Einführung in die Neuroinformatik

Günther Palm und Friedhelm Schwenker

Institut für Neuroinformatik

Universität Ulm

Organisation im Sommersemster 2011

- Vorlesung: Di 12-14 Uhr Raum H21
- Übung: Fr 10-12 Raum H21
- *Einführung in Matlab* am 15.04.2011
- Übungsaufgaben sind schriftlich zu bearbeiten (Schein bei 50% der erreichbaren Punkte und aktiver Teilnahme in der Übungsstunde).
- Bachelor (Medien-)Informatik, Mathematik, Biometrische Methoden
- Master (Medien-)Informatik (Kernmodul): Mathematische und theoretische Methoden der Informatik und Praktische Informatik, aber i.R. nicht in einem Vertiefungs- oder Projekt-modul
- Diplom (Medien-)Informatik (Kernfach): Mathematische und theoretische Methoden der Informatik und Praktische Informatik.
- Diplom (Medien-)Informatik (Vertiefungsfach): Neuroinformatik.

Inhalt

1. Computer und neuronale Netze
2. Biologische neuronale Netze
3. Neuronenmodelle
4. Neuronale Architekturen
5. Lernen in künstlichen neuronalen Netze
6. Überwachtes Lernen in KNN
7. Kompetitives und unüberwachtes Lernen in KNN
8. Assoziativspeicher
9. KNN in der Praxis

Literatur

[1] D. J. Amit. *Modelling Brain Function: The world of attractor neural networks*. Cambridge University Press, Cambridge, 1989.

[2] M. Arbib. *The Metaphorical Brain 2*. Wiley, New York, 1988.

[3] R. Beale and T. Jackson. *Neural Computing an Introduction*. Adam Hilger, Bristol and New York, 1990.

[4] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[5] H. Braun. *Praktikum neuronaler Netze*. Springer, 1996.

[6] R. Brause. *Neuronale Netze*. B. G. Teubner, Stuttgart, 1991.

[7] V. Cherkassky, J.H. Feldman, and H. Wechsler, editors. *From Statistics to Neural Networks*, volume 135 of *Computer and Systems Sciences*. Springer, 1994.

[8] A. Chichocki and R. Unbehauen. *Neural networks for Optimization and Signal Processing*. Wiley, 1993.

[9] M. H. Hassoun. *Fundaments of Artificial Neural Networks*. MIT Press, 1995.

[10] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company, 1994.

[11] J A. Hertz, A Krogh, and R G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991.

[12] T. Kohonen. *Self-Organization and Associative Memory*. Springer, Berlin, 1983.

[13] T. Kohonen. *Self-Organizing Maps*. Springer, 1995.

[14] B. Kosko. *Neural Networks and fuzzy systems*. Prentice-Hall International Editions, 1992.

[15] R. J. Mammone, editor. *Artificial Neural Networks for Speech and Vision*. Chapman& Hall, 1994.

[16] D. Nauck, F. Klawonn, and R. Kruse. *Neuronale Netze und Fuzzy-Systeme*. Vieweg, 1994.

[17] M. L. Minsky S. A. Papert. *Perceptrons*. MIT–Press, 2nd edition, 1988.

[18] R. Rojas. *Theorie der neuronalen Netze*. Springer, 1993.

[19] D. E. Rumelhart and J. L. Mc Clelland. *Parallel Distributed Processing Vol. 1: Foundations*. MIT Press, Cambridge, 1986.

[20] P. D. Wasserman. *Advanced methods in neural computing*. Van Nostrand Reinhold, New York, 1993.

[21] H. White. *Artifical Neural Networks*. Blackwell, 1992.

[22] A. Zell. *Simulation neuronaler Netze*. Addision Wesley, 1994.

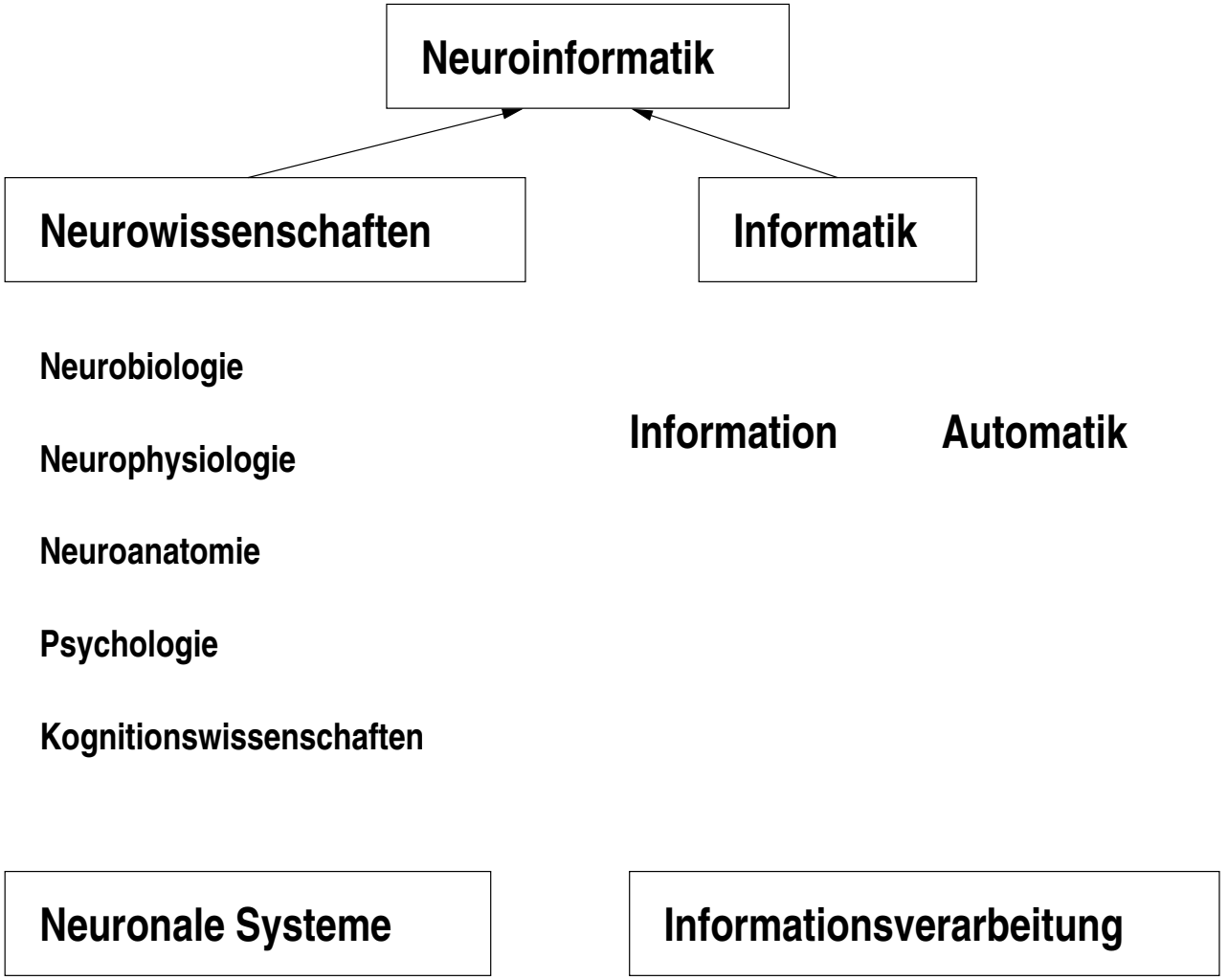
1. Computer und neuronale Netze

1. Was ist die Neuroinformatik?

2. Aufgaben für neuronale Systeme

3. Computer vs. neuronale Netze

Neuroinformatik ist die Wissenschaft, die sich mit der **Informationsverarbeitung** in **neuronalen Systemen** (biologischen und künstlichen) befasst.



Aufgaben für neuronale Systeme

Visuelle Erkennung: Erkennung von Objekten, Gesichtern, Handschriftzeichen.

Akustische Erkennung: Erkennung von kontinuierlich gesprochener Sprache, die Identifizierung eines Sprechers.

Autonome Systeme: Lokalisation im Raum, Ausweichen von Hindernissen, Planung von Wegen.

Arithmetik: Berechnung von $23.45612^{46537829}$, $\pi = 3,1415\ldots$ (eher nicht)

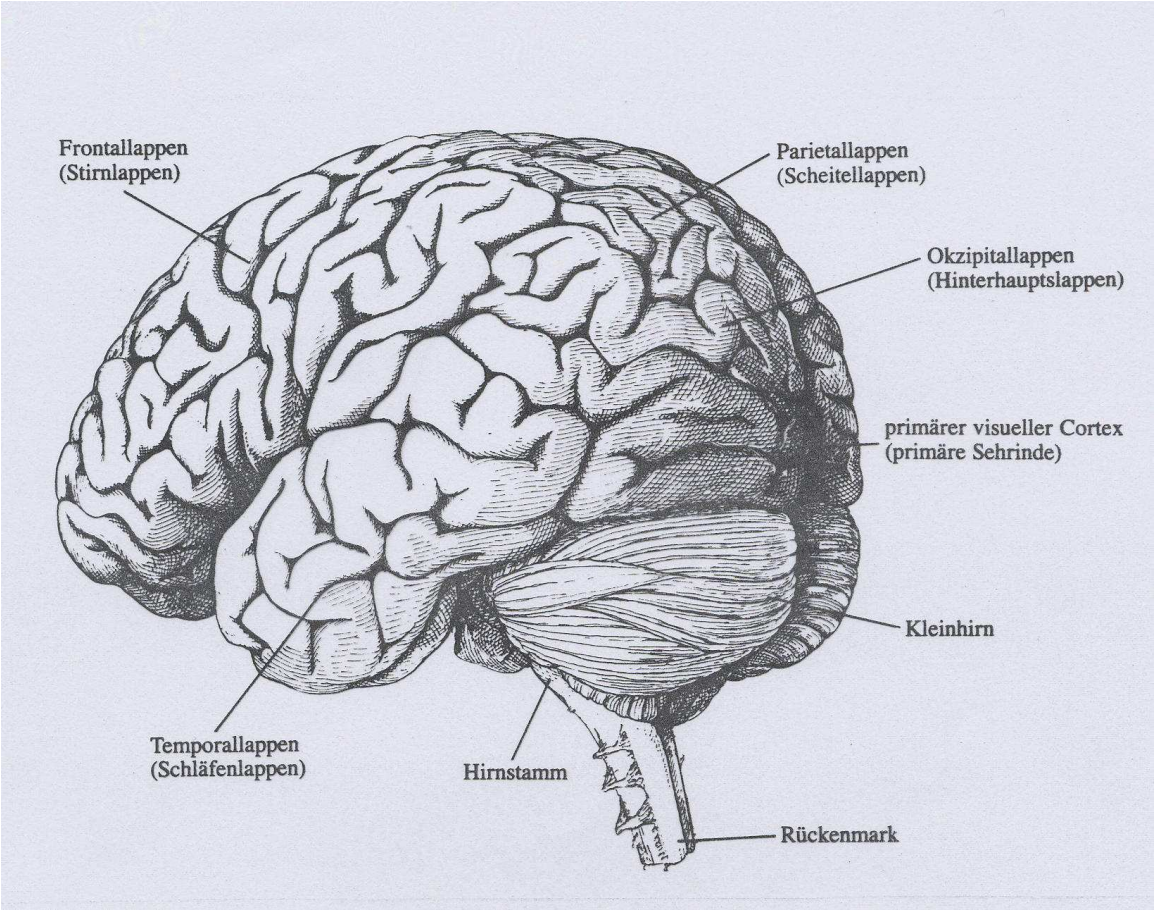
(Mathematische) Beweise: Gibt es natürliche Zahlen x, y und z, so dass die Gleichung $x^n + y^n = z^n$ für natürliche Zahlen $n > 2$ erfüllt ist? (eher nicht)

Computer	**Neuronale Systeme**
wenige aber komplexe Prozessoren $\approx 10^0$ – 10^4	viele einfache Neuronen, z. B. im menschlichen Gehirn $\approx 10^{10}$-10^{11}
niedriger Vernetzungsgrad $\approx 10^1$	hoher Vernetzungsgrad $\approx 10^3$–10^5 Synapsen pro Neuron
Taktfrequenz liegt im GHz-Bereich	wenige hundert Spikes pro Sekunde
kaum fehlertolerant	fehlertolerant (fuzzy, verrauschte oder wiedersprüchliche Daten)
nicht robust; Störung führen meist zum Ausfall	robust bzw. stetig; Störung bedeutet kein Totalausfall, sondern eingeschränkte Funktionalität
explizite Programmierung	lernen durch Beispiele
Trennung von Programmen und Daten	Keine Trennung; die Synapsenstärke beeinflußt die Neuronenaktivität und umgekehrt

2. Biologische Grundlagen neuronaler Netze

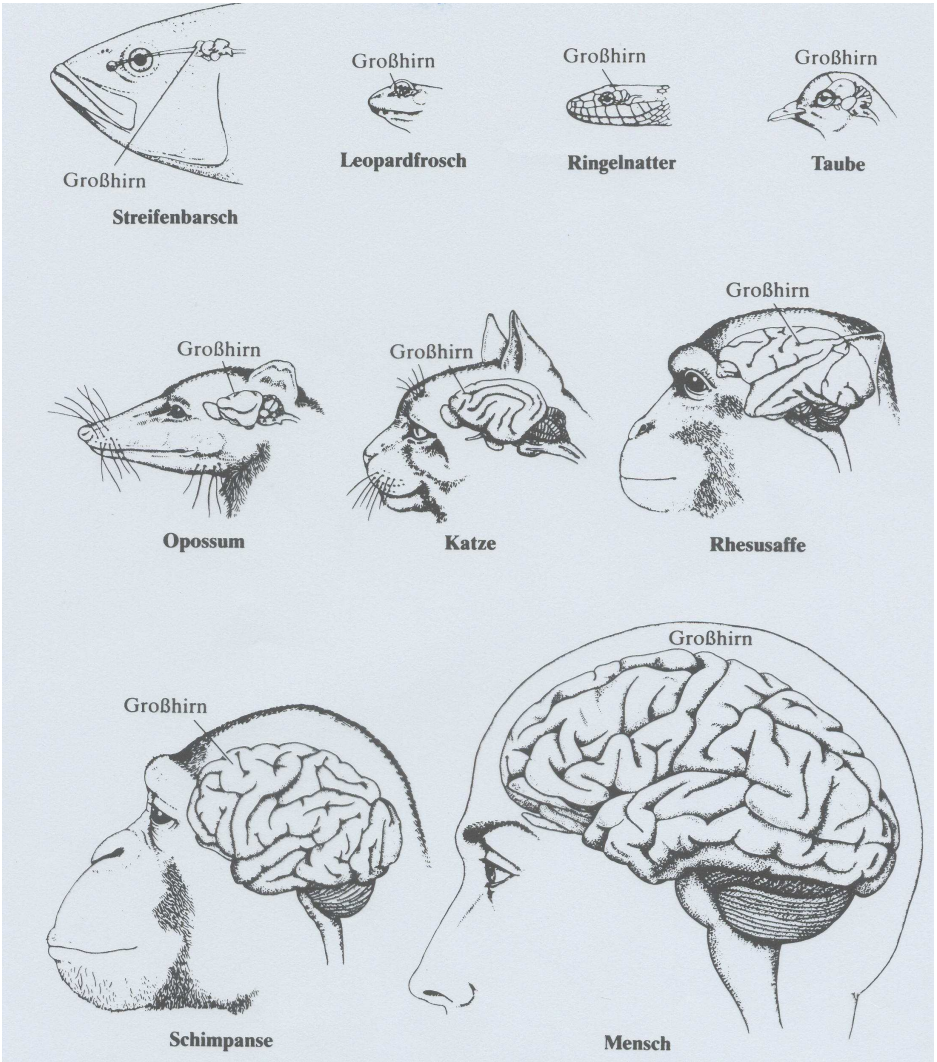
1. Gehirn
2. Neuron
3. Synapse
4. Neuronale Dynamik

Menschliches Gehirn

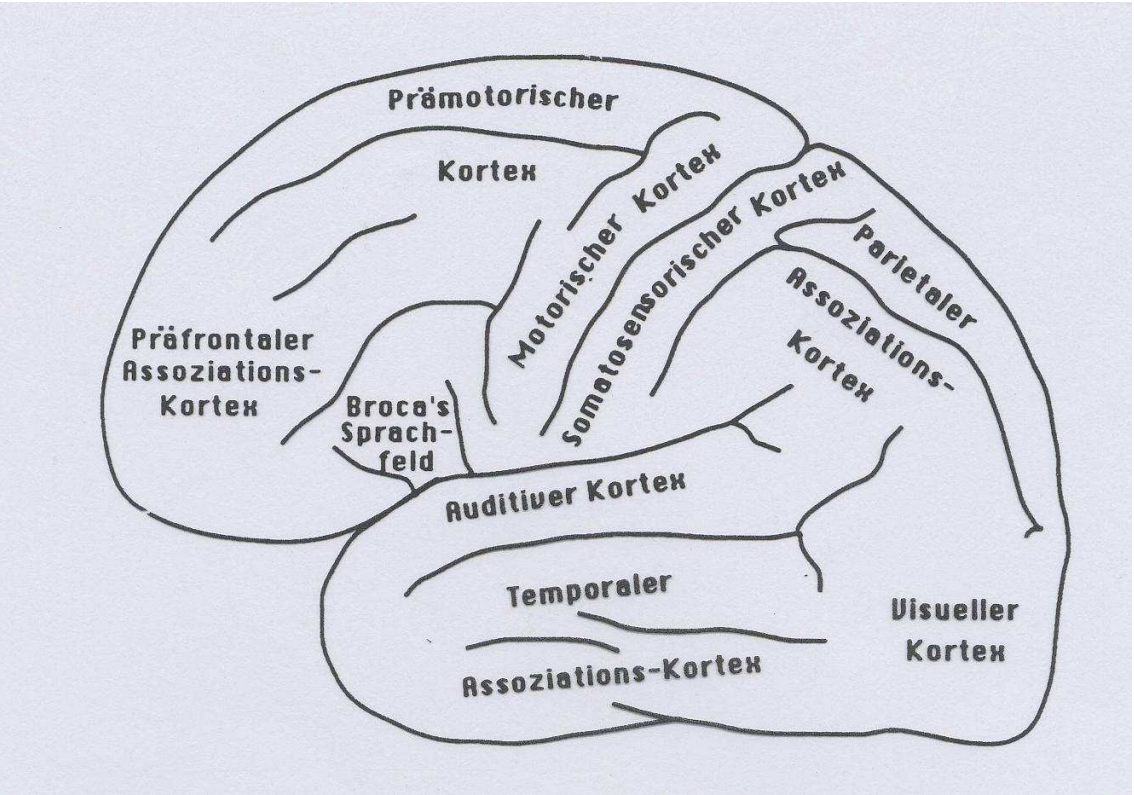


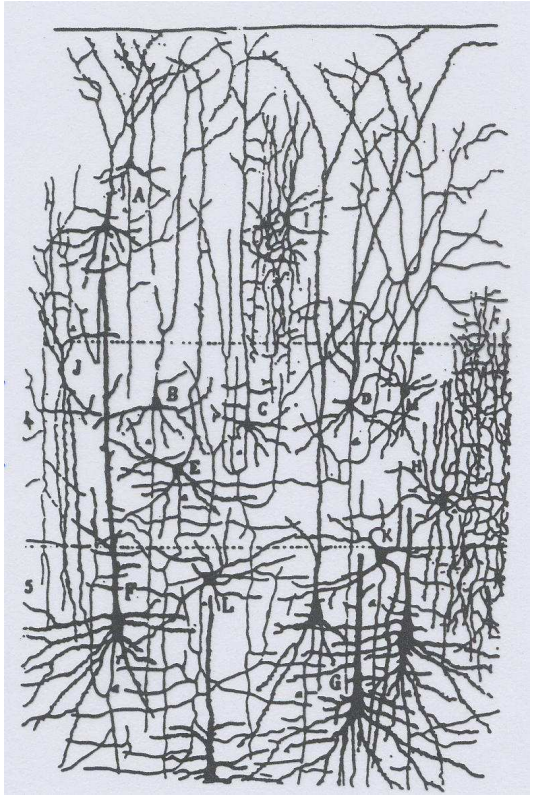
Ansicht eines menschlichen Gehirns (von links-hinten). Großhirnrinde und Kleinhirn sind gezeigt.

Gehirne einiger Tiere



Fortschreitende Vergrößerung (bzgl. des Volumens und der Einfaltungen) des Großhirns bei Wirbeltieren.

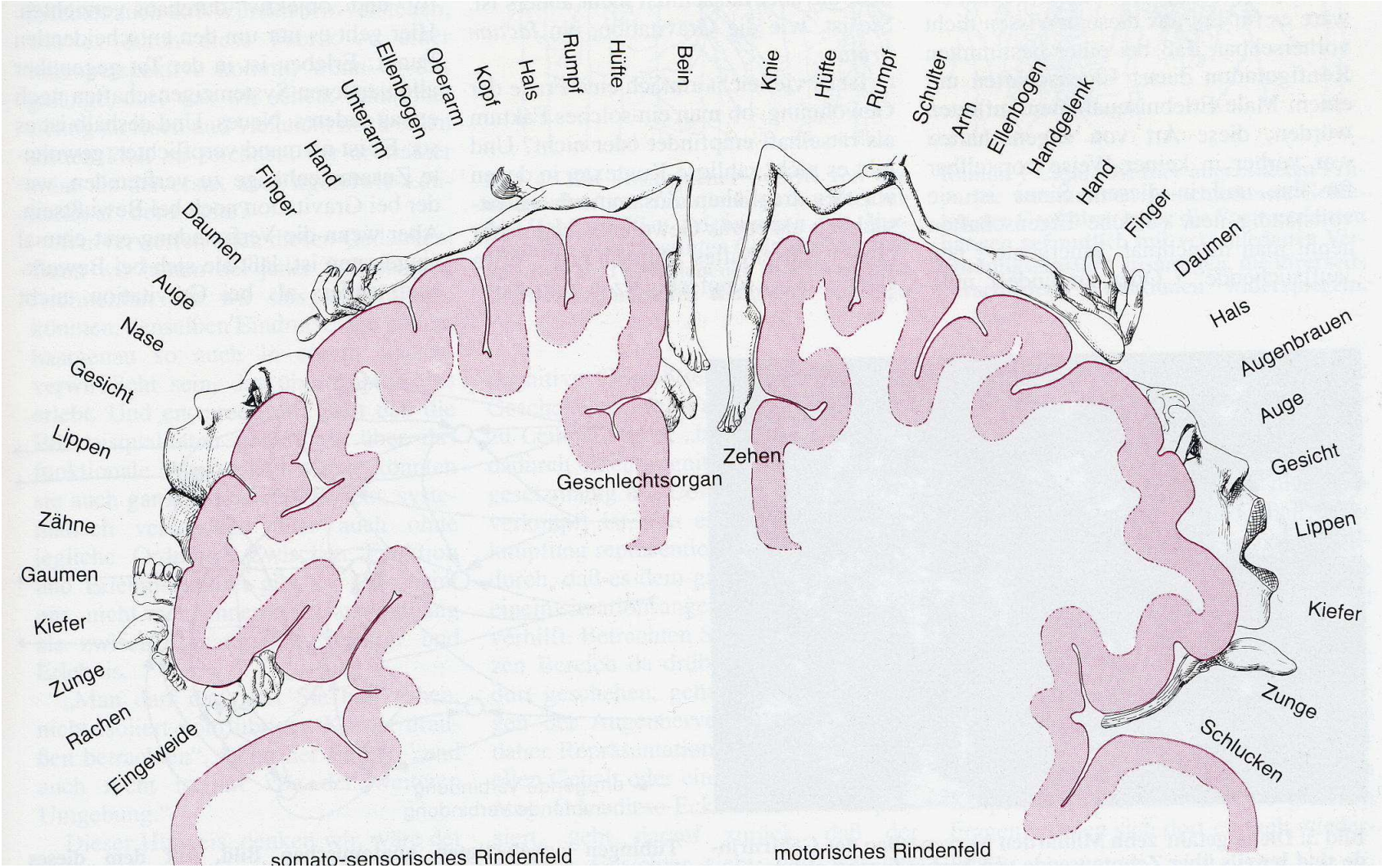


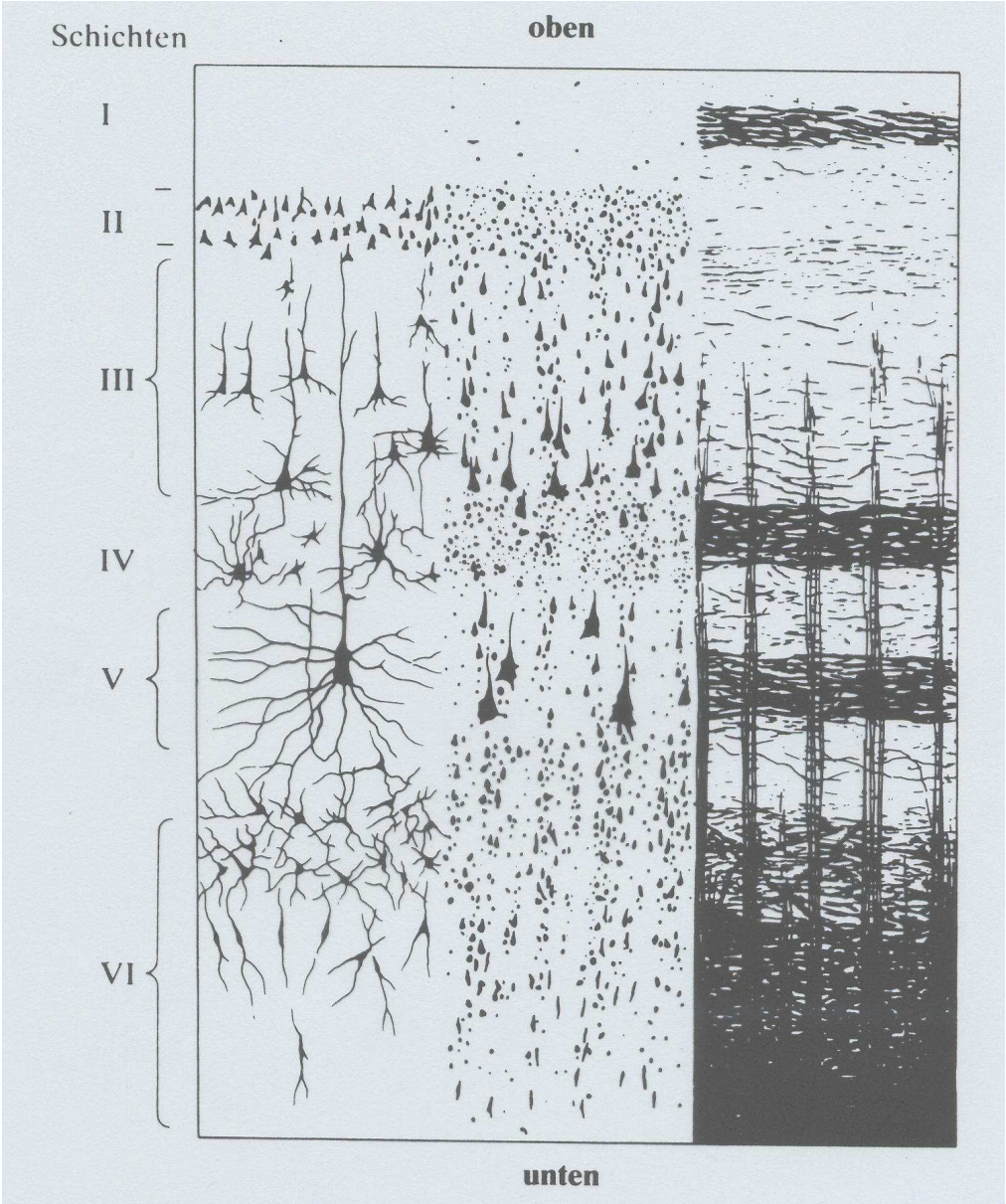


Links: Seitenansicht der linken Gehirnhemisphäre beim Menschen. Die Oberfläche ist stark gefaltet; verschiedene Felder mit ihrer Spezialisierung auf Teilaufgaben sind abgegrenzt.

Rechts: Schnitt durch die Hirnrinde einer Katze. Pyramidenzellen (A-G); Sternzellen (H-M). Etwa 1% der Nervenzellen sind durch spezielle Färbungstechniken gezeigt. Menschliches Gehirn: $10^{10} - 10^{11}$ Neuronen.

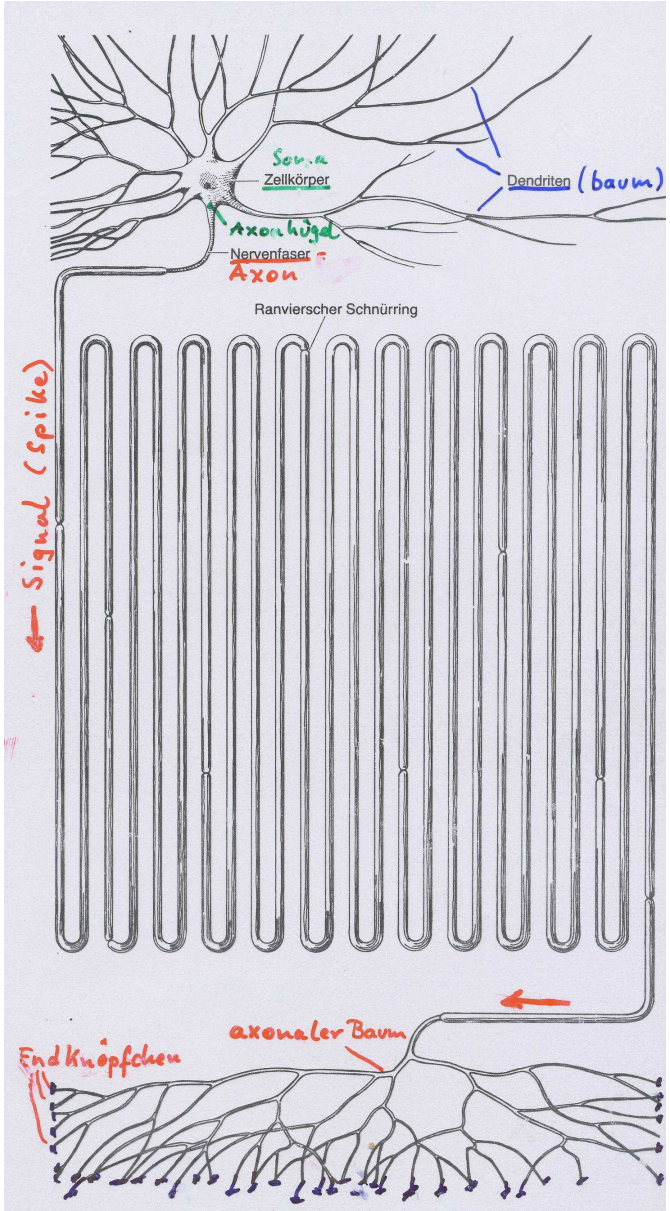
Somatosensorische und motorische Hirnrinde





Schichtenstruktur der Großhirnrinde. Es sind verschiedene neuronale Strukturen gezeigt:

- Neuronen mit Nervenfasern(links)
- Verteilung der Zellkörper (Mitte).
- Organisation der Nervenfasern (rechts).

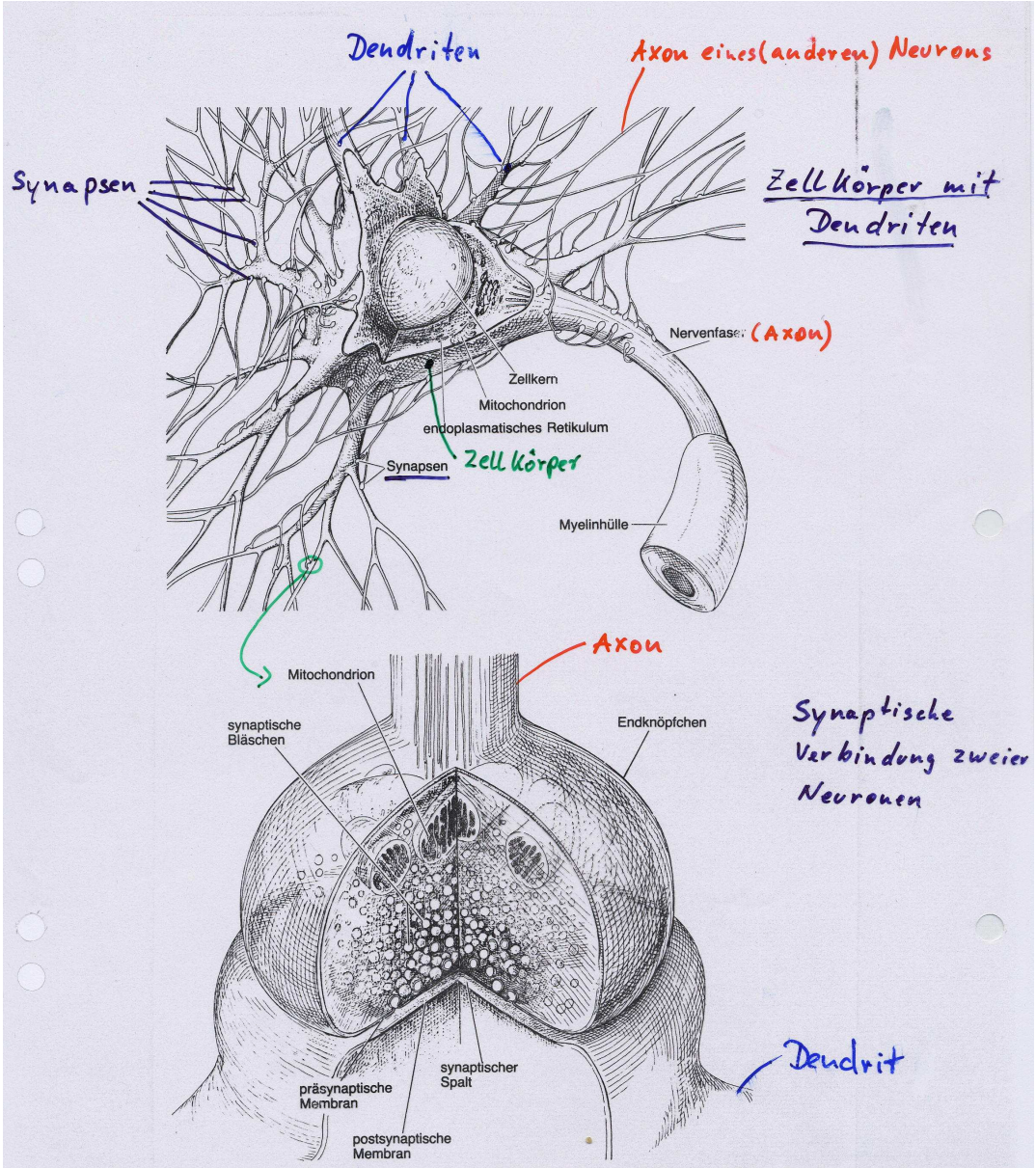


Neuron (schematisch dargestellt)

Anatomische Teilstrukturen eines Neurons:

- Zellkörper (Soma)
- Dendrit
- Axon

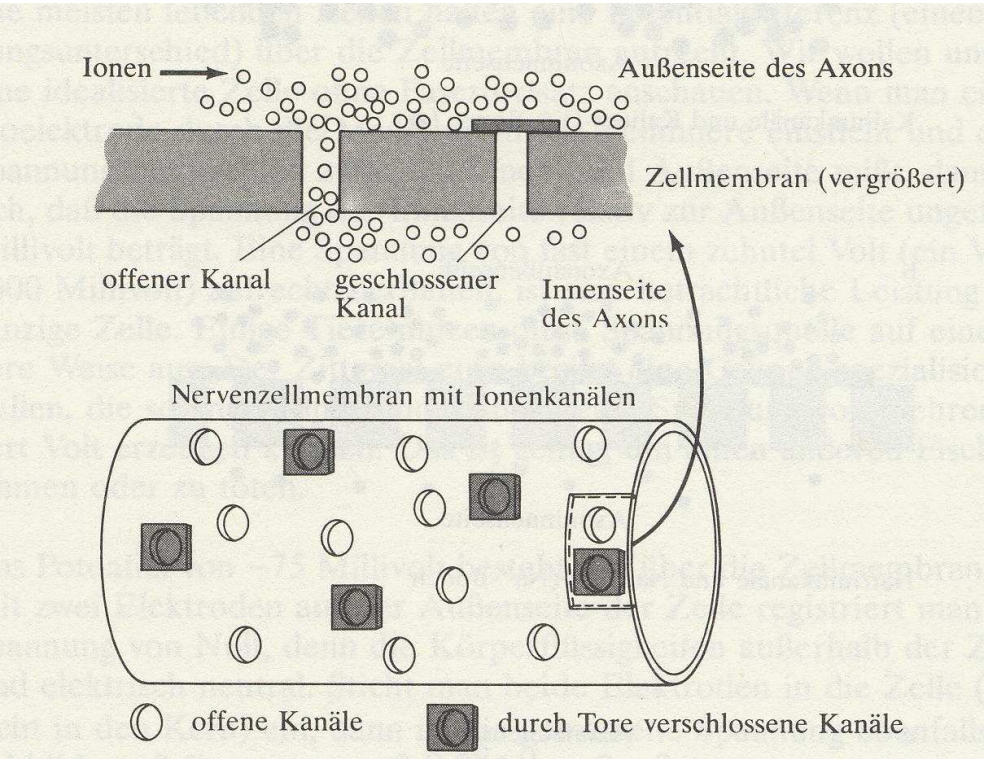
Der Nervenimpuls (*Spike* oder Aktionspotential) breitet sich ausgehend vom Zellkörper (genauer vom Axonhügel) über das gesamte Axon bis in die axonalen Endigungen aus.



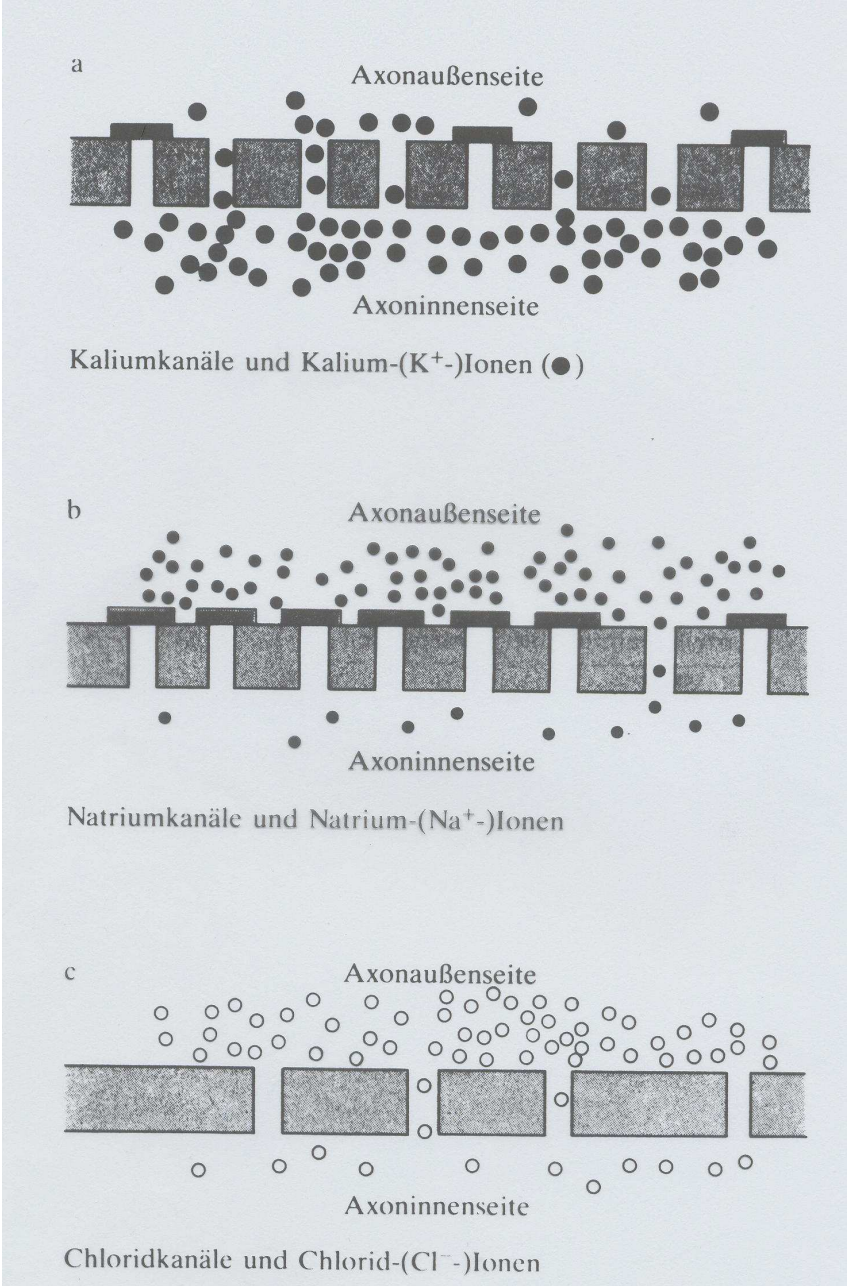
Oben: Zellkörper mit einem stark verzweigten Dendriten(baum). An den Verästelungen des Dendriten befinden sich eine Vielzahl von Kontaktstellen, die sogenannten **Synapsen**, von vorgeschalteten Neuronen.

Unten: Synaptische Kopplung zweier Neuronen. In der präsynaptischen Membran (axonales Endknöpfchen) befindet sich der **Neurotransmitter** in den **synaptischen Bläschen**.

Zellmembran



Sematische Darstellung einer Axonmembran. Offene und durch Tore (*gates*) verschlossene Ionenkanäle sind gezeigt. Verschiedene Ionentypen befinden sich außerhalb und innerhalb der Membran.

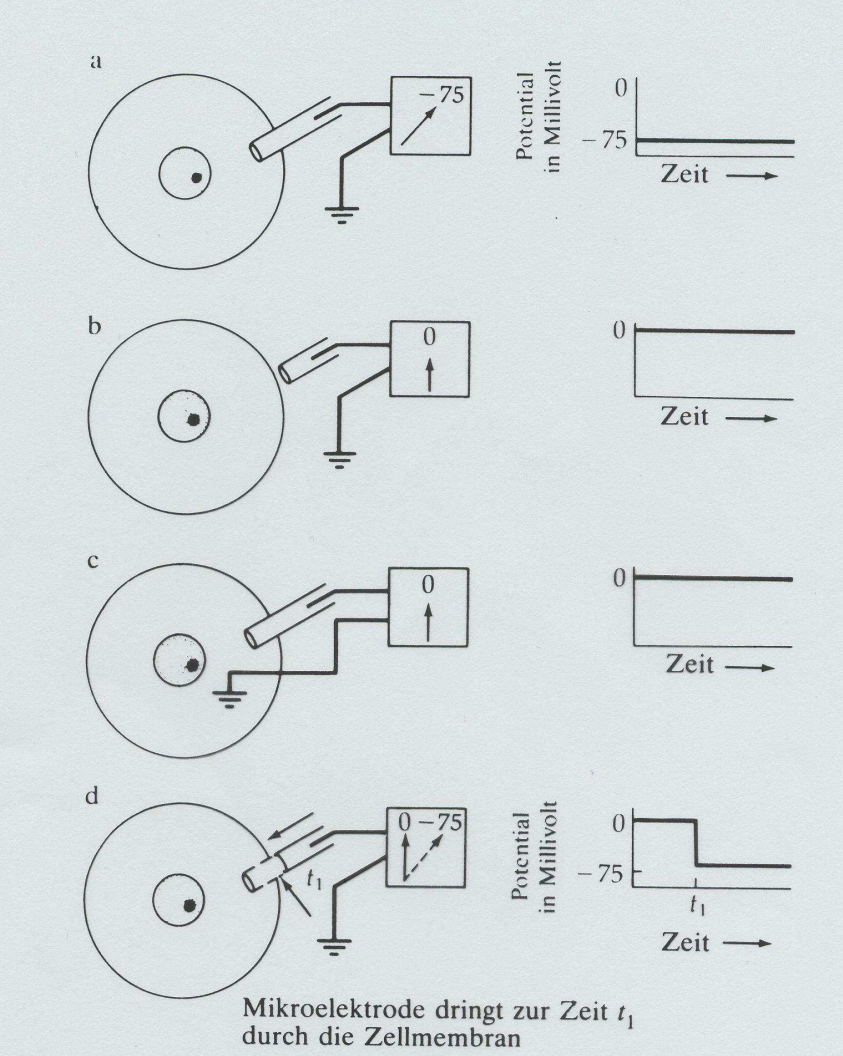


Die häufigsten Ionenkanaltypen in der Axonmembran:

- Kaliumkanäle (oben) gehören (in der Mehrzahl) zum offenen Typ; Kaliumionen sind vor allem innerhalb der Zelle.
- Natriumkanäle (Mitte) sind (in der Mehrzahl) durch *Gates* verschlossen; Natriumionen befinden sich hauptsächlich außerhalb der Zelle.
- Chloridkanäle (unten) sind (in der Mehrzahl) offen; Chloridionen sind überwiegend außerhalb der Zelle.

Im Vergleich zu Kalium- oder Natriumionen sind Chloridionen deutlich in der Minderzahl.

Potential an der Zellmembran



Mikroelektrode dringt zur Zeit t_1 durch die Zellmembran

Messungen des Ruhepotentials an der Zellmembran

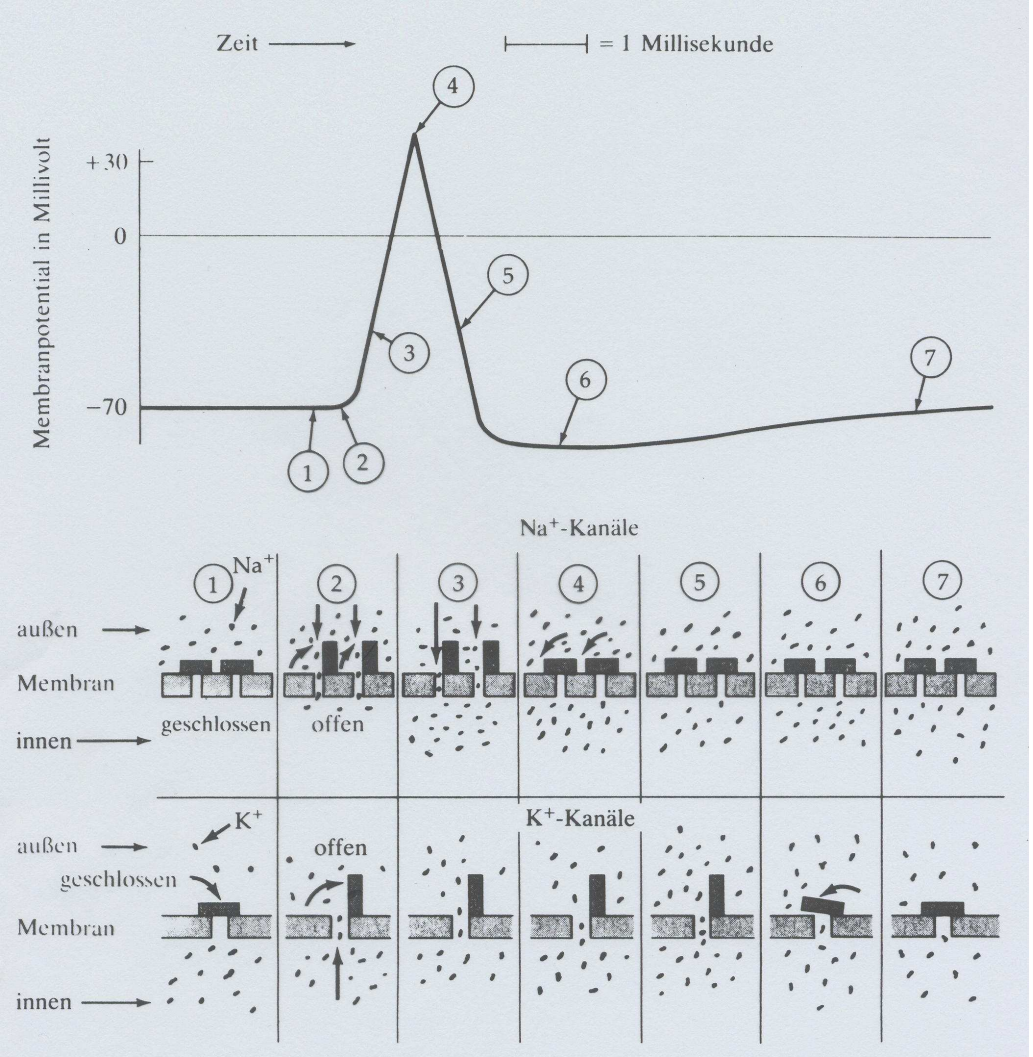
a **Ruhepotential** (hier bei ca. -75mV) zwischen dem Inneren und dem Äußeren der Zelle.

b kein Potential im Extrazellulärraum

c kein Potental im Intrazellulärraum

d Einstechen einer Mikroelektrode in die Zelle zur Zeit t_1. Potential fällt sprunghaft auf das Ruhepotential ab.

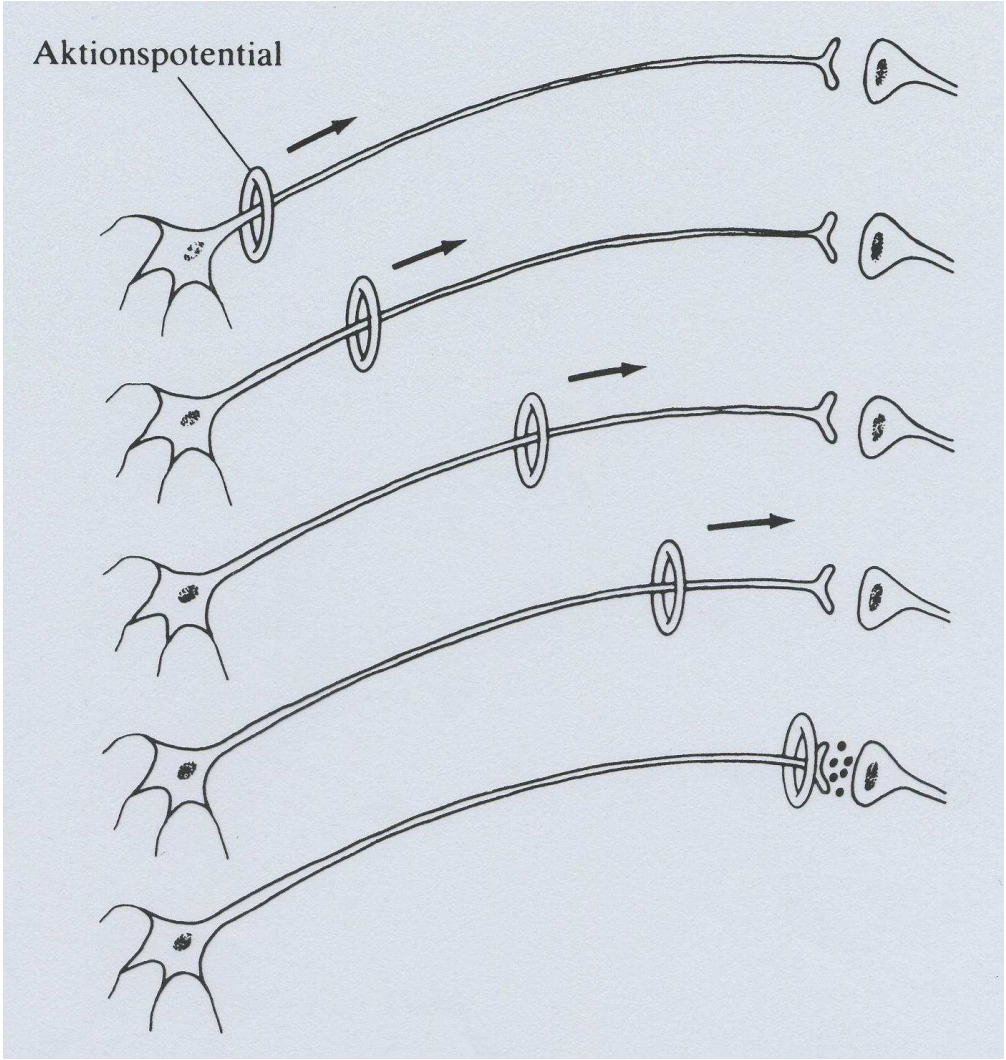
Aktionspotential an der Zellmembran



Phasen des Aktionspotentials

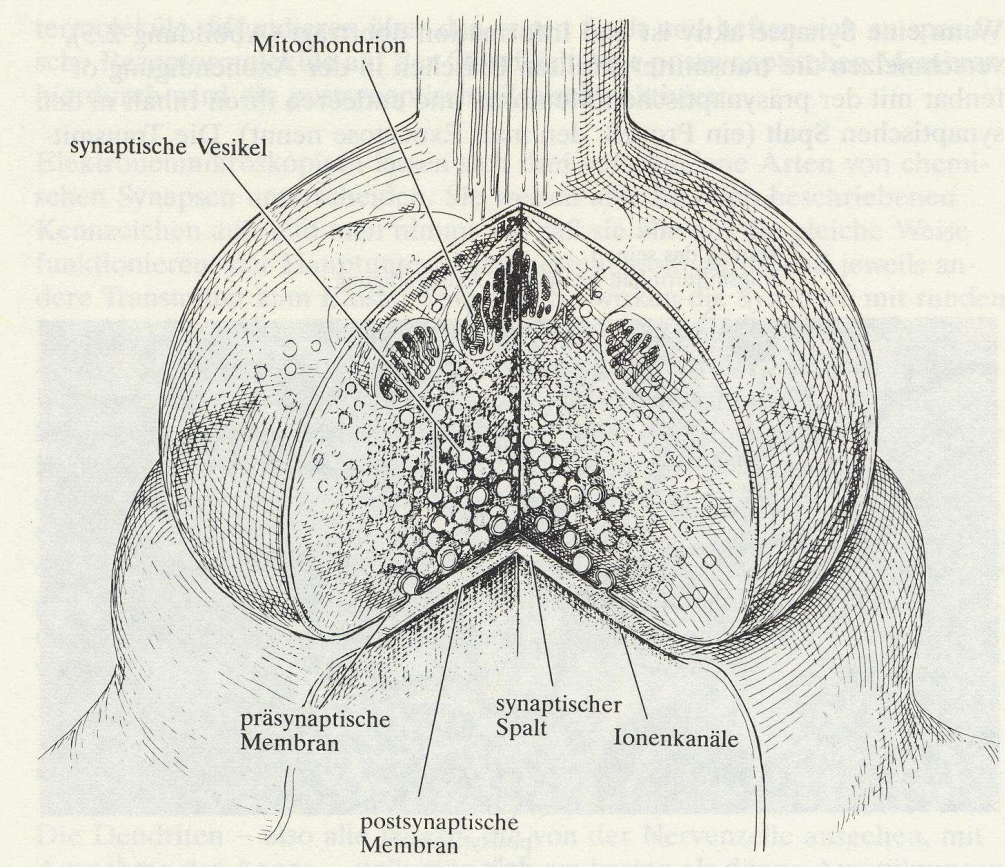
1. Ruhepotential.
2. Spannungsgesteuerte Natrium- und (wenige) Kaliumkanäle springen auf (durch Erregung der Membran). Positive Natriumionen strömen massiv in die Zelle ein.
3. Natriumionen strömen weiter ein.
4. Natriumkanäle schließen sich. Kaliumkanäle bleiben offen.
5. Positive Kaliumionen strömen durch die offenen Kaliumkanäle weiter aus, ebenso positive Natriumionen (durch die Natriumkanäle vom offen Typ).
6. Kaliumkanäle schließen sich wieder.
7. Ruhepotential wieder erreicht.

Aktionspotential - Ausbreitung



1. Aktionspotential (AP) wird am Axonhügel ausgelöst.
2. Fortpflanzung des APs mit konstanter Amplitude.
3. Verzweigung des APs am Ende des Axons, sogenannter axonaler Baum.
4. Ausschüttung von Neurotransmitter in den Synapsen.

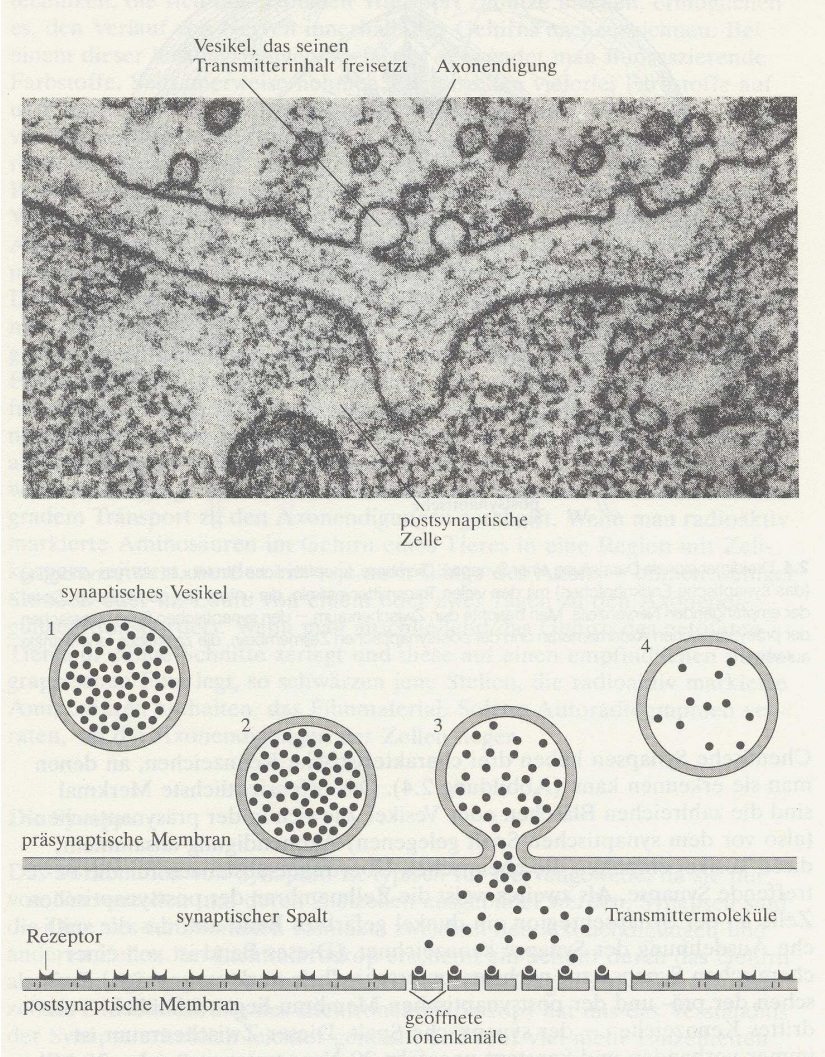
Aufbau einer Synapse



- Präsynaptische Membran (axonales Endknöpfchen des vor der Synapse liegenden Neurons) mit synaptischen Bläschen. Neurotransmitter befindet sich in den synaptischen Bläschen.
- Postsynaptische Membran (Dendrit des nachfolgenden Neurons).
- Neuronen sind getrennt. Synaptischen Spalt sehr klein (einige Nanometer).
- Übertragung des Aktionspotential auf die nachgeschaltete Zelle erfolgt durch Ausschüttung von Neurotransmitter in den synaptischen Spalt.

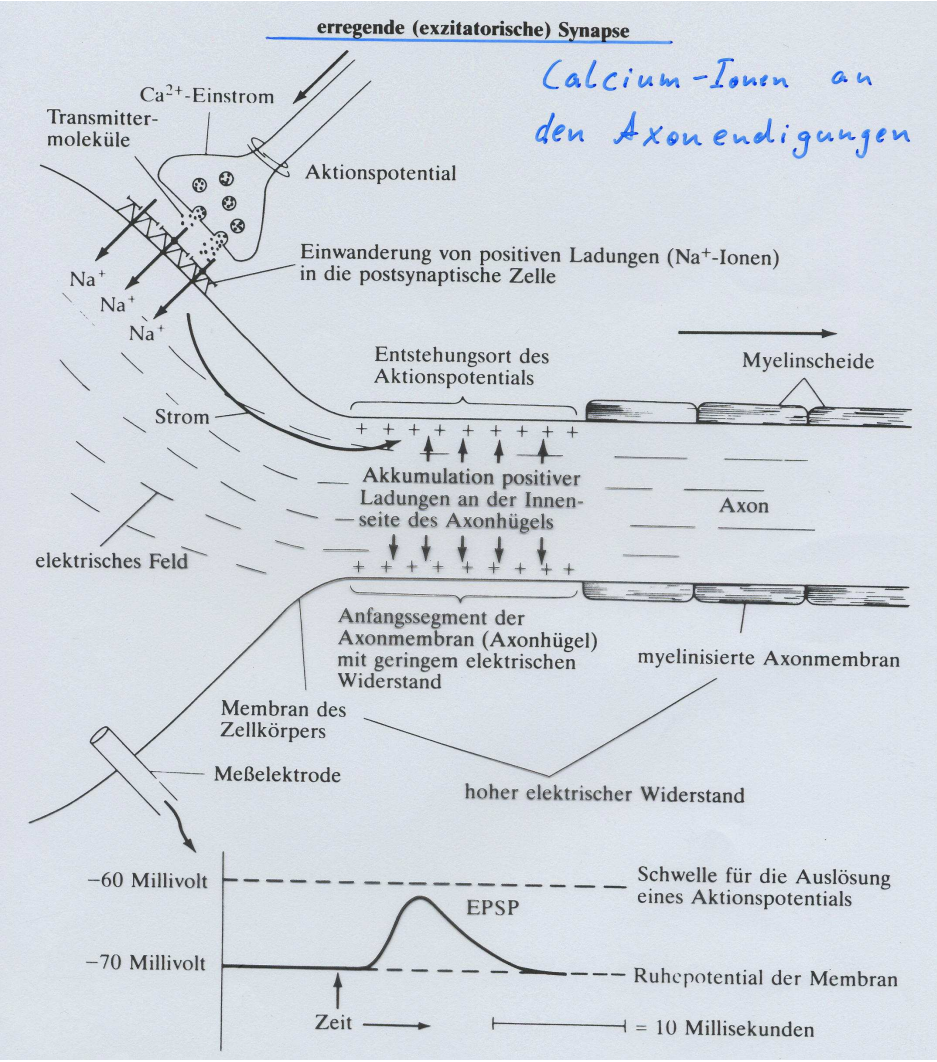
Diese **chemische Synapsen** kommen vor allem in Wirbeltiergehirnen vor.

Synaptische Übertragung



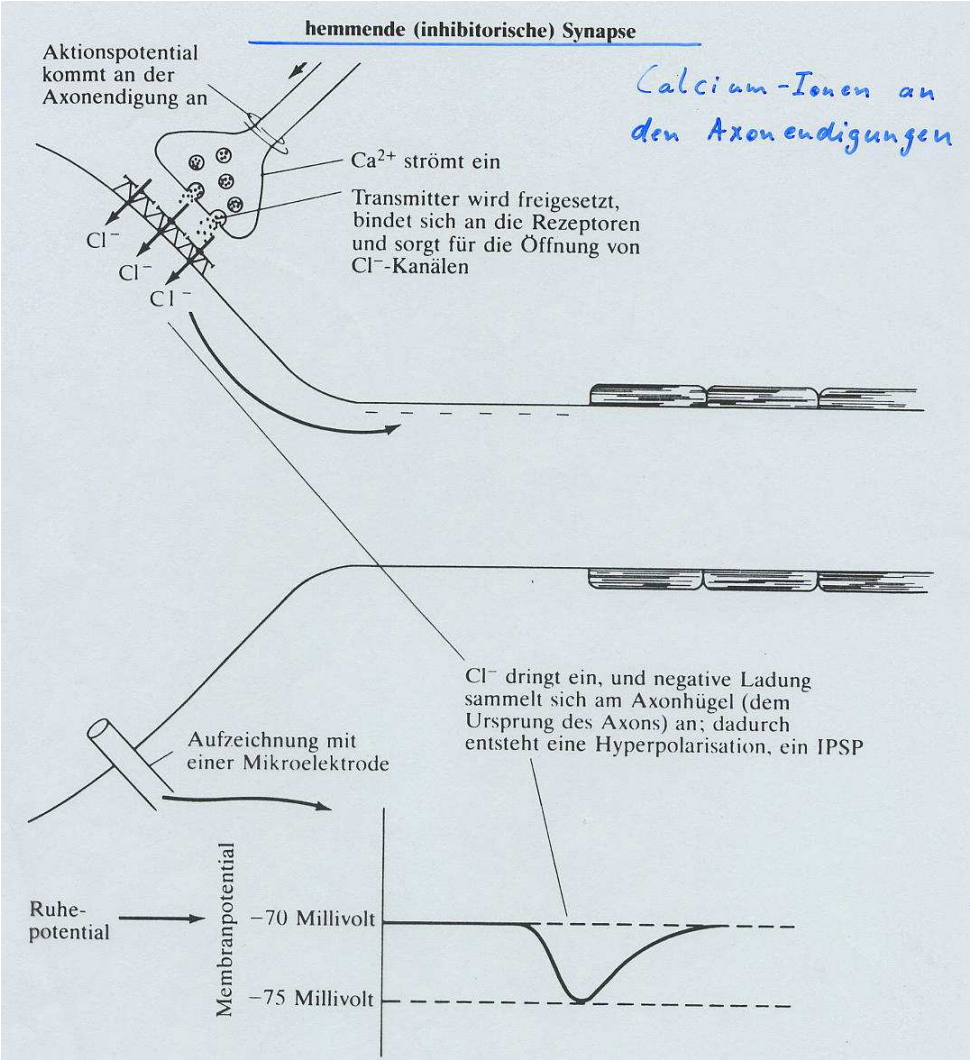
Oben: Elektronenmikroskopische Aufnahme einer Synapse bei der Ausschüttung von Neurotransmitter.

Unten: Sematische Darstellung der Neurotransmitterausschüttung.



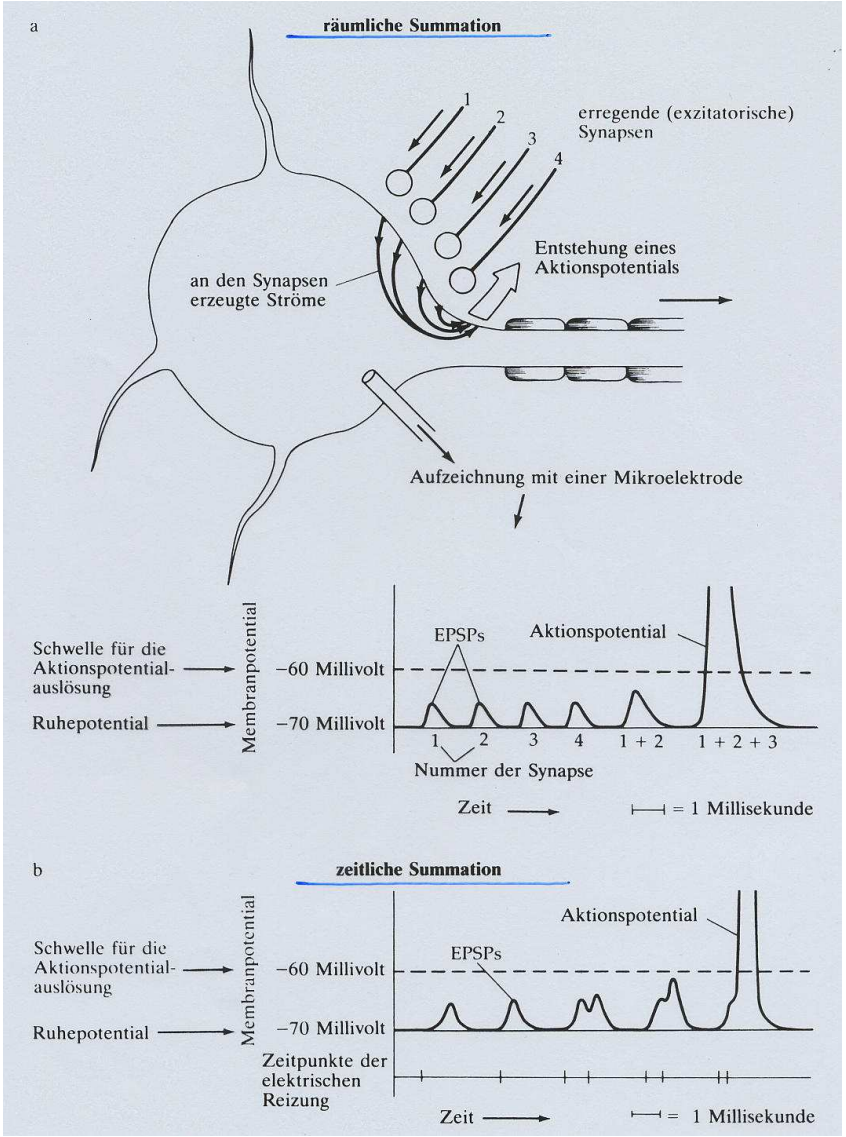
Erregende/Exzitatorische Synapse

- Aktionspotential erreicht die axonalen Endigung
- Exizatorischer Neurotransmitter bewirkt das kurzzeitige Öffnen von Natriumkanälen, so dass im Bereich der Synapse, positiv gelandenen Natriumionen in die Zelle einströmen.
- Dadurch erhöht sich die positive Ladung innerhalb der Zelle.
- Änderung des Potentials der postsynaptischen Zelle (es wird etwas positiver).
- EPSP = exzitatorisches postsynaptisches Potential



Hemmende/Inhibitorische Synapse

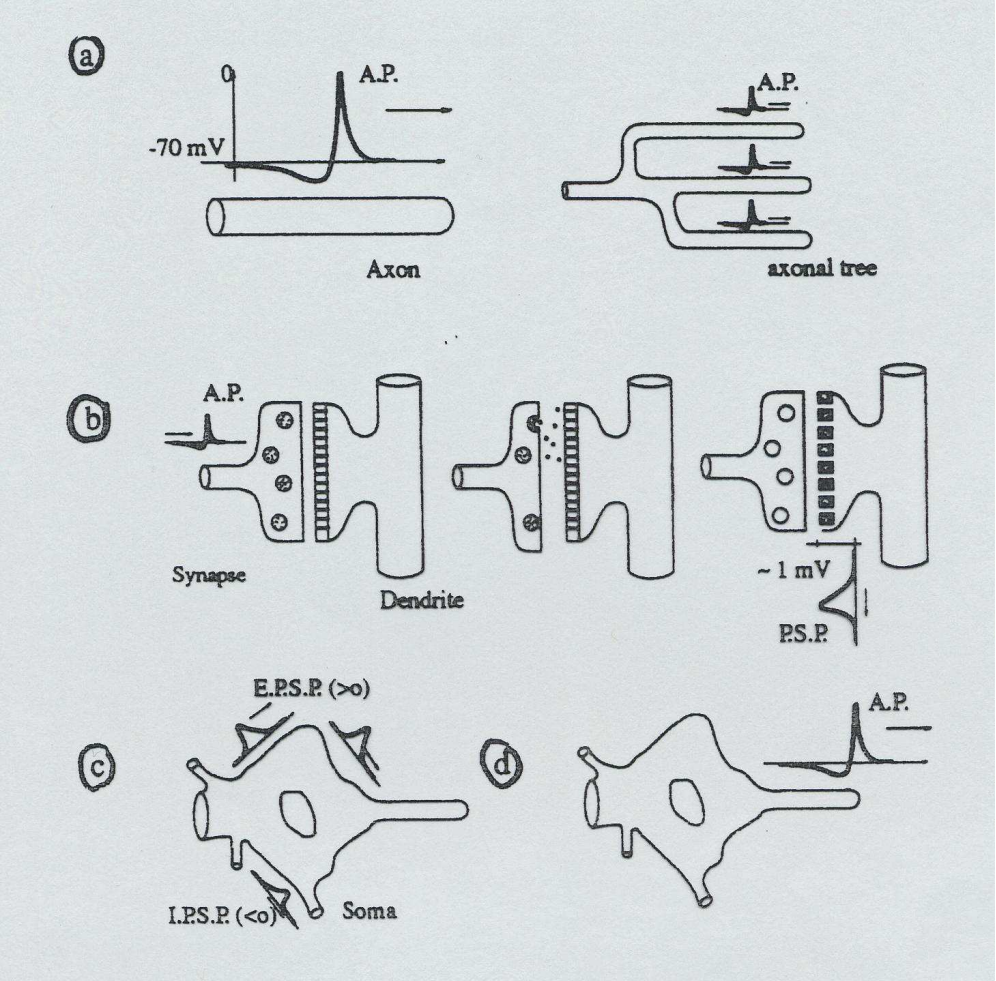
- Aktionspotential erreicht Axonendigung
- Inhibitorischer Neurotransmitter bewirkt, dass z.B. negativ geladene Cloridionen im Bereich der Synapse in die Zelle einströmen können.
- Dadurch erhöht sich die negative Ladung innerhalb der Zelle.
- Änderung des Potentials der postsynaptischen Zelle (es wird etwas negativer).
- IPSP = inhibitorisches postsynaptisches Potential



Räumlich-Zeitliche Summation am Zellkörper

- Durch einzelne EPSPs werden keine Aktionspotentiale ausgelöst.
- Viele EPSPs (räumliche Summation der EPSPs über die Synapsen), die ungefähr gleichzeitig die Zelle erreichen (zeitliche Summation der EPSPs), können die Zelle so stark erregen, dass die **Feuerschwelle** (hier -60mV) erreicht wird.
- Mit dem Erreichen der Feuerschwelle werden Natriumkanäle geöffnet und ein Aktionsspotential (Spike) ausgelöst.

Zyklus der neuronalen Dynamik



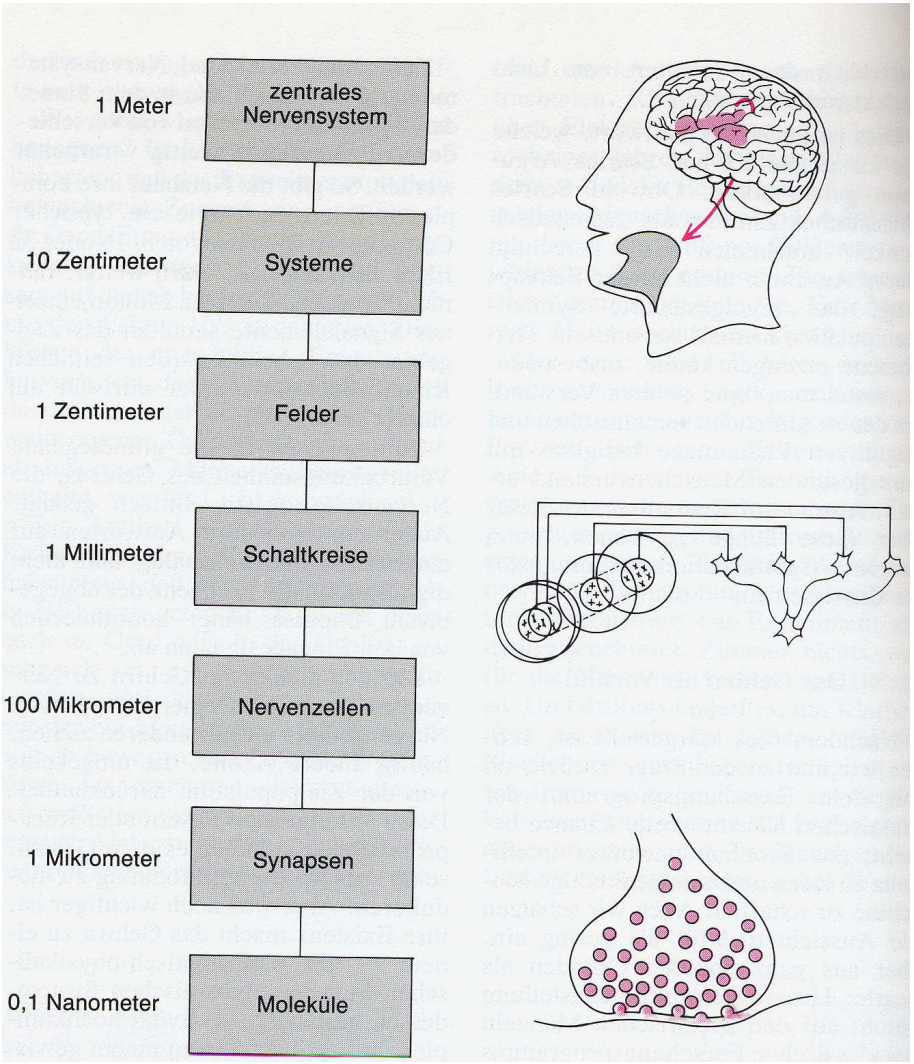
a Ausbreitung des Aktionspotentials auf dem Axon

b Synaptische Übertragung

c Räumlich-Zeitliche Summation der Eingaben

d Auslösung des Aktionspotentials

Längenverhältnisse im Nervensystem



Zusammenfassung

- Warum überhaupt Neuroinformatik?
- Anatomischer Aufbau: Gehirn, Neuron, Synapse,
- Aktionspotential
- Funktionalität der Zellmembran
- Synaptische Übertragung
- Zyklus der Neuronale Dynamik

3. Neuronenmodelle

1. Einleitung zur Modellierung
2. Hodgkin-Huxley-Modell
3. Grundmodell in kontinuierlicher und diskreter Zeit
4. Transferfunktionen (Kennlinien) für Neuronen
5. Neuronenmodelle in Anwendungen

Modellierung neuronaler Netze

- Ein **Neuronenmodell** sollte die relevanten neurophysioligischen Vorgänge der neuronalen Verarbeitung repräsentieren.
- Die **Verknüpfungsstruktur** definiert die Neuronenverbindungen im Netzwerk. Die Verbindungsstruktur kann adaptiv sein und sich durch Lernregeln verändern.
- **Eingänge** und **Ausgänge** für das (Sub)-Netzwerk sollen vorhanden sein, über die es mit anderen (Sub)-Netzen verbunden ist.

Wir unterscheiden

- **Modellierung der neuronalen Dynamik** $=$ zeitliche Änderung der neuronalen Aktivierungen.
- **Modellierung der synaptischen Dynamik** $=$ Änderung der synaptischen Verbindungsstärken (Kapitel 5).

Neuronale Verarbeitung: Zusammenfassung

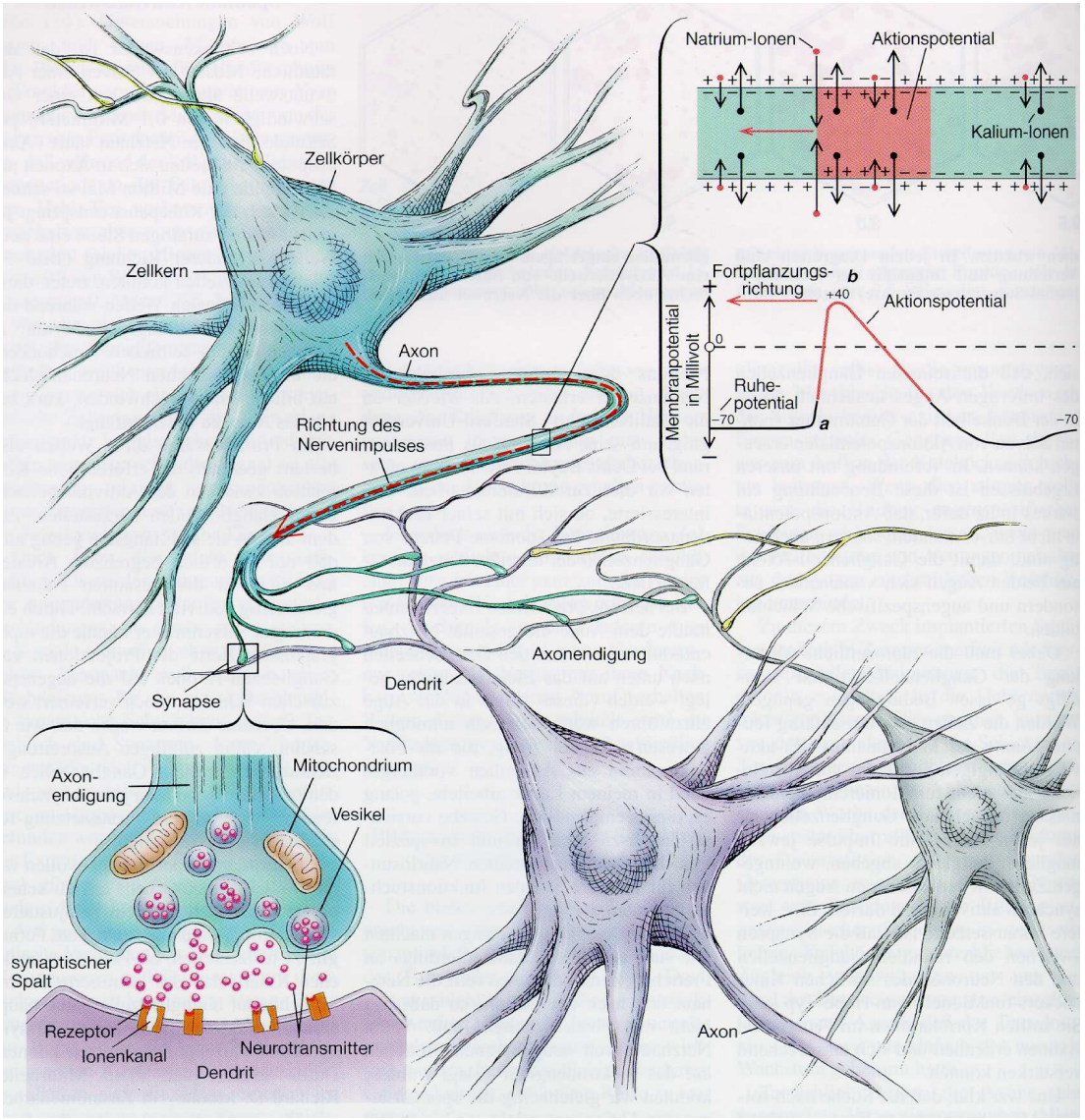
1. Neuronale Netze bestehen aus vielen Einzelbausteinen – den **Neuronen**, die untereinander über **Synapsen** verbunden sind.

2. Neuronen senden über ihre Axon **Aktionspotentiale/Spikes** aus.

3. Neuronen sammeln an ihrem **Zellkörper** und **Dendriten** eingehende Aktionspotentiale (EPSPs und IPSPs) über ihre Synapsen auf. Man spricht von einer **räumlich-zeitlichen Integration/Summation**.

4. Überschreitet die am Dendriten integrierte Aktivität den **Feuerschwellwert** oder kurz **Schwellwert**, so erzeugt das Neuron ein Aktionspotential (Spike). Man sagt: Das Neuron **feuert** oder es ist im **on-Zustand**.

5. Bleibt die am Dendriten und Zellkörper integrierte Aktivität unter dem **Schwellwert**, so erzeugt das Neuron kein Aktionspotential (Spike). Man sagt: Das Neuron ist **still** oder es ist im **off-Zustand**.


Natrium-Ionen
Aktionspotential
Kalium-Ionen
Zellkörper
Zellkern
Axon
Fortpflanzungs-richtung
b
+40
Aktionspotential
Membranpotential in Millivolt
0
Ruhe-potential
−70
a
−70
Richtung des Nervenimpulses
Axonendigung
Dendrit
Synapse
Axon-endigung
Mitochondrium
Vesikel
synaptischer Spalt
Rezeptor
Ionenkanal
Neurotransmitter
Dendrit
Axon

Das Hodgkin-Huxley-Modell

- Von A.L. Hodgkin und A.F. Huxley wurde 1952 ein komplexes Neuronenmodell vorgeschlagen.

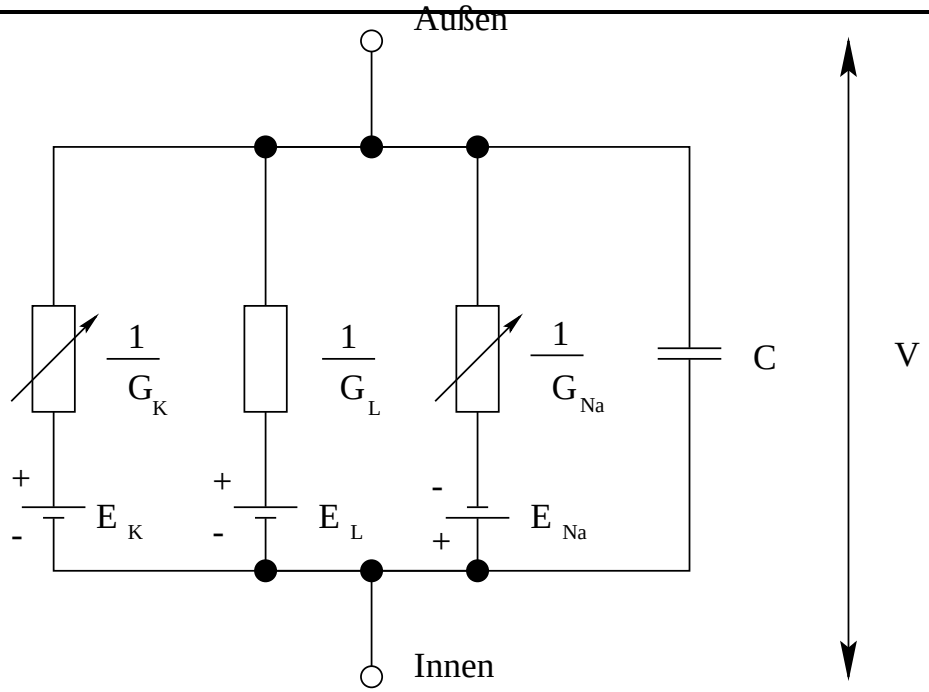
- Grundlage des Hodgkin-Huxley-Modells waren neurophysiologischen Experimenten an der Zellmembran des Riesenaxons des Tintenfisches.

- Erkenntnisse zum Ruhepotential und zur Entstehung und Weiterleitung des Aktionspotential gehen auf diese Experimente zurück.

- Das Hodgkin-Huxley-Modell enthält die wichtigsten elektrophysiologischen Charakteristika (Ströme der Natrium-, Kalium- und Cloridionen) von Zellmembranen und ist in der Lage Aktionspotentiale darzustellen.

- Hodgkin-Huxley-Modell ist für große Netzwerke nicht gut geeignet, da es zu aufwendig.

Hodgkin-Huxley-Modell: Schaltbild



Ersatzschaltbild für Axonmembran

- Kalium-, Natrium- und Cloridionenströme werden modelliert.
- Spannungsabhängiger Membranwiderstand als einstellbarer Widerstand
- Membrankapazität durch Kondensator

Hodgkin-Huxley-Modell: Gleichungen

Gesamtmembranstrom ist Summe der drei beteiligten Ionenströme und der Strom, der die Membran auflädt.

Das Membranpotential V ist durch eine Differentialgleichung beschrieben:

$$C \cdot \frac{dV}{dt} = I_{ext} - I_{Ionen}$$

I_{ext} ein externer Strom und I_{Ionen} die Summe der drei beteiligten Ionenströme I_K (Kalium), I_{Na} (Natrium) und I_L (Leckstrom, hauptsächlich Cloridionen):

$$I_{Ionen} := \underbrace{G_K(V - E_K)}_{I_K} + \underbrace{G_{Na}(V - E_{Na})}_{I_{Na}} + \underbrace{G_L(V - E_L)}_{I_L}.$$

- C_m ist die Kapazität der Zellmembran

- $G_K := g_K \cdot n^4$ die Leitfähigkeit der Zellmembran für Kaliumionen, wobei g_K die Konstante ist, welche die maximal mögliche Leitfähigkeit der Zellmembran für Kaliumionen festlegt

- $G_{Na} := g_{Na} \cdot m^3 \cdot h$ die Leitfähigkeit der Zellmembran für Natriumionen, wobei g_{Na} die Konstante ist, welche die maximal mögliche Leitfähigkeit der Zellmembran für Natriumionen festlegt

- $G_L := g_L = konst$ die Leckleitfähigkeit der übrigen Ionentypen

- E_K das Gleichgewichtspotential der Kaliumionen

- E_{Na} das Gleichgewichtspotential der Natriumionen

- E_L das Gleichgewichtspotential der übrigen Ionentypen

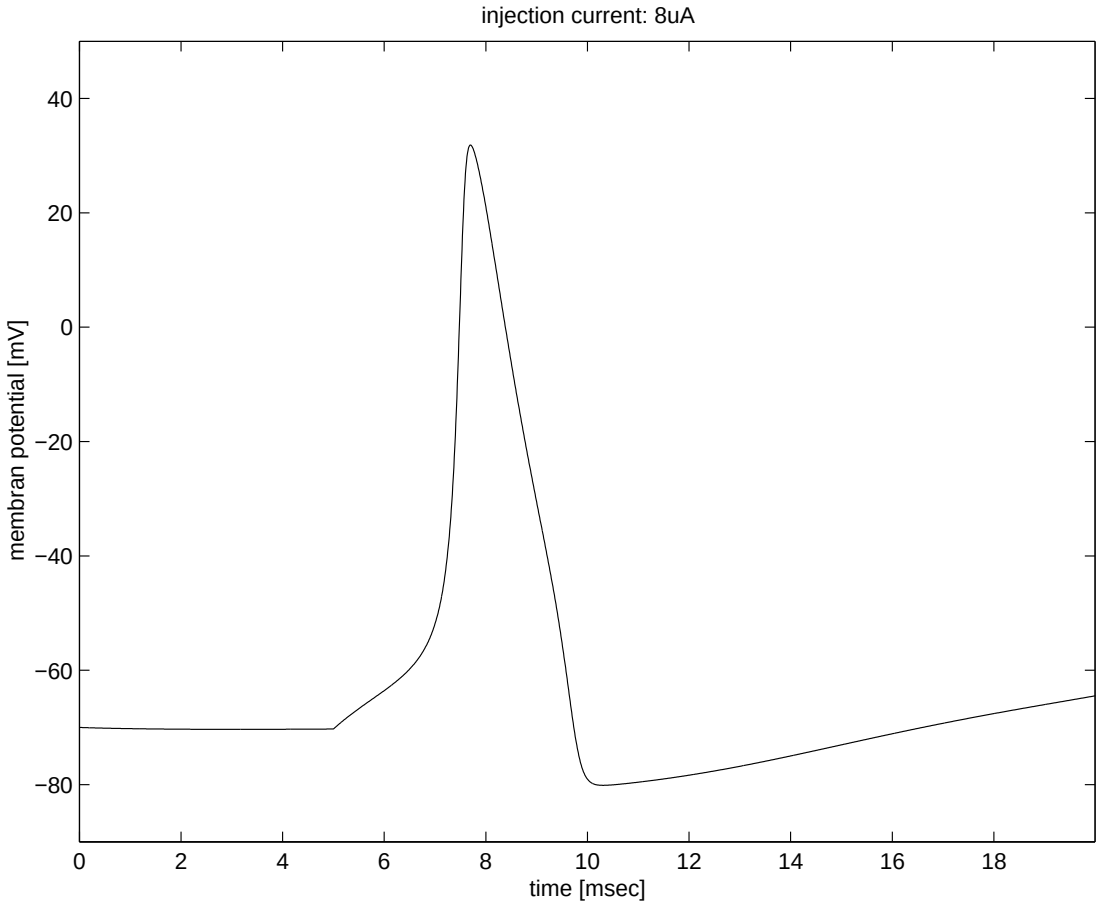
Die Funktionen n, m und h sind DGL vom Typ

$$\frac{dx}{dt} = \alpha_i(V)(1 - x(t)) - \beta_i(V)x(t).$$

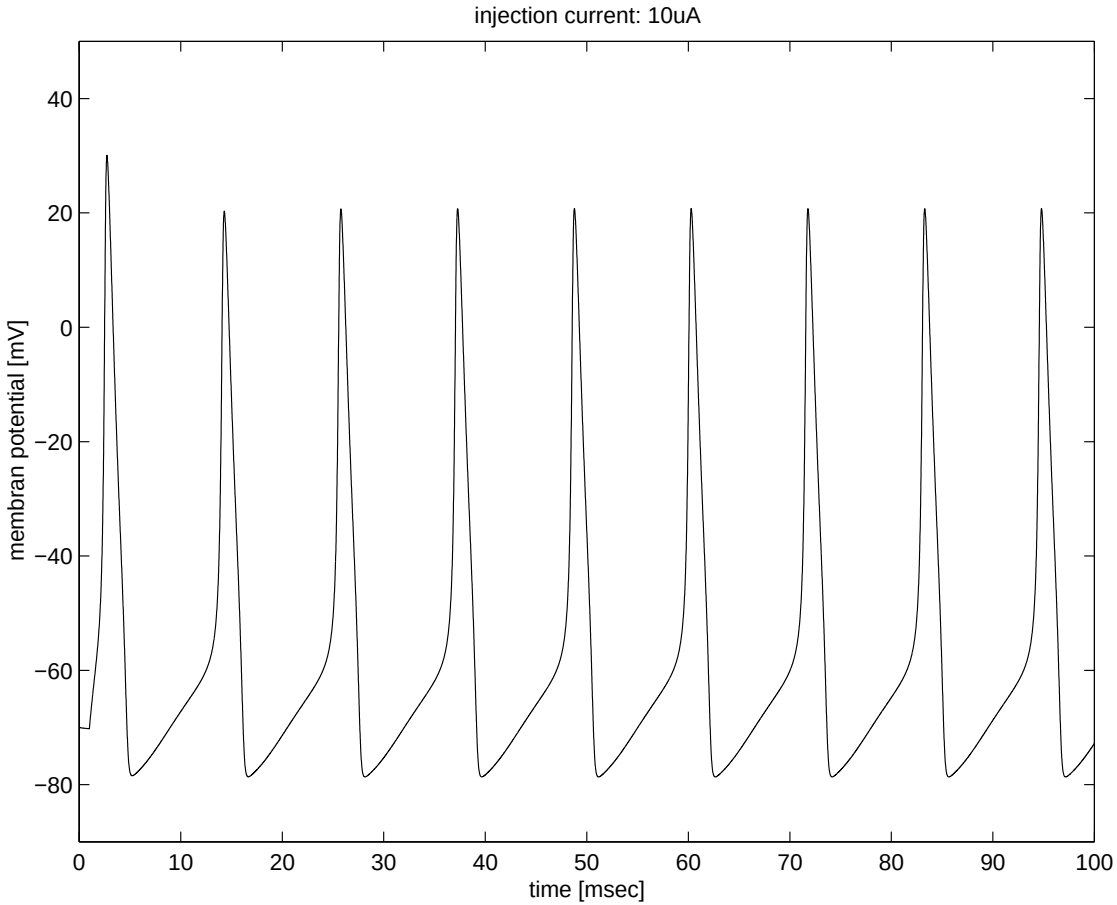
α_i, β_i, $i = n, m, h$, hängen dabei vom Membranpotential V ab

$$\begin{aligned}
\alpha_n(V) &= \frac{0.01(10.0 - V)}{\exp\frac{10.0-V}{10.0} - 1.0} \\
\beta_n(V) &= 0.125 \exp\big(-\frac{V}{80.0}\big) \\
\alpha_m(V) &= \frac{0.1(25.0 - V)}{\exp\frac{25.0-V}{10.0} - 1.0} \\
\beta_m(V) &= 4.0 \exp\big(-\frac{V}{18.0}\big) \\
\alpha_h(V) &= 0.07 \exp\big(-\frac{V}{20.0}\big) \\
\beta_h(V) &= \frac{1.0}{\exp\frac{30.0-V}{10.0} + 1.0}
\end{aligned}$$

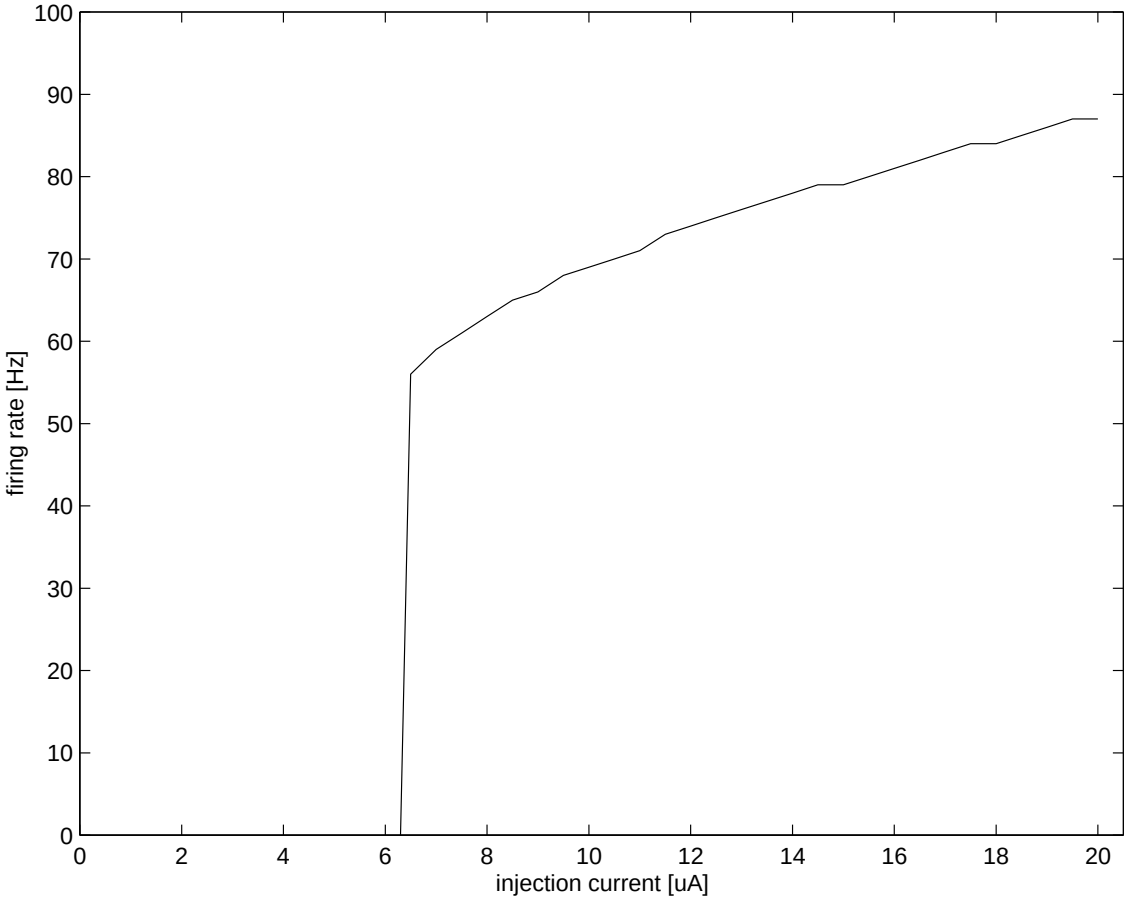
Hodgkin-Huxley-Modell: Simulation



- Eine analytische Lösung dieses gekoppelten nichtlinearen DGL-Systems ist nicht möglich.
- Numerische Näherungsverfahren sind notwendig.
- In der Simulation wurde ein einfaches Näherungsverfahren verwendet (Polygonzugverfahren nach Euler).



- Ruhepotential liegt hier bei etwa -70mV
- Aktionspotentiale $\geq$ 20mV
- Hyperpolarisationsphase (Potential $<$ Ruhepotential).
- Spikedauer ca. 1msec, somit maximale theoretische Spikefrequenz 1000Hz.
- Hier ist das Inter-Spike-Intervall $>$ 10ms, somit ist Spikefrequenz kleiner als 100Hz.



- Unterhalb einer Erregungs-schwelle werden keine Aktions-potentiale erzeugt.
- Auf mittlerem Erregungsniveau erfolgt ein in etwa linearer Anstieg der Feuerrate
- Sättigung der Feuerrate für hohe Erregungsniveaus.

Kontinuierliches Grundmodell

- Keine Modellierung der verschiedenen Ionenkanäle, deshalb sind Aktionspotentiale nicht mehr modellierbar, sondern müssen explizit (durch eine Funktion) modelliert werden.

- Neuron ist durch 2 Modellgleichungen beschrieben.

  1. DGL für die Beschreibung des dendritischen Membranpotentials $u_j(t)$.
  2. Axonales Membranpotential durch Funktionsauswertung $y_j(t) = f(u_j(t))$.

- Synaptische Kopplungen werden durch reelle Zahlen c_{ij} modelliert: exzitatorische Kopplung $c_{ij} > 0$; inhibitorische Kopplung $c_{ij} < 0$.

- Räumlich/Zeitliche Integration erfolgt durch Summation der eingehenden Aktionspotentiale (nach Gewichtung mit den synapischen Kopplungsstärken)

In einem neuronalen Netz mit n Neuronen lauten die Modellgleichungen:

$$\begin{aligned} \tau \dot{u}_j(t) &= -u_j(t) + \underbrace{x_j(t) + \sum_{i=1}^{n} c_{ij} y_i(t - d_{ij})}_{=:e_j(t)} \\ y_j(t) &= f_j(u_j(t)) \end{aligned}$$

- $\tau > 0$ Zeitkonstante
- $u_j(t)$ dendritisches Potential des j-ten Neurons
- $\dot{u}_j(t)$ die (zeitliche) Ableitung von u_j.
- $x_j(t)$ externen Input des Neurons j.
- c_{ij} synaptische Kopplungsstärke von Neuron i zum Neuron j; i.a. ist $c_{ij} \neq c_{ji}$.
- d_{ij} Laufzeit eines Aktionspotentials von Neuron i zum Neuron j.
- $y_j(t)$ axonales Potential des j-ten Neurons.
- f_j Transferfunktion des j-ten Neurons; häufig ist $f_j = f$ für alle Neuronen $j = 1 \ldots, n$ und f eine nichtlineare Funktion.

Diskretes Grundmodell

- Auch dieses DGL-System ist nicht mehr analytisch lösbar (z.B. falls f linear ist und $d_{ij} = 0$).
- Näherungslösung durch Diskretisierung der Zeit
- Annäherung: $\dot{u}_j(t) \approx \frac{u_j(t+\Delta t) - u_j(t)}{\Delta t}$

Einsetzen der Näherung in die DGL liefert:

$$\begin{aligned} \frac{\tau}{\Delta t}(u_j(t+\Delta t) - u_j(t)) &= -u_j(t) + e_j(t) \\ u_j(t+\Delta t) &= (1-\varrho)\cdot u_j(t) + \varrho \cdot e_j(t) \end{aligned} \tag{1}$$

hier ist $\varrho := \frac{\Delta t}{\tau}$, $0 < \varrho \leq 1$. Ferner für das axonale Potential

$$y_j(t) = f(u_j(t)) \tag{2}$$

Transferfunktionen I

1. Die lineare Funktion $f(x) := x$. Ein Neuron mit dieser Transferfunktion heißt *lineares Neuron*.

2. Die Funktion $f(x) := H(x - \theta)$ mit der *Heaviside-Funktion* H. Die Heaviside-Funktion H nimmt für $x \geq 0$ den Wert $H(x) = 1$ und für $x < 0$ den Wert $H(x) = 0$ an.

3. Die beschränkte stückweise lineare Funktionen:

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \in [0, 1] \\ 1 & x > 1 \end{cases}$$

4. Die Funktion $f(x) := F_\beta(x)$ mit $F_\beta(x) = 1/(1 + \exp(-\beta x))$, wobei $\beta > 0$ ist. F_β nennt man auch *logistische* Funktion.

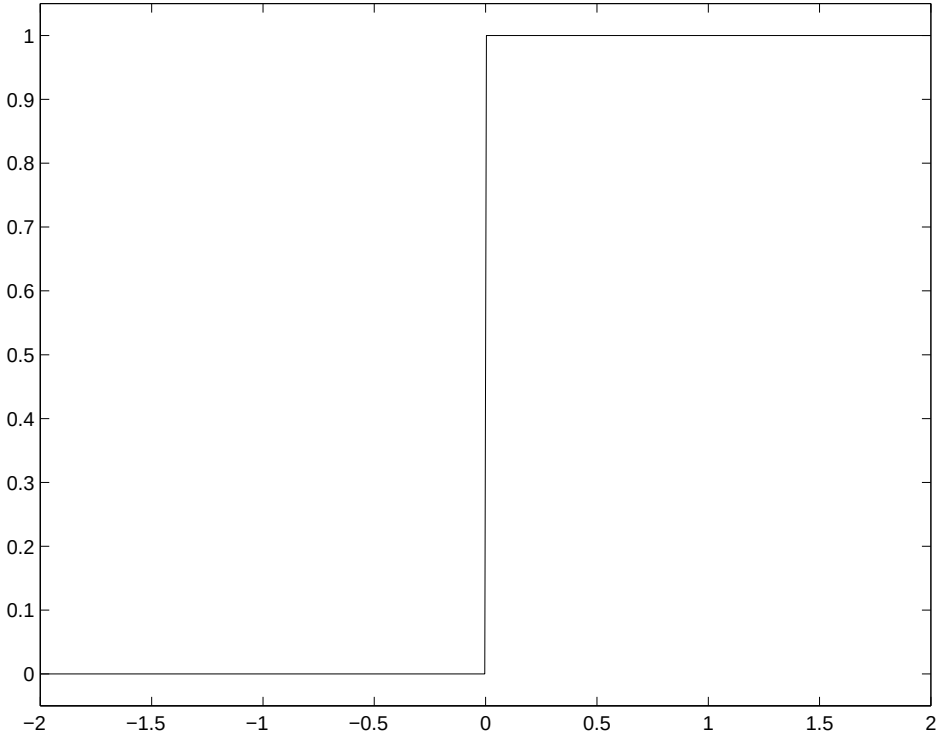
Kodierung der Neuronenausgabe

Das axonale Potential bei den Modellneuronen mit den Transferfunktionen 2.)-4.) liegt im Wertebereich $[0, 1]$. Bei einige neuronalen Modellierungsansätzen (z.B. Hopfield-Netzwerke) wird der Wertebereich der Neuronenausgabe auch als $[-1, 1]$ angenommen.
Eine Transformation $G(x) := 2F(ax + b) - 1$, mit $a > 0$ und $b \in \mathbb{R}$, macht aus einer Transferfunktion F mit Wertebereich $[0, 1]$ eine Transferfunktion mit Wertebereich $[-1, 1]$.

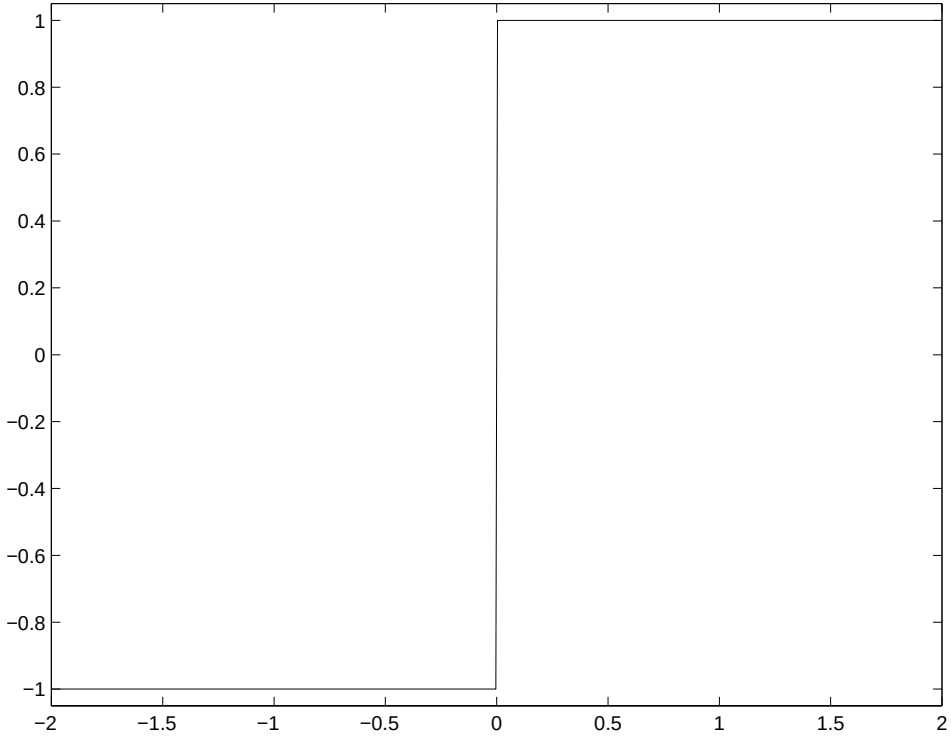
Zum Beispiel:

- Für $a = 1$ und $b = 0$, so wird durch obige Transformation aus der Heaviside-Funktion die Vorzeichen-Funktion.

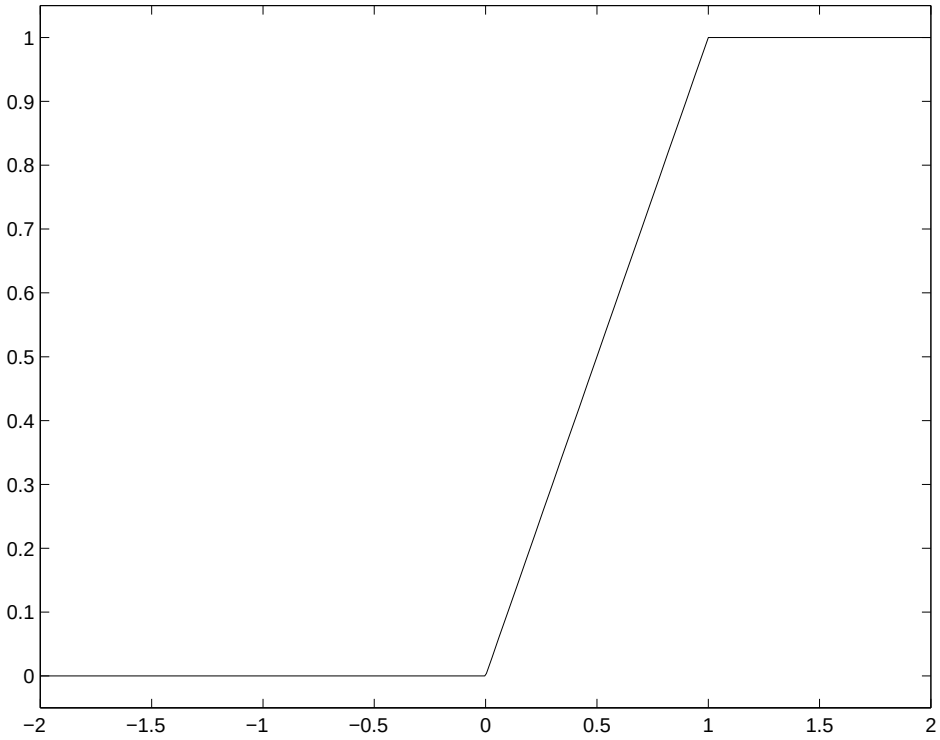
- Für $a = 1/2$ und $b = 0$, so wird aus der Fermi-Funktion die Tangens-Hyperbolikus-Funktion.



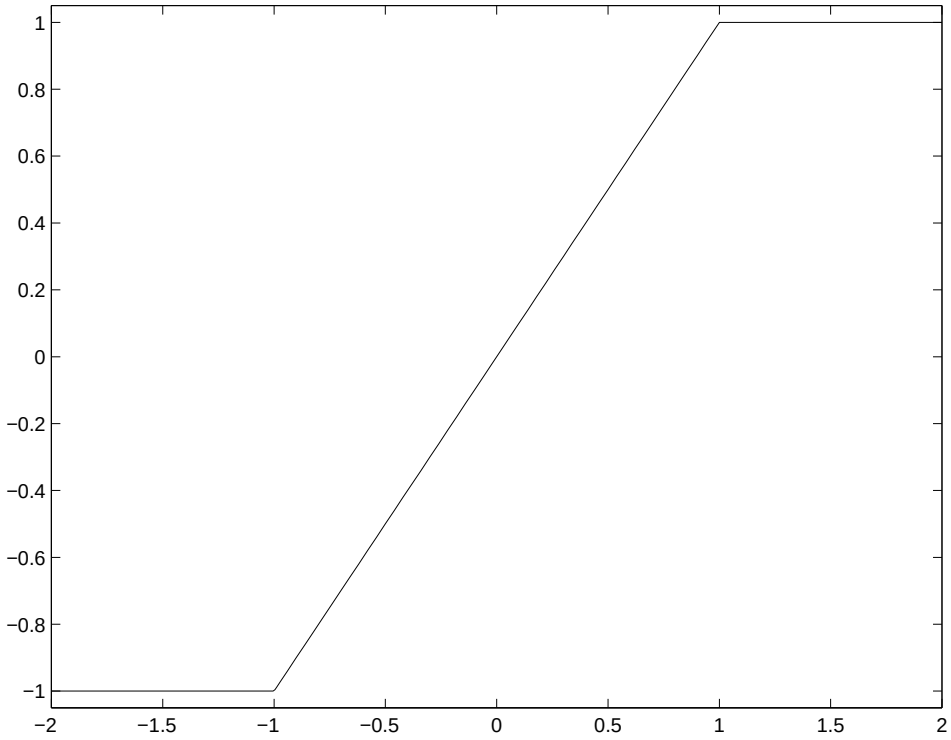
Heaviside-Funktion.



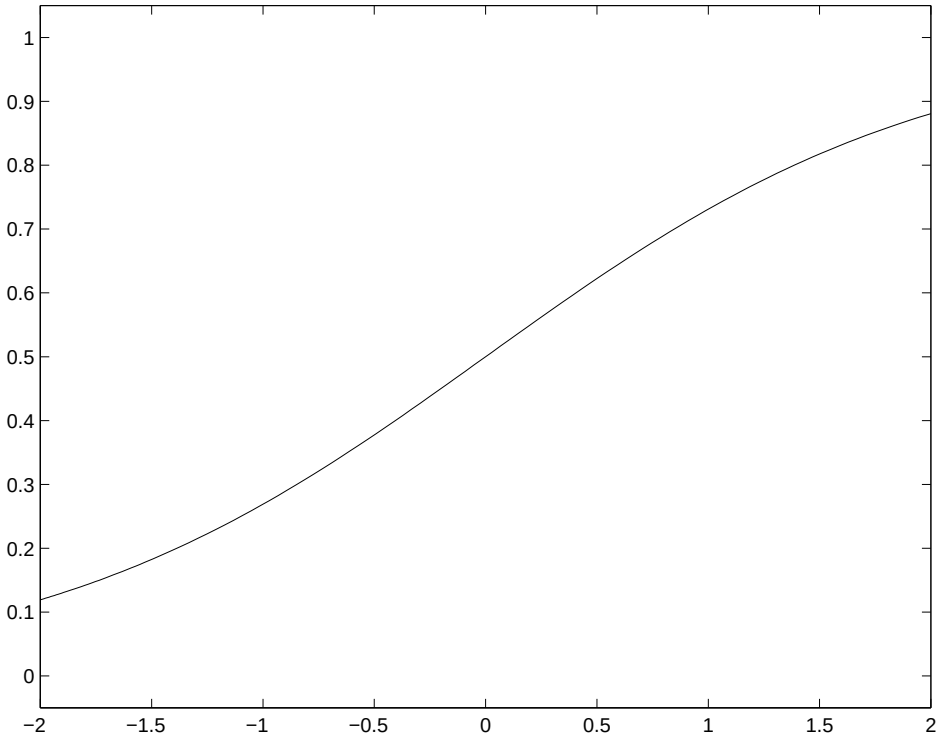
Signum-Funktion.



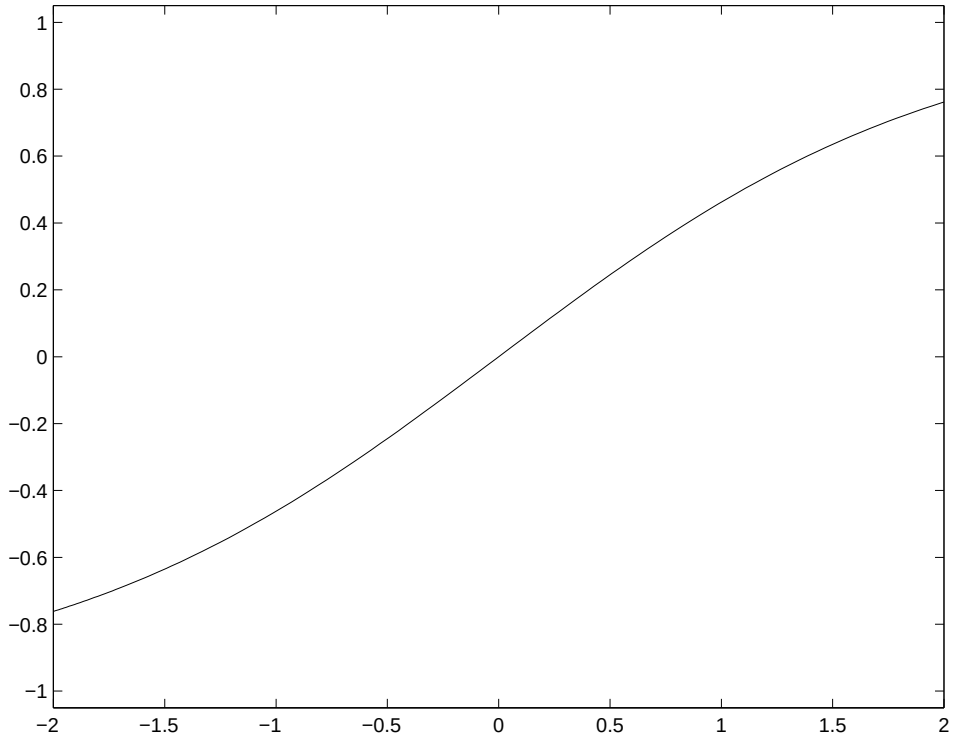
Begrenzte lineare Funktion.



Sym. begrenzte lineare Funktion.

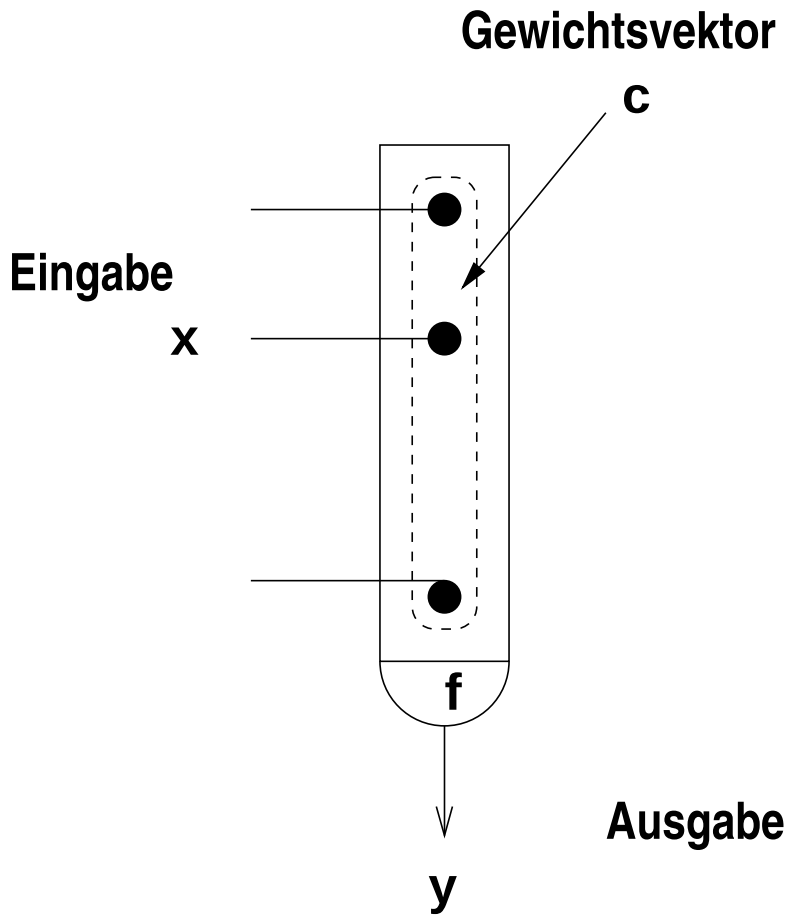


Fermi-Funktion, $\beta = 1$.



Hyperbolischer Tangens, $\gamma = 1$.

Neuronenmodelle in Anwendungen



Vereinfachtes Neuonenmodell

- Neuronen ohne Gedächtnis, d.h. $\varrho = 1$.
- Signallaufzeiten werden vernachlässigt, d.h. $d_{ij} = 0$

Beispiele I

- **Lineares Neuron** mit Gewichtsvektor $\mathbf{c}$ und Konstante θ

$$y(x) = \langle \mathbf{x}, \mathbf{c} \rangle + \theta = \sum_{i=1}^{n} x_i c_i + \theta$$

- **Schwellwertneuron** mit Gewichtsvektor $\mathbf{c}$ und Schwellwert θ

$$y(x) = \begin{cases} 1 & \langle \mathbf{x}, \mathbf{c} \rangle \geq \theta \\ 0 & \text{sonst} \end{cases}$$

- **Kontinuierliches nichtlineares Neuron** mit Gewichtsvektor $\mathbf{c}$ und Konstante θ

$$y(x) = f(\langle \mathbf{x}, \mathbf{c} \rangle + \theta), \quad f(s) = \frac{1}{1 + \exp(-s)}$$

Beispiele II

Die Neuronenmodelle basierten bisher auf der Berechnung der gewichteten Summe zwischen einem Vektor $\mathbf{x}$ und dem synaptischen Gewichtsvektor $\mathbf{c}$ (dies ist das Skalarprodukt $\langle \mathbf{x}, \mathbf{c} \rangle$).

In der Praxis sind auch distanzberechnende Neuronenmodelle gebräuchlich, wir werden diese später bei den Netzen zur Approximation und Klassifikation, sowie bei den Netzen zur neuronalen Clusteranalyse genauer kennenlernen.

- **Distanzberechnendes Neuron**

$$y = \|\mathbf{x} - \mathbf{c}\|_2$$

- **Radial symmetrisches Neuron**

$$y = h(\|\mathbf{x} - \mathbf{c}\|_2), \quad h(r) = \exp(-\frac{r^2}{2\sigma^2})$$

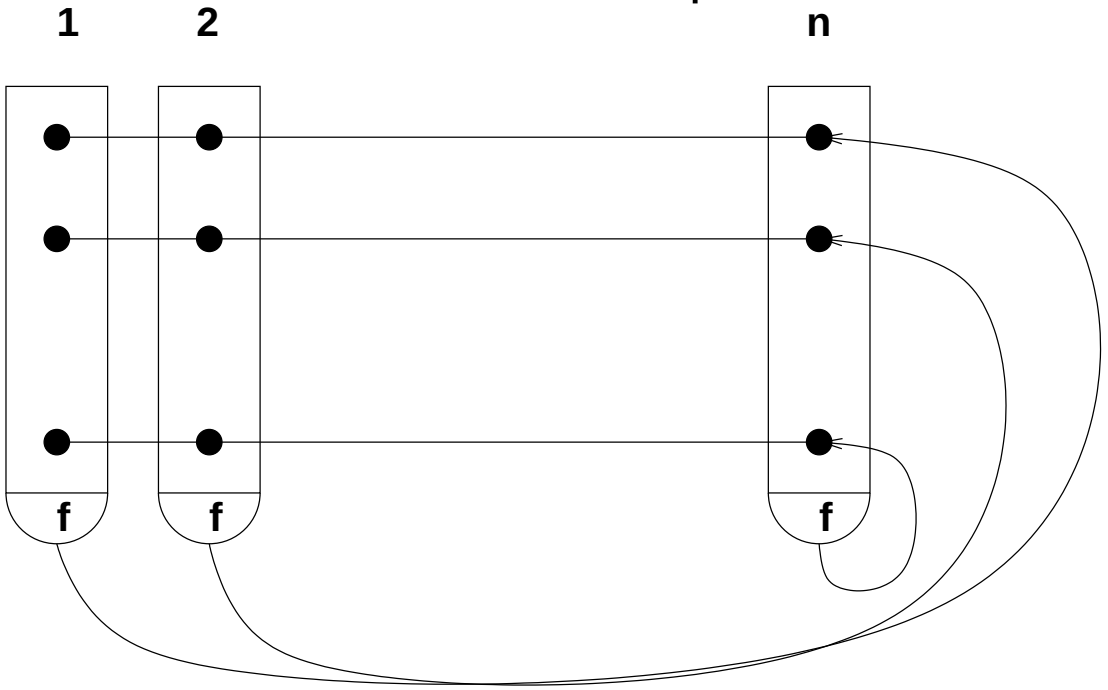
4. Neuronale Architekturen

- Rückgekoppelte neuronale Netzwerke
- Vorwärtsgekoppelte neuronale Netzwerke
- Geschichtete neuronale Netze

Rückgekoppelte Netze

Bei **rückgekoppelten neuronalen Netzwerken** (recurrent/feedback neural networks) sind die Neuronen beliebig miteinander verbunden — potentiell ist jedes Neuron mit jedem anderen Neuron verbunden.

Kopplungsmatrix $\mathbf{C} \in \mathbb{R}^{n^2}$ hat also keine spezielle Struktur.



Vorwärtsgekoppelte neuronale Netze

- Bei den **vorwärtsgekoppelten neuronalen Netzen** ist der Informationsverarbeitungsfluss gerichtet, und zwar von den Eingabeneuronen zu den Ausgabeneuronen.

- Die **Verarbeitung** erfolgt **sequentiell** durch mehrere Neuronen bzw. Neuronenschichten.

- Der Verbindungsgraph der Neuronen, repräsentiert durch die **Kopplungsmatrix** C, ist **zyklenfrei**.

- In Anwendungen, sind die diese **geschichteten Netze** oder **layered neural networks** von großer Bedeutung.

Geschichtete neuronale Netze

- Neuronen der **Eingabeschicht** propagieren (sensorische) Eingaben in das Netz, diese führen keine neuronalen Berechnungen durch. Deshalb wird diese Schicht meist auch nicht mitgezählt (wie bei Boole'schen Schaltnetzen). Man kann sie auch als 0-te Neuronenschicht bezeichnen.

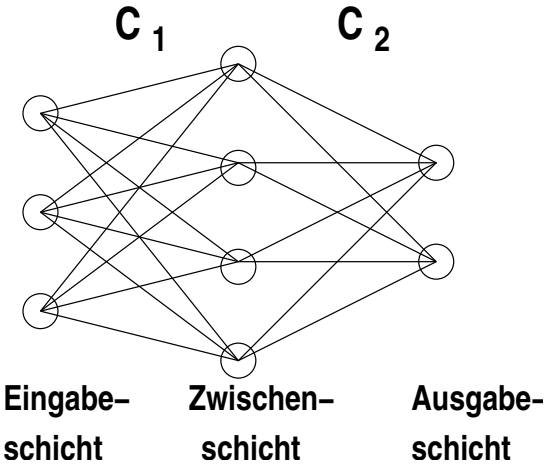
- Die Neuronen in der Schicht k erhalten ihre Eingaben von Neuronen der Schicht $k-1$ und geben ihre Berechnung an die Neuronen der Schicht $k+1$ weiter.
  Diese Schichten nennen wir **Zwischenschicht** oder **versteckte Schichte** oder **hidden layer**.

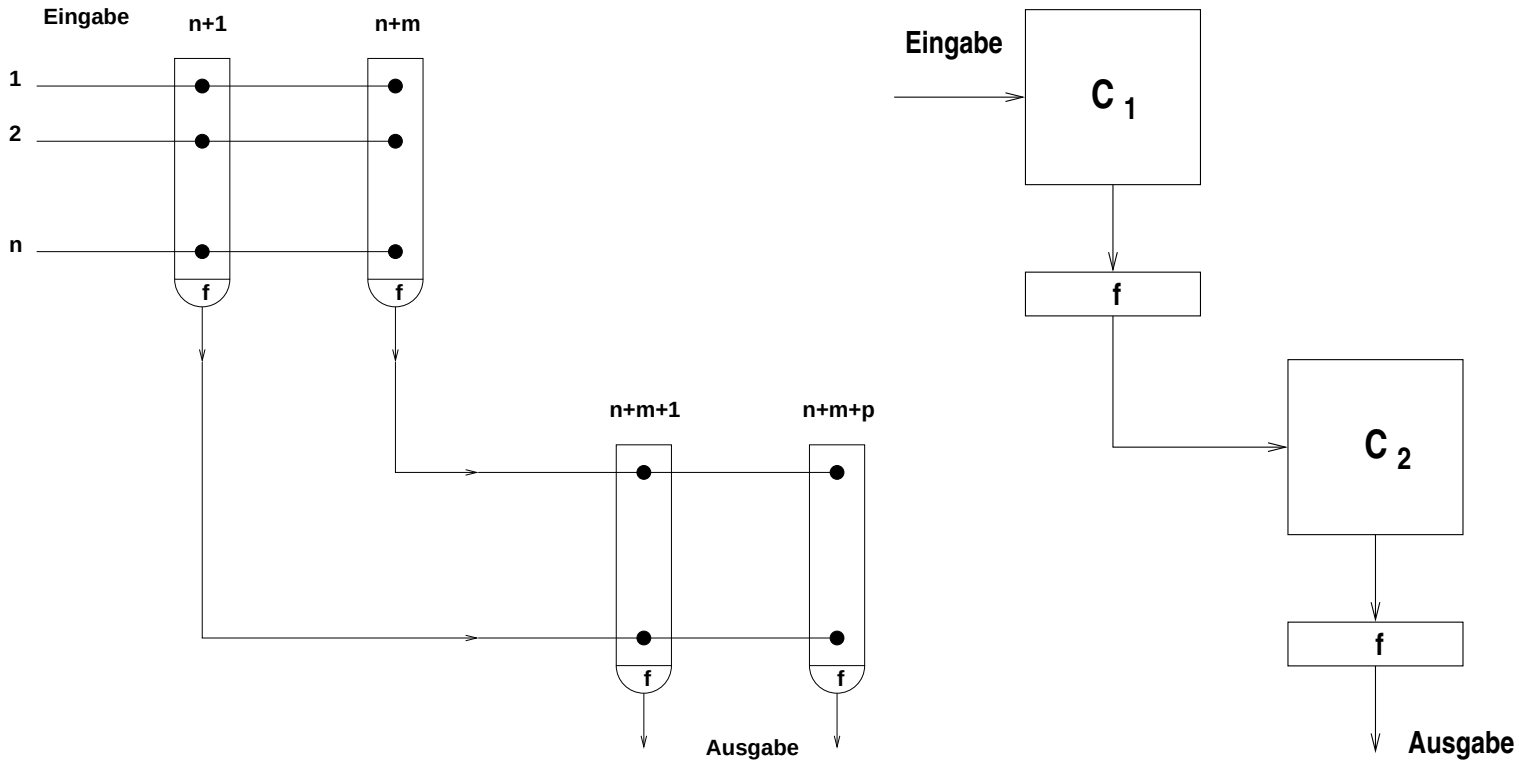
- Die Neuronen ohne Nachfolgeneuron bilden die **Ausgabeschicht**.

Darstellung von Mehrschicht-Netzen

- In der Literatur findet man zahlreiche Darstellungsformen von neuronalen Netzwerken. Sehr verbreitet ist die **Graph-Notation** mit einer Knoten- und Kantenmenge.

- Knoten entsprechen den Neuronen und die synaptischen Kopplungen sind als Kanten dargestellt.





Schematische Darstellung von Einzelneuronen (links) mit den synaptischen Kopplungen (dargestellt als •). Die Neurondarstellung besteht aus dem Dendriten (Rechteck) mit den synaptischen Kopplungen c_{ij}, dem Zellkörper (Halbkreis) mit der Transferfunktion f und dem Axon (ausgehender Pfeil).

In der linken Abbildung ist ein Netz mit $N := n + m + p$ Neuronen gezeigt:

- n Neuronen in der Eingabeschicht, diese werden als Eingabe-Neuronen nicht weiter betrachtet, da sie keine neuronale Berechung durchführen.

- m Neuronen in der Zwischenschicht

- p Neuronen in der Ausgabeschicht

In der rechten Abbildung sind die synaptischen Kopplungen als Matrizen zusammengefasst. Die neuronale Transferfunktion f ist hier als vektorwertige Funktion zu interpretieren. Entlang der Pfeile werden Vektoren $\in \mathbb{R}^n$, $\mathbb{R}^m$ bzw. $\mathbb{R}^p$ prozessiert.

Die Kopplungsmatrix $C \in \mathbb{R}^{N^2}$ des Gesamtnetzes hat die Form

$$C = \begin{pmatrix} 0 & \mathbf{C}_1 & 0 \\ 0 & 0 & \mathbf{C}_2 \\ 0 & 0 & 0 \end{pmatrix} \qquad \text{mit } \mathbf{C}_1 \in \mathbb{R}^n \times \mathbb{R}^m \text{ und } \mathbf{C}_2 \in \mathbb{R}^m \times \mathbb{R}^p.$$

5. Lernregeln für neuronale Netze

1. Allgemeine Lokale Lernregeln

2. Lernregeln aus Zielfunktionen: Optimierung durch Gradientenverfahren

3. Beispiel: Überwachtes Lernen im Einschicht-Netz

Einleitung

- Unter **Lernen** versteht man den Erwerb neuen Wissens, unter **Gedächtnis** die Fähigkeit, dieses Wissen so zu lernen, dass es wiederfindbar ist.

- Mechanismen neuronalen Lernens sind deutlich weniger erforscht als die neurophysiolgischen Vorgänge bei der neuronalen Dynamik (Ruhepotential, Aktionspotential, Ionen-Kanäle, etc).

- Mögliche Formen des neuronalen Lernens:
  - Änderung der Verbindungsstruktur: Anzahl der Neuronen/Neuronenschichten
  - Änderung der Verbindungsstärken

Wir betrachten vor allem Lernen durch Änderung der Verbindungsstärken.

Lernen durch Änderung der Synapsenstärken

- Die Änderung der Synapsenstärken lässt sich wieder durch Differntialgleichungen bzw. Differenzengleichungen (Iterationsgleichungen) beschreiben. Analog zur Beschreibung der neuronalen Dynamik (in kontinuierlicher oder diskreter Zeit).

- Änderung der synaptischen Kopplungen vollzieht sich langsamer, als die Änderung der neuronalen Zustände.
  Dies ist beispielsweise durch verschiedene Zeitkonstanten in den Differntialgleichungen modellierbar.

- In praktischen Anwendungen wird vor allem der Fall betrachten, in dem der Lernprozess vor der eigentlichen Anwendungsphase erfolgt, und dann später, in der Anwendung, kein Lernen mehr stattfinden.

Aufwand beim Lernen in einem Netz mit n Neuronen:

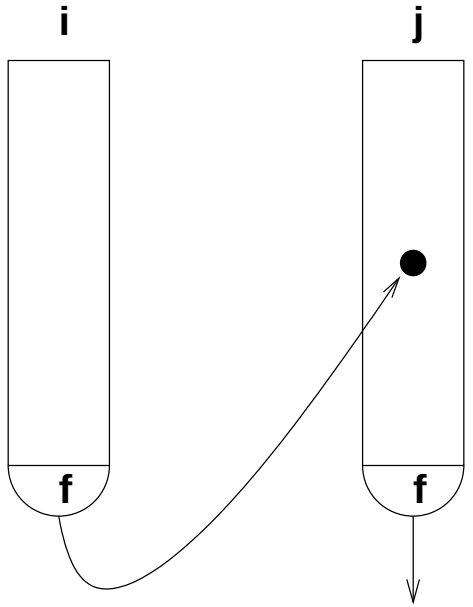
- Für die neuronale Dynamik: Je Neuron wird das dendritsche Potential $x_j(t)$, in Abhängigkeit der axonalen Potentiale $y_i(t - d_{ij})$ aller n Neuronen, berechnet.
  Schließlich wird das axonale Potential $y_j(t) = f(x_j(t))$ bestimmt.
  Wenn man für die Funktionsauswertung konstanten Aufwand (Funktion als Tabelle gespeichert) annimmt, so sind n^2 Multiplikationen auszuführen, um die Zustände aller n Neuronen zu berechnen.

- Für die synaptische Dynamik: Setzt man pro Synapse eine DGl an, so sind insgesamt schon n^2 Operationen durchzuführen, selbst wenn der Aufwand pro Synapse konstant ist.

Allgemeine Lokale Lernregel

Änderung einer synaptischen Kopplung c_{ij} soll durch die **lokal** vorhandenen Größen bestimmt werden.

Globale Netzwerkzustände sollen dabei nicht berücksichtigt werden (erfordern größeren Aufwand und sind auch biologisch nicht plausibel)



c_{ij}

modifizierbare Synapse

Verbindung von Neuron i zum Neuron j

Allgemeine lokale Lernregel als Differenzengleichung:

$$\Delta c_{ij}(t) = c_{ij}(t) - c_{ij}(t - \Delta t) = -v(t)c_{ij}(t) + l(t)\big(y_i(t) - ac_{ij}(t) - b\big)\big(\delta_j(t) - c\big)$$

Hierbei sind:

- $\Delta c_{ij}(t) := c_{ij}(t) - c_{ij}(t - \Delta t)$; (meist wird $\Delta t = 1$ gesetzt)
- $v(t) \geq 0$ eine Vergessensrate. In Anwendungen typischerweise $v(t) = 0$.
- $l(t) \geq 0$ eine Lernrate. Gebräuchlich ist $l(t) > 0$ und $l(t)$ monoton gegen 0 fallend.
- $a, b, c \geq 0$ Konstanten. Häufig $a = b = c = 0$.
- Die Form von $\delta_j(t)$ wird durch das Lernverfahren bestimmt.
  Wir unterscheiden **unüberwachtes** und **überwachtes** Lernen (unsupervised and supervised learning).

Unüberwachtes Lernen

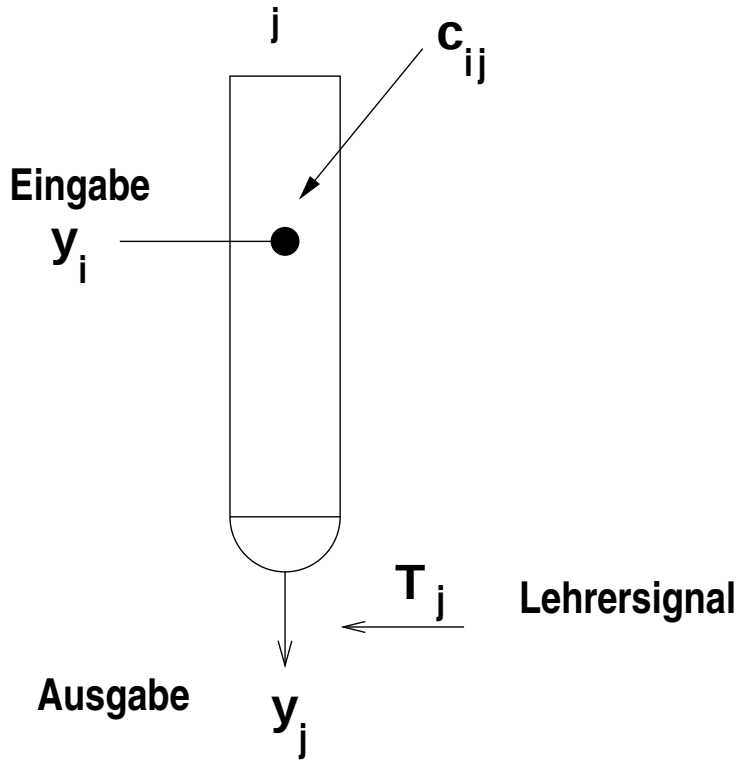
Im unüberwachten Lernen ist $\delta_j(t) = u_j(t)$ (dendritisches Potential) oder $\delta_j(t) = y_j(t)$ (axonales Potential) des postsynaptischen Neurons, also beispielsweise

$$\Delta c_{ij}(t) = l(t) y_i(t) y_j(t)$$

Dieses ist eine sehr bekannte Lernregel. Sie wird zu Ehren von Donald Hebb auch **Hebb'sche Lernregel** genannt.

Postulat von Hebb (1949):
Wenn das Axon der Zelle A nahe genug ist, um eine Zelle B zu erregen, geschieht ein Wachstumsprozess in einer oder in beiden Zellen, so dass die Effizenz von A, als eine auf B feuernde Zelle wächst.

Überwachtes Lernen



- **Lehrersignal** T_j des j-ten Neurons.
- Ausgabe y_j des j-ten Neurons.
- Die Ausgabe y_i beeinflusst auch die Neuronenausgabe y_j (durch c_{ij}).
- Idee: Die Ausgabe y_j soll durch Änderung der synaptischen Kopplungsstärke c_{ij} möglichst auf das Lehrersignal T_j geregelt werden.

Delta Lernregel

Im überwachten Lernen mit der Delta-Regel ist $\delta_j(t) = T_j - y_j(t)$ also:

$$\Delta c_{ij}(t) = l(t) y_i(t) \big(T_j(t) - y_j(t)\big)$$

hierbei ist $T_j(t)$ ein externes Lehrersignal bzw. Sollwert für das Neuron j zur Zeit t.

Lernregeln dieser Art, die einen Vergleich zwischen einem Lehrersignal und der Neuronenausgabe berechnen, werden **Delta-Regeln** genannt.

Viele gebräuchliche Lernregeln sind von diesem Typ — die bekannteste ist die **Perzeptron-Lernregel**.
Sehr viele Lernregeln sind lokal. Wir werden allerdings auch später nicht-lokale Lernregeln kennenlernen.

Verhalten einer Delta-Lerngel für ein nichtlineares kontinuierliches Neuron mit der Ausgabe

$$y_j = f(\sum_{i=1}^{n} c_{ij} y_i), \qquad u_j = \sum_{i=1}^{n} c_{ij} y_i$$

f differenzierbare monoton wachsend, z.B. $f(x) = \frac{1}{1+e^{-x}}$.

Die Änderung der Ausgabe von y_j bei der Delta-Lernregel:

$$\begin{aligned} y_j^{neu} - y_j &= f(\sum_{i=1}^{n} c_{ij}^{neu} y_i) - f(\sum_{i=1}^{n} c_{ij} y_i) \\ &= f'(z)\ (u_j^{neu} - u_j) \qquad z \in (u_j^{neu}, u_j) \\ &= f'(z)\ (c_{ij}^{neu} - c_{ij}) y_i \\ &= l\ f'(z)\ (T_j - y_j)\ y_i^2 \end{aligned}$$

Also: $y_j^{neu} = y_j + l y_i^2 f'(z)(T_j - y_j)$, wobei $l y_i^2 f'(z) > 0$.

Fallunterscheidungen bei der Delta-Regel:

- $T_j = y_j$, dann bleiben c_{ij} und y_j unverändert.
- $y_j > T_j$, also $T_j - y_j < 0$, dann ist $\Delta c_{ij} < 0$, also $y_j^{neu} - y_j < 0$, dh. $y_j^{neu} < y_j$
- $y_j < T_j$, also $T_j - y_j > 0$, dann ist $\Delta c_{ij} > 0$, also $y_j^{neu} - y_j > 0$, dh. $y_j^{neu} > y_j$

Falls die Lernrate $l > 0$ klein ist, so ist nach einem Lernschritt mit der Delta-Lernregel der Fehler $T - y$ kleiner geworden.

Die Delta-Regel zeigt das geforderte Verhalten, nämlich den Fehler zwischen Lehrersignal und Netzausgabe zu minimieren.

Wie können nun Lernregeln systematisch hergeleitet werden?

Konstruktion von Lernregeln

Wir gehen nun davon aus, dass uns eine Menge von Eingabesignalen(vektoren) vorliegt, zu denen ebenfalls Sollausgaben/Lehrersignale definiert sind.

Wir betrachten überwachte Lernverfahren, hier ist das Ziel dadurch definert, dass die Netzausgabe jeweils dem Lehrersignal möglichst nahe kommen soll!

Sei also eine Trainingsmenge gegeben durch Ein-Ausgabepaare:

$$\mathcal{T} = \{(\mathbf{x}^\mu, \mathbf{T}^\mu) \mid \mathbf{x}^\mu \in \mathbb{R}^d, \mathbf{T}^\mu \in \mathbb{R}^n;\ \mu = 1, \ldots, M\}$$

Gesucht ist nun ein neuronales Netz, repräsentiert durch seine Kopplungsmatrix C, so dass für alle Netzausgaben $\mathbf{y}^\mu$ gilt: $\mathbf{T}^\mu \approx \mathbf{y}^\mu$.

Die spezielle Architektur, Anzahl der Neuronen, Anzahl der Schichten, etc soll an dieser Stelle nicht festgelegt werden.

Zielfunktionen

- $\mathbf{T}^\mu \approx \mathbf{y}^\mu$ präzisiere fassen
- Unterschied von $\mathbf{T}^\mu$ und $\mathbf{y}^\mu$ durch Abstandsfunktionen messen

$$\|\mathbf{T}^\mu - \mathbf{y}^\mu\|_p := \Big(\sum_{i=1}^{n} |T_i^\mu - y_i^\mu|^p\Big)^{\frac{1}{p}}$$

hierbei ist $p \geq 1$, also etwa Abstandsfunktionen

$p = 1$ (Manhattan Abstand)

$$\|\mathbf{T}^\mu - \mathbf{y}^\mu\|_1 := \sum_{i=1}^{n} |T_i^\mu - y_i^\mu|$$

oder $p = 2$ (Euklidischer Abstand)

$$\|\mathbf{T}^\mu - \mathbf{y}^\mu\|_2 := \Big(\sum_{i=1}^{n} |T_i^\mu - y_i^\mu|^2\Big)^{\frac{1}{2}}$$

- Oft wird der **quadrierter euklidischer Abstand** benutzt

$$\|\mathbf{T}^\mu - \mathbf{y}^\mu\|_2^2 := \sum_{i=1}^{n} (T_i^\mu - y_i^\mu)^2$$

- Netzausgabe $\mathbf{y}$ ist anhängig von Kopplungsmatrix $\mathbf{C}$, also $\mathbf{y} = \mathbf{y}(\mathbf{C}) = \mathbf{y}_\mathbf{C}$.
- Um die Netzausgaben den Sollausgaben anzupassen, soll nun die Kopplungsmatrix $\mathbf{C}$ adaptiert werden.
- Gesucht ist die Kopplungsmatrix $\mathbf{C}^*$, die die Differenz zwischen den Netzausgaben und den Lehrersignalen minimiert.
- Dieser Unterschied wird durch eine **Zielfunktion** oder **Fehlerfunktion** $E : \mathbb{R}^r \to \mathbb{R}$ gemessen (r die Anzahl adaptierbaren Parameter $\mathbf{C}$), z.B.

$$E(\mathbf{C}) = \sum_{\mu=1}^{M} \sum_{j=1}^{n} (T_j^\mu - y_j^\mu)^2$$

Optimierung von Zielfunktionen

Sei nun eine reellwertige Fehlerfunktion (Zielfunktion) $E(\mathbf{C})$, $E : \mathbb{R}^r \rightarrow \mathbb{R}$ gegeben.

Gesucht ist nun die Kopplungsmatrix $\mathbf{C}^* \in \mathbb{R}^r$, die E minimiert, d.h.

$E(\mathbf{C}^*) \leq E(\mathbf{C})$ für alle $\mathbf{C} \in \mathbb{R}^r$

$\mathbf{C}^*$ heißt dann das **globale Minimum** von E.

In den meisten Fällen kann $\mathbf{C}^*$ nicht analytisch berechnet werden, man muss sich mit numerischen, zumeist suboptimalen lokalen Lösungen, zufrieden geben.

Das einfachst numerische Verfahren zur Berechnung **lokaler Minima** ist das *Gradientenverfahren*.
Voraussetzung hierfür: E muss differenzierbar sein.

Zunächst betrachten wir $r = 1$. Dann führt die folgende Strategie in ein lokales Minimum der Funktion $E : \mathbb{R} \to \mathbb{R}$:

1. Setze t=0 und wähle einen Startwert $\mathbf{C}(t)$.

2. Falls $E'(\mathbf{C}(t)) > 0$, dann setze $\mathbf{C}(t+1)$ auf einen Wert $< \mathbf{C}(t)$

3. Falls $E'(\mathbf{C}(t)) < 0$, dann setze $\mathbf{C}(t+1)$ auf einen Wert $> \mathbf{C}(t)$

Algorithmisch etwas präziser:

1. Setze $t = 0$. Wähle Schrittweite $l > 0$, Startwert $\mathbf{C}(0)$ und $\epsilon > 0$:

2. Iterierte die Vorschrift für $t = 0, 1, \ldots$ bis $|\Delta\mathbf{C}| < \epsilon$:

$$\mathbf{C}(t+1) = \mathbf{C}(t) - l \cdot E'(\mathbf{C}(t))$$

Für mehrdimensionale Funktionen $E : \mathbb{R}^r \to \mathbb{R}$ wird der Gradient grad$E(\mathbf{C})$ statt der Ableitung $E'(\mathbf{C})$ benutzt also

1. Setze $t = 0$. Wähle Schrittweite/Lernrate $l > 0$, Startwert $\mathbf{C}(0)$ und $\epsilon > 0$:

2. Iterierte die Vorschrift für $t = 0, 1, \ldots$ bis $\|\Delta\mathbf{C}\| < \epsilon$:

$$\mathbf{C}(t+1) = \mathbf{C}(t) - l \cdot \text{grad}E(\mathbf{C}(t))$$

Zur Erinnerung für $f : \mathbb{R}^r \to \mathbb{R}$ wird der Gradient grad$f(\mathbf{x})$ definiert durch

$$\text{grad}f(\mathbf{x}) = (\frac{\partial}{\partial x_1}f(\mathbf{x}), \ldots, \frac{\partial}{\partial x_r}f(\mathbf{x})) \in \mathbb{R}^r$$

Probleme bei **Gradienten-Verfahren**: Konvergenz gegen lokales Minimum ist langsam; selbst Konvergenz nicht gesichert (etwa bei zu großen Schrittweiten l.

$l > 0$ ist ein Parameter, der die Größe der Korrektur/Anpassung (von $\mathbf{C}$) bestimmt.

Lernregel für einschichtiges Netz

Sei also eine Trainingsmenge gegeben durch

$$\mathcal{T} = \{(\mathbf{x}^\mu, \mathbf{T}^\mu) \mid \mathbf{x}^\mu \in \mathbb{R}^d, \mathbf{T}^\mu \in \mathbb{R}^n;\ \mu = 1, \ldots, M\}$$

Das neuronale Netz bestehe aus einer einzelnen Schicht (Eingabeschicht nicht mitgezählt) bestehend aus n nichtlinearen Neuronen.

Berechnungen der Neuronen $j = 1, \ldots, n$ bei Eingabe von $\mathbf{x}^\mu \in \mathbb{R}^d$:

1. $u_j^\mu = \langle \mathbf{x}, \mathbf{c}_j \rangle = \sum_{i=1}^d x_i^\mu c_{ij}$

2. Netzausgaben: $y_j^\mu = f(u_j^\mu) = f\left(\sum_{i=1}^d x_i^\mu c_{ij}\right)$

Definition der Zielfunktion $E(\mathbf{C}) := E(\mathbf{c}_1, \ldots, \mathbf{c}_n)$, $\mathbf{c}_i$ der $i-te$ Spaltenvektor

von $\mathbf{C}$:

$$E(\mathbf{C}) = \sum_{\mu=1}^{M} \|\mathbf{T}^{\mu} - \mathbf{y}^{\mu}\|^2 = \sum_{\mu=1}^{M} \sum_{j=1}^{n} (T_j^{\mu} - y_j^{\mu})^2 \to \min$$

Lernregel als Gradienten-Verfahren:

1. grad $E(\mathbf{C}) = E(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ ausrechnen (Kettenregel zweimal anwenden) (hier für ein einzelnes Gewicht c_{ij}) :

$$\frac{\partial}{\partial c_{ij}} E(\mathbf{C}) = \sum_{\mu=1}^{M} 2 \cdot (T_j^{\mu} - y_j^{\mu}) \cdot (-f'(u_j^{\mu})) \cdot x_i^{\mu}.$$

2. Ableitung in die allgemeine Formel einsetzen (hier für ein einzelnes Gewicht c_{ij}):

$$c_{ij}(t+1) := c_{ij}(t+1) + 2l(t) \sum_{\mu=1}^{M} (T_j^{\mu} - y_j^{\mu}) f'(u_j^{\mu}) x_i^{\mu}$$

Dies ist eine **Batch-Modus-Lernregel**, da der gesamte Trainingsdatensatz $\mathcal{T}$ benutzt wird um einen Änderung der Matrix $\mathbf{C}$ durchzuführen.

Die zugehörige **inkrementelle Lernregel** hat die Form

$$c_{ij}(t+1) := c_{ij}(t+1) + 2l(t)(T_j^\mu - y_j^\mu)f'(u_j^\mu)x_i^\mu$$

Hier werden die Gewichtsanpassungen der c_{ij} durch Einzelmuster herbeigeführt.

In beiden Fällen muss der gesamte Trainingsdatensatz mehrfach benutzt werden, bis die berechnete Gewichtsanpassung unter einer a priori definierten Schranke ϵ bleibt.

Inkrementelle Lernregeln erfordern nur die Hälfte des Speicherplatzes als die zugehörigen Batch-Modus-Lernregeln (die Einzeländerungen müssen bei Batch-Modus-Lernregeln zunächst aufsummiert werden bevor der eigentliche Anpassungsschritt durchgeführt werden).

6. Überwachtes Lernen in KNN

1. Einleitung
2. Perzeptron
3. Mehrschichtnetze
4. Netze zum Lernen von Zeitreihen
5. Radiale Basisfunktionsnetze
6. Konstruktive Netze

6.1 Einleitung

- Ein KNN soll eine beliebige Abbildung $T : X \rightarrow Y$ lernen, d.h. zu *jedem* Muster $\mathbf{x}^\mu \in X$ soll die zugehörige Netzausgabe $\mathbf{y}^\mu \in Y$ möglichst gleich $\mathbf{T}^\mu := T(\mathbf{x}^\mu)$ sein

- Eingabevektoren: $\mathbf{x} \in X$ mit
  - $X \subset \mathbb{R}^m$
  - $X \subset \mathbb{Z}^m$
  - $X \subset \{0, 1\}^m$

- Ausgabevektoren: $\mathbf{y} \in Y$ mit
  - $Y \subset \{0, 1\}^n$ oder $Y \subset \mathbb{Z}^n$ (*Klassifikation*)
  - $Y \subset \mathbb{R}^n$ (*Approximation* oder *Regression*)

- resultierendes Netz hat m Eingabeknoten und n Ausgabeneuronen

- **Ziel:** Parameter des KNN sind derart einzustellen, dass Fehler

$$R[T] = \int_X \| T(\mathbf{x}) - \mathbf{y} \| \, dP(\mathbf{x}, \mathbf{y})$$

minimiert wird (*erwartete Fehler*).

Probleme: Wahrscheinlichkeitsverteilung $P(\mathbf{x}, \mathbf{y})$ auf $X \times Y$ ist unbekannt.

- **Ansatz: Minimierung des empirischen Fehlers** Auswahl einer *Trainingsmenge/Stichprobe* mit M Vektorpaaren $(\mathbf{x}^1, \mathbf{T}^1), \ldots, (\mathbf{x}^M, \mathbf{T}^M)$ und Einstellung der Netzparameter, so dass der Trainingsfehler

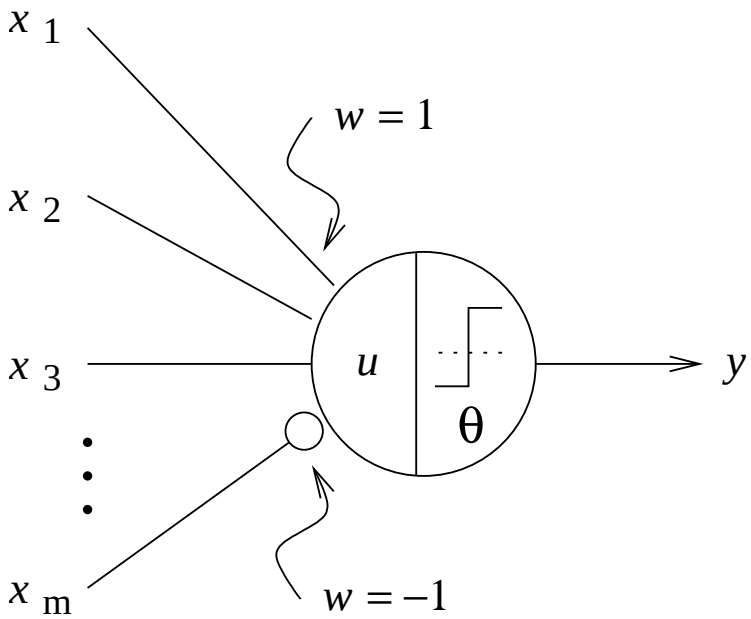
$$R_{\text{emp}}[T] = \frac{1}{M} \sum_{\mu=1}^{M} \| \mathbf{T}^\mu - \mathbf{y}^\mu \|$$

minimiert wird (*empirischer Fehler*).

Problem: Fähigkeit des Netzes zur *Generalisierung*

Exkurs: McCulloch-Pitts Neuron

- Struktur eines Neurons (McCulloch and Pitts, 1943):



- Funktionalität:

$$u = \sum_{i=1}^{m} x_i w_i = \mathbf{x} \cdot \mathbf{w} = \langle \mathbf{x}, \mathbf{w} \rangle$$

$$y = \begin{cases} 1 & \text{für} \quad u \geq \theta \\ 0 & \text{für} \quad \text{sonst} \end{cases}$$

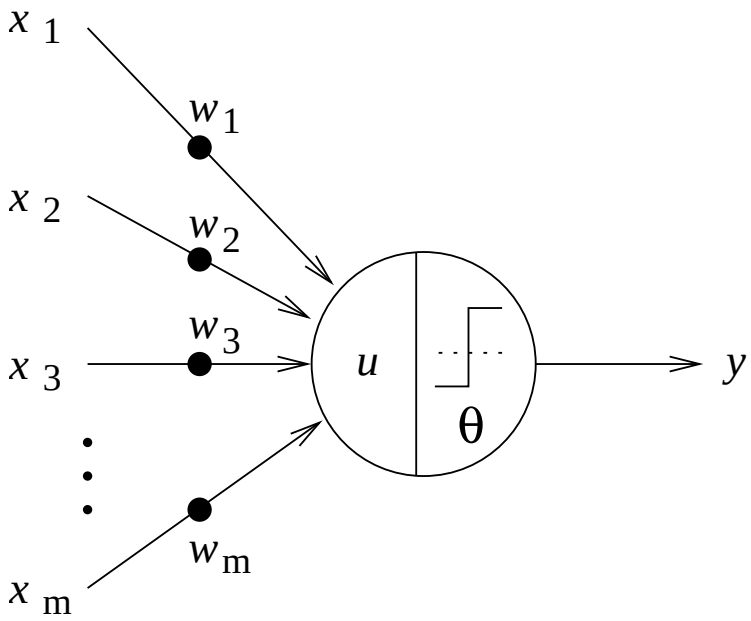
mit $\mathbf{x} \in \{0,1\}^m$, $\mathbf{w} \in \{-1,1\}^m$, $\theta \in \mathbb{Z}$

- **Satz:** Jede beliebige logische Funktion ist mit Netzen aus McCulloch-Pitts Neuronen realisierbar.

- Lernen ist nicht möglich !

6.2 Perzeptron

- Struktur eines Perzeptrons (Rosenblatt 1958):



- Funktionalität:

$$u = \sum_{i=1}^{m} x_i w_i = \mathbf{x} \cdot \mathbf{w} = \langle \mathbf{x}, \mathbf{w} \rangle$$

$$y = \begin{cases} 1 & \text{für} \quad u \geq \theta \\ 0 & \text{für} \quad \text{sonst} \end{cases}$$

mit $\mathbf{x}, \mathbf{w} \in \mathbb{R}^m$ und $\theta \in \mathbb{R}$

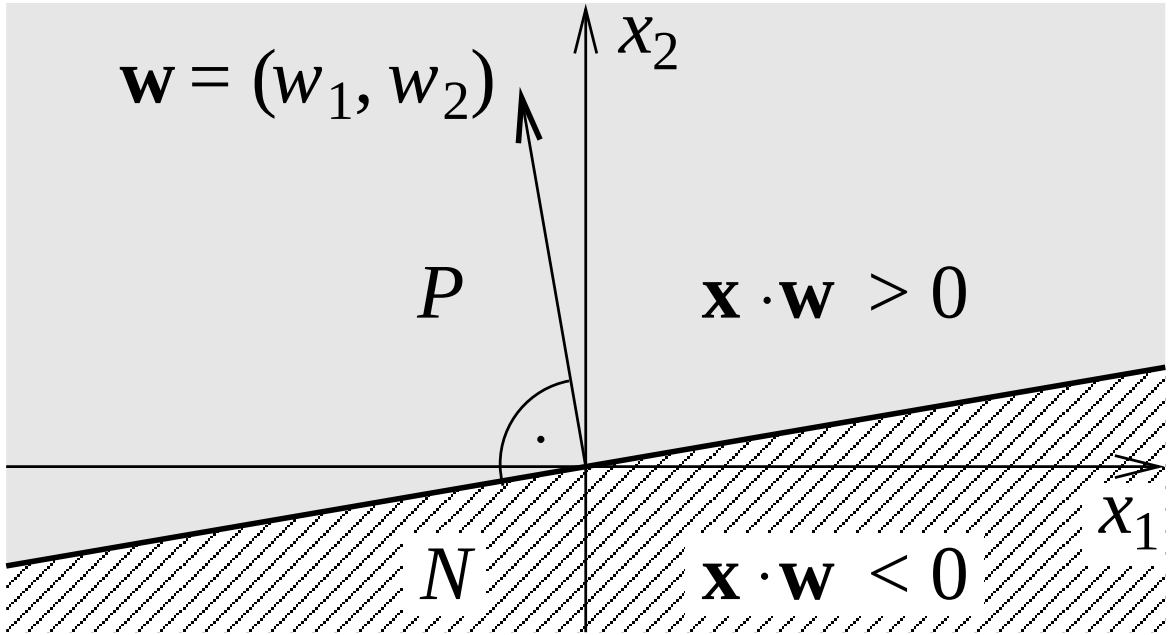
- äquivalente Formulierung mit Bias $b := -\theta$:

$$u = \sum_{i=1}^{m} x_i w_i + b = \mathbf{x} \cdot \mathbf{w} + b \;, \quad y = \begin{cases} 1 & \text{für} \quad u \geq 0 \\ 0 & \text{für} \quad \text{sonst} \end{cases}$$

- gelegentlich wird der Bias $b = -\theta$ auch als Gewicht w_0 eines weiteren (konstanten) Eingangs $x_0 = 1$ angesehen:

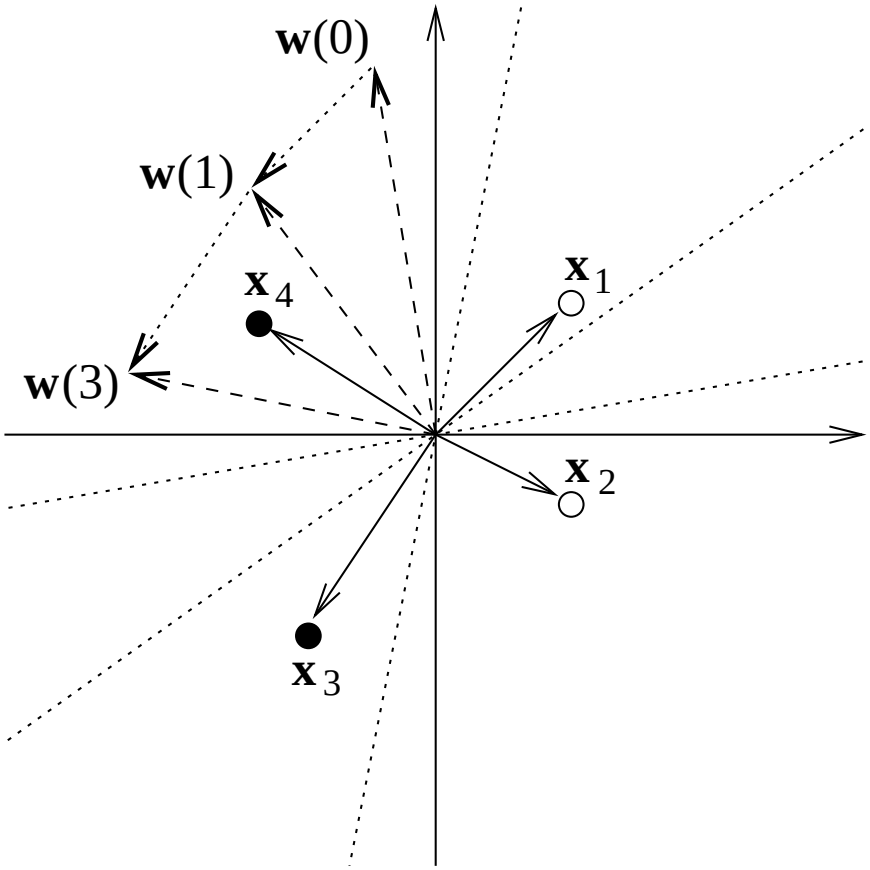
$$u = \sum_{i=0}^{m} x_i w_i = \mathbf{x} \cdot \mathbf{w} \ , \quad y = \begin{cases} 1 & \text{für} \quad u \geq 0 \\ 0 & \text{für} \quad \text{sonst} \end{cases}$$

- Für $m = 2$ beschreibt die Gleichung $\mathbf{x} \cdot \mathbf{w} = 0$ eine Gerade, die die Menge aller Punkte $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ in zwei Halbräume P und N teilt:



Idee des Perzeptron Lernalgorithmus

- Lernproblem:
  - $\mathcal{X}_1$ enthält alle $\mathbf{x}$ mit $f(\mathbf{x}) = 1$
    $\mathcal{X}_0$ enthält alle $\mathbf{x}$ mit $f(\mathbf{x}) = 0$
  - Existiert Gewichtsvektor $\mathbf{w}$ mit
    $\mathbf{x} \cdot \mathbf{w} \geq 0$ für alle $\mathbf{x} \in \mathcal{X}_1$ und
    $\mathbf{x} \cdot \mathbf{w} < 0$ für alle $\mathbf{x} \in \mathcal{X}_0$?
- Lernschritt:
  - falls $(\mathbf{x} \cdot \mathbf{w} \geq 0$ und $\mathbf{x} \in \mathcal{X}_1)$
    oder $(\mathbf{x} \cdot \mathbf{w} < 0$ und $\mathbf{x} \in \mathcal{X}_0)$:
    $\mathbf{w}(t+1) = \mathbf{w}(t)$
  - falls $\mathbf{x} \cdot \mathbf{w} < 0$ und $\mathbf{x} \in \mathcal{X}_1$:
    $\mathbf{w}(t+1) = \mathbf{w}(t) + \mathbf{x}$
  - falls $\mathbf{x} \cdot \mathbf{w} \geq 0$ und $\mathbf{x} \in \mathcal{X}_0$:
    $\mathbf{w}(t+1) = \mathbf{w}(t) - \mathbf{x}$
- Veranschaulichung für $x_1, x_2 \in \mathcal{X}_0$ und $x_3, x_4 \in \mathcal{X}_1$ bei $\theta = 0$:



- einfachere Formulierung der Perzeptron-Lernregel:

  Sei $T \in \{0, 1\}$ der Wert des Lehrersignals (Sollausgabe) für das Eingabemuster $\mathbf{x}$, dann kann Fehler δ am Ausgang des Perzeptrons angegeben werden: $\delta = (T - y)$
  Hiermit lassen sich alle 3 Fälle des Lernschritts einheitlich beschreiben:
  $\mathbf{w}(t + 1) = \mathbf{w}(t) + \delta\mathbf{x} = \mathbf{w}(t) + (T - y)\mathbf{x}$

- oft wird eine zusätzliche Lernrate $\eta > 0$ verwendet:
  $\mathbf{w}(t + 1) = \mathbf{w}(t) + \eta\delta\mathbf{x} = \mathbf{w}(t) + \eta(T - y)\mathbf{x}$

Lernalgorithmus für Perzeptron:

Gegeben:
1 Neuron mit Schwellwert $\theta \in \mathbb{R}$, m Gewichte $w_i \in \mathbb{R}$, $i = 1, \ldots, m$
Menge $\mathcal{T}$ mit von M Mustern $\mathbf{x}^\mu \in \mathbb{R}^m$ mit Lehrersignalen $T^\mu \in \{0, 1\}$

Schritt 1: Initialisierung

Setze $w_i(0), i = 1, \ldots, m$ und $\theta(0)$ beliebig

Schritt 2: Aktivierung

Wähle Muster $\mathbf{x}^\mu \in \mathcal{X}$ mit $T^\mu \in \{0, 1\}$ und berechne $u = \mathbf{x}^\mu \cdot \mathbf{w} - \theta$

Schritt 3: Ausgabe

Setze $y = 1$ für $u \geq 0$ bzw. $y = 0$ für $u < 0$
Berechne Differenz $\delta = T^\mu - y$

Schritt 4: Lernen

Adaptiere Gewichte: $w_i(t+1) = w_i(t) + \eta\,\delta x_i^\mu$
Adaptiere Schwellwert: $\theta(t+1) = \theta(t) - \eta\,\delta$
(bei Standard Perzeptron-Lernregel: $\eta = 1$)

Schritt 5: Ende

Solange es in $\mathcal{T}$ noch Muster mit $\delta \neq 0$ gibt, gehe zurück zu Schritt 2

Satz zur Konvergenz des Perzeptron Lernalgorithmus:
(Minsky und Papert, 1969)

Ist das Klassifikationsproblem *linear separierbar*, dann findet der Perzeptron Lernalgorithmus nach *endlicher* Anzahl von Lernschritten eine Lösung.

Konvergenzbeweis des Perzeptron Lernalgorithmus:

Annahmen und Vereinfachungen:

1. Es gibt einen Lösungsvektor $\mathbf{w}^*$ mit $||\mathbf{w}^*|| = 1$,
   der $\mathcal{X}_0$ und $\mathcal{X}_1$ absolut linear separiert

2. Für alle $\mathbf{x} \in \mathcal{X}_0$ setze $\mathbf{x} = -\mathbf{x}$
   $\Rightarrow$ Für alle $\mathbf{x} \in \mathcal{X} = \mathcal{X}_0 \cup \mathcal{X}_1$ gilt: $\mathbf{w}^* \cdot \mathbf{x} > 0$

3. Alle Vektoren $\mathbf{x} \in \mathcal{X}$ sind normiert: $||\mathbf{x}|| = 1$

4. Gewichtsvektor $\mathbf{w}(t)$ hat $\mathbf{x} \in \mathcal{X}$ falsch klassifiziert

Betrachte $\cos\tau = \dfrac{\mathbf{w}^* \cdot \mathbf{w}(t+1)}{||\mathbf{w}^*||\ ||\mathbf{w}(t+1)||} = \dfrac{\mathbf{w}^* \cdot \mathbf{w}(t+1)}{||\mathbf{w}(t+1)||}$

Zähler: $\mathbf{w}^* \cdot \mathbf{w}(t+1) = \mathbf{w}^* \cdot (\mathbf{w}(t) + \mathbf{x})$

$= \mathbf{w}^* \cdot \mathbf{w}(t) + \mathbf{w}^* \cdot \mathbf{x}$

$\geq \mathbf{w}^* \cdot \mathbf{w}(t) + \delta$

$\geq \mathbf{w}^* \cdot \mathbf{w}(0) + (t+1)\delta$

Nenner: $||\mathbf{w}(t+1)||^2 = (\mathbf{w}(t) + \mathbf{x}) \cdot (\mathbf{w}(t) + \mathbf{x})$

$= ||\mathbf{w}(t)||^2 + 2\mathbf{w}(t)\mathbf{x} + ||\mathbf{x}||^2$

$\leq ||\mathbf{w}(t)||^2 + ||\mathbf{x}||^2$

$\leq ||\mathbf{w}(t)||^2 + 1$

$\leq ||\mathbf{w}(0)||^2 + (t+1)$

Einsetzen liefert: $\cos\tau \geq \frac{\mathbf{w}^* \cdot \mathbf{w}(0) + (t+1)\delta}{\sqrt{||\mathbf{w}(0)||^2 + (t+1)}}$

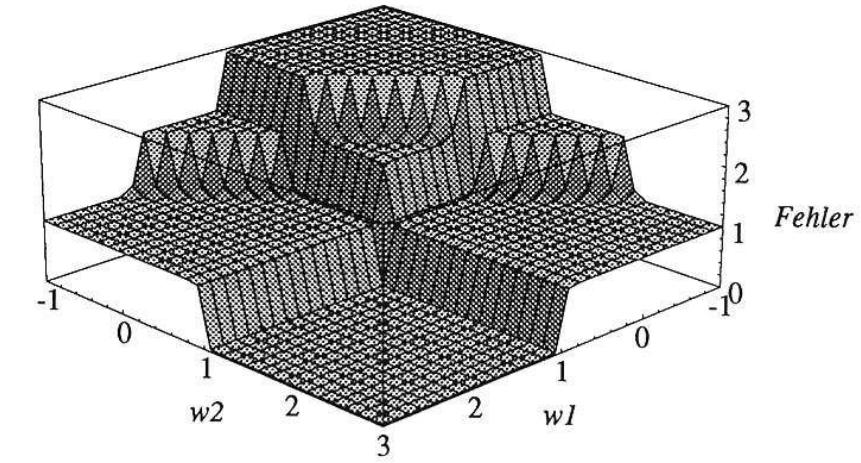
$\Rightarrow t$ ist endlich, da stets gilt: $\cos\tau \leq 1$

Fehlerflächen

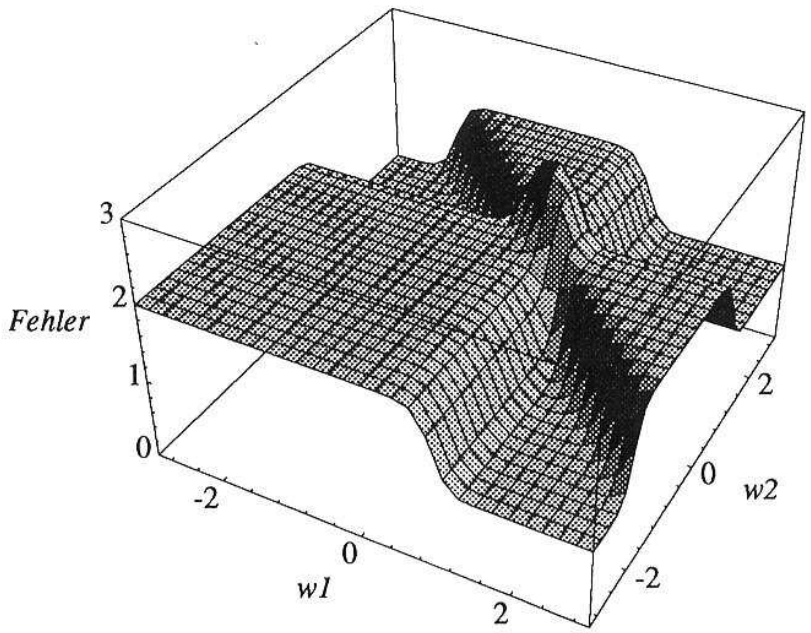
- Eine Visualisierung des Fehlers am Ausgang eines Perzeptrons in Abhängigkeit von den Gewichten bezeichnet man als *Fehlerfläche*.

- Beispiele (für $m = 2$ Eingänge, $\theta = 1$):

1) Fehlerfläche der Funktion OR



2) Fehlerfläche der Funktion XOR



Bemerkungen zum Perzeptron

- Liegt statt eines Klassifikationsproblems mit *zwei* Klassen ein Klassifikationsproblem mit n Klassen vor, so werden i.a. n Perzeptronen verwendet.

  Hierbei lernt Perzeptron i, die Muster der Klasse i von den Mustern aller anderen Klassen $j \neq i$ zu trennen.

- Anteil linear separierbarer Probleme sinkt bei Erhöhung der Eingabedimensionalität m

  Beispiel: linear separierbare Boole'sche Funktionen mit m Variablen

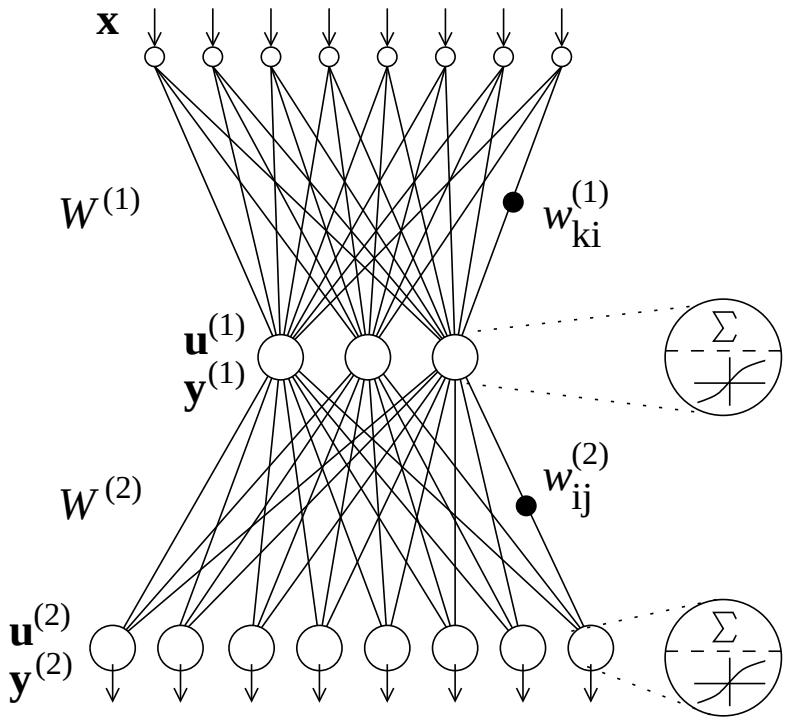
m	Anzahl linear separierbarer Fkt.	Anteil
2	14 (von 16)	87.5%
3	104 (von 256)	40.6%
4	1772 (von 65536)	2.7%

  $\Rightarrow$ *Lösung:* mehrschichtige Netze aus Perzeptronen ...

6.3 Mehrschichtiges Perzeptron (MLP)

- Idee: Lernen mittels Gradientenabstieg
  (erfordert jedoch eine differenzierbare Transferfunktion $\Rightarrow$ Verwendung der sigmoiden Funktion $f(x) = 1/(1 + \exp(-\beta x))$ anstatt der Schwellwertfunktion !)

- Aufbau eines zweischichtigen MLP mit sigmoiden Neuronen:



- Berechnung der Netzausgabe $\mathbf{y}^{(2)}$:

$$u_i^{(1)} = \sum_{k=1}^{m} x_k w_{ki}^{(1)} - \theta_i^{(1)}$$

$$y_i^{(1)} = f(u_i^{(1)})$$

$$u_j^{(2)} = \sum_{i=1}^{h} y_i^{(1)} w_{ij}^{(2)} - \theta_j^{(2)}$$

$$y_j^{(2)} = f(u_j^{(2)})$$

Lernen im mehrschichtigen Perzeptron

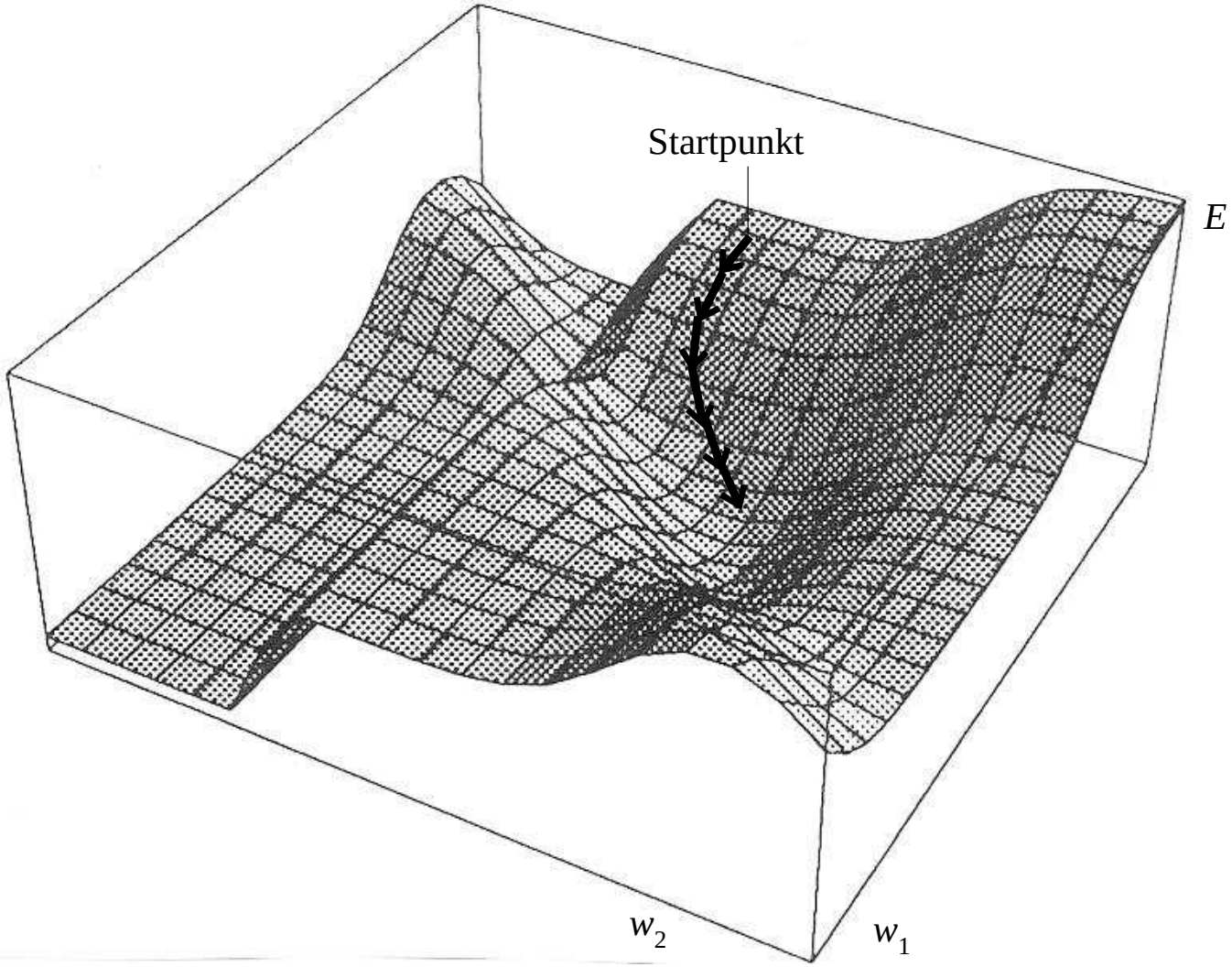
Quadratischer Fehler am Netzausgang:

für Muster μ: $E_\mu = \|\mathbf{T}_\mu - \mathbf{y}_\mu\|^2 = \sum_{j=1}^{m} (T_{\mu j} - y_{\mu j})^2$

für alle Muster: $E = \sum_{\mu=1}^{M} E_\mu = \sum_{\mu=1}^{p} \|\mathbf{T}_\mu - \mathbf{y}_\mu\|^2 = \sum_{\mu=1}^{M} \sum_{j=1}^{m} (T_{\mu j} - y_{\mu j})^2$

mit:
m : Anzahl Neuronen der Ausgabeschicht
M : Anzahl Muster in der Trainingsmenge
$T_{\mu j}$: Sollwert für Neuron j bei Muster μ
$y_{\mu j}$: Ausgabe von Neuron j bei Muster μ

Beispiel einer Fehlerfläche und eines Gradientenabstiegs:



$\Rightarrow$ häufig Abstieg in lokale Minima !

Herleitung der Error Backpropagation Lernregel

Für Gewichte $w_{ij} = w_{ij}^{(2)}$ der Ausgabeschicht gilt:

$$
\begin{aligned}
\frac{\partial E_\mu}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \|\mathbf{T} - \mathbf{y}^{(2)}\|^2 \\
&= \frac{\partial}{\partial w_{ij}} \sum_{\tilde{j}} (T_{\tilde{j}} - y_{\tilde{j}}^{(2)})^2 \quad = \quad \frac{\partial}{\partial w_{ij}} (T_j - y_j^{(2)})^2 \\
&= -2(T_j - y_j^{(2)}) \; \frac{\partial}{\partial w_{ij}} f(\underbrace{\sum_{\tilde{i}} y_{\tilde{i}}^{(1)} \, w_{\tilde{i}j}}_{u_j^{(2)}}) \\
&= -2(T_j - y_j^{(2)}) \; f'(u_j^{(2)}) \; y_i^{(1)} \quad = \quad -2 \, y_i^{(1)} \, \delta_j^{(2)}
\end{aligned}
$$

wobei $\delta_j^{(2)} = (T_j - y_j^{(2)}) \; f'(u_j^{(2)})$ den Fehler von Neuron j der Ausgabeschicht für ein Muster bezeichnet

Resultierende Lernregeln für Gewichte $w_{ij} = w_{ij}^{(2)}$ der Ausgabeschicht:

im *Online-Modus:*

(d.h. Anpassung der Gewichte nach der Präsentation eines *einzelnen* Musters μ)

$$w_{ij} = w_{ij} - \eta \frac{\partial E_\mu}{\partial w_{ij}} = w_{ij} + \eta\, y_i^{(1)} \delta_j^{(2)}$$

im *Batch-Modus:*

(d.h. Anpassung der Gewichte erst nach der Präsentation *aller* Muster μ)

$$w_{ij} = w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} = w_{ij} + \eta \sum_\mu y_{\mu i}^{(1)}\ \delta_{\mu j}^{(2)}$$

Für Gewichte $w_{ki} = w_{ki}^{(1)}$ der versteckten Schicht gilt:

$$\begin{aligned}
\frac{\partial E_\mu}{\partial w_{ki}} &= \frac{\partial E_\mu}{\partial y_i^{(1)}} \frac{\partial y_i^{(1)}}{\partial w_{ki}} = \frac{\partial}{\partial y_i^{(1)}} \sum_j (T_j - y_j^{(2)})^2 \frac{\partial y_i^{(1)}}{\partial w_{ki}} \\
&= -2 \sum_j (T_j - y_j^{(2)}) \frac{\partial}{\partial y_i^{(1)}} f(\underbrace{\sum_{\widetilde{i}} y_{\widetilde{i}}^{(1)} w_{\widetilde{i}j}^{(2)}}_{u_j^{(2)}}) \frac{\partial}{\partial w_{ki}} f(\underbrace{\sum_{\widetilde{k}} x_{\widetilde{k}} w_{\widetilde{k}i}}_{u_i^{(1)}}) \\
&= -2 \sum_j \underbrace{(T_j - y_j^{(2)}) f'(u_j^{(2)})}_{\delta_j^{(2)}} w_{ij}^{(2)} f'(u_i^{(1)}) x_k \\
&= -2 \sum_j \delta_j^{(2)} w_{ij}^{(2)} f'(u_i^{(1)}) x_k = -2 x_k \delta_i^{(1)}
\end{aligned}$$

wobei $\delta_i^{(1)} = \sum_j \delta_j^{(2)} w_{ij}^{(2)} f'(u_i^{(1)})$ den Fehler von Neuron i der versteckten Schicht für ein Muster bezeichnet

Resultierende Lernregeln für Gewichte $w_{ki} = w_{ki}^{(1)}$ der versteckten Schicht:

im *Online-Modus:*

(d.h. Anpassung der Gewichte nach der Präsentation eines *einzelnen* Musters μ)

$$w_{ki} = w_{ki} - \eta \frac{\partial E_\mu}{\partial w_{ki}} = w_{ki} + \eta \, x_k \, \delta_j^{(1)}$$

im *Batch-Modus:*

(d.h. Anpassung der Gewichte erst nach der Präsentation *aller* Muster μ)

$$w_{ki} = w_{ki} - \eta \frac{\partial E}{\partial w_{ki}} = w_{ki} + \eta \sum_\mu x_{\mu k} \; \delta_{\mu i}^{(1)}$$

Error Backpropagation Lernalgorithmus (Online) für MLP

Gegeben: Zweischichtiges m-h-n MLP
Menge $\mathcal{T}$ mit M Musterpaaren $(\mathbf{x}, \mathbf{T})$ mit Eingabemustern $\mathbf{x} = (x_1, \ldots, x_m) \in X$ und zugehörigen Lehresignalen $\mathbf{T} = (T_1, \ldots, T_n) \in Y$

Schritt 1: Initialisierung

Setze alle $w_{ki}^{(1)}$, $w_{ij}^{(2)}$, $\theta_i^{(1)}$ und $\theta_j^{(2)}$ für $k = 1, \ldots, m$, $i = 1, \ldots, h$, $j = 1, \ldots, n$ auf kleine zufällige Werte

Schritt 2: Berechnung der Netzausgabe $\mathbf{y}^{(2)}$ (Vorwärtsphase)

Wähle nächstes Muster $\mathbf{x} \in \mathcal{T}$ und berechne:

$$u_i^{(1)} = \sum_{k=1}^{m} x_k w_{ki}^{(1)} - \theta_i^{(1)} \quad \text{und} \quad y_i^{(1)} = f(u_i^{(1)}) \quad \text{für } i = 1, \ldots, h$$

$$u_j^{(2)} = \sum_{i=1}^{h} y_i^{(1)} w_{ij}^{(2)} - \theta_j^{(2)} \quad \text{und} \quad y_j^{(2)} = f(u_j^{(2)}) \quad \text{für } j = 1, \ldots, n$$

Schritt 3: Bestimmung des Fehlers am Netzausgang

Berechne $\delta_j^{(2)} = (T_j - y_j^{(2)})\, f'(u_j^{(2)})$ für $j = 1, \ldots, n$

Schritt 4: Fehlerrückvermittlung (Rückwärtsphase)

Berechne $\delta_i^{(1)} = \sum_j w_{ij}^{(2)}\, \delta_j^{(2)}\, f'(u_i^{(1)})$ für $i = 1, \ldots, h$

Schritt 5: Lernen

Adaptiere Gewichte: $w_{ij}^{(2)} = w_{ij}^{(2)} + \eta\, y_i^{(1)} \delta_j^{(2)}$

$$w_{ki}^{(1)} = w_{ki}^{(1)} + \eta\, x_k\, \delta_i^{(1)}$$

Adaptiere Schwellwerte: $\theta_j^{(2)} = \theta_j^{(2)} - \eta \delta_j^{(2)}$

$$\theta_i^{(1)} = \theta_i^{(1)} - \eta \delta_i^{(1)}$$

für $k = 1, \ldots, m$, $i = 1, \ldots, h$, $j = 1, \ldots, n$

Schritt 6: Ende Epoche

Solange noch weitere Muster $\mathbf{x} \in \mathcal{T}$ vorhanden, gehe zurück zu Schritt 2

Schritt 7: Ende Training

Berechne Fehler $E = \sum_{\mu=1}^{M} \|\mathbf{T}_\mu - \mathbf{y}_\mu\|^2$

Solange $E > \epsilon$ und max. Anzahl Iterationen noch nicht überschritten, gehe zurück zu Schritt 2

Bemerkung: Error Backpropagation Lernalgorithmus kann auch für MLPs mit beliebig vielen Schichten verallgemeinert werden !

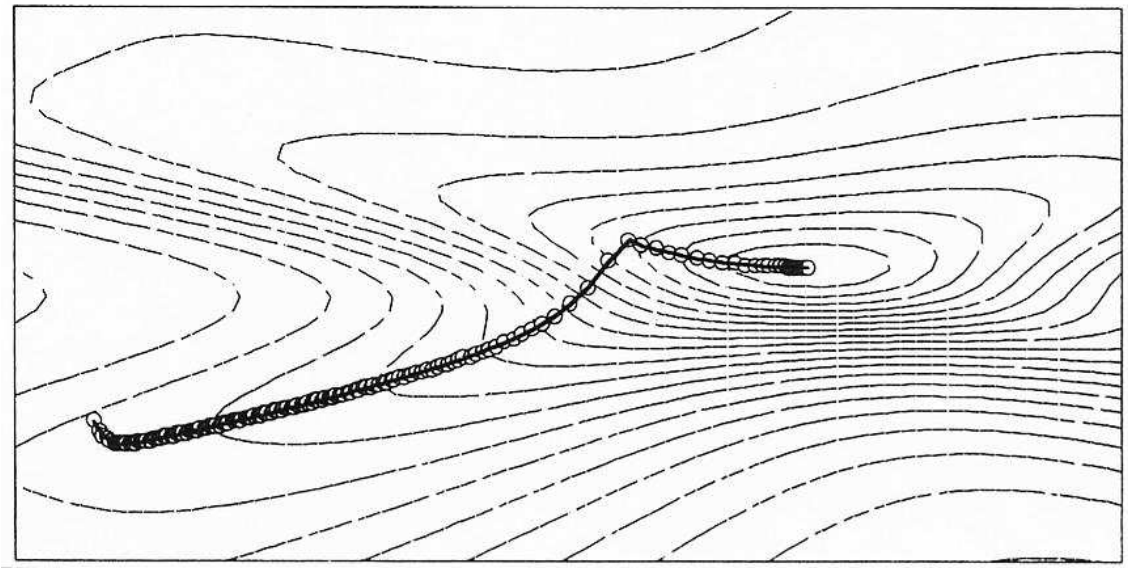
Bemerkungen zum MLP mit Error Backpropgation

- Schwellwerte θ_j wurden in Herleitung wegelassen; Lernregel resultiert aus Betrachtung von θ_j als ein mit dem konstanten Eingang $x_0 = -1$ verbundenes Gewicht $w_{0j} = \theta_j$.

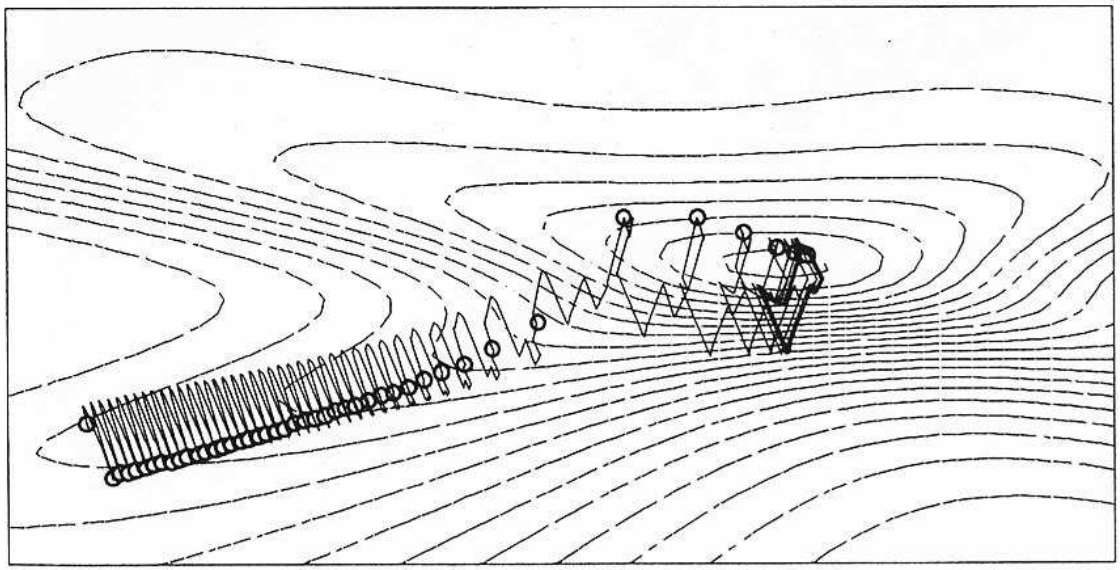
- einige sigmoide Funktionen und ihre Ableitungen:
  - logistische Funktion: $f(x) = 1/(1 + e^{-x})$
    mit Ableitung $f'(x) = e^{-x}/(1 + e^{-x})^2 = f(x)(1 - f(x))$
  - Fermi-Funktion: $f(x) = 1/(1 + e^{-\beta x})$
    mit Ableitung $f'(x) = \beta e^{-x}/(1 + e^{-\beta x})^2 = \beta f(x)(1 - f(x))$
  - Tangens hyperbolicus: $f(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$
    mit Ableitung $f'(x) = 1/\cosh^2(x) = 4/(e^x + e^{-x})^2$

- die *Ausgangsneuronen* eines MLP dürfen auch *lineare* Neuronen sein:
  $f(x) = x, \quad f'(x) = 1$
  $\Rightarrow$ sinnvoll bei Einsatz eines MLP zur Approximation !

mögliche Trajektorie
bei Batch-Lernen:

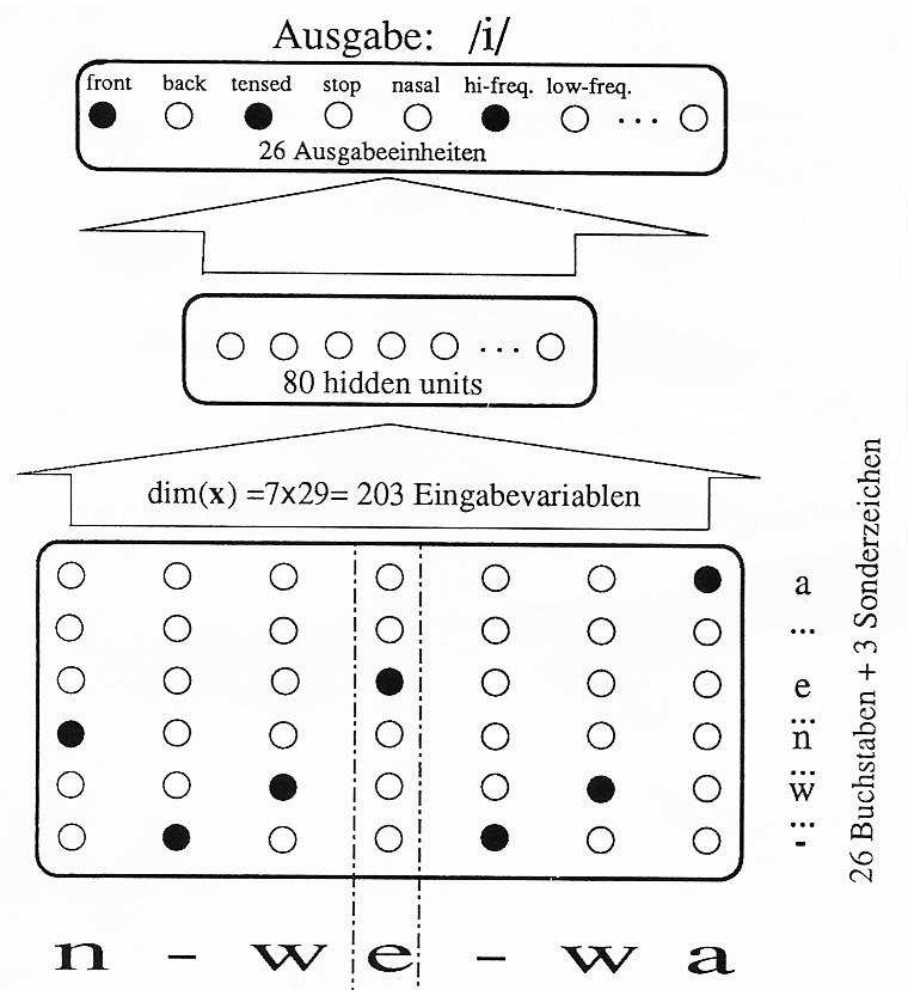


mögliche Trajektorie
bei Online-Lernen:



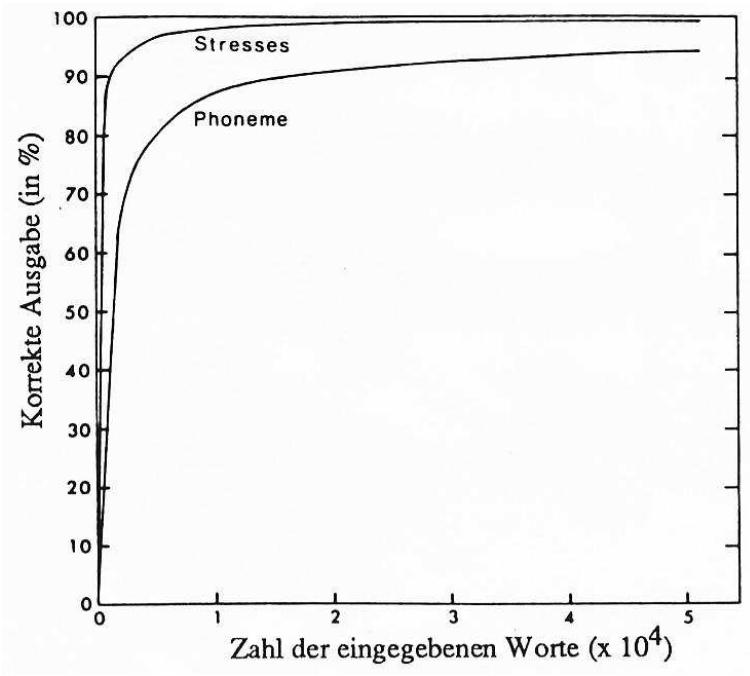
NETtalk als historische Anwendung von Backpropagation

- Architektur des MLP:



- Aufgabe: Lernens der Aussprache einfacher englischer Sätze
  [Sejnowski, Rosenberg 1986]

- Zahl korrekt ausgesprochener Worte:



Sätze zur Repräsentierbarkeit mit einem MLP

Zweischichtiges MLP mit sigmoiden Neuronen kann jede stetige Funktion $[0,1]^n \rightarrow [0,1]^m$ beliebig genau approximieren [Cybenko 89].

Korollar: Zweischichtiges MLP kann jede stetige Entscheidungsgrenze zwischen zwei Klassen beliebig genau approximieren.

Zweischichtiges MLP mit sigmoiden Neuronen in der verdeckten Schicht und linearen Neuronen in der Ausgangsschicht kann jede stetige Funktion mit $I^n \rightarrow \mathbb{R}^m$ auf einem kompakten Intervall $I^n \subset \mathbb{R}^n$ beliebig genau approximieren [Funahashi 89].

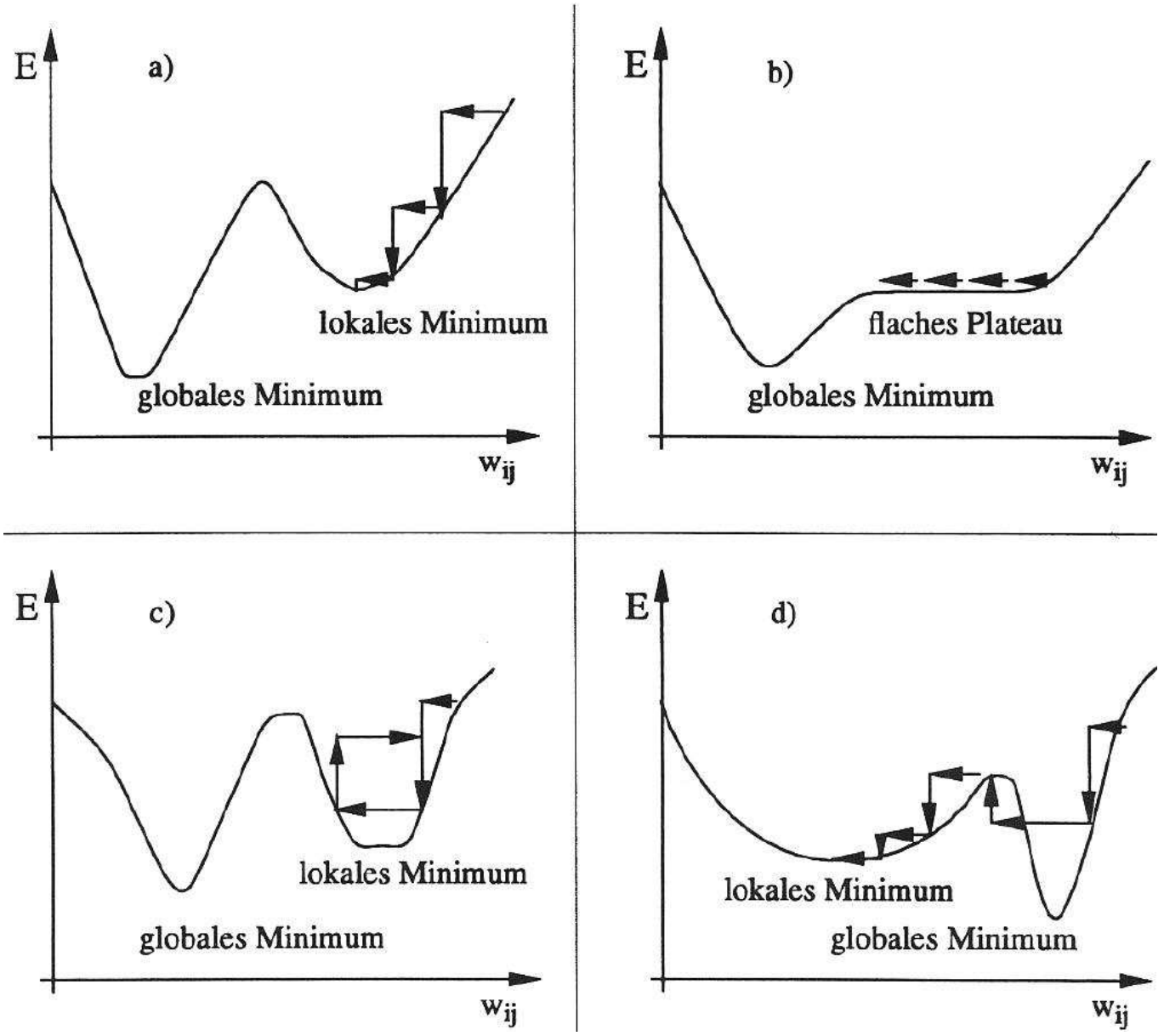
Zweischichtiges MLP mit linearen Ausgansneuronen kann jede stetige Funktion mit $I^n \rightarrow \mathbb{R}^m$ auf einem kompakten Intervall $I^n \subset \mathbb{R}^n$ beliebig genau approximieren, wenn die Transferfunktionen in der verdeckten Schicht beschränkt, stetig und nicht konstant sind [Hornik 91].

Probleme von MLP und Error Backpropagation

1. allgemeine Probleme eines Gradientenabstiegs:
   a) Erreichen lokaler Minima
   b) Flaches Plateau in der Fehlerfläche
   c) Oszillationen
   d) Überspringen guter Minima
2. kritische Wahl einer Lernrate η
3. kritische Wahl der Anzahl Neuronen h in der verdeckten Schicht
4. Sättigung
   (im Sättigungsbereich der sigmoiden Transferfunktion gilt für Neuron j: $f'(x_j) \approx 0$)
5. hohe Symmetrie
6. gute zufällige Initialisierung aller Gewichte w_{ij} nötig
   (typisch $w_{ij} \in [-\frac{2}{m}, \frac{2}{m}]$ bei einem MLP mit m Eingabeknoten)

Veranschaulichung von Problem 1:

(für eindimensionale Fehlerfläche;
aus: [Zell 1994])



Error Backpropagation mit Momentumterm

- *Idee:* Berücksichtigung von Gewichtsänderung aus Schritt $t-1$ in Schritt t

- Einführung eines *Momentumterms* (Rumelhart 1986):

  $$\Delta w_{ij}(t) \;=\; -\eta \frac{\partial E(t)}{\partial w_{ij}} + \alpha \cdot \Delta w_{ij}(t-1) \;=\; \eta\, y_i\, \delta_j + \alpha \Delta w_{ij}(t-1)$$

  mit *Momentumfaktor* $0 \leq \alpha < 1$

- zwei Fälle:

  a) $\text{sgn}(\Delta w_{ij}(t-1)) = \text{sgn}(-\frac{\partial E(t-1)}{\partial w_{ij}}) = \text{sgn}(-\frac{\partial E(t)}{\partial w_{ij}}) \;\Rightarrow$ Beschleunigung

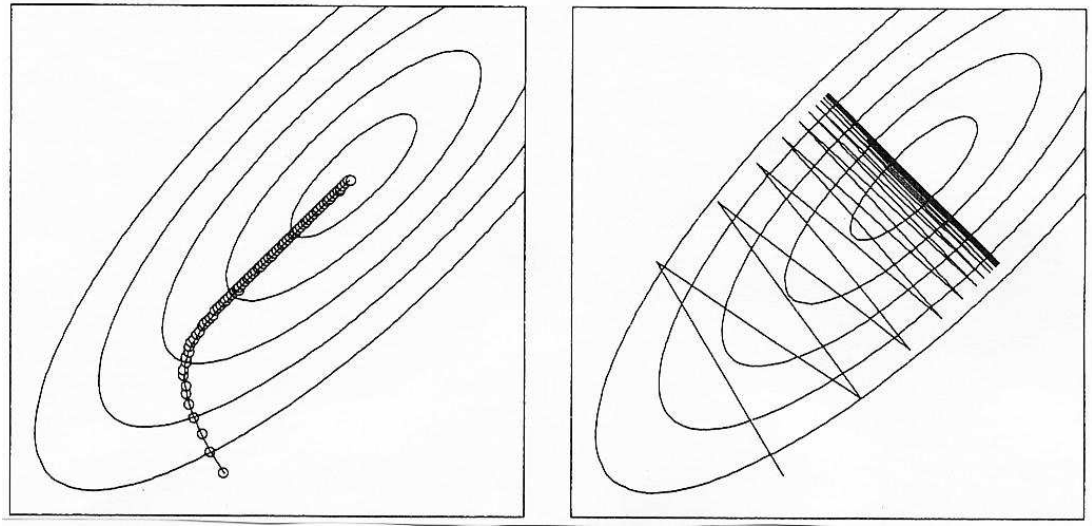
  b) $\text{sgn}(\Delta w_{ij}(t-1)) = \text{sgn}(-\frac{\partial E(t-1)}{\partial w_{ij}}) \neq \text{sgn}(-\frac{\partial E(t)}{\partial w_{ij}}) \;\Rightarrow$ Stabilisierung

- Einsetzen liefert:

$$\begin{aligned}
\Delta w_{ij}(t) &= -\eta \frac{\partial E(t)}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-1) \\
&= -\eta \frac{\partial E(t)}{\partial w_{ij}} + \alpha \left(-\eta \frac{\partial E(t-1)}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-2) \right) \\
&= -\eta \frac{\partial E(t)}{\partial w_{ij}} + \alpha \left(-\eta \frac{\partial E(t-1)}{\partial w_{ij}} - \eta \alpha \frac{\partial E(t-2)}{\partial w_{ij}} \right) \\
&= \ldots \\
&= -\eta \sum_{k=0}^{t} \alpha^k \, \frac{\partial E(t-k)}{\partial w_{ij}}
\end{aligned}$$

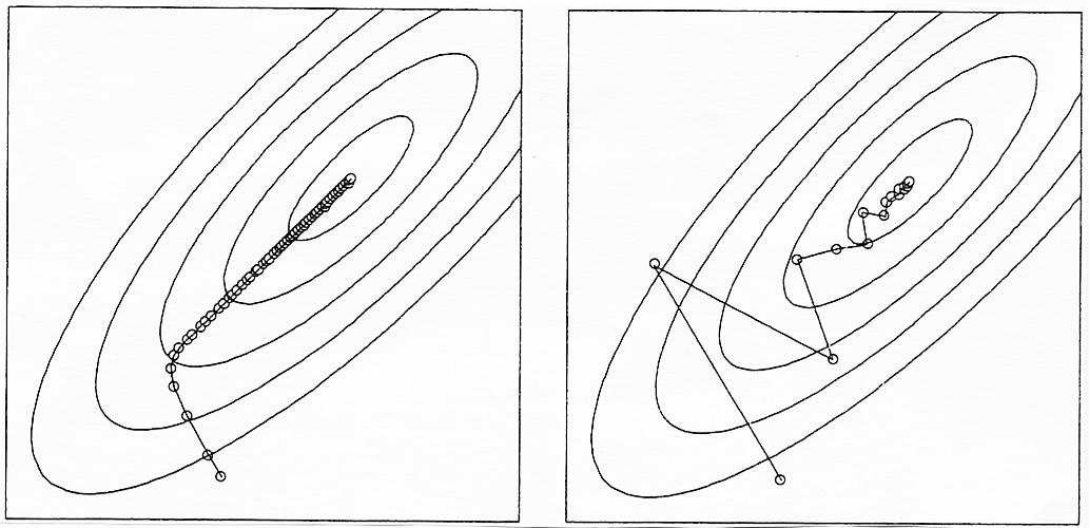
- effektive Lernrate für $t \to \infty$ im Fall a): $\eta_{\text{eff}} = \eta \sum_{k=0}^{\infty} \alpha^k = \frac{\eta}{(1-\alpha)}$

mögliche Trajektorie
ohne Momentumterm:



(a) kleine Lernrate (b) große Lernrate

mögliche Trajektorie
mit Momentumterm:



(a) kleine Lernrate (b) große Lernrate

Quickprop

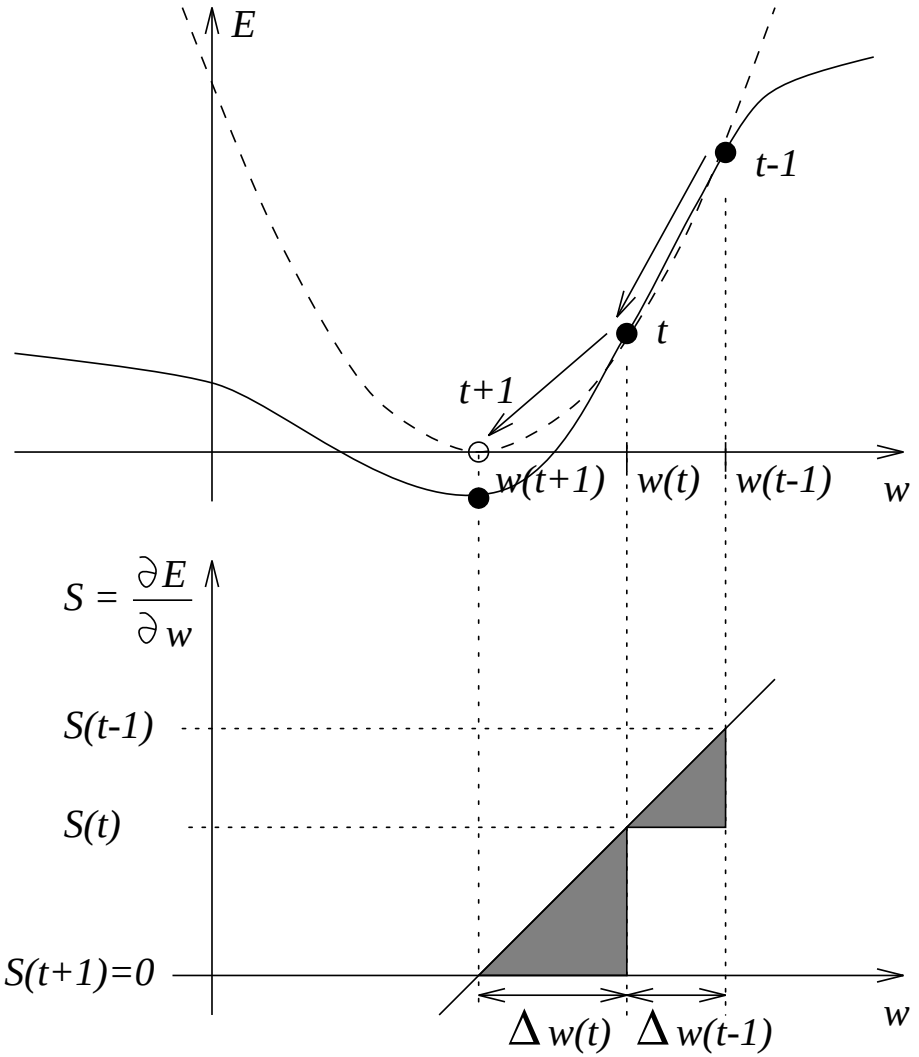
- *Idee:* Annäherung der Fehlerfunktion durch Parabel
- Lernregel (Fahlman 1989):

$$\Delta w_{ij}(t) = \beta_{ij}(t) \cdot \Delta w_{ij}(t-1)$$

mit $\beta_{ij}(t) = \dfrac{S_{ij}(t)}{S_{ij}(t-1) - S_{ij}(t)}$

und $S_{ij}(t) = \dfrac{\partial E}{\partial w_{ij}}(t)$

- viele Sonderfälle (s. [Zell94]):
  - für $t = 0$ oder $\Delta w_{ij}(t-1) = 0$ gilt Backpropagation Lernregel
  - für $\beta_{ij}(t) > \beta_{\max}$ gilt Lernregel $\Delta w_{ij}(t) = \beta_{\max} \cdot \Delta w_{ij}(t-1)$



Super Self Adapting Backprop (SuperSAB)

- *Idee:* gewichtsspezifische, adaptierbare Lernraten η_{ij}
- SuperSAB Lernregel (Tollenaere 1990):

$$w_{ij}(t+1) = w_{ij}(t) - \eta_{ij}(t)\frac{\partial E}{\partial w_{ij}}(t)$$

mit $\eta_{ij}(0) = \eta_{\text{start}}$

$$\eta_{ij}(t) = \begin{cases} \eta^+ \cdot \eta_{ij}(t-1) & \text{falls} \quad \frac{\partial E}{\partial w_{ij}}(t-1)\,\frac{\partial E}{\partial w_{ij}}(t) > 0 \\ \eta^- \cdot \eta_{ij}(t-1) & \text{falls} \quad \frac{\partial E}{\partial w_{ij}}(t-1)\,\frac{\partial E}{\partial w_{ij}}(t) < 0 \\ \eta_{ij}(t-1) & \text{sonst} \end{cases}$$

- Typische Wahl der Parameter: $\eta^+ = 1.1 \ldots 1.2,\ \eta^- = 0.5 \ldots 0.8$
- Verwendung nur im Batch-Lernmodus !

Resilient Backpropagation (RPROP)

- Problem aller bisher betrachteter Lernalgorithmen für MLPs: $\Delta w \sim -\frac{\partial E}{\partial w}$

  d.h. auf *flachen Plateaus* der Fehlerfläche ist Δw *sehr klein*;
  auf *steilen Flanken* der Fehlerfläche ist Δw *sehr groß*

- *Idee*: $\Delta w \sim -\text{sgn}\frac{\partial E}{\partial w}$

- RPROP Lernregel (Riedmiller, Braun 1993):

$$w_{ij}(t+1) = w_{ij}(t) - \eta_{ij}(t)\text{sgn}\frac{\partial E(t)}{\partial w_{ij}}$$

  (Vgl. dazu SuperSAB)

- Wahl von $\eta_{ij}(t)$: (wie beim SuperSAB)

$$\eta_{ij}(t) \;=\; \begin{cases} \eta^+ \cdot \eta_{ij}(t-1) & \text{falls} \quad \dfrac{\partial E}{\partial w_{ij}}(t-1)\,\dfrac{\partial E}{\partial w_{ij}}(t) > 0 \\ \eta^- \cdot \eta_{ij}(t-1) & \text{falls} \quad \dfrac{\partial E}{\partial w_{ij}}(t-1)\,\dfrac{\partial E}{\partial w_{ij}}(t) < 0 \\ \eta_{ij}(t-1) & \text{sonst} \end{cases}$$

- Es muß stets gelten: $0 < \eta^- < 1 < \eta^+$

- Typische Wahl der Parameter:

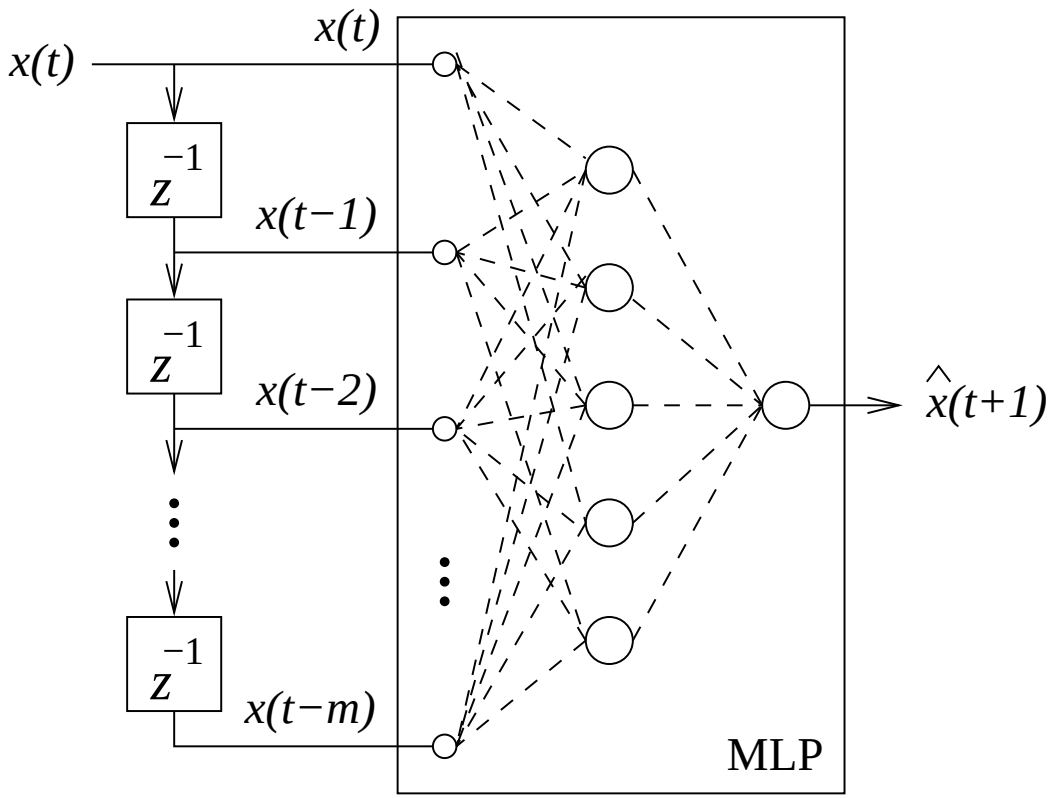
  $\eta^+ = 1.2$
  $\eta^- = 0.5$
  $\eta(0) = 0.1$ (Startwert für alle Lernraten η_{ij})
  $\eta_{\max} = 1.0$ (Beschränkung der Lernraten nach oben)
  $\eta_{\min} = 10^{-6}$ (Beschränkung der Lernraten nach unten)

- Verwendung nur im Batch-Lernmodus !

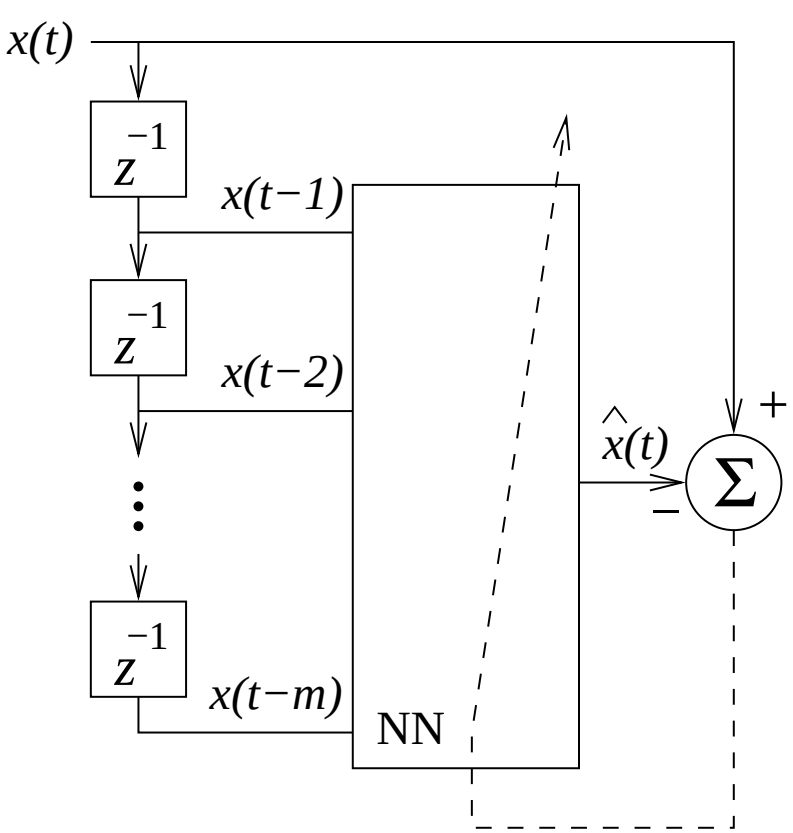
6.4 Neuronale Netze zur Verarbeitung von Zeitreihen

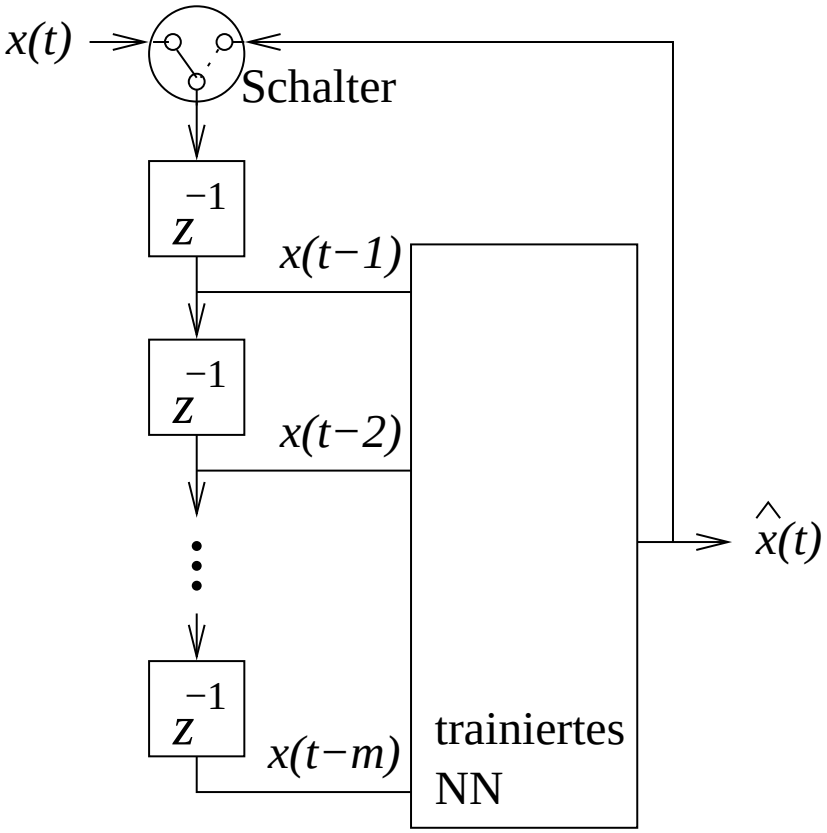
- *Aufgabe:* Erlernen einer Zeitreihe $x(t+1) = f(x(t), x(t-1), x(t-2), \ldots)$

- *Idee:* Verzögerungskette am Eingang eines neuronalen Netzwerks, z.B. eines m-h-1 MLPs:



- *Training* und *Prognose* von $\hat{x} = f(x(t-1), \ldots, x(t-m))$:

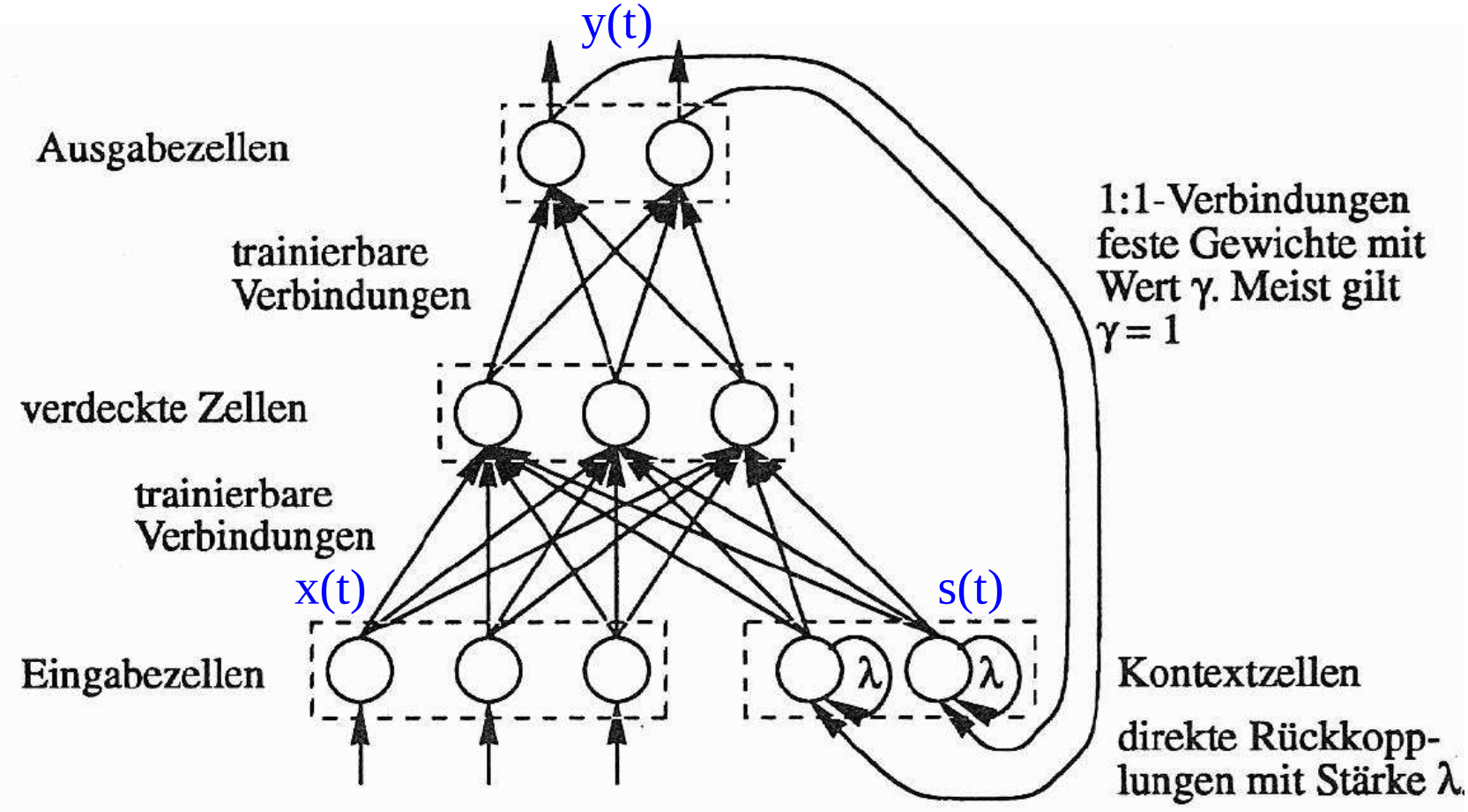




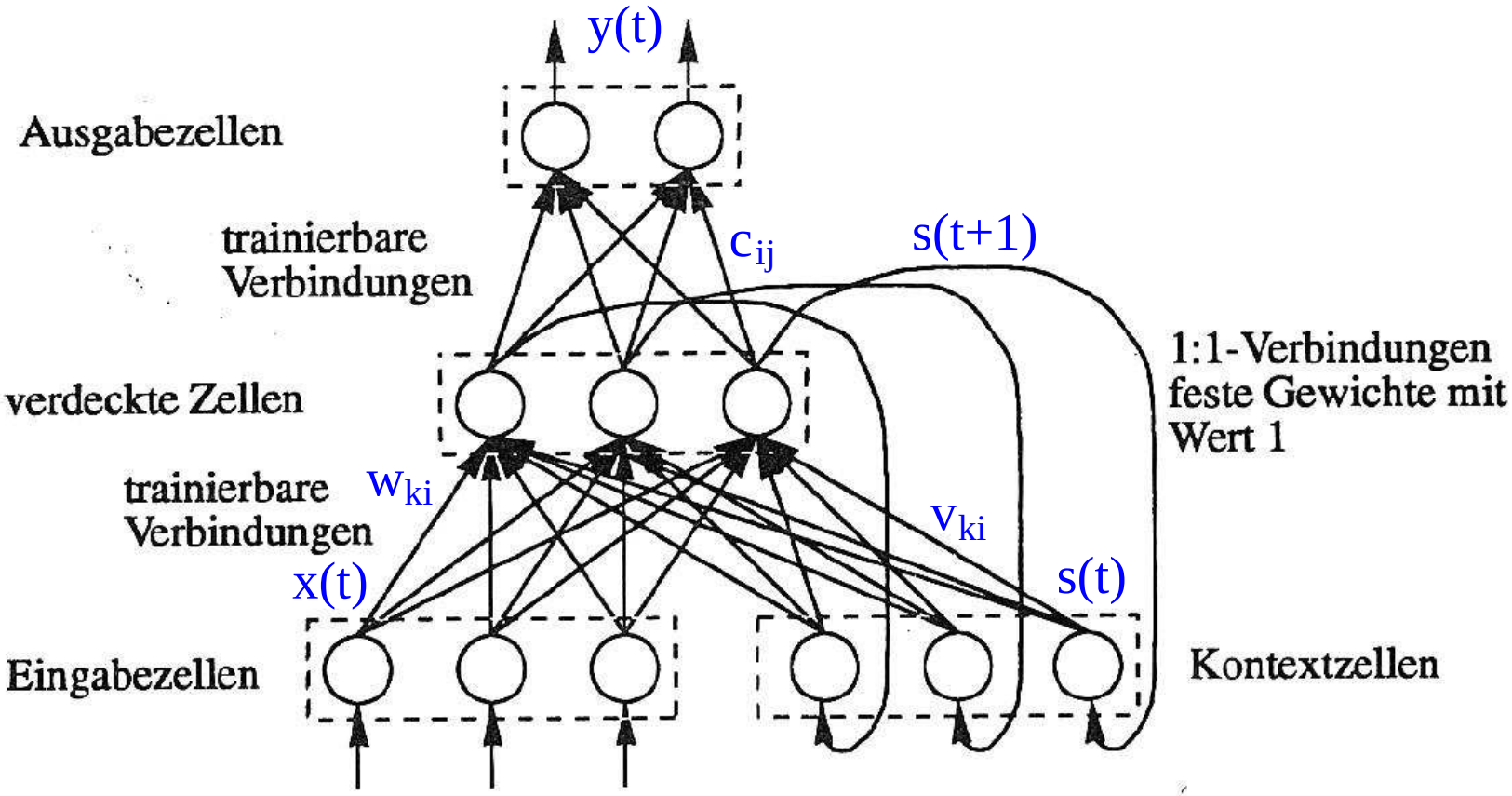
- *Probleme:* m ist fester Architekturparameter, zwei gleiche Teilfolgen bewirken unabhängig vom Kontext immer die gleiche Ausgabe

Architekturen partiell rekurrenter Netze

- Jordan Netzwerk [Jordan 1986]:



- Elman Netzwerk [Elman 1990]:



- *Vorteile* des Elman-Netzwerks: interne Repräsentation einer Sequenz ist unabhängig von der Ausgabe $\mathbf{y}$, Zahl der Kontextzellen ist unabhängig von der Ausgabedimension!

Training eines partiell rekurrenten Netzes

Möglichkeit A: Modifikation eines Lernverfahrens für nichtrekurrente Netze (z.B. Error Backpropagation, RPROP, Quickprop)

- *Algorithmus* (für Elman-Netzwerk):

  Seien w_{ki} und v_{ki} die Gewichte von Eingabeknoten u_k bzw. Kontextknoten s_k zum verdeckten Neuron i und c_{ij} die Gewichte der zweiten Netzwerkschicht

  1) Setze $t = t_0$, initialisiere Kontextzellen $\mathbf{s}(t_0) = 0$
  2) Berechne $\Delta w_{ki}(t)$, $\Delta v_{ki}(t)$ und $\Delta c_{ij}(t)$ gemäß Lernregel für eine Eingabe $\mathbf{x}(t)$ mit Sollwert $\mathbf{T}(t)$ ohne Beachtung rekurrenter Verbindungen
  3) Setze $t = t + 1$, aktualisiere die Ausgabe $\mathbf{s}(t)$ der Kontextzellen und gehe zu 2)

- *Eigenschaften:* Fehler von $\mathbf{y}(t) = f(\mathbf{x}(t))$ wird minimiert, keine Klassifikation von Sequenzen möglich.

Möglichkeit B: Verwendung eines Lernverfahrens für rekurrente Netze (z.B. BPTT [Rumelhart 86], RTRL [Willliams 89])

- Idee von BPTT (*"Backpropagation Through Time"*): Entfaltung des Netzwerks in der *Zeit* !

- Gradientenabstieg zur Minimierung von $E = \sum_{t=t_0}^{t_{\max}} E(t)$

  mit $E(t) = \begin{cases} ||\mathbf{T}(t) - \mathbf{y}(t)|| & \text{falls } \mathbf{T}(t) \text{ zum Zeitpunkt } t \text{ vorliegt} \\ 0 & \text{sonst} \end{cases}$

- *Eigenschaften:* Fehler von $\mathbf{y}(t_{\max}) = f(\mathbf{x}(t_0), \ldots, \mathbf{x}(t_{\max}))$ wird minimiert, auch Klassifikation von Sequenzen variabler Länge möglich!

BPTT Algorithmus für Elman Netzwerk

Gegeben sei ein $(m+h)-h-n$ Elman Netzwerk

mit:
- w_{ki} : Gewichte von Eingabeknoten u_k zum verdeckten Neuron i
- v_{ki} : Gewichte von Kontextknoten s_k zum verdeckten Neuron i
- c_{ij} : Gewichte vom verdeckten Neuron i zum Ausgabeneuron j
- $\delta_j^{(y)}$: Fehler am Ausgabeneuron j
- $\delta_i^{(s)}$: Fehler am verdeckten Neuron i

Lineare Ausgabeneuronen $j = 1, \ldots, n$:

Annahme: $E(t) = 0$ für $t \neq t_{\max}$

für $t = t_{max}$ gilt:
$$\delta_j^{(y)}(t) = T_j(t) - y_j(t)$$
$$\Delta c_{ij} = s_i(t+1)\,\delta_j^{(y)}(t)$$

für $t < t_{max}$ gilt:
$$\delta_j^{(y)}(t) = 0$$
$$\Delta c_{ij} = 0$$

Sigmoide verdeckte Neuronen $i = 1, \ldots, h$ **:**

für $t = t_{max}$

$$\delta_i^{(s)}(t) = \sum_{j=1}^{n} c_{ij}\, \delta_j^{(y)}(t) \cdot s_i'(t+1)$$

$$\Delta v_{ki}(t) = s_k(t)\, \delta_i^{(s)}(t)$$

$$\Delta w_{ki}(t) = x_k(t)\, \delta_i^{(s)}(t)$$

für $t_0 \leq t < t_{max}$

$$\delta_i^{(s)}(t) = \sum_{k=1}^{h} v_{ki}\, \delta_k^{(s)}(t+1) \cdot s_i'(t+1)$$

$$\Delta v_{ki}(t) = s_k(t)\, \delta_i^{(s)}(t)$$

$$\Delta w_{ki}(t) = x_k(t)\, \delta_i^{(s)}(t)$$

Resultierende Lernregeln:

$$c_{ij} = c_{ij} + \eta_1 \Delta c_{ij}, \qquad w_{ki} = w_{ki} + \eta_2 \sum_{t=t_0}^{t_{\max}} \Delta w_{ki}(t), \qquad v_{ki} = v_{ki} + \eta_2 \sum_{t=t_0}^{t_{\max}} \Delta v_{ki}(t)$$

6.5 Radiale Basisfunktionen

- *Motivation:* Es wird eine Lösung für das folgende Interpolationsproblem gesucht:

  Gegeben sei eine Menge $M = \{(\mathbf{x}^\mu, \mathbf{T}^\mu) \ : \ \mu = 1, \ldots, p\}$ mit $\mathbf{x}^\mu \in \mathbb{R}^m$ und $\mathbf{T}^\mu \in \mathbb{R}^n$.

  Gibt es eine Funktion $g \ : \ \mathbb{R}^m \to \mathbb{R}^n$ mit $g(\mathbf{x}^\mu) = \mathbf{T}^\mu \ \ \forall \mu = 1, \ldots, p$?

- *Idee:* Annäherung von g durch eine *Linearkombination* von p Funktionen $h_\nu(\mathbf{x}) = h(||\mathbf{x}^\nu - \mathbf{x}||)$ mit (beliebig oft differenzierbarer) radialsymmetrischer Funktion $h \ : \ \mathbb{R}^+ \to \mathbb{R}^+$

- Das Interpolationsproblem hat (hier für $n = 1$) die Form

$$g(\mathbf{x}^\mu) = \sum_{\nu=1}^{p} w_\nu h_\nu(\mathbf{x}^\mu) = \sum_{\nu=1}^{p} w_\nu h(||\mathbf{x}^\nu - \mathbf{x}^\mu||) \quad \forall \mu = 1, \ldots, p$$

- w_ν können analytisch bestimmen werden:

  $$g(\mathbf{x}^\mu) = \sum_{\nu=1}^{p} w_\nu h(||\mathbf{x}^\nu - \mathbf{x}^\mu||) = \mathbf{T}^\mu, \ \mu = 1, \ldots, p.$$

  In Matrixnotation gilt: $\mathbf{Hw} = \mathbf{T}$ mit $\mathbf{H} := (H_{\mu\nu})$ und $H_{\mu\nu} := h_\mu(x^\nu)$
  $\Rightarrow$ Falls $\mathbf{H}$ invertierbar ist, so gilt: $\mathbf{w} = \mathbf{H}^{-1}\mathbf{T}$, mit den Vektoren $\mathbf{w} = (w_1, \ldots w_p)^t$ und $\mathbf{T} = (T^1, \ldots, t^p)^t$.

- einige radialsymmetrische Basisfunktionen (mit $r = ||\mathbf{x} - \mathbf{x}^\mu||$):
  - Gauss-Funktion:

    $h(r) = \exp(-\frac{r^2}{2\sigma^2})$ mit $\sigma \neq 0$
  - Inverse multiquadratische Funktionen:

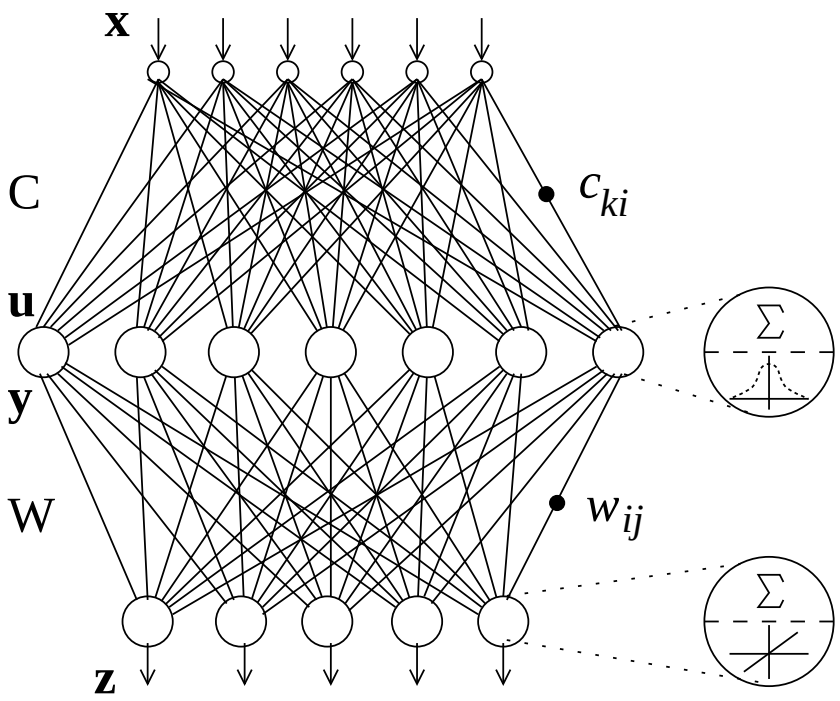
    $h(r) = \frac{1}{(r^2 + c^2)^\alpha}$ mit $c \neq 0$ und $\alpha > 0$
  - Multiquadratische Funktionen:

    $h(r) = (r^2 + c^2)^\beta$ mit $c \neq 0$ und $0 < \beta \leq 1$

Radiales Basisfunktionen-Netzwerk (RBF)

- Aufbau eines m-h-n RBF-Netzwerks:

  m Eingabeknoten
  h RBF-Neuronen
  n lineare Neuronen



- Berechnung der Netzausgabe $\mathbf{z}$:

$$u_i = \sqrt{\sum_{k=1}^{m} (x_k - c_{ki})^2} = ||\mathbf{x} - \mathbf{c}_i||$$

$$y_i = h(u_i)$$

$$z_j = \sum_{i=1}^{h} y_i w_{ij} \; [\, + \text{bias}_j \,]$$

Herleitung der Lernregel für RBF-Netzwerk

Es soll eine auf Gradientenabstieg basierende Lernregel für das RBF-Netzwerk hergeleitet werden.

Für die Gewichte w_{ij} der Ausgabeschicht gilt:

$$\begin{aligned}
\frac{\partial E_\mu}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \|\mathbf{T} - \mathbf{z}\|^2 \\
&= \frac{\partial}{\partial w_{ij}} \sum_j (T_j - z_j)^2 \\
&= -2(T_j - z_j) \frac{\partial}{\partial w_{ij}} \sum_{\tilde{i}} y_{\tilde{i}} w_{\tilde{i}j} \\
&= -2\, y_i\, \delta_j
\end{aligned}$$

wobei $\delta_j = (T_j - z_j)$ den Fehler von Neuron j der Ausgabeschicht für ein Muster bezeichnet

resultierende Lernregeln für Gewichte w_{ij} der Ausgabeschicht:

im *Online-Modus:*

$$w_{ij} = w_{ij} - \eta_1 \frac{\partial E_\mu}{\partial w_{ij}} = w_{ij} + \eta_1 \, y_i \, \delta_j$$

$$[\text{bias}_j = \text{bias}_j - \eta_1 \frac{\partial E_\mu}{\partial w_{ij}} = \text{bias}_j + \eta_1 \, \delta_j]$$

im *Batch-Modus:*

$$w_{ij} = w_{ij} - \eta_1 \frac{\partial E}{\partial w_{ij}} = w_{ij} + \eta_1 \sum_\mu y_i^\mu \, \delta_j^\mu$$

$$[\text{bias}_j = \text{bias}_j - \eta_1 \frac{\partial E}{\partial w_{ij}} = \text{bias}_j + \eta_1 \sum_\mu \delta_j^\mu]$$

Für Gewichte c_{ki} der Eingabeschicht gilt:

$$\begin{aligned}
\frac{\partial E_\mu}{\partial c_{ki}} &= \frac{\partial E_\mu}{\partial y_i} \frac{\partial y_i}{\partial c_{ki}} \\
&= -2 \sum_j (T_j - z_j) \frac{\partial}{\partial y_i} \sum_{\tilde{i}} y_{\tilde{i}} \, w_{\tilde{i}j} \frac{\partial}{\partial c_{ki}} h(u_i) \\
&= -2 \sum_j (T_j - z_j) \, w_{ij} \, h'(u_i) \frac{\partial}{\partial c_{ki}} \sqrt{\sum_{\tilde{k}} (x_{\tilde{k}} - c_{\tilde{k}i})^2} \\
&= -2 \sum_j (T_j - z_j) \, w_{ij} \, h'(u_i) \frac{1}{2u_i} (-2) \, (x_k - c_{ki}) \\
&= 2 \sum_j \delta_j \, w_{ij} \, h'(u_i) \frac{1}{u_i} (x_k - c_{ki})
\end{aligned}$$

Speziell für $h(u) = e^{-u^2/2\sigma^2} = e^{-u^2 s}$ ergibt sich:

$$\begin{aligned}
\frac{\partial E_\mu}{\partial c_{ki}} &= -2 \sum_j (T_j - z_j) \, w_{ij} \, e^{-u_i^2 s} \, 2s(x_k - c_{ki}) \\
&= -4 \sum_j \delta_j \, w_{ij} \, y_i \, s \, (x_k - c_{ki})
\end{aligned}$$

resultierende Lernregeln für Gewichte c_{ki} der Eingabeschicht bei Verwendung der radialen Basisfunktion $h(u) = e^{-u^2/2\sigma^2} = e^{-u^2 s}$:

im *Online-Modus:*

$$c_{ki} = c_{ki} - \eta_2 \frac{\partial E_\mu}{\partial c_{ki}} = c_{ki} + \eta_2 \left(x_k - c_{ki}\right) y_i \, s \sum_j \delta_j w_{ij}$$

im *Batch-Modus:*

$$c_{ki} = c_{ki} - \eta_2 \frac{\partial E}{\partial c_{ki}} = c_{ki} + \eta_2 \sum_\mu (x_k^\mu - c_{ki}) \, y_i^\mu s \sum_j \delta_j^\mu w_{ij}$$

Lernalgorithmus (online) für RBF-Netzwerk

Gegeben: m-h-n RBF-Netzwerk
verwendete radiale Basisfunktion: $y = h(u) = e^{-u^2/2\sigma^2} := e^{-u^2 s}$
Menge von p gelabelten Mustern $(\mathbf{x}^\mu, \mathbf{T}^\mu) \in \mathbb{R}^m \times \mathbb{R}^n$

Schritt 1: Initialisierung

Setze w_{ij} für $i = 1, \ldots, h$, $j = 1, \ldots, n$ auf kleine Zufallswerte und wähle c_{ki} für $k = 1, \ldots, m$, $i = 1, \ldots, h$ derart, dass die Daten im Eingaberaum überdecken

Schritt 2: Berechnung der Netzausgabe $\mathbf{z}$

Wähle nächstes gelabeltes Musterpaar $(\mathbf{x}, \mathbf{T})$ und berechne:

$$u_i^2 = \sum_{k=1}^{m} (x_k - c_{ki})^2 \quad \text{für } i = 1, \ldots, h$$

$$y_i = e^{-u_i^2/2\sigma^2} = e^{-u_i^2 s} \quad \text{für } i = 1, \ldots, h$$

$$z_j = \sum_{i=1}^{h} y_i w_{ij} \quad \text{für } j = 1, \ldots, n$$

Schritt 3: Bestimmung des Fehlers am Netzausgang

Berechne $\delta_j = (T_j - z_j)$ für $j = 1, \ldots, n$

Schritt 4: Lernen

Adaptiere Gewichte: $w_{ij} = w_{ij} + \eta_1 \, y_i \, \delta_j$

$$c_{ki} = c_{ki} + \eta_2 \, (x_k - c_{ki}) \, y_i \, s \sum_{j=1}^{n} \delta_j w_{ij}$$

für $k = 1, \ldots, m$, $i = 1, \ldots, h$, $j = 1, \ldots, n$

Schritt 5: Ende

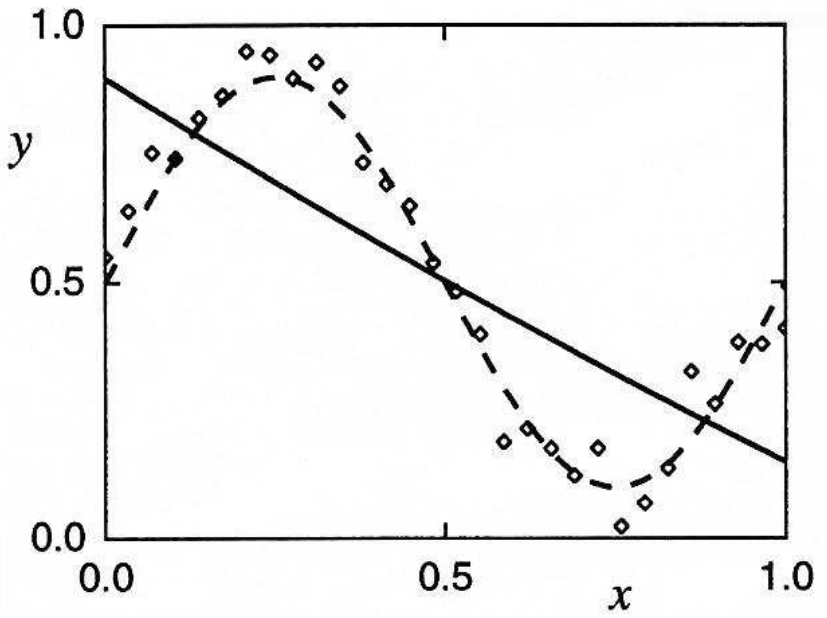
Falls Endekriterium nicht erfüllt, gehe zurück zu *2*

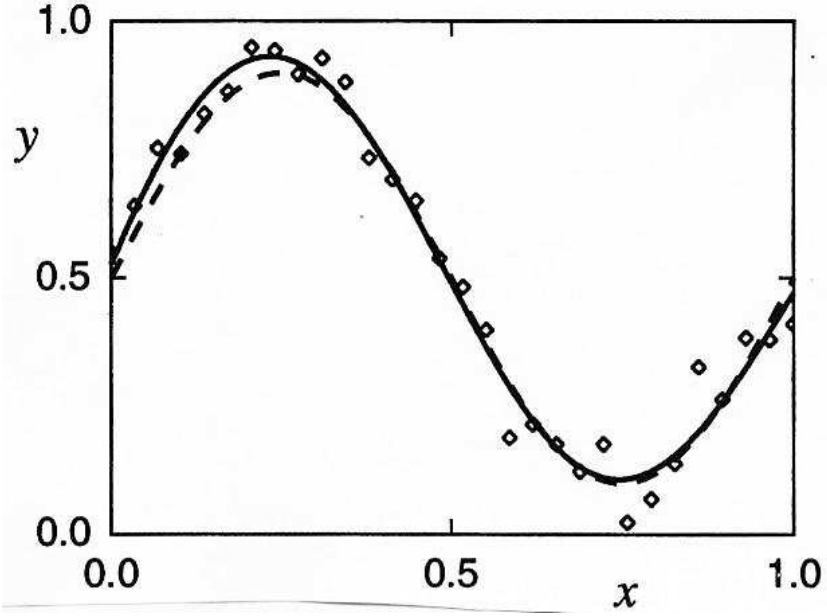
Bemerkungen zum RBF-Netzwerk

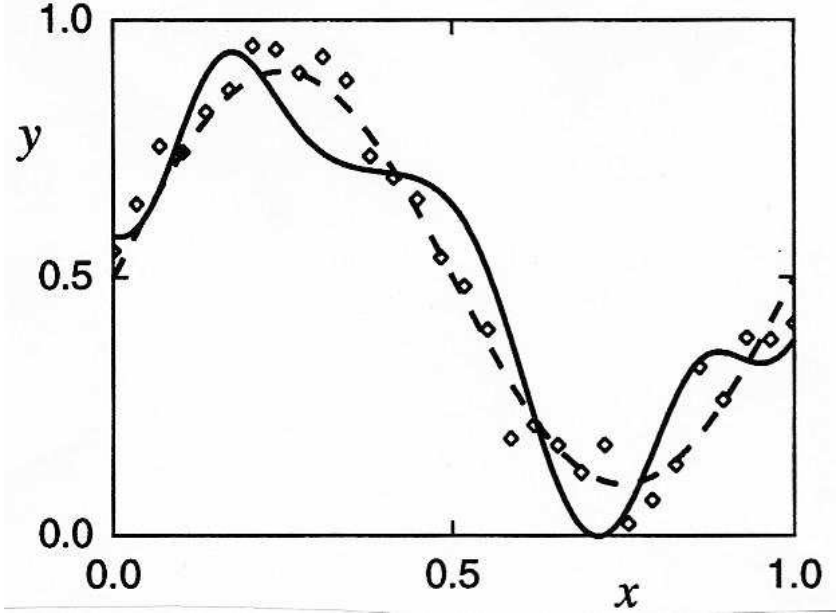
- Anzahl der RBF-Neuronen ist kleiner als Anzahl Trainingsmuster
- Vektor $\mathbf{c}_i$ wird auch als *Prototyp* bzw. *Zentrum* bezeichnet
- Initialisierung der Gewichte c_{ki} z.B. durch
  - äquidistante Verteilung in einem Intervall [min,max]^m
  - durch Clusteranalyse (vgl. Kapitel 8)
- Verhalten des RBF-Netzwerks stark abhängig von der Wahl eines guten Wertes für σ bzw. s

  z.B.: $\sigma \in [d_{\min}, 2 \cdot d_{\min}]$ (mit $d_{\min}$ = kleinster Abstand zwischen zwei Zentren)
- Parameter σ_i bzw. s_i kann auch für jedes RBF-Neuron i individuell gewählt und adaptiert werden ($\Rightarrow$ Übung)
- Adaption von c_{ki}, w_{ij} und σ_i bzw. s_i ist simultan oder sequentiell möglich !

Approximation einer verrauschten Funktion mit RBF-Netz:
(mit 5 RBF-Neuronen bei $\sigma = 10$, mit $\sigma = 0.4$, $\sigma = 0.08$)





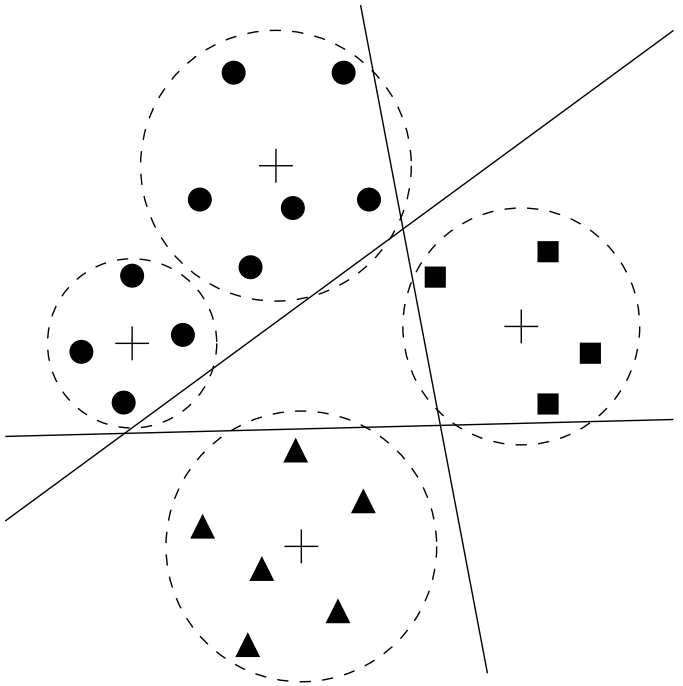


Unterschiede MLP und RBF

- bei Klassifikation:

  MLP: Trennung durch *Hyperebenen*

  RBF: *Hyperkugeln* umfassen Punkte einer Klasse



- bei MLP ist Repräsentation in verdeckter Schicht *verteilt*, bei RBF *lokal*

- Initialisierung der Gewichte bei MLP *zufällig*, bei RBF *datenabhängig*

- MLP *und* RBF können Funktionen beliebig genau approximieren

- i.a. schnellere Konvergenz mit RBF bei guter Initialisierung

6.6 Konstruktive Lernverfahren

- *Problem:* In allen bisherigen neuronalen Netzmodellen wird die Architektur vor Start des Lernverfahrens festgelegt

  $\Rightarrow$ während des Trainings ist keine Anpassung der Architektur an das zu lernende Problem möglich.

- *Idee:* Man startet mit einem kleinen neuronalen Netzwerk und fügt während des Trainings bei Bedarf noch weitere Neuronen oder Neuronenschichten hinzu.

- Beispiele:

  1. Upstart Algorithmus (konstruiert Binärbaum aus Perzeptronen)
  2. Cascade Correlation (konstruiert pyramidenartige Architektur aus sigmoiden Neuronen)

Upstart Algorithmus [Frean 1990]

- rekursiver Algorithmus zum Erlernen *beliebiger* binärer Abbildungen mit Schwellwertneuronen:

```
upstart(T_0, T_1) {
   trainiere ein Perzeptron Z mit T_0 ∪ T_1
   wenn mind. 1 Muster aus T_0 falsch klassifiziert wird:
      bilde T_0^- ⊆ T_0
      generiere neues Perzeptron X
      trainiere X mit upstart(T_1, T_0^-)
   wenn mind. 1 Muster aus T_1 falsch klassifiziert wird:
      bilde T_1^- ⊆ T_1
      generiere neues Perzeptron Y
      trainiere Y mit upstart(T_0, T_1^-)
   berechne MAX= max_μ | Σ_i w_i x_i^(μ) | von Z
   verbinde X mit Z durch w < −MAX und Y mit Z durch w > MAX
}
```

mit: $\mathcal{T}_0, \mathcal{T}_1$ Trainingsmenge für Ausgabe 0 bzw. 1

$\mathcal{T}_0^-, \mathcal{T}_1^-$ Menge falsch klassifizierter Muster aus $\mathcal{T}_0$ bzw. $\mathcal{T}_1$

Cascade Correlation

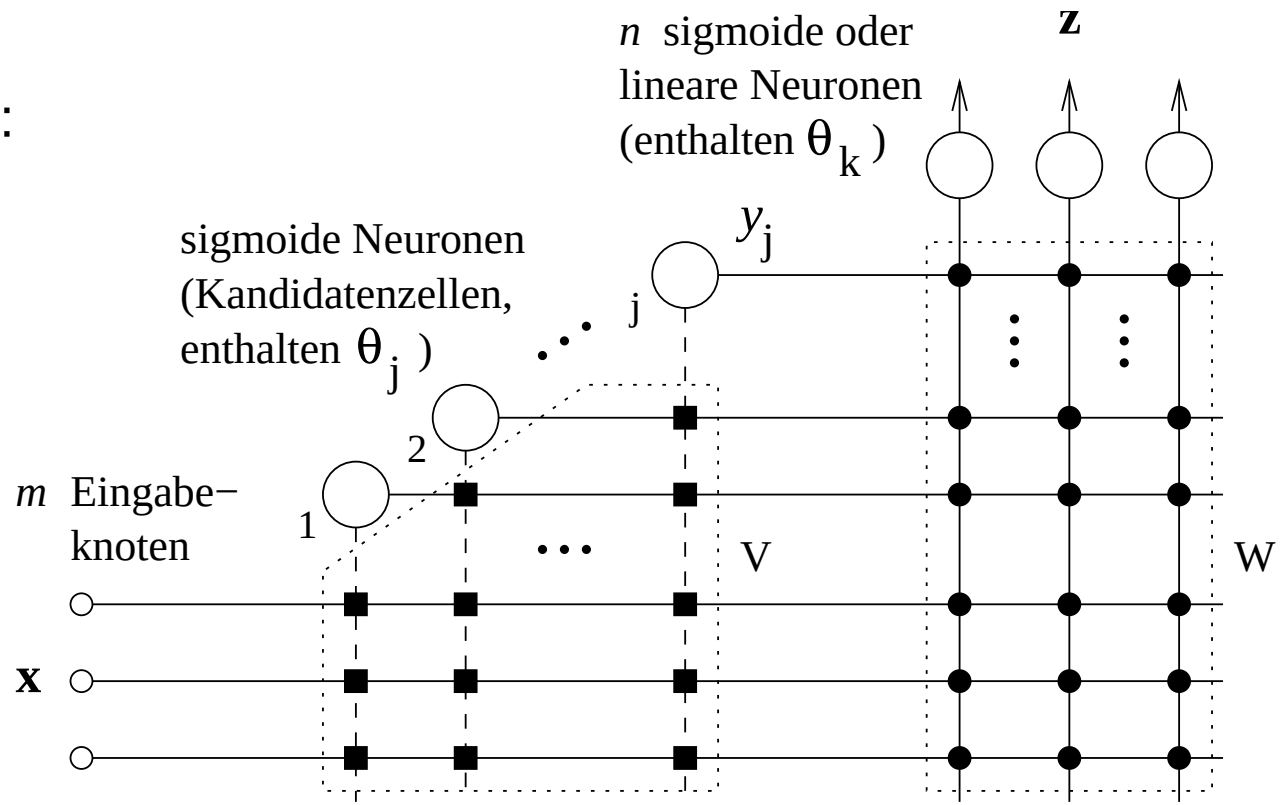
- *Motivation* : bei großem Fehler am Ausgang für ein Muster A werden *alle* Gewichte adaptiert

  $\Rightarrow$ aufgrund fehlender Absprache zwischen den Neuronen der verdeckten Schicht kann Fehler für ein Muster B wachsen (*"moving target problem"*)

- *Idee* :
  - Gewichte v_{ij} eines jedes verdeckten Neurons j werden *separat* trainiert, indem die Korrelation zwischen der Ausgabe des Neurons und dem Fehler maximiert wird
  - anschließend werden die Gewichte v_{ij} von Neuron j *eingefroren*
  - Hinzufügen weiterer verdeckter Neuronen (als *Kandidatenzellen* bezeichnet), bis Fehler am Ausgang ausreichend klein ist

- automatische Bestimmung einer guten Topologie !

- Cascade Correlation Architektur [Fahlman 90]:



- *Ziel:* Maximierung der Korrelation $S_j = \sum_k |\sum_p (y_{pj} - \overline{y_j})(\delta_{pk} - \overline{\delta_k})|$

mit
- y_{pj} Ausgabe der Kandidatenzelle j für Muster p
- $\overline{y_j}$ mittlere Ausgabe der Kandidatenzelle j
- δ_{pk} Fehler der Ausgangszelle k für Muster p
- $\overline{\delta_k}$ mittlerer Fehler der Ausgangszelle k

- Für Eingangsgewichte v_{ij} der j-ten Kandidatenzelle gilt:

$$
\begin{aligned}
\frac{\partial S_j}{\partial v_{ij}} &= \sum_k \frac{\partial}{\partial v_{ij}} \left| \sum_p (y_{pj} - \overline{y_j})(\delta_{pk} - \overline{\delta_k}) \right| \\
&= \sum_k \sigma_k \sum_p \frac{\partial}{\partial v_{ij}} (y_{pj} - \overline{y_j})(\delta_{pk} - \overline{\delta_k}) \\
&= \sum_k \sigma_k \sum_p \frac{\partial}{\partial v_{ij}} y_{pj} \, (\delta_{pk} - \overline{\delta_k}) \\
&= \sum_k \sigma_k \sum_p \frac{\partial}{\partial v_{ij}} f(\sum_i I_{pi} v_{ij})(\delta_{pk} - \overline{\delta_k}) \\
&= \sum_k \sigma_k \sum_p f'(x_{pj}) I_{pi} (\delta_{pk} - \overline{\delta_k})
\end{aligned}
$$

mit $\sigma_k = \text{sgn}\left(\sum_p (y_{pj} - \overline{y_j}\,)(\delta_{pk} - \overline{\delta_k}) \right)$

$$
I_{pi} = \begin{cases} u_{pi} & \text{für externen Eingang } i \\ y_{pi} & \text{für Kandidatenzelle } i < j \end{cases}
$$

Lernalgorithmus für Cascade Correlation

Schritt 1:
Trainiere die Gewichte w_{ik} und θ_k der Ausgangsknoten mit einem geeigneten Algorithmus:

- Perzeptron-Lernalgorithmus oder
- Delta-Lernregel: $\Delta w_{ik} = u_i(T_k - y_k)f'(x_k)$
- Quickprop-Verfahren

bis Fehler E am Netzausgang nicht weiter sinkt

Schritt 2:
- Stop, wenn Fehler E ausreichend klein
- Ansonsten generiere eine neue Kandidatenzelle j und initialisiere v_{ij} zufällig

Schritt 3:

Bestimme Gewichte v_{ij} und θ_j der Kandidatenzelle j derart, daß Korrelation S_j maximiert wird. Adaptiere hierzu v_{ij} und θ_j durch Gradientenaufstieg gemäß

$$v_{ij} = v_{ij} + \eta \sum_k \sigma_k \sum_p f'(x_{pj}) I_{pi} (\delta_{pk} - \overline{\delta_k})$$

$$\theta_j = \theta_j + \eta \sum_k \sigma_k \sum_p f'(x_{pj}) (\delta_{pk} - \overline{\delta_k})$$

bis Korrelation S_j nicht weiter steigt

Schritt 4:

Friere alle Gewichte v_{ij} und θ_j der Kandidaten-zelle j ein

Schritt 5:

Gehe nach (1)

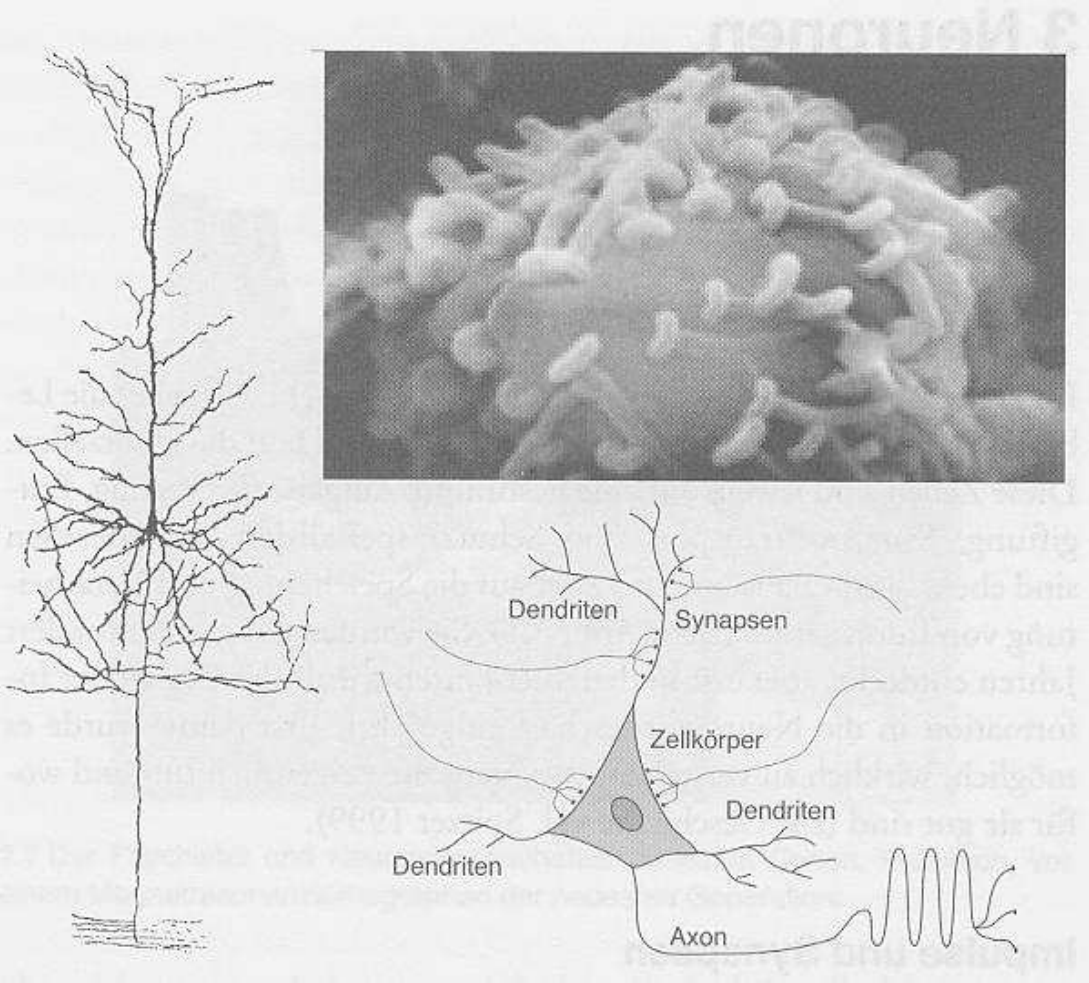
7. Unüberwachte Lernverfahren

1. Neuronale Merkmalskarten
2. Explorative Datenanalyse
3. Methoden zur Datenreduktion der Datenpunkte
4. Methoden zur Merkmalsreduktion der Merkmale

7.1 Neuronale Merkmalskarten

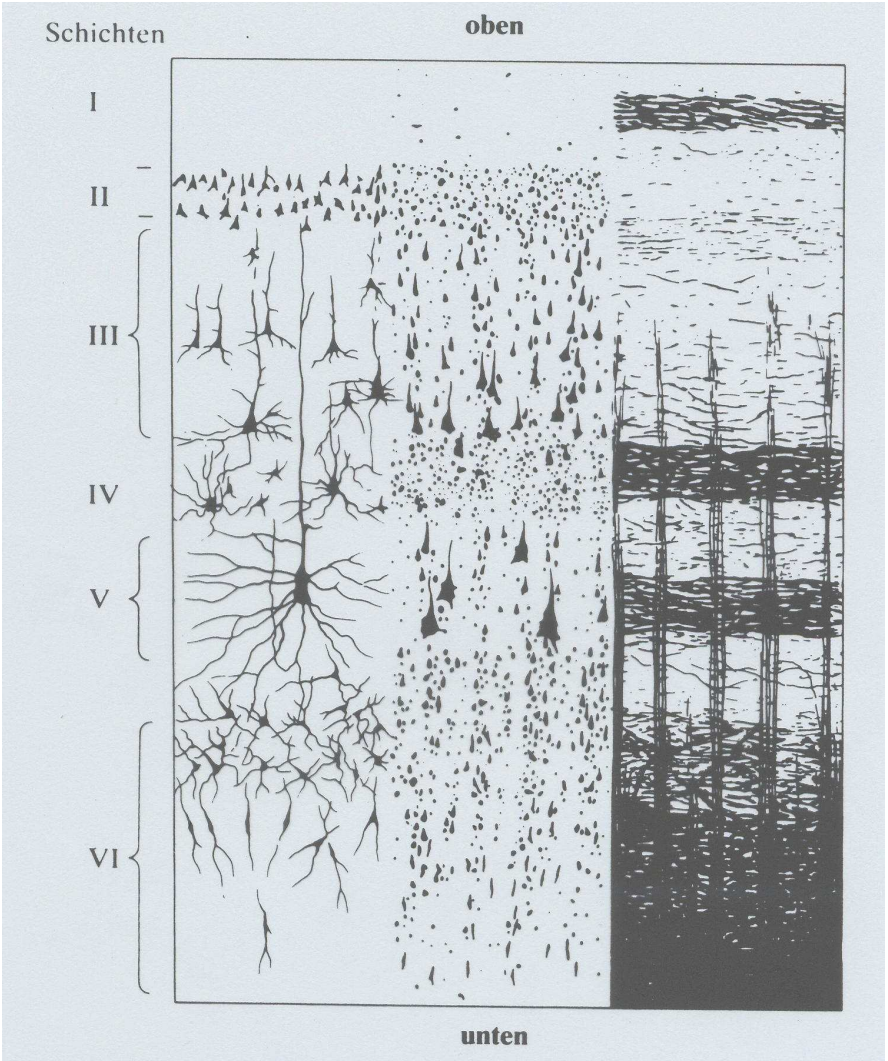
- Neuron
- Schichten der Großhirnrinde
- Kortikale Säulenarchitektur
- Laterale Hemmung
- Karten im Kortex

Neuron



Zur Erinnerung:
Lichtmikroskopische Aufnahme eines Neurons (linke) (Golgi-Färbung) und eine elektromikroskopische Aufnahme eines Zellkörpers (rechts oben), sowie eine schematische Skizze eines Neurons (rechts unten).

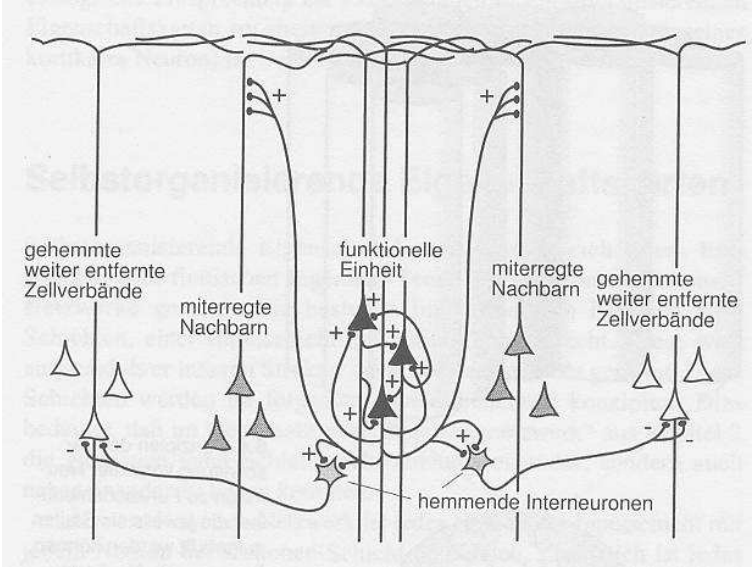
Schichten der Großhirnrinde



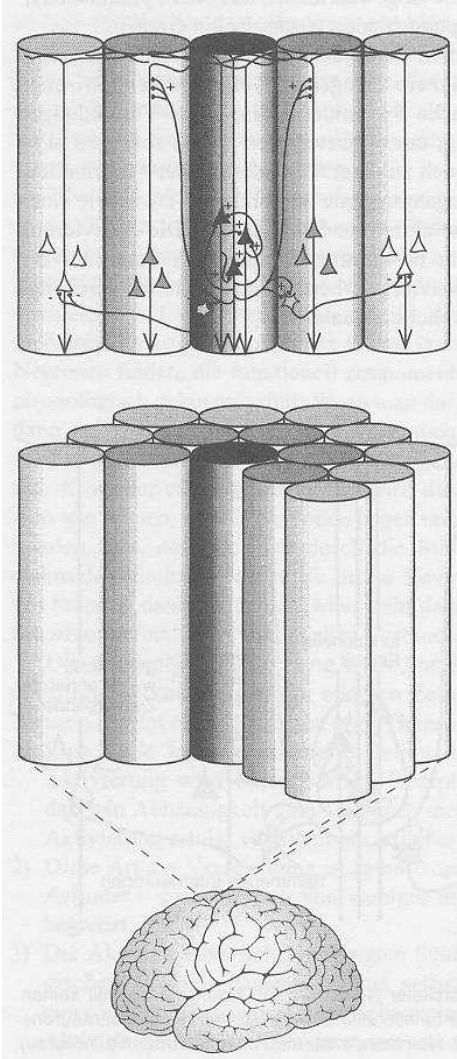
Schichtenstruktur (6 Schichten) der Großhirnrinde (Kortex):
Neuronen mit Dendriten und Axonen durch Golgi-Färbung sichtbar(links). Nissl-Färbung: Nur die Zellkörper der Neuronen sind sichtbar(Mitte). Weigert-Färbung: Sichtbar sind die Nervenfasern (rechts).

Kortikale Säulenarchitektur I

- Der Kortex weist eine sechsschichtige horizontale Gliederung auf, insbesondere durch verschiedene Zelltypen.
- Verbindungsstrukur vertikal und horizontal; Verbindungen von Pyramidenzellen in Schicht 5 liegen etwa 0,3 bis 0,5 mm um die vertikale Achse, was eine zylindrische Geometrie kortikaler Informationsverarbeitungsmodule nahelegt.
- Ungeklärt ist, ob es sich bei den kortikalen Säulen sogar um konkrete anatomische Strukturen oder um abstrakte (momentane) funktionelle Struktur handelt.
- Die unmittelbaren Nachbarn eines Neurons bilden eine funktionelle Einheit.
- Nachbarneuronen werden leicht erregt.
- Weiter entfernt liegende Neuronen werden über Interneuronen inhibiert, man spricht von *lateraler Inhibition*.

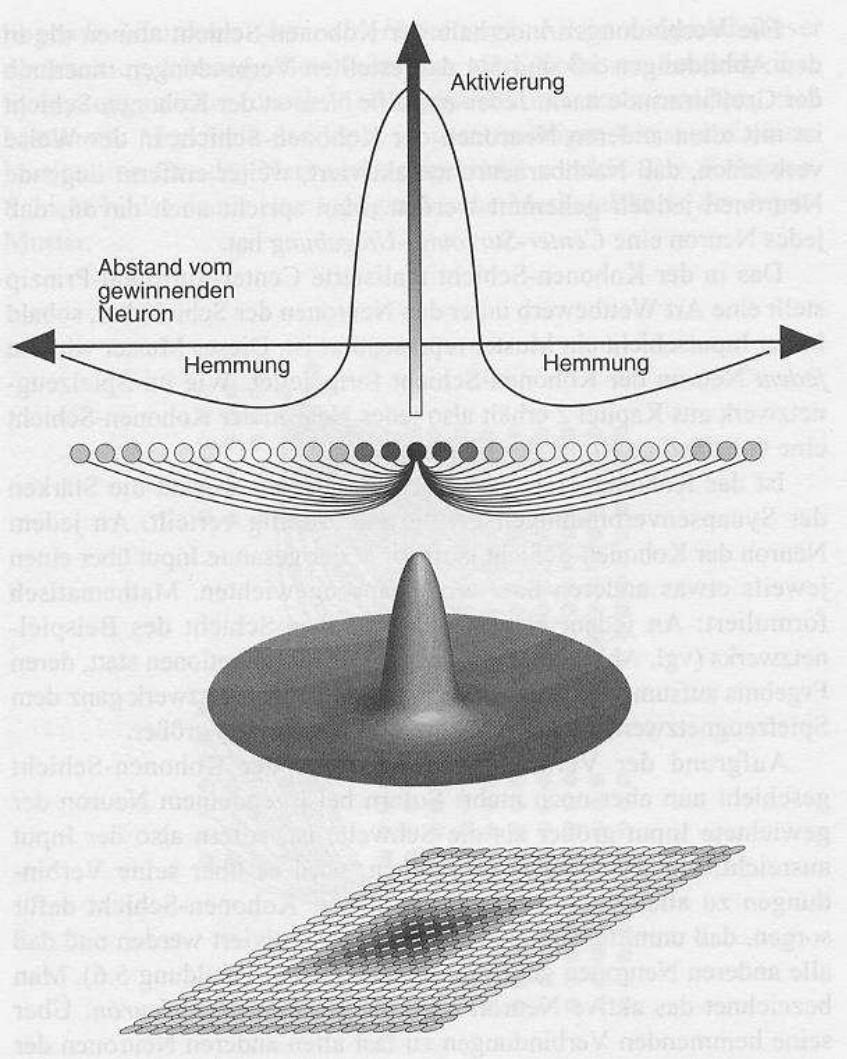


Kortikale Säulenarchitektur II



- Prinzip der kortikalen Säulen: Hierbei sind eigentlich nicht einzelne Neurone gemeint, auch wenn die kortikalen Säulen in der Simulation so aufgefasst werden.
- Der gesamte Kortex zeigt diese Säulenarchitektur.

Laterale Inhibition



- *Center-Surround-Architektur*. Jedes Neuron ist mit jedem anderen verbunden.
- Verbindungen zu benachbarten Neuronen sind exzitatorisch, weiter entfernt liegende Neuronen werden inhibiert.
- Oben: Querschnitt durch diese Aktivierung.
- Mitte: Zweidimensionale Darstellung der Aktivierung. Wegen der Form wird sie auch *mexican hat function* genannt.
- Unten: Schematische Anordung der kortikalen Säulen.
- Die laterale Inhibition bewirkt, dass das Neuron welches bei einer Eingabe am stärksten aktiviert wird, letztlich nur selbst aktiv bleibt: **The winner takes it all**
- In der Simulation wird das Neuron bestimmt, das durch eine Eingabe am stärksten aktiviert wurde: **Winner detection**.

Karten im Kortex

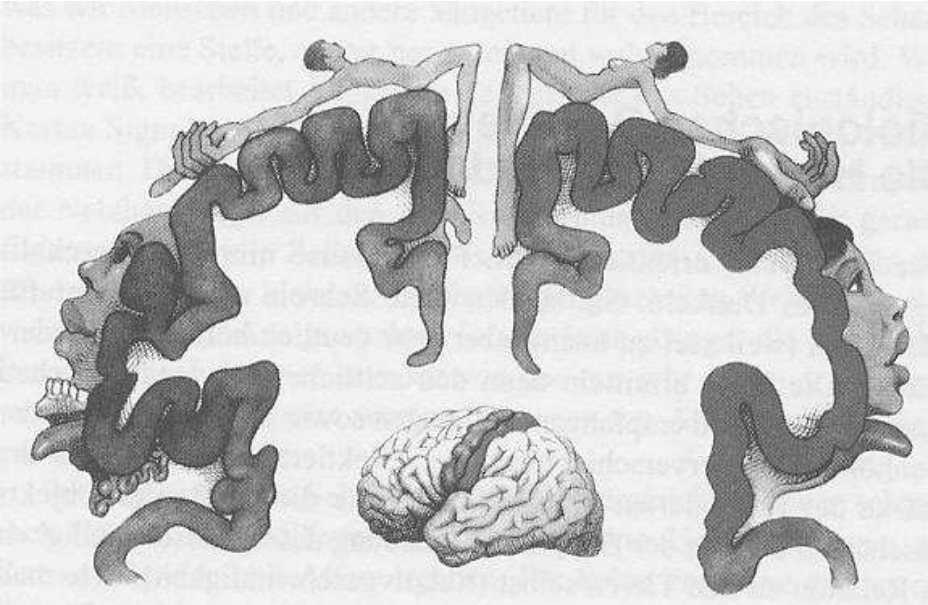


Abbildung oben:

Motorischer (links) und sensorischer Kortex (rechts).

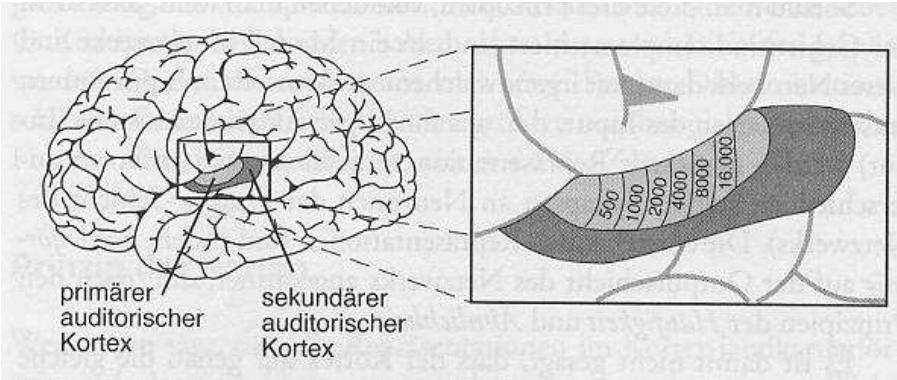
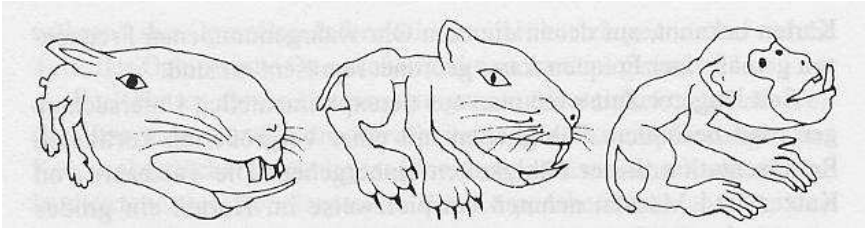


Abbildung unten:

Der auditorische Kortex (genauer das Areal A1) enthält Neuronen, die Töne nach der Frequenz sortiert kartenförmig repräsentieren.



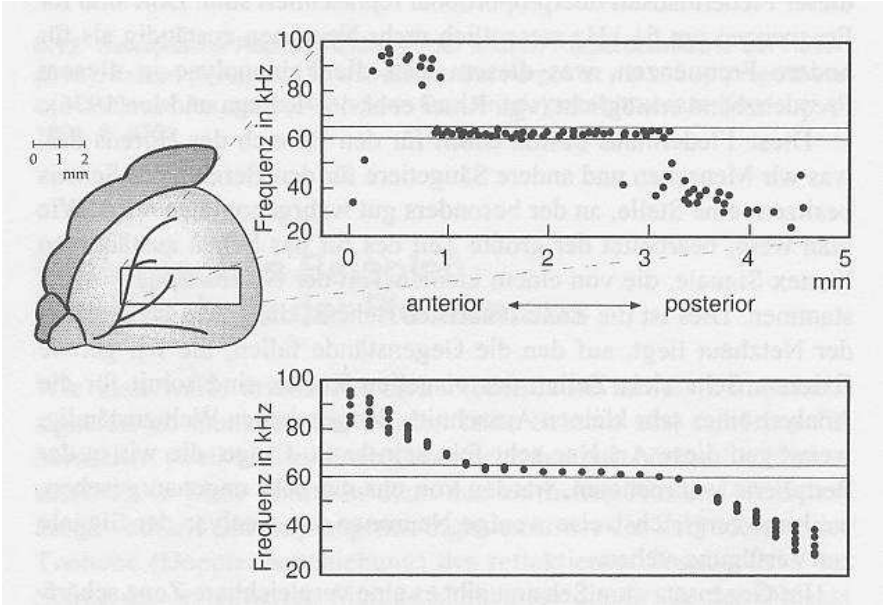


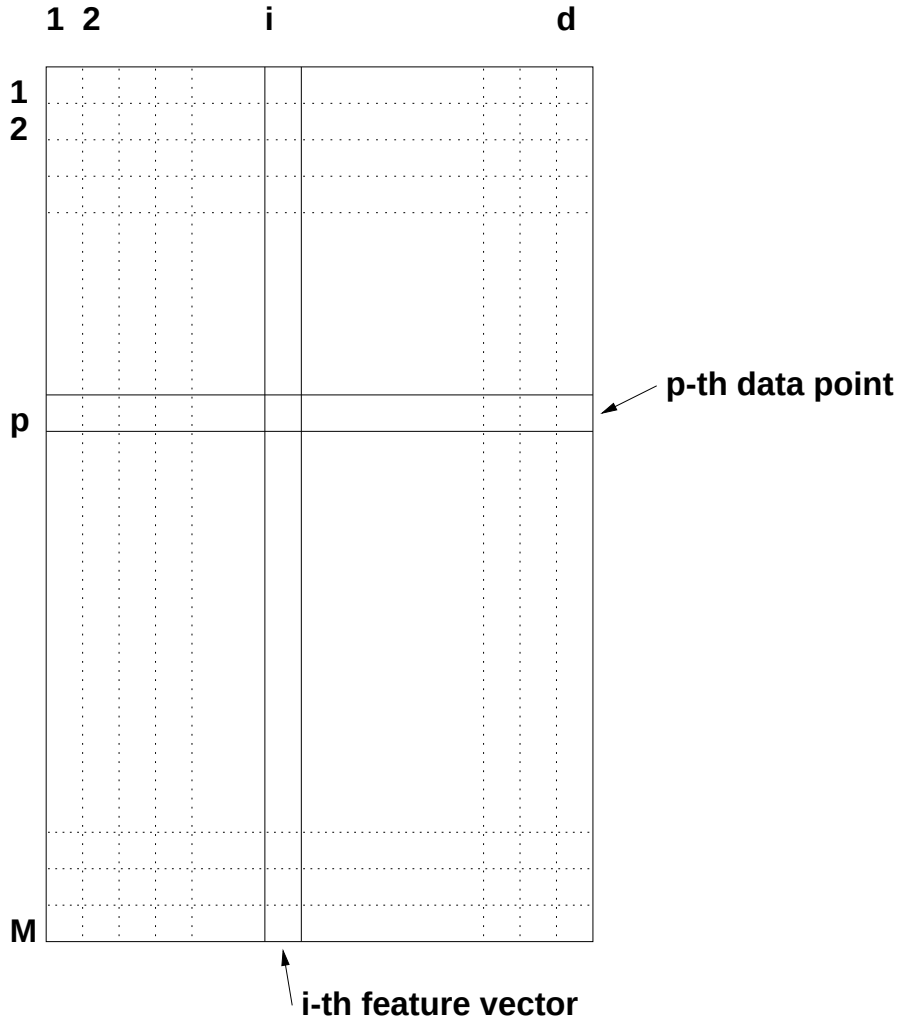
Abbildung oben:

Sensorische Repräsentation der Körperoberfläche bei Hasen (links), Katzen (Mitte) und Affen (rechts). Körper ist proportional zur Größe der ihn repräsentierenden Großhirnrinde gezeigt.

Abbildung unten:

Auditorischer Kortex der Fledermaus ist schematisch dargestellt (links). Die Verteilung der von den Neuronen repäsentierten Frequenzen ist dargestellt (rechts oben). Für Frequenzen um 61 kHz ist etwa die Hälfte der Neuronen sensitiv. Signale in diesem Frequenzbereich nutzt die Fledermaus um aus der Tonhöheverschiebung des reflektierten Signals die Relativgeschwnidigkeit eines Objektes und dem Tier selbst zu bestimmen. Ergebnis einer Computersimulation (rechts unten).

7.2 Explorative Datenanalyse



Probleme

- Viele Datenpunkte.
- Hohe Dimension des Merkmalsraumes.
- Datenpunkte zu Beginn der Analyse möglicherweise nicht vollständig bekannt.
- **Keine Lehrersignale**.

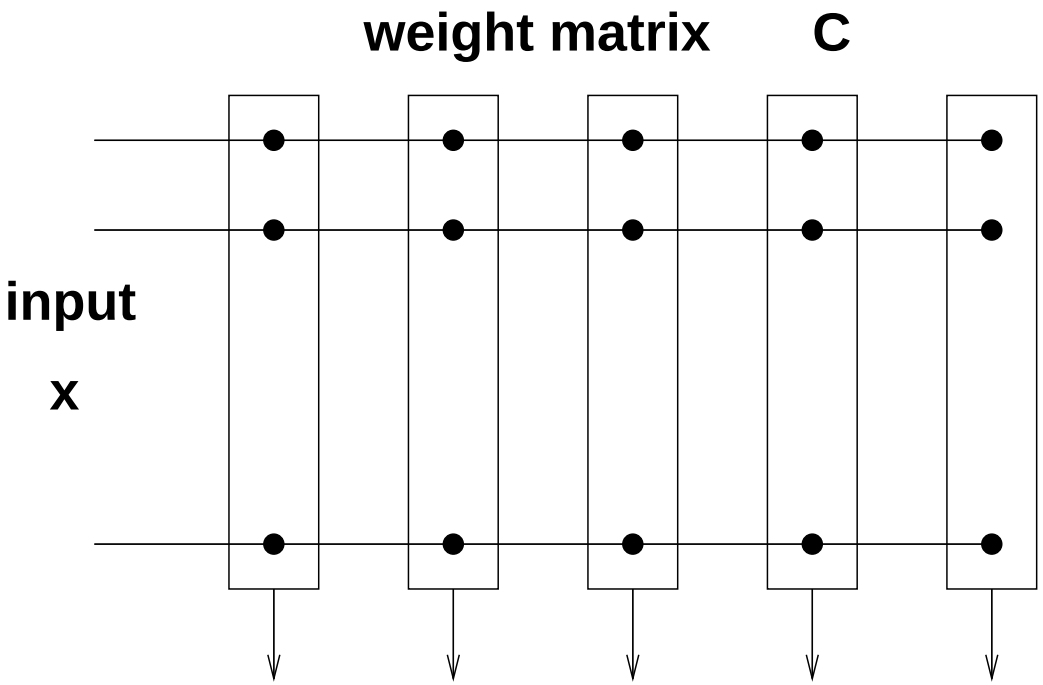
7.3 Methoden zur Reduktion der Datenpunkte

- Zielsetzung
- Kompetitive Netze, Distanz oder Skalarprodukt
- Selbstorganisierende Merkmalskarten
- Einfache kompetitive Netze und k-means Clusteranalyse
- ART-Netzwerke
- Verwandte Verfahren: Lernende Vektorquantisierung (LVQ)

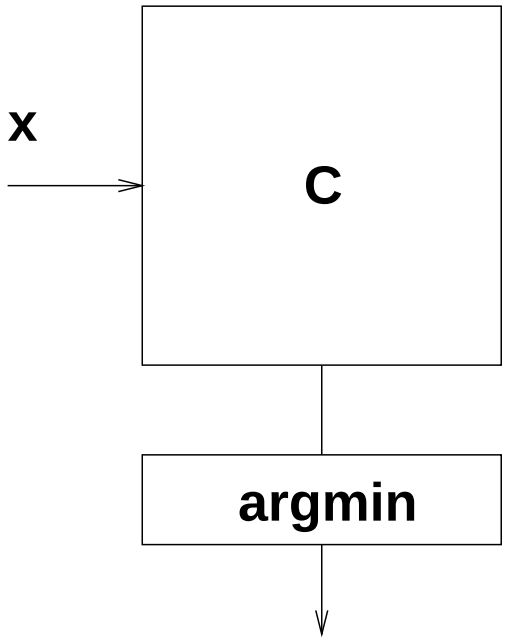
Zielsetzung

- Es soll eine Repräsentation der Eingabedaten bestimmt werden.
- Nicht Einzeldaten, sondern prototypische Repräsentationen sollen berechnet werden (Datenreduktion).
- Ähnliche Daten sollen durch einen Prototypen repäsentiert werden.
- Regionen mit hoher Datendichte sollen duch mehrere Prototypen repräsentiert sein.
- Durch spezielle neuronale Netze sollen sich Kartenstrukturen ergeben, d.h. benachbarte Neuronen (Prototypen) auf der Karte sollen durch ähnliche Eingaben aktiviert werden.

Kompetitives neuronales Netz



$$j = \underset{k}{\operatorname{argmin}} \; \| x - c_k \|$$



Distanz vs Skalarprodukt zur Gewinnerdetektion

Die Euklidische Norm ist durch das Skalarprodukt im $\mathbb{R}^n$ definiert:

$$\|x\|_2 = \sqrt{\langle x, x \rangle}$$

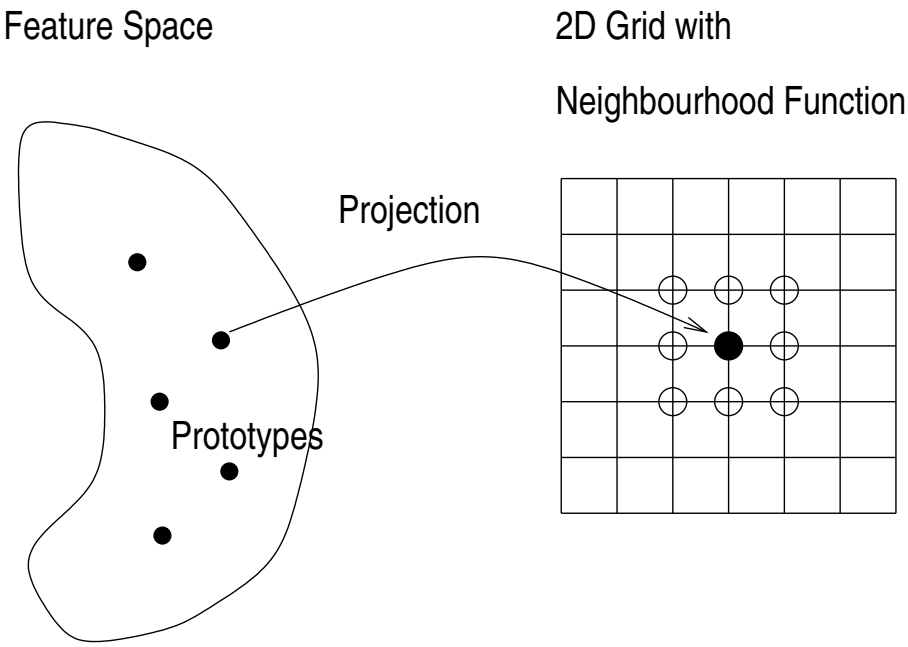
Für den Abstand zweier Punkte $x, y \in \mathbb{R}^n$ gilt demnach:

$$\|x - y\|_2^2 = \langle x - y, x - y \rangle = \langle x, x \rangle - 2\langle x, y \rangle + \langle y, y \rangle = \|x\|_2^2 - 2\langle x, y \rangle + \|y\|_2^2$$

Für die Gewinnersuche bei der Eingabe x unter den Neuronen, gegeben durch Gewichtsvektoren $c_1, \ldots, c_k$ und $\|c_i\|_2 = 1$ mit $i = 1, \ldots, k$ gilt dann offenbar:

$$j^* = \text{argmax}_i \langle x, c_i \rangle \qquad \Longleftrightarrow \qquad j^* = \text{argmin}_i \|x - c_i\|_2$$

Kohonen's Selbstorganisierende Karte



Zielsetzung beim SOM-Lernen

- Berechnung von Prototypen im Merkmalsraum, also $c_i \in \mathbb{R}^d$, die die gegebenen Daten $x \in \mathbb{R}^d$ gut repräsentieren.
- Nachbarschaftserhaltende Abbildung der Prototypen $c_i \in \mathbb{R}^d$ auf Gitterpositionen g_i auf ein Gitter im $\mathbb{R}^2$ (1D oder 3D Gitter sind auch gebräuchlich).

- **Kohonen Lernregel:** $\Delta c_j = l_t \cdot \mathcal{N}_t(g_j, g_{j^*}) \cdot (x - c_j)$
- j^* ist der Gewinner
- $\mathcal{N}_t(g_j, g_{j^*})$ eine Nachbarschaftsfunktion.
- g_j die Gitterposition des j-ten Neurons
- l_t und σ_t trainingszeitabhängige Lernraten bzw. Nachbarschaftsweitenparameter, die bei wachsender Trainingszeit gegen 0 konvergieren.
- Beispiel: $\mathcal{N}_t(g_j, g_{j^*}) = exp(-\|g_j - g_{j^*}\|^2/2\sigma_t^2)$

Kohonen Lernalgorithmus

Input: $X = \{x_1, \ldots, x_M\} \subset \mathbb{R}^d$

1. Wähle $r, s \in \mathbb{N}$, eine Clusterzahl $k = rs \in \mathbb{N}$, eine Lernrate $l_t > 0$, eine Nachbarschaftsfunktion $\mathcal{N}$ und max. Lernepochenzahl N. Setze $t = 0$.
2. Initialisiere Prototypen $c_1, \ldots, c_k \in \mathbb{R}^d$ ($k \times d$ Matrix C)
3. Jeder Prototypen c_i auf genau eine Gitterposition $g_i \in \{1, \ldots, r\} \times \{1, \ldots, s\}$.
4. **repeat**

   Wähle $x \in X$ und $t = t + 1$
   $j^* = \text{argmin}_i \|x - c_i\|$ (winner detection)
   for all j:
   $c_j = c_j + l_t \mathcal{N}_t(g_j, g_{j^*})(x - c_j)$ (update)
5. **until** $t \geq N$

SOM-Beispiele

Table 3.4. Animal names and their attributes

		dove	hen	duck	goose	owl	hawk	eagle	fox	dog	wolf	cat	tiger	lion	horse	zebra	cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

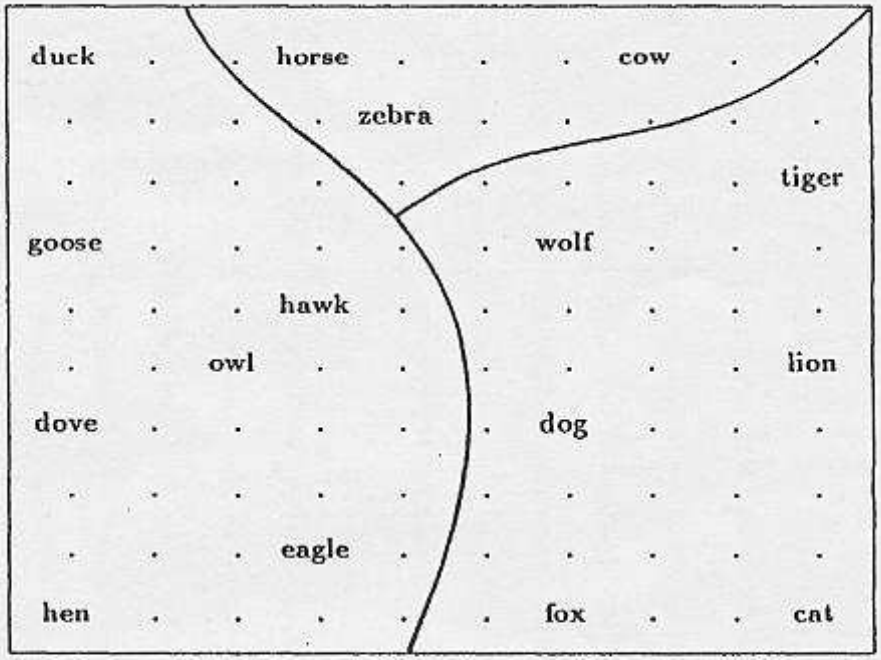
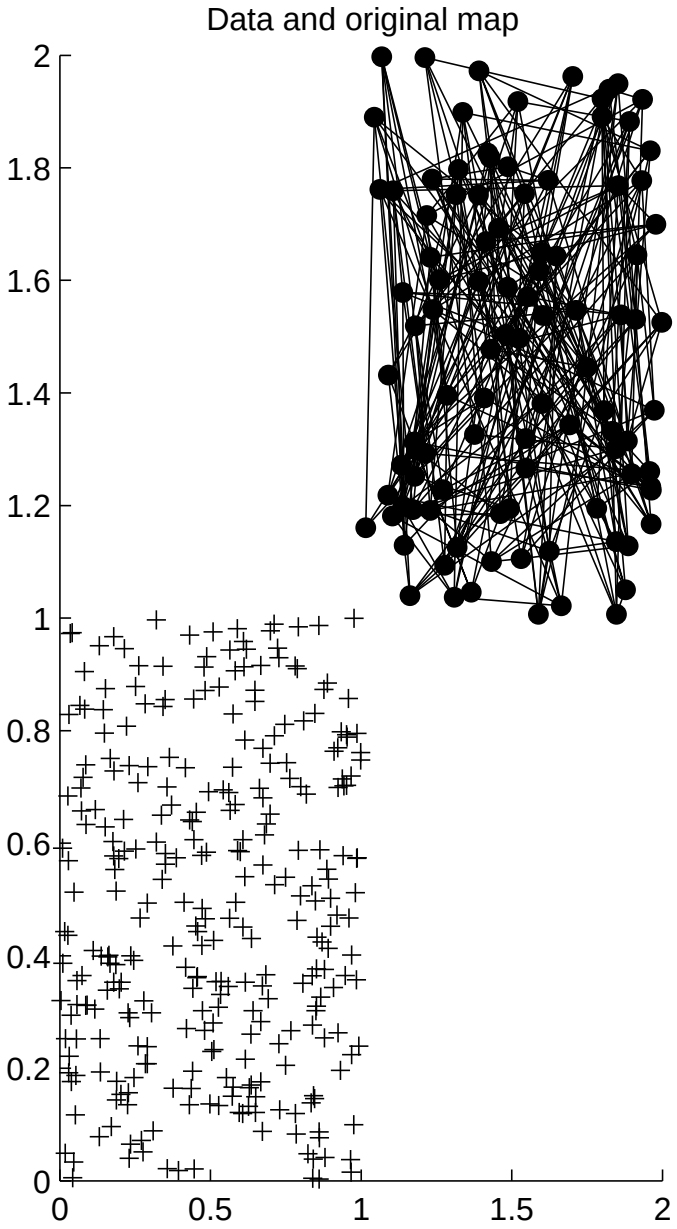
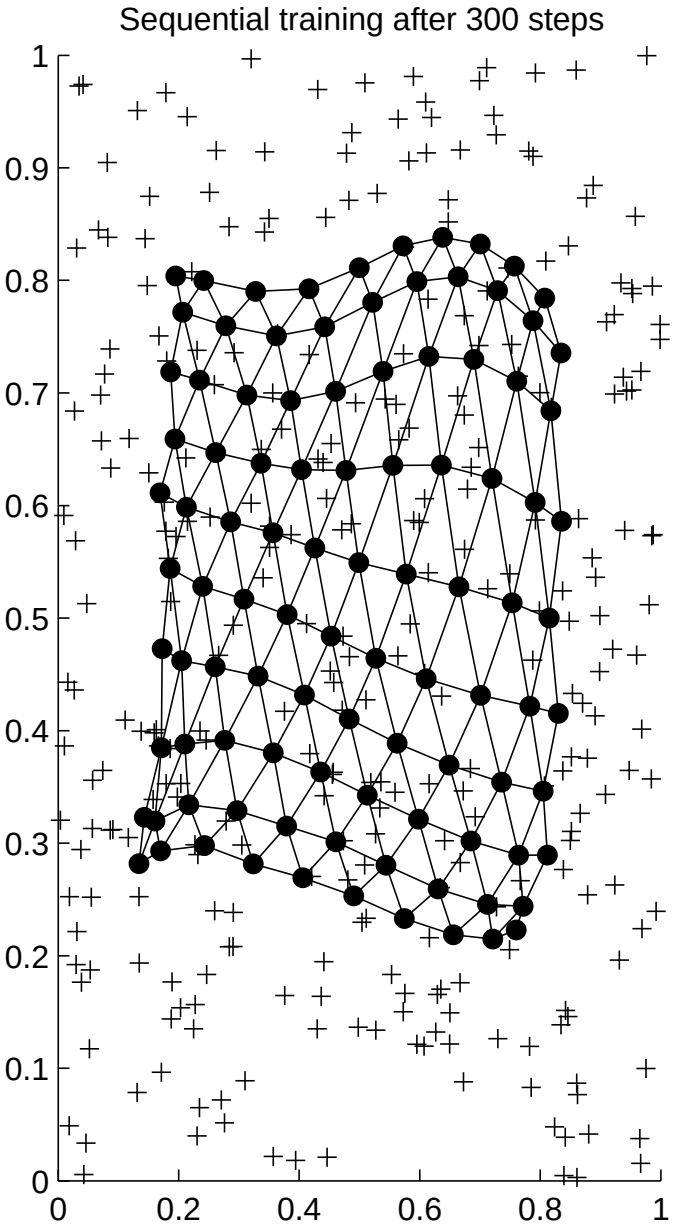


Fig. 3.22. After the network had been trained with inputs describing attribute sets from Table 3.4, the map was calibrated by the columns of Table 3.4 and labeled correspondingly. A grouping according to similarity has emerged


Data and original map
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2


Sequential training after 300 steps
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

Kompetitives Lernen (Euklidische Distanz)

Input: $X = \{x_1, \ldots, x_M\} \subset \mathbb{R}^d$

1. Wähle Clusterzahl $k \in \mathbb{N}$, eine Lernrate $l_t > 0$ und N. Setze $t = 0$.

2. Initialisiere Prototypen $c_1, \ldots, c_k \in \mathbb{R}^d$ ($k \times d$ Matrix C)

3. **repeat**

   Wähle $x \in X$ und $t = t + 1$
   $j^* = \text{argmin}_j \|x - c_j\|$ (winner detection)
   $c_{j^*} = c_{j^*} + l_t(x - c_{j^*})$ (winner update)

4. **until** $t \geq N$

k-means Clusteranalyse

Datenpunkt $x \in \mathbb{R}^d$ wird dem nächsten Clusterzentrum c_{j^*} zugeordnet:

$$j^* = \text{argmin}_j \|x - c_j\|.$$

Anpassung des Clusterzentrums:

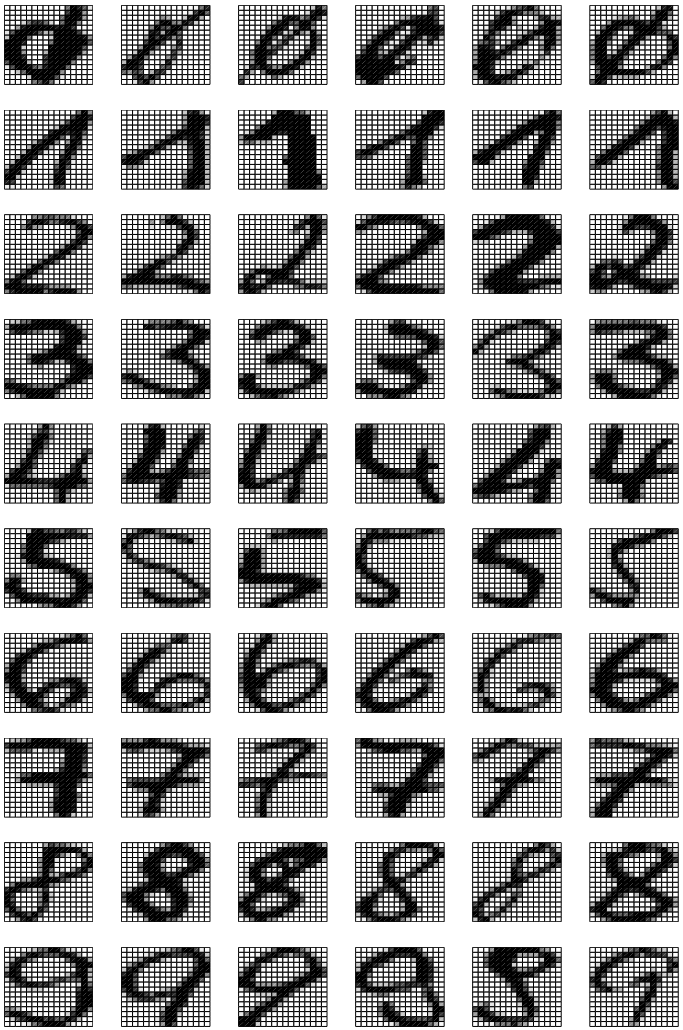
$$\Delta c_{j^*} = \frac{1}{|C_{j^*}| + 1}(x - c_{j^*})$$

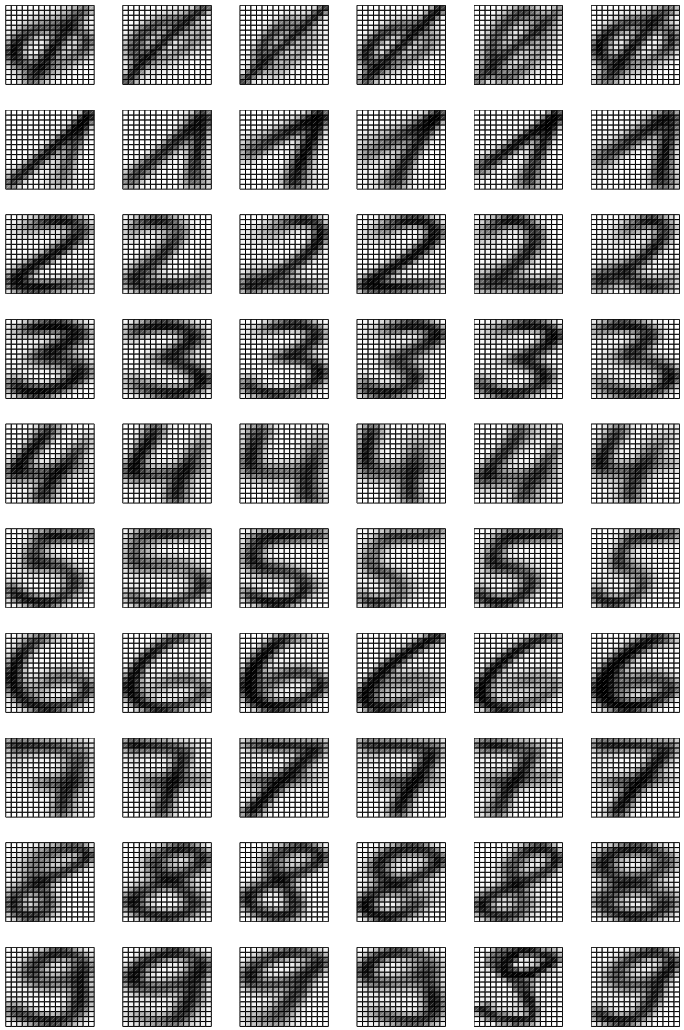
Zum Vergleich beim **Kompetitiven Lernen**

$$\Delta c_{j^*} = l_t(x - c_{j^*})$$

$l_t > 0$ eine Folge von Lernraten mit $\sum_t l_t = \infty$ und $\sum_t l_t^2 < \infty$.
Beispiel: $l_t = 1/t$.

Beispiel: Kompetitives Lernen





ART-Netzwerke

- Adaptive-Resonanz-Theorie (ART) entwickelt von Stephen Grossberg und Gail Carpenter
- Hier nur **ART1** Architektur
- Erweiterungen: ART 2, ART 3, FuzzyART, ARTMAP
- ART-Netze sind kompetitive Netze
- Besonderheit: ART-Netze sind wachsende Netze, d.h. die Zahl der Neuronen kann während des Trainings wachsen (aber nach oben beschränkt); ein Neuron kann allerdings nicht wieder gelöscht werden.
- Spezialität bei den ART 1 Netzen: Die Eingabedaten und die Gewichtsvektoren (Prototypen) sind binäre Vektoren.

ART1 Lernen : Idee

- Inputvektoren und Prototypen $\in \{0,1\}^d$.
- Es wird höchstens der Gewichtsvektor des Gewinnerneurons c_{j^*} adaptiert.
- Ist die Ähnlichkeit zwischen Inputvektor x und c_{j^*} zu gering, so definiert x einen neuen Prototypen und der Gewinnerprototyp c_{j^*} bleibt unverändert.
- Die Ähnlichkeit wird durch das Skalarprodukt $x \cdot c_j = \langle x, c_j \rangle$ gemessen.
- Schranke für die Mindestähnlichkeit wird durch den sogenannten **Vigilanz-parameter** gemessen.
- Ist die Ähnlichkeit groß genug, so wird c_{j^*} durch komponentenweises **AND** von x und c_{j^*} adaptiert.
- Maximale Zahl von Neuronen wird vorgegeben. Aus dieser Grundidee lassen sich viele mögliche Algorithmen ableiten.

ART1 : Bezeichnungen

- $x_\mu \in \{0,1\}^d$ die Eingabevektoren $\mu = 1, \ldots, M$
- $c_i \in \{0,1\}^d$ die Gewichtsvektoren der Neuronen (Prototypen)
- $\mathbf{1} = (1,1,\ldots,1) \in \{0,1\}^d$ der Eins-Vektor mit d Einsen.
- k Anzahl der maximal möglichen Neuronen.
- $\|x\|_1 = \sum_{i=1}^{d} x_i$ die l_1-Norm ($=$ Anzahl der Einsen).
- $\varrho \in [0,1]$ der Vigilanzparameter.

ART1 : Algorithmus

1. Wähle $k \in \mathbb{N}$ und $\varrho \in [0, 1]$.
2. Setze $c_i = \mathbf{1}$ für alle $i = 1, \ldots, k$.
3. **WHILE** noch ein Muster x vorhanden **DO**

   Lies x und setze $I := \{1, \ldots, k\}$

   REPEAT

   $j^* = \text{argmax}_{j \in I} \langle x, c_j \rangle / \|c_j\|_1$ (winner detection)

   $I = I \setminus \{j^*\}$

   UNTIL $I = \emptyset \vee \langle x, c_{j^*} \rangle \geq \varrho \|x\|_1$

   IF $\langle x, c_{j^*} \rangle \geq \varrho \|x\|_1$

   THEN $c_{j^*} = x \wedge c_{j^*}$ (winner update)

   ELSE keine Bearbeitung von x
4. **END**

Verwandte Verfahren : LVQ

- Lernende Vektorquantisierung (LVQ) sind **überwachte Lernverfahren zur Musterklassifikation.**
- LVQ-Verfahren sind heuristische kompetitive Lernverfahren; entwickelt von Teuvo Kohonen
- Euklidische Distanz zur Berechnung der Ähnlichkeit/Gewinnerermittlung.
- Es wird nur **LVQ1** vorgestellt.
- Erweiterungen: LVQ2 und LVQ3 (ggf. Adaptation des 2. Gewinners); OLVQ-Verfahren (neuronenspezifische Lernraten).

LVQ1 : Algorithmus

Input: $X = \{(x_1, y_1), \ldots, (x_M, y_M)\} \subset \mathbb{R}^d \times \Omega$ hierbei ist
$\Omega = \{1, \ldots, L\}$ eine endliche Menge von Klassen(-Labels).

1. Wähle Prototypenzahl $k \in \mathbb{N}$, eine Lernrate $l_t > 0$ und N. Setze $t = 0$.

2. Initialisiere Prototypen $c_1, \ldots, c_k \in \mathbb{R}^d$.

3. Bestimme für jeden Protypen c_i die Klasse $\omega_i \in \Omega$.

4. **repeat**

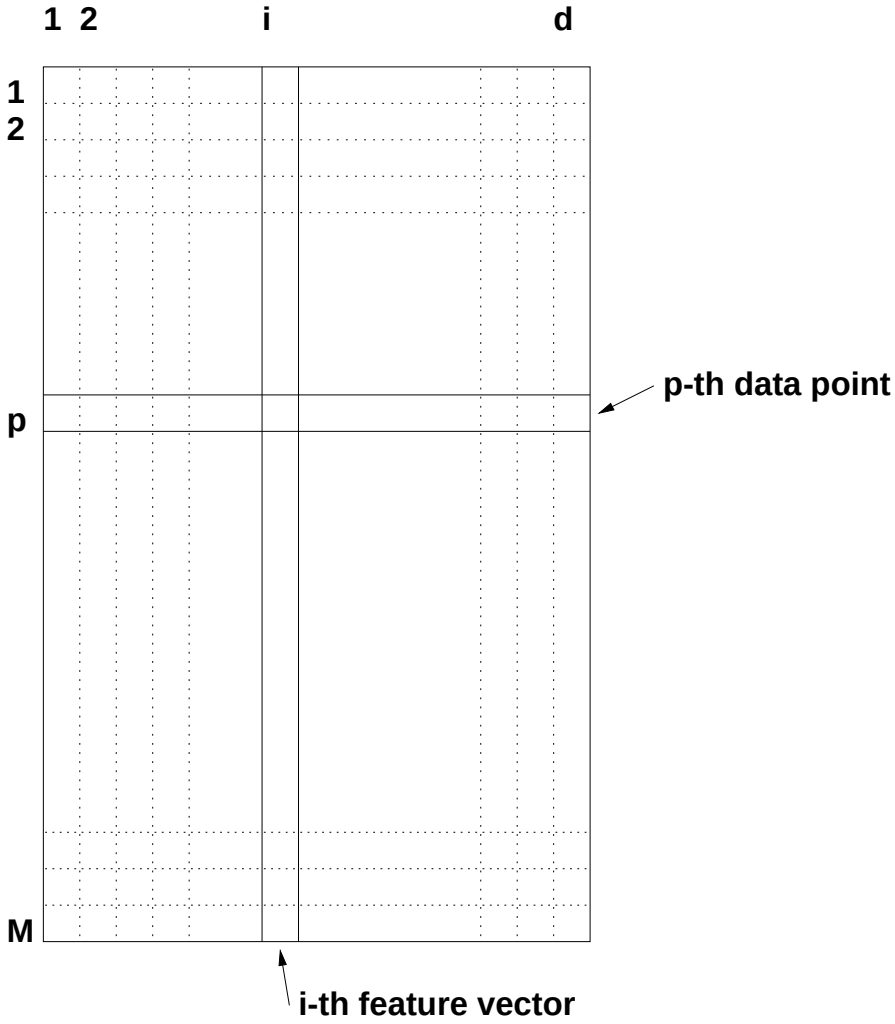
   Wähle Paar $(x, y) \in X$ und setze $t := t + 1$
   $j^* = \text{argmin}_i \|x - c_i\|$ (winner detection + class from nearest neighbor)
   if $\omega_{j^*} \neq y$ then $\Delta = -1$ else $\Delta = 1$ (correct classification result?)
   $c_{j^*} = c_{j^*} + l_t \Delta (x - c_{j^*})$ (winner update)

5. **until** $t \geq N$

8.4 Methoden zur Reduktion der Merkmale

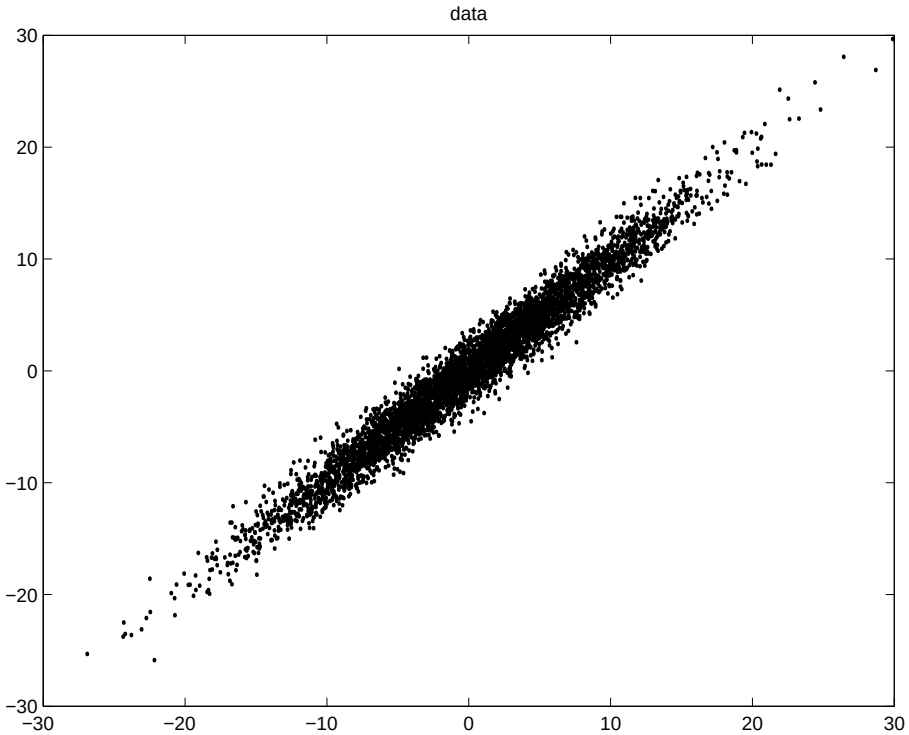
- Zielsetzung
- Hauptachsentransformation (lineare Merkmalstransformation)
- Neuronale Hauptachsentransformation (Oja und Sanger Netze)

Zielsetzung



- Einfache kompetitive Netze wie ART, adaptive k-means Clusteranalyse und auch LVQ-Netze führen eine Reduktion der Datenpunkte auf einige wenige repräsentative Prototypen (diese ergeben sich durch lineare Kombination von Datenpunkten) durch.
- Kohonen's SOM: Reduktion der Datenpunkte auf Prototypen und gleichzeitig Visualisierung der Prototypen durch nichtlineare nachbarschaftserhaltende Projektion.
- Nun gesucht Reduktion der Datenpunkte auf repräsentative Merkmale, die sich möglichst durch **lineare Kombination der vorhandenen Merkmale** ergeben.

Beispieldaten



- Variation der Daten in Richtung der beiden definierten Merkmale x_1 und x_2 ist etwa gleich groß.
- In Richtung des Vektors $v_1 = (1, 1)$ ist die Variation der Daten sehr groß.
- Orthogonal dazu, in Richtung $v_2 = (1, -1)$ ist sie dagegen deutlich geringer.

Hauptachsen

- Offensichtlich sind Merkmale in denen die Daten überhaupt nicht variieren bedeutungslos.

- Gesucht ist nun ein Orthonormalsystem $(v_i)_{i=1}^{l}$ im $\mathbb{R}^d$ mit ($l \leq d$), das die Daten mit möglichst kleinem Rekonstruktionsfehler (bzgl. der quadrierten Euklidischen Norm) bei festem ($l < d$) erklärt. Dies sind die sogenannten Hauptachsen.

- 1. Hauptachse beschreibt den Vektor $v_1 \in \mathbb{R}^d$ mit der größten Variation
  2. Hauptachse den Vektor $v_2 \in \mathbb{R}^d$ mit der zweitgrößten Variation der senkrecht auf v_1 steht.
  l. Hauptachse ist der Vektor $v_l \in \mathbb{R}^d$, v_l senkrecht auf $V_{l-1} = \text{lin}\{v_1, \ldots, v_{l-1}\}$ und die Datenpunkte x_r^μ mit $x^\mu = x_r^\mu + v$, $v \in \text{lin}\{v_l\}$ haben in Richtung v_l die stärkste Variation.

Hauptachsentransformation

- Gegeben sei eine Datenmenge mit M Punkten $x^\mu \in \mathbb{R}^d$, als $M \times d$ Datenmatrix X.
- Die einzelnen Merkmale (sind die Spaltenvektoren in der Datenmatrix X) haben den Mittelwert 0. Sonst durchführen.
- Für einen Vektor $v \in \mathbb{R}^d$ und $x^\mu \in X$ ist $\langle v, x^\mu \rangle = \sum_{i=1}^{d} v_i \cdot x_i^\mu$ die Projektion von x^μ auf v.
- Für alle Datenpunkte X ist Xv der Vektor mit den Datenprojektionen.
- Es sei nun $(v_i)_{i=1}^d$ ein Orthonormalsystem in $\mathbb{R}^d$.
- Sei nun $l < d$. Für $x \in \mathbb{R}^d$ gilt dann hierbei ist $\alpha_i = \langle x, v_i \rangle$.

$$x = \underbrace{\alpha_1 v_1 + \ldots + \alpha_l v_l}_{=:\tilde{x}} + \alpha_{l+1} v_{l+1} + \ldots + \alpha_d v_d$$

- Dann ist der Fehler zwischen x und $\tilde{x}$

$$e_l(x) := \|x - \tilde{x}\|_2^2 = \| \sum_{j=l+1}^{d} \alpha_j v_j \|_2^2$$

- Gesucht wird ein System $\{v_i\}$, so dass e_l möglichst klein ist:

$$\sum_{\mu} e_l(x^\mu) = \sum_{\mu} \Big\langle \sum_{j=l+1}^{d} \alpha_j^\mu v_j, \sum_{j=l+1}^{d} \alpha_j^\mu v_j \Big\rangle = \sum_{\mu} \sum_{j=l+1}^{d} (\alpha_j^\mu)^2 \to min$$

Dabei ist $(\alpha_j^\mu)^2$

$$(\alpha_j^\mu)^2 = \alpha_j^\mu \cdot \alpha_j^\mu = (v_j^t x^\mu)((x^\mu)^t v_j) = v_j^t (x^\mu (x^\mu)^t) v_j$$

Mitteln über alle Muster liefert:

$$\frac{1}{M}\sum_{\mu}\sum_{j=l+1}^{d} v_j^t (x^\mu (x^\mu)^t) v_j = \sum_{j=l+1}^{d} v_j^t \underbrace{\frac{1}{M}\sum_{\mu}(x^\mu (x^\mu)^t)}_{=:R} v_j$$

R ist die *Korrelationsmatrix* der Datenmenge X.

- Es ist damit folgendes Problem zu lösen:

$$\sum_{j=l+1}^{d} v_j^t R v_j \to \min$$

- Ohne Randbedingungen an die v_j ist eine Minimierung nicht möglich. Normierung der $v_j^t v_j = \|v_j\|^2 = 1$ als Nebenbedingung.

- Minimierung unter Nebenbedingungen (Siehe auch Analysis 2 (multidimensionale Analysis), speziell die Anwendungen zum Satz über implizite

Funktionen) führt auf die Minimierung der Funktion

$$\varphi(v_{l+1}, \ldots, v_d) = \sum_{j=l+1}^{d} v_j^t R v_j - \sum_{j=l+1}^{d} \lambda_j (v_j^t v_j - 1)$$

mit **Lagrange Multiplikatoren** $\lambda_j \in \mathbb{R}$.

- Differenzieren von φ nach v_j und Nullsetzen liefert:

$$\frac{\partial \varphi}{\partial v_j} = 2Rv_j - 2\lambda v_j = 0$$

- Dies führt direkt auf die Matrixgleichungen

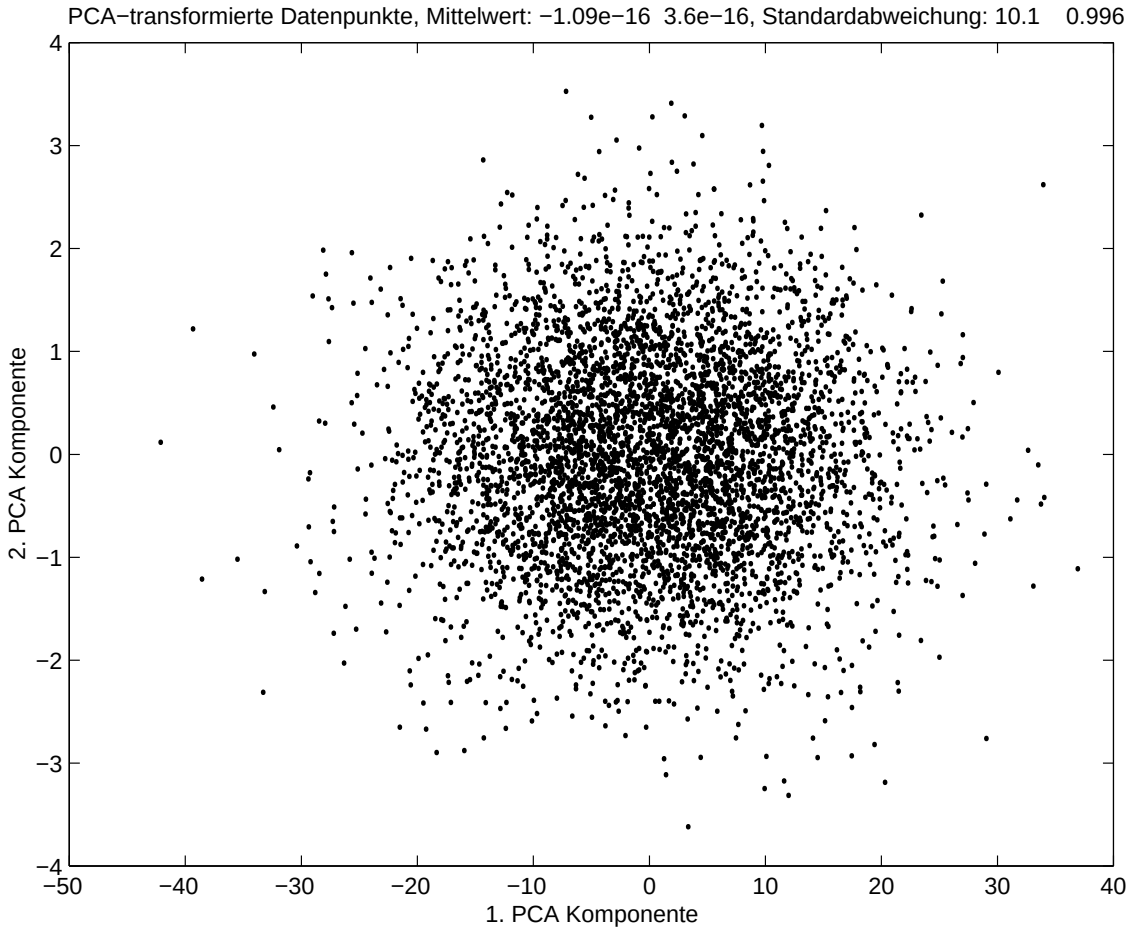
$$Rv_j = \lambda_j v_j \quad j = l+1, \ldots, d$$

- Das gesuchte System $(v_j)_{j=1}^d$ sind also die Eigenvektoren von R.

- R ist symmetrisch und nichtnegativ definit, dh. alle Eigenwerte λ_j sind reell und nichtnegativ, ferner sind die Eigenvektoren v_j orthogonal, wegen der Nebenbedingungen sogar orthonormal.

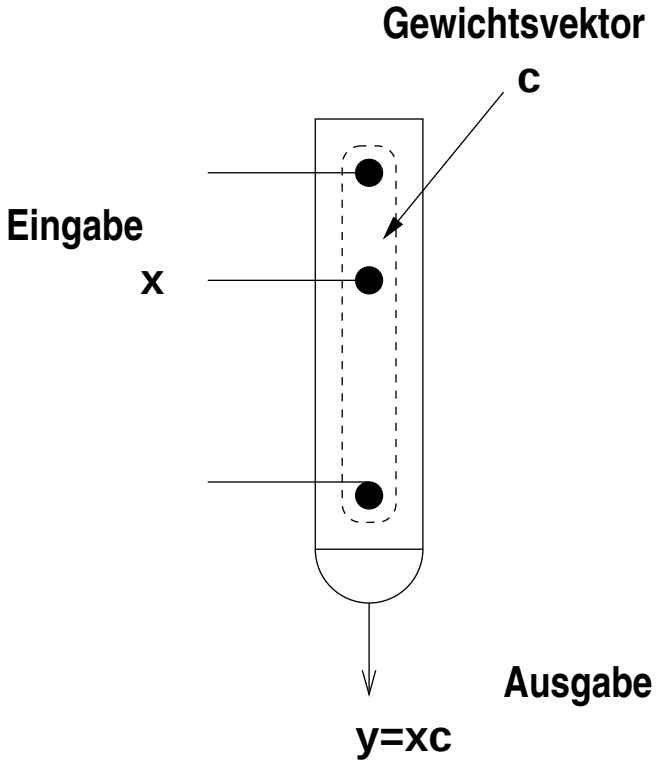
- Vorgehensweise in der Praxis:
  - Merkmale auf Mittelwert $= 0$ transformieren;
  - Kovarianzmatrix $C = X^t X$ berechnen
  - Eigenwerte $\lambda_1, \ldots, \lambda_d$ und Eigenvektoren $v_1, \ldots, v_d$ (die Hauptachsen) von C bestimmen
  - Daten X auf die $d' \leq d$ Hauptachsen projezieren, d.h. $X' = X \cdot V$ mit $V = (v_1, v_2, \ldots, v_{d'})$. Die Spalten von V sind also die Eigenvektoren v_i.
  - Dies ergibt eine Datenmatrix X' mit M Zeilen (Anzahl der Datenpunkte) und d' Merkmalen.

Hauptachsentransformierte Beispieldaten



Neuronale PCA - Oja-Lernregel

Lineares Neuonenmodell mit Oja-Lernregel



- Lineare Verrechnung der Eingabe x und dem Gewichtsvektor c

$$y = \langle x, c \rangle = \sum_{i=1}^{n} x_i c_i$$

- Lernregel nach Oja

$$\Delta c = l_t(yx - y^2 c) = l_t y(x - yc)$$

- **Satz von Oja** (1985): Gewichtsvektor c konvergiert gegen die 1. Hauptachse v_1 (bis auf Normierung), hierbei muss gelten: $l_t \to 0$ bei $t \to \infty$, $\sum_t l_t = \infty$ und $\sum_t l_t^2 < \infty$.

Neuronale PCA - Sanger-Lernregel

- Verallgemeinerung auf $d' \leq d$ lineare Neuronen mit d' Gewichtsvektor c_j. Die Ausgabe des j-ten Neurons ist dabei:

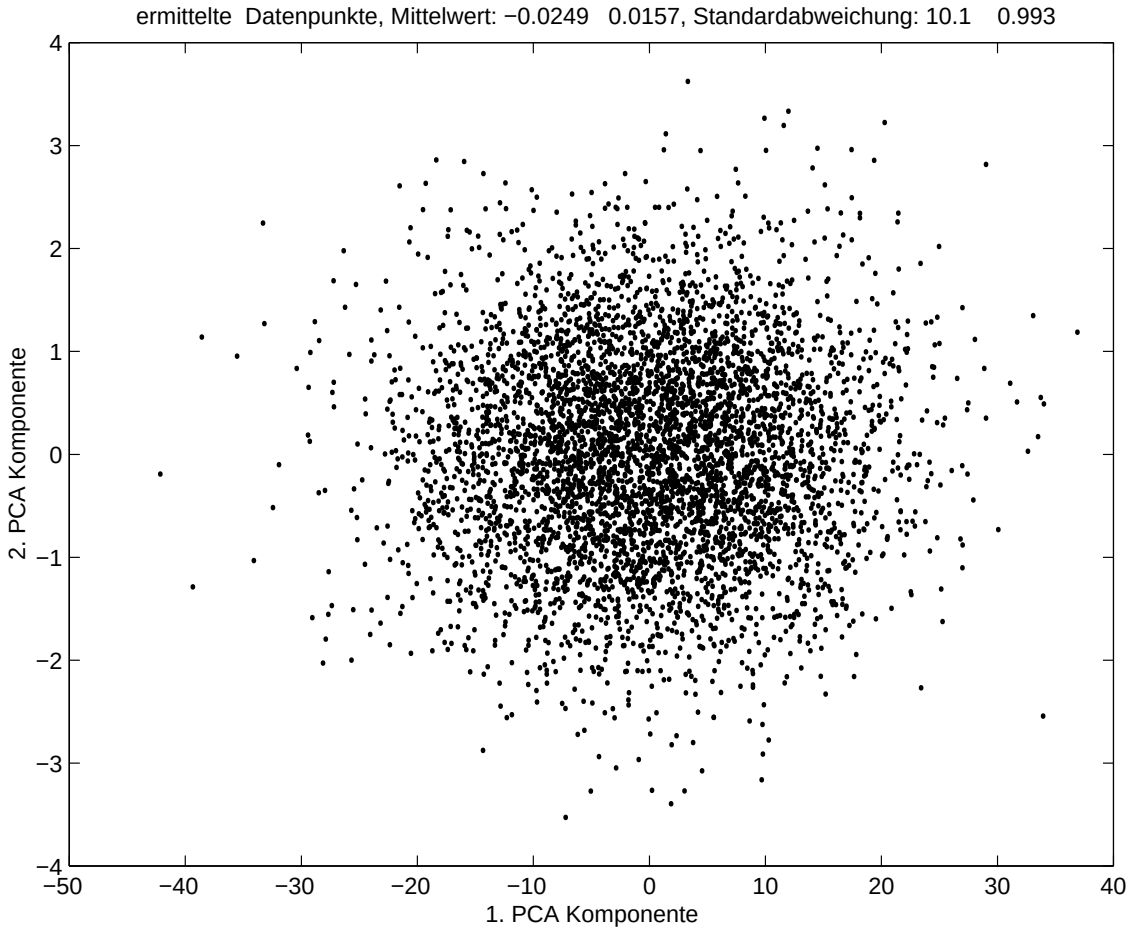
$$y_j = \langle x, c_j \rangle$$

- Lernregel nach Sanger

$$\Delta c_{ij} = l_t y_j (x_i - \sum_{k=1}^{j} y_k c_{ik})$$

- **Satz von Sanger** (1989): c_l konvergiert gegen die Hauptachsen v_l (bis auf Normierung). Es muss gelten $l_t \to 0$ bei $t \to \infty$, $\sum_t l_t = \infty$ und $\sum_t l_t^2 < \infty$.

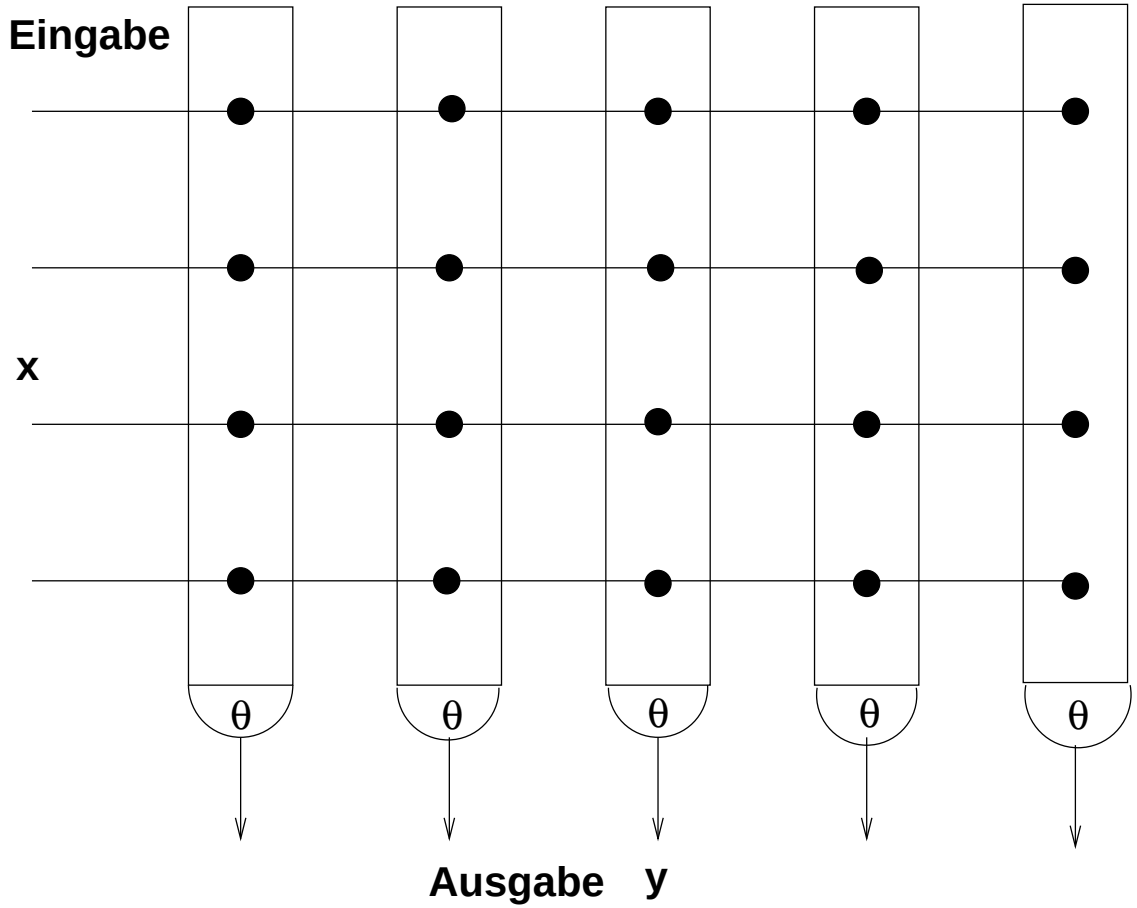
Sanger-Transformierte Beispieldaten



8. Neuronale Assoziativspeicher

1. Architektur und Musterspeicherung
2. Hetero-Assoziation
3. Auto-Assoziation
4. Bidirektionale NAMS
5. Hopfield-Netze

Architektur



Musterspeicherung

M binäre Musterpaare (x^μ, T^μ) $(\mu = 1, \ldots, M)$ werden gespeichert durch Hebb-Lernregeln

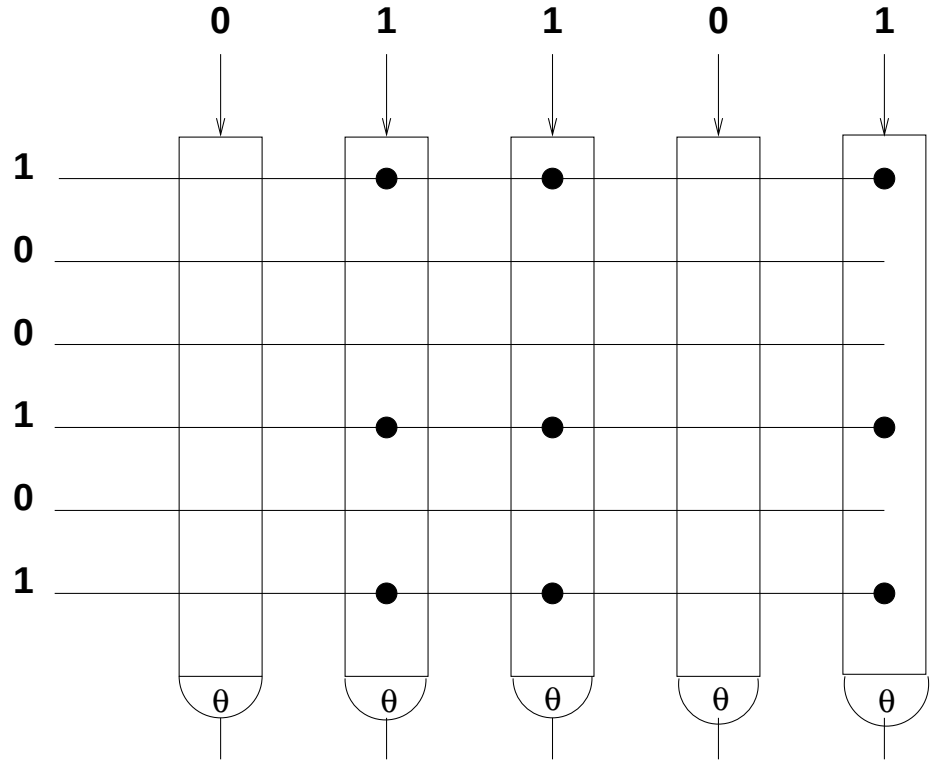
$$c_{ij} = \bigvee_{\mu=1}^{M} x_i^\mu T_j^\mu \qquad \textbf{binäre Hebbregel}$$

$$c_{ij} = \sum_{\mu=1}^{M} x_i^\mu T_j^\mu \qquad \textbf{additive Hebbregel}$$

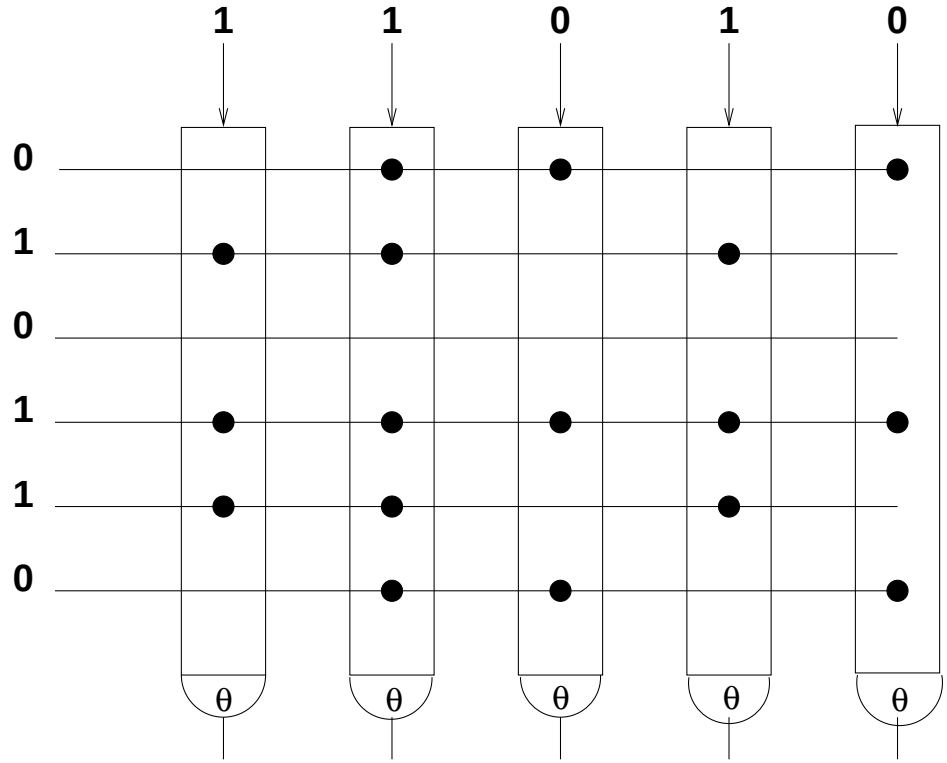
Auslesen des Antwortmusters zur Eingabe x^μ

$$y_j = \begin{cases} 1 & \sum_i x_i^\mu c_{ij} \geq \theta := |x^\mu| \\ 0 & \text{sonst} \end{cases}$$

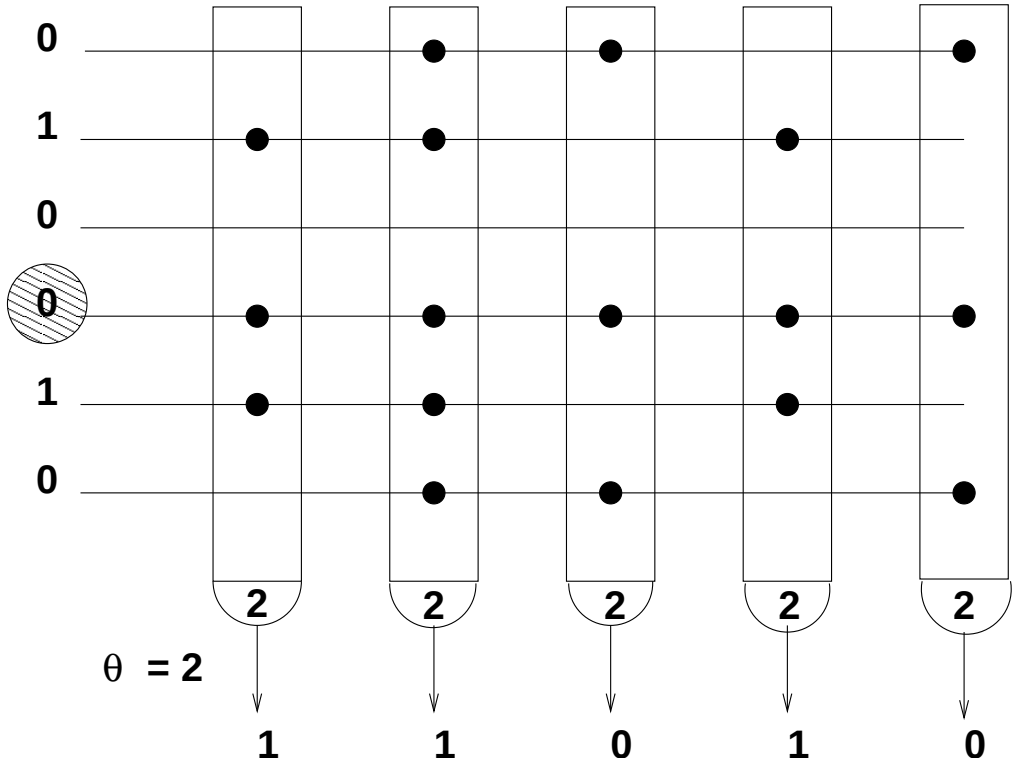
	Input x		Output y
A:	1 0 0 1 0 1	→	0 1 1 0 1
B:	0 1 0 1 1 0	→	1 1 0 1 0



	Input x		Output y
A:	1 0 0 1 0 1	→	0 1 1 0 1
B:	0 1 0 1 1 0	→	1 1 0 1 0



	Input x		Output y
A:	1 0 0 1 0 1	→	0 1 1 0 1
B:	0 1 0 1 1 0	→	1 1 0 1 0



Hetero-Assoziation

Muster $\{(x^\mu, T^\mu) \ : \ \mu = 1, \ldots, M\}$, $x^\mu \in \{0,1\}_k^m$, $T^\mu \in \{0,1\}_l^n$, $p := \frac{k}{m}$, $q = \frac{l}{n}$

Lernregel: $c_{ij} = \bigvee_{\mu=1}^{M} x_i^\mu T_j^\mu$

Retrieval: $z_j = \mathbf{1}_{[x \cdot C \geq \theta]}$ dabei ist $\theta = k$

Fehler: $f_0 = p[z_j = 0 \mid T_j = 1] = 0$ und

$$f_1 = p[z_j = 1 \mid T_j = 0] = (1 - p_0)^k \qquad \Longrightarrow pm = k = \frac{\ln f_1}{\ln(1 - p_0)} \tag{3}$$

Dabei ist

$$p_0 = p[c_{ij} = 0] = (1 - pq)^M \approx e^{-Mpq} \implies M = \frac{-\ln p_0}{pq} \tag{4}$$

Das Retrieval ist sehr genau, wenn $f_1 \leq \delta \cdot q$ mit $\delta < 1$. δ: Gütekriterium

Kapazität bei Hetero-Assoziation

Falls δ klein, dann ist die Information über das Ausgabemuster T^μ:

$$I_\mu \approx n \cdot (-q \log_2 q - (1-q) \log_2 (1-q)) \approx -nq \cdot \log_2 q$$

Relative Speicherkapazität:

$$C = \frac{M \cdot I_\mu}{m \cdot n} = \frac{-I_\mu \ln p_0}{pqmn} = \frac{\ln p_0 \ln(1-p_0)}{qn \cdot \ln f_1} nq \cdot \log_2 q$$

Maximieren nach p_0: $\Longrightarrow p_0 = \frac{1}{2}$ und damit

$$C_{max} = \frac{(\ln 2)^2 \log_2 q}{\ln q + \ln \delta} = \ln 2 \frac{\ln q}{\ln q + \ln \delta} \longrightarrow \ln 2$$

Der Limes wird erreicht für $q \to 0$ und $\delta \to 0$, aber δ langsamer, so dass $\frac{\ln \delta}{\ln q} \to 0$.

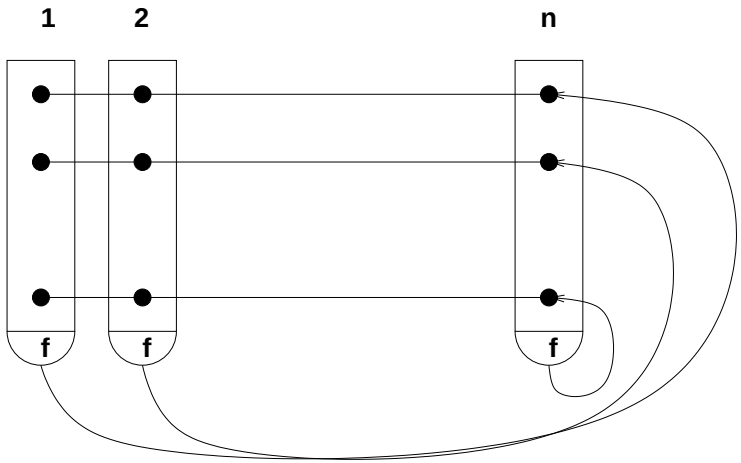
Für große Matrizen ist $C \approx \ln 2$ bei kleinem $\delta > 0$ erreichbar, wenn q sehr klein gewählt wird. $\Longrightarrow$ Spärlichkeit.

Autoassoziativer Speicher

Hetero-Assoziation: $x \neq T$ (pattern mapping)

Auto-Assoziation: $x = T$ (pattern completion)

Für Auto-Assoziation iteratives Retrieval durch Rückkopplung der Netzausgabe:



Speichern und Retrieval

Muster $\{(x^\mu) \ : \ \mu = 1, \ldots, M\}$, $x^\mu \in \{0,1\}_k^m$, , $p := \frac{k}{m}$.

Lernregel: $c_{ij} = \bigvee_{\mu=1}^{M} x_i^\mu x_j^\mu$

Retrieval: $z_j^{t+1} = \mathbf{1}_{[z^t . C \geq \theta^t]}$ dabei $\theta = |z^t|$, $t = 1$ und $\theta^t = k$

Abbruch: $z^t \subset z^{t+1}$ für $t > 1$

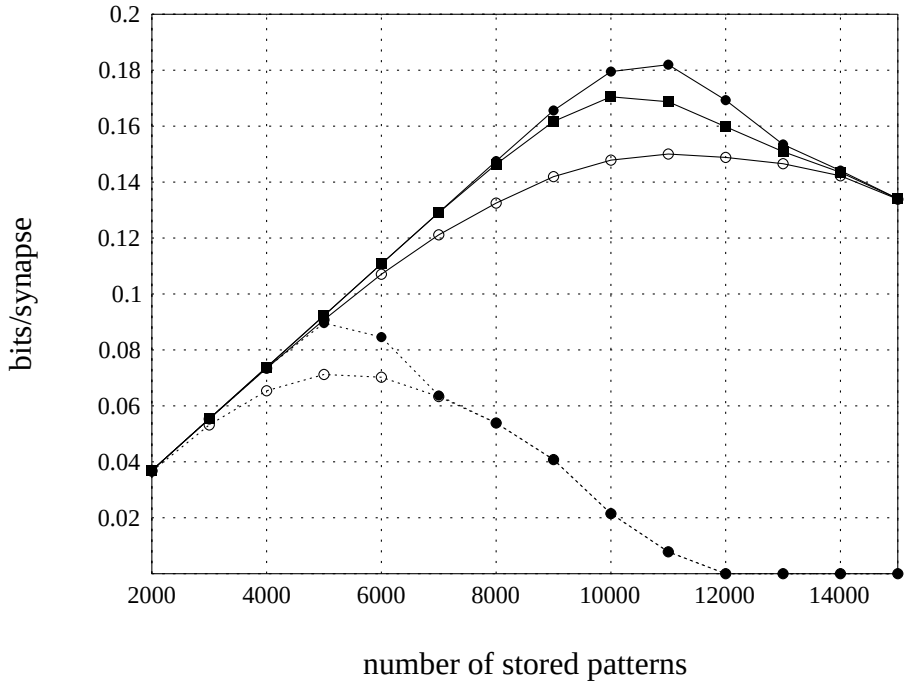
Fehler: $f_0 = p[z_j = 0 \mid x_j = 1] = 0$ und

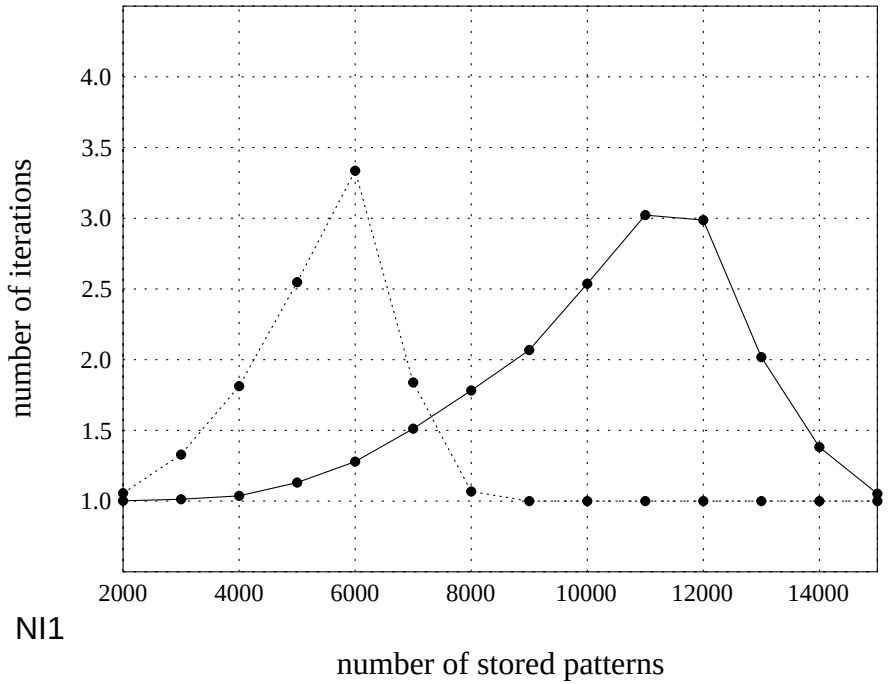
$$f_1 = p[z_j = 1 \mid T_j = 0] = (1 - p_0)^k \qquad \Longrightarrow pm = k = \frac{\ln f_1}{\ln(1 - p_0)} \tag{5}$$

Dabei ist

$$p_0 = p[c_{ij} = 0] = (1 - p^2)^M \approx e^{-Mp^2} \implies M = \frac{-\ln p_0}{p^2} \tag{6}$$

Das Retrieval ist sehr genau, wenn $f_1 \leq \delta \cdot p$ mit $\delta < 1$. δ: Gütekriterium



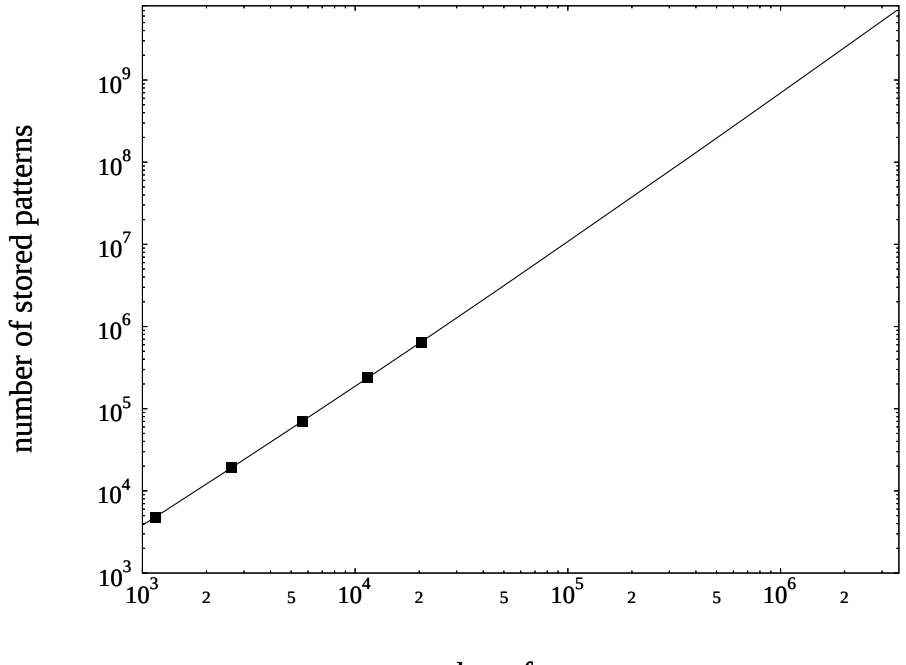


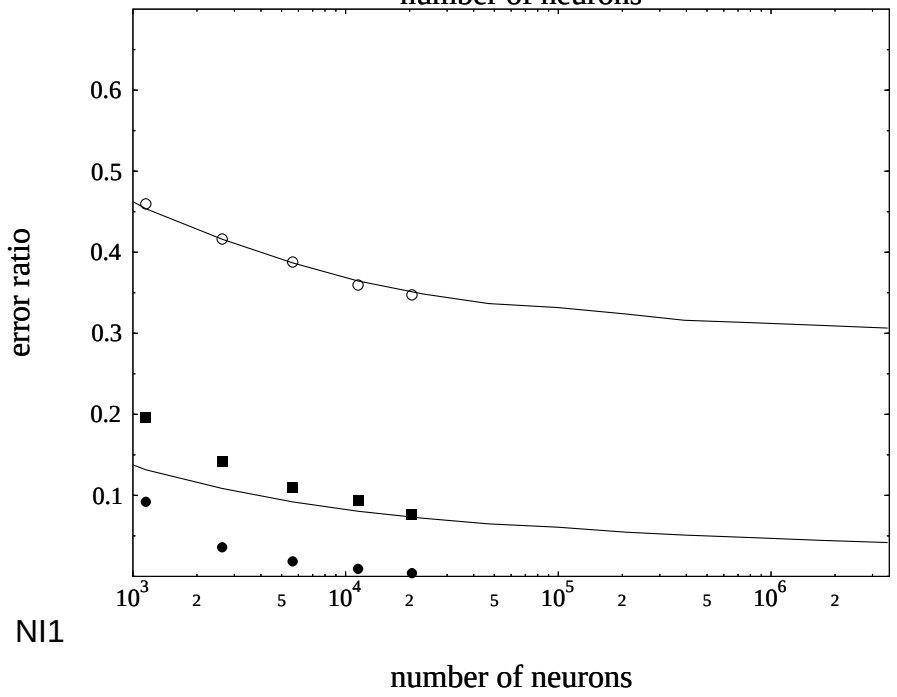
Speicherkapazität und mittlere Iterationszeit

- Autoassoziation mit n=2048 Neuronen
- Additive (unterbrochene Linie) und binäre (durchgezogene Linie) Hebb-Regel.
- 1-Schritt- (○), 2-Schritt- (■), und Fixpunkt-Retrieval (●)

Resultate

- Höhere Speicherkapazität mit binärer Hebbregel !
- Nur wenige Iterationsschritte (≈ 3) notwendig.





Fehlerwahrscheinlichkeit und Musterzahl

- Näherungsrechung für 1-Schritt- und 2-Schritt-Retrieval (durchgezogene Linien)
- Experimentelle Ergebnisse für
  1-Schritt- (○),
  2-Schritt- (■)
  Fixpunkt-Retrieval (●)

  Resultate
- Fehlerwahrscheinlichkeit ist klein (für große Netze $\rightarrow 0$)
- Auslesegüte wird durch iteratives Retrieval verbessert

Bidirektionaler Assoziativspeicher

Muster $\{(x^\mu, T^\mu) \ : \ \mu = 1, \ldots, M\}$, $x^\mu \in \{0,1\}_k^m$, $p := \frac{k}{m}$, $q = \frac{l}{n}$.

Lernregel: $c_{ij} = \bigvee_{\mu=1}^{M} x_i^\mu x_j^\mu$

Retrieval:

$$z_j^{2t+1} = \mathbf{1}_{[z^{2t} \cdot C \geq \theta^{2t}]}$$

mit $\theta^1 = |z^1|$, $t = 1$ und $\theta^t = k$

$$z_j^{2t} = \mathbf{1}_{[C \cdot z^{2t-1} \geq \theta^{2t-1}]}$$

mit $\theta^t = k$

Abbruch: $z^t \subset z^{t+2}$ für $t \geq 2$

Hopfield-Netze

- Autoassoziationsnetzwerk für Muster $x^\mu \in \{-1, 1\}$, $\mu = 1, \ldots, M$
- Hopfield-Lernregel

$$c_{ij} = \sum_{\mu=1}^{M} x_i^\mu x_j^\mu \qquad \text{und} \quad c_{ii} = 0$$

- Aus der Lernregel folgt: Matrix C ist symmetrisch!
- Die Muster sind nicht spärlich kodiert, sondern

$$prob[x_i^\mu = -1] = prob[x_i^\mu = 1] = 1/2$$

- Iteratives Retrieval mit asynchronem Neuronen-Update:

x^0 sei das Startmuster

Wähle ein Neuron und berechne seinen neuen Zustand!

$$x_i^{t+1} = \text{sgn}\left(x^t \cdot C\right)$$

- Satz: Für jeden Startwerte x^0 gibt es einen Fixpunkt x^* mit

$$x^* = \lim_{t \to \infty} x^t$$

- Beweis: Das Funktional (Lijapunov-Funktion) $H(C) = \sum_i \sum_j x_i c_{ij} x_j$ ist monoton fallend (und nach unten beschränkt).

- Hopfield-Satz gilt für allgemeine symmetrische Matrizen C mit $c_{ii} \geq 0$.

- Die Fixpunkte x^* können nicht gelernte Muster sein!

9. KNN in der Praxis

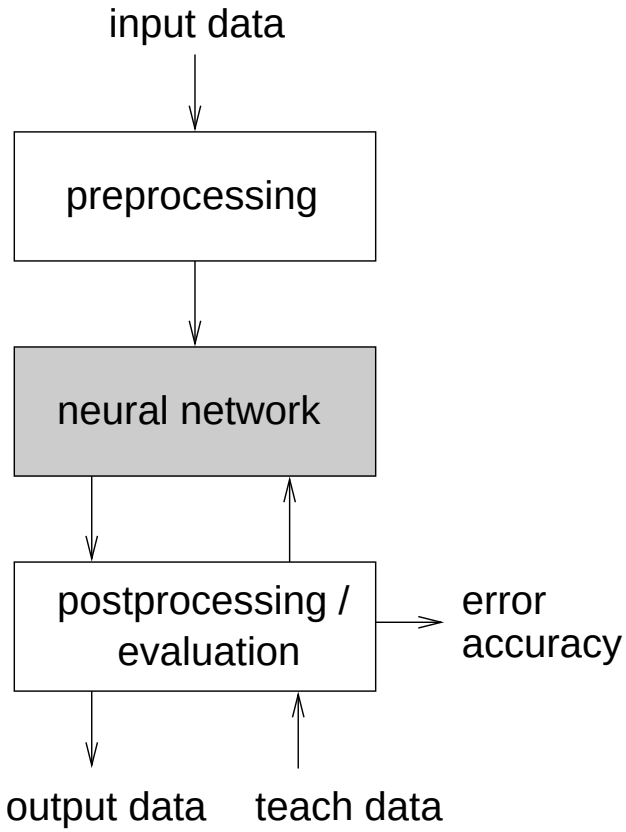
Integration eines KNN in eine Anwendung:

Schritte der **Vorverarbeitung** (Auswahl):

A) problemspezifische Transformationen
B) lineare Transformationen
C) Berücksichtigung von Invarianzen
D) Dimensionsreduktion
E) Beseitigung von Mängeln in Daten

Schritte der **Auswertung** (Auswahl):

A) Bestimmung des Fehlers
B) Bestimmung der Klassifikationsrate
C) Aufstellen einer Verwechslungsmatrix



9.1 Vorverarbeitung

A) **problemspezifische Transformationen:**

- Kodierung, z.B. Umwandlung abstrakter, symbolischer oder unscharfer Eingaben in Vektoren aus binären bzw. reellen Zahlen
- Segmentierung kontinuierlicher Signale

B) **lineare Transformationen:**

- Normierung:

$$\bar{u}_i = \frac{1}{P} \cdot \sum_{\mu=1}^{P} u_i^{(\mu)} \quad , \quad \sigma_i^2 = \frac{1}{P-1} \cdot \sum_{\mu=1}^{P} (u_i^{(\mu)} - \bar{u}_i)^2 \quad \Rightarrow \quad \tilde{u}_i^{(\mu)} = \frac{u_i^{(\mu)} - \bar{u}_i}{\sigma_i}$$

- sinnvoll i.a. bei unterschiedlichen Wertebereichen in den Dimensionen
- symmetrische Wertebereiche sind für viele neuronale Netzmodelle vorteilhaft!

C) **Berücksichtigung von Invarianzen:**

- Netzausgabe soll *invariant* sein bzgl. räumlicher oder zeitlicher Verschiebung, Skalierung, Rotation der Eingangsdaten

- Lösungsmöglichkeiten:

  1) Berücksichtigung im Trainingsdatensatz, z.B. durch Replikation jedes Musters in vielen Varianten (ggf. auch künstlich erzeugt)

  $\Rightarrow$ Trainingsdatensatz kann sehr groß werden!

  2) Vorverarbeitung durch geeignete Transformationen, z.B. Fourier-Transformation

  3) Extraktion invarianter Merkmale, z.B. durch Bestimmung von *Momenten* (statistische Parameter, die aus den Daten extrahiert werden)

  Beispiel: Man erhält translationsinvariante Merkmale aus einem Bild $B(x, y)$ durch Berechnung der Zentral-Momente $\mu_{pq} = \sum_x \sum_y (x - x_c)^p (y - y_c)^q B(x, y)$

  mit Schwerpunkt: $x_c = m_{10}/m_{00}, \;\; y_c = m_{01}/m_{00}$

  und Momenten $(p + q)$-ter Ordnung: $m_{pq} = \sum_x \sum_y x^p y^q B(x, y)$

  $\Rightarrow$ hoher Rechenaufwand erforderlich!

D) **Dimensionsreduktion:**

- Fortlassen irrelevanter Datendimensionen
- Merkmalkombination: Zusammenfassen mehrerer Datendimensionen
  z.B. durch eine geeignete Linearkombination
- Merkmalselektion:

  1) Festlegen eines Auswahlkriteriums

  z.B. ist ein Merkmal oder eine Menge von Merkmalen besser, wenn nach Training eines neuronalen Klassifikators eine geringere Fehlklassifikationsrate erreicht wird

  2) Algorithmische Suche nach einer guten Untermenge von k aus d Merkmalen:

  z.B. durch vollständige Suche, Heuristiken, *Branch & Bound*, . . .

  (insgesamt $\frac{d!}{(d-k)!k!}$ Möglichkeiten)

  $\Rightarrow$ sehr hoher Rechenaufwand!
- Hauptachsentransformation (PCA), ggf. auch neuronal mit Lernregel nach Sanger

E) **Beseitigung von Mängeln in Daten**

- *Ausreißer*

  1) Entfernen aus Datensatz
  (da sie einen zu hohen Einfluss auf Gewichte beim Training haben)

  2) Verwendung anderer Fehlerfunktionen ($\Rightarrow$ andere Lernregel)
  (z.B. $E_R = \sum_\mu ||\mathbf{y} - \mathbf{t}||^R$ mit $R < 2$)

- *Rauschen* oder andere *Störungen*

  1) lokale Glättungsoperationen (z.B. Mittelwert-, Median-, oder Gauß-Filter)

  2) frequenzabhängiges Filter (z.B. Bandpass, Gabor-Filter)

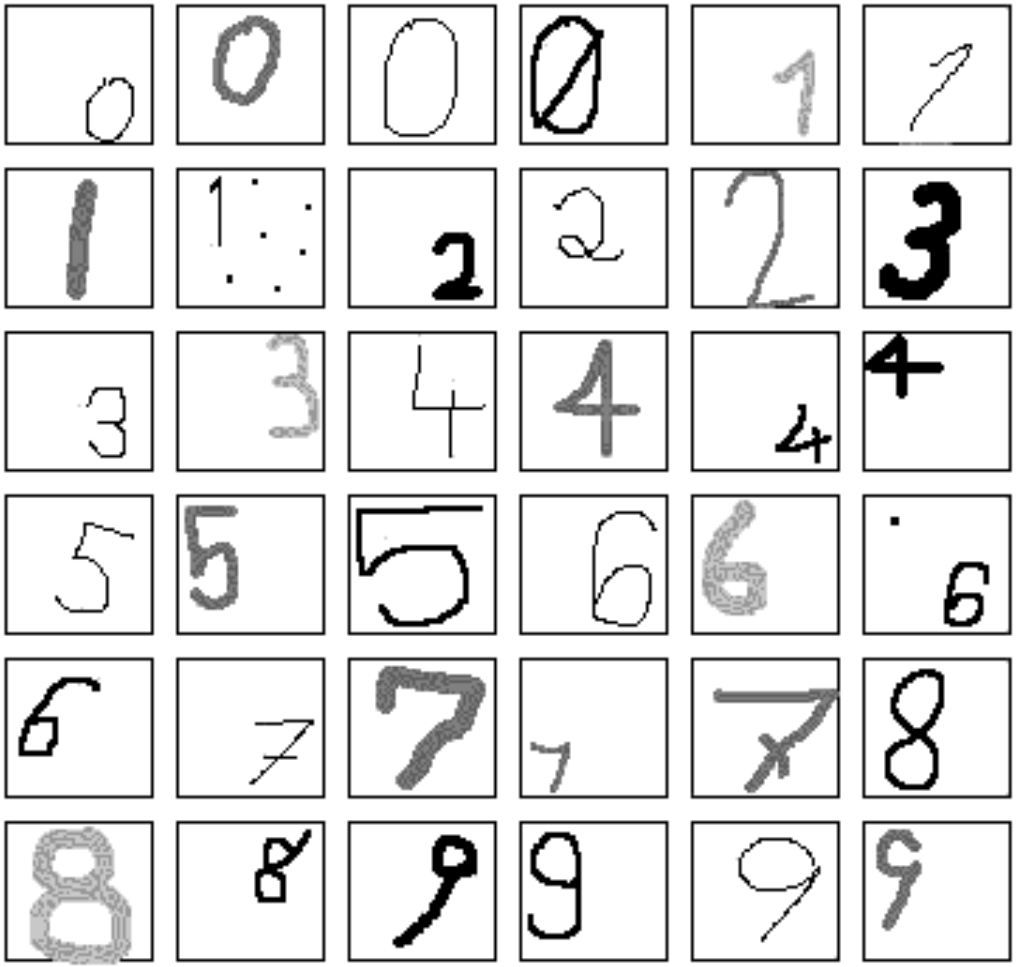
- *Fehlender Wert* in i-ter Dimension bei einem Musterverktor $\mathbf{u}^{(\nu)}$;

  1) Entfernen von Muster $\mathbf{u}^{(\nu)}$ aus Datensatz

  2) Einsetzen von Mittelwert $u_i^{(\nu)} = \frac{1}{P-1} \cdot \sum_{\mu \neq \nu} \; u_i^{(\mu)}$

  3) Ermitteln von $u_i^{(\nu)}$ durch Interpolation

Beispiel 1: Erkennung handgeschriebener Ziffern



Schritte der Vorverarbeitung:

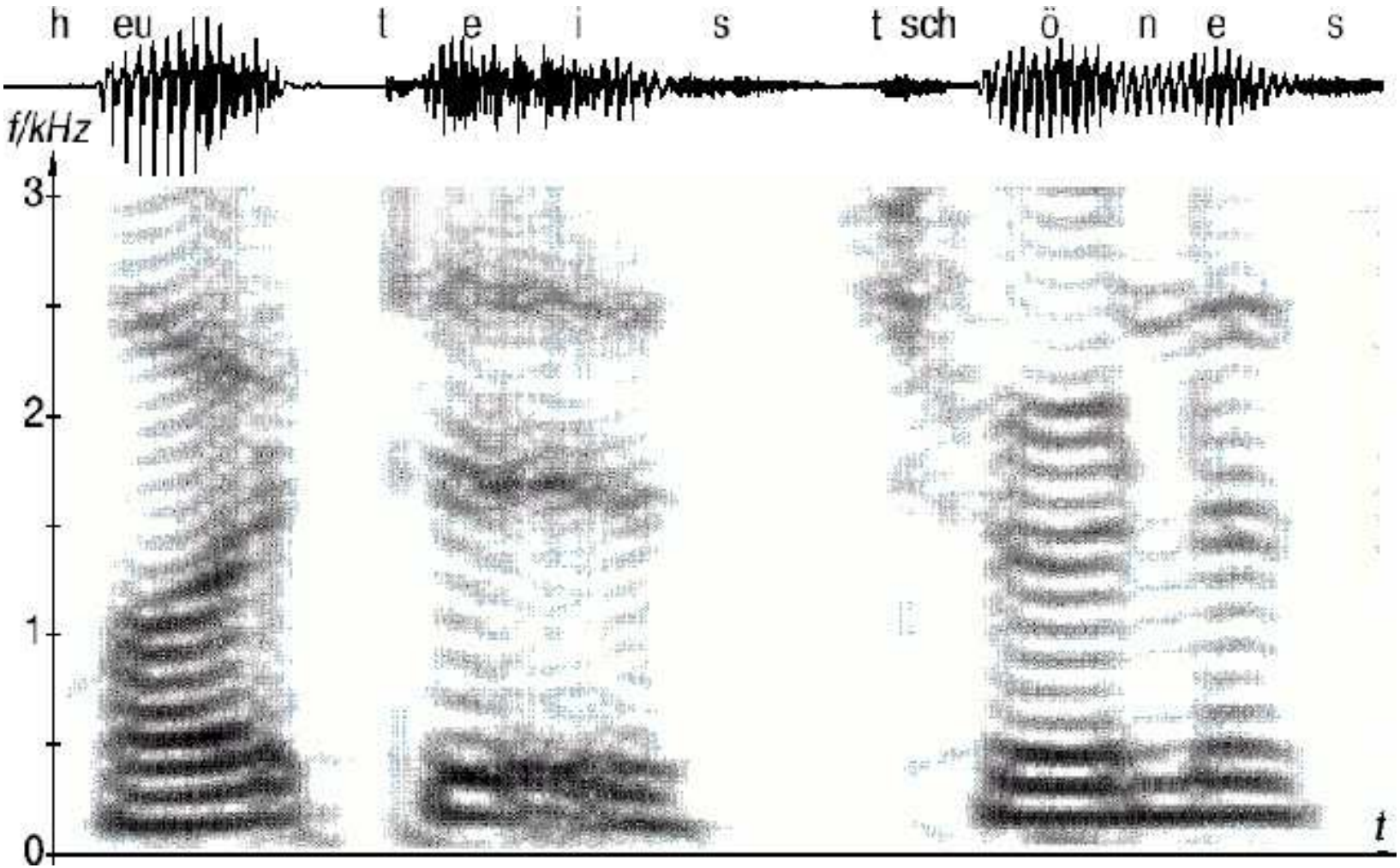
...

Beispiel 2: Verifikation von Fingerabdrücken

Schritte der Vorverarbeitung:

...

Beispiel 3: Sprecher-Verifikation/Identifikation



(aus W. Hess, Grundlagen der Sprachsignalverarbeitung)

Schritte der Vorverarbeitung:

- **problemspezifische Schritte:**
  Abtastung und Quantisierung, Segmentierung des Signals, Einteilung in überlappende Fenster von ca. 10ms bis 30ms Länge unter Anwendung einer Hamming-Fensterfunktion
- **Beseitigung von Störungen:**
  Filtern (z.B. Bandpass) zur Unterdrückung von Rauschen oder Hintergrundgeräuschen
- **Extraktion zeitinvarianter Merkmale:**
  1. Bestimmung der Signalenergie in einem Fenster
  2. Berechnung der Kurzzeitspektren mittels diskreter Fourier-Transformation, Bestimmung der Grund- und Formantfrequenzen
  3. Berechnung von LPC-Koeffizienten (*"Linear Predictive Coding"*, Betrachtung eines Signalwertes $s(t) = \sum_k w_k \cdot s(t-k)$ als Linearkombination aus früheren Signalwerten)

$\Rightarrow$ Praktikum im folgenden Semester!

9.2 Auswertung

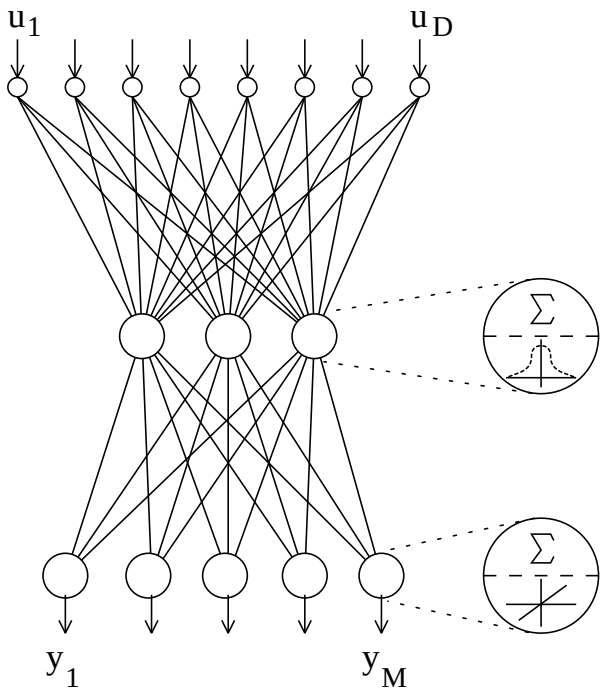
A) Bestimmung des Fehlers:

- bisher nur SSE (*Sum of Squared Errors*): $E_{\text{SSE}} = \sum_{\mu=1}^{P} (\mathbf{y}^{(\mu)} - \mathbf{t}^{(\mu)})^2$
- besser ist MSE (*Mean-Square Error*): $E_{\text{MSE}} = E/P$
- oder RMSE (*Root Mean-Square Error*): $E_{\text{RMSE}} = \sqrt{E_{\text{MSE}}}$

B) Bestimmung der Gewinnerklasse k^* aus Netzausgabe $\mathbf{y}$:

- Maximumsbestimmung: $k^* = \text{argmax}_k\{y_k\}$
- Umrechnung der Netzausgaben in Wahrscheinlichkeiten für Klassenzugehörigkeiten
- Berücksichtigung einer Toleranz TOL (*typisch* TOL=0.2 oder TOL=0.5):
  Es muß ein k^* geben mit: $y_{k^*} \geq 1 - \text{TOL}$
  und für alle $k \neq k^*$ muß gelten: $y_k < \text{TOL}$

Beispiel: Ausgabevektoren $\mathbf{y}^{(\mu)}$ für verschiedene Muster $\mathbf{u}^{(\mu)}$ nach Training eines Klassifikations-problems auf einem RBF-Netzwerk



0.1	0.2	0.6	0.1	0.4
0.1	0.2	0.1	0.1	0.15
0.0	0.2	0.9	0.1	0.25
0.7	0.2	0.1	0.1	0.1
0.5	0.0	0.0	0.1	0.4
0.2	1.0	1.0	0.1	0.0

C) Bestimmung der Klassifikationsrate ACC (*Accuracy*):

$$\text{ACC} = \frac{1}{P} \cdot \sum_{\mu=1}^{P} \alpha(\mathbf{y}^{(\mu)}, \mathbf{t}^{(\mu)})$$

mit (bei Maximumsbestimmung, $k \geq 2$)

$$\alpha(\mathbf{y}^{(\mu)}, \mathbf{t}^{(\mu)}) = \begin{cases} 1 & \text{falls } k^* = \text{argmax}_k\{y_k\} \text{ und } t_{k^*} = 1 \\ 0 & \text{sonst} \end{cases}$$

oder (bei Verwendung einer Toleranz)

$$\alpha(\mathbf{y}^{(\mu)}, \mathbf{t}^{(\mu)}) = \begin{cases} 1 & \text{falls } |t_k - y_k| < \text{TOL für alle } k \\ 0 & \text{sonst} \end{cases}$$

D) Überprüfung oder Weiterverarbeitung der Netzausgabe anhand von *Kontextwissen*

E) Aufstellen einer *Verwechslungsmatrix* V:
$V_{i,j}$ gibt an, wie oft ein Muster der Klasse i vom Klassifikator einer falschen Klasse j zugeordnet wurde; Diagonale enthält richtige Klassifikationen

Beispiel: Verwechslungsmatrix bei einer Klassifikation handgeschriebener Ziffern (1000 Trainingsmuster je Klasse)

Digit	Classified as										Total
	0	1	2	3	4	5	6	7	8	9	
0	964	2	7	7	4	1	3	1	11	0	1,000
1	0	961	1	1	3	1	7	2	23	1	1,000
2	9	6	897	17	19	2	10	14	26	0	1,000
3	4	5	6	952	1	9	3	3	12	5	1,000
4	1	7	2	1	942	0	3	2	9	33	1,000
5	5	2	4	31	2	927	13	1	10	5	1,000
6	3	10	1	1	6	2	970	0	7	0	1,000
7	4	11	2	8	2	0	0	917	5	51	1,000
8	5	13	2	16	8	16	2	5	924	9	1,000
9	3	4	0	7	9	2	0	11	15	949	1,000
Total	998	1021	992	1041	996	960	1011	956	1,042	1,053	10,000

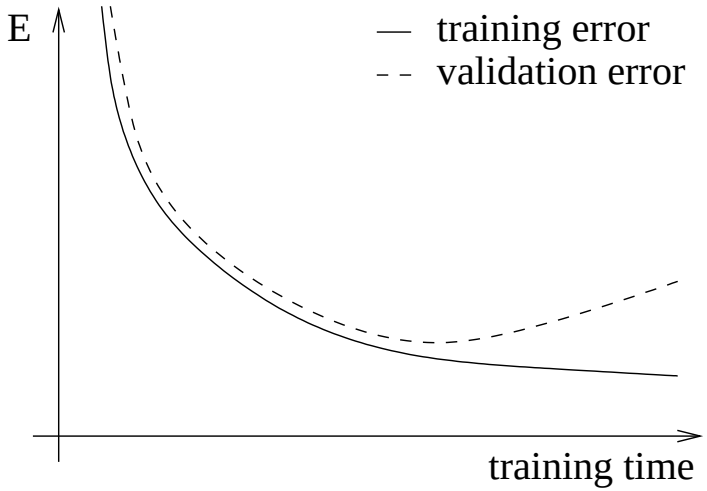
9.3 Lernen und Generalisierung

- Ziel eines neuronalen Trainings ist es nicht, die Trainingsmuster $(\mathbf{u}^{(\mu)}, \mathbf{t}^{(\mu)})$ exakt zu speichern, sondern den zugrundeliegenden Datengenerierungsprozeß möglichst gut zu modellieren.

- **Generalisierung:** Eigenschaft, neue (d.h. beim Training noch nicht verwendete) Eingabewerte $\mathbf{u}$ gut zu klassifizieren bzw. zu approximieren

  Zur Bestimmung der **Generalisierungsfähigkeit** ist Aufteilung des Datensatzes in mindestens einen Trainingsatz und mindestens einen Testsatz erforderlich

- Fehler auf Testsatz (*validation error*) ist oft größer als der Fehler auf Trainingssatz (*training error*):



Annahmen: Netzwerk mit einzelnem Ausgang y, Lernverfahren minimiert quadratischen Fehler $E_{\text{SSE}} = (y - t)^2$

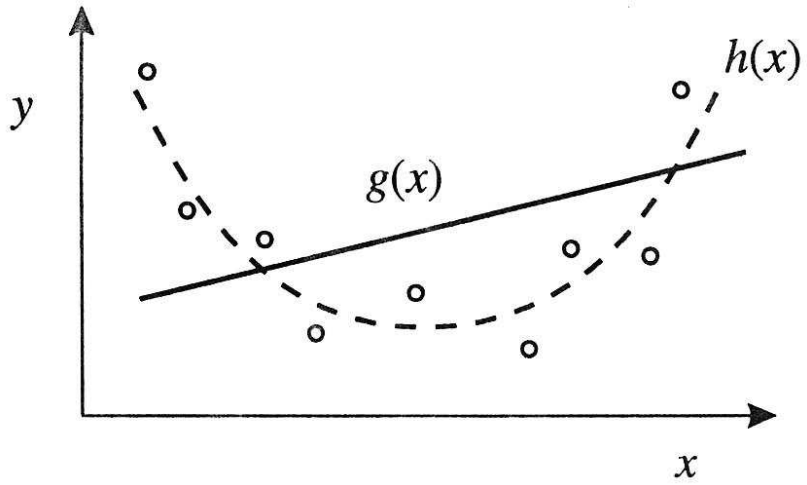
Zerlegung des Erwartungswertes von E, gemittelt über alle Datensätze D_i:

$$
\begin{aligned}
\mathcal{E}_D[E] &= \mathcal{E}_D[(y(\mathbf{u}) - t(\mathbf{u}))^2] \\
&= \mathcal{E}_D[(y(\mathbf{u}) - \mathcal{E}_D[y(\mathbf{u})] + \mathcal{E}_D[y(\mathbf{u})] - t(\mathbf{u}))^2] \\
&= \mathcal{E}_D[(y(\mathbf{u}) - \mathcal{E}_D[y(\mathbf{u})])^2 + (\mathcal{E}_D[y(\mathbf{u})] - t(\mathbf{u}))^2 \\
&\quad + \underbrace{2(y(\mathbf{u}) - \mathcal{E}_D[y(\mathbf{u})])(\mathcal{E}_D[y(\mathbf{u})] - t(\mathbf{u}))}_{\to 0}] \\
&= \mathcal{E}_D[(y(\mathbf{u}) - \mathcal{E}_D[y(\mathbf{u})])^2] + \mathcal{E}_D[(\mathcal{E}_D[y(\mathbf{u})] - t(\mathbf{u}))^2] \\
&= \underbrace{\mathcal{E}_D[(y(\mathbf{u}) - \mathcal{E}_D[y(\mathbf{u})])^2]}_{\text{Varianz}} + \underbrace{(\mathcal{E}_D[y(\mathbf{u})] - t(\mathbf{u}))^2}_{(\text{Bias})^2}
\end{aligned}
$$

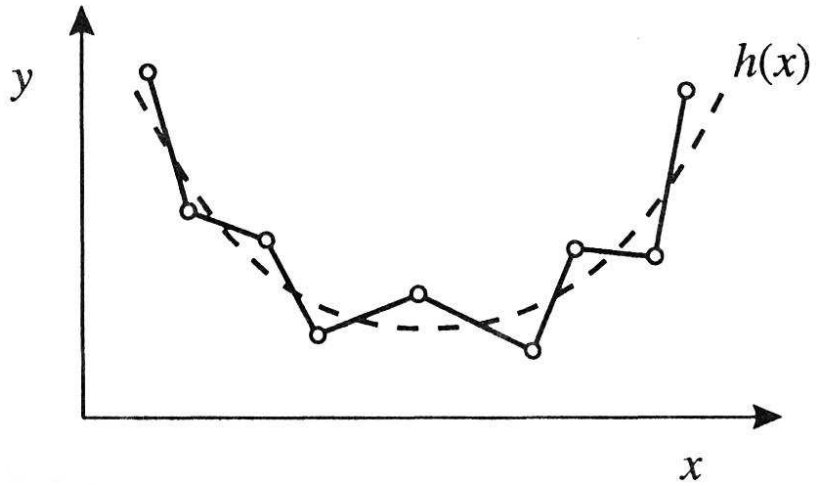
Bias : Unterschied zwischen der (über alle Datensätze) gemittelten Netzwerkausgabe und der Sollfunktion

Varianz : Sensitivität der Netzwerkausgabe bzgl. bestimmter Datensätze D_i

- **Problem:** gleichzeitige Minimierung von Bias **und** Varianz



Approximation von $h(x)$ mit großem Bias und kleiner Varianz



Approximation von $h(x)$ mit kleinem Bias und großer Varianz

- Lösungsmöglichkeiten:
  - **1)** Rechtzeitiger Trainingsstop
  - **2)** Verrauschen der Daten des Trainingsdatensatzes $\mathcal{D}$
  - **3)** Anpassung von Netzarchitektur (z.B. Anzahl verdeckter Neuronen) und $|\mathcal{D}|$
  - **4)** Wahl einer adäquaten Trainings-/Testmethode

Verschiedene Trainings-/Testmethoden:

- bisher nur **Resubstitution**:
  Trainingsdatensatz $=$ Testdatensatz
  $\Rightarrow$ *Bias klein, Varianz groß, zu optimistische Werte* E_{RMSE} *und ACC*

- **Holdout**:
  Trainingsdatensatz $\mathcal{D}/\mathcal{D}_t$: 2/3 des Datensatzes
  Testdatensatz $\mathcal{D}_t$: 1/3 des Datensatzes
  $\Rightarrow$ *Bias größer, Varianz kleiner, zu pessimistische Werte* E_{RMSE} *und ACC*

- k**-fache Kreuzvalidierung** (*Cross-Validation*):
  Datensatz $\mathcal{D}$ wird zufällig in k (ungefähr) gleich große Teile aufgeteilt, k Trainings-/Testläufe mit $i = 1, \ldots, k$:

  Training mit jeweils $k-1$ Teilen $\mathcal{D}/\mathcal{D}_t^{(i)}$, Test nur mit Teil $\mathcal{D}_t^{(i)}$

  Auswertung: $\text{ACC} = \frac{1}{k}\sum_{i=1}^{k} \text{ACC}(\mathcal{D}_t^{(i)})$, $E_{\text{RMSE}} = \frac{1}{k}\sum_{i=1}^{k} E_{\text{RMSE}}(\mathcal{D}_t^{(i)})$

  $\Rightarrow$ *realistische Werte* E_{RMSE} *und ACC möglich!*

10. Zusammenfassung

Einordnung von künstlichen Neuronalen Netzen im Kontext verwandter Disziplinen: