10. Künstliche Neuronale Netze

auch: *Konnektionismus*; *subsymbolische Wissensverarbeitung*

informationsverarbeitende Systeme, bestehen aus meist großer Zahl einfacher Einheiten (Neuronen, Zellen)

- einfache Einheiten senden Information in Form von "Aktivierung" über gerichtete Verbindungen (connections, links) an andere einfache Einheiten
- (teilweise) Ähnlichkeit mit erfolgreichen biologischen Systemen
- massiv parallele Verarbeitung und Lernfähigkeit
- "subsymbolischer" Ansatz der KI zur Erstellung intelligenter Systeme

Leistungsvergleich menschliches Gehirn vs. Rechner (nach Hermes 2002)

	Gehirn	*Rechner*
Anzahl der Verar-beitungselemente	ca 10^{11} Neuronen	ca. 10^{9} Transistoren
Art der Verarbeitung	massiv parallel	i.allg. seriell
Speicherung	assoziativ	adressbezogen
Schaltzeit eines Elements	ca. 1 ms (10^{-3} s)	ca. 1 ns (10^{-9} s)
Schaltvorgänge pro Sekunde	ca. 10^{3}	ca.10^{9}
Schaltvorgänge pro Sekunde insges. (theoretisch)	ca. 10^{13}	ca. 10^{18}
Schaltvorgänge pro Sekunde insges. (praktisch)	ca. 10^{12}	ca. 10^{10}

"100-Schritt-Regel" (als Argument für die Notwendigkeit massiver Parallelverarbeitung):

ein Mensch erkennt eine/n (ihm bekannte/n) Person / Gegenstand in ca. 0,1 s (nach Messungen von Psychologen). Bei einer Schaltzeit von ca. 1 ms entspricht das maximal 100 Verarbeitungsschritten.
Keine Aussage über die Anzahl der Verarbeitungselemente!

Was kann wohl ein von-Neumann-Rechner in 100 Schritten (Assemblerbefehlen) tun?

Vorteile von künstlichen neuronalen Netzen:

- Lernfähigkeit: knN werden (meist) nicht programmiert, sondern trainiert
- Parallelität: vom Ansatz her schon parallel, daher auch Eignung zur Implementierung auf Parallelrechnern
- verteilte Wissensrepräsentation: Wissen ist in lokal definierten Gewichtsfunktionen gespeichert
- Fehlertoleranz: durch die verteilte Repräsentation kann ein knN bei Ausfall einzelner Neuronen eine höhere Fehlertoleranz besitzen als herkömmliche Systeme
- assoziative Speicherung: es können auch ähnliche Muster abgerufen werden
- Robustheit gegen Störungen und Rauschen: bei richtigem Trainung reagieren knN meist weniger empfindlich
- spontane Generalisierung: knN bilden oft automatisch Prototypen von Eingabemustern – auch wenn Eingabe unvollst., kann knN etwas erkennen
- aktive Repräsentation: Wissen ist aktiv (durch die Gewichte) repräsentiert, d.h. kein Programm greift von außen auf Wissen zu und Gewichte sind unmittelbar an der Verarbeitung von Input (Anfragen) beteiligt

Nachteile:

- Wissenserwerb nur durch Lernen: es kann kein Basiswissen mitgegeben werden
- keine Introspektion: keine Analyse des eigenen Wissens und seines Zustandekommens, im Gegensatz zur Erklärungskomponente eines Expertensystems
- logisches (serielles) Schließen schwierig, keine Inferenzketten unmittelbar programmierbar
- Lernen ist relativ langsam: alle verbreiteten Lernverfahren für knN lernen sehr langsam
- beim Lernen können kontingente Zusammenhänge mitgelernt werden, was manchmal nicht erwünscht ist

Historischer Abriss:

Anfänge: Anfang der 40er bis Mitte der 50er Jahre

- McCulloch & Pitts (1943) zeigten, dass auch einfache Klassen von knN prinzipiell jede arithmetische oder logische Funktion berechnen können
- McCulloch & Pitts (1947): Erkennen räumlicher Muster
- 1949 Hebb'sche Lernregel: einfaches universelles Lernkonzept, bis heute noch Basis fast aller knN-Lernverfahren
- Lashley (1950): durch Versuche an Ratten wurde die These über eine verteilte Wissensrepräsentation im Gehirn erhärtet

Das erste "Hoch" (Mitte der 50er bis Ende der 60er Jahre)

- Rosenblatt, Withman et al. (1957-58): erste erfolgreiche Entwicklung eines Neurocomputers am MIT für Mustererkennungs-Aufgaben (20×20 Pixel Bildsensor für einfache Ziffernerkennung), funktionierte mit 512 motorgetriebenen Potentiometern (je ein Poti für ein variables Gewicht)
- Selfridges Pandämonium (1958): dynamische, interaktive Mechanismen zur Morse-Code-Übersetzung
- Widrow & Hoff (1960): ADALINE-System für schnelles Lernen

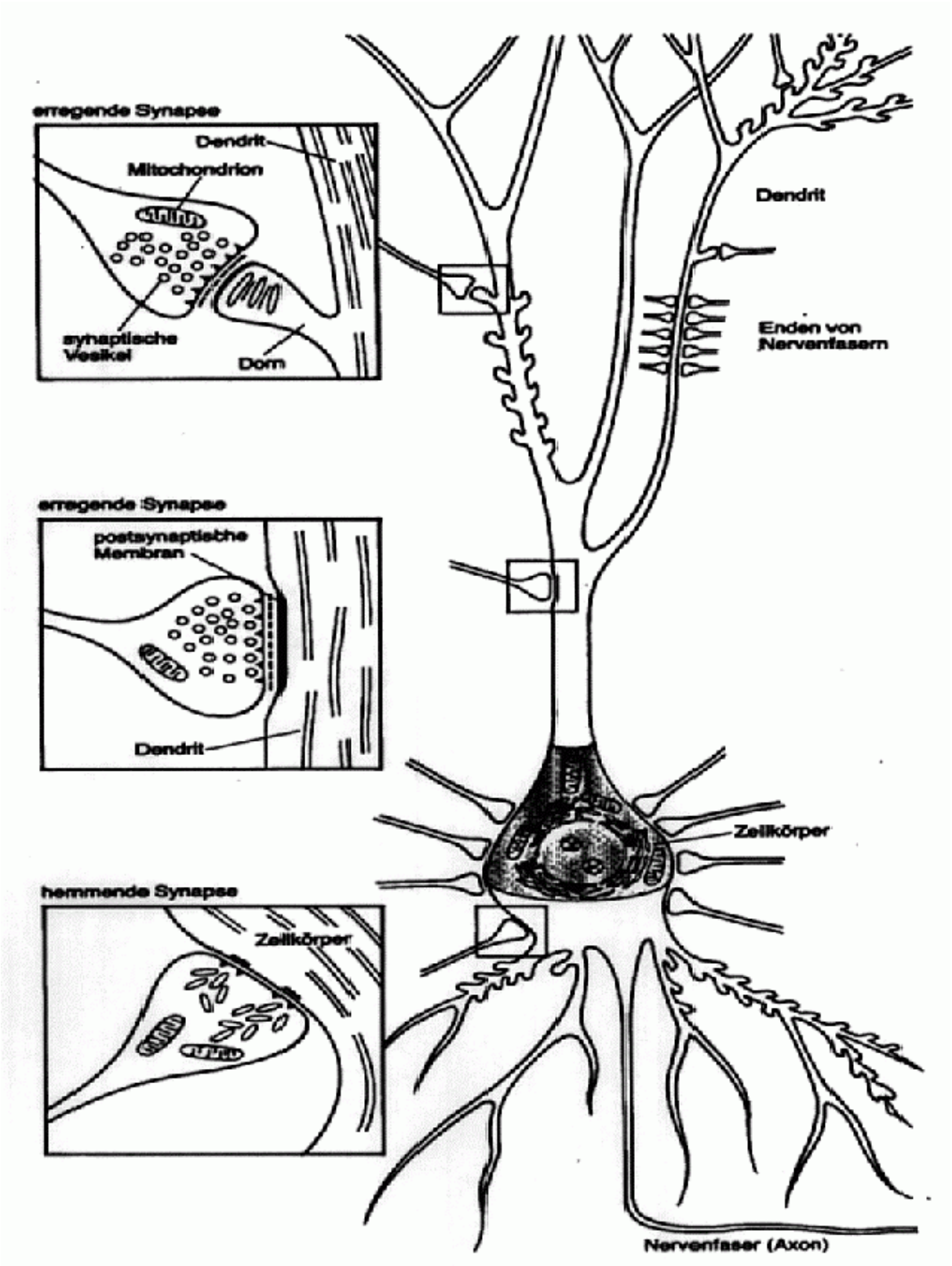
- Steinbuch (1961): Lernmatrix – einfache technische Realisierung von Assoziativspeicher, Realisierung für "Pawlowsche Reflexe"

"Ruhige Phase": Ende der 60er bis Anfang/Mitte der 80er

- Minsky & Papert (1969): genaue math. Analyse von Perzeptrons; zeigten, dass bestimmte Probleme (XOR-Problem) mit diesem Typ von knN nicht lösbar, da nicht repräsentierbar sind; Aussage: knN-Forschung "dead end"; führte dazu, dass fast keine Forschungsgelder mehr flossen (für ca. die nächsten 15 Jahre!), auch DARPA förderte nicht mehr.
- Kohonen (1972): Correlation Matrix Memories
- von der Malsburg (1973): nichtlineares Neuronenmodell
- Grossberg (1976-80): einer der ersten, die sigmoide Aktivierungsfunktionen verwendeten; ART (Adaptive Resonance Theory)
- McClelland & Rumelhart (1981): Constraint satisfaction networks zur Buchstaben- und Worterkennung
- Hopfield (1984): knN in Äquivalenz zu Ising-Gittermodellen in der Physik

Wiederaufleben (ab Anfang / Mitte der 80er Jahre bis heute)

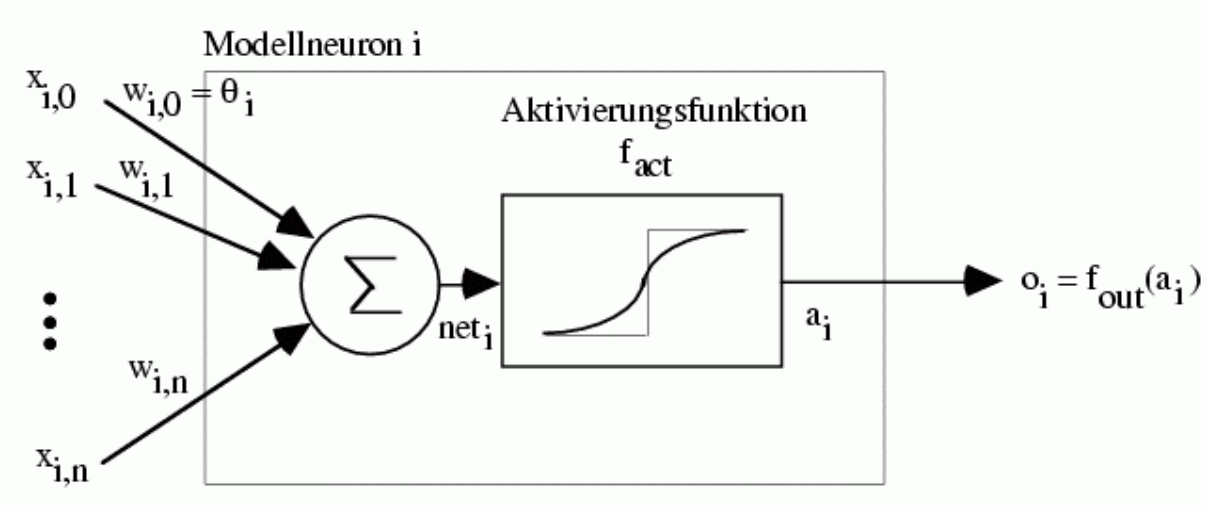
- Hopfield (1985): Einsatz von knN für Optimierung
- ca. 1985 (Wieder-) Entwicklung des Backpropagation Lernverfahrens (Rumelhart, Hinton & Williams; Parker, Le Cun) – Algorithmus war schon 1974 von Werbos entdeckt worden, war aber wieder in Vergessenheit geraten



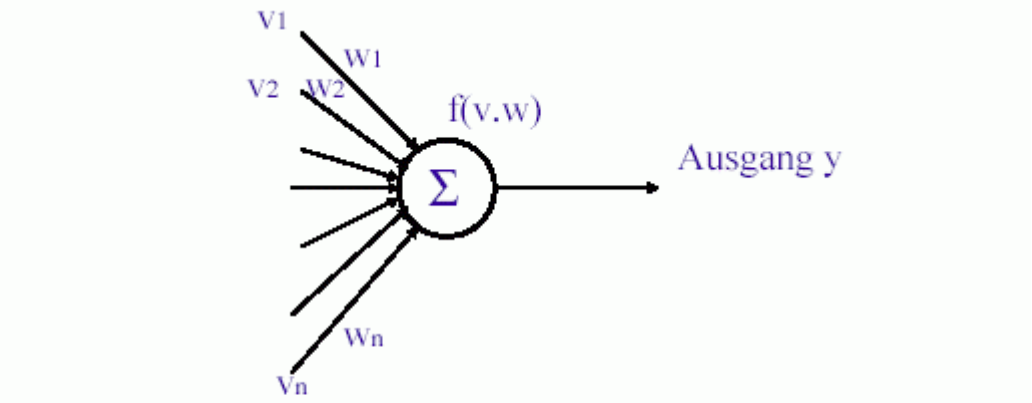
"echte" Nervenzelle (aus Levi 2002)

Modellneuron:

Dendriten und Synapsen $\rightarrow$ gewichtete Verbindungen w_{ij}
Hemmung (Inhibition) $\rightarrow$ $w_{ij} < 0$
Erregung (Exzitation) $\rightarrow$ $w_{ij} > 0$
keine Verknüpfung $\rightarrow$ $w_{ij} = 0$
Zellkörper $\rightarrow$ Aktivierungszustand a_i, Aktivierungsfunktion f_{act}
Axon $\rightarrow$ Ausgabefunktion f_{out}



vereinfacht:



Gesamteingang:

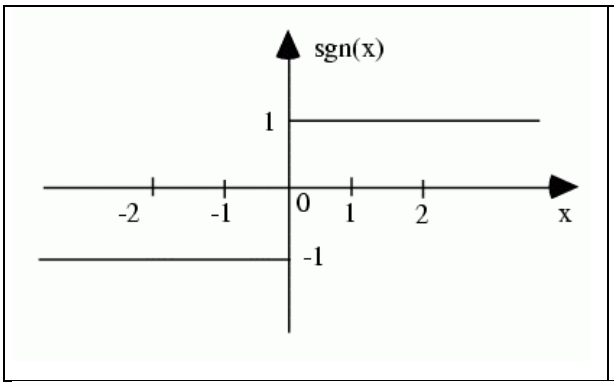
$$x = \sum_{i=1}^{n} v_i w_i \qquad x = \sum_{i=1}^{n} v_i w_i - \theta$$

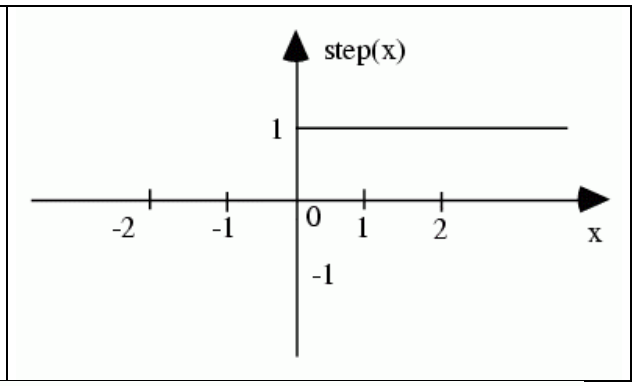
Transferfunktion:

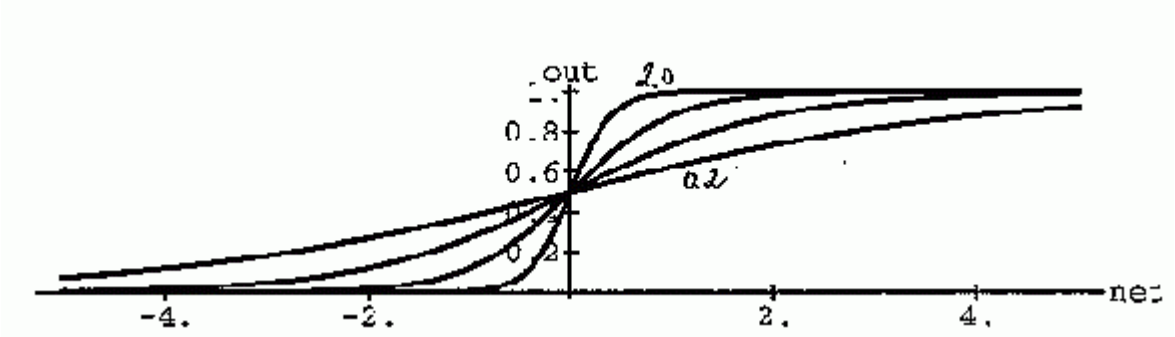
$$f(x) = \begin{cases} 0 & \text{if } x \le 0 \\ 1 & \text{if } x > 0 \end{cases} \qquad f(x) = \frac{1}{1+e^{-x}}$$

(Beispiele)

häufig verwendete Aktivierungs- (Transfer-) Funktionen:

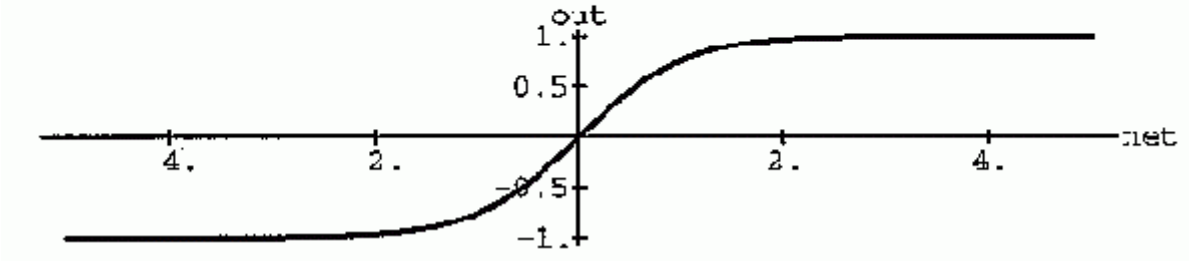




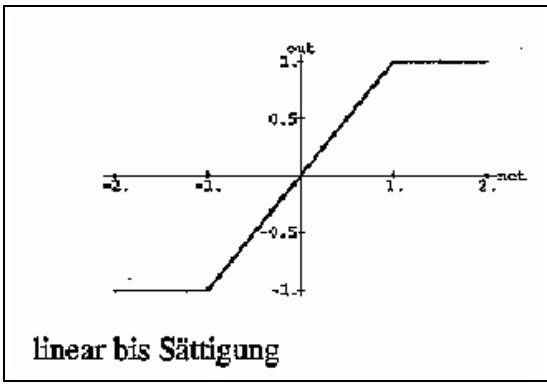


Logistische Funktion $f_{act}(\beta x) = 1/(1+exp(-\beta x))$

$$f_{act} = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Tangens hyperbolicus



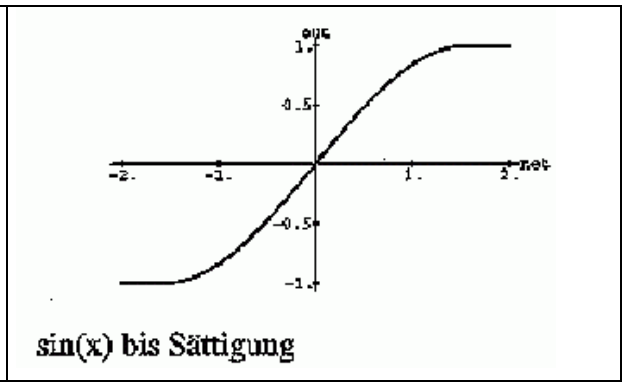


Tabelle 2-2 Eingangsfunktionen

Typ	Kennwerte	Formel
Skalarprodukt	w_j Gewichte	$\varepsilon = we = \sum_{j=1}^{n} w_j e_j$
Sigma-Pi	w^0, w_j^1, w_{jk}^2,... Gewichte	$\varepsilon = w^0 + \sum_{j=1}^{n} w_j^1 e_j + \sum_{j,k=1}^{n} w_{jk}^2 e_j e_k + \sum_{j,k,l=1}^{n} w_{jkl}^3 e_j e_k e_l + \ldots$

Tabelle 2-3 Aktivierungsfunktionen

Typ	Kennwerte	Formel
lineare Aktivierungs-funktion	s Skalierungsfaktor $s > 0$	$c = s\varepsilon$
BSB-Aktivierungs-funktion	s Skalierungsfaktor $s > 0$ d Abklingkonstante $0 < d \leq 1$ c_0 Ruhewert	$c(t+1) = c(t) + s\varepsilon - d\,[c(t) - c_0]$
DMA-Aktivierungs-funktion	s Skalierungsfaktor $s > 0$ d Abklingkonstante $0 < d \leq 1$ c_0 Ruhewert m Minimum M Maximum $m < c_0 < M$	$c(t+1) =$ $\begin{cases} c(t) + s\varepsilon\left[c(t) - m\right] - d\left[c(t) - c_0\right] & \text{für } \varepsilon < 0 \\ c(t) + s\varepsilon\left[M - c(t)\right] - d\left[c(t) - c_0\right] & \text{für } \varepsilon \geq 0 \end{cases}$
Hopfield	m *Minimum* $m = -1$ oder $m = 0$ je nach Modell	$c(t+1) = \begin{cases} m & \text{für } \varepsilon < 0 \\ c(t) & \text{für } \varepsilon = 0 \\ 1 & \text{für } \varepsilon > 0 \end{cases}$

Tabelle 2-4 Ausgangsfunktionen

Typ	Kennwerte	Formel
Schwellenwert-funktion	m Minimum M Maximum ϑ Schwelle	$a(c) = \begin{cases} m & \text{für } c < \vartheta \\ M & \text{für } c \geq \vartheta \end{cases}$
Fermi-Funktion	m Minimum M Maximum ϑ Schwelle σ Steigung	$a(c) = m + \frac{M-m}{1+\exp\left(-4\sigma\frac{c-\vartheta}{M-m}\right)}$
Sinus-Ausgangs-funktion	m Minimum M Maximum ϑ Schwelle σ Steigung	s. Gl. 2-10
Rampenfunktion	m Minimum M Maximum ϑ Schwelle σ Steigung	$a(c) = \begin{cases} m & \text{für } c-\vartheta < -\frac{M-m}{2\sigma} \\ \sigma(c-\vartheta)+\frac{m+M}{2} & \text{für } -\frac{M-m}{2\sigma} \leq c-\vartheta \leq \frac{M-m}{2\sigma} \\ M & \text{für } c-\vartheta > \frac{M-m}{2\sigma} \end{cases}$
Lineare Schwellenwert-funktion	m Minimum ϑ Schwelle σ Steigung	$a(c) = \begin{cases} m & \text{für } c < \vartheta \\ \sigma(c-\vartheta)+m & \text{für } c \geq \vartheta \end{cases}$
Lineare Aus-gangsfunktion	ϑ Schwelle σ Steigung	$a(c) = \sigma(c-\vartheta)$
Boltzmann-Aus-gangsfunktion	ϑ Schwelle T Tempera-tur	$\mathrm{P}(a=1) = \frac{1}{1+e^{-(c-\vartheta)/T}}$
Wettbewerbs-Ausgangs-funktion	m Minimum M Maximum	$a(c) = \begin{cases} M & \text{für } c = \max\limits_k c_k \\ m & \text{für } c < \max\limits_k c_k \end{cases}$

Tabelle 2-5 Spezielle Ausgangsfunktionen

Typ	Grundform	Kennwerte	Formel
Stufen-funktion	Schwellenwert-funktion	$m = 0$ $M = 1$ $\vartheta = 0$	$a(c) = \Theta(c) = \begin{cases} 0 & \text{für } c < 0 \\ 1 & \text{für } c \geq 0 \end{cases}$
Signum-Funktion	Schwellenwert-funktion	$m = -1$ $M = 1$ $\vartheta = 0$	$a(c) = \operatorname{sgn}(c) = \begin{cases} -1 & \text{für } c < 0 \\ 1 & \text{für } c \geq 0 \end{cases}$
einfache Fermi-Funktion	Fermi-Funktion	$m = 0$ $M = 1$ $\vartheta = 0$ $\sigma = 1/4$	$a(c) = \frac{1}{1 + e^{-c}}$
tanh	Fermi-Funktion	$m = -1$ $M = 1$ $\vartheta = 0$ $\sigma = 1$	$a(c) = \tanh c = \frac{1 - e^{-2c}}{1 + e^{-2c}} = \frac{e^{c} - e^{-c}}{e^{c} + e^{-c}}$
einfache Sinus-Funktion	Sinus-Ausgangs-funktion	$m = -1$ $M = 1$ $\vartheta = 0$ $\sigma = 1$	$a(c) = \begin{cases} -1 & \text{für } c < -\frac{\pi}{2} \\ \sin c & \text{für } -\frac{\pi}{2} \leq c \leq \frac{\pi}{2} \\ 1 & \text{für } c > \frac{\pi}{2} \end{cases}$

(aus Hoffmann 1993)

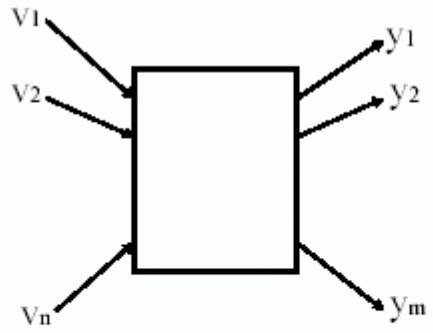
Tabelle 2-6 Standard-Neurontypen

Typ	Ausgangs-Wertebereich	Formel	Eingangsfunktion	Aktivierungsfunktion	Ausgangsfunktion
McCulloch-Pitts	{0,1}	$a = \Theta(\varepsilon)$	Skalarprodukt	Identität	Stufenfunktion
ADALINE	{–1,1}	$a = \operatorname{sgn} \varepsilon$	Skalarprodukt	Identität	Signum-Funktion
Linear	$(-\infty,+\infty)$	$a = \sigma(\varepsilon - \vartheta)$	Skalarprodukt	Identität	Linear
Fermi	(0,1)	$a = \frac{1}{1+e^{-\varepsilon}}$	Skalarprodukt	Identität	einfache Fermi-Funktion
tanh	(–1,1)	$a = \tanh \varepsilon$	Skalarprodukt	Identität	tanh
BSB	[–1,1]	s. Abschn. 4.3.2	Skalarprodukt	BSB	Rampenfunktion
DMA	[–1,1]	s. Abschn. 4.3.3	Skalarprodukt	DMA	Identität
Hopfield	{0,1} (oder {–1,1})	$a(t+1) = \begin{cases} 0 & \text{für } \varepsilon < 0 \\ a(t) & \text{für } \varepsilon = 0 \\ 1 & \text{für } \varepsilon > 0 \end{cases}$	Skalarprodukt	Hopfield	Identität
Boltzmann	{0,1}	$P(a=1) = \frac{1}{1+e^{-(\varepsilon-\vartheta)/T}}$	Skalarprodukt	Identität	Boltzmann

(Hoffmann 1993)

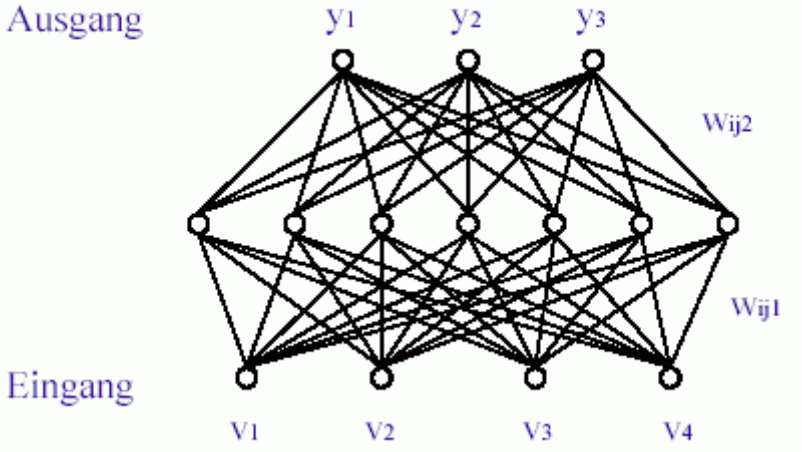
Neuronales Netz:

- Zusammenschaltung mehrerer Neuronen
- Ausgang eines Neurons wird Eingang eines anderen Neurons
- imitiert die hohe Anzahl von Verbindungen einfacher Neuronen im menschlichen Hirn
- Netzwerk besteht aus mehreren Eingängen und mehreren Ausgängen

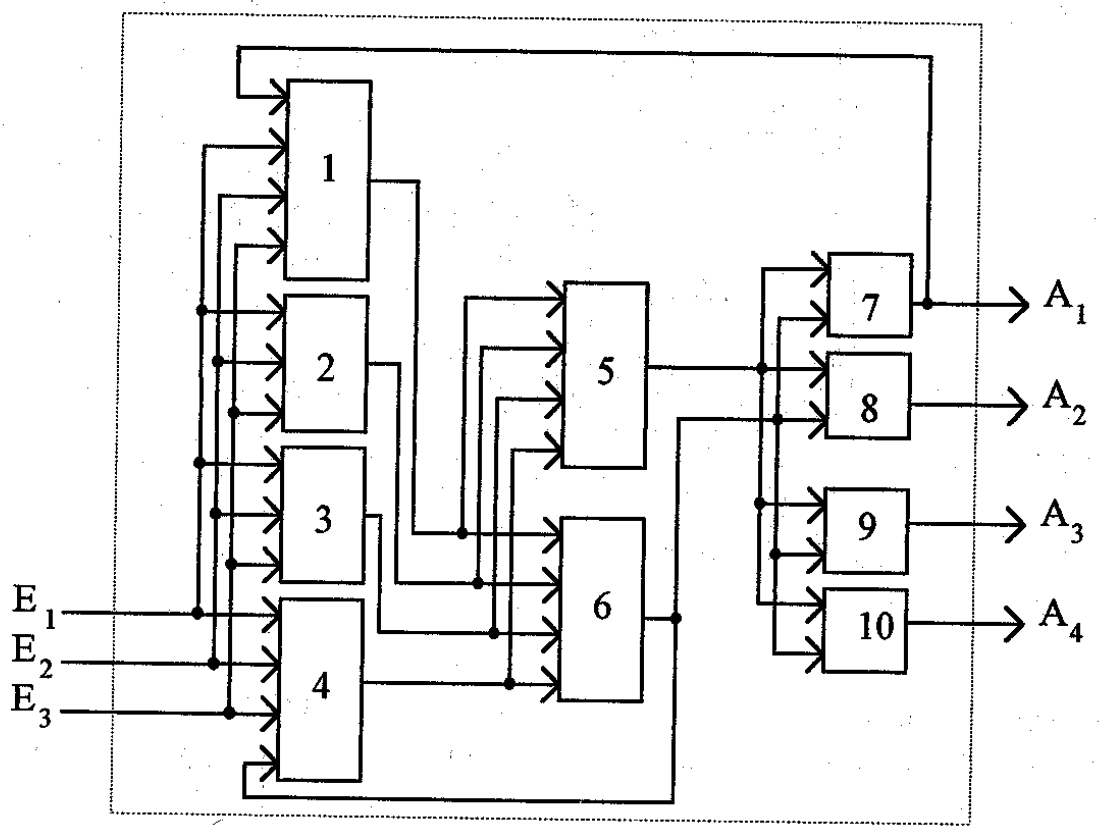


Intern: viele, stark vernetzte Neuronen oder stark strukturierte Verbindungen (Layers)

Beispiel: Feed-Forward-Network mit 1 hidden layer:



Neuronales Netz mit 3 Eingängen, 4 Ausgängen und 10 Neuronen:



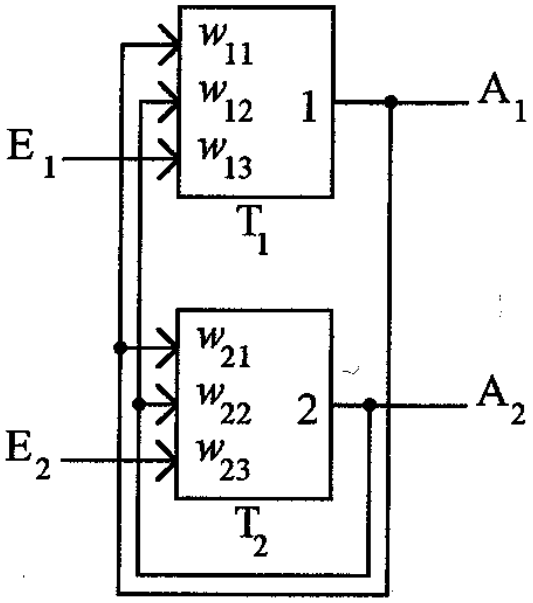
beachte die Rückkopplung von Neuron 6 an Neuron 4.

Berechnungsschema für ein rückgekoppeltes Netz:

Neuroneingänge (und Aktivitäten) auf Startwerte setzen
Eingangswerte ans Netz anlegen Nicht an Neuronen weitergeben
Neuronausgänge berechnen Nicht weitergeben
Neuroneingänge mit neuen Werten belegen
Wiederholen, bis Abbruchkriterium erfüllt

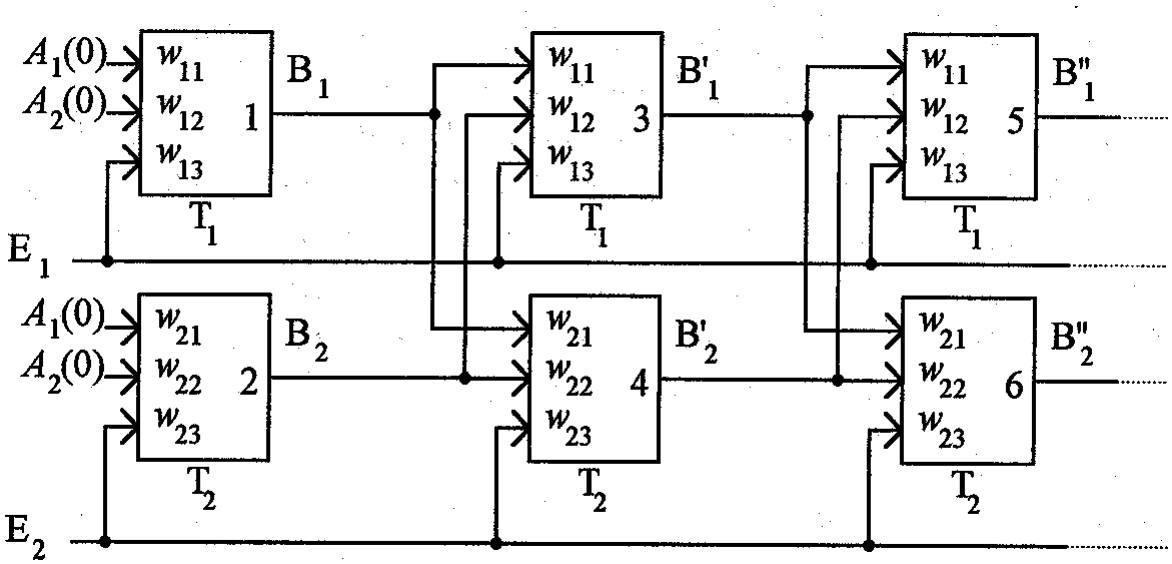
- Ohne Rückkopplung: vorwärtsgekoppelte Netze (*feedforward*-Netze)
- Mit Rückkopplung: rückgekoppelte (rekurrente) Netze (*feedback*-Netze)

Emulation rückgekoppelter Netze durch vorwärtsgekoppelte Netze:

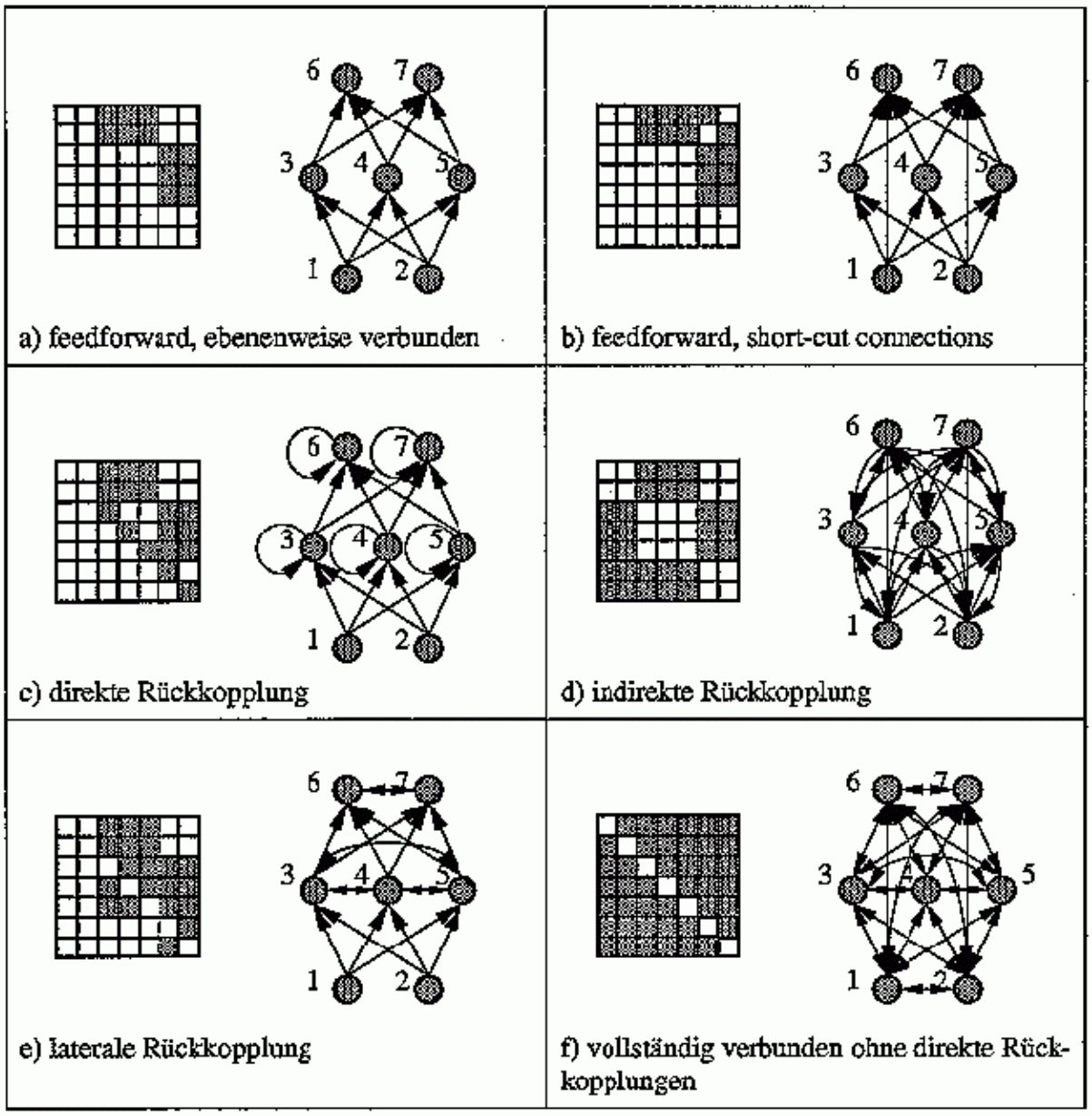


einfaches rückgekoppeltes Netz

Emulation:



Beispiele für Netztopologien mit zugehörigen Matrix-darstellungen:



Einteilung der Schichten in

- Eingabeschicht
- verdeckte Schichten (*hidden layers*) (optional)
- Ausgabeschicht.

Achtung: in der Literatur wird bei der Zählung der Schichten die Eingabeschicht oft nicht mitgezählt.

Lernverfahren

(a) überwachtes Lernen (*supervised learning*), "Lernen mit Lehrer":

- Trainingsmenge von Eingabe- und Ausgabemustern
- zu jedem Eingabemuster existiert ein eindeutiges korrektes (bestes) Ausgabemuster

Lernen:
Die Gewichte und evtl. Schwellenwerte werden solange durch nochmaliges Anlegen der Eingabemuster verändert, bis die Paarung (Eingabemuster, Ausgabemuster) für die Trainingsmenge stimmt.

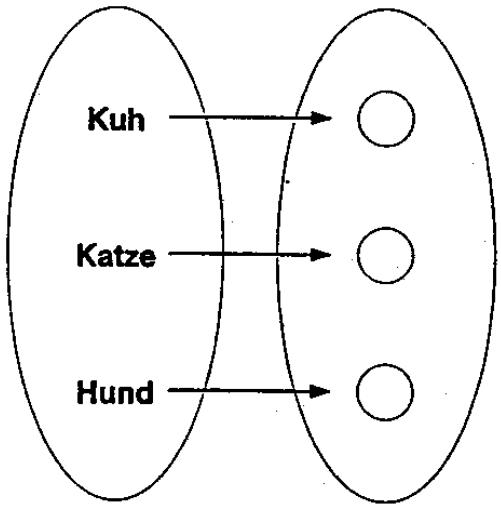
Generalisierung:
Ähnliche Eingabemuster, die nicht zur Trainingsmenge gehören, werden nach der Trainingsphase entweder in bereits trainierte Ausgabemuster (Perzeptron) oder in ähnliche Ausgabemuster (Backpropagation-Netzwerke) überführt.

(b) unüberwachtes Lernen (*unsupervised* / *self-organized learning*):
Lernen erfolgt durch Selbstorganisation. Ähnliche Eingabemuster werden assoziativ als ähnlich klassifiziert.
Beisp.: Kohonen-Netze.

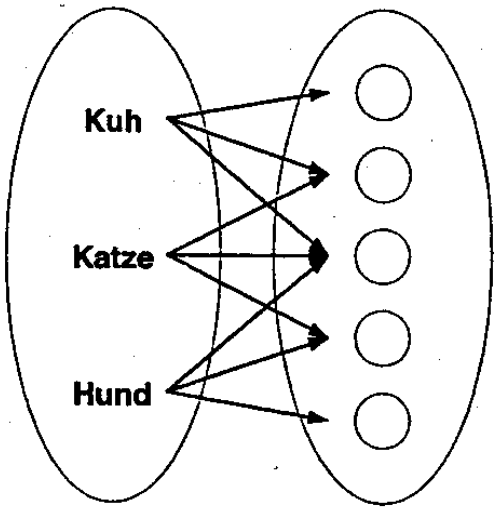
Überwachtes Lernen wird am häufigsten verwendet, obgleich vom biologischen Standpunkt nicht plausibel.

Repräsentation:
Lernen für konkrete Aufgaben setzt voraus, dass Abbildungsvorschriften von der Welt, in der sich das Netzwerk befindet, auf die Eingabe- und Ausgabeneuronen existieren (Codierung; Repräsentation).

- Lokale Repräsentation: Bijektion zwischen Konzepten und Ein- bzw. Ausgabeneuronen
- verteilte Repräsentation: 1 Konzept kann durch eine Gruppe von Neuronen repräsentiert werden (flexibler; Fehlertoleranz)



lokal



verteilt

bestimmte Repräsentationsformen sind für knN besser geeignet als andere, Beisp.: für Zahlenwerte besser Gray-Code als den Standard-Binärcode wählen!

Lernregeln:

Hebb'sche Regel (Donald Hebb, Neurophysiologe, 1949):

"Wenn Neuron *i* und Neuron *j* zur gleichen Zeit stark aktiviert sind, dann erhöhe das Gewicht w_{ij}, das diese beiden Neuronen verbindet"
in Formeln:

$$w_{ij}^{neu} = w_{ij}^{alt} + \Delta w_{ij}$$

$$\Delta w_{ij} = \alpha y_i y_j$$

(y_i : Ausgabewert von Neuron i)

Motivation: Beobachtungen an "echten" neuronalen Netzen
Nachteil: Gewichte können nur größer werden $\Rightarrow$ zu wenig flexibel.
Variante für einschichtige Netze: statt der Ausgangswerte werden Eingangswert und Sollwert eingesetzt

$$\Delta w_{ij} = \alpha \omega_i E_j$$

α heißt "Lernrate".

Delta-Lernregel (Widrow-Hoff-Regel):

$$\Delta w_{ij} = \alpha(\omega_j - y_j)y_i$$

(ω_j = Sollwert für Neuron *j*)

- Gewichte können größer und kleiner werden
- Gewichtsänderung ist proportional zum Fehler an den Ausgabe-Neuronen
- in dieser Form nur für Feedforward-Netzwerke mit 2 Schichten sinnvoll (Perzeptron)

- Für Perzeptron kann Erfolg der Lernregel garantiert werden
- aber nur eingeschränkte Mächtigkeit: nur Teilmenge der möglichen (Eingabe, Ausgabe)-Funktionen kann gelernt werden

Verallgemeinerung der Delta-Regel für Netzwerke mit mehr als 2 Schichten (d.h. mit hidden layers):

Training durch Backpropagation (Fehlerrückführungs-Netz)

Fehlerfunktion (soll minimiert werden):
$E = \sum (y_i - \omega_i)^2$ (Summation über alle Ausgabeneuronen),
Änderung der Gewichte durch Gradientenverfahren (dem steilsten Gradienten folgend):

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

(α heißt Lernfaktor).

Durchführung: Änderungsfaktor für die Gewichte wird zunächst für die Ausgabeschicht, dann sukzessive für die vorher-gehenden Schichten berechnet

- Gewichte anpassen, daß der Fehler minimiert wird:

$$w_{ij}(k+1) = w_{ij}(k) + \varepsilon \delta_j z_i(k)$$

- ε ... Lernkonstante, Lernrate
- $z_i(k)$ ... Ausgang des Knotens i
- δ_j ... Fehler der nächsthöheren Lage

$$\delta_j = y_j(1-y_j)(\omega_j - y_j) \quad \text{for output node j}$$

$$\delta_j = z_j(1-z_j)\left(\sum_l \delta_l w_{jl}\right) \quad \text{for hidden node j}$$

Trainingsvorgang:

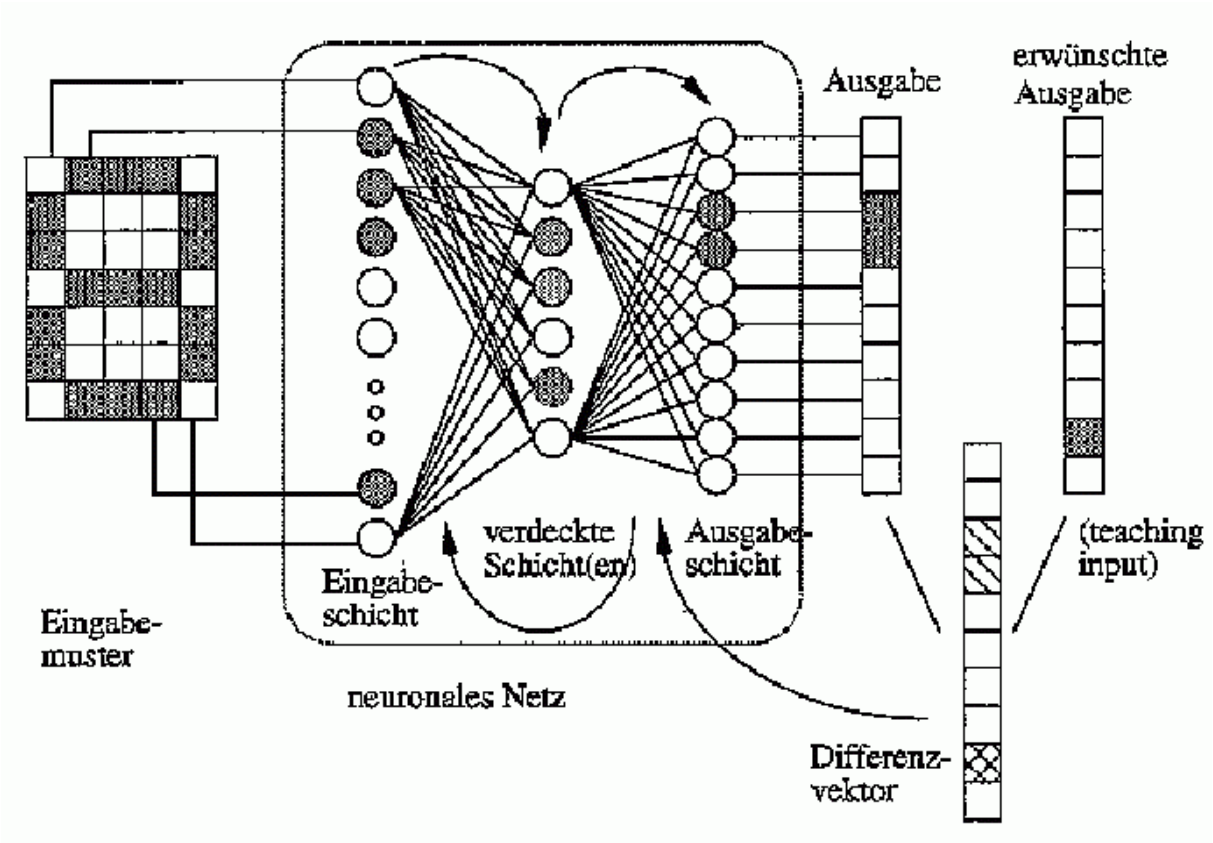
- Beliebige Initialisierung der Gewichte
- Präsentation eines Beispielvektors
  - läuft bis zum Ausgang durch
  - Adaptierung der Gewichte durch Backpropagation
- bis die Gewichte konvergieren

Klassifikation: Vektoren werden durchs Netzwerk geschickt.

Netz für alle Muster reproduzieren Zwischenergebnisse speichern
Fehlermaße der Ausgangsneuronen berechnen (Gl. 4-13)
Letzte verborgene Schicht wählen
Fehlermaße der gewählten Schicht berechnen (Gl. 4-14)
Vorhergehende Schicht wählen
Wiederholen, bis alle Fehlermaße berechnet sind
Alle Gewichte ändern (Gl. 4-12)

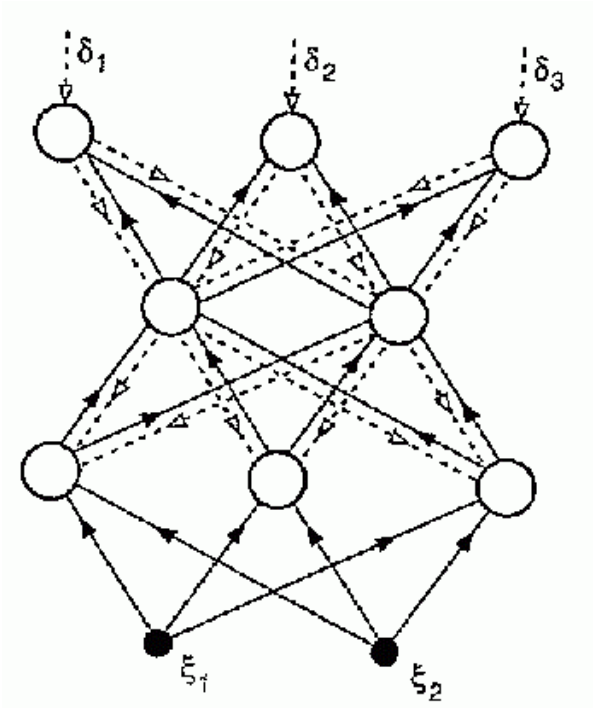
Vereinfachung des Lernschritts:

Wiederhole für alle Muster	
	Netz reproduzieren
	Fehlermaße der Ausgangsneuronen berechnen (Gl. 4-13)
	Fehlermaße der verborgenen Neuronen berechnen (Gl. 4-14)
	Alle Gewichte ändern (Gl. 4-12)



Backpropagation-Netz (aus Levi 2002)

Beispielarchitektur:



Komplexität der Entscheidungsflächen bei der Klassifikation mit Feedforward-Netzen mit Backpropagation in Abhängigkeit von der Zahl der Schichten (nach Beichel 2002):

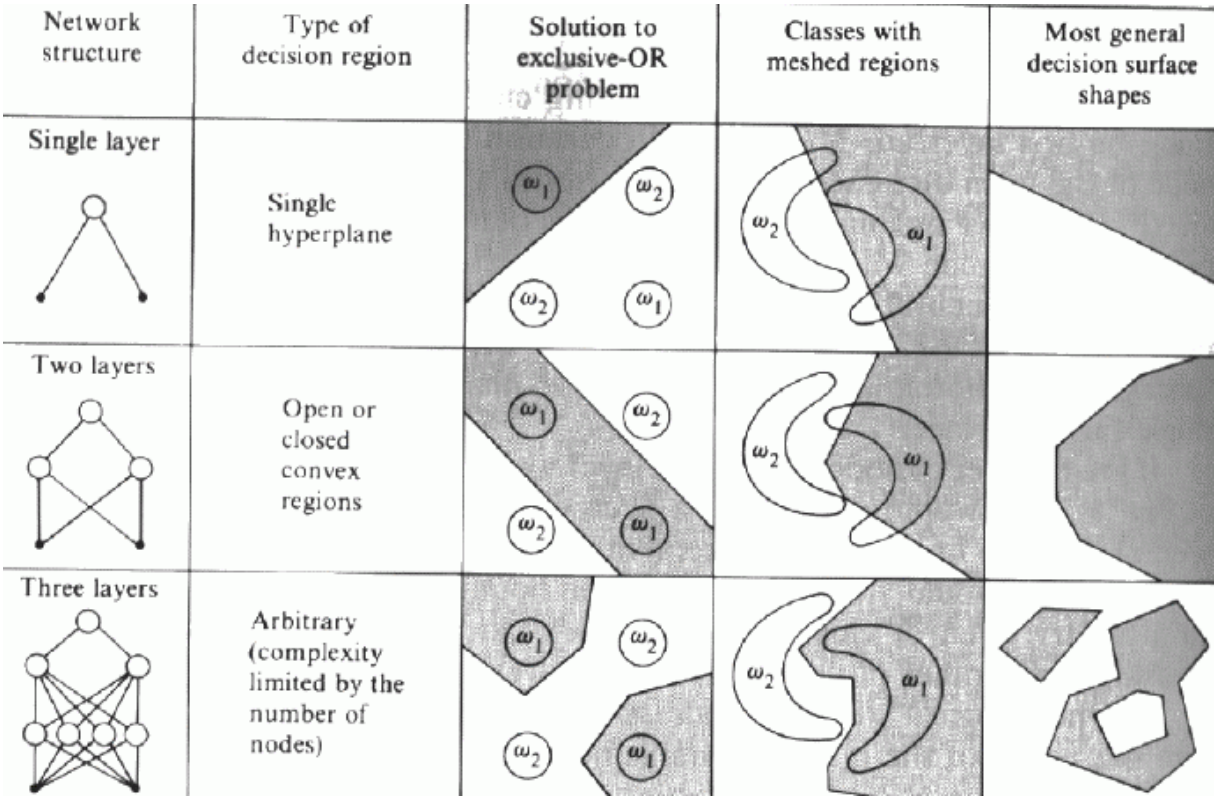


Tabelle 5-3 Überwacht lernende Netze (Kap. 5)
Die Neurontypen (Spalte „Neurontyp") sind in Abschn. 2.4.2, Tabelle 2-6 beschrieben.

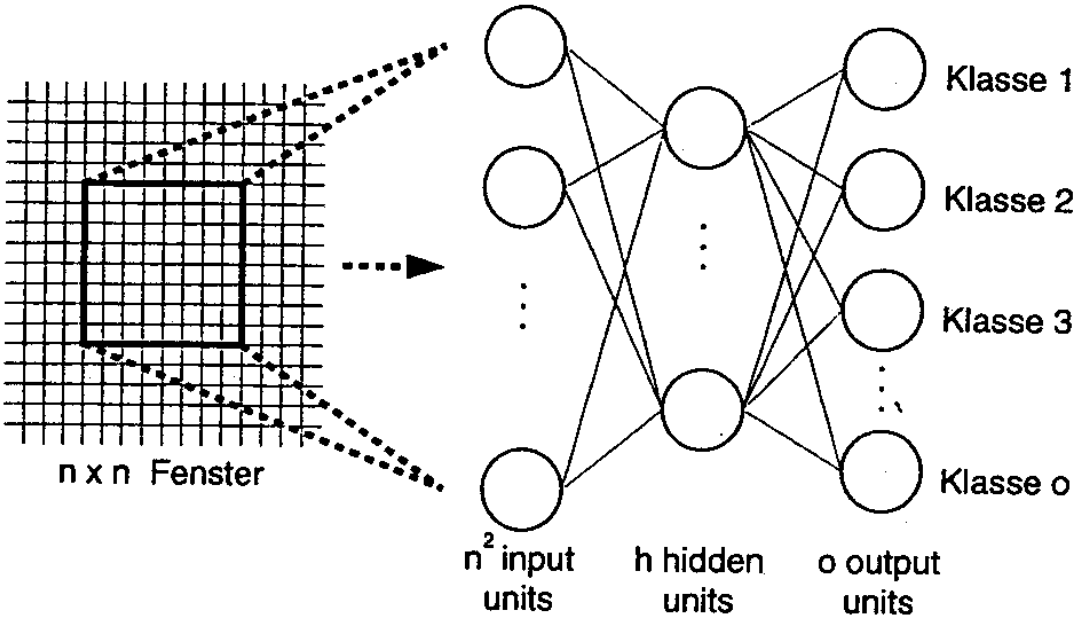
Netz	Aufbau	Arbeitsweise	Neurontyp	Lernregel	Literatur
Hopfield	1-schichtig rückgekopppelt	auto-assoziativ	Hopfield	Hopfield (= Hebb)	Brause 1991 Hertz 1991 Pao 1989 Ritter 1991 Schöneburg 1990
BAM	2-schichtig rückgekoppelt	hetero-assoziativ	Hopfield	Hopfield (Variante)	Brause 1991 Kratzer 1990 Schöneburg 1990
Boltzmann	3-schichtig rückgekoppelt	hetero-assoziativ	Boltzmann	Boltzmann	Ackley 1985 Brause 1991 Hinton 1988 Kratzer 1990 Schöneburg 1990
Gegenstrom	2-schichtig vorwärtsg.	hetero- und autoass.	s. 5.3.2	s. 5.3.3	Hertz 1991 Schöneburg 1990

Tabelle 5-4 Überwacht lernende Netze (Kap. 4)
Die Neurontypen (Spalte „Neurontyp") sind in Abschn. 2.4.2, Tabelle 2-6 beschrieben.

Netz	Aufbau	Arbeitsweise	Neurontyp	Lernregel	Literatur
Muster-Assoziator	1-schichtig vorwärts-gekoppelt	hetero-assoziativ	beliebig	Hebb, Delta	Brause 1991 Ritter 1991 Schöneburg 1990
Willshaw	wie Muster-Ass.	hetero-assoziativ	McCulloch-Pitts	speziell	Ritter 1991
Perzeptron	2-schichtig vorwärtsgekoppelt	hetero-assoziativ	McCulloch-Pitts	Perzeptron (= Delta)	Block 1962 Brause 1991 Kratzer 1990 Minsky 1988 Ritter 1991 Rosenblatt 1958 Schöneburg 1990
ADALINE	Muster-Assoziator mit Bias	hetero-assoziativ	ADALINE	Delta (Variante)	Brause 1991 Kratzer 1990 Schöneburg 1990
MADALINE	2-schichtige Erweiterung des ADALINE	hetero-assoziativ	speziell	speziell	Kratzer 1990 Schöneburg 1990
Auto-Assoziator	1-schichtig rückgekoppelt	auto-assoziativ	beliebig	Hebb, Delta	Hoffmann 1992 McClelland 1988b
BSB	wie Auto-Assoziator	auto-assoziativ	BSB	Delta	Hoffmann 1992 McClelland 1988b
DMA	wie Auto-Assoziator	auto-assoziativ	DMA	Delta	Hoffmann 1992 McClelland 1988b McClelland 1988d
Fehlerrückführung	mehrschichtig vorwärts-gekoppelt	hetero-assoziativ	differenzierbare Ausgangs-funktion	Fehlerrückführung	Ruiz 1991 Bellido 1991 Brause 1991 Kratzer 1990 Müller 1990 Ritter 1991 Rumelhart 1988b Schöneburg 1990

Anwendungen

Beispiel eines Feedforward-Netzwerks für die Bildinterpretation:



(aus Pinz 1994)

Einteilung der Anwendungen nach der grundsätzlichen Arbeitsweise des Netzes (aus Hoffmann 1993):

Tabelle 7-1 Einteilung der Arbeitsweise eines Netzes

Diese Einteilung beschränkt sich auf das Reproduktionsverhalten des Netzes. Das Verhalten ist vom Netztyp und in vielen Fällen auch von den Gewichten und vom angelegten Eingangsmuster abhängig. Die verschiedenen Verhaltensmöglichkeiten korrespondieren mit Problemtypen (in der Tabelle fett umrandet), für die das jeweilige Netz geeignet ist.

<table>
<tr><td colspan="6">Grundsätzliche Arbeitsweise eines Netzes:
Vorgegeben: Folge von Eingangsmustern $E(t)$, $t = 0,1,2,...$
Ergebnis: Folge von Ausgangsmustern $A(t)$, $t = 0,1,2,...$</td></tr>
<tr><td colspan="5">*Statische* Vorgabe: ein statisches Eingangsmuster wird an das Netz angelegt</td><td>*Dynamische* Vorgabe: eine Folge von Eingangsmustern wird angelegt</td></tr>
<tr><td colspan="4">Ausgang erreicht oder approximiert statischen Zustand</td><td colspan="2">Ausgang ist für beliebig große Zeiten veränderlich (nur bei rückgekoppelten Netzen möglich)</td><td>Zeitreihe</td></tr>
<tr><td>Höchstens ein Ausgang ist aktiv</td><td colspan="3">Mehrere Ausgänge sind aktiv</td><td>Grenzzyklus</td><td>Chaos</td></tr>
<tr><td>Klassifikation</td><td colspan="2">Ausgangsmuster = Eingangsmuster</td><td colspan="2">Ausgangsmuster mit Eingangsmuster assoziiert</td><td>Ausgangsmusterfolge</td><td>unbrauchbar [1]</td></tr>
<tr><td></td><td>autoassoziativer Speicher: Musterergänzung</td><td>Vorgabe: gelerntes Muster</td><td>Vorgabe: neues Muster</td><td colspan="2"></td></tr>
<tr><td></td><td></td><td>heteroassoziativer Speicher</td><td>Generalisierung</td><td colspan="2"></td></tr>
</table>

[1] es sei denn, man möchte Chaos erzeugen!

Anwendung in der Mustererkennung:

Anwendung	Problemtyp	Beispiel	Netz	Literatur
Muster-erkennung	Klassifizierung	Schrift-erkennung	Muster-Assoziator	Hoffmann 1992
			ART	Dimitriadis 1991
			Neocognitron	Kap. 11
	autoassoziati-ver Speicher	Rekonstrukti-on von Bildern	Auto-Assoziator	Hoffmann 1992
			Hopfield-Netz	Hoffmann 1992
		Rekonstruktion von Gesichtern	Auto-Assoziator	Kohonen 1988
		Invariante Mu-stererkennung	Netz mit Vor-verarbeitung durch Fourier-transformation	Haken 1989 Müller 1990
			Netz mit ΣΠ-Neuronen	Abschn. 5.5
			Neocognitron	Kap. 11