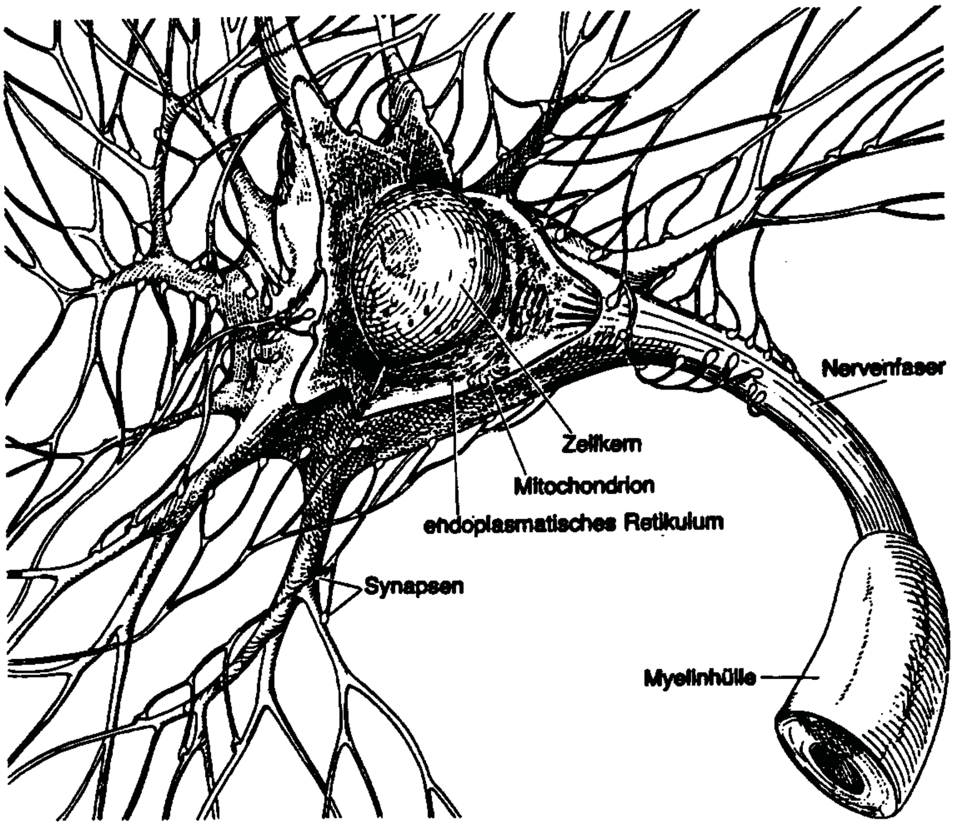
Einführung von Neuronalen Netzen und die Anwendung auf die Formel von Black-Scholes

Jens Fischer, 08.12.1998

---

1. Grundlagen Neuronale Netze
2. Architekturen
3. Wissensabbildung in Neuronalen Netzen
4. RBF-Netze und MLP-Netze
5. Anwendung: Black-Scholes-Formel

Neuron



- Zellkörper mit Zellkern
- baumartig verzweigte Dendriten
- Axon (Nervenfaser)
- Synapsen (Endköpfe des Axons)

Aufgaben

- *Zellkern*
  Eingangsignale verarbeiten und Ausgangsignale erzeugen

- *Axon (Nervenfaser)*
  Weiterleitung der Ausgangsignale

- *Dendriten*
  Eingangssignale aufnehmen

- *Synapsen*
  Kontaktstelle zwischen Axon und Dendriten anderer Zellen

Daten zur Biologie : Mensch

- Anzahl Neuronen: ca. 10^{11}

- Konnektivität: ca. 10^4 Synapsen/Neuron

- Länge eines Axons: einige mm bis 1 m

- Ruhepotential: -70 mV

- Nervenimpulsdauer: ca. 1 ms

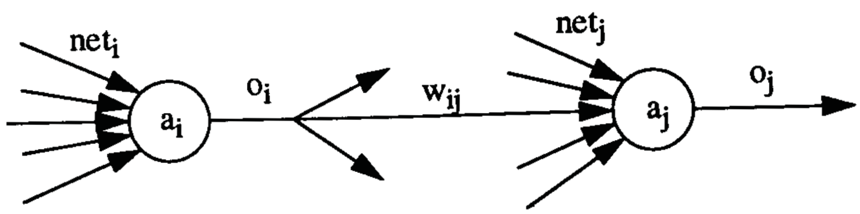
künstliche Neuronale Netze

„*Ein Neuronales Netz ist ein massiv parallel verteilter Prozessor, der eine natürliche Neigung zum Speichern und Verfügbarmachen von auf Erfahrung basierendem Wissen hat. Es ähnelt dem Gehirn in zwei Bereichen:*
1. Wissen wird durch einen <u>Lernprozeß</u> angeeignet
2. Die <u>Stärke von Verbindungen</u> zwischen den Neuronen, auch bezeichnet als Synapsen wird genutzt, um <u>Wissen zu speichern .</u>“
Aleksander und Morton 1990

<u>Bestandteile eines Neuronalen Netzes</u>

- Neuronen
- Verbindungsnetzwerk
- Propagierungsfunktion
- Lernregel

Neuronen



- *Aktivierungszustand* $a_i(t)$

  Grad der Aktivierung des Neurons

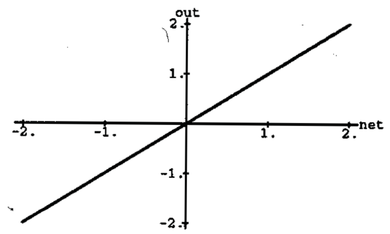
- *Aktivierungsfunktion* f_{act}
  errechnet neuen Aktivierungszustand

$$a_j(t+1) = f_{act}(a_j(t), net_j(t), \theta_j(t)) \qquad (1)$$

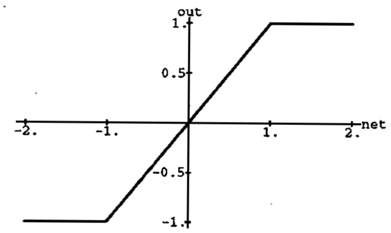
- *Ausgabefunktion* f_{out}
  bestimmt Ausgabe aus Aktivierung

$$o_j = f_{out}(a_j) \qquad (2)$$

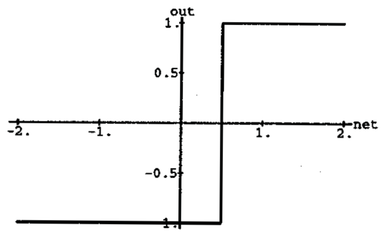
Neuronen
Aktivierungs- bzw. Ausgabefunktionen



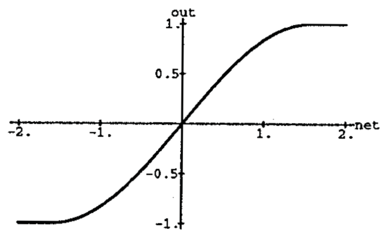
Identität ($o_j = net_j$)



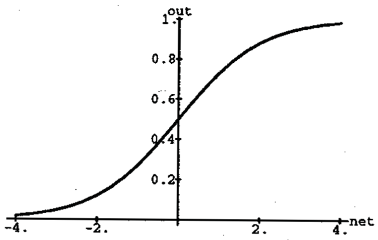
linear bis Sättigung



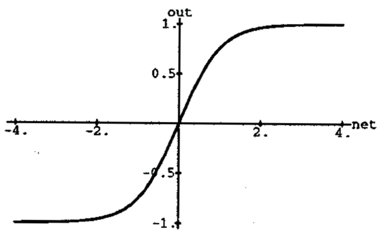
binäre Schwellenwertfunktion



sin(x) bis Sättigung



logistische Funktion (1 / (1+exp(-x)))



tanh(x) (tangens hyperbolicus)

Propagierungsfunktion $net_j(t)$

- *berechnet die Netzeingabe eines Neurons*

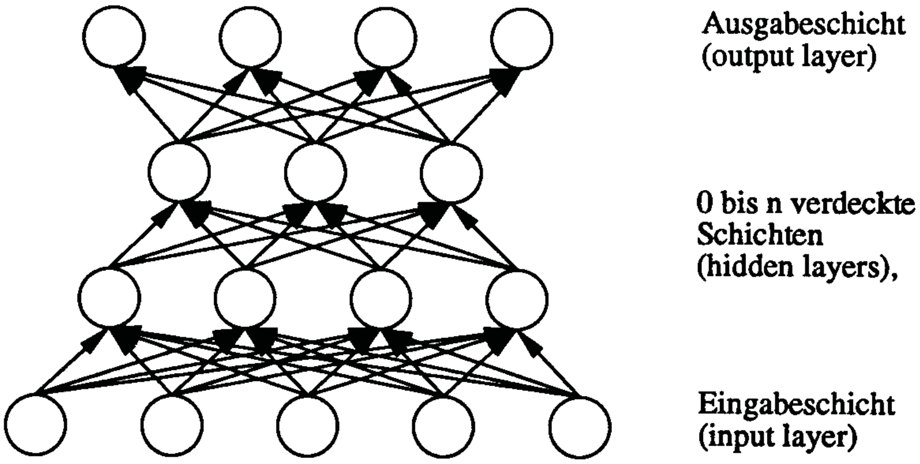
$$net_j(t) = \sum_i o_i(t) w_{ij}. \qquad (3)$$

Lernregel

- *Algorithmus, gemäß dem das Neuronale Netz lernt*

  (meistens Modifikation der Stärke der Verbindungen)

Verbindungsnetzwerk



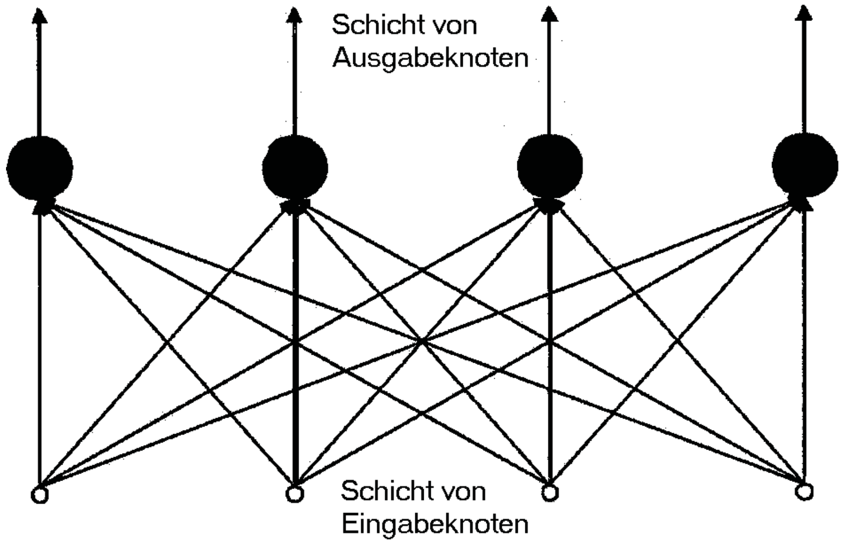
- *Gewichte* w_{ij}
  Faktor der Verbindung von Neuron i nach j

- *Eingabeschicht (input layer)*
  Neuronenschicht, die aus Umwelt Signale empfängt

- *Ausgabeschicht (output layer)*
  Neuronenschicht, die an die Umwelt Signale weitergibt

- *verdeckte Schichten (hidden layer)*
  übrige Neuronenschichten

Einschicht feedforward Netze

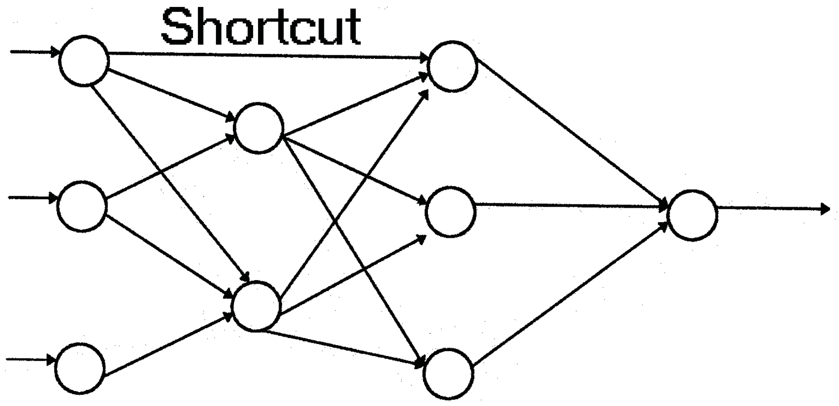


- *feedforward Netz*
  keine Daten werden an vordere Schichten zurückgegeben

- *nur eine Eingabe- und Ausgabeschicht*

- *Beispiel - linearer Assoziierer*
  Netz lernt bestimmte Eingaben mit bestimmten Ausgaben zu assoziieren (z.B. Ziffernerkennung)

Mehrschicht feedforward Netze

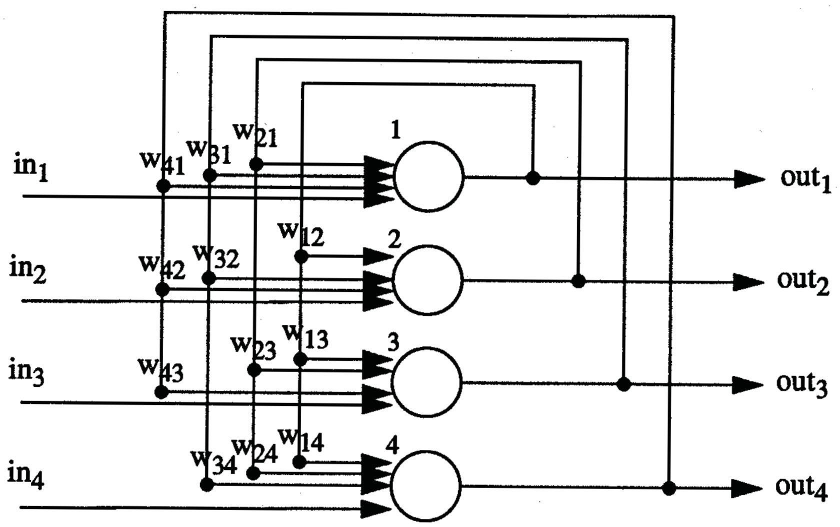


- *verdeckte Schichten vorhanden*
  Fähigkeit des Netzes hinsichtlich Analysemöglichkeiten und Speicherung von Daten nimmt zu

- *vollständig bzw. partiell Verbunden*
  jedes Neuron mit jedem der Nachfolgeschicht verbunden

- *Shortcuts*
  Verbindungen von Neuronen zwischen denen mindestens eine Schicht liegt

Rekursive Neuronale Netze



- *direct, indirect, lateral feedback*
  Eigenrückkopplung, Rückkopplung innerhalb einer Schicht

- *Beispiel - Hopfield-Netze*
  Gewichtsmatrix symmetrisch und Diagonalelemente gleich Null

Lernen

Parameter des Netzes anpassen

- *Verbindungen*

  Löschen und Entwicklung

- *Gewichte*

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

- *Modifikation der Schwellenwerte*

- *Modifikation der Funktionen*

- *Entwicklung und Löschen von Zellen*

Lernparadigmen

- **überwachtes Lernen (supervised learning)**
  Lernen mit Lehrer durch Input-Output Beispiele
  Lehrer bringt externes Wissen aus Umwelt mit
  Beispiele: Backpropagation, Delta-Lernregel

- **Lernen durch Verstärkung**
  (reinforcement learning)
  „*Wenn auf eine Aktion eines Lernsystems irgendein positiver Zustand folgt, wird die Tendenz des Systems, diese bestimmte Aktion erneut hervorzurufen, verstärkt. Andernfalls wird die Tendenz abgeschwächt.* “

- **Nichtüberwachtes Lernen**
  unsupervised learning
  kein Lehrer und keine Bewertung
  Beispiele: SOM

Hebbsches Lernen (D. Hebb 1949)

1. Wenn zwei Neuronen an den Enden einer Synapse simultan reagieren, so wird die Synapse gestärkt

2. Wenn andererseits zwei Neuronen an den Enden einer Synpase nicht simultan reagieren, so wird die Stärke der Synapse verringert.

Eigenschaften

- Zeitabhängigkeit
- Lokalität
- Interaktivität
- Korrelation

mathematische Formulierung

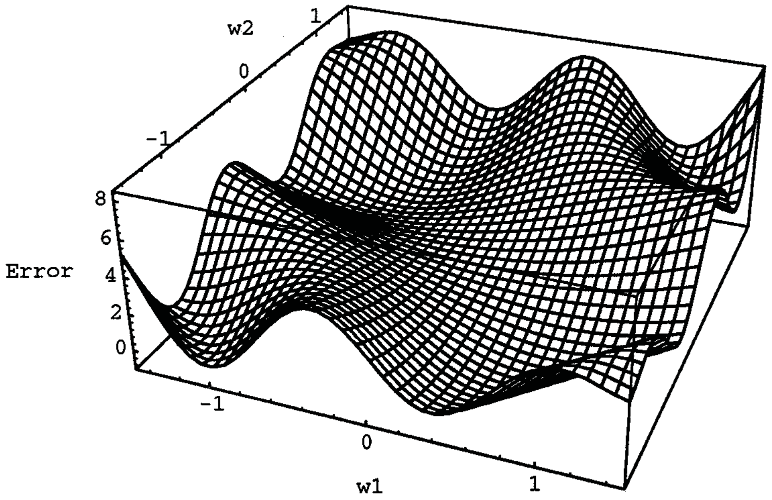
$$\Delta w_{ij} = \eta o_i a_j \tag{4}$$

Lernen durch Fehlerkorrektur

Gradientenabstiegsverfahren

- **Fehlerfunktion** $E(W) = E(w_1, ..., w_n)$

$$E = \sum_p E_p \quad \text{mit} \quad E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \qquad (5)$$



- **Idee: Gradientenverfahren**
Fehlerminimierung durch Gewichteänderung in Richtung des negativen Gradienten der Fehlerfunktion

$$\Delta W = -\eta \nabla E(W) \Rightarrow \Delta w_{ij} = -\eta \frac{\partial}{\partial w_{ij}} E(W) \qquad (6)$$

Gradientenabstiegsverfahren
Delta-Regel

- lineare Aktivierungsfunktionen

- eine Schicht trainierbarer Neuronen

Gewichtsänderung proportional zur Differenz δ_j der aktuellen Aktivierung und der erwarteten Aktivierung t_j

<u>mathematische Formulierung</u>

- **batch Trainingsverfahren (offline)**

$$\Delta w_{ij} = \eta \sum_p o_{pi}(t_{pj} - o_{pj}) = \eta \sum_p o_{pi}\delta_{pj} \qquad (7)$$

- **online Trainingsverfahren**

$$\Delta_p w_{ij} = \eta o_{pi}(t_{pj} - o_{pj}) = \eta o_{pi}\delta_{pj} \qquad (8)$$

Gradientenabstiegsverfahren
Backpropagation-Regel

- Netze mit mehreren Schichten

- semilineare Aktivierungsfunktionen

Fehler δ_{pj} kann mit Vorteil rekursiv berechnet werden, Fehler wird von vorne nach hinten zurückgegeben (backtracking)

<u>mathematische Formulierung</u>

$$\Delta_p w_{ij}(n) = \eta o_{pi} \delta_{pj}. \tag{9}$$

mit

$$\delta_{pj} = \begin{cases} o_{pj}(1 - o_{pj})(t_{pj} - o_{pj}) & \text{j Ausgabeneuron} \\ o_{pj}(1 - o_{pj}) \sum_k \delta_{pk} w_{jk} & \text{j verdecktes Neuron} \end{cases} \tag{10}$$

(hier logistische Aktivierungsfunktion verwendet)

Konkurrierendes Lernen
LVQ und SOM

- **Winner-takes-it-all-Prinzip**
  Neuronen konkurrieren um das Recht aktiv zu sein

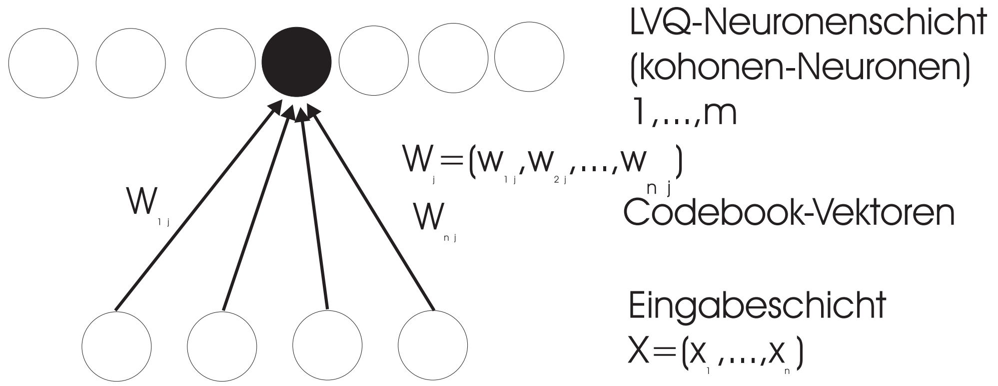
- **einschichtige Netze gleicher Neuronen**
  Kohonen-Neuronen (evtl. mit vorgeschalteter Eingabeschicht)

- **LVQ - Lernende Vektorquantisierung**
  überwachtes Lernverfahren, bei dem zu dem Eingabevektor die Klassenzugehörigkeit bekannt sein muß

- **SOM - Selbstorganisierende Karten**
  unüberwachtes Lernverfahren, Klasseninformation nicht notwendig

- **Mustererkennung und Klassifizierung**

LVQ - Lernende Vektorquantisierer



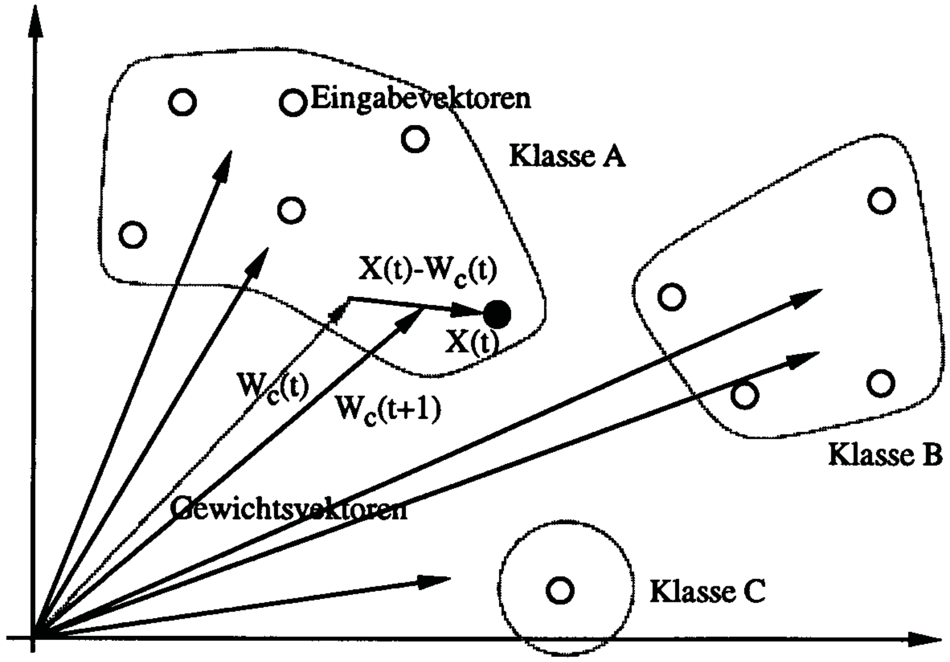
- **Codebook-Vektoren**

  Referenzvektoren, den Eingaberaum abdeckend verteilt
  (Gewichtsvektor eines Kohonenneurons)

- **Klasse**

  Eingabevektor wird mit allen CB-Vektoren verglichen, der ähnlichste gibt die Klassenzugehörigkeit an

LVQ - Lernende Vektorquantisierer
Lernalgorithmus



- **Gewinnerermittlung**
  Neuron dessen Gewichtsvektor W_C (CB-Vektor) zur Eingabe am ähnlichsten ist

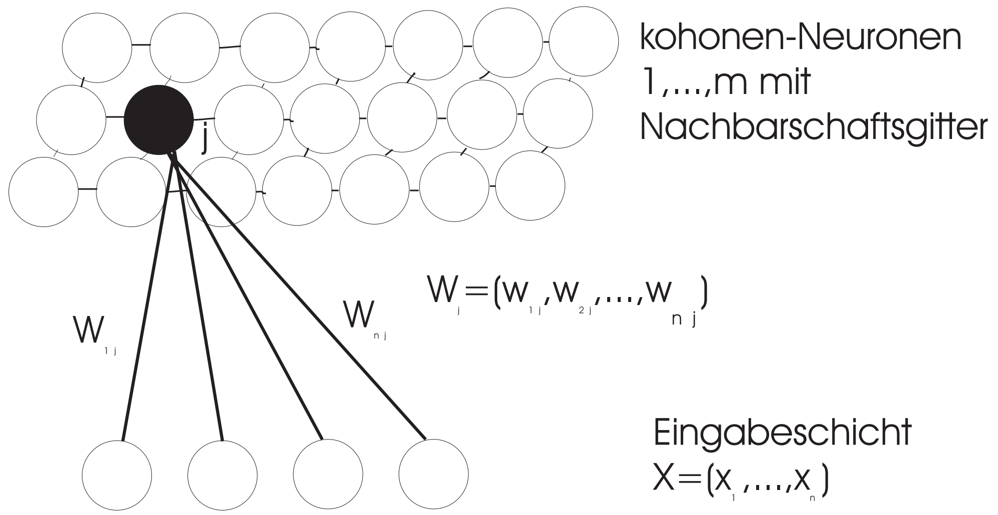
$$\|X - W_C\| = min_j(\|X - W_j\|) \qquad (11)$$

- **Modifikation von W_C nach LVQ1**

$$W_C(t+1) = \begin{cases} W_C(t) + \alpha(t)[X(t) - W_C(t)] & \text{Kl}(W_C)\text{=Kl(X)} \\ W_C(t) - \alpha(t)[X(t) - W_C(t)] & \text{Kl}(W_C)\neq\text{Kl(X)} \end{cases} \qquad (12)$$

$$W_j(t+1) = W_j(t) \quad \text{für alle j} \neq \text{c} \qquad (13)$$

SOM - Self Organizing Maps



- **Codebook-Vektoren**
  im n-dim. Eingaberaum verteilt wie bei LVQ

- **Nachbarschaftsbeziehung**
  Neuronen in m-dim. Gitter

- **elastisches Gitternetz**
  Änderung an einer Stelle beeinflußt die Umgebung je nach Abstand mit

- **topologieerhaltende Abbildung Clustering**
  vom n-dim. Eingaberaum auf m-dim. Gitter

SOM - Self Organizing Maps
Lernalgorithmus

- **Gewinnerermittlung**
  Neuron dessen Gewichtsvektor W_C zur Eingabe am ähnlichsten ist

$$\|X - W_C\| = min_j(\|X - W_j\|) \tag{14}$$

- **Gewichtsmodifikation**
  Gewichte in der topologischen Nachbarschaft von W_C werden ebenfalls mit verändert

$$W_j(t+1) = W_j(t) + \eta(t) h_{Cj}(t)[X(t) - W_j(t)] \tag{15}$$

- **Nachbarschaftsfunktion $h_{Cj}(t)$, Lernrate $\eta(t)$**

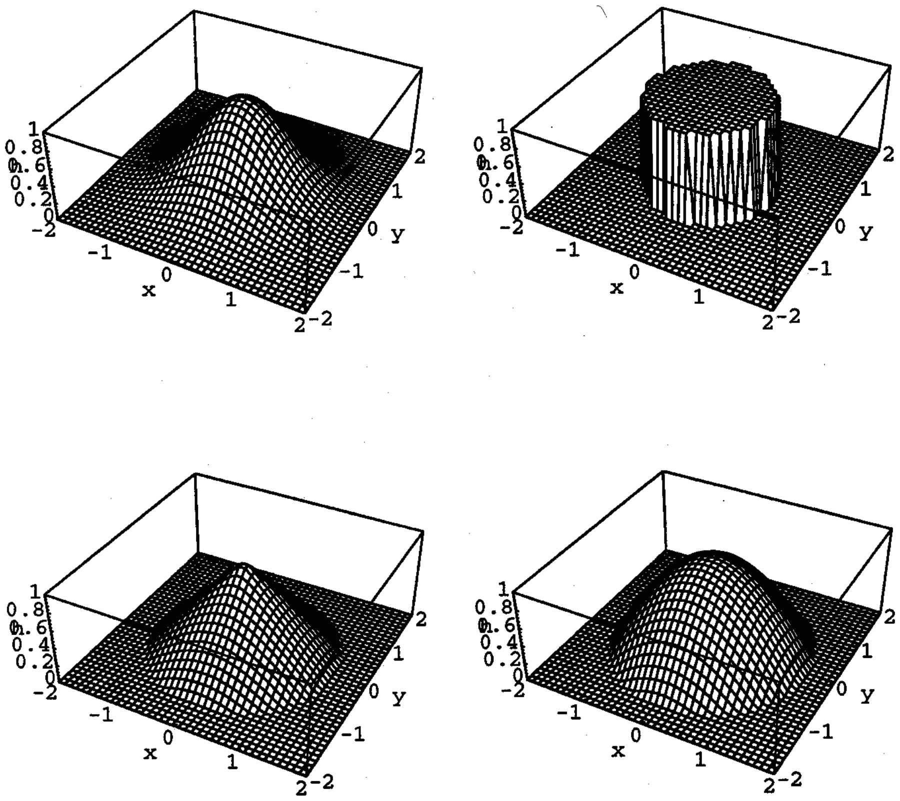
$$h_{Cj}(t) = h(\|r_C - r_j\|, t) = h(z, t) = \widehat{h}(z, d) \tag{16}$$

mit

$$z = \|r_C - r_j\| \to \infty \Rightarrow h_{Cj}(t) \to 0 \tag{17}$$

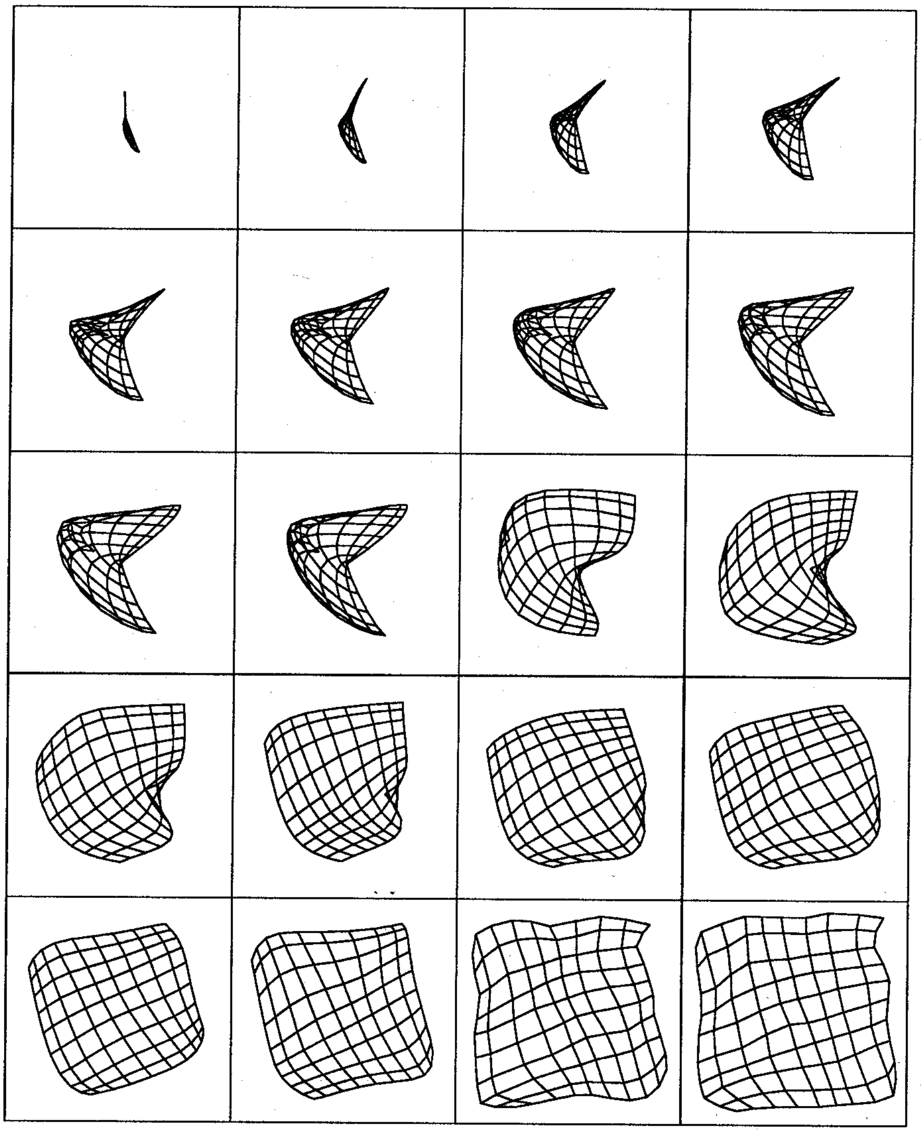
SOM - Self Organizing Maps

- **Distanzfunktionen**
  $h_{gauss}, h_{cylinder}, h_{cone}, h_{cos}$

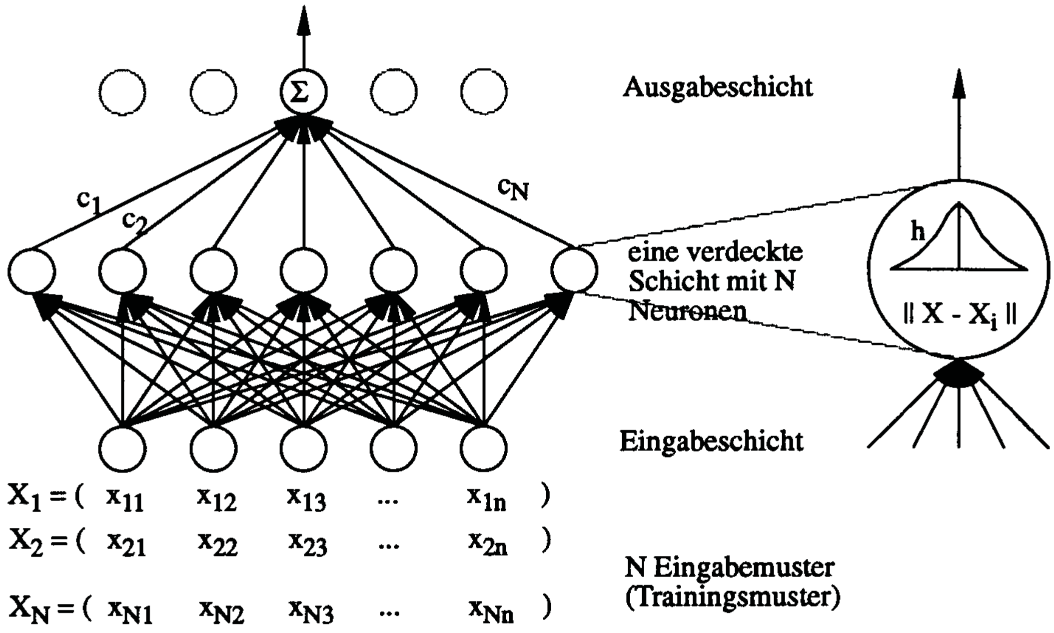


SOM - Self Organizing Maps

- **Entfalten einer Kohonenkarte**



Radial Basis Function - Netze



Eigenschaften

- **feedforward, eine verdeckte Schicht**

- **radialsymmetrische Aktivierungsfunktionen**

  Basisfunktionen zur Approximation mehrdimensionaler Funktionen anhand von Stützvektoren

  Gewichte direkt berechenbar und mit Backpropagation nachtrainierbar

Radial Basis Function - Netze
Lernalgorithmus

- **Approximationsansatz**
  mit Satz von Punkten nichtlineare Funktion f : $\Re^d \to \Re$ approximieren :

$$y_l = f(\vec{x}_l) + \epsilon_l \wedge E\left[\epsilon_l | \vec{x}_l\right] = 0. \tag{18}$$

- **Variationsansatz liefert**

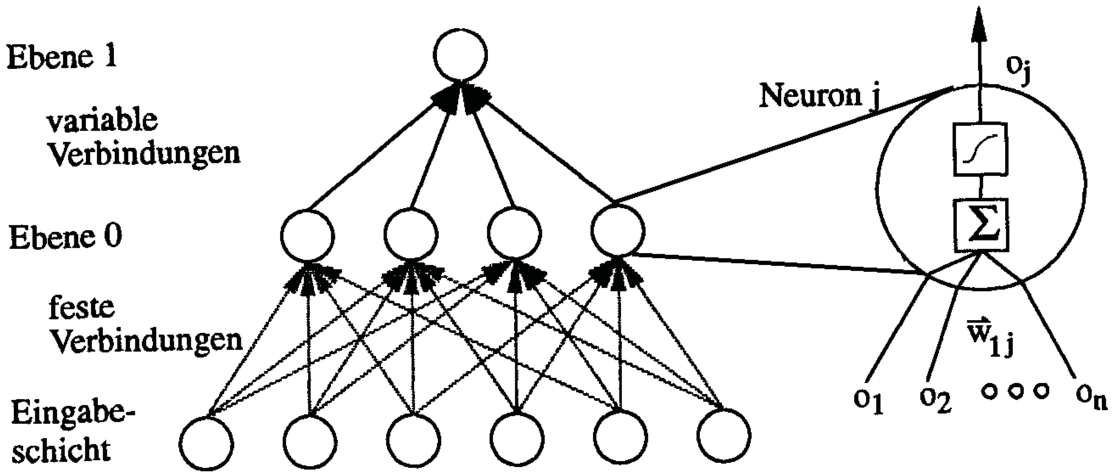
$$H[\hat{f}] = \sum_{i=1}^{T} \left(\|\hat{y}_l - \hat{f}(\vec{x}_l)\|^2 + \lambda \|P\hat{f}(\vec{x}_l)\|^2 \right) \tag{19}$$

- **Lösung des Minimierungsproblems**

$$\hat{f}(\vec{x}) = \sum_{i=1}^{k} c_i h_i \left((\vec{x} - \vec{z}_i) W W (\vec{x} - \vec{z}_i) \right) + \alpha_0 + \vec{\alpha_1} \vec{x}. \tag{20}$$

- **passende Basisfunktionen**
  Gauß-Funktionen: $e^{-\frac{x}{\sigma^2}}$

Multi-Layer-Perceptrons



- **nichtlineare Ausgabefunktionen**
  (z.B. sigmoide Funktion)

- **feedforward, mehrere verdeckte Schichten**

- **Backpropagation trainiert**

- **Funktionsapproximation**

$$\widehat{f}(\vec{x}) = h\left(\sum_{i=1}^{k} \delta_i h(w_{0i} + \vec{w}_{1i}\vec{x}) + \delta_0\right). \tag{21}$$

Anwendung
Black-Scholes-Formel

- **fairer Preis einer Calloption**

$$G(S(t), t) = S(t)N(d_1) - Ee^{-r(T-t)}N(d_2) \tag{22}$$

(aktueller Aktienpreis S(t), Erfüllungspreis E, Zeitraum der Option T-t, N(x) ist $\aleph$(0,1)-Gauss-Verteilung)

$$d_1 = \frac{\log(\frac{S(t)}{E}) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{23}$$

$$d_1 = \frac{\log(\frac{S(t)}{E}) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{24}$$

- **Kann Black-Scholes-Formel durch Neuronales Netz gelernt werden?**
  RBF und MLP-Netze können komplexe nichtlineare Funktionen approximieren!

- **Wie groß ist der Aufwand?**
  Schon 4 verdeckte Neuronen ergeben gutes Ergebnis!

Black-Scholes-Formel
Monte-Carlo-Simulation

- **Trainingspfad 2 Jahre Tagesaktienpreise**

  NYSE Aktie, Annahme Brownsche Bewegung

- **korrespondierender Optionspreispfad**

  CBOE Option, durch Black-Scholes

- **Netze trainieren**

  MLP - online Backpropagation

  RBF - Levenberg-Marquardt im batch mode

  Eingaben: $\frac{S(l)}{E}$ und $T - t$

- **Ergebnisse - 6 Monate-Test**

  Optionspreise: 99 Prozent out-of-sample R^2

  Delta-Hedging: bis zu 36 Prozent besser als Black-Scholes

Black-Scholes-Formel
S&P500 Daten

- **5 Jahresperiode**
  Aktienkursverlauf und Optionspreise in 10 mal 6 Monate, 6 Monatsperiode Training darauf Test 6 Monate

- **Black-Scholes**
  historische Volatilität aus Markt

- **Ergebnisse**
  Optionspreise: RBF 93.25, MLP 95.52, BS 84.76
  Delta-Hedging: bis zu 65 Prozent besser als Black-Scholes