Künstliche Neuronale Netze

Skript zur Vorlesung
an der FH-Landshut
bei Prof. A. Harasim

WS 2002 / 03

von Reinhard Ostermeier

Inhalt

Vorbemerkung

Alle Angaben und Formeln in diesem Skript sind ohne Gewähr auf Fehler.
Eine Ausnahme stellen die Hervorgehobenen Formeln dar, welche 1:1 aus der Formelsammlung von Prof. A. Harasim übernommen wurden.

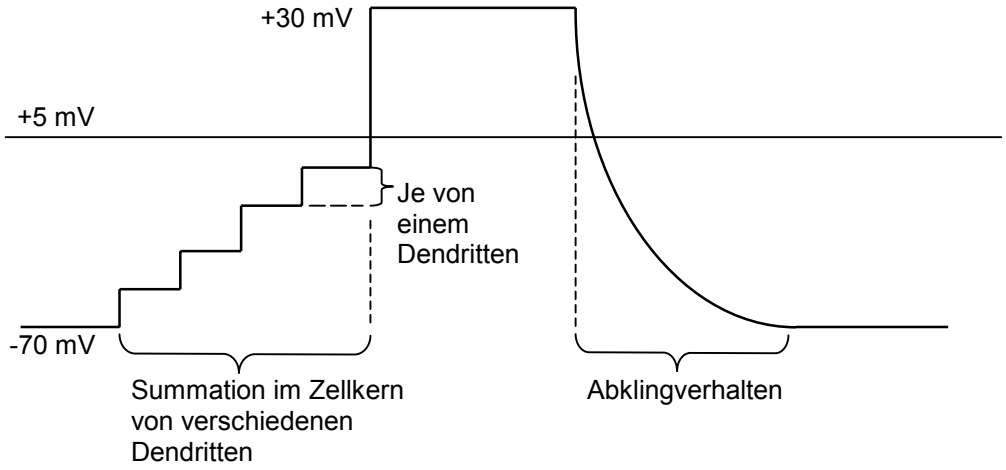
Markierung korrekter Formeln:

Formel 

1. Funktionsweise eines biologischen Neurons

Neuron: Zellkörper (Kern) mit Ausläufern.
Axon: Unverzweigte Ausläufer der Nervenzelle, die sich erst im Zielgebiet verzweigen.
Dendritten: Signalempfänger der Nervenzelle, welche sehr stark verzweigt.

Neuronen eingebettet in Natrium-, Kalium- Lösung ($^{+}$Na, $^{+}$K). Diese Ionen sind für die Signalfortpflanzung entscheidend. Es gibt keine elektrischen Leitungen im sinne von "Drahtleitungen", sondern eine Potentialausbreitung welche vom Axonhügel ausgeht. Da im Ruhezustand die Konzentration von $^{+}$Na-Ionen in den Zellen um den Faktor 10 geringer ist als Außen, entsteht ein Ruhepotential von -70 mV.



Bei Schwellüberschreitung des Summationssignals findet eine Potentialumkehr statt. $^{+}$Na-Ionen fließen in die Zelle und $^{+}$K-Ionen heraus. Es entsteht ein Potentialsprung auf +30 - +40 mV der sich entlang des Axons ausbreitet. Am Ende des Axons erfolgt die Informationsweitergabe rein chemisch über Synapsen zum Dendritten der benachbarten Nervenzelle.

Synapsen: Kontaktlose Schaltstellen zwischen Dendrit und Axon. Synaptischer Spalt beträgt ca. 20 nm. Signalvermittler sind Bläschen mit Transmitterstoffen, die sich an Ventile der Dendritten- Membran heften und sie für eine gewisse Zeit geöffnet halten. Dadurch können Calzium-Ionen von außen einströmen und einen Potentialwechsel verursachen.

Es gibt grundsätzlich hemmende und erregende Synapsen. Die hemmenden sitzen am Zellkörper, die Erregenden an den Dendritten- Dornen (siehe Bild 1.5).

Die Zahl der ausgeschütteten Transmitterstoffe hängt vom Lernzustand der Synapse ab.

Die Synapse gewichtet die Signalübertragung zur nächsten Zelle.

Transmitterstoffe (Neurotransmitter):

Glutaminsäure: erregender Transmitter
GABA: hemmender Transmitter
} $^1/_3$ der Synapsen verwenden diese Transmitter. Mangel führt zu unkoordinierten Bewegungen.

Dopamin: Steuert emotionale Reaktionen. Chemisch ähnlich dem Kokain.

Serotomin: Reguliert Körpertemperatur und Wahrnehmung von Sensorsignalen, und wird durch Alkohol und Antidepressiva beeinflusst. Ähnelt chemisch dem LSD und Extasy- Derivaten.

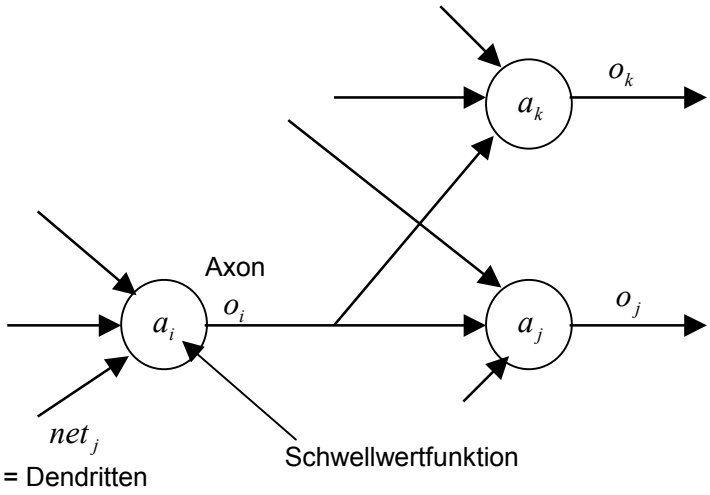
Neuropeptide: Steuern das Schmerzempfinden und Gefühlsregungen. Reagieren auf Opiate.

Neuradrenalin: Steuert Stimmungslage und Wachzustand (Panik). Reagiert auf Meskalin.

Diese Vielfalt kann bisher in der Simulation nicht abgedeckt werden.

2. Modelleigenschaften

2.1. Bestandteile des Modellneurons



2.1.1. net_j = Propagierungsfunktion

einfachste Propagierungsfunktion ist das Skalarprodukt:

$$net_j = \sum_{i=1}^{n} w_{ij} \cdot o_i$$

Fasst man den Eingang $o_1 \cdots o_n$ als Vektor auf, und $w_{i1} \cdots w_{in}$ ebenfalls als Vektor, dann:

$$net_j = \vec{w}_i \circ \vec{o}$$

Das Skalarprodukt bestimmt wie ähnlich die beiden Vektoren $\vec{w}_i$ und der Eingang $\vec{o}$ zueinander sind. Sind die Vektoren parallel (ähnlich), so ist das Skalarprodukt maximal.

2.1.2. Aktivierungszustand $a_i(t)$

Der Aktivierungszustand kennzeichnet die Aktivität einer Zelle und berechnet die neue Aktivität $a_i(t+1)$.

$$a_j(t+1) = f_{Akt}\left[a_j(t), net_j, \vartheta\right]$$

Aktivität der Zelle zum Zeitpunkt t

Propagierungsfunktion

$$\sum_{i=1}^{n} w_{ij} o_i$$

Schwellwert

2.1.3. Ausgabefunktion o_j

$$o_j = f_{out}\left[a_j\right]$$

Im Folgenden wird meist die Identität verwendet.

$$\boxed{o_j = a_j}$$ 

2.1.4. Lernregel

Die Lernregel gibt an, wie die Gewichte w_{ij} verändert werden müssen, um ein Trainingsergebnis zu erzielen.

2.1.5. Netztopologie (Netztstruktur)

Bezeichnet die:
- Anordnung der Neuronen in Schichten
- Verteilschicht (für Eingänge)

2.2. Schwellwertfunktionen

(Aktivierungsfunktionen, Transferfunktionen)

Step-Funktion: (FS: einfache Schwellwertfunktion) (siehe Bild 2.2.2)

$$\boxed{a_j = \Theta\left(net_j - J_j\right) = \begin{cases} 0 & für\ net_j - J_j < J_j \\ 1 & für\ net_j - J_j \geq J_j \end{cases}}$$ 

Verallgemeinerte Step-Funktion: (FS: binäre Schwllwerfunktion) (siehe Bild 2.2.3)

$$\boxed{a_j = (M-m)\cdot\Theta\left(net_j - J_j\right) + m = \begin{cases} m & für\ net_j - J_j < J_j \\ M & für\ net_j - J_j \geq J_j \end{cases}}$$ 

Sigmoid-Funktion: (Fermi Funktion) (siehe Bild 2.2.4)

$$\boxed{a_j = \sigma\left(net_j - J_j\right) = \frac{1}{1+e^{-\left(net_j - J_j\right)}}}$$ 

Verallgemeinerte Sigmoid-Funktion: (siehe Bild 2.2.5)

$$\boxed{a_j = \sigma\left(net_j - J_j\right) = m + \frac{M-m}{1+e^{-\frac{4\cdot\sigma}{M-m}\left(net_j - J_j\right)}}}$$ 

bestimmt Übergangsbreite der Funktion

Lineare Schwellwertfunktion: (siehe Bild 2.2.6)

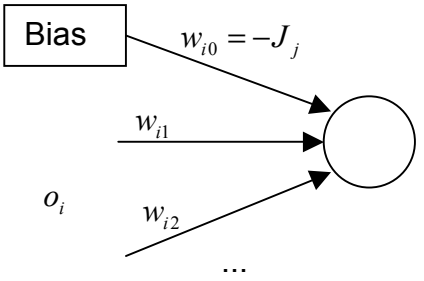
$$\boxed{a_j = \begin{cases} m & für\ net_j - J_j < J_j \\ a\cdot\left(net_j - J_j\right) + m & für\ net_j - J_j \geq J_j \end{cases}}$$ 

Einführung des Bias-Neurons: (auch ON-Neuron)

Jedes Neuron erhält einen zusätzlichen Eingang $o_0 = 1$, der mit dem Ausgang des Bias-Neurons verbunden wist.
Wählt man als Gewicht für das Bias-Neuron:

$w_{i0} = -J_i$, dann:

$$net_j = \sum w_{ij} \cdot o_i + \underbrace{w_{i0} \cdot o_0}_{-J_j} = net_j^{neu}$$



$$net_j^{neu} = \sum w_{ij} \cdot o_j$$

Im folgenden wird $net_j^{neu} = net_j$ verwendet.

(Da dies auch in der Formelsammlung verwendet wird, weichen die obigen Formeln aus 2.2 leicht von der Formelsammlung ab: $net_j - J_j \rightarrow net_j$)

2.3. Lernverfahren

Prinzipiell gibt es 3 Lernverfahren:
- überwachtes Lernen
- bestärktes Lernen
- unüberwachtes Lernen

Überwachtes Lernen

$$\begin{pmatrix} o_1 \\ \vdots \\ o_n \end{pmatrix} \xrightarrow{\text{externer Lehrer}} \begin{pmatrix} o_{j1} \\ \vdots \\ o_{jn} \end{pmatrix}$$

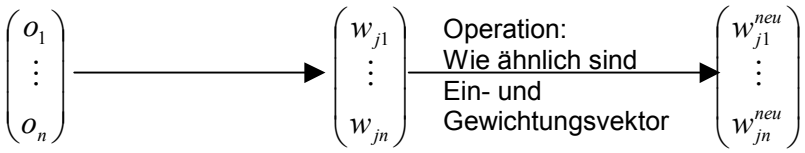
Eingangsvektor Ausgabevektor

Zu jedem Eingangsvektor $\vec{o}_i$ gibt es genau einen Ausgangsvektor.

Bestärktes Lernen:

$$\begin{pmatrix} o_1 \\ \vdots \\ o_n \end{pmatrix} \qquad \begin{pmatrix} gut \\ schlecht \end{pmatrix}$$

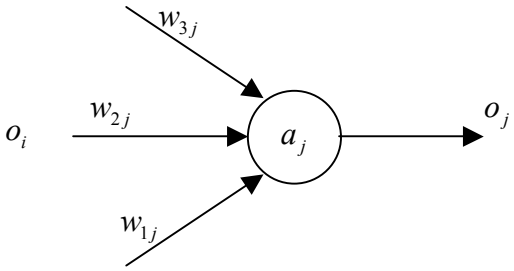
Unüberwachtes Lernen:



Gewichtsvektor w_{i*}^{neu} wird dem Eingangsvektor noch ähnlicher gemacht.

Lernalgorithmen:

a) Hebbscher Lernalgorithmus:



$$w_{ji}^{neu} = w_{ji}^{alt} + \Delta w_{ji}$$
$$\Delta w_{ji} = \eta \cdot o_i \cdot a_j \quad o_i, a_j \in \{0,1\}$$

Nur momentane Aus- und Eingangswerte o_i, o_j werden berücksichtigt, nicht die Sollwerte.
Wenn o_i und o_j gleichzeitig aktiviert sind und positive reele Zahlen sind, so wird Gewicht w_{ij} um Δw_{ij} vergrößert.
Das Gewicht w_{ij} vergrößert sich, so dass die Zelle ihren Schwellwert schneller erreicht.

Lernrate $\eta < 1$, typisch: $0{,}1 < \eta < 0{,}3$

b) Delta-Lernregel:
Die Delta-Lernregel berücksichtigt Abweichungen zwischen Soll- und Ist-Wert.
Der Sollwert wird im folgenden mit "Teaching-Vektor" bezeichnet (t_j).

$$\Delta w_{ij} = \eta \cdot (t_j - a_j) \cdot o_i$$
$$t_j - a_j := \delta_j \text{ = Fehlermaß}$$
$$\Rightarrow \Delta w_{ij} = \eta \cdot \delta_j \cdot o_i$$

Somit ist die Delta-Lernregel ein Spezialfall der Hebbschen-Lernregel.
Die Delta-Lernregel ist nur auf einschichtige Netze anwendbar!

c) Backpropagation Algorithmus: (BPA)
Bei Mehrschichtigen Netzen wird der Backpropagation Algorithmus verwendet.
(siehe Kapitel 3)

2.4. Übersicht über Netzstrukturen

Feed-Forward-Netze: (siehe Bild 2.4.a)
Verbindungen erfolgen schichtweise von einer zur nächsten. Sie werden am meisten verwendet. Diese Netze lernen Trainingsmuster.

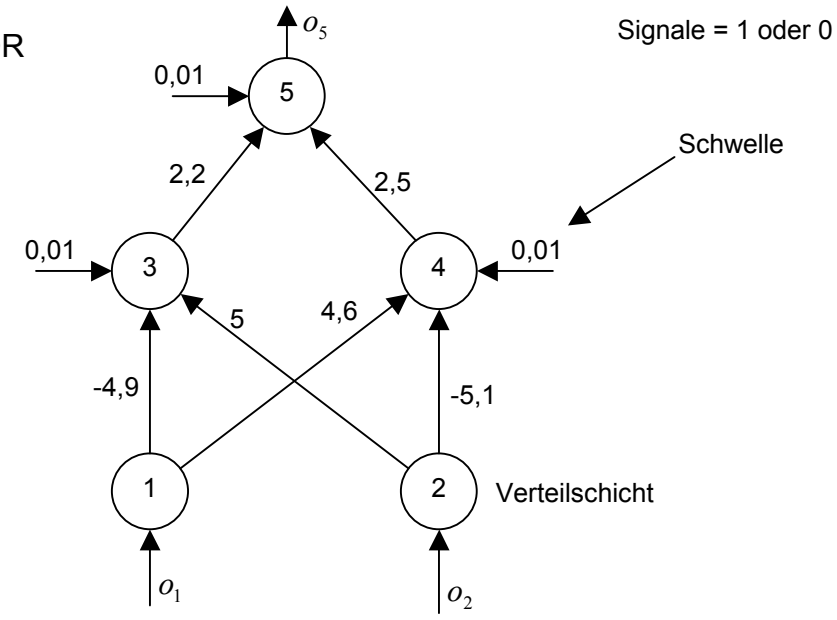
Feed-Forward-Netze mit shortcut Verbindungen: (siehe Bild 2.4.b)
Sie haben zusätzlich zu Feed-Forward-Netzen Verbindungen, welche eine Schicht überspringen.

Netze mit Rückkopplung:

- Direkte Rückkopplung: (siehe Bild 2.4.c)
  Neuron nimmt nur Grenzzustände an, da die eigene Aktivierung verstärkt wird.
- Indirekte Rückkopplung: (siehe Bild 2.4.d)
  Verbindungen von höherer Schicht zu niedrigerer Schicht, um Aufmersamkeit auf bestimmte Eingabemerkmale zu erreichen.
- Laterale Rückkopplung: (siehe Bild 2.4.e)
  Jedes Neuron einer Schicht erhält hemmende Verbindungen. Das Neuron mit stärkster Aktivierung hemmt die anderen Neuronen dieser Schicht (Winner takes all -Netzwerk).
- Vollständig verbundenes Netz: (Hopfield-Netze) (siehe Bild 2.4.f)
  Sie werden zur assoziativen Speicherung von Bildfolgen verwendet.

2.5. Beispiel eines Feed-Forward-Netzes mit 3 Neuronen

Logisches OR

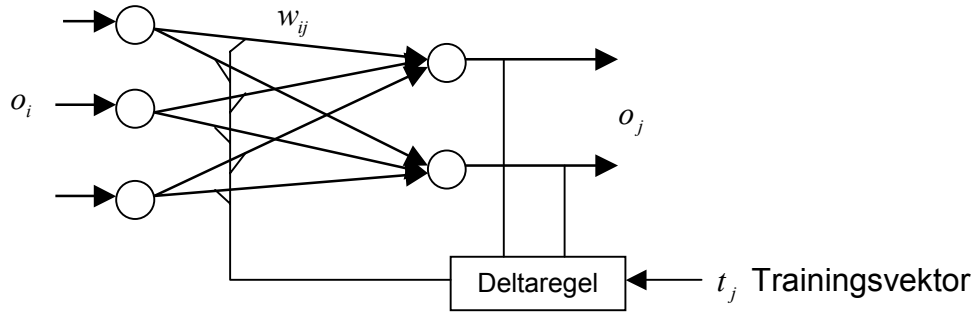


o_1	o_2	net_3	o_3	net_4	o_4	net_5	o_5
0	0	$0\cdot(-4{,}9)+0\cdot 5=0$	0	$0\cdot 4{,}6+0\cdot(-5{,}1)=0$	0	$0\cdot 2{,}2+0\cdot 2{,}5=0$	0
0	1	$0\cdot(-4{,}9)+1\cdot 5=5$	1	$0\cdot 4{,}6+1\cdot(-5{,}1)=-5{,}1$	0	$1\cdot 2{,}2+0\cdot 2{,}5=2{,}2$	1
1	0	$1\cdot(-4{,}9)+0\cdot 5=-4{,}9$	0	$1\cdot 4{,}6+0\cdot(-5{,}1)=4{,}6$	1	$0\cdot 2{,}2+1\cdot 2{,}5=2{,}5$	1
1	1	$1\cdot(-4{,}9)+1\cdot 5=0{,}1$	1	$1\cdot 4{,}6+1\cdot(-5{,}1)=-0{,}5$	0	$1\cdot 2{,}2+0\cdot 2{,}5=2{,}2$	1

3. Backpropagation Algorithmus (BPG)

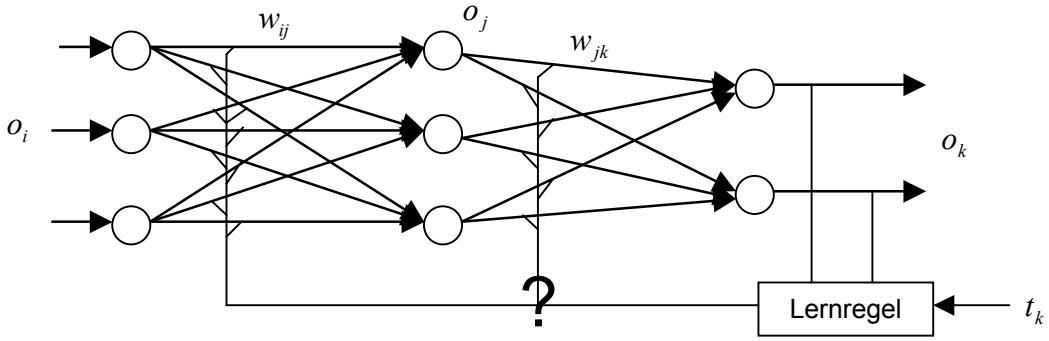
3.1. Hintergrund

Für Einschichtige Netze ist Delta-Regel ausreichen:



Deltalernregel: $\Delta w_{ij} = \eta\left(t_j - o_j\right)\cdot o_i$

Bei 2-Schichtigen Netzen ist Deltalernregel nicht mehr anwendbar, da keine Vorschrift verfügbar ist, wie die Gewischte der verdeckten Schichten verändert werden sollen.



Wie wird die Änderung der Gewichte verteilt ?

Ziel: Fehlerfreie Abbildung der Eingangsmuster o_i auf Trainingsmuster t_k.

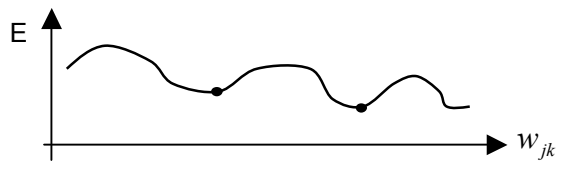
Methode: Bei vorwärts gerichteten Netzen ist es der Backpropagation Algorithmus, der 1086 von Rumelhart entwickelt wurde.

Fehlermaß für fehlerfrei Abbildung ist:

$$E = \sum_p E_p \quad E_p = \frac{1}{2}\sum_k \left(t_{pk} - o_{pk}\right)^2$$ ☺ (E=Errorfunktion, p=Pattern)

$\Rightarrow$ Quadratischer Fehler über alle Musterpaare (Eingang/Ausgang) muss minimal werden.

Im Eindimensionalen Fall: $o_1 \rightarrow t_1$



$$E = \left(t_1 - o_1\right)^2 = \left(t_1 - \Theta\underbrace{\left(\sum_j w_{jk}\cdot o_j \right)}_{net_j} \right)^2$$

Forderung: $\Delta w_{jk} \approx$ -Steigung der Fehlerfunktion E

Genauer:

$$\boxed{\Delta w_{jk} = \sum_p -\eta \cdot \frac{\delta E_p}{\delta w_{jk}}}$$

Bei 2-schichtigen Netzen gilt:

$$E = \sum_p \left(\sum_k \left(t_k - o_k \right) \right)^2$$

$$\uparrow \Theta\left(\sum_j w_{jk} \cdot o_j \right)$$

$$\uparrow \Theta\left(\sum_i w_{ij} \cdot o_i \right)$$

$$\Delta w_{jk} = -\eta \cdot \frac{\delta E_p}{\delta w_{jk}} = -\eta \cdot \frac{\delta E_p}{\delta net_k} \cdot \frac{\delta net_k}{\delta w_{jk}}$$

Man definiert das Fehlersignal δk:

$$\delta k = -\frac{\delta E_p}{\delta w_{jk}} \qquad \frac{\delta net_k}{\delta w_{jk}} = \frac{\delta \sum w_{jk} \cdot o_j}{\delta w_{jk}} = o_j$$

Somit ist:

$$\Delta w_{jk} = -\eta \cdot \frac{\delta E}{\delta w_{jk}} = +\eta \cdot o_j \cdot \delta k$$

Für verdeckte Schicht ergibt sich:

$$\Delta w_{jk} = \eta \cdot o_i \cdot \delta j \qquad \text{wobei} \qquad \delta j = \sigma'\left(net_j\right) \cdot \sum_k w_{jk} \cdot \delta k$$

alle nachfolgenden Neuronen

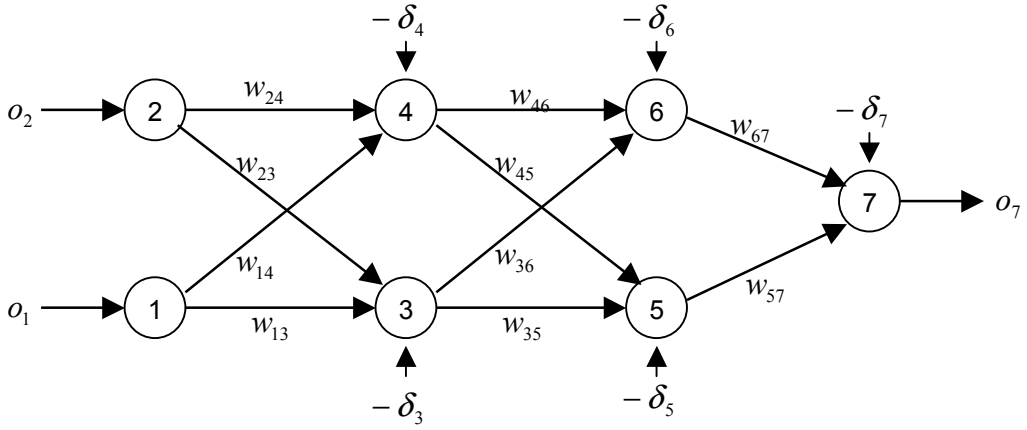
$$\boxed{\sigma'\left(net_j\right) = o_j \cdot \left(1 - o_j\right)}$$

Zusammenfassung:

$$\boxed{\Delta w_{ij} = \eta \cdot \sum_p o_{pi} \cdot \delta_{pj}}$$

$$\boxed{\begin{aligned} \delta_{pj} &= \begin{cases} \sigma'\left(net_j\right) \cdot \left(t_{pj} - o_{pj}\right) & falls\ j\ Ausgabezelle \\ \sigma'\left(net_j\right) \cdot \sum_l \delta_{pk} \cdot w_{jk} & falls\ j\ verdeckte\ Zelle \end{cases} \\ \delta_{pj} &= \begin{cases} o_j \cdot \left(1 - o_j\right) \cdot \left(t_{pj} - o_{pj}\right) & falls\ j\ Ausgabezelle \\ o_j \cdot \left(1 - o_j\right) \cdot \sum_l \delta_{pk} \cdot w_{jk} & falls\ j\ verdeckte\ Zelle \end{cases} \end{aligned}}$$

Beispiel:



$$\delta_7 = o_7 \cdot (1 - o_7) \cdot (t - o_7)$$
$$\Rightarrow \Delta w_{67} = \eta \cdot o_6 \cdot \delta_7$$
$$\Delta w_{57} = \eta \cdot o_5 \cdot \delta_7$$

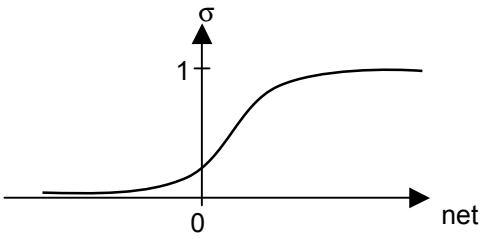
$$\delta_6 = o_6 \cdot (1 - o_6) \cdot w_{67} \cdot \delta_7 \qquad \delta_5 = o_5 \cdot (1 - o_5) \cdot w_{57} \cdot \delta_7$$
$$\Rightarrow \Delta w_{36} = \eta \cdot o_3 \cdot \delta_6 \qquad \Rightarrow \Delta w_{35} = \eta \cdot o_3 \cdot \delta_5$$
$$\Delta w_{46} = \eta \cdot o_4 \cdot \delta_6 \qquad \Delta w_{45} = \eta \cdot o_4 \cdot \delta_5$$

$$\delta_3 = o_3 \cdot (1 - o_3) \cdot (w_{36} \cdot \delta_6 + w_{35} \cdot \delta_5 + w_{67} \cdot \delta_7 + w_{57} \cdot \delta_7)$$

3.2. Probleme und deren Beseitigung

3.2.1. Bildbereich der Sigmoidfunktion $\sigma(net) = [0,1]$

Ausgabewerte 0 und 1 können nur schwer erreicht werden, da nur asymptotische Annäherung an 0 und 1 möglich ist.



Abhilfe:

a) Änderung der binären Eingabevektoren:

[0, 1] → [-0.5, 0.5] oder

[0, 1] → [0.2, 0.8]

b) Verschiebung des Wertebereichs:

[0, 1] → [-0.5, 0.5]

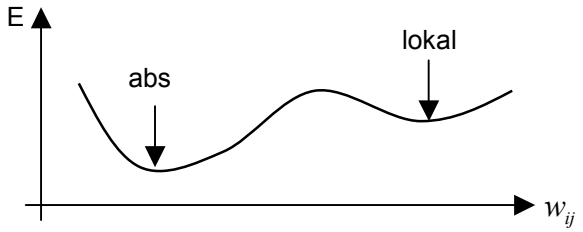
$\sigma_{\text{mod}}(net) = \frac{1}{1 + e^{net}} - \frac{1}{2}$ (modifizierte Sigmoidfunktion)

3.2.2. Symetry breaking

Die Startgewichte dürfen nicht alle gleich groß gewählt werden, da ansonsten die vorgelagerten Schichten keine unterschiedlichen Gewichte annehmen können.

Abhilfe:
Verwendung kleiner zufälliger Initialisierungswerte.

3.2.3. Lokale Minima

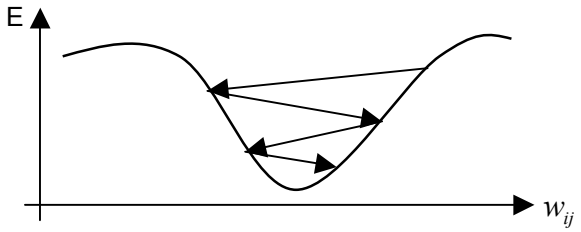


BPA locht auf lokales Minimum ein und findet kein absolutes Minimum.

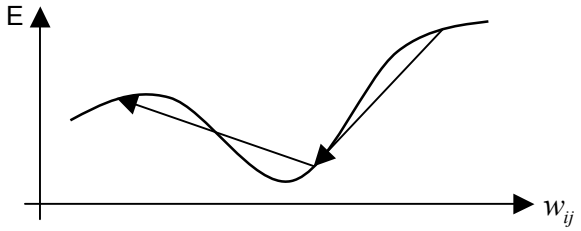
Abhilfe:
Dies ist ein inhärentes Problem beim BPA. Kann nur durch genetische Verfahren gelöst werden.

3.2.4. Oszillationen und verlassen guter Minima

BPA springt von einer Seite des Minima auf die andere.
$\Rightarrow$ Errorfunktion oszilliert



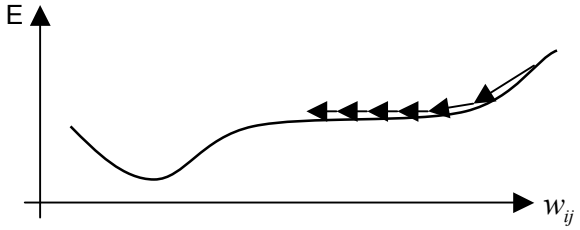
BPA verlässt ein gutes Minima



Abhilfe:
η kleiner wählen.

3.2.5. Flache Plateaus

Es sind sehr viele Schritte notwendig, um da Plateau zu verlassen.



Abhilfe:
Einführung eines **Momentumterms**.

$$\Delta w_{ij}(t+1) = \eta \cdot \sum_p o_{pi} \cdot \delta_{pj} + \mu \cdot \Delta w_{ij}(t)$$

Momentumterm

μ berücksichtigt die Gewichtsänderung vom vorherigen Zeitpunkt und bewirkt eine Beschleunigung in flachen Plateaus und ein abbremsen in stark zerklüfteten Bereichen. μ steuert das Erinnerungsvermögen. $0.6 < \mu < 0.9$

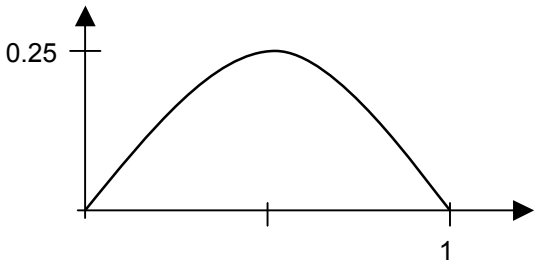
3.2.6. Flat Spot

Im Sättigungsbereich (0 oder 1) der Sigmoidfunktion kann ein Neuron sein Gewicht kaum ändern.

$$\sigma'(net) = (1 - o_j) \cdot o_j$$

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot o_i$$

$$\sigma'(net) \cdot \sum w_{jk} \cdot \delta_k$$



Abhilfe:
Addition von 0.1 zu $\sigma'(net)$. Dann läuft kurve zwischen 0.1 und 0.35.
$\Rightarrow$ Reduzierung der Lernzeit um Faktor 2.

Fehlersignal = $\delta_j = \begin{cases} (\sigma'(net) + c) \cdot (t_j - o_j) & \text{für Ausgangszelle} \\ (\sigma'(net) + c) \cdot \sum_k \delta_k \cdot w_{jk} & \text{für verdeckte Zelle} \end{cases}$

3.2.7. Weight Decay

Große Gewichte machen Fehlerfläche steiler und Zerklüfteter, wodurch vermehrt Oszillationen auftreten und gute Minima verlassen werden.

Abhilfe:

$$\Delta w_{ij}(t+1) = \eta \cdot o_i \cdot \delta_j - d_{\max} \cdot w_{ij}(t)$$

Zurückführen auf modifizierte Fehlerfunktion

$$E^{\text{mod}} = E + \frac{d_{\max}}{2} \cdot \underbrace{\sum_{i,j} (w_{ij})^2}_{\text{Bestraft große Gewichte}}$$

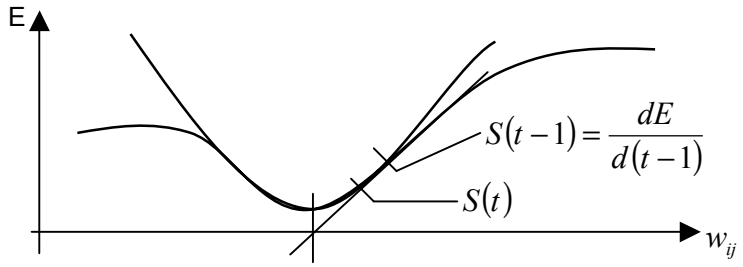
3.3. Beschleunigte Lernverfahren

3.3.1. Manhattan Training

$$\boxed{\Delta w_{ij} = \eta \cdot o_i \cdot sign(\delta_j)}$$ ☺

Es ist nicht mehr $|\delta_j|$ wichtig, sondern das Vorzeichen.

3.3.2. Quickprop



Statt vieler Schritte wird das Minimum einer angepassten Parabel berechnet.

$$\frac{S(t-1) - S(t)}{\Delta w_{ij}(t-1)} = \frac{S(t) - 0}{\Delta w_{ij}(t)}$$

$$\boxed{\Rightarrow \Delta w_{ij}(t) = \frac{S(t)}{S(t-1) - S(t)} \cdot \Delta w_{ij}(t-1)}$$ ☺

3.3.3. Rprop (Resilent Propagation)

Die Resilent Propagation ist eine Kombination aus Manhattan Training und Quickprop.

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & falls\ S(t-1)\cdot S(t) > 0 \wedge S(t) > 0 \\ \Delta_{ij}(t) & falls\ S(t-1)\cdot S(t) > 0 \wedge S(t) < 0 \\ -\Delta_{ij}(t-1) & falls\ S(t-1)\cdot S(t) < 0 \\ -sign\left(S(t)\cdot \Delta_{ij}(t)\right) & sonst \end{cases}$$

wobei

$$\Delta_{ij}(t) = \begin{cases} -\Delta_{ij}(t-1)\cdot \eta^{+} & falls\ S(t-1)\cdot S(t) > 0 \\ \Delta_{ij}(t-1)\cdot \eta^{-} & falls\ S(t-1)\cdot S(t) < 0 \\ \Delta_{ij}(t-1) & sonst \end{cases}$$

mit $0 < \eta^{-} < 1 < \eta^{+}$

Diese Formel bewirkt:
Die Gewichtänderung hängt nicht mehr vom Betrag der Steigung ab, sondern nur noch vom Vorzeichen.. Sind die Vorzeichen gleich, dann wird Δw_{ij} erhöht, sind sie unterschiedlich, dann wird Δw_{ij} erniedrigt

Der Vorzeichenwechsel sagt aus, dass über ein Minima hinweggesprungen wurde. Daher ist die Gewichtsänderung in Abhängigkeit vom Vorzeichenwechsel der Steigung sinnvoll.

4. Netzmodelle

4.1. PERCEPTRON

(1958) von Rosenblatt)

Ein PERCEPTRON-Netz ist ein vorwärts gekoppeltes Netz (siehe Bild 4.1) mit festen Gewichten die zufällig gewählt wurden in der 1. Schicht, und lernbaren Gewichten in der 2. Schicht.
1. Schicht nennt man **S-Schicht**.
2. Schicht nennt man **R-Schicht** (Response).

Lernregel, angewandt auf die 2. Schicht, entspricht der Delta-Regel:

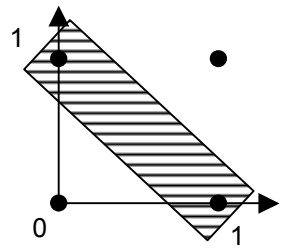
$$w_{ij}^{neu} = w_{ij}^{alt} + \Delta w_{ij}$$

$$\Delta w_{ij} = \eta \cdot (t_i - a_i) \cdot o_j$$

t_i: Sollwert, a_i: Ausgangswert, o_j: Eingangswert

Nachteil des PERCEPTRON-Modells:
XOR nicht lösbar, da das Netz prinzipiell nur einschichtig ist.

Trägt man die Wertemenge in einer Ebene auf, und legt die Bildmenge als schraffierte Fläche darüber,



so haben Mensky und Papert 1969 gezeigt, dass PERCEPTRON keine XOR-Separierung vornehmen kann

4.2. Separierfähigkeit vorwärtsgekoppelter Netze

(siehe Bild 4.2)

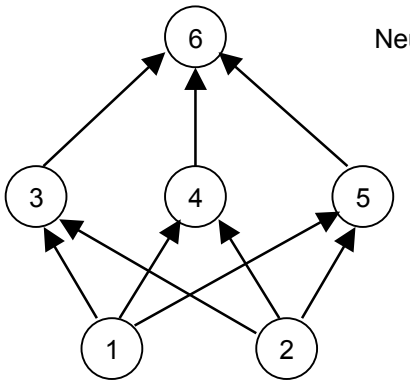
Einschichtige Netze können nur eine lineare Separierung durchführen.

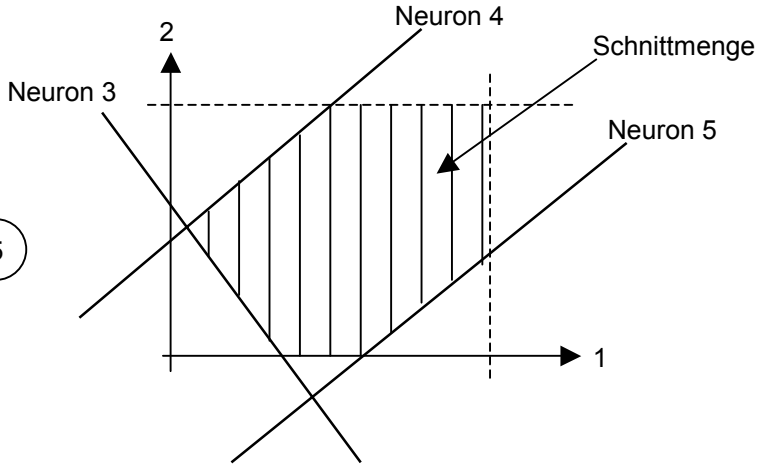
Zweischichtige Netze können eine konvexe Separierung durchführen.

Dreischichtige Netze können eine konvexe / konkave Separierung durchführen. Somit können bei einer entsprechenden Netzstruktur auch "löchrige" topologische Mengen separiert werden.

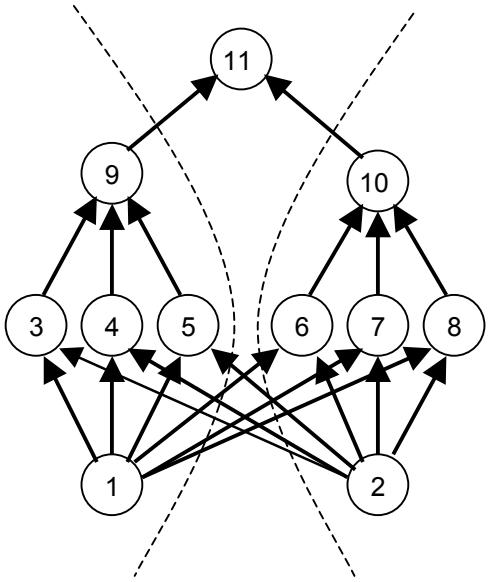
Weitere Schichten bringen keine zusätzlichen Fähigkeiten der Separierbarkeit von Mengen.

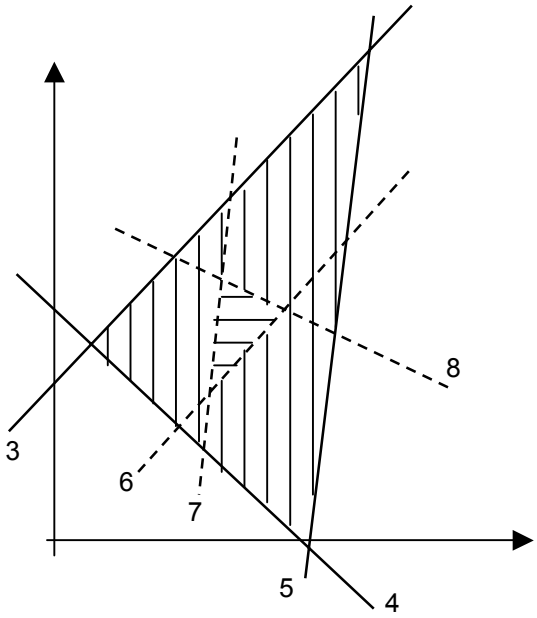
Beispiele:
2-Schichtiges Netz:





3-Schichtiges Netz:





4.3. Rückgekoppelte Netze

(Jordan & Elman Netze)

Zweck:
Klassifizierung und Erkennung zeitveränderlicher Muster bei denen die zeitliche Abfolge entscheidet.

Jordan-Netze: (siehe Bild 4.4 oben)
Die Ausgabe des vorherigen Musters wird über Kontextzellen zwischengespeichert, und als Eingabe mit dem zeitlich neuen Muster den verdeckten Zellen präsentiert.
Die Kontextzellen sind über feste Gewichte w=1 mit den Ausgabezellen verbunden. Das Erinnerungsvermögen der Kontextzellen wird über direkte Rückkopplung der Stärke λ gesteuert.

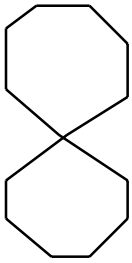
z.B.: $\lambda = 1$: vollständige Erinnerung
$\lambda = 0.5$ halbe Erinnerung
$\lambda = 0$ keine Erinnerung

Eigenschaften:

- Assoziiert Ausgabesequenzen bei festen Eingabevektoren
- kein Lernverfahren für Kontextzellen
- Zahl der Kontextzellen = Zahl der Ausgabezellen
- Lernverfahren: modifizierte
  - Backpropagation
  - Quickprop
  - Rprop

  Modifikation betrifft nur Kontextzellen. d.h. Summierungsgrenzen werden um die Zahl der Kontextzellen erweitert.

Beispiel:
Abfolge von Strichen für eine 8 (siehe Übung 3)



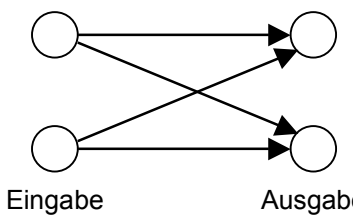
Elman-Netze: (siehe Bild 4.4 unten)
Hier werden die verdeckten Zellen über Kontextzellen Zwischengespeichert, so dass man ohne direkte Rückkopplung in den Kontextzellen auskommt. Ansonsten sind Elman-Netze analog zu Jordan-Netze.

4.4. Cascade Correlations (Cascor)

Probleme bei Feed-Forward-Netzen:

- Netztopologie muss von Anwender vorgegeben werden
- Vorwärts- und Rückwärts-Propagierung beim lernen nötig.
- Verdeckte Zellen einer Ebene können nicht miteinander kommunizieren. Beim Lernvorgang konzentrieren sich einige Neuronen besonders auf die Lösung eines Teilproblems, weil sie nur die Vorgänger- und Nachfolgezellen sehen, aber nicht ihre Nachbarzellen.

Cascad- Strategie:
1. Man startet mit minimalem Netz, ohne verdeckte Neuronen, nur Ein- und Ausgabezellen mit Feed-Forward Verbindungen.



Lernen mit Deltaregel,
welche in BPA enthalten ist.

2. Hinzufügen eines Verdeckten Neurons als Kandidatenzelle pro Zeitschritt, ohne Anbindung an Ausgabe-Neuronen.
Maximierung der Korrelation zwischen Ausgabe der Verdeckten Zelle (Kandidatenzelle) und Restfehler des Netzes.
Positive Korrelation $\Rightarrow$ Negative Verbindung
Negative Korrelation $\Rightarrow$ Positive Verbindung

Korellationsfunktion S_j:

$$S_k = \sum_p \left(o_{pj} - \overline{o}_j\right) \cdot \left(\delta_{pk} - \overline{\delta}_k\right)$$
$$S_j = \sum_k \left|S_k\right|$$

mit: S_j = Summe der Beträge der Korrelation zwischen der Anzahl der Kandidatenzelle o_j und dem Netzfehler δ_k.

$\overline{o}_j$ = Mittlere Ausgabe über alle Muster $= \frac{1}{n} \cdot \sum_{j=0}^{n} o_j$, n = Anzahl der Muster.

$\overline{\delta}_k$ = Mittlerer Ausgabefehler über alle Muster der Ausgabezelle $k = \frac{1}{n} \sum_{k=0}^{n} \delta_k$

p = Pattern
k = Index der Ausgabezelle

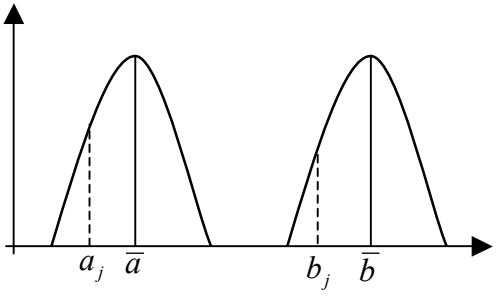
Maximierung von S_j:

Erfolgt analog zum BPA mit dem Unterschied, dass anstelle eines Gradientenabstiegs ein Gradientenaufstieg gewählt werden muss:

$$\frac{\partial S_j}{\partial w_{ij}} = \sum_k \sum_p sign\left(S_k\right) \cdot \sigma'\left(net_j\right) \cdot o_{pi} \cdot \left(\delta_{pk} - \overline{\delta}_k\right)$$

Nach der Maximierung werden die Gewichte eingefroren und die Verbindungen zu den Ausgabezellen hergestellt. Fehlerbestimmung δ_k und hinzufügen einer neuen Kandidatengruppe, aus der das Neuron mit der maximalen Korrelation als neues Neuron hinzugefügt wird.

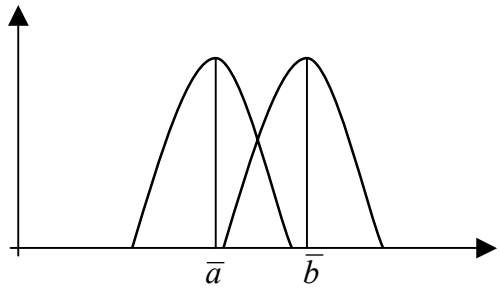
Begriff der Korrelation:
Eine Korrelation stellt fest, inwieweit zwei Größen voneinander abhängig sind.
z.B.: 2 Variablen a, b mit Mittelwerten $\overline{a}, \overline{b}$



2 Gaußverteilungen

Ist die Gaußverteilung symmetrisch, so ist die Korrelationsfunktion:
$S = \sum (a_j - \bar{a}) \cdot (b_j - \bar{b}) = 0$
d. h. a und b sind unabhängig voneinander.

Ist Gaußverteilung unsymmetrisch, so ist $S \neq 0$



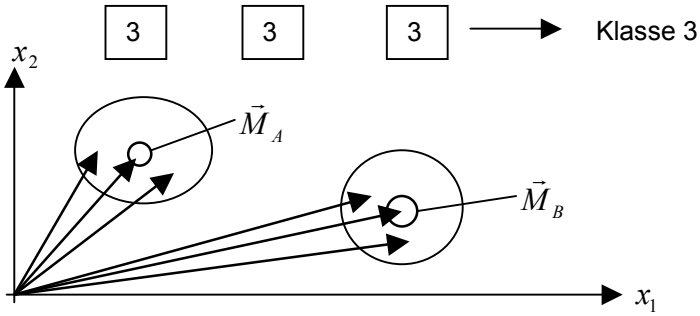
Es kommt zur Überlappung von a und b, was Abhängigkeit anzeigt.

4.5. Kohonen- Netze (SOMs)

Man unterscheidet zwischen:
- überwachte Kohonen-Netze, die "Learning-Vektor Quantisation" LVQ genannt werden oder DLVQ (Dynamic LVQ)
- SOMs (self organizing maps)
- Counterpropagation

4.5.1. DLVQ:

Aufgabe: natürliche Gruppierung der Daten zu finden



Prinzip: Eingabevektoren $\vec{x}$, die zur selben Klasse gehören werden mit einem Mittelwertvektor $\vec{\mu}$ einer Datenwolke zugeordnet. Die Klassifizierung ist vorher bekannt.

Separierung der Klasse: Messung der Distanzen $|\vec{\mu} - \vec{x}|$ von $\vec{x}$ von allen Klassifikationsneuronen und Zuordnung von $\vec{x}$ zu nächstem mittleren Vektor $\vec{\mu}_i$. Es wird das Minimum der Distanzen berechnet und der Vektor $\vec{\mu}_i$ mit $\vec{w}_i$ zugeordnet.
Bei Falschzuordnung von $\vec{x}$ zum Vektor $\vec{\mu}_B$ wird Vektor $\vec{\mu}_A$ auf $\vec{x}$ zu bewegt und $\vec{\mu}_B$ von $\vec{x}$ weg bewegt.
Daraus ergeben sich drei Lernparameter:

η^+ bei richtiger Zuordnung
η^- bei falscher Zuordnung
cycles: Zahl der Klassifizierungsneuronen

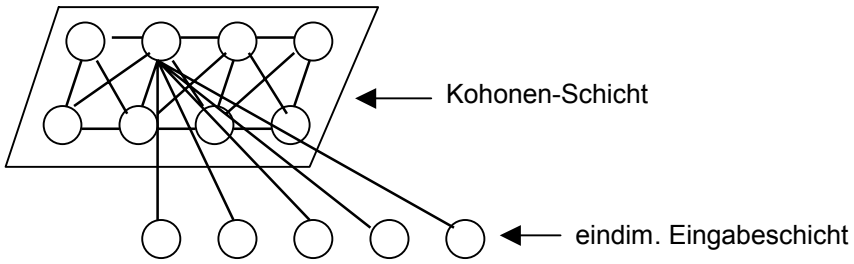
Vorgehensweise:

1. Berechnung der $\vec{\mu}_i$ für jede Klasse des Datensets
2. Zuordnung des Pattern zu einem Referenz- bzw. Klassifizierungs- Vektor $\vec{\mu}_i$
3. Erkennung der Falschzuordnung und Verschiebung der Mittelwertsvektoren $\vec{\mu}_A$ und $\vec{\mu}_B$

Neustart:
Eingabevektor $\vec{x}$
Ausgabevektor 3

4.5.2. SOMs

Ziel: Einteilung der Daten in Klassen, wobei die Zuordnung nicht bekannt ist.



In der Kohonen-Schicht gibt es Querverbindungen zu den nächsten Nachbarn. Hat man 16 Komponenten des Eingabevektors, so muss Kohonen-Schicht aus mindestens 16x16 Neuronen bestehen.
Eine weitere Eigenschaft ist die Nummerierung der Kohonen-Schicht. Die Zuordnung des Eingabevektors zum Kohonen-Neuron erfolgt durch:

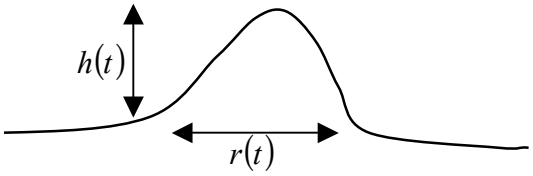
$$\max\left(\vec{X} \cdot \vec{W}_j\right) = \vec{X} \cdot \vec{W}_C$$

d. h. Neuron mit maximalem Skalarprodukt gewinnt. Topologische Ordnung wird durch Nachbarschaftsbeziehungen hergestellt. Anders als bei LVQ's wird nicht nur einer, sondern mehrere Vektoren der Nachbarschaft verändert.

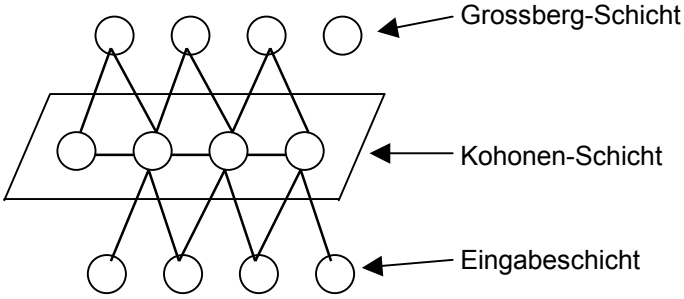
Der Radius ist einstellbar:

$$\Delta w_{ij}(t) = h(t) \cdot e^{-\left(\frac{\|\vec{w}_j - \vec{w}_c\|}{r(t)}\right)^2} \cdot \left(x_i(t) - w_{ij}(t)\right)$$

$r(t)$ = Radius der Nachbarschaft
$h(t)$ = Adaptionshöhe $0 \le h(t) < 1$



Counterpropagation:
Kombination von Kohonennetz mit einer Grossberg-Schicht:

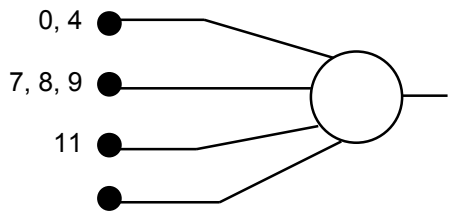


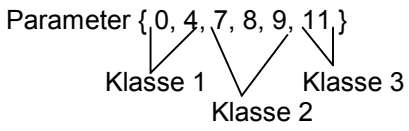
Kohonen-Schicht bildet ein "winner takes all"- Netz und aktiviert Master in Grossberg-Schicht.

4.6. Netze mit Rezeptiven Feldern

4.6.1. Optimale Datenrepräsentation

1. Kodierung des Eingabebereichs eines Parameters in einzelnen Neuronen:

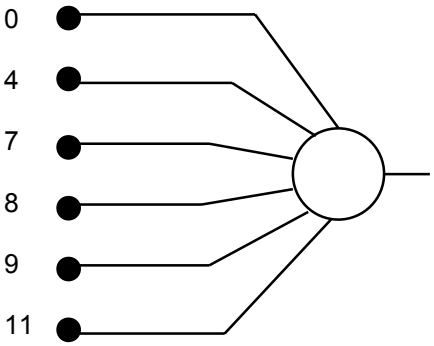




Nachteile:

- Nur ein Gewicht pro Zahl kann optimal eingestellt werden
- Trennung in nur 2 lineare Mengen möglich

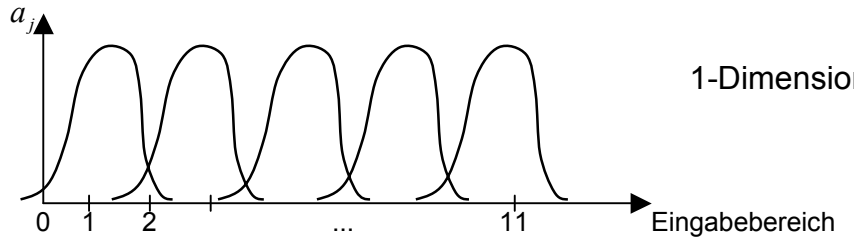
2. Channel Modell:
Man verteilt jeden Wert auf ein Neuron:



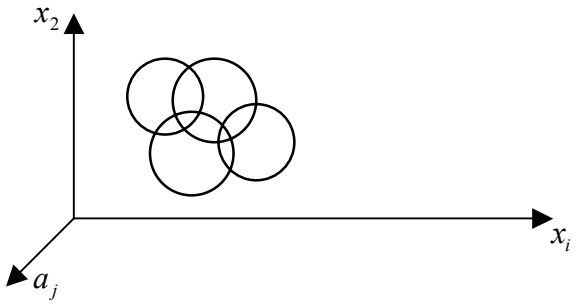
Nachteil:

- Diese Methode ist unpraktisch

3. Topologische Kodierung in rezeptiven Feldern:



1-Dimensional

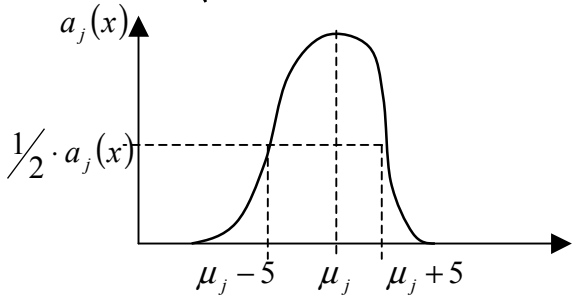


2-Dimensional

Verteilung des Wertebereichs auf mehrere Neuronen und Aktivierung der Neuronen mit überlappenden Radial Basis Functions (RBF).

Eine Möglichkeit eines RBF's währe die Gauß-Verteilungskurve:

$$a_j(x) = \frac{1}{\sqrt{2 \cdot \sigma}} \cdot e^{\frac{-1/2 \cdot (x - \mu_j)}{\sigma^2}}$$



$2 \cdot \sigma$ ist die Halbwertsbreite der Gaußkurve

4.6.2. Radial Basis Function (RBF)-Netze

Prinzip: Aufwendige Initialisierung spart viele Lernzyklen.

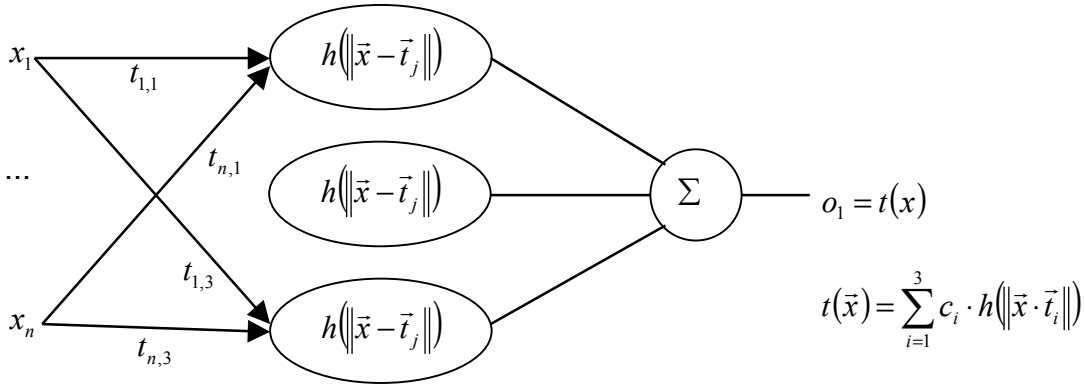
Abbildung: $\vec{x} \to \vec{y} = \sum c_i \cdot h\left(\left|\vec{x} - \vec{t}_i\right|\right)$

mit h = radiale Basis-Funktion
t_i = Klassen bzw. Zentren
c_i = Koeffizienten

Jede beliebige Funktion kann nach diesem Verfahren approximiert werden. Die c_i können nach Approximationstheorie berechnet werden, wobei die Fehlerfunktion

$H(t) = \sum (y_i - f(x_i))^2 + \lambda \cdot \|P_t\|^2$ λ = Glättungsfaktor

hierbei minimiert wird.



Struktur erweiterbar auf:

- mehrere Ausgänge
- Breite der RBF's als Basis-Neuronen abspeicherbar
- Verwendung einer umkehrbaren Schwellwertfunktion ζ zur Berechnung der Aktivität des Ausgangsneurons

$$o_k = \zeta(f_k(\vec{x})) = \zeta\left(\sum c_{jk} \cdot h\left(\left\|\vec{x} - \vec{t}_j\right\|, \underbrace{p_j}_{\substack{\text{Breite} \\ \text{des RBF's}}}\right)\right) + \underbrace{\sum_{i=1}^{n} (d_{ik} \cdot x_i)}_{\substack{\text{Schorcut} \\ \text{Connections}}} + b_k$$

c_i bzw. c_{jk} werden aus Gleichungssystem berechnet.
Vorteil: Gewichte des RBF-Netzes werden aus Matrixtheorie berechnet.

Es gibt 3 Radial Basis Funktionen:

- $h(r, p) = e^{-p \cdot r^2}$
- $h(r^2, p) = \sqrt{p + r^2}$
- $h(r^2, p) = (p \cdot r^2) \cdot \ln(p \cdot r)$

mit $(\vec{r})^2 = (\vec{x} - \vec{t})^2$ = Distanzfunktion

Initialisierung läuft folgendermaßen ab:

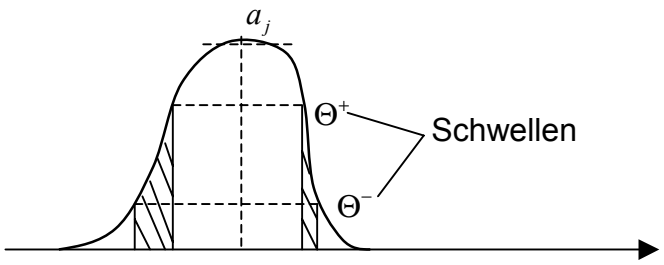
1. Jedem Teaching-Vektor wird ein Gewichts-Vektor von $\vec{x} \to h\left(\left\|\vec{x} - \vec{t}\right\|\right)$ zugeordnet.
2. Berechnung der c_i durch Matrixverfahren.
3. Lernen bedeutet: Die Zentren der t_i in Richtung der aktuellen Eingabe-Vektoren zu verschieben.

Ausdehnung auf mehrdimensionale Funktionen ist ohne weiteres möglich.

4.6.3. RBF-DDA-Netze

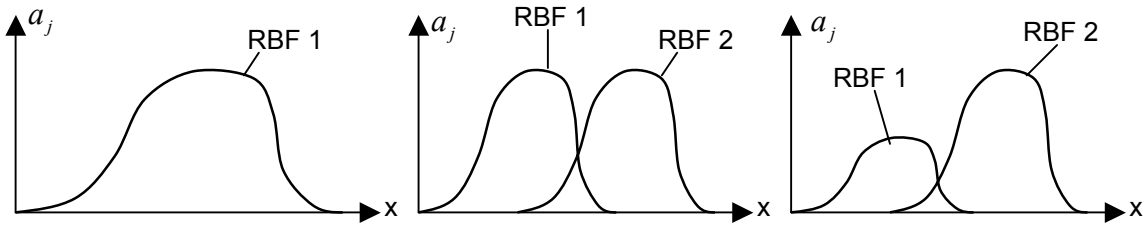
DDA = Dynamic Decay Adjustment

Konstruktives Lernverfahren, dass neue RBF's kreiert, wenn bestimmte Schwellbereiche über- bzw. unterschritten werden.
Neuer Prototyp eines RBF's wird hinzugefügt:



Schwellen Θ^+ und Θ^- definieren einen Bereich, in dem kein anderer Prototyp existieren darf.

Der Trainingsvektor muss sich innerhalb von Θ^+ befinden. Tut er es dies nicht, so wird ein neuer RBF definiert, und der Radius des alten RBF verkleinert, so dass er nicht im kritischen Bereich lieft.



gute Werte: $\Theta^+ \geq 0{,}4$

$\Theta^- < 0{,}25$

Bedingung: Eingabewerte müssen auf [0, 1] normalisiert werden.

4.7. Übersicht über Netztopologien

Netz	Lern-Algorithmus	Anwendung	Vor- Nachteile
Feed-Forward	Backpropagation, Quickprop, RProp	• Mustererkennung • Steuerung	- langsames Lernverfahren - Netzstruktur muss vorgegeben werden - keine Kommunikation innerhalb einer Schicht - Plastizitäts-/Stabilitätsproblem
Jordan & Elman-Netze	Modifizierter Backpropagation (mit Kontextzellen)	• Zeitliche Abfolgen	- Netzstruktur muss vorgegeben werden - Plastizitäts-/Stabilitätsproblem
Cascor	Korrelationsfunktion + Deltalernregel	• Mustererkennung • Musterklassifizierung	+ konstruktives Netz + kein Plastizitäts- / Stabilitätsproblem
Kohonen-Netze DLVQ, SOMS's	Überwachtes LV: Abstand zum Klassifikationsvektor, Unüberwachtes LV	• Örtliche Separierung der Muster • Musterklassifizierung	- sehr große Anzahl von Neuronen in Kohonen-Schicht + kein Plastizitäts- / Stabilitätsproblem
RBF-Netze	Deterministisches Verfahren, welches auf Approximationsverfahren beruht (keine zufälligen Init-Werte)	• Mustererkennung • Musterklassifizierung	+ Vorhersagbar (Berechenbar) + bei RBF-DDA keine Vorgabe der Netzstruktur notwendig - maximal 2-Schichtig

Plastizitäts- / Stabilitätsproblem:
Neue Muster können bereits gelernte zerstören.

5. Spracherkennung

5.1. Lauterzeugung

(Bild 6.2 oben)

Anregung erfolgt durch Stimmritze (Glottis) und Stimmbandsystem durch Luftsrom aus der Lunge.

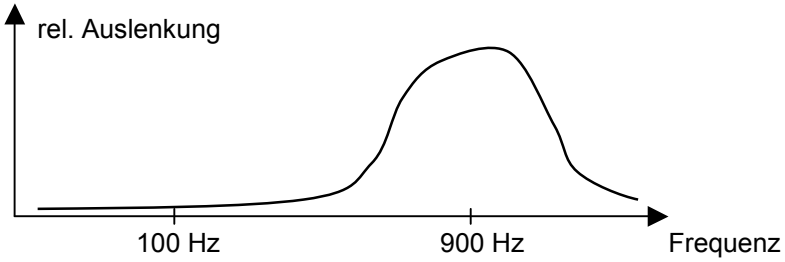
- Stimmlose Laute haben Rauschförmigen Verlauf (t, s, p)
  Glottis ist weit geöffnet und Stimmbänder entspannt.
  - keine Grundfrequenz
  - spektralmodifizierter rauschförmiger Verlauf
- Stimmhafte Laute (a, we, be)
  - periodisches Signal mit Grundfrequenz von 50 - 400 Hz

Ziel der Spracherkennung ist die Trennung dieser Funktionen und deren Auswertung.

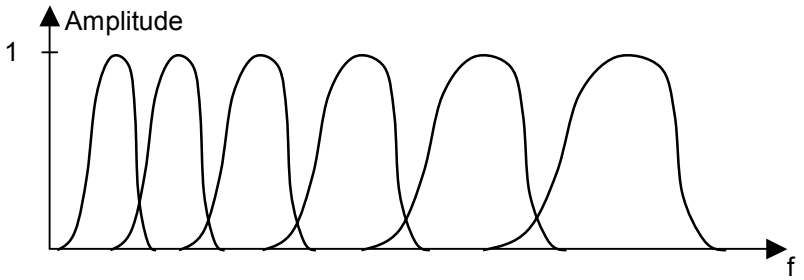
5.2. Gehörgang

- Ohrmuschel logarithmiert die Lautstärke über die Frequenz
- Schwingungen des Trommelfells werden mechanisch über Steigbügel und Ambos verstärkt
- Schwingungen werden über Schnecke auf basilar Membran übertragen, auf der feine Härchen (ca. 25000) sitzen, welche die Fourier-Transformation liefern
- Vorzugsfrequenzen werden durch Schneckengänge hervorgehoben

Verlauf der Empfindlichkeit:



Filterung des Signal über Härchen erfolgt durch:



5.3. Künstliche Spracherkennung

Es gibt 3 Verfahren:

- Cepstrum Analyse (klassisch)
- MEL-Cepstrum (Nico's Toolkit)
- Bark Cepstrum (Nico's Toolkit)

5.3.1. Klassiche Cepstrum Analyse

1. Diskretisierung des kontinuirlichen Sparchsignals: (Tool: Audacity)
Samplingrate gemäss Shannonschem Abtasttheorem:

$$2 \cdot \underbrace{f_G}_{Grenzfrequenz} = 2 \cdot 5{,}5kHz = 11kHz$$

$$f_n(t) = \sum \delta(n-t) \cdot \underbrace{\delta(t - n \cdot T)}_{Deltafunktion}$$

T = Zeitdauer eines Frames = $\frac{1}{f_G} = \frac{1}{11kHz} = 9ms$

2. Signal-Fensterung:
Sprachsignale werden im Bereich von 5 - 25ms als quasi Stationär betrachtet. Fenster mit Rechteck-Funktion ist ungeeignet, da an den Rändern Signalsprünge auftreten, die zu ausgeprägten Seitenbändern führen (bei Fourier-Transformation). Deshalb nimmt man Hammig- oder Hannig-Fenster, welche cos-Funktionen beinhalten:

Hannig-Funktion: $w_n = 0{,}5 - 0{,}5 \cdot \cos\left(\frac{2 \cdot \pi \cdot n}{N-1}\right)$

Hammig-Funktion: $w_n = 0{,}54 - 0{,}46 \cdot \cos\left(\frac{2 \cdot \pi \cdot n}{N-1}\right)$

n = Index des Samplewertes
N = Anzahl der Samplewerte im Zeitfenster t_w

$$\boxed{\Rightarrow f_n(t) = f_n \cdot w_n}$$

3. Wortgrenzen Detektion:
Gemittelte Leistung $P_m = {}^{1}\!/_{N} \cdot \sum (f_n(t))^2$

4. Diskrete Fourier-Transformation:

$$F_n = \sum f_n(t) \cdot e^{-\frac{j \cdot 2\pi \cdot n}{N}} = U(j \cdot \omega) \cdot V(j \cdot \omega) \cdot R(j \cdot \omega)$$

liefert Verteilung der Intensität über Frequenz, das sog. Spektrum.

maximale Frequenzauflösung ist $F_{\max} = \frac{F_A}{2}$

Intensität $F(j \cdot \omega) = \sqrt{(\mathrm{Re})^2 + (\mathrm{Im})^2}$

5. Quadrierung der Intensität:

$|F(j\cdot\omega)|^2$ liefert das Energiespektrum.

Dadurch werden kleine Signalwerte unterdrückt und große verstärkt.

6. Logarithmierung des Energiespektrums:

Logarithmierung liefert Separierung von Anregungsfunktion und Vokaltraktfunktion und Lippenabstrahlungsfunktion:

$$\log|F(j\cdot\omega)|^2 = \log\left(U(j\cdot\omega)\cdot V(j\cdot\omega)\cdot R(j\cdot\omega)\cdot U^*(j\cdot\omega)\cdot V^*(j\cdot\omega)\cdot R^*(j\cdot\omega)\right)$$
$$= \log U(j\cdot\omega)+\log V(j\cdot\omega)+\log R(j\cdot\omega)+\log U^*(j\cdot\omega)+\log V^*(j\cdot\omega)+\log R^*(j\cdot\omega)$$

7. Cepstral-Analyse:

$$CEP: \underbrace{JDFT}_{\substack{Inverse\ Fourier-\\ transformatik}}\left(\log|DFT(f(t))|^2\right)$$

Als Merkmals-Vektor werden die Koeffizienten 2-12 und höher herangezogen.

5.4. Time-Delay-Netze

Zweck: Zeitinvariante Mustererkennung.

Aufbau: Feed-Forward-Netz mit Zeitverzögerten Gliedern. Einzelnes Neuron der Eingabeschicht (Merkmal) wird um Zeitverzögerungen t+Δt, t+2Δt, ... erweitert. Ein Neuron der Verdeckten Schicht sieht nur einen kurzen Zeitabschnitt aus der Eingabeschicht z.B. t und t+3Δt (siehe Bild 5.9, 5.10).

Eigenschaften: Reduzierung der Gewichte, dadurch dass nur 3x16 Verbindungen (im Bild 5.10) zur gleichen Zeit aktiviert sind, obwohl Eingangsschicht vollständig mit verdeckter Schicht verbunden ist.

Lernverfahren: Modifiziertes BPG, bei dem alle gleichen Verbindungen der Zeitversetzten Muster gemittelt und die Gewichte mit diesem Mittelwert verändert werden.

$$\Delta w_{ij} = {}^{1}\!/_{N}\cdot\sum_{j=1}^{N}\Delta w_{ij}(t+j)$$

6. Genetische Algorithmen (GA)

- GA liefert absolute Mini- und Maxima
- Generierung von Programmen
- Auffinden von Formeln als Approximation an Messreihe
- Lösung des Topologie-Problems bei KNN's
- Verschlüsselung von Informationen (und Entschlüsselung)
- Ermittelung eine optimalen Spielstrategie

6.1. Verfahren und Elemente von GA's

Verfahren:

1. Codierung von Chromosomen
2. Initialisierung einer Population (100-10000)
3. Bewertung der Population und Berechnung der Fitness für jedes Chromosom
4. Heiratsschema
5. Reproduktion: Zeugung von Kindern mittels Crossover der Elternteil-Chromosome
6. Mutation innerhalb eines Gens
7. Selektion

6.1.1. Kodierung von Chromosomen

Beispiel 1:

$$f_{soll}(x) = x^2$$

Approximation mittels Polynomen:

$$y = f_{ist}(x) = a_1 x^{b_1} + a_2 x^{b_2} + \cdots + a_n x^{b_n}$$

a_i, b_i nennt man Gene.

Gene werden zu Chromosomen zusammengefügt:

a_1	b_1	a_2	b_2	...	a_n	b_n

Anstelle von Reell-Wertigen Genen a_1, b_1, ... bevorzugt man bei GA's binäre Zahlen: Sei z.B. a_1, $a_n \in$ [u, o] so ist:

$$a_{i,reell} = u + (o-u) \cdot \frac{a_{i,decimal}}{2^8 - 1}$$

d.h. aus dem Intervall werden nur 256 Stützstellen verwendet und zur Berechnung herangezogen.

Ergebnis:

0	0	1	2	0	0	...

Beispiel 2:
Approximation aus Datenpool von Funktionen.

Funktion	$1/x$	$x+Wert$	$x \cdot Wert$	e^x	$\sin(x)$	$\cos(x)$	$\sqrt{x}$	...
Kodierung	0001	0010	0011	0100	0101	0110	0111	...

Formel-Kodierer: $y = g(h(f(x)))$

Um eine Formel aus 3 Operatoren zu kodieren, benötigt man 6 Gene, d.h. 6 Bytes

Sei z.B. ein Chromosom:

0000 0011	1000 1111	0000 0010	1000 1100	0000 0111	xxxx xxxx

Interpretation

x * 4,98 x + 0,98 $\sqrt{x}$ (egal)

$\Rightarrow \sqrt{x \cdot 4{,}98 + 0.98}$ (Zahlen sind zufällig gewählt, siehe verfahren in Beispiel 1)

Jeder Formel-Kodierer hat eine entsprechende Grammatik.
Bezogen auf Beispiel 2:

- ungerades Byte enthält Operator.
- gerades Byte enthält Operanden.
- +, -, *, ... sind zweiwertige Operatoren.
- $\sqrt{x}, e^x$, ... sind einwertige Operatoren, bei denen das folgende Byte im Chromosom nicht berücksichtigt wird.

6.1.2. Initialisierung der Population

Die Population wird durch Zufallswerte initialisiert.
Die Ausgangspopulation von etwa 100 bis 10000 Chromosomen wird durch die Zufällige Auswahl von Operationen aus dem Funktions-Pool und Zufälligen Operanden erzeugt.
Jedes Chromosom wird anhand der Grammatik auf seine Gültigkeit überprüft.

6.1.3. Bewertung und Fitnessfunktion

Die Bewertung gibt an, wie gut ein Chromosom das Optimum (Sollwert) approximiert.
Die Fitness gibt an, wie sich aus Bewertung die Chance errechnet, sich in der nächsten Generation zu reproduzieren.
d.h.: Fitness = f(Bewertung)

Bewertungs-Funktionen:
Absolute Abweichung: $\sum_i |f_{soll}(x_i) - f_{ist}(x_i)|$

Quadratische Abweichung: $\sum_i (f_{soll}(x_i) - f_{ist}(x_i))^2$

Fitness-Funktionen:

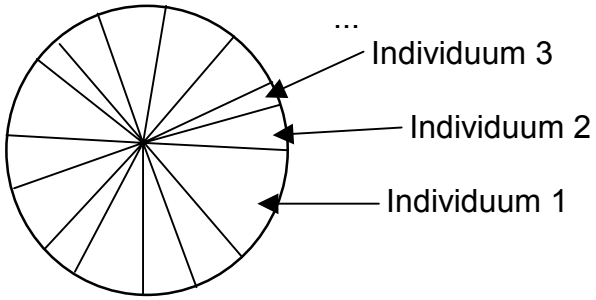
$$\frac{1}{\sum_i Bewertung(x_i)} \text{ oder } \frac{1}{\sum_i Bewertung(x_i) - b_i}$$

$\Rightarrow$ kleiner Fehler $\rightarrow$ hohe Fitness

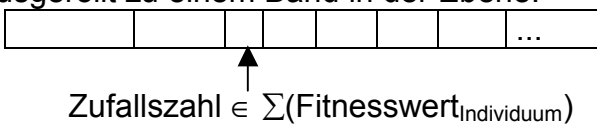
6.1.4. Heiratsschema

Ziel: Selektion der Auszuwählenden Chromosomen zu Chromosom-Paaren soll die Fittesten berücksichtigen, aber den weniger fitten eine Chance lassen, sich zu paaren.

Gängigstes Verfahren ist das Roulette-Wheel Verfahren:



Ein Segment des Rades repräsentiert die Fitness eines Individuums.
Ausgerollt zu einem Band in der Ebene:

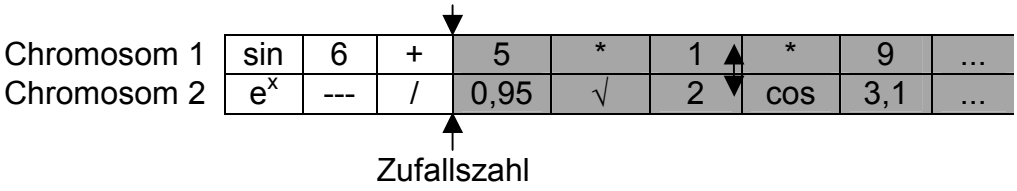



```
// Generiere Zufallszahl Z ∈ [0, Gesammt-Fitness]
Z = rand( "Gesammt-Fittnes" );
sum = 0;
i = 0;

while( sum < Z )
{
   sum += Fitness( i );
   ++i;
}

return i;

// Wähle 2. Individuum (!= 1. Individuum)
```

$\Rightarrow$ von 100 Chromosomen werden 50 Paare gebildet.

6.1.5. Reproduktion

Zuerst werden exakte Kopien der beiden Elternteile angelegt.
Diese Kopien werden einem Crossover unterzogen:



Einschränkung:
Gehört ein Elternteil zur Elite von n Individuen, dann wird Elternteil unverändert übernommen (Geklont).

6.1.6. Mutation

= Bitweise Änderung innerhalb eines Gens, falls Zufallszahl $\in$ [0, 1] unter anzugebender Mutations-Wahrscheinlichkeit liegt.

Das umklappen eines Bits bei binärer Codierung ist allerdings Problematisch, da jedes Bit eine andere Wertigkeit besitzt.

Auswege:

- Übersetzung von Binär- in Gray-Code:
  0 : 0000, 1 : 0001, 2: 0010, 3: 0011, 4: 0111, 5: 0110
  (maximal eine Stelle wird geändert)
- Man multipliziert die Mutations-Wahrscheinlichkeit mit der Wertigkeit des Bits (Meist verwendet)

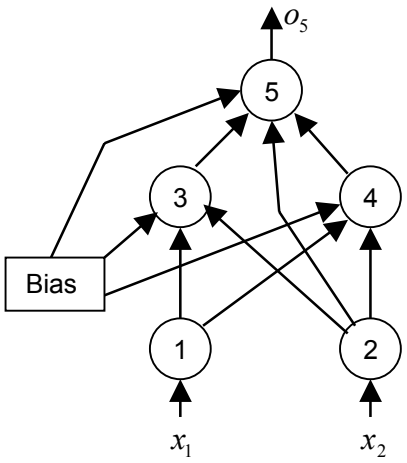
6.1.7. Selektion

(nach Selektionsschema)

- generational Replacement:
  Man ersetzt die aktuelle Population vollkommen durch die Nachkommen.
- Elitismus: (meist verwendet)
  Übernehme die n Besten (n vorgegeben) Chromosomen in die nächste Generation, und ersetze die anderen durch Nachkommen.
- Delete-n-last:
  Ersetze die n schlechtesten der aktuellen Population durch die n Nachkommen der aktuellen Population (die Fittesten).
- Delete-n-Elitismus:
  Ersetze n zufällig gewählte Chromosomen ohne Elite durch n Nachkommen der aktuellen Population.

6.2. Codierung und Decodierung einer KNN-Topologie

Kodierung:



Sei T() = Transfer-Funktion

$o_5(x_1, x_2) = \Omega(\vec{x}) = T_5(w_{05} \cdot 1 + w_{25} \cdot x_2 + w_{45} \cdot T_4(e_4) + w_{35} \cdot T_3(e_3))$

mit $e_4 = w_{04} \cdot 1 + w_{14} \cdot x_1 + w_{24} \cdot x_2$

und $e_3 = w_{03} \cdot 1 + w_{13} \cdot x_1 + w_{23} \cdot x_2$

Alphabet:

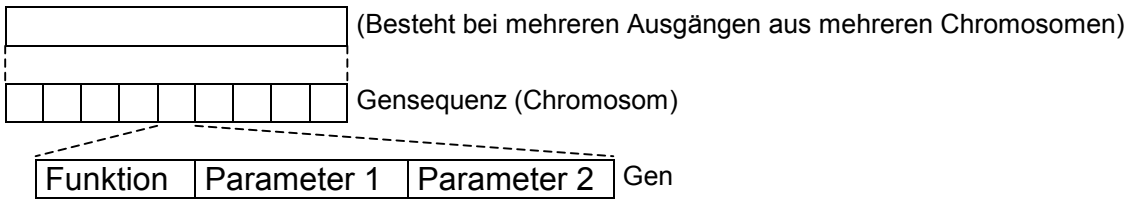
$x_1, \cdots x_n \in [0,1]$

$\Omega_1, \cdots \Omega_n \in [0,1]$

$w_{ij} = Parameter \in [-10,10]$

$T_i \in \{T_1(\ ), \cdots T_Z(\ ), x(\ )\} \quad x(\ ) = Identität$

Chromosom-Kodierung:



Chromosom aus Beispiel:

*	w_1	w_0	*	w_2	x_1	*	w_2	x_2	T_1	w_4	?	*	w_5	x_0	*	w_5	x_1	*	w_7	x_2

...

T_1	w_8	?	*	w_9	x_0	*	w_{10}	x_2	T_1	?	?

Grammatik:

Chromosom ist gültig wenn:

- mindestens 2 Knoten (2 T_i) vorkommen
- alle Inputs x_0, x_1, x_2 vorkommen
- vor jedem Knoten müssen mindestens ein Bias-Gen [* | w_i | x_0] und ein Bias-Gen [* | w_i | x_j] mit j ∈ {1, 2} vorkommen
- letztes Gen eines Chromosoms muss ein Knoten sein

Länge des Chromosoms wird beim Start des GA festgelegt und ist fix. Um variable Länge zu erhalten, sollen die Chromosomen bevorzugt werden, welche viele x() enthalten.
Dazu baut man in die Fitness-Funktion eine Theologen ein, der die Anzahl der Identitäts-Funktionen zählt und belohnt.

Theologe:
$h(k) = \alpha \cdot k$ oder $h(k) = 2^k$ oder $h(k) = e^k$
k = Anzahl der Identitäts-Funktionen
α = Prportionalitäts-Konstante

Fitness = $\frac{1}{1/n \cdot \sum (f_{soll} - f_{ist})^2} + h(k)$