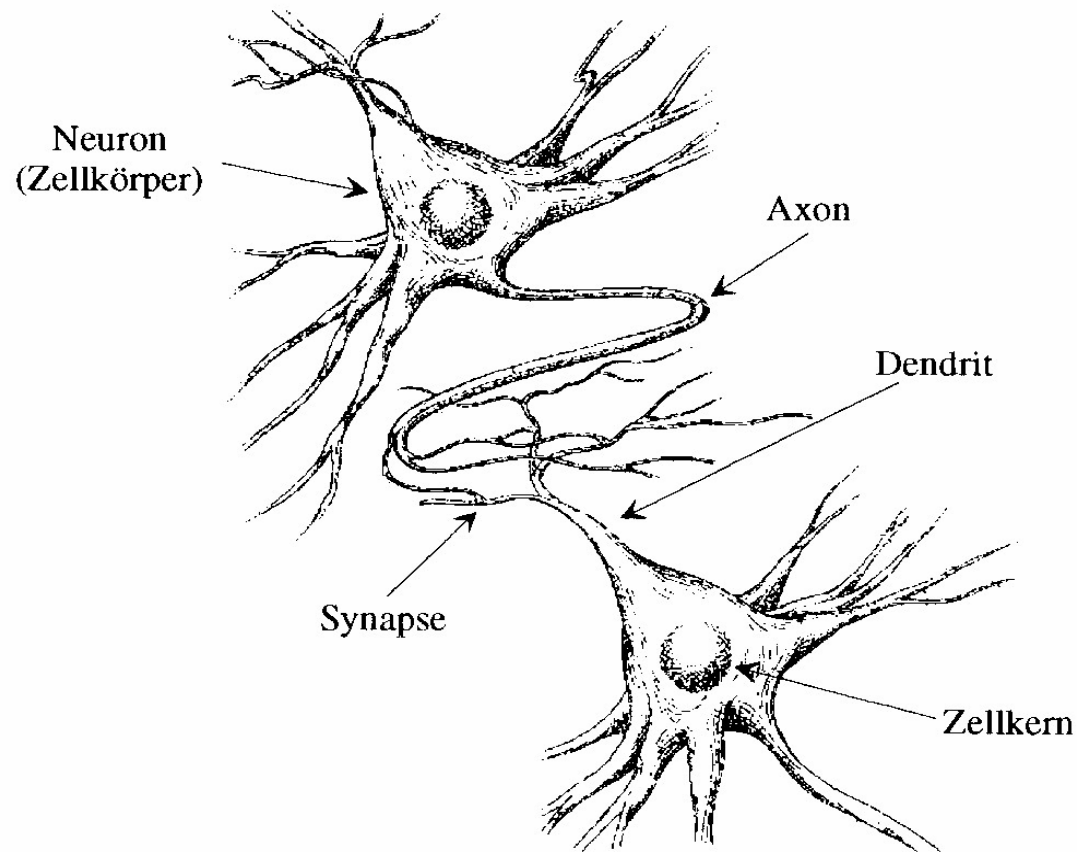


Neuronale Netze

Bisher: **KI von „oben“**: Modellierung eines intelligenten Agenten durch algorithmische Realisierung von bestimmten Aspekten rationalen Handelns.

Jetzt: **KI von „unten“**: Nachbildung der Struktur und der Verarbeitungsmechanismen des Gehirns (grob):
viele Prozessoren (Neuronen) und Verbindungen (Synapsen), die parallel und lokal Informationen verarbeiten.

Natürliche (biologische) Neuronen



Schematische Darstellung zweier Neuronen

Konventionelle Rechner vs. Gehirn

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates	10^{11} neurons
Storage units	10^9 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec.	10^{-3} sec.
Bandwith	10^9 bits/sec	10^{14} bits/sec
Neuro updates/sec.	10^5	10^{14}

Konventionelle Rechner vs. Gehirn

Beachte: **Hirnschaltzeit langsam (10^{-3} s), aber Updates erfolgen parallel. Dagegen braucht die serielle Simulation auf einem Rechner mehrere hundert Zyklen für 1 Update.**

Vorteile natürlicher neuronaler Netze:

- **Hohe Verarbeitungsgeschwindigkeit durch massive Parallelität.**
- **Funktionstüchtig selbst bei Ausfall von Teilen des Netzes (Fehlertoleranz).**
- **Langsamer Funktionsausfall bei fortschreitenden Ausfällen von Neuronen. (Graceful Degradation).**
- **Gut geeignet für induktives Lernen.**

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

Neuronale Netze

Vom mathematischen Standpunkt:

■ **Neuronale Netze als Methode, Funktionen zu repräsentieren:**

- **durch Netzwerke von einfachen Berechnungselementen (vergleichbar mit logischen Schaltkreisen),**
- **die aus Beispielen gelernt werden können.**
- *(Sichtweise in dieser Vorlesung!)*

Vom biologischen Standpunkt:

■ **Neuronale Netze als stark vereinfachtes Modell des Gehirns und seiner Funktionsweise:**

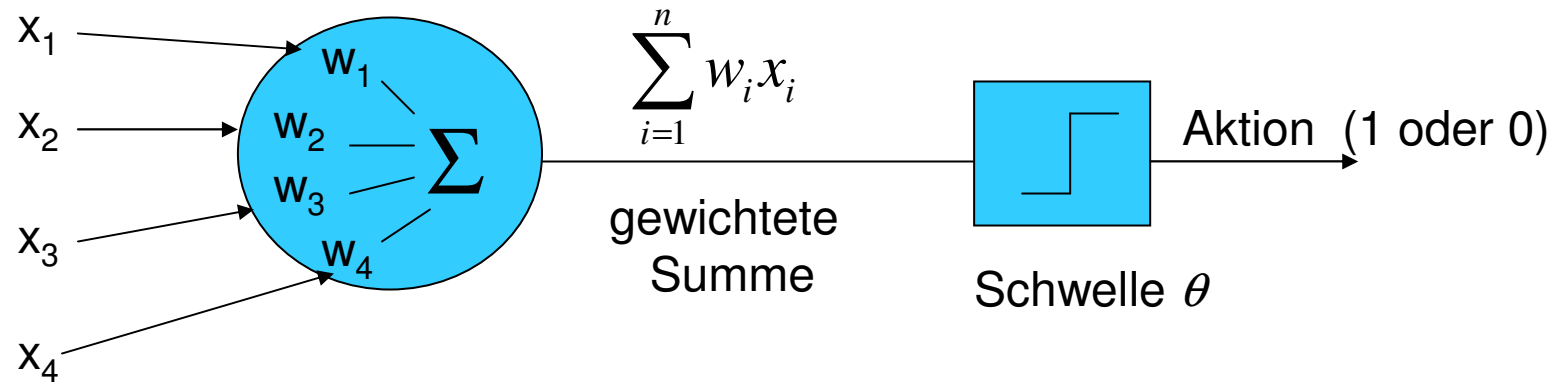
- **Konnektionismus**
- *(Nicht in dieser Vorlesung!)*

3.1 Perzeptrons

Perzeptrons (Schwellenwert-Elemente)

- **Es gibt eine ganze Reihe verschiedener Möglichkeiten S-R-Agenten zu implementieren (letztendlich repräsentiert ein solcher Agent eine Funktion)**
- **Im Folgenden betrachten wir für diese Zwecke ein Perzeptron („threshold logic unit“, TLU).**

Ein Perzeptron



Formale Definition:

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

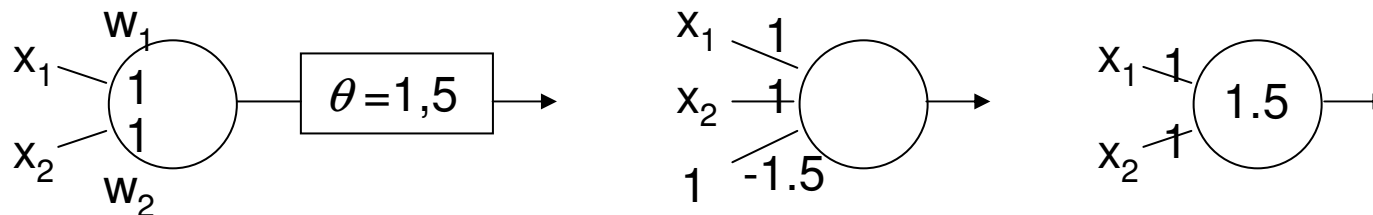
$$\vec{x} \rightarrow f_{w_1, \dots, w_n, \theta}(\vec{x})$$

Perzeptrons: Vereinfachte Darstellung

- Setzt man $x_{n+1} := 1$ und $w_{n+1} = -\theta$, so erhält man mit $\vec{x} = (x_1, \dots, x_n, 1)$ und $\vec{w} = (w_1, \dots, w_n, -\theta)$ die besonders einfache Darstellung:

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

- Beispiel (äquivalente Darstellungen):



Geometrische Interpretation

■ **Gewichtsvektor:** $\vec{w} = (w_1, \dots, w_n)$

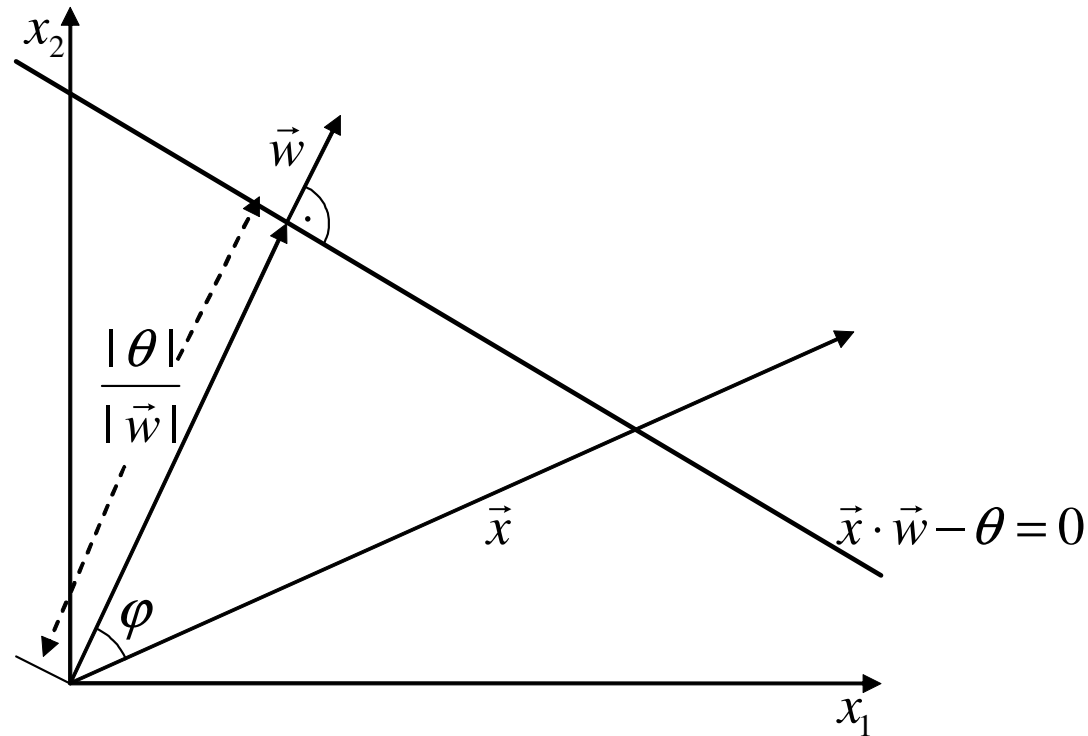
■ **Schwellenwert:** θ

■ **Eingabevektor:** $\vec{x} = (x_1, \dots, x_n)$

■ **Ausgabewert:**
$$f(\vec{x}) = \begin{cases} 1, & \text{falls } \vec{w} \cdot \vec{x} \geq \theta \\ 0, & \text{sonst.} \end{cases}$$

■ **Trennende Hyperebene:** $\vec{w} \cdot \vec{x} - \theta = 0$

Geometrische Interpretation

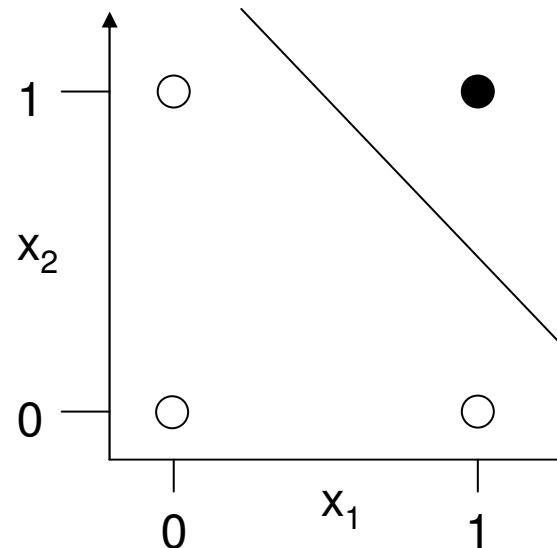


- θ negativ, falls Ursprung auf der Seite der Ebene, in die der Normalvektor zeigt, positiv sonst.
- $\vec{w} \cdot \vec{x} - \theta > 0$, falls x auf der Seite der Ebene liegt, zu der der Normalenvektor zeigt.

Geometrische Interpretation

- Ein Perzeptron kann genau die Menge der linear separablen Funktionen darstellen.
- AND ist linear separabel und somit durch ein Perzeptron darstellbar:

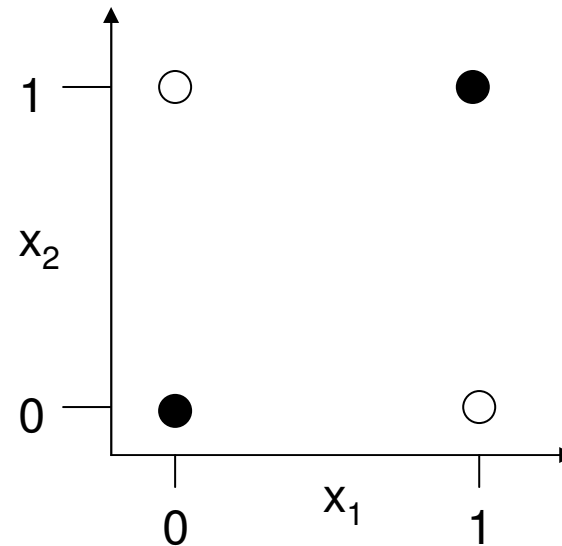
x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1



Geometrische Interpretation

- Die Biimplikation ist nicht linear separabel und es gibt somit kein Perzeptron, das die gerade Parität als Funktion berechnet:

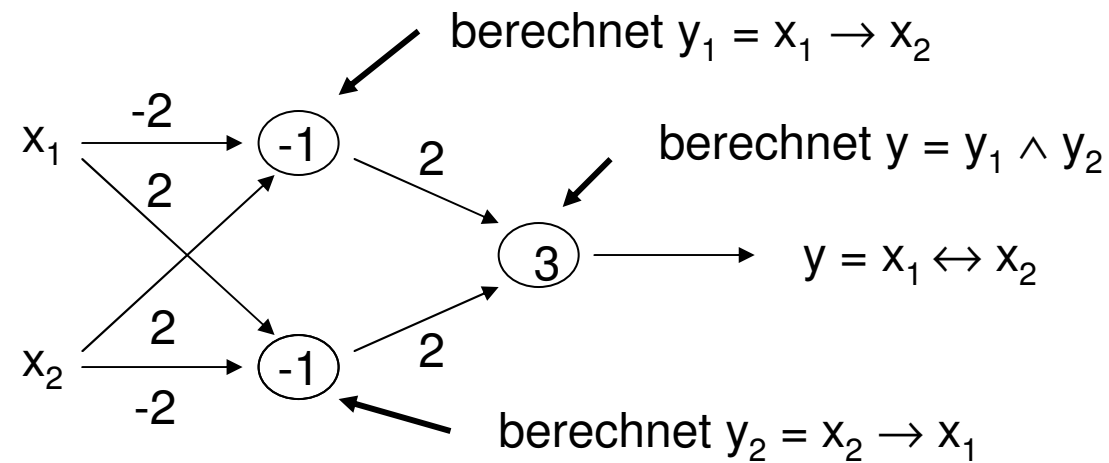
x_1	x_2	$g(x_1, x_2)$
0	0	1
1	0	0
0	1	0
1	1	1



Es gibt keine Trenngerade, die den Lösungsraum aufteilt

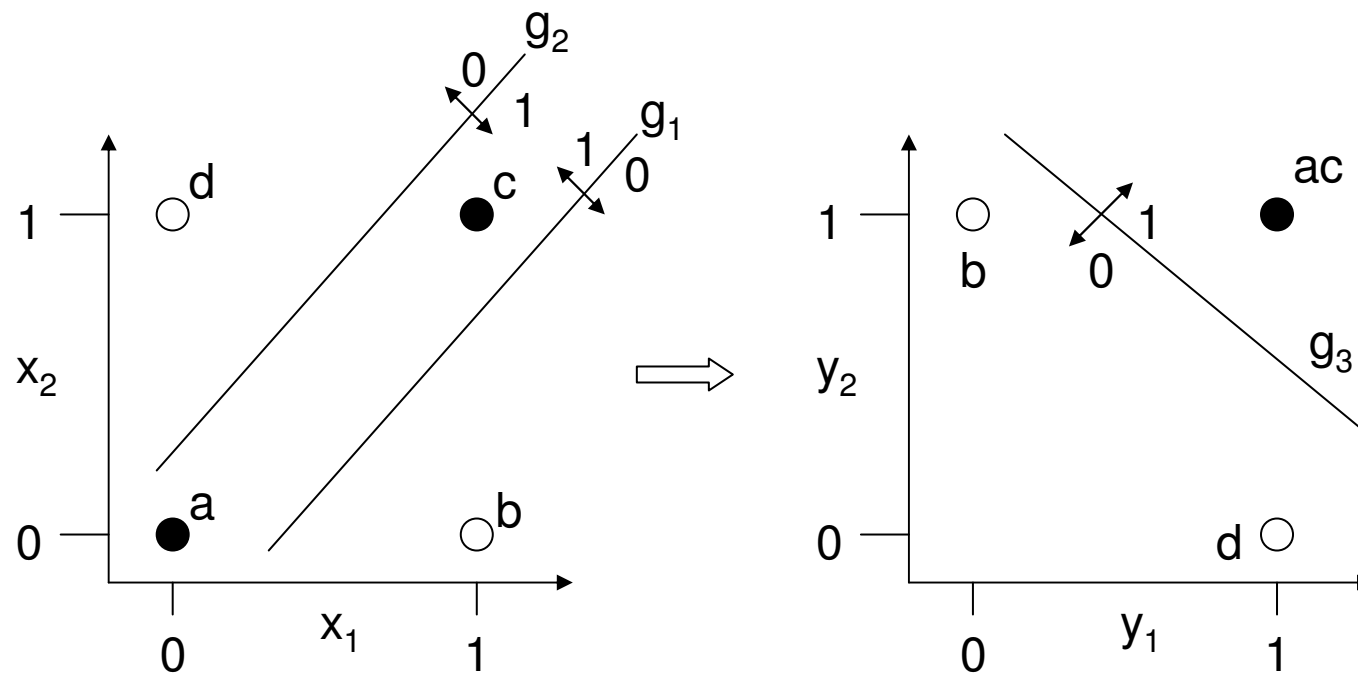
Biimplikationsproblem

- Lösung: Zusammenschalten mehrerer Schwellenwertelemente



Geometrische Deutung

- Geometrische Deutung des Zusammenschaltens mehrerer Schwellenwertelemente zur Berechnung der Biimplikation:



3.2 Lernverfahren für Perzeptrons

Problemstellung

Gegeben:

Eine Lernstichprobe $L = \{(x_1, d_1), \dots, (x_n, d_n)\}$ von Merkmalen mit zugehörigen Aktionen. Man spricht auch von einer Trainingsmenge, die von einem Lehrer gegeben wurde (überwachtes Lernen; supervised learning).

Gesucht:

Eine Funktion $f: \{0,1\}^n \rightarrow \{0,1\}$, die zu der Trainingsmenge „passt“.

3.3 Trainieren einzelner Perzeptrons

- Beim Trainieren werden die Gewichte und der Schwellenwert der TLU so verändert, dass für die Elemente der Lernstichprobe $\vec{x}_i$ der vorgegebene Wert d_i und der von einem Perceptron tatsächlich berechnete Wert (möglichst) gut übereinstimmen. Dies kann man erreichen, in dem man den quadratischen Fehler

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

minimiert.

- Lernen entspricht also einer Optimierungsaufgabe.

$$\mathcal{E} = \sum_{(\underline{x}, d) \in L} (d - f_{\vec{w}}(\vec{x}))^2 = \sum_{i=1}^m (d_i - f_i)^2$$

Gradientenabstiegsverfahren

- Ein typisches Optimierungsverfahren ist das Gradientenabstiegsverfahren. Wir wählen eine vereinfachte Version, bei der wir nicht gleichzeitig für alle Elemente der Lernstichprobe, sondern die Fehler der Reihe nach (inkrementell) minimieren.
- Idee: Bestimmung der Abhängigkeit des Fehlers von den Gewichten durch Berechnung der partiellen Ableitungen:

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

- Anschließend Berechnung neuer Gewichte nach:

$$\text{Nabla - Operator} : \nabla_{\vec{w}} \varepsilon = \frac{\partial \varepsilon}{\partial \vec{w}} = \left[\frac{\partial \varepsilon}{\partial w_1}, \dots, \frac{\partial \varepsilon}{\partial w_{n+1}} \right]$$

wobei c eine Lernrate ist, die die „Schrittweite“ des Abstiegs in Richtung des Gradienten festlegt:

$$\vec{w}^{(neu)} := \vec{w}^{(alt)} + c \cdot \nabla_{\vec{w}} \varepsilon$$

Gradientenabstiegsverfahren

- Gradientenabstiegsverfahren anschaulich:

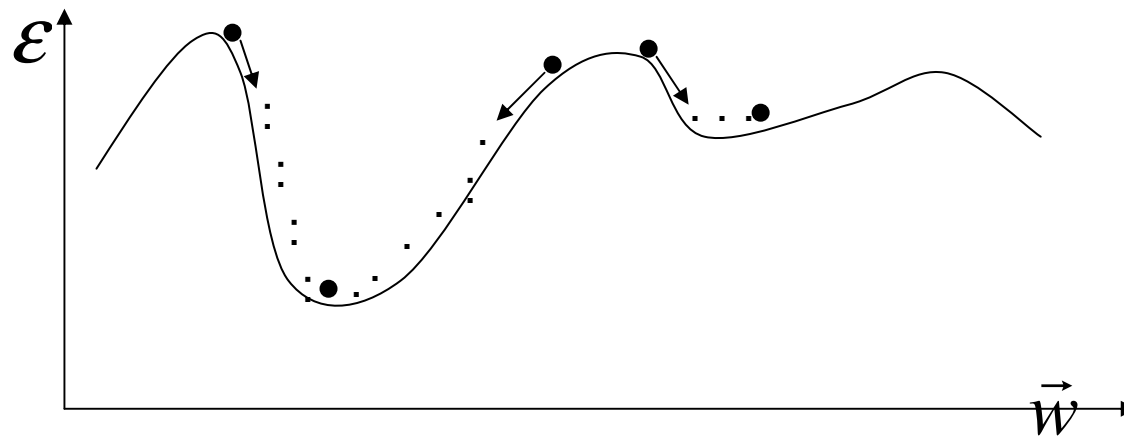
Zu jedem Gewichtsvektor w gehört ein Fehlerwert. Man hat also ein „Fehlergebirge“ über dem durch die Gewichte aufgespannten Raum. Beim Gradientenabstiegsverfahren geht man von einem gegebenen Punkt in diesem „Gebirge“ solange talwärts, bis man eine Talsohle erreicht hat.

- Problem:

Je nach „Fehlergebirge“ und Startpunkt kann man in lokalen Minima hängen bleiben (siehe Skizze)

Gradientenabstiegsverfahren

■ Graphische Veranschaulichung:



Gradientenabstieg für Perzeptron

■ Fehler: $\varepsilon = (d - f)^2$

■ Gradient: $\frac{\partial \varepsilon}{\partial \vec{w}} = \left[\frac{\partial \varepsilon}{\partial w_1}, \dots, \frac{\partial \varepsilon}{\partial w_{n+1}} \right]$

Der Fehler hängt vom Gewichtsvektor über das Skalarprodukt mit dem Eingangsvektor ab

$$= \frac{\partial \varepsilon}{\partial s} \cdot \frac{\partial s}{\partial \vec{w}}$$

(da $s = \vec{x} \cdot \vec{w}$)

$$= \frac{\partial \varepsilon}{\partial s} \cdot \vec{x}$$

(da $\frac{\partial s}{\partial \vec{w}} = \vec{x}$)

$$= -2(d - f) \cdot \frac{\partial f}{\partial s} \cdot \vec{x}$$

■ Problem: ε ist nicht stetig differenzierbar (wegen des Schwellenwertes)

Delta-Regel

Fehler Korrektur-Verfahren (Delta-Regel):

- **Eine Änderung des Gewichtsvektors wird nur im Falle eines Fehlers vorgenommen.**
- **Es wird eine Lernrate $c > 0$ eingeführt.**
- **Es gilt:**

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

Algorithmus:

- Initialisierung von w (zufällig oder gesetzt)**
- $\varepsilon := 0$**
- Für jedes Element (x, d) der Lernstichprobe werden folgende Schritte durchgeführt:**

Delta-Regel (Algorithmus)

- $\mathcal{E} := \mathcal{E} + (d - f)^2 = \mathcal{E} + (d - f_{\vec{w}}(\vec{x}))^2$

- Bestimme für $i=1, \dots, n+1$:

$$\Delta_{(\vec{x}, d)} w_i = \begin{cases} 0, & \text{falls } d = f_w(\vec{x}) \\ c \cdot x_i, & \text{falls } f_w(\vec{x}) = 0 \text{ und } d = 1 \\ -c \cdot x_i, & \text{falls } f_w(\vec{x}) = 1 \text{ und } d = 0. \end{cases}$$

(Beachte: $+\Delta\theta = -\Delta_{(\Delta, d)} w_{n+1}$, $x_{n+1} = 1$, $\theta = -w_{n+1}$)

- Bestimme neue Gewichte: $\vec{w} := \vec{w} + \Delta_{(\vec{x}, d)}(\vec{w})$

(iv) if ε minimal then end else weiter mit (ii) fi

Delta-Regel (Anmerkungen)

- Die Berechnung der delta-Werte lässt sich allgemein auch schreiben als
bzw. als Vektoren

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

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

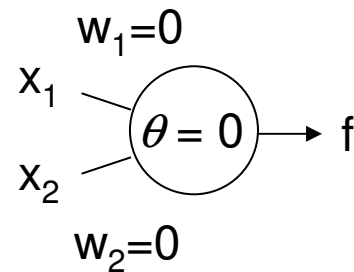
- Oft fordert man als Abbruchbedingung nur ε kleiner als eine vorgegebene Schranke.
- Ein Durchgang heißt Lernperiode.
- Für linear separable Lernstichproben terminiert dieser Algorithmus.

Delta-Regel (Beispiel AND)

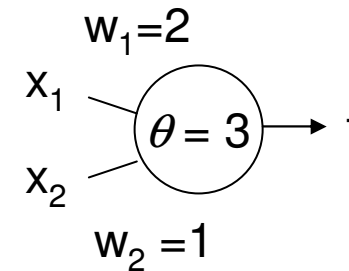
■ Lernvorgang für AND liefert

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

Initialisierung:



Nach dem Lernen:



■ (eine von vielen möglichen Lösungen!)

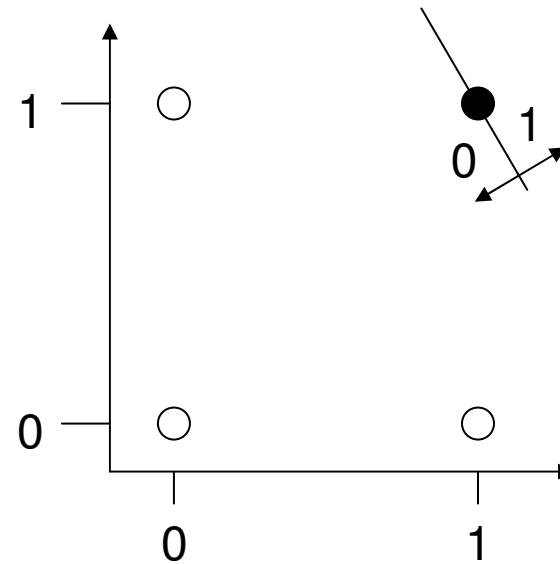
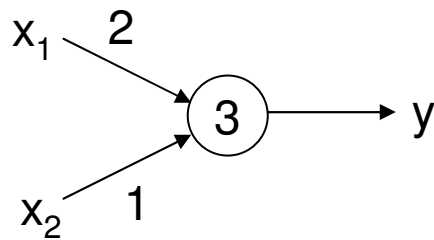
Delta-Regel (Beispiel AND)

Epoche	x_1	x_2	d	$\bar{x}w$	y	ε	$\Delta\theta$	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

Delta-Regel (Beispiel AND)

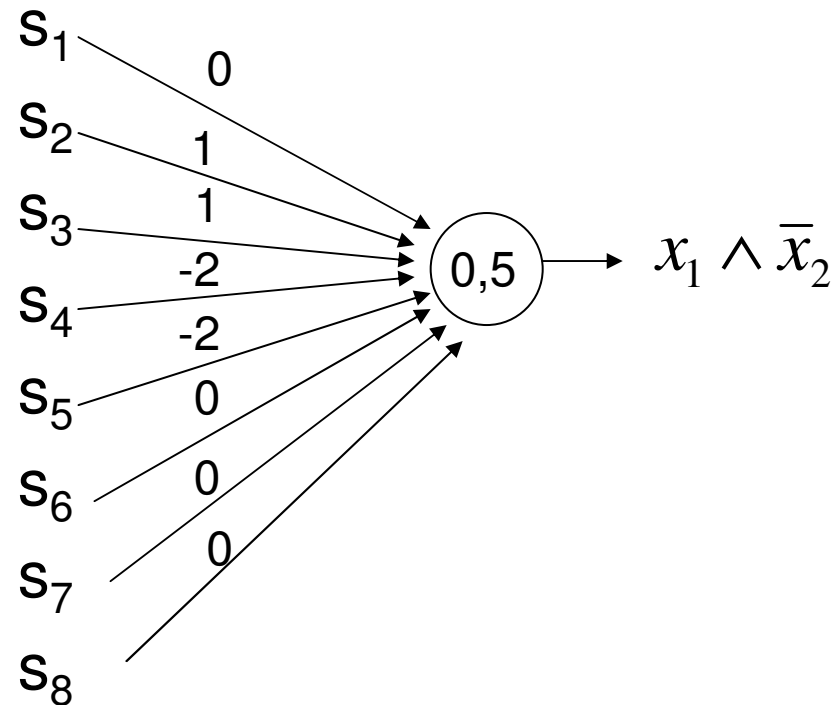
- Geometrie des gelernten Schwellenwertelementes für $x_1 \wedge x_2$
- Die rechts gezeigte Gerade hat die Gleichung

$$2x_1 + x_2 = 3$$



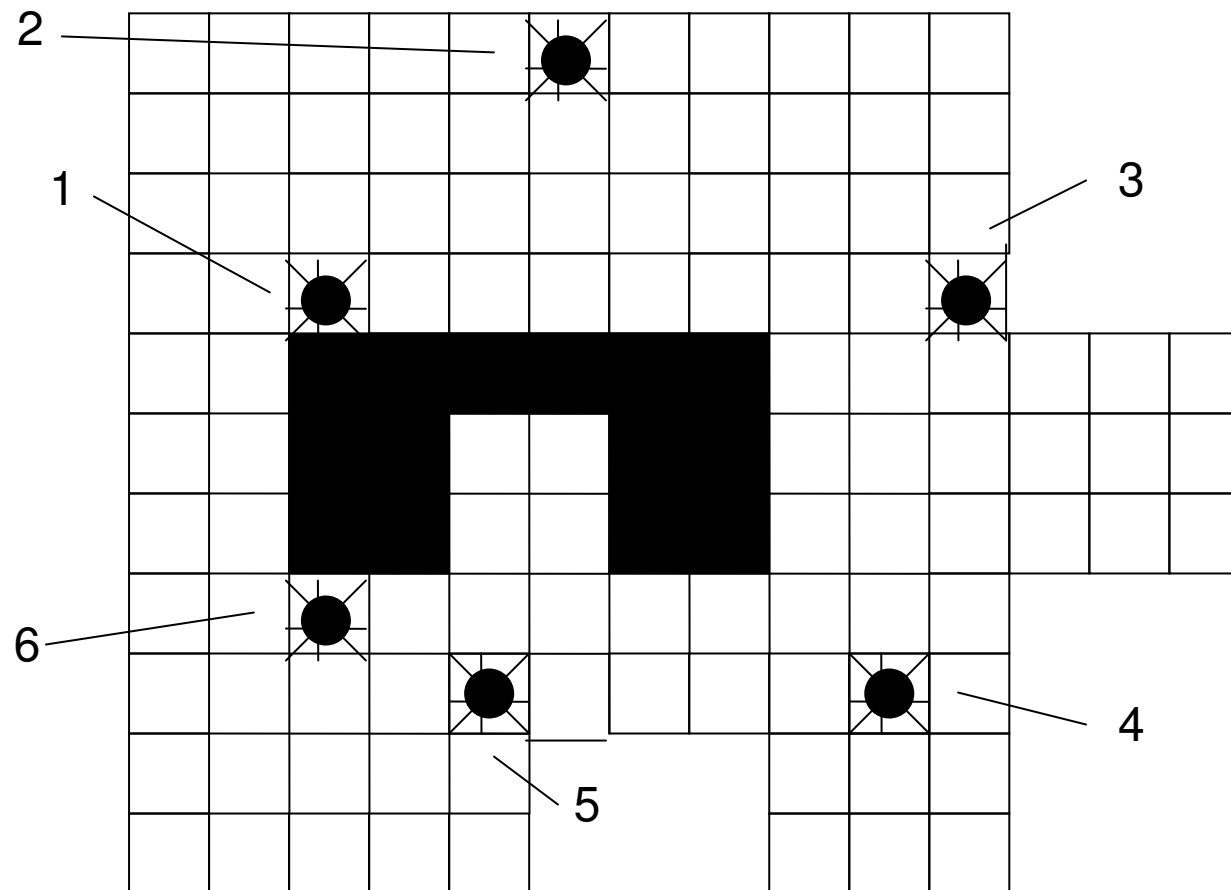
Perzeptren in der Grid-World

- move east $\leftarrow x_1 = 1 \text{ and } x_2 = 0 \leftarrow x_1 \wedge \bar{x}_2 = (s_2 \vee s_3) \wedge \bar{s}_4 \wedge \bar{s}_5$
- Dieses Perceptron liefert genau dann 1, wenn der Roboter nach Osten gehen soll:



Lerndatensatz („nach Osten gehen“)

- Man kann Gewichte anhand von Beispielen erlernen (hier lohnt das nicht, nur bei komplexen Beispielen mit unbekannten Funktionen):



Lerndatensatz („nach Osten gehen“)

Eingabe- nummer	Sensordaten-vektor	$x_1 \bar{x}_2$ (move east)
1	00001100	0
2	11100000	1
3	00100000	1
4	00000000	0
5	00000100	0
6	01100000	1

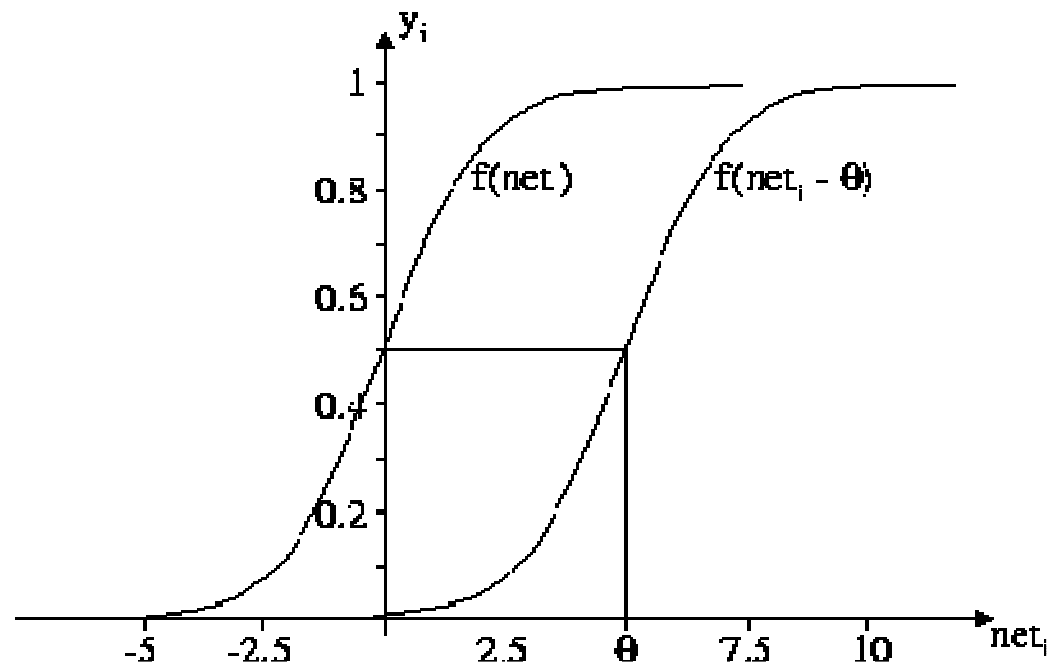
Verallgemeinerte Delta-Regel

- Idee: Ersetze die Schwellenfunktion durch eine s-förmige (sigmoide) differenzierbare Funktion, wie z.B. die logistische Funktion:

- Es gilt hier:

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

$$\frac{\partial f}{\partial s} = f(1 - f)$$



Verallgemeinerte Delta-Regel

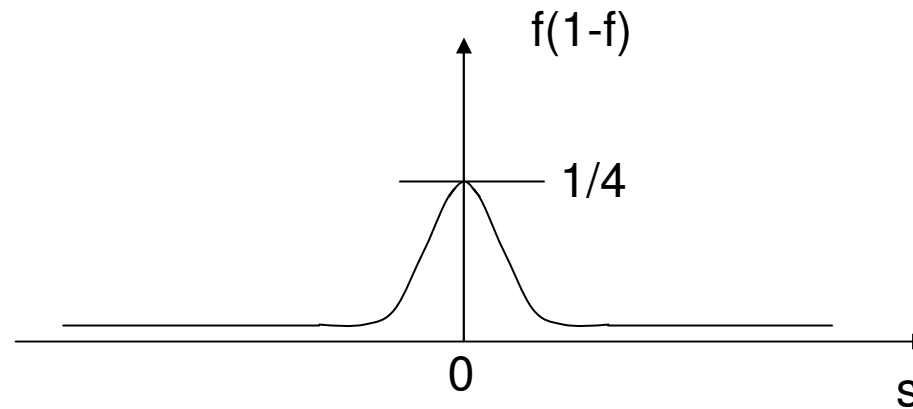
■ Genaue Ableitung:

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

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

Verallgemeinerte Delta-Regel

- Graph von $f(1-f)$:



- Wirkung: Nahe an der durch den Gewichtsvektor beschriebenen Hyperebene ist die Änderung stark, weiter davon entfernt gering (bei gleichem Fehler).

Verallgemeinerte Delta-Regel

- Für den Fehlergradienten erhält man somit:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = -2(d - f)f(1 - f) \cdot \vec{x}$$

- Wie bei der Delta-Regel erhält man damit eine neue Schätzung für w :

$$\vec{w}^{neu} := \vec{w}^{alt} + c \cdot (d - f)f(1 - f) \cdot \vec{x}$$

- Hierbei wird eine „variable“ Lernrate c eingeführt, mit der man die Änderungsstärke einstellen kann.
- Weitere Verbesserungen des Lernverfahrens sind möglich.

Delta-Regel (Beispiel)

- Online-Lernvorgang eines Schwellenwertelementes für die Negation mit Startwerten $\theta = \frac{3}{2}$, $w = 2$ und Lernrate 1

Epoche	x	d	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1,5	2
1	0	1	-1,5	0	1	-1	0	0,5	2
	1	0	1,5	1	-1	1	-1	1,5	1
2	0	1	-1,5	0	1	-1	0	0,5	1
	1	0	0,5	1	-1	1	-1	1,5	0
3	0	1	-1,5	0	1	-1	0	0,5	0
	1	0	-0,5	0	0	0	0	0,5	0
4	0	1	-0,5	0	1	-1	0	-0,5	0
	1	0	0,5	1	-1	1	-1	0,5	-1
5	0	1	-0,5	0	1	-1	0	-0,5	-1
	1	0	-0,5	0	0	0	0	-0,5	-1
6	0	1	0,5	1	0	0	0	-0,5	-1
	1	0	-0,5	0	0	0	0	-0,5	-1

Delta-Regel (Beispiel)

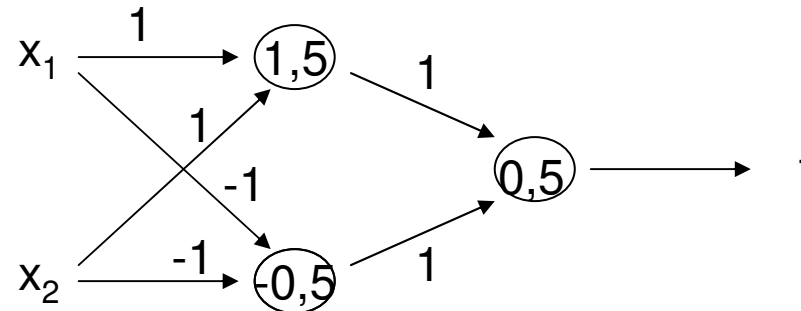
- Batch-Lernvorgang eines Schwellenwertelementes für die Negation mit Startwerten $\theta = \frac{3}{2}$, $w = 2$ und Lernrate 1.

Epoche	x	d	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1,5	2
1	0	1	-1,5	0	1	-1	0		
	1	0	0,5	1	-1	1	-1	1,5	1
2	0	1	-1,5	0	1	-1	0		
	1	0	-0,5	0	0	0	0	0,5	1
3	0	1	-0,5	0	1	-1	0		
	1	0	0,5	1	-1	1	-1	0,5	0
4	0	1	-0,5	0	1	-1	0		
	1	0	-0,5	0	0	0	0	-0,5	0
5	0	1	0,5	1	0	0	0		
	1	0	0,5	1	-1	1	-1	0,5	-1
6	0	1	-0,5	0	1	-1	0		
	1	0	-0,5	0	0	0	0	-0,5	-1
7	0	1	0,5	1	0	0	0		
	1	0	-0,5	0	0	0	0	-0,5	-1

3.4 Mehrschichtige Perzeptrons

- Im Falle nicht separabler Funktionen verwendet man mehrschichtige Neuronale Netze.
- Beispiel: Funktion „gleiche Parität“

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



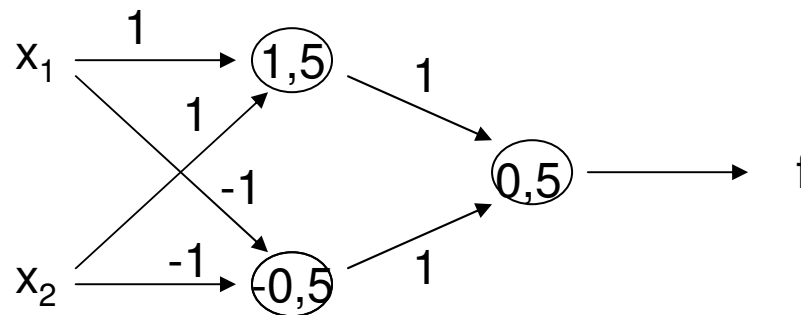
- Zweischichtiges Neuronales Netz
- Neuronen in der ersten Schicht werden auch als verborgene oder innere Neuronen bezeichnet

Mehrschichtige Perzeptrons

■ Propagation im 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
		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



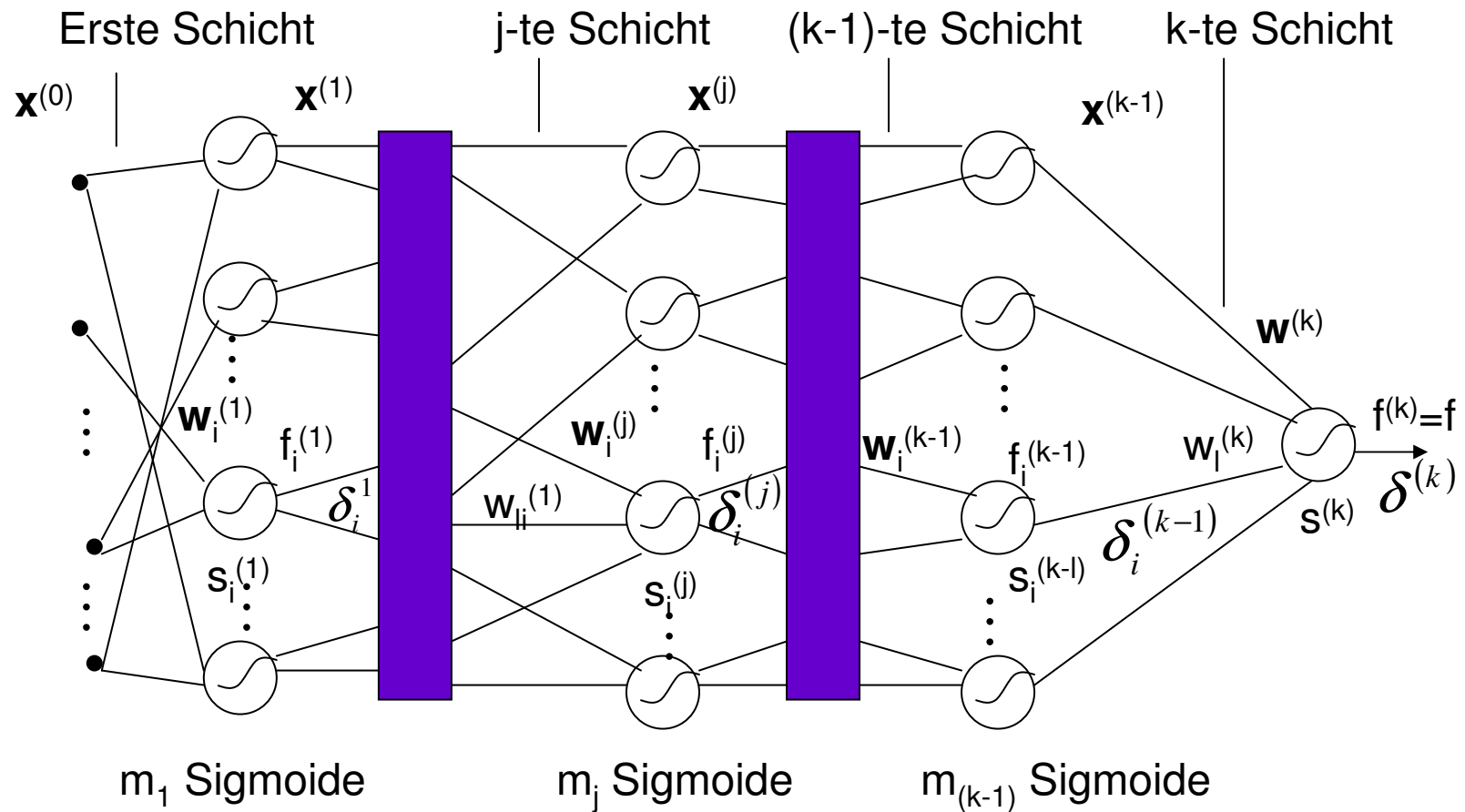
Mehrschichtige Perzeptrons

Definition:

- Ein k -schichtiges Perceptron besteht aus k Schichten, wobei in der Schicht j die m_j Neuronen mit Sigmoiden die Ausgabe x^j liefern,
- x^0 ist die Eingabe,
- f die Ausgabe,
- $\vec{w}_i^j$ ist der Gewichtsvektor des i -ten Sigmoids in der Schicht j , wobei die letzte Komponente jeweils der Schwellenwert ist.
- Wir nennen die Summe $s_i^j = \vec{x}^{j-1} \cdot \vec{w}_i^j$

die „Aktivierung“ des sigmoiden Neurons.

Mehrschichtige Perzeptrons



Lernen in mehrschichtigen Netzen

- Ein spezielles Trainingsverfahren heißt Backpropagation (der Fehler wird durch das Netz zurückgeschickt).
- Idee: Analog Gradientenabstieg wird ein Fehlergradient berechnet.

Herleitung des Verfahrens:

- Fehler für Ausgabeschicht: $\mathcal{E} = (d - f)^2$

- Gradient:
$$\frac{\partial \mathcal{E}}{\partial \vec{w}_i^j} = \left[\frac{\partial \mathcal{E}}{\partial w_{1i}^j}, \dots, \frac{\partial \mathcal{E}}{\partial w_{li}^j}, \dots, \frac{\partial \mathcal{E}}{\partial w_{m_{j-1}+1,i}^j} \right],$$

wobei $\vec{w}_{li}^j$ die l -te Komponente von $\vec{w}_i^j$ ist

Backpropagation (Herleitung)

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

Berechnung der neuen Gewichte :

$$(\vec{w}_i^j)^{neu} := (\vec{w}_i^j)^{alt} + c_i^j \cdot \delta_i^j \vec{x}^{j-1}$$

wobei c_i^j eine Lernrate ist. (Meistens wird ein Wert für das gesamte Netz genommen.)

Backpropagation (Herleitung)

Gewichtsänderungen in der Ausgabeschicht:

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

$$\vec{w}^k := \vec{w}^k + c^k (d - f) f (1 - f) \vec{x}^{k-1}$$

Anmerkungen:

- Gilt bei (logistischer) sigmoider Funktion mit Bias 0 (wegen zusätzlicher Eingabe θ)
- Nur ein Wert, daher keine Indices; sonst wie bei verallgemeinerter Deltaregel

Backpropagation (Herleitung)

Gewichtsänderungen in verborgener Schicht:

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

Der Summand muss noch berechnet werden.

Betrachte hierzu zunächst:

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

(γ ist der Index über die einzelnen Vektorelemente)

Backpropagation (Herleitung)

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

Somit erhalten wir ein rekursives Gleichungssystem zur Berechnung der δ_i^j :

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

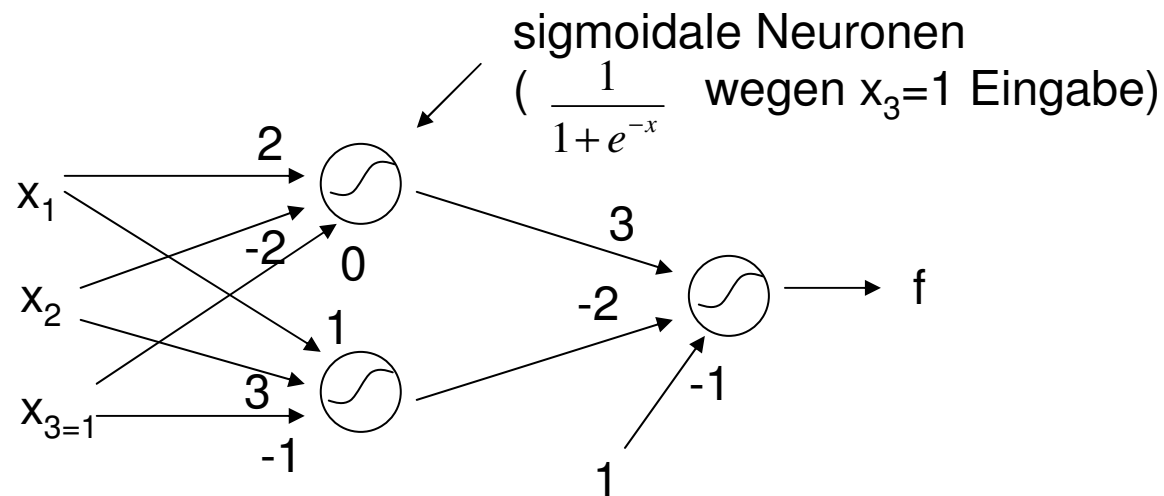
Und schließlich zur Berechnung der neuen Gewichte:

$$\vec{w}_i^j := \vec{w}_i^j + c_i^j \cdot \delta_i^j \vec{x}^{j-1}$$

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:

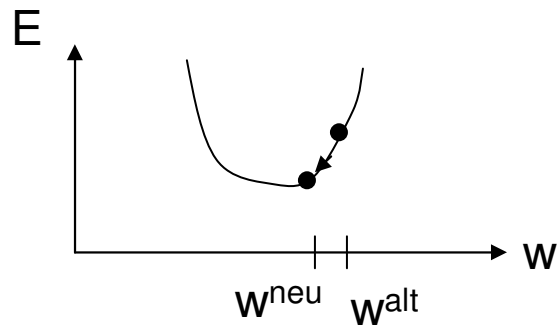
$$\rightarrow \begin{cases} \delta^2 = -0,148; \delta_1^1 = -0,047; \delta_2^1 = 0,074 \\ c = 1; \vec{w}_1^1 = (1,935; -2; -0,047) \end{cases}$$

$$\vec{w}_2^1 = (1,074; 3; -0,926)$$

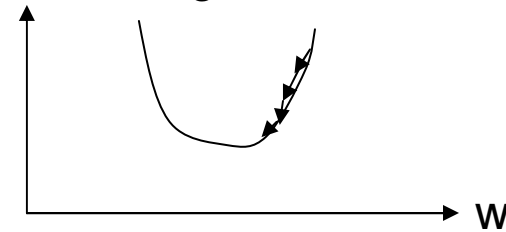
$$\vec{w}^2 = (2,870; -2,074; -1,148)$$

Effekt der Lernrate

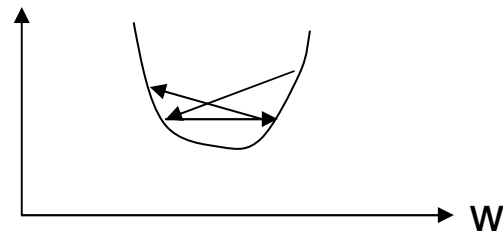
Anmerkungen zum Effekt der Lernrate:



zu klein:
zu langsam lernen

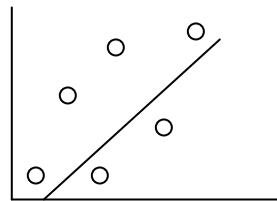
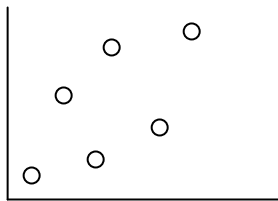


zu groß:
PingPong Effekt

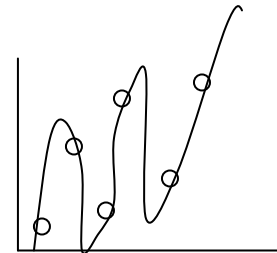


Eigenschaften von Neuronalen Netzen

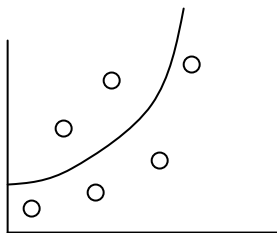
- NN können generalisieren, d.h. Vektoren klassifizieren, die nicht in der Trainingsmenge enthalten sind. Man kann die Güte der Generalisierung messen.
- Analogie: Funktionsapproximation



zu einfach,
schlechte Güte



zu kompliziert,
kein Fehler,
Daten auswendig gelernt,
oft schlechte Generalisierung,
overfitting



Mittelweg,
Occams razor

Vorgehensweise beim Lernen

- **Lernstichprobe = Trainingsmenge + Validierungsmenge**
 - Mit Trainingsmenge (ca. 2/3 der Daten) Gewichte lernen.
 - Mit Validierungsmenge Güte schätzen.
- **Cross Validation:**
 - Lernstichprobe in n disjunkte Menge teilen.
 - Jeweils eine Menge als Validierungsmenge und $n-1$ Mengen zum Training verwenden.
 - Die n out-of sample errors danach mitteln.
- **Beispiele:**
 - Neuronale Netze werden in verschiedensten Gebieten wie Gesichtserkennung oder Börsenprognosen eingesetzt.
 - NN auch im Data Mining genutzt (Siehe z.B. unseren Artikel in Spektrum der Wissenschaften, Nov. 2002, ab S. 80)