Intelligente Systeme

Einführung

Prof. Dr. Rudolf Kruse Georg Ruß
Christian Moewes

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

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

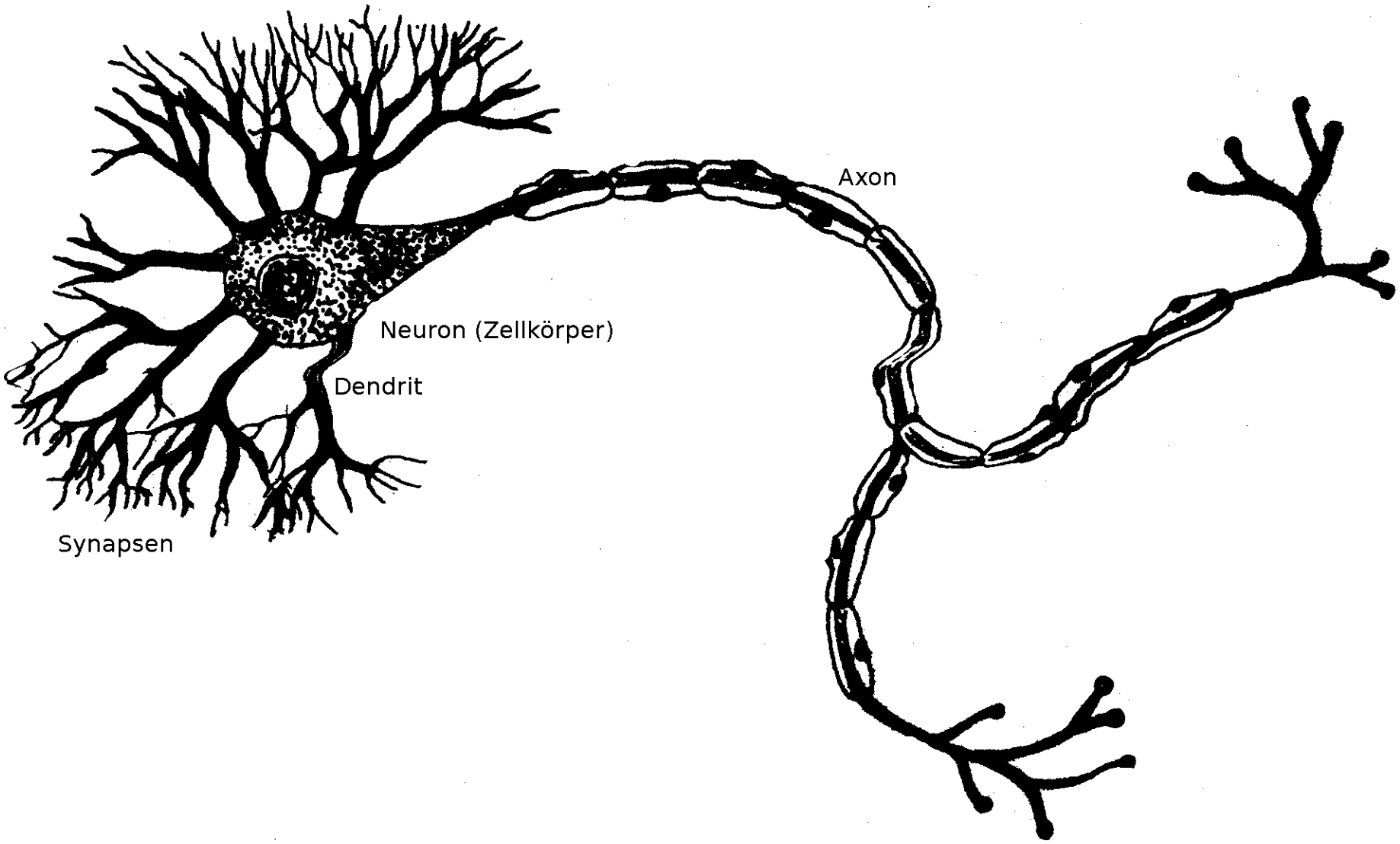
Neuronale Netze

Gliederung der Vorlesung

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
Verarbeitungs-geschwindigkeit	10^{-8} sec.	10^{-3} sec.
Bandbreite	$10^9 \frac{bits}{s}$	$10^{14} \frac{bits}{s}$
Neuronale Updates pro sec.	10^5	10^{14}

Konventionelle Rechner vs. Gehirn

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

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

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

Neuronale Netze

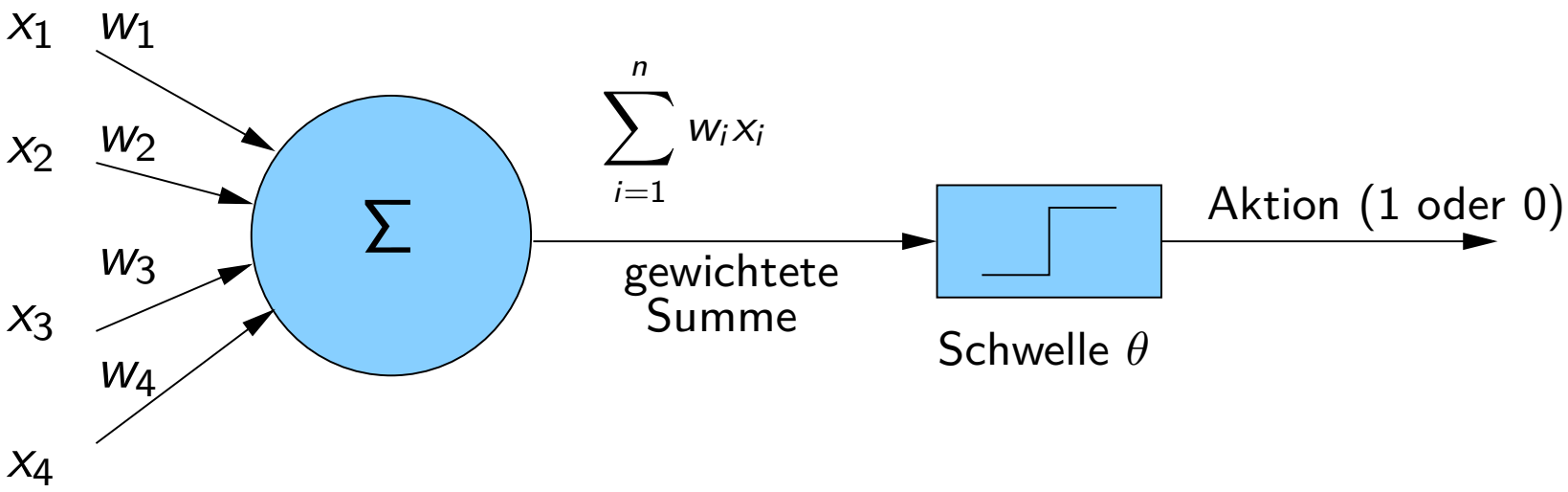
- **Vom mathematischen Standpunkt:** Neuronale Netze als Methode, Funktionen zu repräsentieren
  - durch Netzwerke von einfachen Berechungselementen (vergleichbar mit logischen Schaltkreisen)
  - die aus Beispielen gelernt werden können
  - (*Sichtweise in dieser Vorlesung!)*
- **Vom biologischen Standpunkt:** Neuronale Netze als stark vereinfachtes Modell des Gehirns und seiner Funktionsweise
  - Konnektionismus
  - (*Nicht in dieser Vorlesung!)*

Gliederung der Vorlesung

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:

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

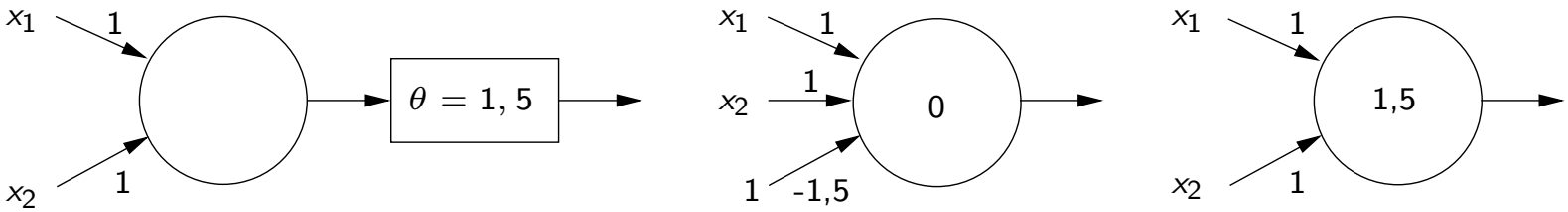
$$\mathbf{x} = (x_1, \ldots, x_n)$$

Perzeptrons: Vereinfachte Darstellung

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

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

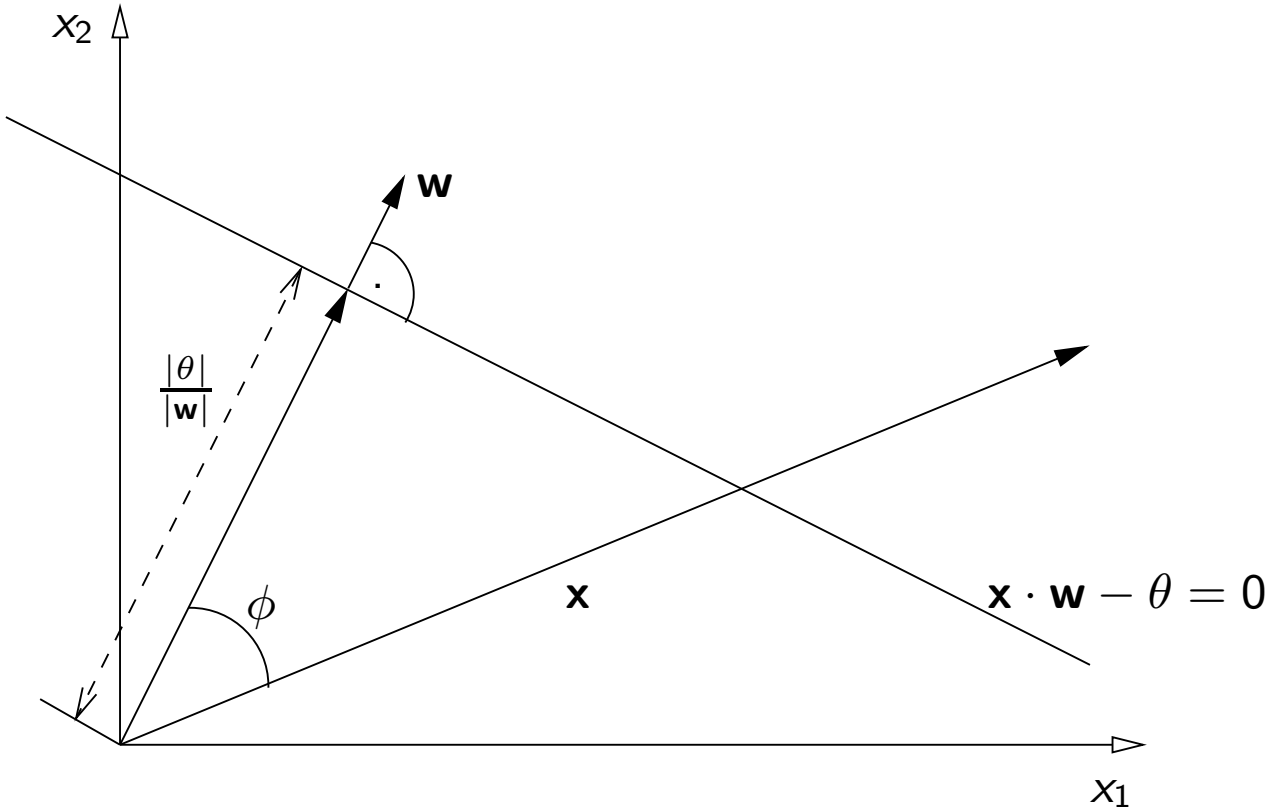
- Beispiel für äquivalente Darstellungen:



Geometrische Interpretation

- Gewichtsvektor: $\mathbf{w} = (w_1, \ldots, w_n)$
- Schwellenwert: θ
- Eingabevektor: $\mathbf{x} = (x_1, \ldots, x_n)$
- 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: $\mathbf{w} \cdot \mathbf{x} - \theta = 0$

Geometrische Interpretation

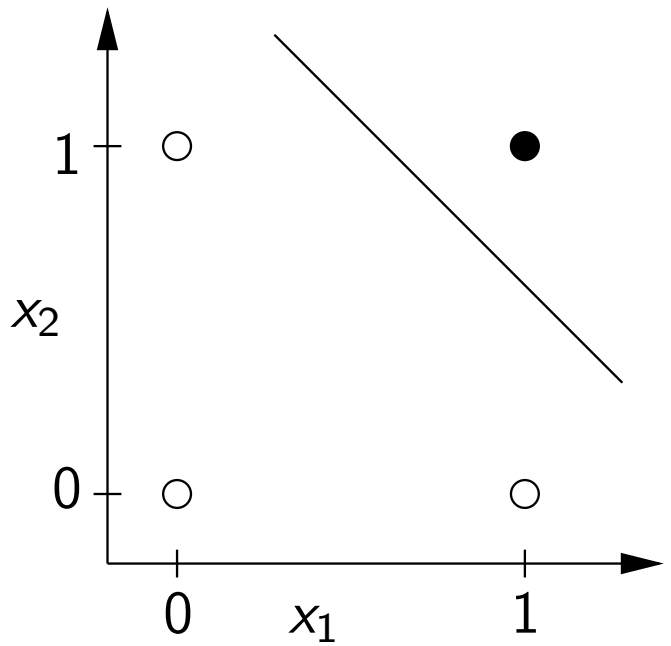


- θ negativ, falls Ursprung auf der Seite der Ebene, in die der Normalenvektor zeigt, positiv sonst
- $\mathbf{w} \cdot \mathbf{x} - \theta > 0$, falls x auf der Seite der Ebene liegt, in deren Richtung der Normalenvektor zeigt

Geometrische Interpretation

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

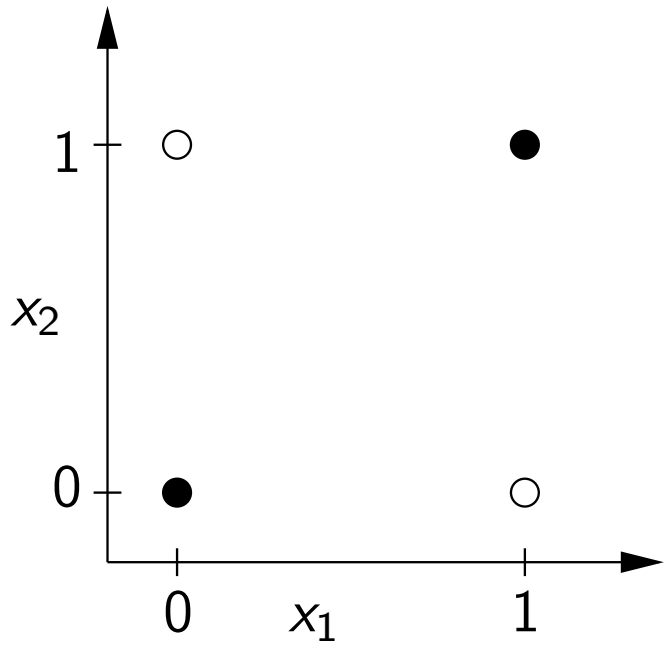
x_1	x_2	$x_1 \wedge x_2$
0	0	0
1	0	0
0	1	0
1	1	1



Geometrische Interpretation

- Die Biimplikation ist nicht linear separabel. Somit gibt es kein Perzeptron, das die Biimplikation (Äquivalenz, gerade Parität) als Funktion berechnet:

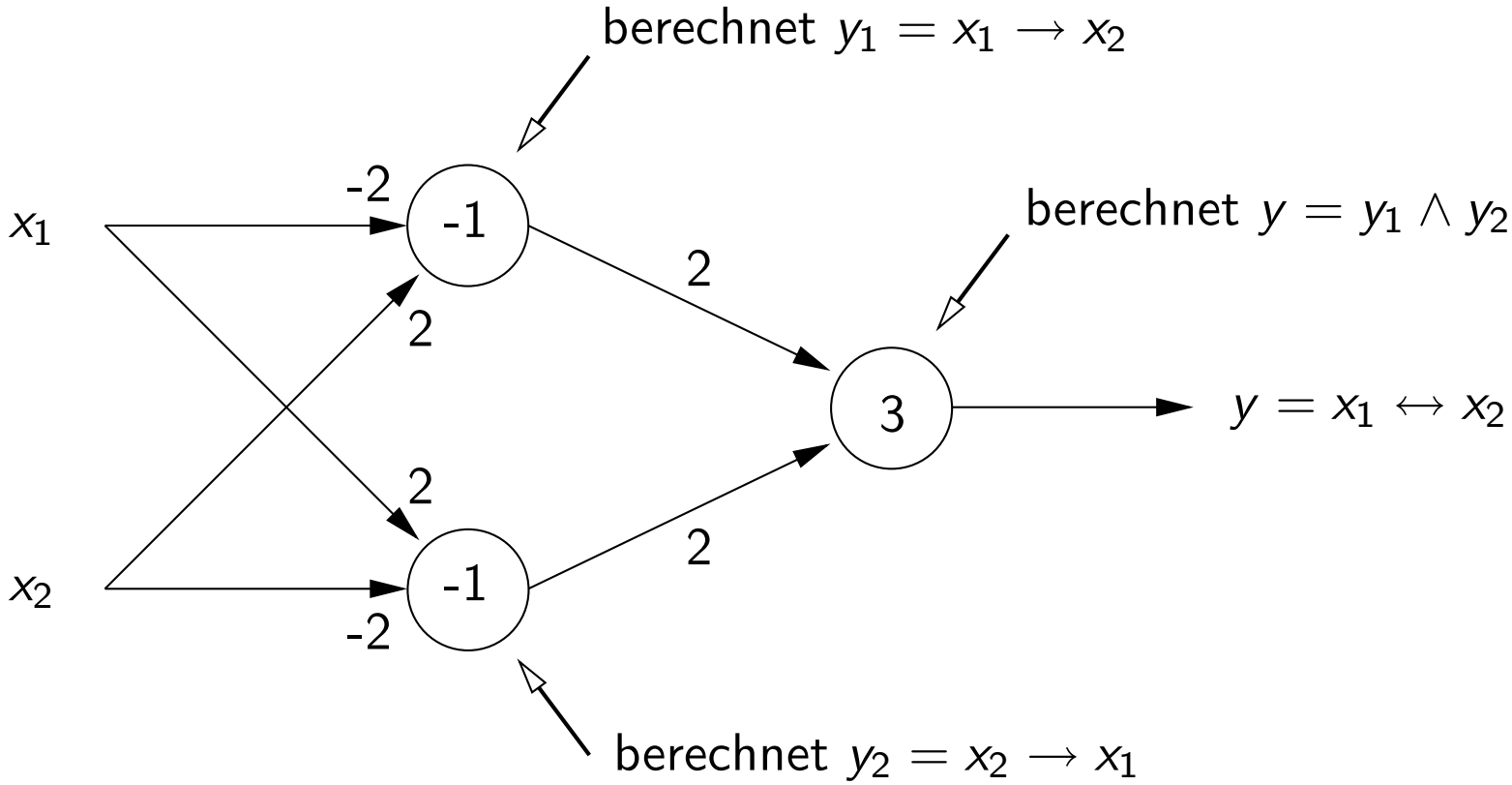
x_1	x_2	$x_1 \leftrightarrow x_2$
0	0	1
1	0	0
0	1	0
1	1	1



Es gibt keine Trenngerade, die den Lösungsraum aufteilt.

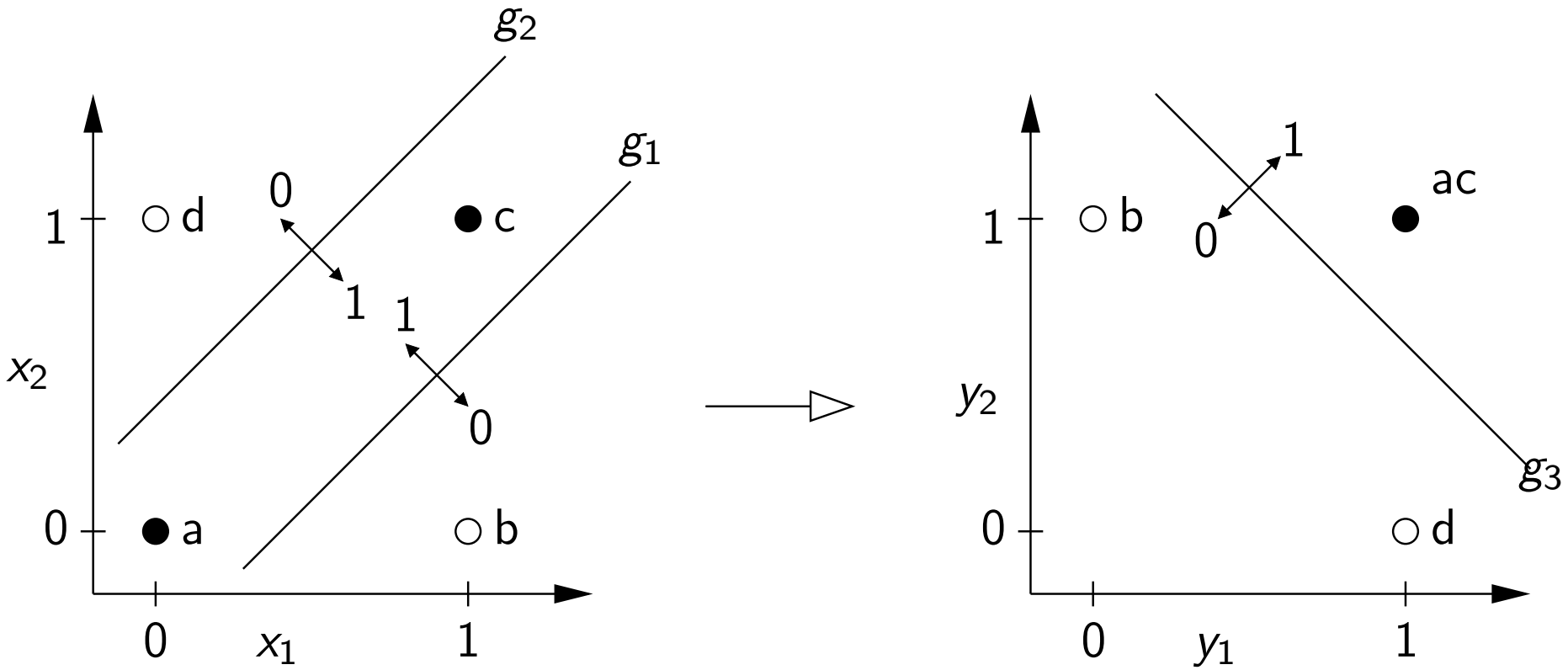
Biimplikationsproblem

- Lösung: Zusammenschalten mehrerer Schwellenwertelemente



Geometrische Deutung

- Geometrische Deutung des Zusammenschaltens mehrerer Schwellenwertelemente zur Berechnung der Biimplikation



Lernverfahren für Perzeptrons

Problemstellung:

- Gegeben:
  - Eine Lernstichprobe $L = (x_1, d_1), \ldots, (x_n, d_n)$ von Merkmalen mit zugehörigen Aktionen. Man spricht auch von einer Trainingsmenge, die von einem Lehrer gegeben wurde (überwachtes Lernen, *supervised learning*).

- Gesucht:
  - Eine Funktion $f : \{0,1\}^n \to \{0,1\}$, die zu der Trainingsmenge "passt".

Trainieren einzelner Perzeptrons

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

$$f_i = f_{w_1,\ldots,w_n,\theta}(\mathbf{x_i})$$

  minimiert.

- Lernen entspricht also einer Optimierungsaufgabe:

$$\varepsilon = \sum_{(\mathbf{x},d)\in L} (d - f_{\mathbf{w}}(\mathbf{x}))^2 = \sum_{i=1}^{m} (d_i - f_i)^2 \stackrel{!}{=} min$$

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:

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

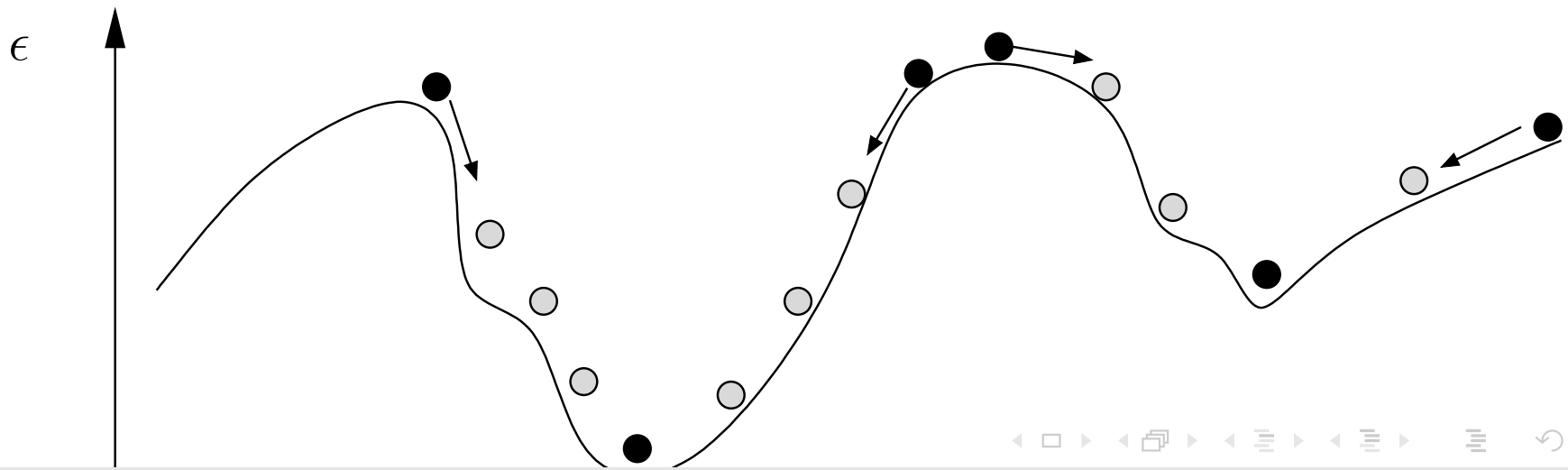
  wobei c eine Lernrate ist, die die "Schrittweite" des Abstiegs (in entgegengesetzter Richtung des Gradienten) festlegt:

$$\mathbf{w}^{(\text{neu})} := \mathbf{w}^{(\text{alt})} - c \cdot \nabla_{\mathbf{w}} \varepsilon$$

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.

- Je nach "Fehlergebirge" und Startpunkt kann man in lokalen Minima hängenbleiben:



Gradientenabstieg für Perzeptron

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

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

- Problem: ε ist nicht stetig differenzierbar wegen des Schwellenwertes

Delta-Regel

Fehlerkorrekturverfahren *Delta-Regel*:

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

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

Algorithmus:

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

Delta-Regel (Algorithmus)

3. (...)
   - $\varepsilon := \varepsilon + (d - f)^2 = \varepsilon + (d - f_{\mathbf{w}}(\mathbf{x}))^2$
   - Bestimme für $i = 1, \ldots, n+1$:
     $$\Delta_{(\mathbf{x},d)} w_i = \begin{cases} 0, & \text{falls } d = f_w(\mathbf{x}) \\ c \cdot x_i, & \text{falls } f_w(\mathbf{x}) = 0 \text{ und } d = 1 \\ -c \cdot x_i, & \text{falls } f_w(\mathbf{x}) = 1 \text{ und } d = 0. \end{cases}$$
     (Beachte: $+\Delta\theta = -\Delta_{(\Delta,d)} w_{n+1}, x_{n+1} = 1, \theta = -w_{n+1}$)
   - Bestimme neue Gewichte: $\mathbf{w} := \mathbf{w} + \Delta_{(\mathbf{x},d)}(\mathbf{w})$
4. falls ε minimal, terminiere, sonst weiter mit Schritt 2

Delta-Regel (Anmerkungen)

- Die Berechnung der Δ-Werte lässt sich allgemein auch schreiben als:

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

  bzw. in Vektorschreibweise:

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

- Oft fordert man als Abbruchbedingung nur, dass ε kleiner als eine vorgegebene Schranke sein soll.

- Ein Durchgang heißt Lernperiode.

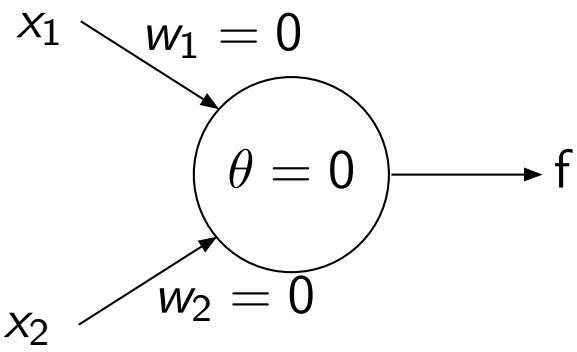
- Für linear separable Lernstichproben terminiert dieser Algorithmus.

Delta-Regel (Beispiel AND)

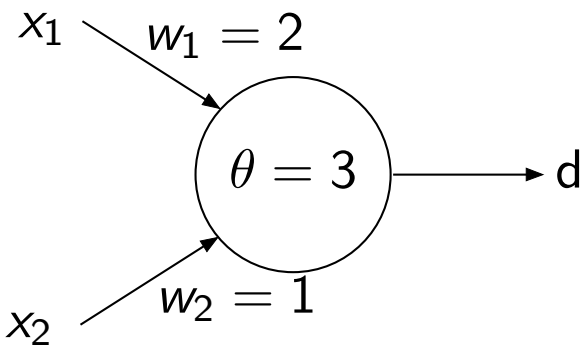
- Lernvorgang für AND liefert

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

Initialisierung



Nach dem Lernen



Der Lernvorgang verbleibt als Übungsaufgabe. Hinweis: es ist sinnvoll, eine Tabelle mit folgenden Werten anzulegen:

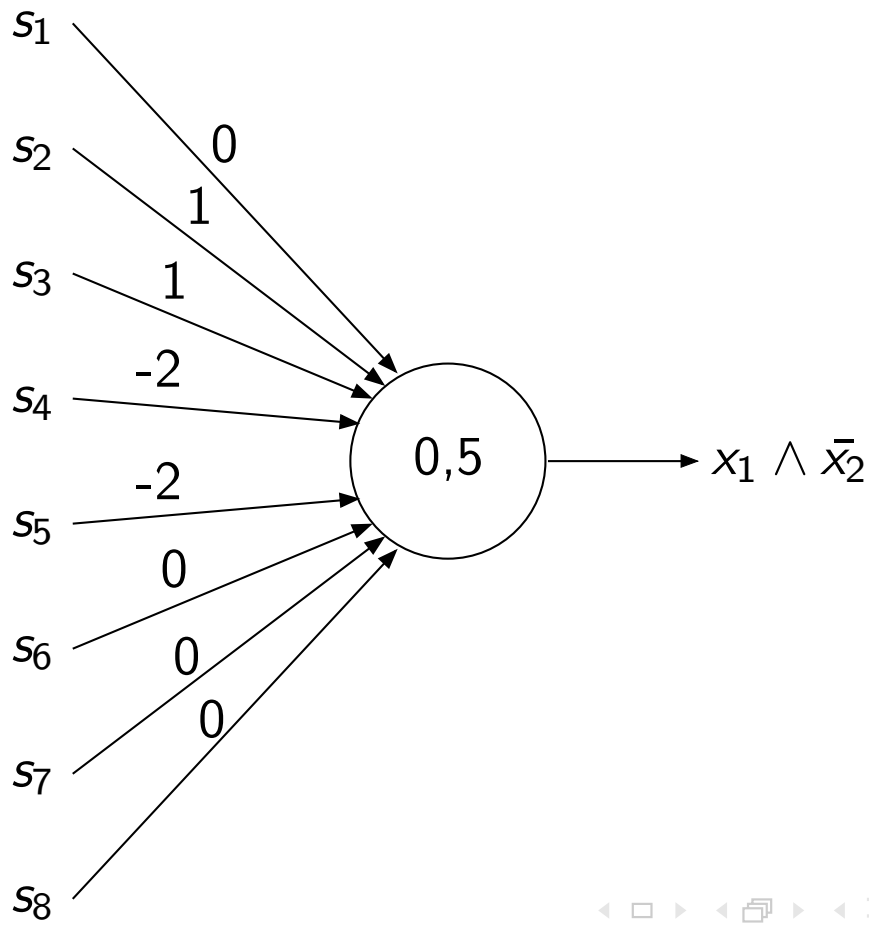
Epoche	x_1	x_2	d	**xw**	y	ε	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2

(Eine Epoche besteht hierbei aus einem Durchlauf aller Lernbeispiele.)

Perzeptren in der *grid world*

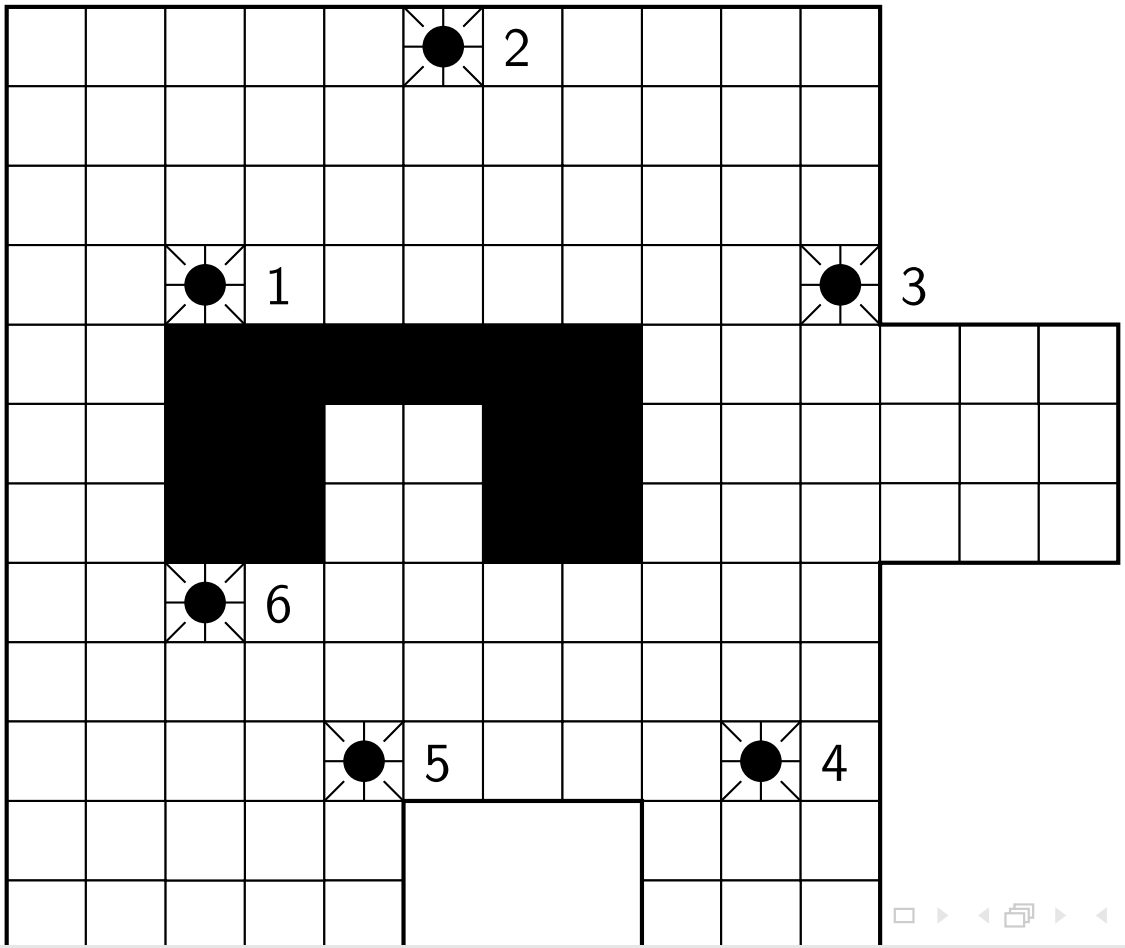
- `move east` $\leftarrow x_1 = 1 \wedge x_2 = 0 \leftarrow x_1 \wedge \bar{x_2} = (s_2 \vee s_3) \wedge \bar{s_4} \wedge \bar{s_5}$

- Das folgende Perzeptron liefert genau dann 1, wenn der Roboter nach Osten gehen soll:



Lerndatensatz (“nach Osten gehen”)

- Man kann Gewichte anhand von Beispielen erlernen. In diesem Beispiel lohnt das allerdings nicht, sondern nur bei komplexen Beispielen mit *unbekannten* Funktionen.
- Für die sechs markierten Positionen werden Eingabevektoren gesucht:



Lerndatensatz (“nach Osten gehen”)

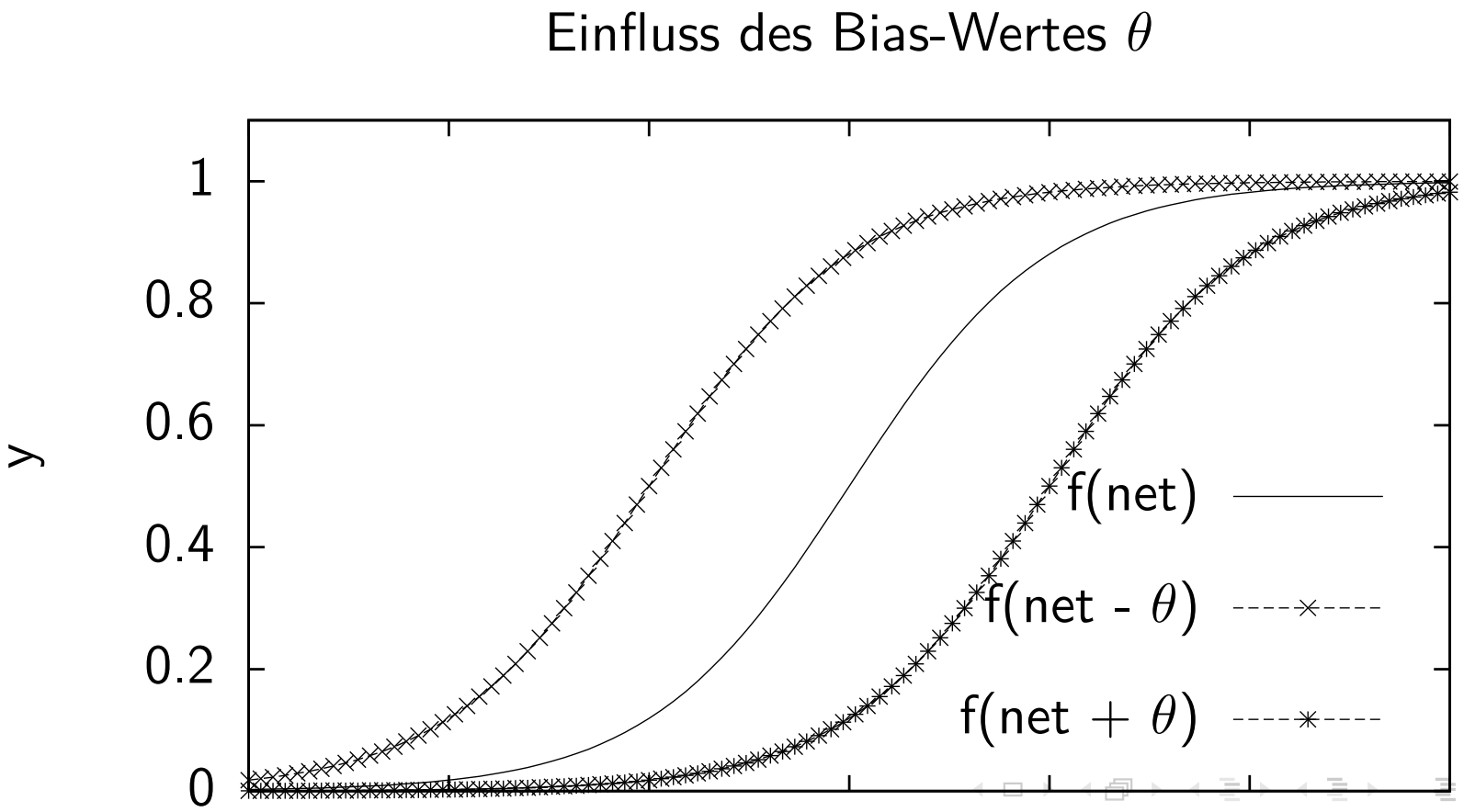
Der Lerndatensatz für dieses Beispiel sieht wie folgt aus:

Eingabenummer	Vektor der Sensordaten	$x_1 \wedge \bar{x}_2$ (`move east`)
1	00001100	0
2	11100000	1
3	00100000	0
4	00000000	0
5	00000100	0
6	01100000	1

Verallgemeinerte Delta-Regel

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

- Es gilt hier: $f(s) = \frac{1}{1+e^{-s}}$ und $\frac{\partial f}{\partial s} = f(1-f)$



Verallgemeinerte Delta-Regel

Genaue Ableitung:

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

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

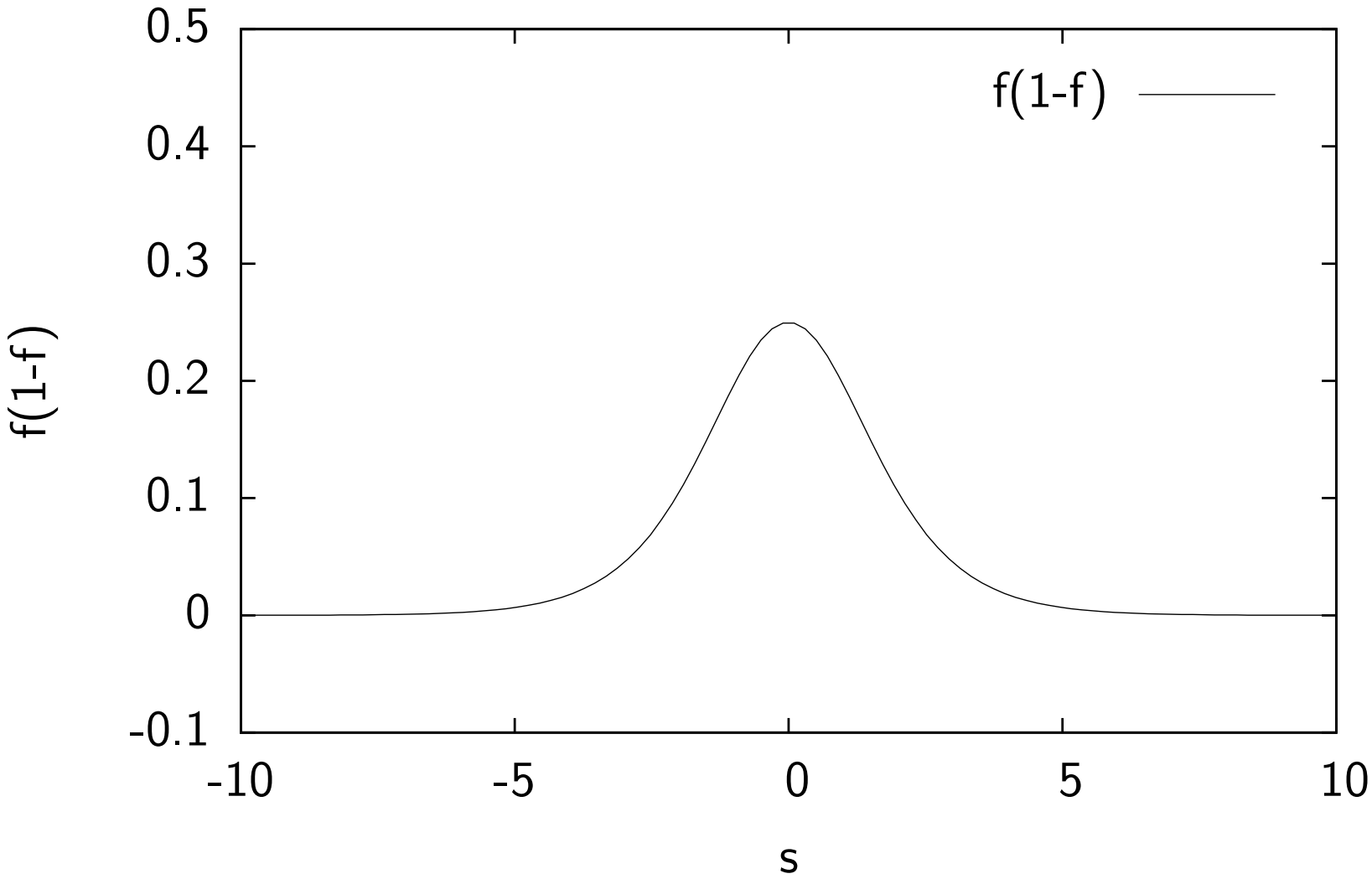
$$= \frac{1+e^{-s}}{(1+e^{-s})^2} - \frac{1}{(1+e^{-s})^2}$$

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

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

Verallgemeinerte Delta-Regel

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



Verallgemeinerte Delta-Regel

- Für den Fehlergradienten erhält man somit:

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

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

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

- Hierbei wird eine "variable" Lernrate c eingeführt, mit der man die Änderungsstärke einstellen kann.

- Weitere Verbesserungen des Lernverfahrens sind möglich.

Der Lernvorgang verbleibt wiederum als Übungsaufgabe.

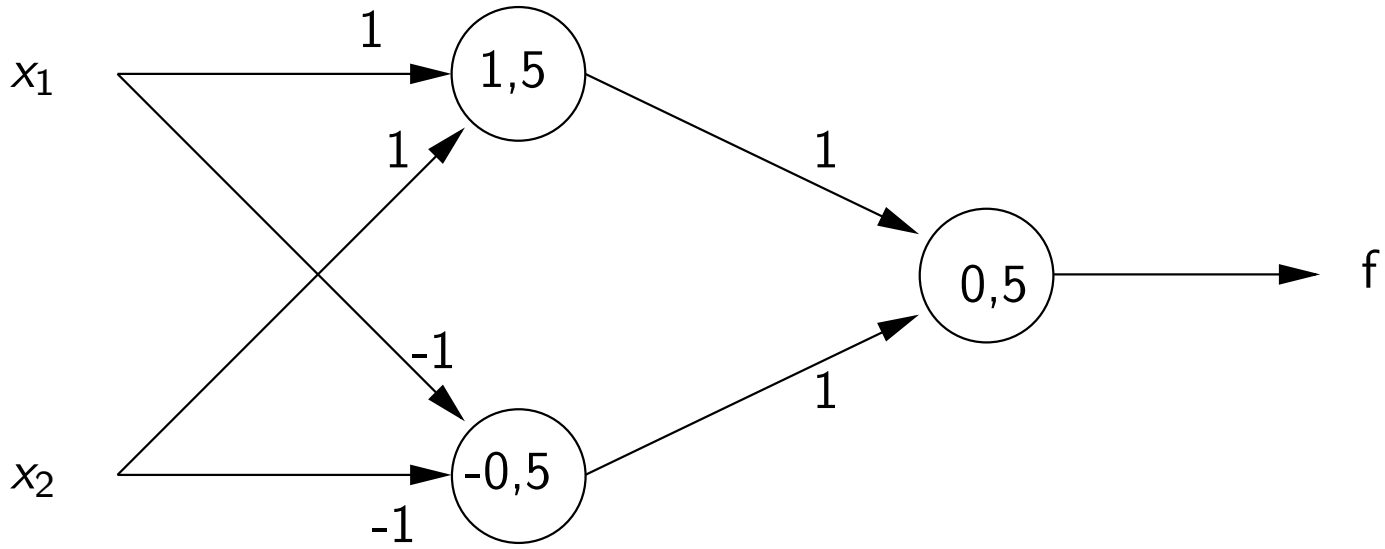
Epoche	x_1	x_2	d	**xw**	y	ε	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2

(Eine Epoche besteht hierbei aus einem Durchlauf aller Lernbeispiele.)

Gliederung der Vorlesung

Mehrschichtige Perzeptrons

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

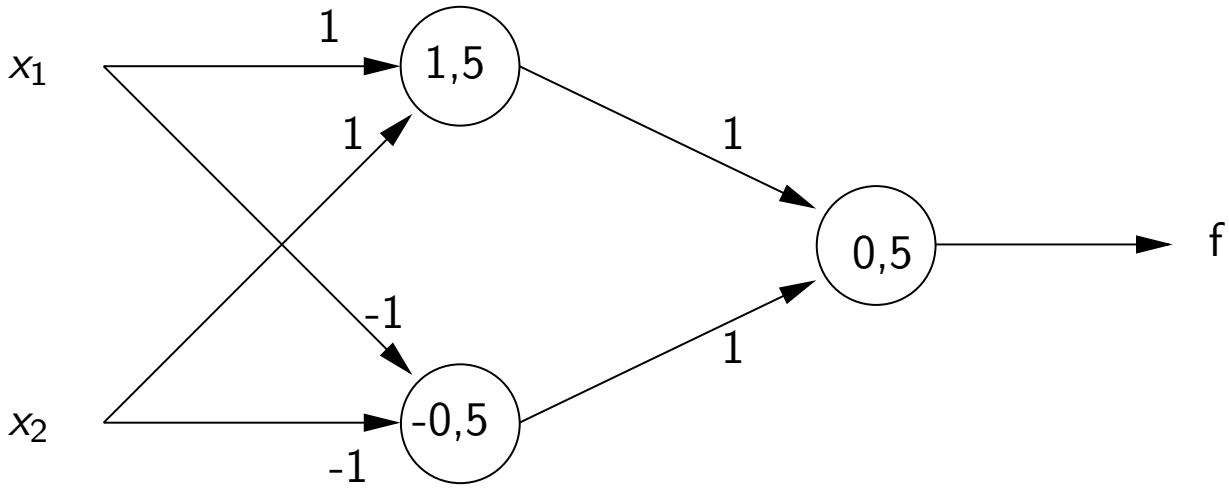


- zweischichtiges Neuronales Netz (falls die Eingabeschicht mitgezählt wird, kann es auch, je nach Literatur, als dreischichtiges Netz bezeichnet werden)
- Neuronen in der mittleren Schicht werden auch als verborgene oder innere Neuronen bezeichnet.

Mehrschichtige Perzeptrons

- 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

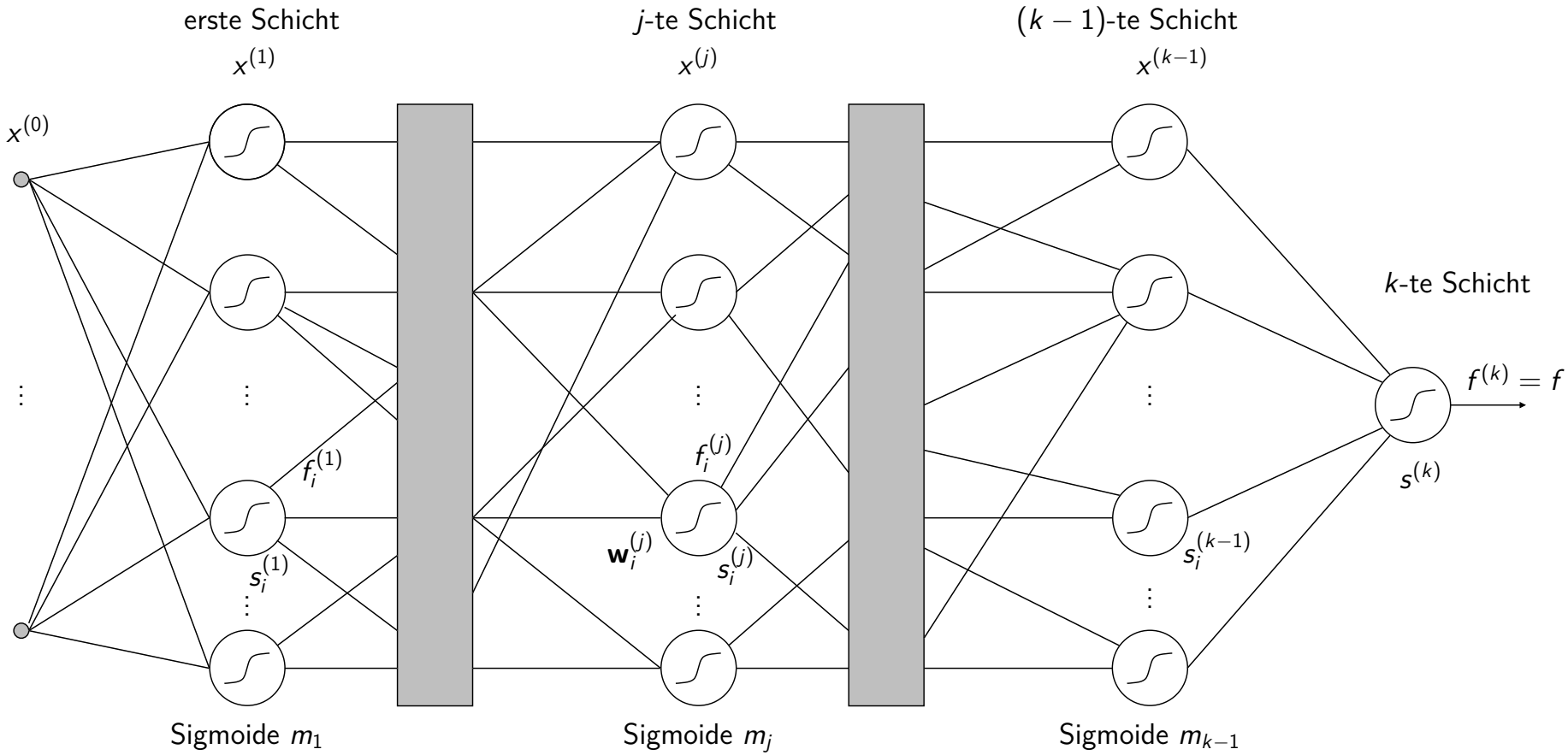


Mehrschichtige Perzeptrons

Definition:

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

Mehrschichtige Perzeptrons



Mehrschichtige Perzeptrons

- Ein spezielles Trainingsverfahren heißt *Backpropagation* (der Fehler wird durch das Netz schichtweise zurückgeschickt).
- *Idee:* Analog zum Gradientenabstieg wird ein Fehlergradient berechnet.
- Herleitung des Verfahrens:
  - Fehler für Ausgabeschicht: $\varepsilon = (d - f)^2$
  - Gradient: $\frac{\partial \varepsilon}{\partial \mathbf{w}_i^j} = \left[\frac{\partial \varepsilon}{\partial w_{1i}^j}, \ldots, \frac{\partial \varepsilon}{\partial w_{li}^j}, \ldots, \frac{\partial \varepsilon}{\partial w_{m_{j-1}+1,i}^j}, \right]$
    wobei w_{li}^j die l-te Komponenten von $\mathbf{w}_i^j$ ist

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 der neuen Gewichte: $(\mathbf{w}_i^j)^{(\text{neu})} := (\mathbf{w}_i^j)^{(\text{alt})} + c_i^j \cdot \delta_i^j \mathbf{x}^{j-1}$,
- Hierbei ist c_i^j eine Lernrate. Meistens wird für c_i^j ein Wert für das gesamte Netz verwendet.

Backpropagation (Herleitung)

- Gewichtsänderungen in der Ausgabeschicht:

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

- Anmerkungen:
  - Gilt bei (logistischer) sigmoider 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:

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

- Der 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 der Index über die einzelnen Vektorelemente)

Backpropagation (Herleitung)

$$\frac{\partial s_l^{j+1}}{\partial s_i^j} = \frac{\partial \left[\sum_{\gamma=1}^{m_{j+1}} f_\gamma^j w_{\gamma l}^{j+1} \right]}{\partial s_i^j}$$

$$= \sum_{\gamma=1}^{m_{j+1}} w_{\gamma l}^{j+1} \frac{\partial f_\gamma^j}{\partial s_i^j}$$

$$= w_{il}^{j+1} f_i^j (1 - f_i^j) \quad \left(\text{da } \frac{\partial f_\gamma j}{\partial s_i^j} = 0 \text{ für } \gamma \neq i, \text{ sonst } \frac{\partial f_\gamma^j}{\partial s_\gamma^j = f_\gamma^j (1 - f_\gamma^j)} \right)$$

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

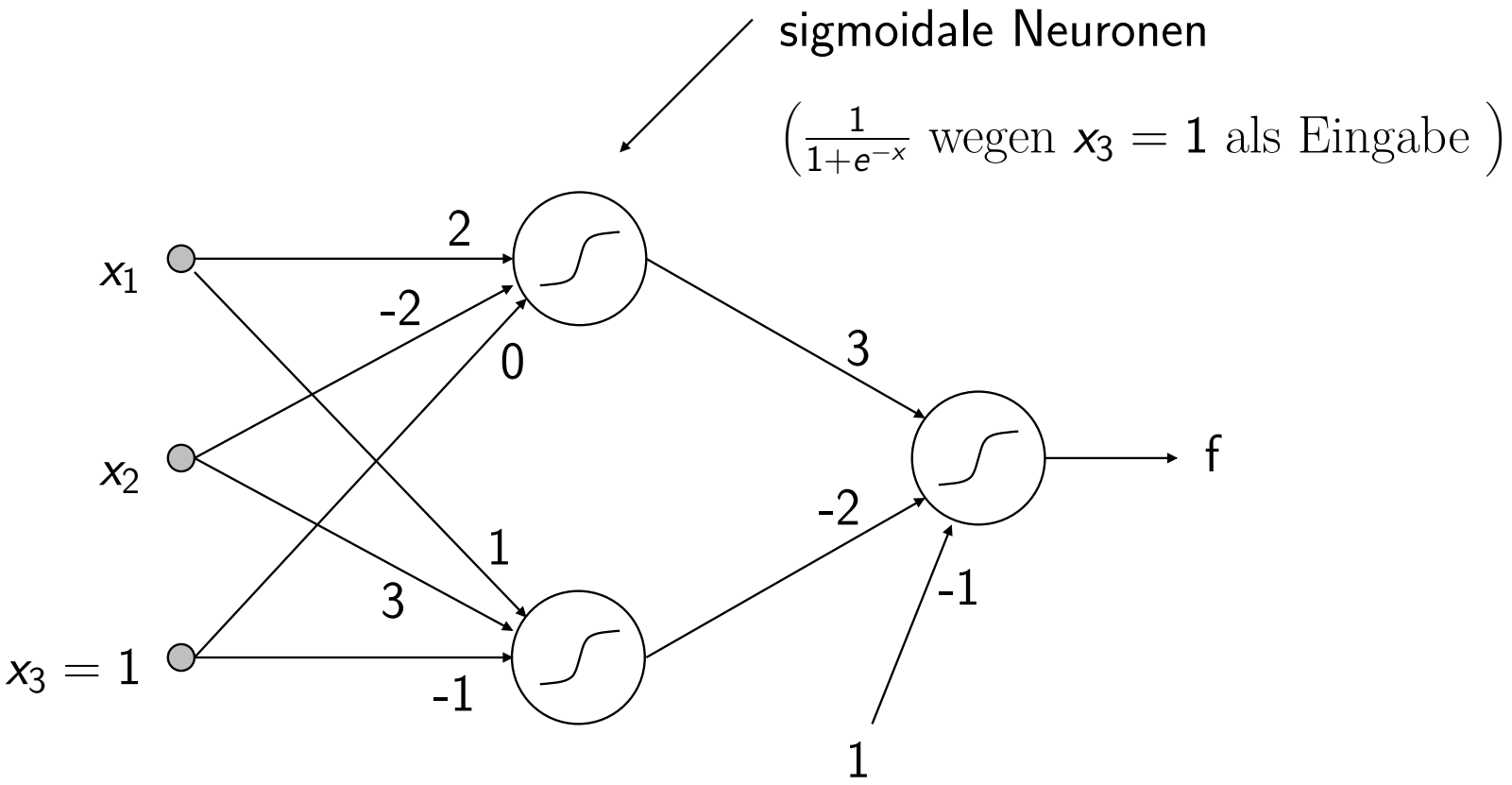
$$\delta_i^j = f_i^j (1 - f_i^j) \sum_{l=1}^{m_{j+1}} \delta_l^{j+1} \cdot w_{il}^{j+1} \text{ und } \delta^k = (d - f) f (1 - f)$$

- Und schließlich zur Berechnung der neuen Gewichte:

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

Backpropagation

- Beispiel (Netz mit zufällig generierten Gewichten):



Backpropagation (Beispiel)

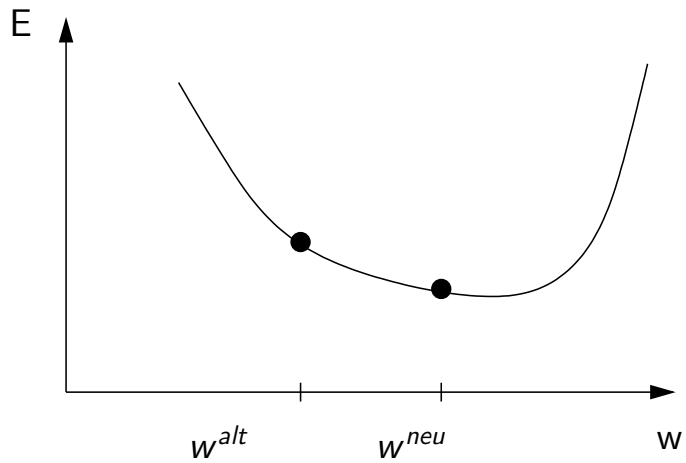
- Lernstichprobe:
  $\{((1,0,1),0)),((0,1,1),0),((0,0,1),1),((1,1,1),1)\}$
- Propagation: $(1,0,1) \rightarrow f_1^1 = 0,881; f_2^1 = 0,5; f = 0,655$
- Backpropagation:
  - Fehlersignale:
    $$\delta^{(2)} = -0,148; d_1^{(1)} = -0,047; d_2^{(1)} = 0,074$$
  - neue Gewichte mit $c = 1$:

$$\begin{aligned} \mathbf{w}_1^{(1)} &= (1,935; -2; -0,047) \\ \mathbf{w}_2^{(1)} &= (1,074; 3; -0,926) \\ \mathbf{w}^{(2)} &= (2,870; -2,074; -1,148) \end{aligned}$$

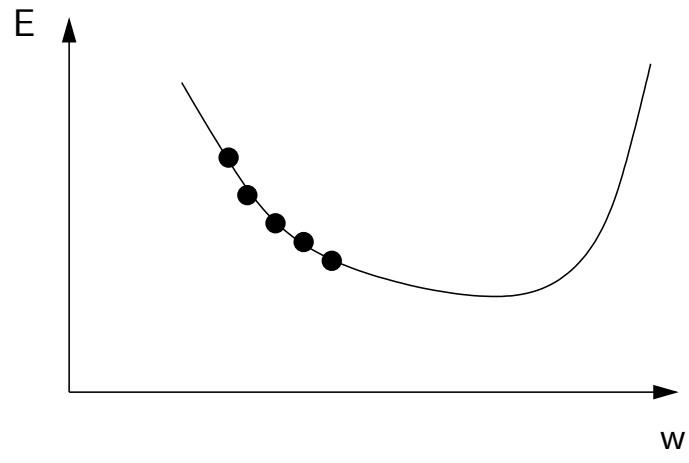
Effekt der Lernrate

- Anmerkungen zum Effekt der Lernrate:

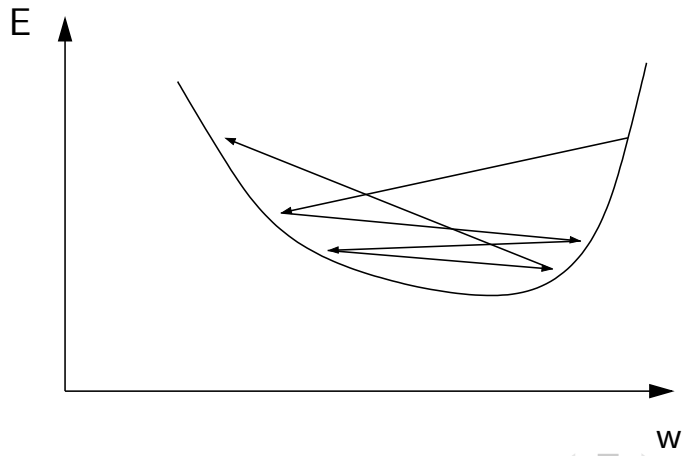
gewünschtes Verhalten:



zu klein: Lernen verläuft zu langsam



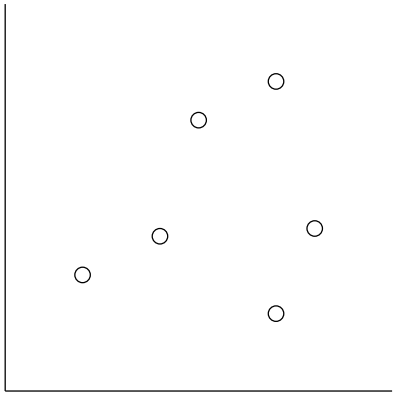
zu groß: Pingpong-Effekt



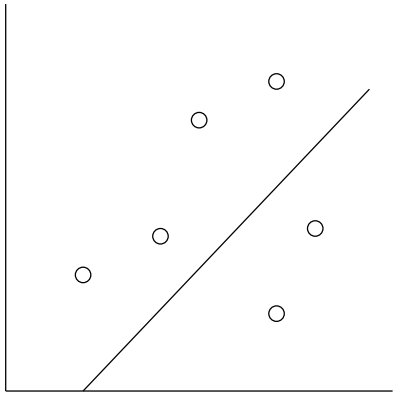
Eigenschaften von Neuronalen Netzen

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

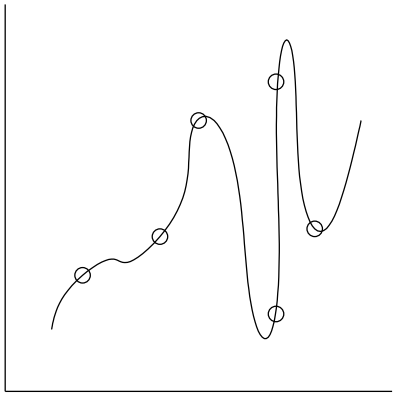
- Analogie: Funktionsapproximation



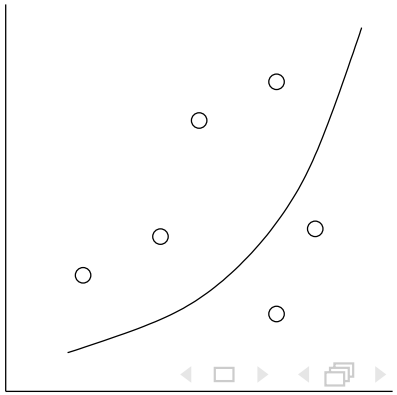
Datensatz



zu einfach,
schlechte Güte



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



Mittelweg
Occam's razor

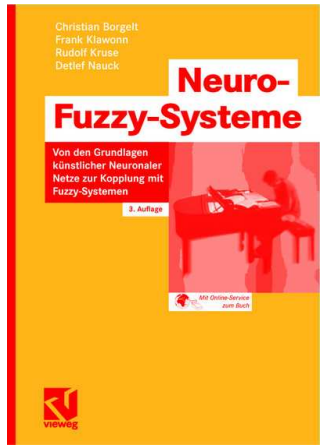
Vorgehensweise beim Lernen

- Lernstichprobe = Trainingsmenge + Validierungsmenge
  - Mit Trainingsmenge (ca. $\frac{2}{3}$ der Daten) lernen.
  - Mit Validierungsmenge Güte schätzen.

- Kreuzvalidierung (*cross validation*)
  - Lernstichprobe in n disjunkte Mengen teilen.
  - Jeweils eine Menge als Validierungsmenge und $n-1$ Mengen zum Training verwenden.
  - Die n out-of-sample errors danach mitteln.

- Beispiele
  - NN werden in verschiedensten Gebieten wie Gesichtserkennung oder Börsenprognosen eingesetzt.
  - NN werden auch im Data Mining genutzt (siehe z.B. unser Artikel in *Spektrum der Wissenschaften*, Nov, 2002, S. 80ff)

weitere Details . . .



Im Sommersemester findet unsere Vorlesung *Neuronale Netze* statt. Dort werden die hier vorgestellten Prinzipien detailliert vorgestellt und weitergehende Themen und Netztypen präsentiert.
Weitere Informationen finden sich auch in unserem Buch [Nauck et al., 2003]:

Literatur I

Nauck, D., Borgelt, C., Klawonn, F., and Kruse, R. (2003).
Neuro-Fuzzy-Systeme — Von den Grundlagen Neuronaler Netze zu modernen Fuzzy-Systemen.
Vieweg, Wiesbaden, Germany.