Neuronale Netze

Uwe Reichel

IPS, LMU München

reichelu@phonetik.uni-muenchen.de

2. Juli 2008

Inhalt

- Einführung
- Neurobiologische Grundlagen
- Neuronenmodell
- Aktivierungsfunktionen
- Lernen
- Netztypen
  - Perzeptron
  - Einschichtige lineare Netze
  - Mehrschichtige Netze
  - Selbstorganisierende Netze
  - Weitere Netztypen

- **Anmerkungen zur Praxis**
  - **Vorverarbeitung**
  - **Generalisierung**
  - **Überspringen nicht-optimaler Minima**
  - **Festlegung der Lernrate**
  - **Alternativen zum Gradientenabstiegsverfahren**
- **Notation**

Einführung

- Computersimulation biologischer Neuronenverbände

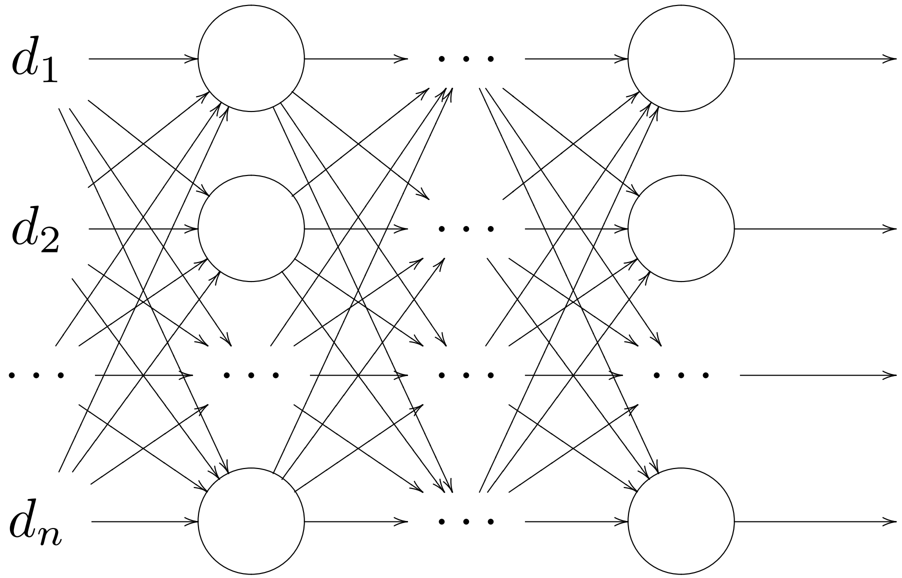

Abbildung 1: Grundarchitektur eines neuronalen (Feed-Forward-) Netzes. Von links nach rechts: Umweltreiz als Merkmalsvektor $\boldsymbol{d}$ codiert. Optionale Hidden-Layers zur Weiterverarbeitung des Reizes. Output-Layer: Antwort des Netzes. (**Feed-Forward:** Erregungsausbreitung in eine Richtung von Input zum Output)

- Input führt zu Aktivitäten im Neuronenverband. Anhand der Aktivität der Ausgabeneuronen lässt sich das Ergebnis der Zielfunktion ablesen (z.B. Klassifikation, numerische Approximation von Funktionen)
- alternative Begriffe: *ANN, konnektionistische Modelle, parallel distributed processing*
- Pioniere: McCulloch & Pitts (1943)
- für **überwachtes** und **unüberwachtes** Lernen einsetzbar
- **Variablentypen**
  - unabhängige Variablen: kategorial (binärcodiert[1]), kontinuierlich
  - abhängige Variable: kategorial, kontinuierlich

[1]Beispiel **Binärcodierung** BC:
POS = {noun, verb, adj, kard}
nötig zur Codierung von 4 Werten: 2 Bit
BC(noun) $\longrightarrow$ 00
BC(verb) $\longrightarrow$ 01
BC(adj) $\longrightarrow$ 10
BC(kard) $\longrightarrow$ 11

Neurobiologische Grundlagen

- Neuronenverbände: Neuronen über **Synapsen** miteinander verbunden

Übermittlung von Umweltreizen

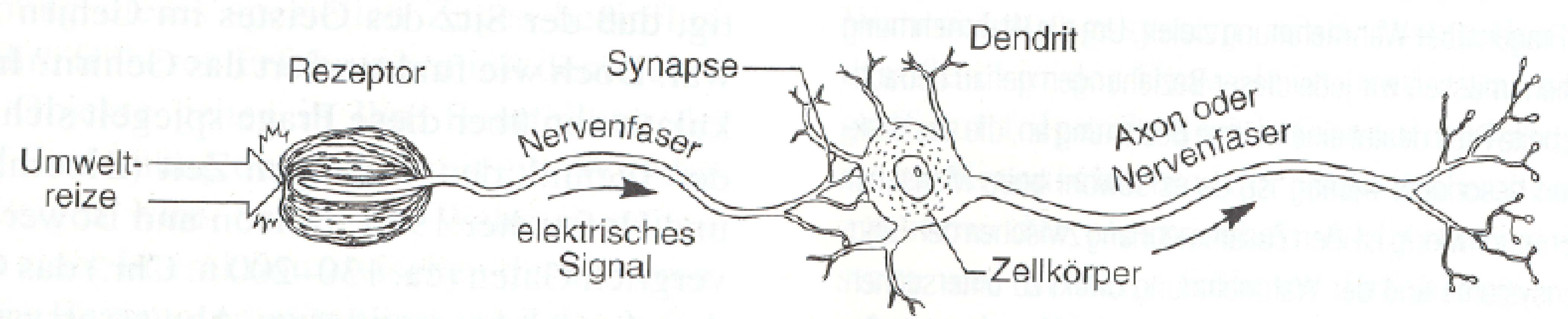

Abbildung 2: Übermittlung von Umweltreizen in das Nervensystem. Rezeptoren sind Nervenzellen, die äußere Reize aufnehmen und weiterleiten.

Informationsübertragung

- **zellintern:** in Form von Aktionspotentialen (AP's; sich fortpflanzende Spannungsänderung an Zellmembran)

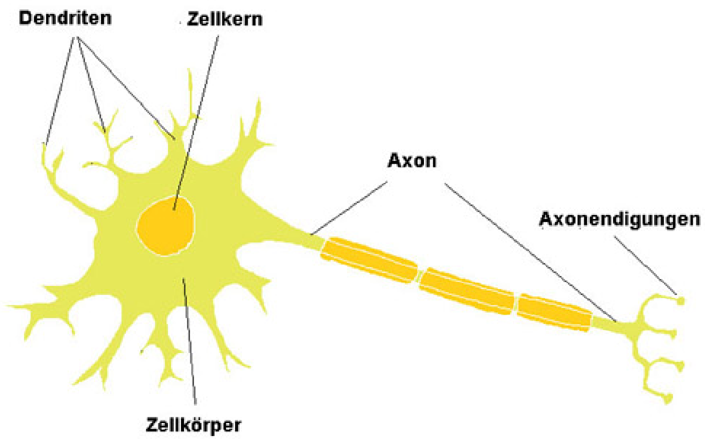

Abbildung 3: *Nervenzelle. Über die Dendriten werden die von anderen Neuronen kommenden Impulse zum* **Zellkörper** *(***Soma***) geleitet. Von dort breiten sie sich über das* **Axon** *weiter zu den nächsten Neuronen aus. Neuronen sind über* **Synapsen** *miteinander verbunden (s.u.). Der* **Zellkern** *im Soma enthält u.a. die Erbinformation, in der der Strukturplan der Zelle enthalten ist. Das Axon ist häufig von einer isolierenden Schicht, der* **Myelinscheide** *umgeben, die an einigen Stellen, den* **Ranvierschen Schnürringen***, unterbrochen ist. Sie sorgt für eine verlustfreie Erregungsleitung.*

- **zwischen Zellen:** über Synapsen; ankommende APs führen in der präsynaptischen Zelle zur Ausschüttung von **Neurotransmittern** in den **synaptischen Spalt**, die an der postsynaptischen Zelle andocken und dort erneut APs auslösen, sofern ein nötiges **Schwellenpotential** überschritten wird.

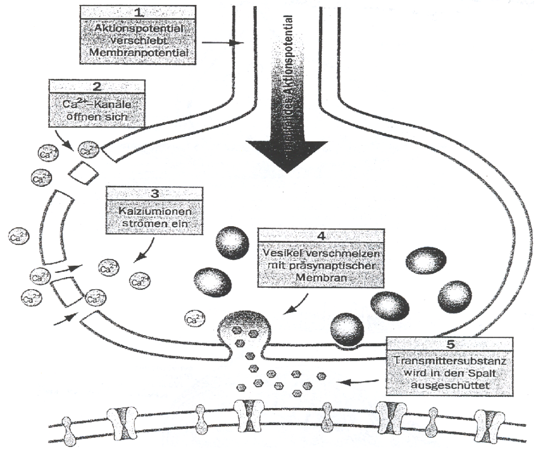

Abbildung 4: Synaptische Übertragung neuronaler Impulse zwischen Neuronen. Die freigegebene Transmittersubstanz dockt an entsprechenden Rezeptoren des postsynaptischen Neurons an und löst dort im Falle einer **exzitatorischen** Verknüfung erneut ein AP aus. Liegt eine **inhibitorische** Verknüpfung vor, so wird das postsynaptische Neuron in seiner Aktivität gehemmt. Zur Beeinflussung des Verhaltens der postsynaptischen Zelle ist wieder eine Schwelle zu überschreiten (Mindestmenge an andockendem Transmitter).

- je niedriger das **Ruhepotential** der postsynaptischen Zelle (Membranpotential im Ruhezustand) und je höher die Erregungsschwelle, desto höhere präsynaptische Aktivität zur Überschreitung der Schwelle nötig.
- **Codierung der Reizstärke:** AP-Frequenz, Menge der freigesetzten Neurotransmitter

- **räumliche Summation:** mehrere präsynaptische Zellen konvergieren an der selben Synapse; Summierung ihrer Aktivitäten bei Auslösung der APs in der postynaptischen Zelle

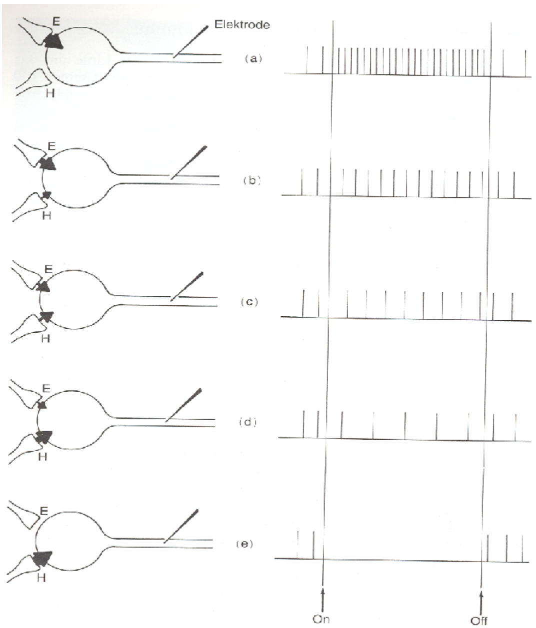

Abbildung 5: Neuronen können durch andere in ihrer Aktivität erregt oder gehemmt werden (**exzitatorische** vs. **inhibitorische** Verbindungen. Münden wie in diesem Fall mehrere präsynaptische Neuronen im selben postsynaptischen Neuron, spricht man von **Konvergenz**.)

- **laterale Hemmung:** inhibitorische Verknüpfung benachbarter Zellen zur **Kontrastverstärkung**

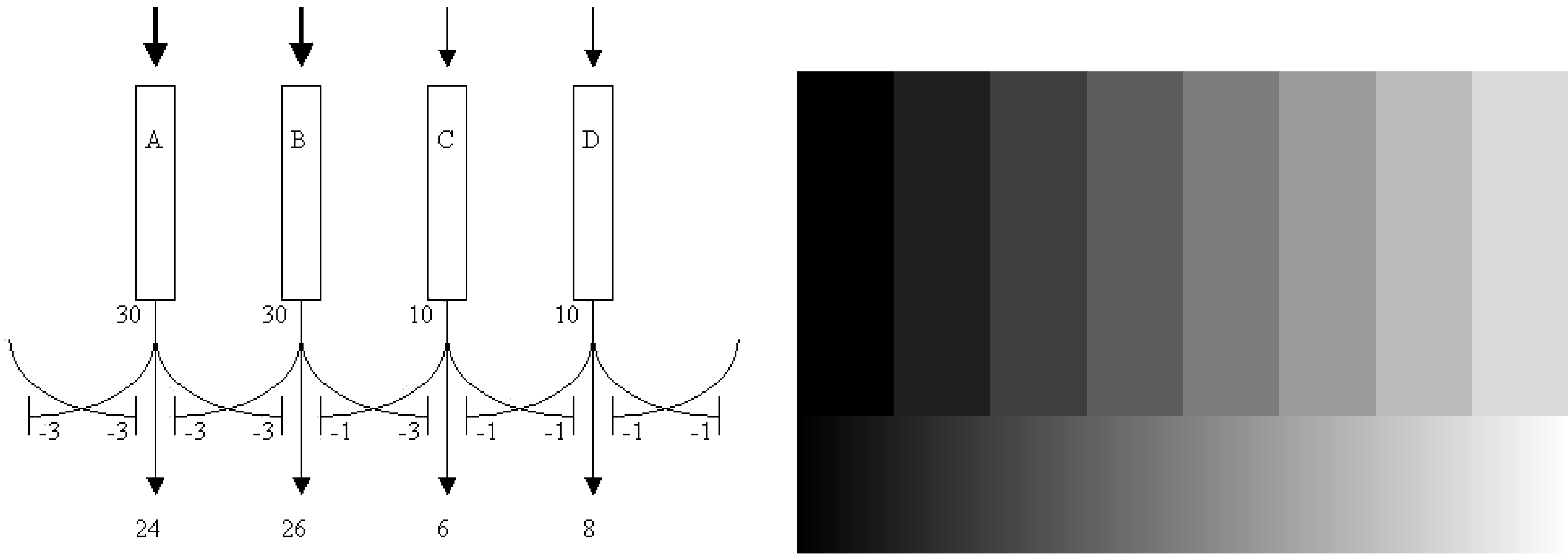

Abbildung 6: Laterale Hemmung benachbarter Rezeptoren (**links**) führt zu Kontrastverstärkung, z.B. in Form von **Mach'schen Streifen** an den Graustufenübergängen (**rechts**).

Lernen:

– basale Lerntypen: **Konditionierung, Sensibilisierung, Adaptierung**

– **Konditionierung, Sensibilisierung:** Stärkung der synaptischen Verbindung zwischen Neuronen, d.h. u.a.: das präsynaptische Neuron entlädt nach dem Lernvorgang eine höhere Menge an Neurotransmittern in den synaptischen Spalt.

– **Adaptierung:** Schwächung der synaptischen Verbindung (Begriff hat andere Bedeutung im Zusammenhang mit ANNs: dort gleich Anpassung an Trainingsdaten!)

Neuronenmodell

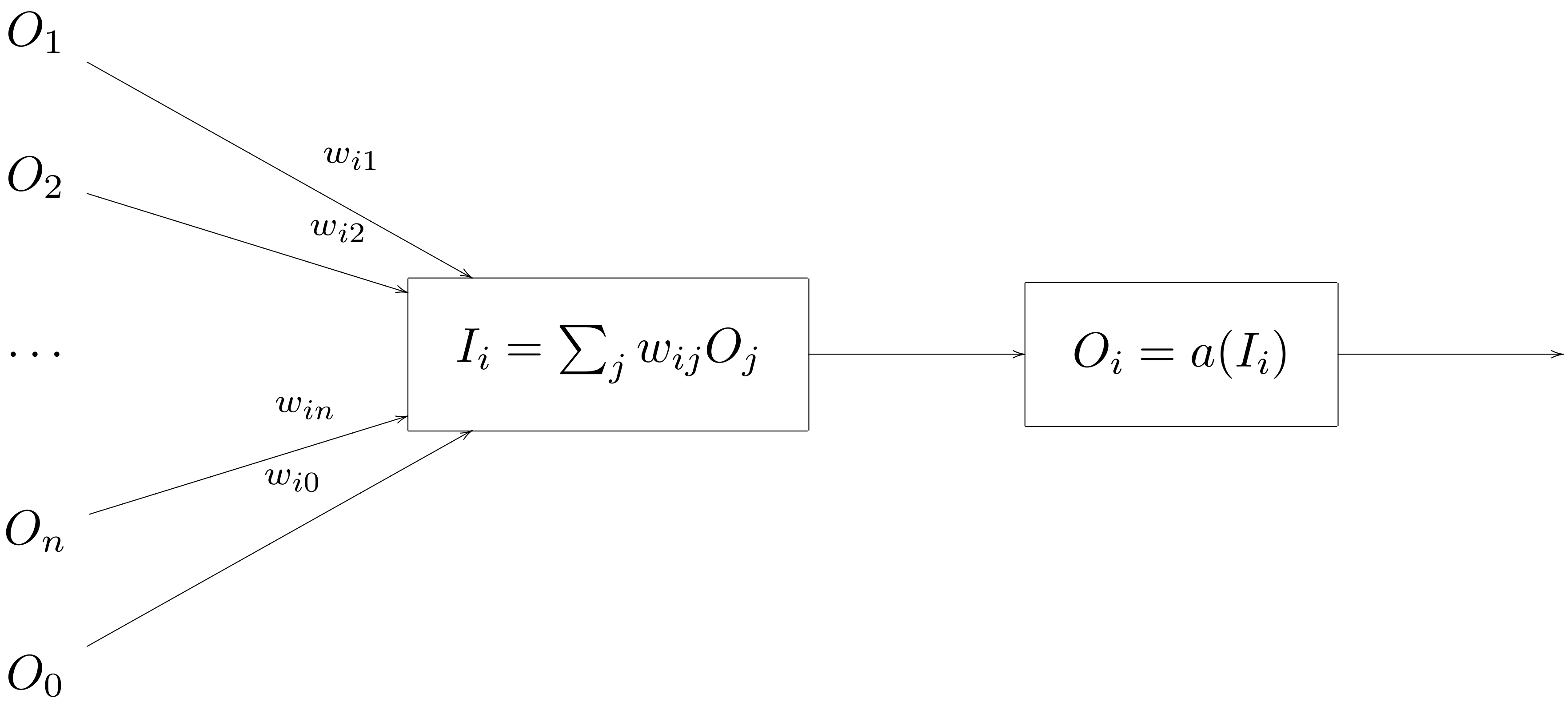

Abbildung 7: Modell eines Neurons.

- **Input des Neurons i:** $I_i = \sum_j w_{ij} O_j$: gewichtete Summe der Aktivitäten der vorgeschalteten Neuronen/ der Reizeigenschaften
- Gewichte w_{ij} = synaptische Verbindungsstärken/ Relevanz einer Reizeigenschaft
- positive/ negative Gewichte für exzitatorische/ inhibitorische Verbindung
- $\sum$: räumliche Summation
- **Bias $w_{i0} O_0$:** "Grundaktivität" θ_i des Neurons i (auch als Kehrwert des **Ruhepotentials** interpretierbar)
- O_j: Aktivität des Neurons j in Abhängigkeit von I_j und der Aktivierungsfunktion a.

Aktivierungsfunktionen

- **binär:**
  hardlim:

$$a(I) = \begin{cases} 1 & : \quad I \geq 0 \\ 0 & : \quad I < 0 \end{cases}$$

  hardlims (Heavyside):

$$a(I) = \begin{cases} 1 & : \quad I \geq 0 \\ -1 & : \quad I < 0 \end{cases}$$

- linear:
  purelin: $a(I) = I$
  satlin:

$$a(I) = \begin{cases} 1 & : \quad I \geq 1 \\ I & : \quad 0 < I < 1 \\ 0 & : \quad I \leq 0 \end{cases}$$

- sigmoid: *tansig*, *logsig* $a(I) = \frac{1}{1+e^{-I}}$

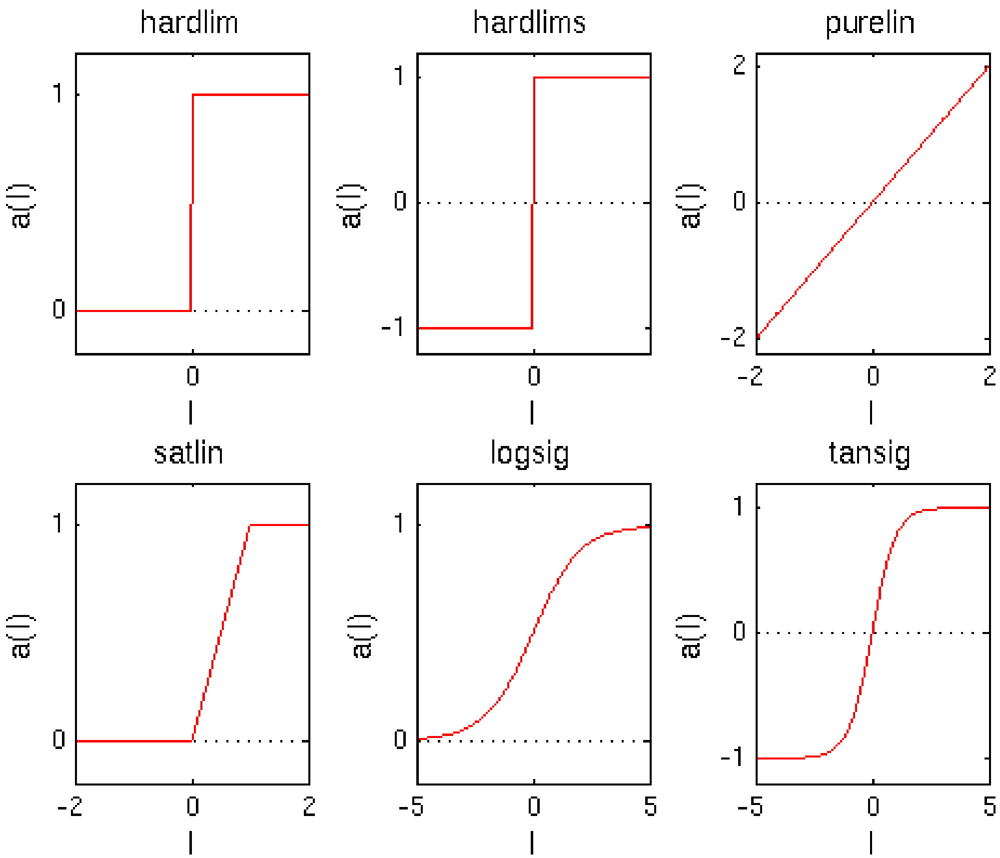

Abbildung 8: Aktivierungsfunktionen

Lernen

- **Wissen:** Gewichte w_{ij} zwischen den Neuronen (als Gewichtsmatrix repräsentierbar)
- **Lernen:** Veränderung der Gewichte w_{ij} zwischen Neuron i und j (Modifizierung der synaptischen Verbindungsstärke):

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} + \Delta w_{ij}^{(n)}$$

(n: Iterationsindex), wobei die Berechnung von Δw_{ij} vom jeweiligen Lernverfahren abhängt.

- Veränderung der Gewichte w_{ij}, um
  - Differenz zwischen beobachtetem und gewünschtem Output zu minimieren (überwachtes Lernen), oder
  - die Sensibilität von Neuronen gegenüber bestimmten Merkmalsvektoren zu erhöhen, bzw. abzuschwächen (unüberwachtes Lernen)
- Gewünschter Output (**Zielfunktion**): bestimmtes Erregungsmuster im Output-Layer

Beispiel: Hebb'sches Lernen

- nach dem Psychologen Hebb (1949): Wenn zwei miteinander verknüpfte Neuronen wiederholt gemeinsam aktiv werden, verstärkt sich die synaptische Verbindung zwischen diesen Neuronen (**assoziatives Lernen, Konditionierung**).
- erhöhe das Gewicht zwischen zwei gemeinsam aktiven Neuronen (Aktivitäten a_i, a_j)
- $\Delta w_{ij} = \eta a_i a_j$
- η: **Lernrate**, mit der das Ausmaß der Gewichtsveränderung festgelegt wird

Aktualiserungsintervalle der Gewichte

- **Batch-Verfahren:** Gewichte werden erst am Ende eines Lerniterationsschritts modifiziert.

$$\Delta w_j^{(n)} = \sum_{\boldsymbol{d}} \Delta_{\boldsymbol{d}} w_j^{(n)}$$

  Modifizierung von Gewicht w_j nach Iterationsschritt n als Summe der Veränderungen für jeden Inputvektor $\boldsymbol{d}$.

- **Online-Training:** Gewichte werden nach jedem Inputvektor modifiziert; hier ist Präsentationsreihenfolge der Inputvektoren von Belang.

Einschichtiges Perzeptron

- Einschichtiges Feed-Forward-Netz mit **Heavyside-Aktivierungsfunktion** für Klassifizierungsaufgaben.

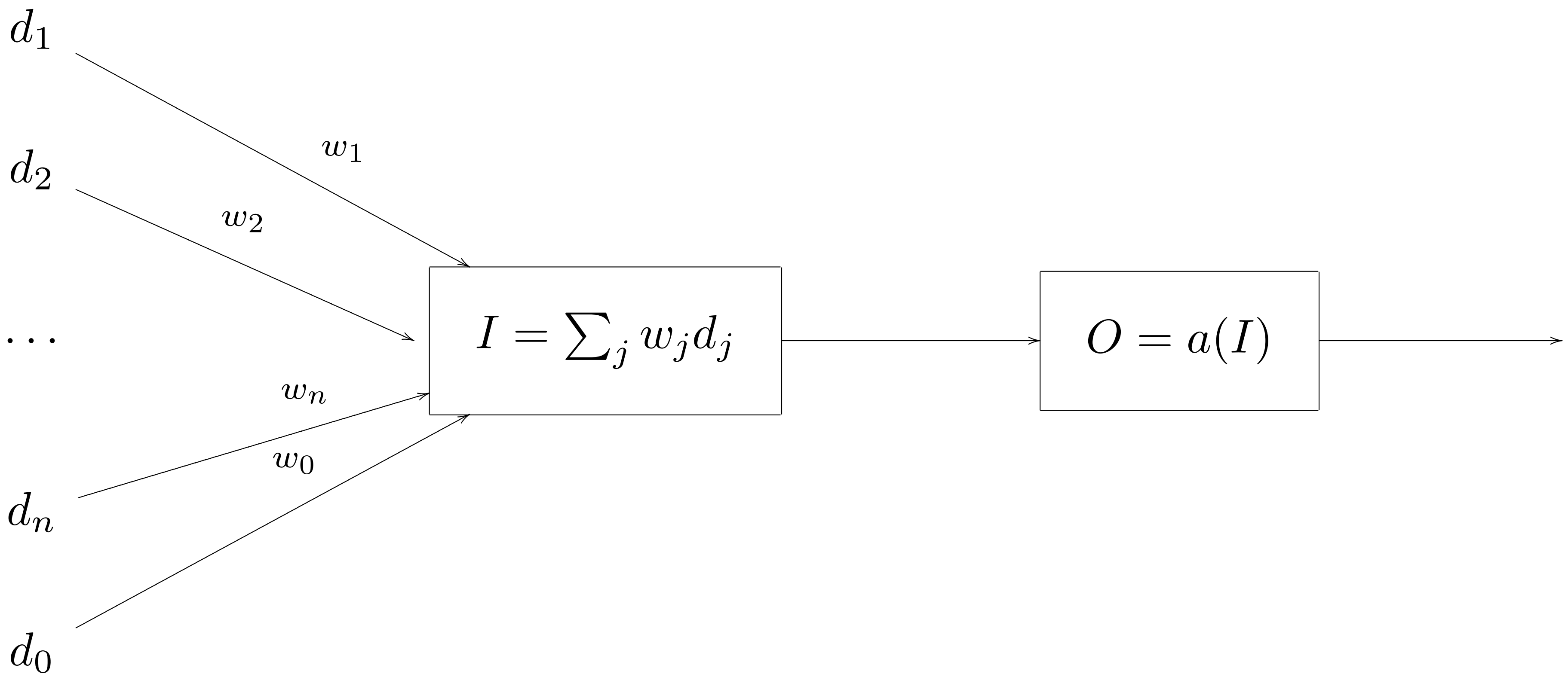

Abbildung 9: Perzeptron-Neuron

- **linearer Klassifikator**: jedes Neuron repräsentiert eine Gerade im Merkmalsraum zur Partitionierung der Menge der Merkmalsvektoren.
- Notation: Merkmalsvektor $\boldsymbol{d}$ setzt sich zusammen aus den Features $[d_1, d_2, \ldots, d_n]$
- Voraussetzung für das Finden einer geeigneten Geraden: Klassen sind **linear separierbar**, also durch eine Gerade trennbar.
- Die Gerade ergibt sich durch die Linearkombination $\sum w_j d_j$ für alle Features d_j.
- Im Falle von Bias $\theta = 0$ geht sie durch den Ursprung ($\theta = w_0 d_0$).

Perzeptron-Lernregel

initialisiere Gewichte $\{w_j\}$ (zufällig)
foreach Vektor $\boldsymbol{d}$ in den Trainingsdaten
  if (Netz gibt falsche Antwort)
    ändere die Gewichte w_j (incl. *Bias*) durch Addition von: $\Delta w_j = t_{\boldsymbol{d}} d_j$,
    wo Zielwert $t_{\boldsymbol{d}} = \{-1, 1\}$
  endif
endforeach

- Konvergenz garantiert, wenn Objekte **linear separierbar** sind.

Einschichtige lineare Netze (Adaline)

- lineare Aktivierungsfunktion
- Vorhersage kontinuierlicher oder kategorialer Werte
- Klassifikation: Entscheidung für die mit dem aktivsten Output-Neuron assoziierte Klasse
- lineare Separierbarkeit kein Konvergenzkriterium mehr
- zu allen mehrschichtigen linearen Netzen gibt es ein äquivalentes einschichtiges lineares Netz.

Lernen: Delta-Regel (Adaline-, Widrow-Hoff-Regel)

- **Ziel:** minimiere an einem Neuron den quadratischen Fehler E zwischen beobachteter $o_{\boldsymbol{d}}$ und gewollter $t_{\boldsymbol{d}}$ Aktivität für alle Vektoren $d \in D$ durch Anpassung des Gewichtsvektors $\boldsymbol{w}$ an der Vektor-Neuron-Schnittstelle.

$$E(\boldsymbol{w}) := \frac{1}{2} \sum_{\boldsymbol{d} \in D} E_{\boldsymbol{d}}(\boldsymbol{w}) \quad := \quad \frac{1}{2} \sum_{\boldsymbol{d} \in D} (t_{\boldsymbol{d}} - o_{\boldsymbol{d}})^2$$

- mittels **Gradientenabstiegsverfahren:** iterative Suche nach einem (lokalen) Minimum der Fehlerfunktion eines Lernproblems durch Abstieg in der Gradientenrichtung anstelle der Prüfung aller Parameterkombinationen (hier: Gewichte $\boldsymbol{w}$.)
- **Gradient**: Vektor der Länge n, der für eine Funktion im n-dimensionalen Raum an einem Punkt die Steigung in jeder der Dimensionen angibt.[2]
- Gradient $\nabla E(\boldsymbol{w})$ beschreibt für die Fehlerfunktion E im Gewichtsraum am Punkt $\boldsymbol{w}$ die Steigung in jeder der Dimensionen (=Anzahl der Gewichte).
- Ermittlung der Steigungen durch **partielle Ableitungen** der Fehlerfunktion
- **partielle Ableitung**: Ableitung einer Funktion mit mehreren Variablen nach einer dieser Variablen, während die anderen Variablen als Konstanten behandelt werden.

[2] **Geometrische Interpretation**: Vektor, der mit seinem Betrag und Winkel Ausmaß und Richtung der stärksten Steigung einer Funktion an einem Punkt angibt.

- $\longrightarrow$ Gradient $\nabla E(\boldsymbol{w})$: Vektor mit partiellen Ableitung der Fehlerfunktion nach allen Gewichten w_j.

$$\nabla E(\boldsymbol{w}) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}]$$

- **Modifikation der Gewichte:** bewege sie in Richtung des lokal steilsten Gefälles im Fehlerfunktionsgebirge

$$\begin{aligned} \Delta \boldsymbol{w} &= -\eta \nabla E(\boldsymbol{w}) \\ \Delta \boldsymbol{w}_j &= -\eta \frac{\partial E}{\partial \boldsymbol{w}_j} \end{aligned}$$

- $-\nabla$, da Gewichte zum Gefälle hin verschoben werden sollen.

- Ermittlung der $\frac{\partial E}{\partial \boldsymbol{w}_j}$ (Herleitung s. Mitchell 1997, S. 92):

$$\frac{\partial E}{\partial \boldsymbol{w}_j} = \eta \sum_{\boldsymbol{d} \in D} (t_{\boldsymbol{d}} - o_{\boldsymbol{d}}) d_j$$

- garantierte Konvergenz in bester Lösung (E minimiert) bei hinreichend kleiner Lernrate η.
- Grund: **quadratische Fehlerfunktion** E (Parabel) besitzt **genau ein Minimum**, das mittels des Gradientenabstiegsverfahrens erreicht wird.
- im Falle linearer Separierbarkeit: $E = 0$

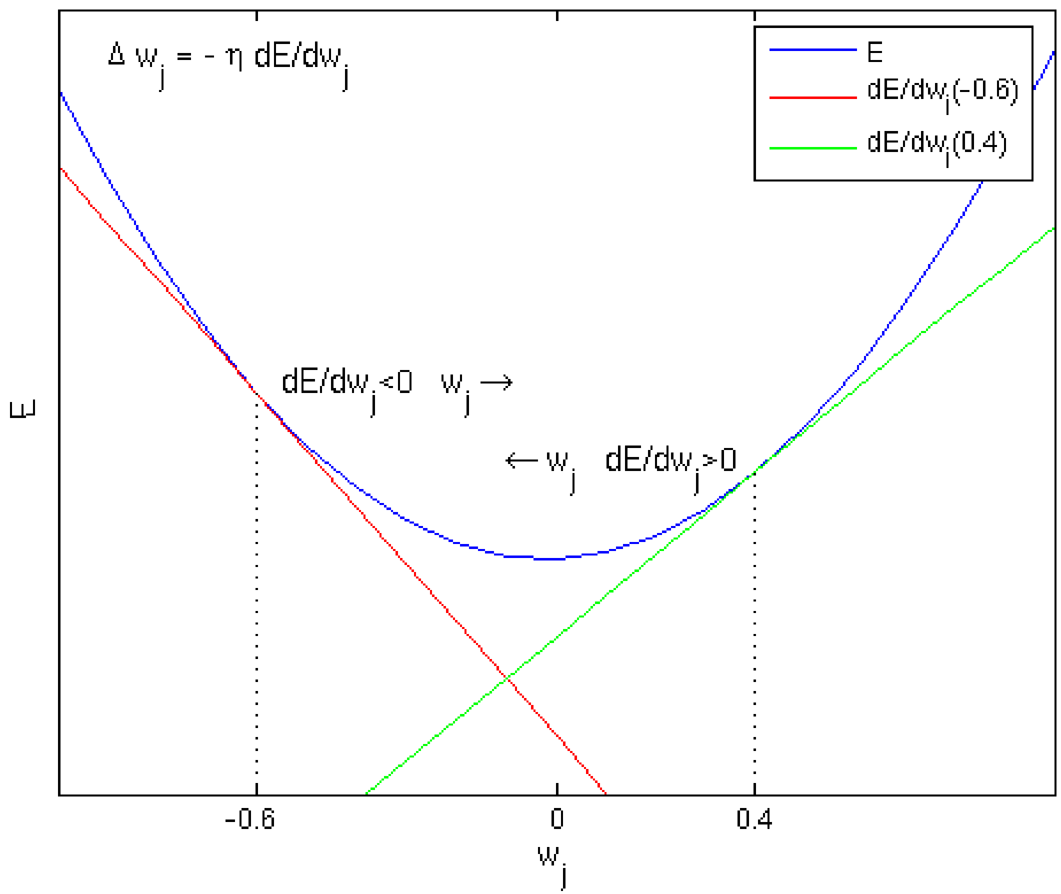

Abbildung 10: Veränderung des Gewichts w_j in Abhängigkeit der Ableitung $\frac{dE}{dw_j}$ der Fehlerfunktion E nach w_j. Ist die Ableitung und damit die Steigung für ein gegebenes w_j negativ, so wird w_j erhöht, ist sie positiv, so wird w_j verringert. Somit landet w_j in einem (lokalen) Minimum der Fehlerfunktion.

Mehrschichtige nichtlineare Netze

Unzulänglichkeit des einschichtigen Perzeptrons: das XOR-Problem

- mögliches **logisches Zusammenwirken** von Features: UND, ODER, EXKLUSIVES ODER (XOR; “entweder nur . . . oder nur . . . ”)
- XOR-Wahrheitswerte

d_1	d_2	$t_{\boldsymbol{d}}$
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

- Beispiel für XOR-bezogene Klassifikationsaufgabe: *Flunder* vs. *sonstige Fische*
  d_1: Auge(n) auf der rechten Körperhälfte
  d_2: Auge(n) auf der linken Körperhälfte

- **Problem:** einschichtige Perzeptrons können für XOR keine lineare Diskrimantionsfunktion repräsentieren: es gibt keine Gerade, die die 4 Punkte mit den Koordinaten d_1 und d_2 aus der XOR-Wahrheitstabelle angemessen trennen kann, d.h. die Punkte korrekt in *Flundern* ($t_{\boldsymbol{d}}$=1) und *sonstige Fische* ($t_{\boldsymbol{d}}$=-1) partitioniert.
- **Lösung:** Hidden Layer

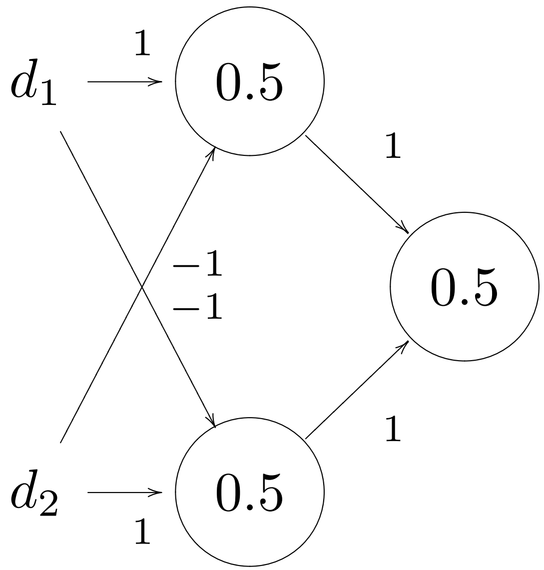

Abbildung 11: Hidden-Layer zur Lösung des XOR-Problems. Mehrschichtiges Perzeptron (Heavyside-Aktivierungsfunktion). In den Knoten sind die Schwellen angetragen, die unter Verwendung des *Bias* eingestellt werden können.

Unzulänglichkeit von linearen Netzen

- Modellierung von Aktivitäts-Schwellen, wie sie in biologischen Neuronenverbänden auftreten, ist nicht möglich.
- Das **universelle Approximierungstheorem** gilt nicht.

Sigmoide Aktivierungsfunktion

- Vorteil gegenüber Heavyside: differenzierbar und daher für Gradientenabstiegverfahren verwendbar
- Vorteil gegenüber linearer Funktion: Modellierung von Schwellen

Universelles Approximierungstheorem

- Jede Funktion kann mit **nicht-linearen Netzen mit einem Hidden Layer** mit beliebiger Genauigkeit approximiert werden.

Lernen mittels Backpropagation

- **Verallgemeinerung der Delta-Regel** für beliebige Aktivierungsfunktionen und beliebig viele Schichten
- **Ziel:** (**inkrementelles** Herangehen):
  für jeden Input $d \in D$: minimiere den Fehler

$$E_d(w) \quad = \quad \frac{1}{2} \sum_{k \in K} (t_k - o_k)^2$$

  in Abhängigkeit der Gewichte w. D bezeichnet die Menge der Merkmalsvektoren und K die Menge der Output-Neuronen.
- **Problem:** direkte Bestimmung von Fehlern in den Hidden Units nicht möglich
- **Lösung:**
  1. Outputlayer: bestimme die Fehler der Output-Neuronen durch Vergleich von Target- und beobachtetem Wert
  2. Hidden-Layer: ermittle für jedes Neuron j den Fehler anhand der Fehler der mit j verbundenen Neuronen in der folgenden Schicht.

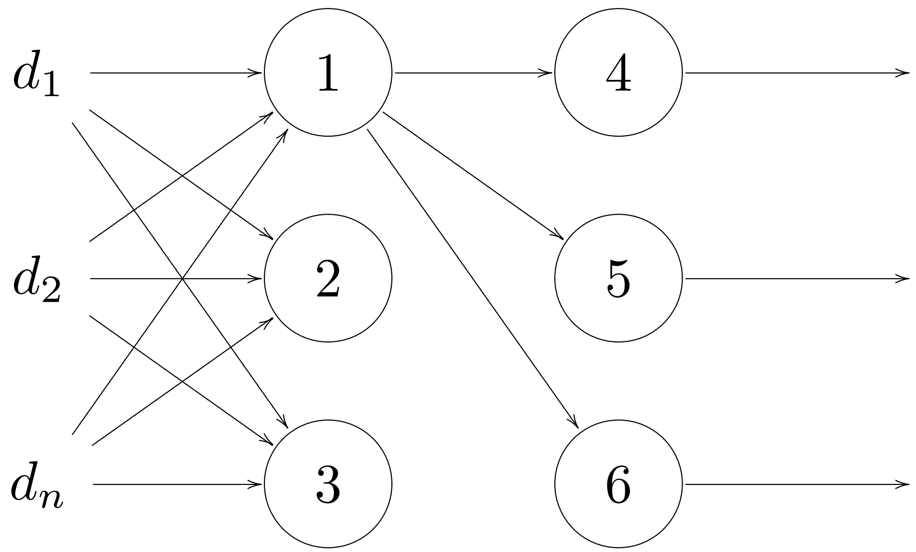

Abbildung 12: Backpropagation: I. Direkte Ermittlung der Fehler der Neuronen **4, 5, 6**. II. Indirekte Ermittlung des Fehler des Neurons 1 anhand der Fehler von **4, 5, 6**.

- **Forward Pass:** Anlegen von Reizen ans Netz, Berechnung der Output-Aktivität
- **Fehlerbestimmung** zwischen gewünschter und beobachteter Aktivität
- **Backward Pass:** iteratives Ausbreitung des Fehlers zurück zur Inputschicht (*Backpropagation*), dabei Adjustierung der Gewichte in Abhängigkeit ihres Einflusses auf den Fehler
- Adjustierung z.B. durch **Gradientenabstiegsverfahren**.

- Fehlerterm eines Neurons k im **Output-Layer:**

$$\delta_k \quad = \quad o_k(1 - o_k)(t_k - o_k),$$

  wo $(t_k - o_k)$ wie bei Delta-Regel, und $o_k(1 - o_k)$: erste Ableitung der sigmoiden Aktivierungsfunktion.

- Fehlerterm eines Neurons h in einem **Hidden-Layer:**

$$\delta_h \quad = \quad o_h(1 - o_h) \sum_{k \in \mathrm{NL}} w_{kh} \delta_k,$$

  wo NL die Menge der Neuronen der folgenden Schicht bezeichnet. Der Fehler ergibt sich also durch Aufsummieren der mit dem jeweiligen w_{kh} gewichteten Fehler an den mit h verbundenen Neuronen k der nächsten Schicht.

- Modifizierung des Gewichts w_{ji} um

$$\Delta w_{ji} = \eta \delta_j z_{ji},$$

  wo $z_{ji} = w_{ji} O_i$ gleich dem Input von Neuron i zu Neuron j.

Selbstorganisierende Netze

- Entwickler: Kohonen, 1995
- verwendbar für **unüberwachtes Lernen**: Clustering, Vektorquantisierung
- Typen:
  - **Kompetitives Lernen**
  - **Selbstorganisierende Karten**
- keine Fehlerfunktion zur Minimierung gegeben
- Lernen durch **Konkurrenz** und **Verstärkung**

Kompetitives Lernen

- **Netzarchitektur:** einschichtig, feed-forward
- **Idee:** suche Neuron i, das durch Input-Vektor $\boldsymbol{d}$ am stärksten aktiviert wird, und passe die Gewichte von i so an, daß seine Aktivierung durch $\boldsymbol{d}$ in Zukunft noch höher ausfällt.
- vorab anzugeben: Anzahl der Neuronen (=Anzahl der zu gewinnenden Cluster, vgl. *kmeans*)
- Höhe der Aktivierung abhängig von **euklidischer Distanz** zwischen Inputvektor $\boldsymbol{d}$ und Gewichtsvektor $\longrightarrow$ der dem Vektor $\boldsymbol{d}$ ähnlichste Gewichtsvektor wird $\boldsymbol{d}$ noch ähnlicher gemacht

Kohonen-Lernregel für Gewichte

- inkrementelles Lernen

1. wähle Inputvektor $\boldsymbol{d}$
2. ermittle Neuron i mit maximaler Erregung ($=$ minimale euklidische Distanz zwischen Gewichtsvektor w_i und $\boldsymbol{d}$)
3. aktualisiere w_i folgendermaßen: $w_i \longleftarrow w_i + \eta(\boldsymbol{d} - w_i)$
4. falls gewünschte Anzahl der Lerniterationen noch nicht erreicht, zurück zu 1 mit reduzierter Lernrate η.

Transferfunktion

$$a(I_i) = \begin{cases} 1 & : \quad i = \arg\min_i[\mathrm{dist}(w_i, \boldsymbol{d})] \\ 0 & : \quad \text{sonst} \end{cases}$$

Bias-Lernregel

- **Problem:** Neuronen, die sich zu Beginn bei keinem der Trainingsvektoren nicht durchsetzen können, können es auch in Zukunft nicht: nutzlose **tote Neuronen**
- **Lösung:** Erhöhe den Bias solcher Neuronen, um sie ins Spiel zu bringen. Reduziere den Bias von Neuronen, die sich häufig durchsetzen.
- Bias-Update eines Neurons in Abhängigkeit seines online ermittelten Output-Mittels. Geringes Output-Mittel $\longrightarrow$ Erhöhung; hohes Ouput-Mittel $\longrightarrow$ Absenkung des Bias.
- $\longrightarrow$ gleichmäßige Aufteilung der Trainingsdaten auf die Neuronen

Selbstorganisierende Karten

- Erweiterung des kompetitiven Lernens um eine Berücksichtigung von Neuronen-Nachbarschaften

Netzarchitektur

- einschichtig, feedforward, Output-Neuronen bilden niedrigdimensionales Gitter

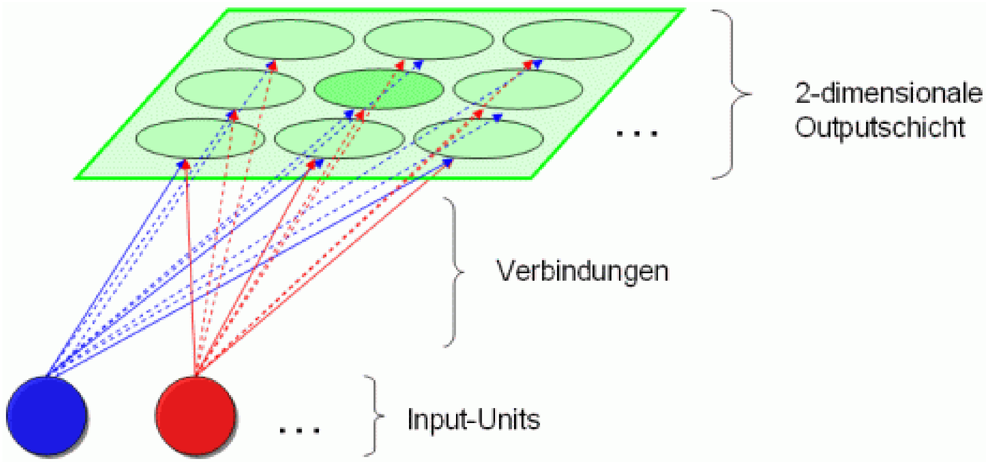

Abbildung 13: Selbstorganisierende Karte

- **Spezifizierung des Gitters**

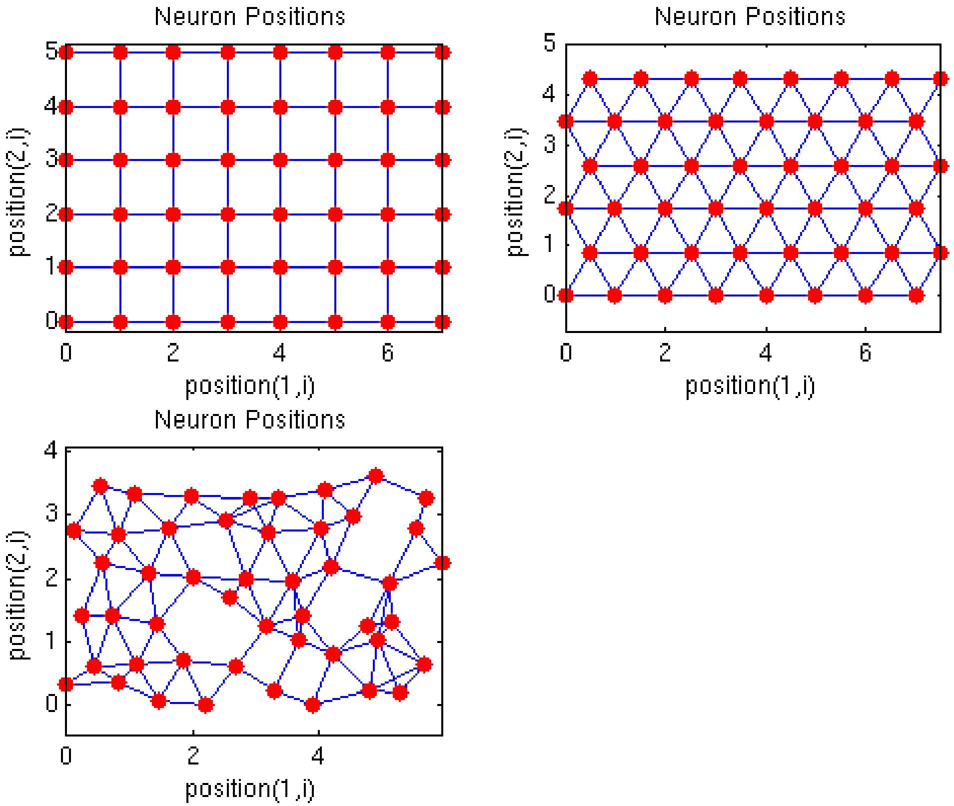

Abbildung 14: **Links oben:** rechteckiges Gitter, **links oben:** hexagonale Anordnung, **links unten:** zufällige Anordung.

Topologieerhaltende Abbildung der Außenwelt

- benachbarte Neuronen sprechen auf benachbarte Merkmalsvektoren am stärksten an (vgl. Tonotopie auf Basilarmembran)
- **erreicht wodurch?** Nachbarschaftsfunktion $n(q_{ij})$, die von der Distanz q_{ij} zwischen den Neuronen i und j im Gitter abhängt und die Modifizierung der Gewichtsvektoren beeinflußt
- $n(q_{ij})$ z.b. Heaviside mit Schwelle (Radius) r:

$$n(q_{ij}) = \begin{cases} -1 & : \quad q_{ij} \geq r \\ 1 & : \quad q_{ij} < r \end{cases}$$

- **Ermittlung der Distanz** q_{ij}
  - Euklid'sche Distanz
  - Manhattan-Distanz
  - Link-Distanz: Anzahl der Kanten auf dem kürzesten Weg von Neuron i zu Neuron j
  - Box-Distanz: Radius (1-D), bzw. halbe Seitenlänge des Quadrats (2-D)/ Würfels (3-D)/ Hypercubus (n-D) um ein Neuron

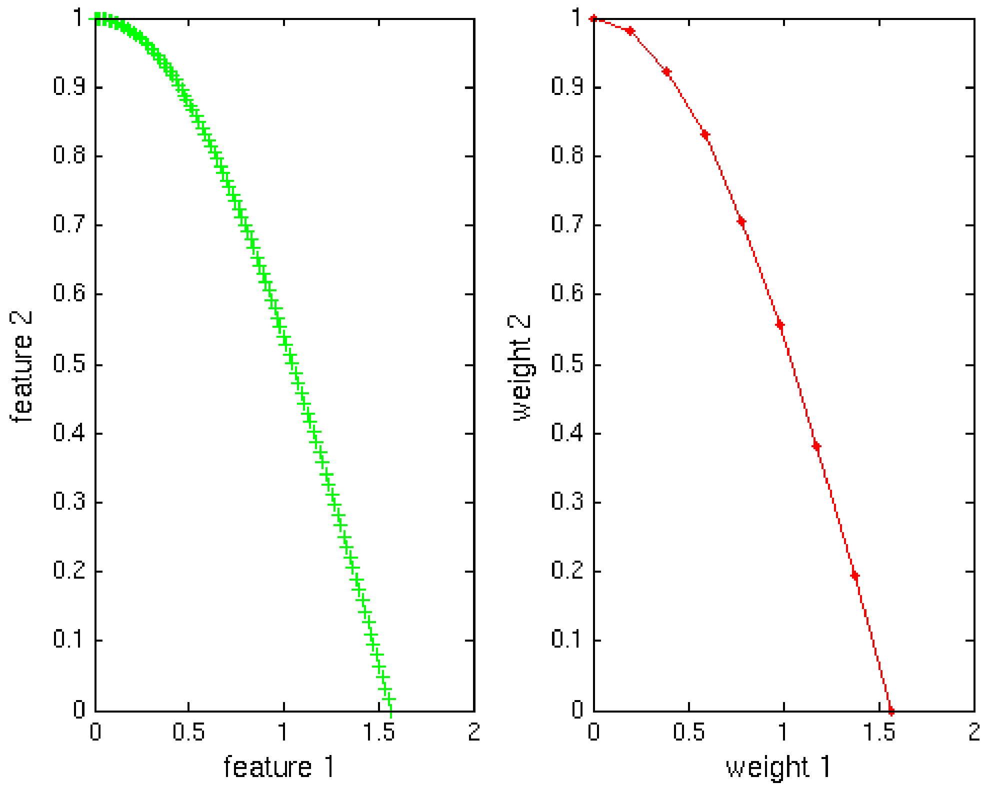

Abbildung 15: **Links:** Anordnung der Trainingsobjekte im Merkmalsraum, **links:** topologieerhaltende Anordnung der Neuronen im Gewichtsraum.

Kohonen-Lernregel für Selbstorganisierende Karten:

- inkrementelles Lernen

1. wähle Inputvektor $\boldsymbol{d}$
2. ermittle Neuron i mit maximaler Erregung ($\longrightarrow$ minimale Distanz q zwischen Gewichtsvektor w_i und $\boldsymbol{d}$)
3. aktualisiere die Gewichtsvektoren w_j aller Neuronen j im Gitter folgendermaßen: $w_j \longleftarrow w_j + \eta n(q_{ij})(\boldsymbol{d} - w_j)$
4. falls gewünschte Anzahl der Lerniterationen noch nicht erreicht, zurück zu 1 mit reduzierter Lernrate η.

- Durch $n(q_{ij})$ werden auch eng benachbarte Neuronen hinsichtlich ihrer Gewichte an $\boldsymbol{d}$ angepasst, während weiter entfernte Neuronen auf $\boldsymbol{d}$ zukünftig schwächer reagieren.

- **Phasen während einer Trainingsepoche:**
  1. **Ordering-Phase:** strukturerhaltende Anordnung der Neuronen im Gewichtsraum bezüglich der Anordnung der Feature-Vektoren im Merkmalsraum.
  2. **Tuning-Phase:** Feinanpassung der Gewichte ohne Veränderung der relativen Anordnung.

Weitere Netztypen

RBF-Netze

- Hidden-Layer mit *Radial basis functions* als Aktivierungsfunktion (Werte nur Abhängig vom Betrag und nicht vom Winkel eines Vektors)
- Output-Layer mit linearen Aktivierungsfunktionen
- Für Klassifikations- und Approximierungsaufgaben

Dynamische Netze, Rekurrente Netze

- Netze mit **Rückkopplung** (**Rekurrenz**) und/ oder **Delays**
- Delay: Reaktion des Netzes nicht nur auf aktuellen sondern auch auf vergangenen Input, z.B. modelliert als Rückkopplung einer Neuronenantwort zum Netzeingang

- **Adalines als adaptive Filter**
  - **Adaptives Filter:** selbständige Veränderung der eigenen Übertragungsfunktion im laufenden Betrieb.
  - Übertragungsfunktion hier repräsentiert durch Gewichtsmatrix.
  - hierfür nötig: **Delay**. Der Input des Netzes zum Zeitpunkt t beinhaltet auch den Input zu den Zeitpunkten $t-n, \ldots, t-1$.
  - **Anwendungen:** Vorhersage von Werten in einem Zeitsignal anhand vorangehender Werte (vgl. **Lineare Prädiktion**)[3] , Rauschunterdrückung.

[3] LP: Vorhersage des n-ten Samples anhand der p vorangegangenen. $\hat{s}[n] = \sum_{k=1}^{p} a_k s[n-k]$

- **Elman-Netz**
  - zweischichtig mit Kontext-Neuronen am Netzeingang zu denen die Einheiten des Hidden-Layers rückkoppeln.
  - Vorhersagen in Zeitreihen, Modellierung von LZI-Systemen
- **Hopfield-Netz**
  - alle Verbindungen sind **symmetrisch**
  - Heavyside-Aktivierungsfunktion
  - z.B. Modellierung von **Top-Down-Prozessen** (vgl. **Trace-Modell**)

Stochastische Netze

- incl. zufälliger Parameter-Variationen (z.B. Gewichtsänderungen)
- **Boltzmann-Maschine**
  - stochastische Entsprechung des Hopfield-Netzes

Anmerkungen zur Praxis

Vorverarbeitung

- Entfernen von **Ausreißern**, **Normalisierung** (*vgl. Clustering-Folien*)
- **Hauptkomponentenanalyse:** Zusammenfassung korrelierter Features zu orthogonalen Hauptkomponenten, die zusammen n % der gesamten Varianz in den Trainingsdaten erklären $\longrightarrow$ Feature-Reduktion, Vermeidung von Überadaption (s.u.)

Wahl der Aktivierungsfunktion

- Abhängigkeit vom Netztyp
  - Perzeptron $\longrightarrow$ *hardlim, hardlims*
  - linear $\longrightarrow$ *purelin*
  - Backpropagation-Netze $\longrightarrow$ differenzierbar (also nicht *hardlim, hardlims*)
- bei Output-Neuronen abhängig von Wertebereich W der Zielwerte
  - $W = [0\ 1] \longrightarrow$ *hardlim, logsig, satlin*
  - $W = [-1\ 1] \longrightarrow$ *hardlims, tansig*
- ggf. Zielwerte entsprechend der gewählten Aktivierungsfunktion normalisieren

Generalisierung

- Methoden zur Vermeidung von **Überadaption** (*Over-Fitting*): **Regularisierung**, **Early Stopping**
- **Regularisierung**
  - modifizierte Fehlerfunktion E_{mod}

$$E_{mod} \quad = \quad \gamma E_{ms} + (1-\gamma) m_{sw},$$

    wo E_{ms} gleich dem mittleren quadratischen Fehler zwischen beobachtetem und erwünschten Output (s.o.), m_{sw} gleich dem Mittelwert der quadrierten Gewichte und Biase im Netz: $m_{sw} = \frac{1}{n}\sum_{j=1}^{n} w_j^2$, und γ gleich dem festzulegenden Performanzverhältnis.
  - m_{sw} sorgt für kleine Gewichte und Biase, was eine **Glättung** der Netzantworten bewirkt und damit einer Überadaption entgegenwirkt.

- **Early Stopping**
  - Aufteilung der Trainingsdaten in Trainings- und Validierungskorpus
  - in jedem Iterationsschritt n:
    - Update der Gewichte anhand der Trainingsdaten.
    - Evaluierung des Updates anhand des Validierungskorpusses, Bestimmung der Güte $V^{(n)}$
    - Abbruch, wenn $V^{(n)} < V^{(n-1)}$

Überspringen nicht-optimaler Minima

- manchmal möglich durch modifizierte Gewichtsaktualisierung
- Aktualisierung von $\Delta w_j^{(n)}$ zum Iterationsschritt n in Abhängigkeit der **vorangegangenen** Gewichtsveränderung $\Delta w_j^{(n-1)}$

$$\Delta \boldsymbol{w}_j{}^{(n)} = -\eta \frac{\partial E}{\partial \boldsymbol{w}_j} + \alpha \Delta \boldsymbol{w}_j{}^{(n-1)}$$

- **Effekt: Trägheitsmoment**, das die Gewichtsveränderung in die Richtung der vorangehenden Aktualisierung lenkt.
- $0 \leq \alpha \leq 1$: Gewichtung des Trägheitsmoments

Lernrate η

- klein $\longrightarrow$ geringfügige Aktualisierungen, lange Dauer bis zur Konvergenz
- groß $\longrightarrow$ Gefahr, dass lokale Minima übersprungen werden; Oszillation
- Lösung: Verkleinerung von η im Laufe der Iteration
- nicht einfach bestimmbar, i.d.R. $0.01 \leq \eta \leq 0.05$
- wähle bei Gradientenabstiegsverfahren für Batch-Lernen kleinere η-Werte als für inkrementelles Lernen, da Gradienten vor Gewichtsänderung aufsummiert werden.

Alternativen zum Gradientenabstiegsverfahren

- schnellere Konvergenz, weniger speicheraufwändig
- Resilient Backpropagation
- Conjugate Gradient-Methoden
- Quasi-Newton-Algorithmen

Notation

I_i	Input für Neuron i
θ (auch $w_0 \cdot d_0$, $w_0 \cdot O_0$)	Bias
$a(I)$	Aktivierungsfunktion
O_i	Output (Aktivität) des Neurons i
$\boldsymbol{w}$	Gewichtsvektor (bei Betrachtung eines Neurons) Gewichtsmatrix (bei Betrachtung eines Netzes)
w_{ij}	Gewicht zwischen Neuron j (sendend) und Neuron i (empfangend)
w_j	Gewicht des j-ten Inputs bei Betrachtung eines bestimmten Neurons Gewichtsvektor des Neurons j bei Betrachtung eines Netzes
Δw	Gewichtsveränderung
$\Delta w^{(n)}$	Gewichtsveränderung im n-ten Iterationsschritt
η	Lernrate
$\boldsymbol{d}$	Input-Merkmalsvektor
d_j	j-ter Featurewert im Vektor $\boldsymbol{d}$
$\boldsymbol{D}$	Menge aller $\boldsymbol{d}$
$t_{\boldsymbol{d}}$	gewünschter Ausgabewert (Zielwert, Target) zu Inputvektor d
$o_{\boldsymbol{d}}$	beobachteter Ausgabewert zu Inputvektor d
$E_{\boldsymbol{d}}$	Fehler zwischen $t_{\boldsymbol{d}}$ und $o_{\boldsymbol{d}}$
E	Gesamtfehler