

# **Neuronale Netze/ Soft Computing**

## **Teil 2**

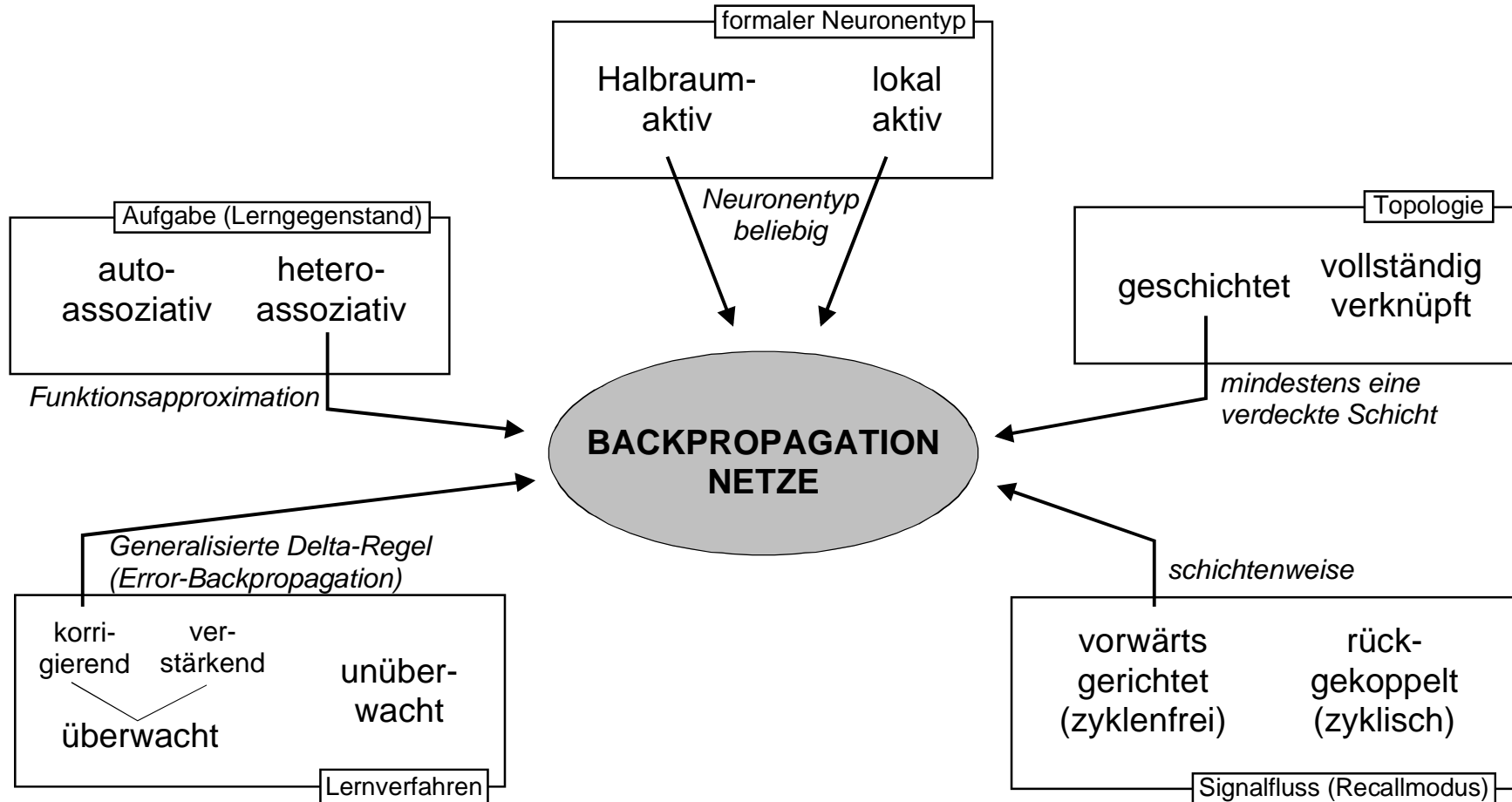
**BiTS, Wintersemester 2004/2005  
Dr. Stefan Kooths**

# Gliederung

---

1. Einführung und Einordnung
2. Neuronale Netze 1: Grundlagen
- 3. Neuronale Netze 2: Konzeption und Anwendung**
4. Neuro-Fuzzy-Systeme
5. Genetische Algorithmen
6. Zusammenfassung und Ausblick

# Backpropagation Netze

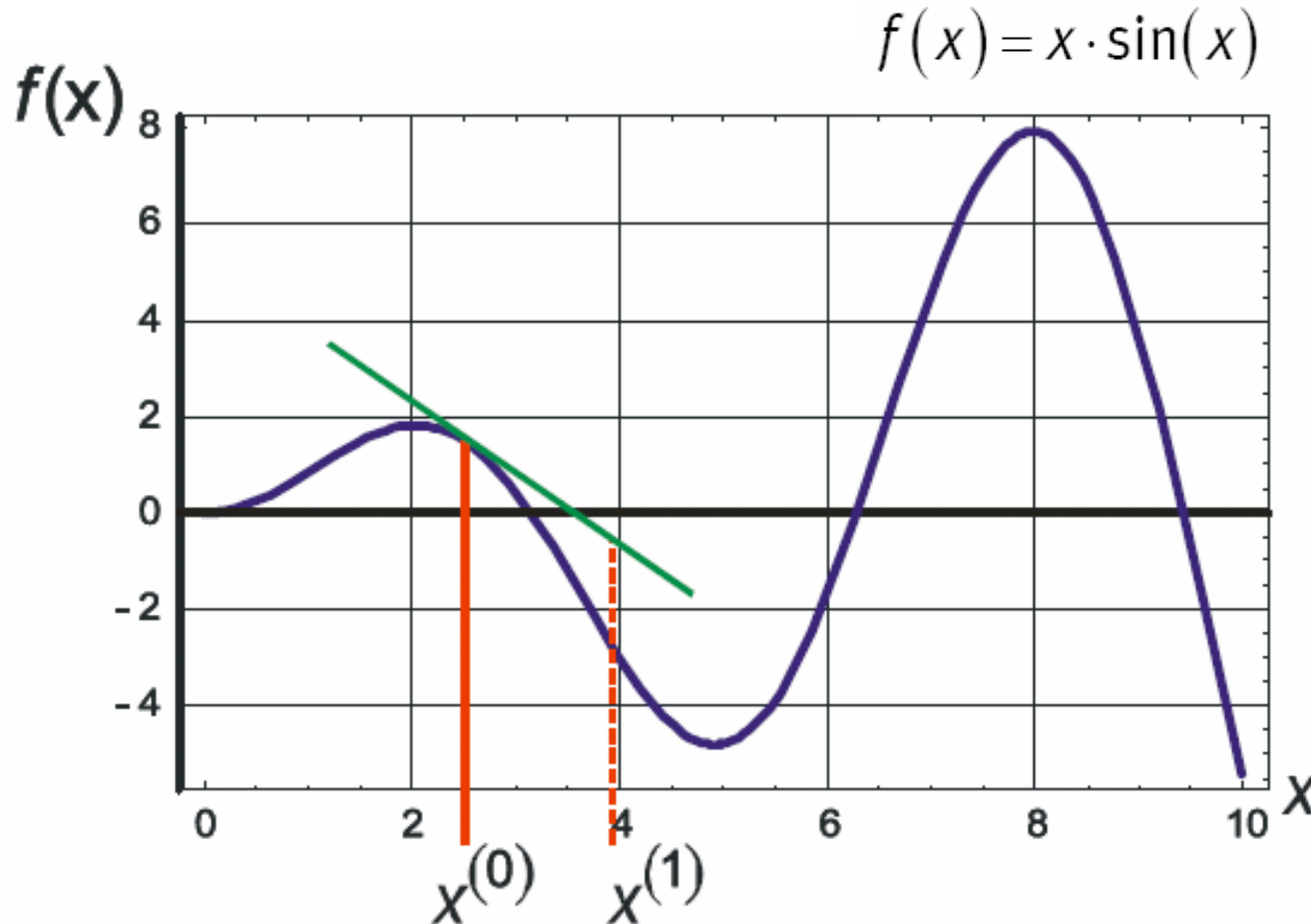


# Grundidee des EBP-Algorithmus

---

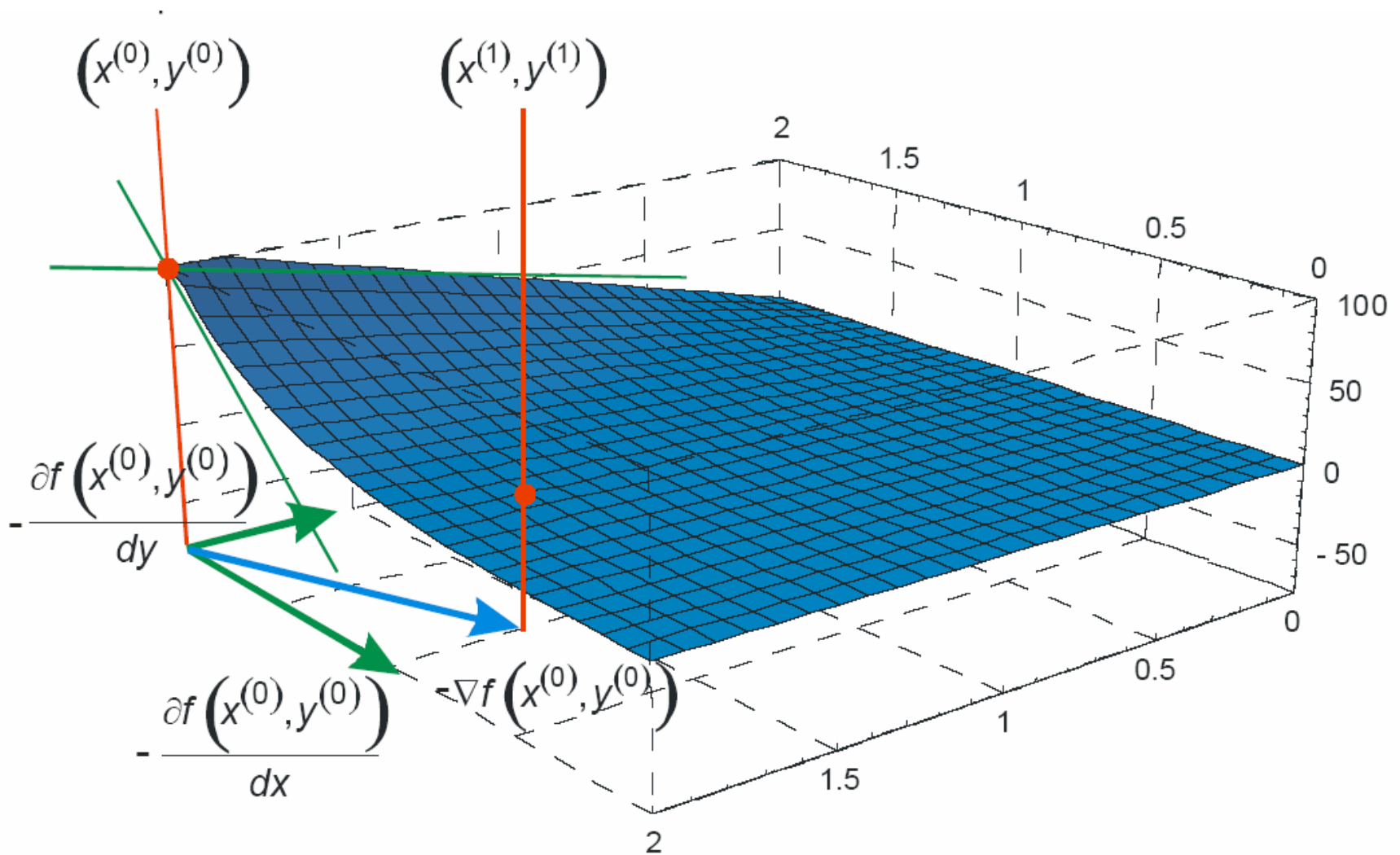
- alle Neuronen sind potentiell direkt (Outputneuronen) oder indirekt (versteckte Neuronen) am Zustandekommen des Netzfehlers beteiligt
- Stellgröße zur Änderung des Netzoutputs sind die Netzgewichte
- Netzgewichte spannen ein Fehlergebirge auf
- Gradientenabstieg im Fehlergebirge
- Schrittweite wird durch Lernrate bestimmt
- 3 Phasen
  - Forward Pass
  - Bestimmung des Netzfehlers
  - Backward Pass (Zurückpropagieren der Fehlersignale)

# Gradientenabstieg (eindimensional)



⇒ lineare Approximation des Funktionsverlaufs

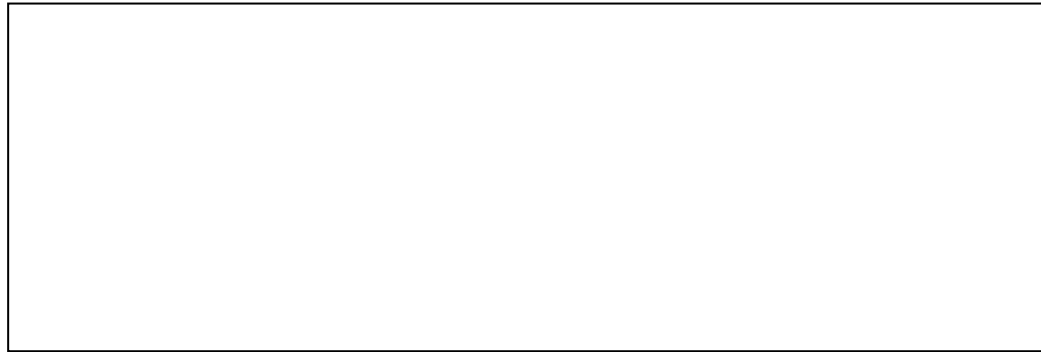
# Gradientenabstieg (zweidimensional)



# Gradientenabstieg für Gewichtsanzpassung

---

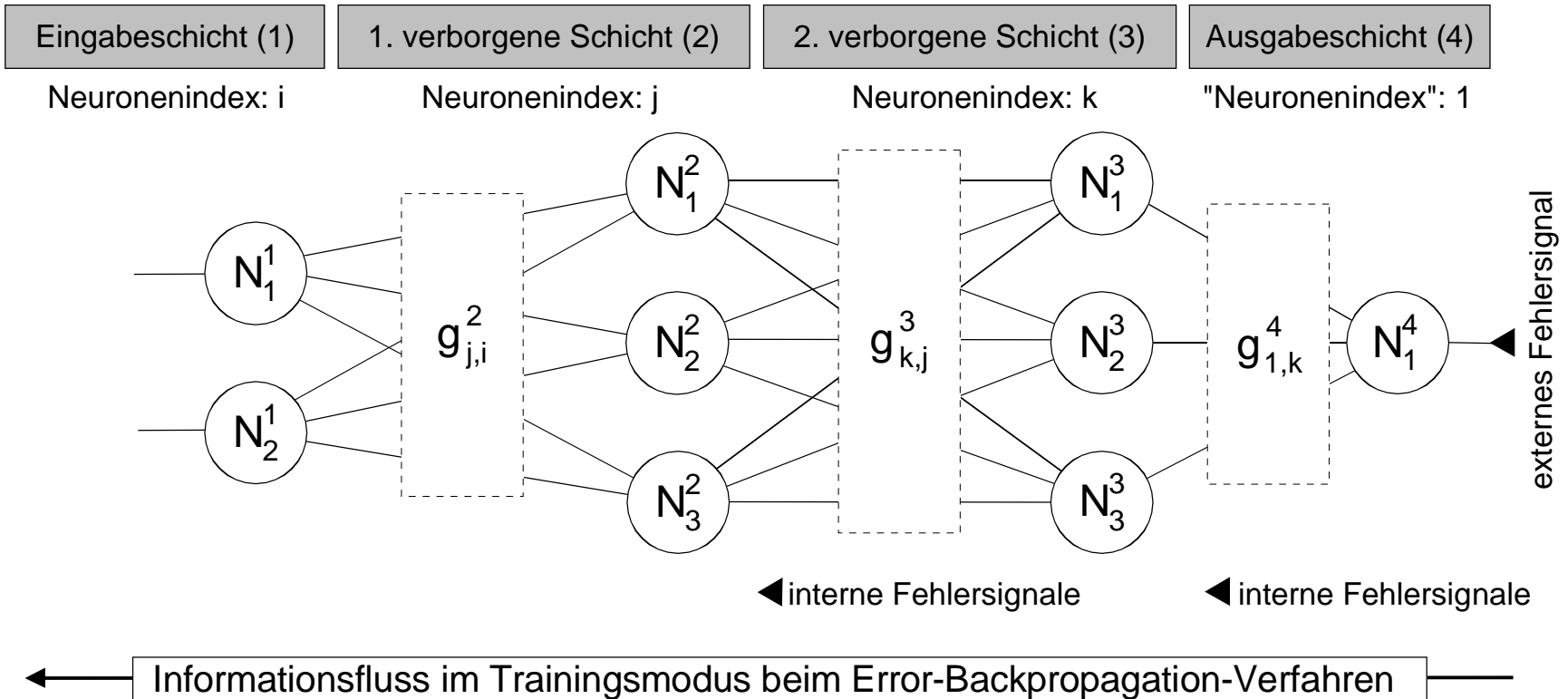
- Änderung der Gewichte erfolgt proportional zum jeweiligen negativen Grenzeinfluss eines Gewichtes auf den Netzfehler



⇒ Weg des steilsten Abstiegs im Fehlergebirge

$$E = 0,5 \cdot \sum [\text{out}(p) - y(p)]^2$$

# Vierschichtiges BP-Netz: 2-3-3-1



# EBP: Gewichtsanpassung für Ausgabeschicht

---

- Grenzeinfluss der Gewichte auf Netzfehler

- Ableitung des Nettoinputs nach den Gewichten

- Fehlersignal (marginale Reaktion von E auf netin)

# EBP: Gewichtsanpassung für Ausgabeschicht (Forts.)

---

- Ergebnis



- entspricht für Identität als Transferfunktion der Delta-Regel (Fehlersignal = musterbezogene Soll-Ist-Differenz des Netzes)

# EBP:

## Gewichtsanpassung für Schicht 3

---

- generalisierte Delta-Regel
- Reaktion des Netzfehlers auf Gewichtsänderung (mehrmalige Anwendung der Kettenregel)



- Einfluss des Neuronenoutputs auf den nachgelagerten Nettoinput (3. Faktor)



# EBP:

## Gewichtsanpassung für Schicht 3 (Forts.)

---

- Fehlersignal

- Reaktion des Nettoinputs auf Gewichtsänderung

- Ergebnis

# EBP:

## Gewichtsanpassung für Schicht 2

---

- Grenzeinfluss der Gewichte auf Netzfehler

- Ableitung des Nettoinputs nach den Gewichten

- Fehlersignal (marginale Reaktion von E auf netin)

# EBP:

## Gewichtsanpassung für Schicht 2 (Forts.)

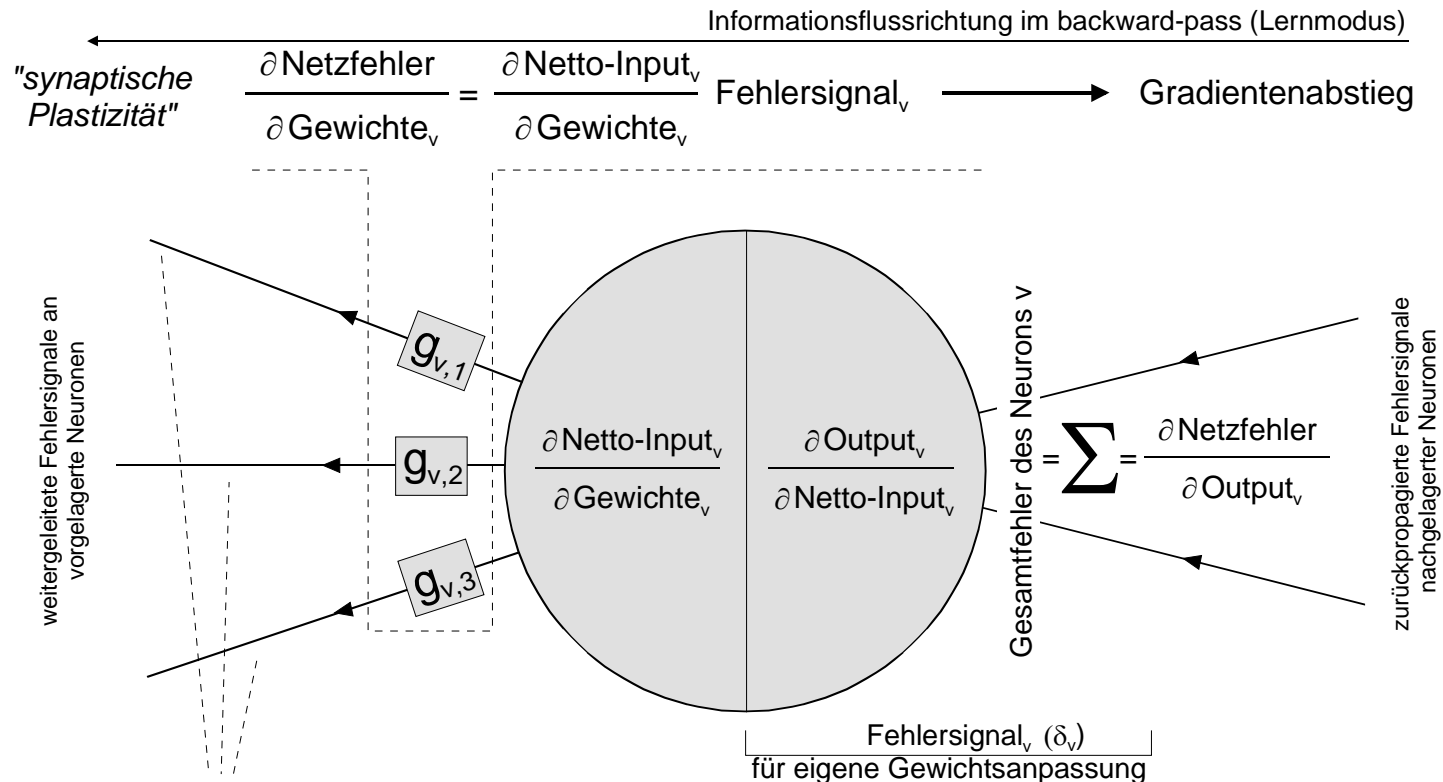
---

- Fehlersignal

- Reaktion des Nettoinputs auf Gewichtsänderung

- Ergebnis

# EBP-Schema für verdeckte Neuronen



## zurückpropagierte Fehlersignale:

$$\frac{\partial \text{Netto-Input}_v}{\partial \text{Output}_w \text{ vorgelagert } v} \text{ Fehlersignal}_v = \frac{\partial \text{Netto-Input}_v}{\partial \text{Output}_w \text{ vorgelagert } v} \frac{\partial \text{Output}_v}{\partial \text{Netto-Input}_v} \frac{\partial \text{Netzfehler}}{\partial \text{Output}_v}$$

# EBP-Flussdiagramm

Schritt 1: Zuerst Lesen des Eingabemusters und des assoziierten Ausgabemusters.  
 CONVERGE = TRUE

Schritt 2: Für die Eingabeschicht – weisen Sie als Netzeingabe jeder Einheit das entsprechende Element im Eingabevektor zu. Die Ausgabe für jede Einheit ist ihre Netzeingabe.

Lesen des nächsten Eingabemusters und dessen assoziiertes Ausgabemuster.

Schritt 3: Für die ersten Einheiten der verborgenen Schichten – berechnen Sie die Netzeingabe und -ausgabe:

$$net_j = w_0 + \sum_{i=1}^n x_i w_{ij}, o_j = \frac{1}{1 + \exp(-net_j)}$$

Wiederholen Sie Schritt 3 für alle nachfolgenden verborgenen Schichten.

Schritt 4: Für die Einheiten der Ausgabeschicht – berechnen Sie die Netzeingabe und -ausgabe

$$net_j = w_0 + \sum_{i=1}^n x_i w_{ij}, o_j = \frac{1}{1 + \exp(-net_j)}$$

Schritt 5: Befindet sich der Unterschied zwischen Ziel- und Ausgabemuster innerhalb des Toleranzbereichs?

if Nein CONVERGE = FALSE.

Schritt 6: Berechnen Sie für jede Ausgabeeinheit ihren Fehler:

$$\delta_j = (t_j - o_j) o_j (1 - o_j)$$

Schritt 7: Berechnen Sie für die letzte verborgene Schicht den Fehler jeder Einheit:

$$\delta_k = o_j (1 - o_j) \sum_k \delta_k w_{kj}$$

Wiederholen Sie Schritt 7 für alle nachfolgenden verborgenen Schichten.

Schritt 8: Aktualisieren Sie für alle Schichten die Gewichte jeder Einheit:

$$\Delta g_{j,i}^s = \eta \cdot \delta_j \cdot out_i^{s-1}$$

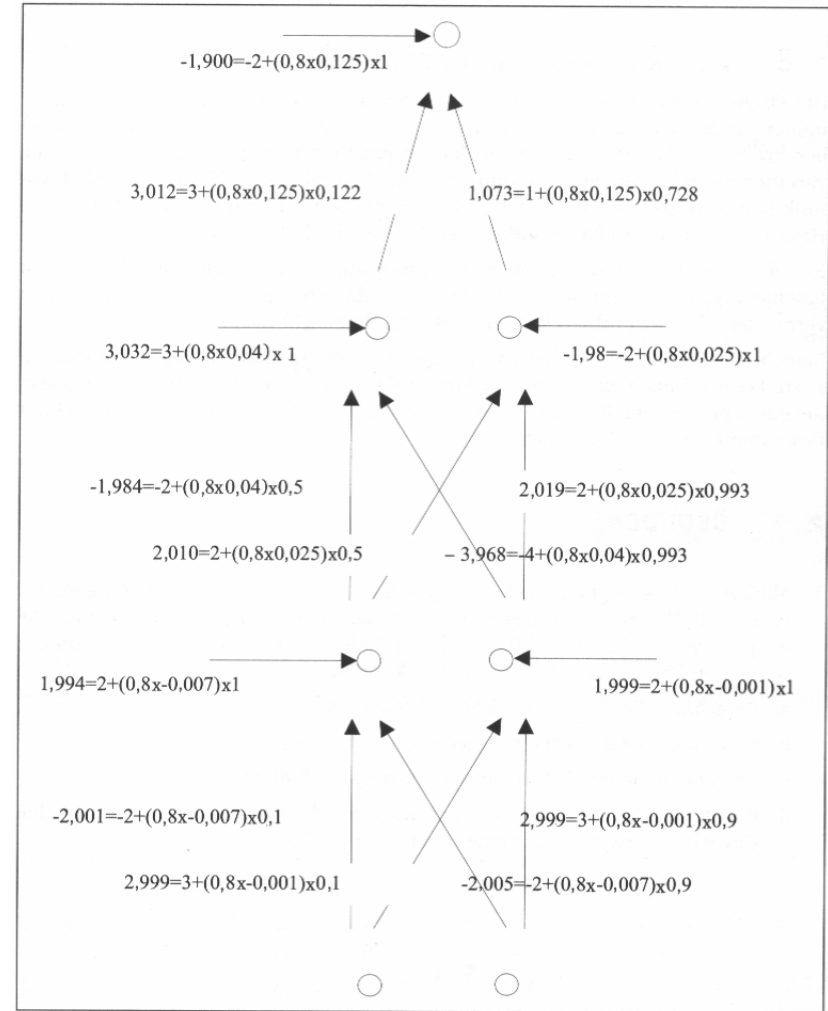
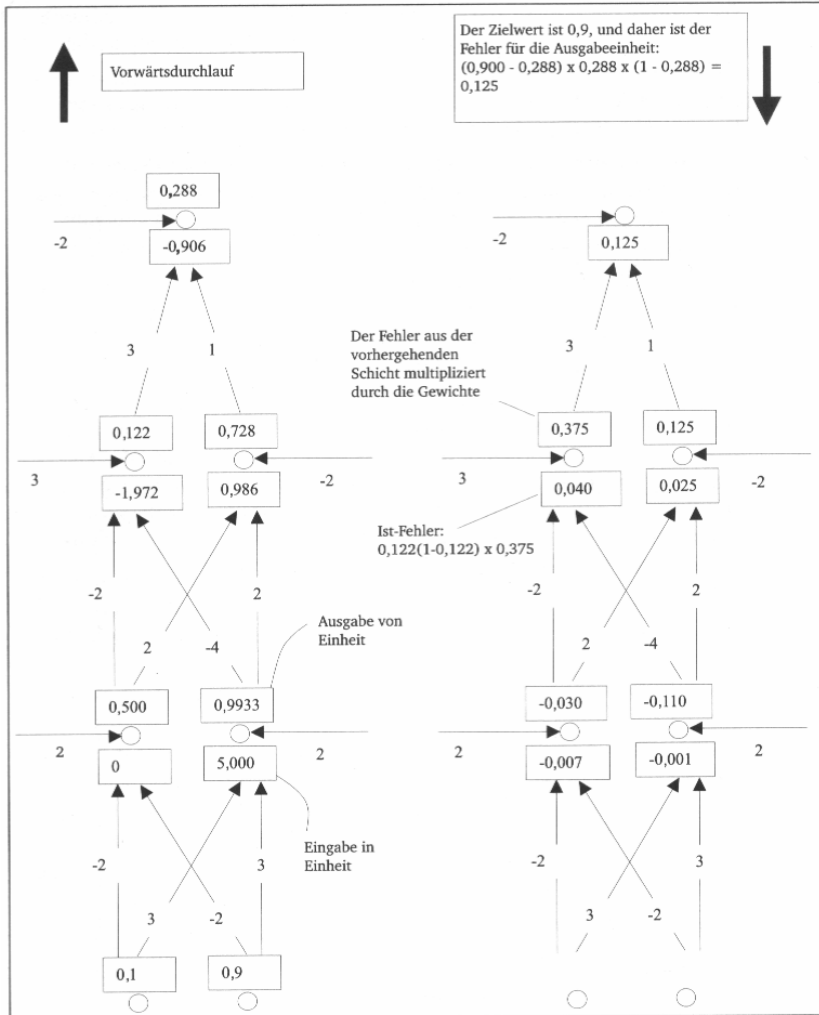
Letztes Muster?

CONVERGE == TRUE

STOP

nein

# Beispiel



# Vereinfachung

---

- approximative Ableitung der Transferfunktion über Differentialquotienten

# Praktische Hinweise

---

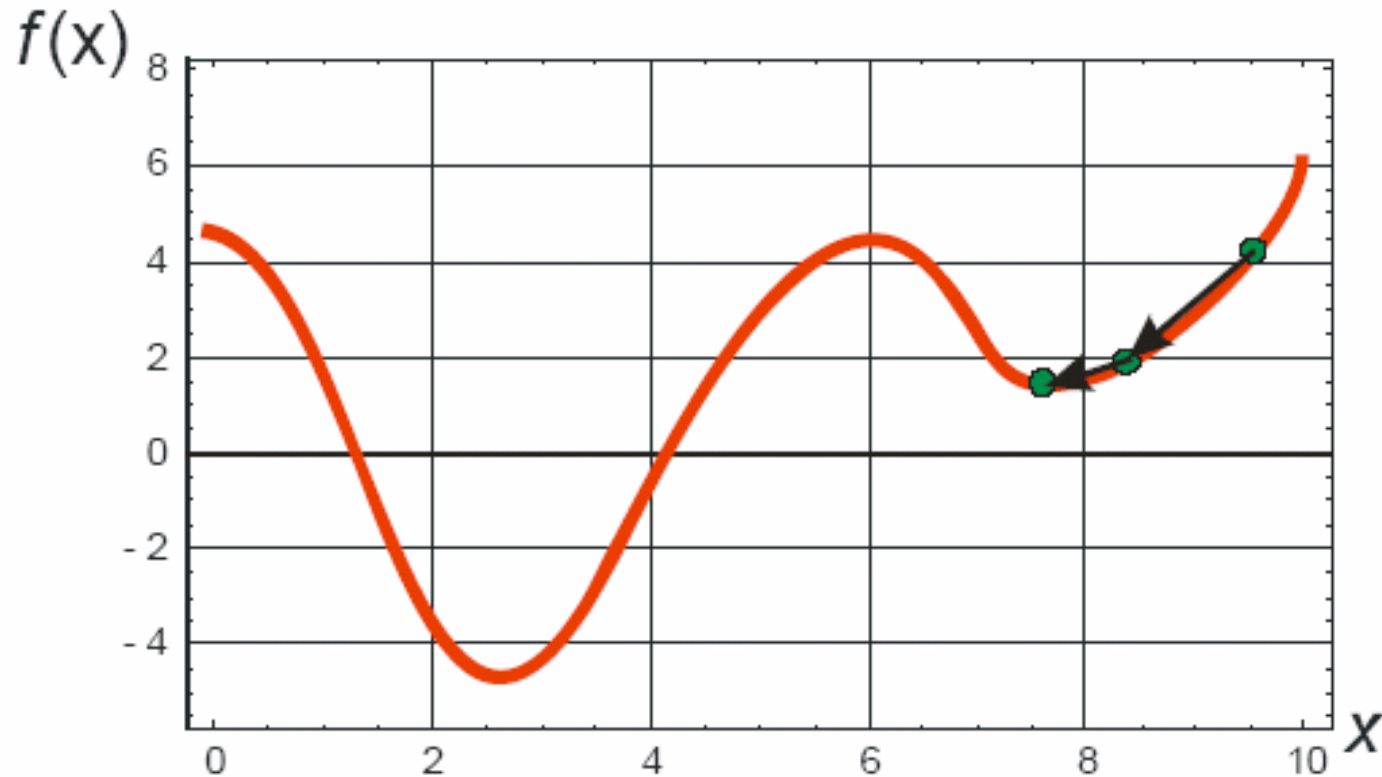
- Einfluss des Trainingsmodus
  - Batch-Modus liefert gute Abschätzung des Gradienten
  - Online-Modus
    - Reihenfolge der Lernbeispiele sollte zufällig sein
    - verringert Gefahr des Verharrens in lokalen Minima (s.u.)
- Gewichtsinitialisierung
  - $w \neq 0$
  - $w_i \neq w_j$  (symmetry breaking)
- Kodierung
  - Datenvorverarbeitung
  - Soll-Ausgabe muss erreichbar sein
  - Soll-Ausgabe nicht als 0-1-kodieren (Sättigung vermeiden)

# Mögliche Probleme des EBP-Algorithmus

---

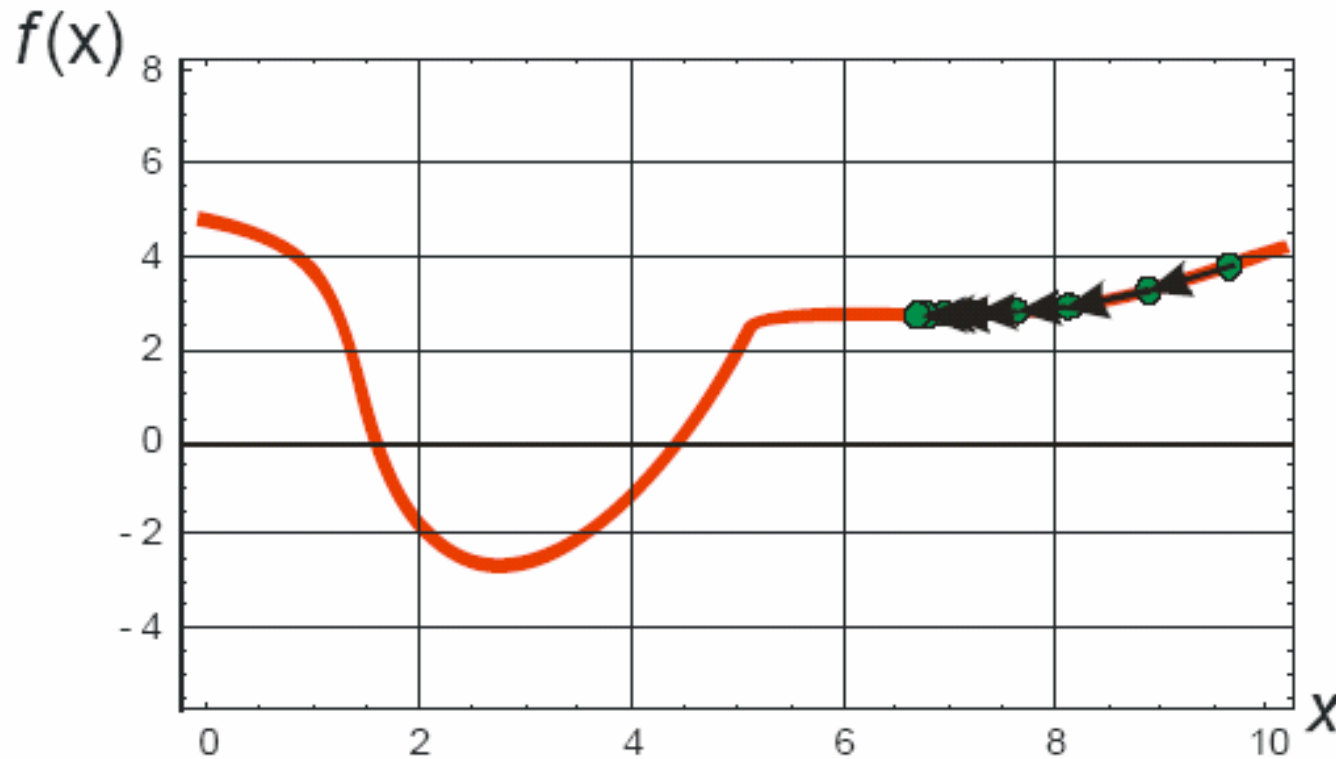
- grundsätzlich: keine Garantie für Konvergenz
- lokale Minima
- Plateaus
- Oszillation
- Verlassen guter Minima

# EBP-Probleme: Lokale Minima



- kleine Lernrate
- zufällige Initialisierung der Gewichte
- mehrere Trainingsdurchläufe

# EBP-Probleme: Plateaus



- Gefahr des Lernstillstands
- „Gedächtnis“ für letzte Gewichtsänderung  
⇒ Momentum-Term

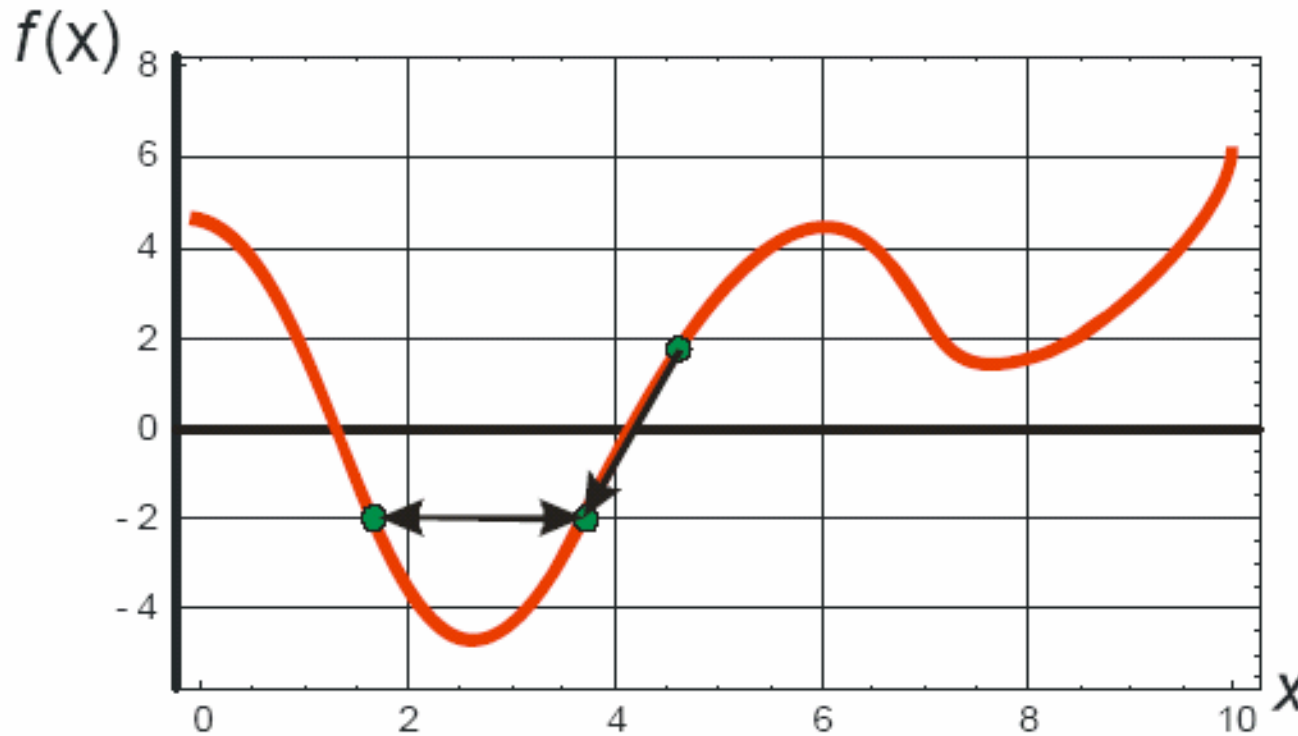
# Momentum-Term

---

$$\Delta w_t = \eta dE/dw + \alpha \Delta w_{t-1}$$

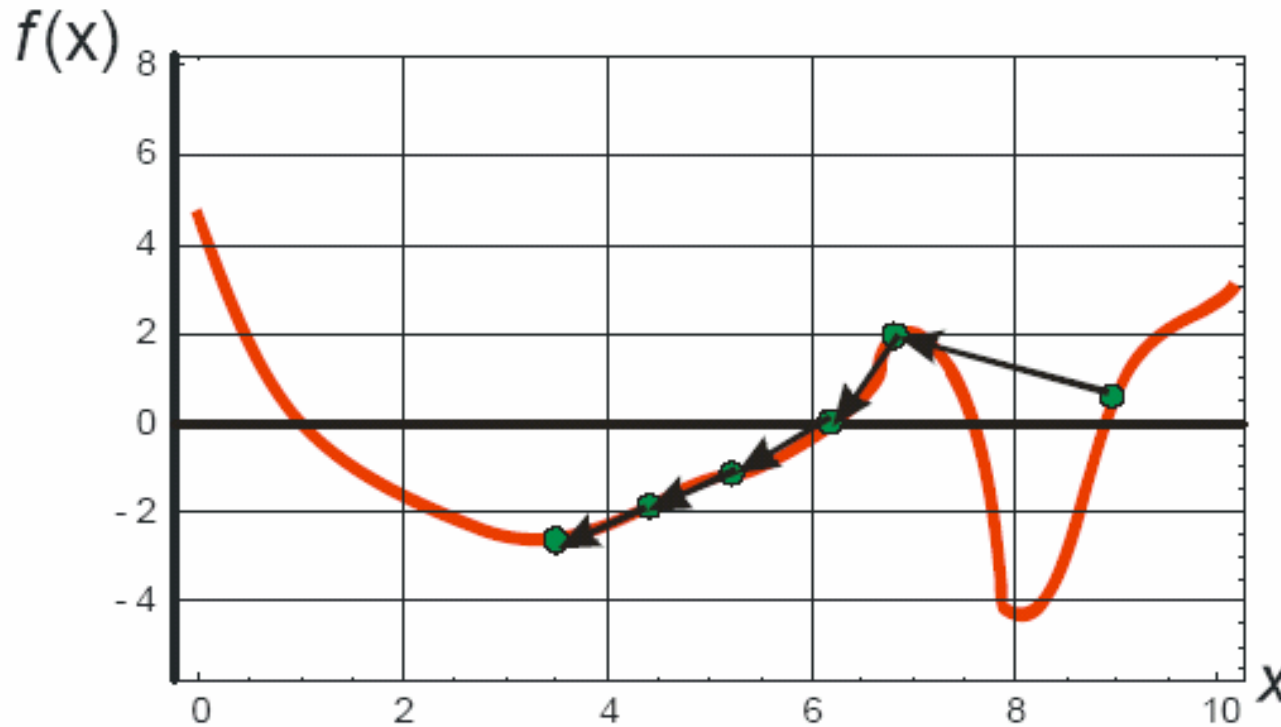
- exponentiell gewichtete Zeitreihe der vergangenen Gewichtsänderungen (Gradienten)
- für Konvergenz muss gelten:  $0 < \alpha < 1$
- gleiche VZ aufeinander folgender Gradienten
  - ⇒ Gewichtsänderung beschleunigt sich (Anhebung der Schrittweite)
- alternierende VZ aufeinander folgender Gradienten
  - ⇒ Gewichtsänderung wird gebremst (Reduktion der Schrittweite)

# EBP-Probleme: Oszillation



- Lernrate (= Schrittweite) verkleinern
- Momentum-Term

# EBP-Probleme: Verlassen guter Minima



- Lernrate (= Schrittweite) verkleinern

# Lernrate: Diskussion

---

- **Konsequenzen einer großen Lernrate**
  - bewirkt große Sprünge auf der Fehleroberfläche
  - Flache Plateaus werden schneller durchlaufen
  - Minima werden eventuell übersprungen oder verlassen
  - Gefahr von Oszillation nimmt zu
- **Konsequenzen einer verkleinerten Lernrate**
  - Geringfügige Bewegung auf der Fehleroberfläche.
  - Minima werden selten übersprungen oder verlassen
  - Komplexe Daten werden besser gelernt
  - Eine große Datendichte wird besser bewältigt
  - Die Umgebung eines Minimums wird nicht mehr verlassen
  - Oszillation wird immer unwahrscheinlicher.
  - Auf flachen Plateaus kann das Verfahren stagnieren.

# Weitere Modifikationen des BP-Algorithmus

---

- Flat-Spot Elimination
- Weight Decay
- Delta-Delta-Regel
- Delta-Bar-Delta-Regel
- SuperSAB-Regel
- Manhattan-Training
- Quickprop
- ...

# Flat-Spot Elimination

---

- Reaktion auf Fälle, in denen die Ableitung der Transferfunktion praktisch null ist
  - Sättigungsbereiche der sigmoiden Funktion
  - Neuron ist stark „an“ oder stark „aus“
- Verschiebung der Ableitung um einen konstanten Wert nach oben

# Weight Decay

---

- Strafterm für große Gewichte in Fehlerfunktion
- große Gewichte
  - biologisch unplausibel
  - erschweren die Generalisierungsfähigkeit des Netzes
  - bewirken steiles und zerklüftetes Fehlergebirge (Gefahr von großen Sprüngen und Oszillationen nimmt zu)
- erweiterte Fehlerfunktion ( $0,005 < d < 0,03$ )

(1)

- Gewichtsanzpassungsregel

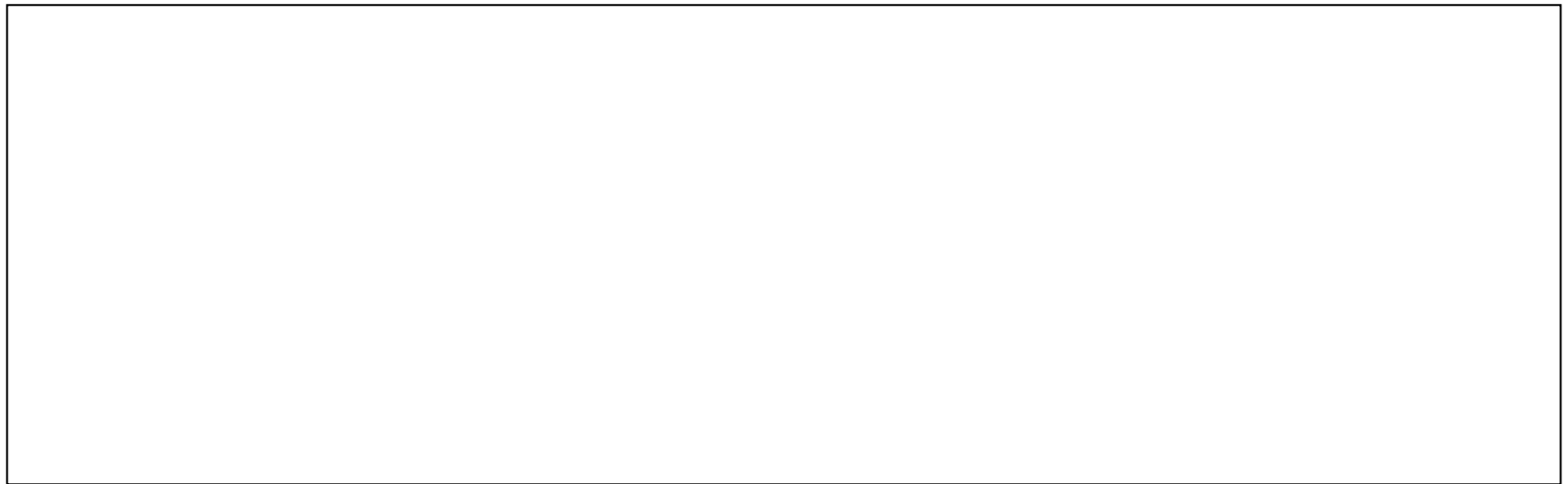
(2)

# Delta-Delta-Regel

---

- Ausgangspunkt: Lernrate ist an der Fehlerentwicklung beteiligt
- individuelle Lernraten für jedes Gewicht
- Lernrate „erfolgsabhängig anpassen“

(1)



# Wirkung der Delta-Delta-Regel

---

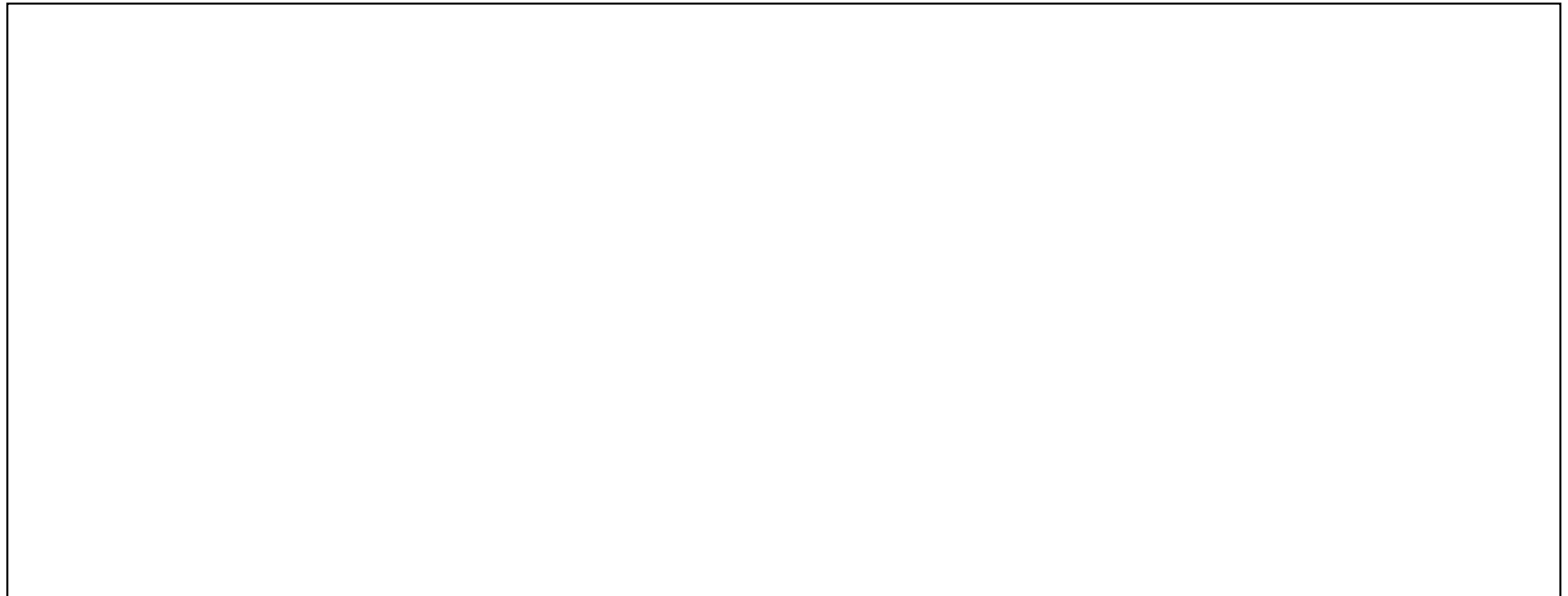
- Vergleich der Vorzeichen des Gradienten für aufeinander folgende Lernschritte
- Fall 1: gleiches Vorzeichen
  - Lernrate wird vergrößert
  - BP wird in dieser Richtung schneller
- Fall 2: ungleiches Vorzeichen
  - Lernrate wird verringert
  - BP wird langsamer
- Problem: Anpassung der Lernrate kaum steuerbar

# Delta-Bar-Delta-Regel

---

- „Gedächtnis“ für bisherige Gewichtsänderungen

(1)



# Wirkung der Delta-Bar-Delta-Regel

---

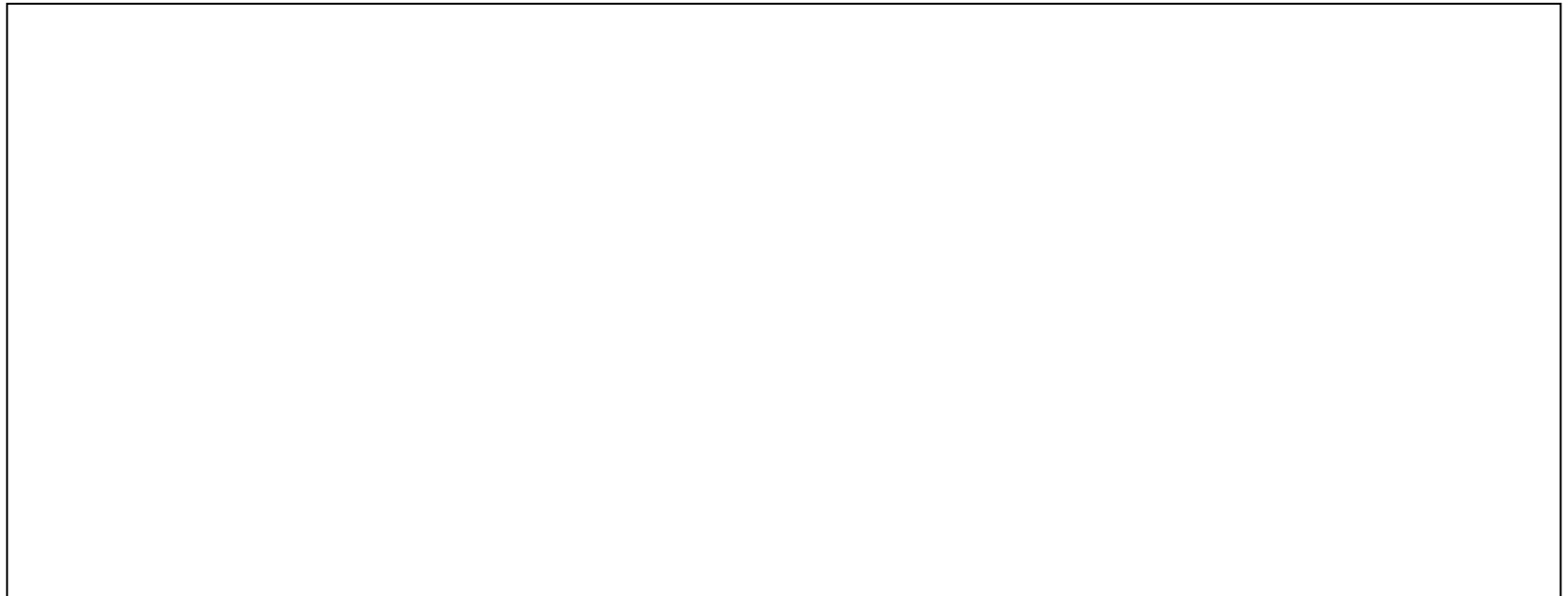
- Vorzeichenvergleich von D und S
- Fall 1: gleiches Vorzeichen
  - Erhöhung der Lernrate um konstanten Wert
  - Verhinderung eines zu starken Anstiegs (kontrollierte Erhöhung)
- Fall 2: ungleiches Vorzeichen
  - Rückgang der Lernrate um konstanten Prozentsatz
  - Abschmelzen verhindert negative Lernraten
- Nachteil: erhöhter Speicherbedarf gegenüber Standard-BP

# SuperSAB-Regel

---

- ähnlich wie Delta-Bar-Delta-Regel, aber exponentielle Lernratenanpassung in beiden Richtungen (multiplikative Änderung der Lernrate)

(1)



# Manhattan-Training

---

- nur Vorzeichen, nicht Wert des Gradienten wird berücksichtigt (nur Richtungsinformation wird genutzt)

(1)



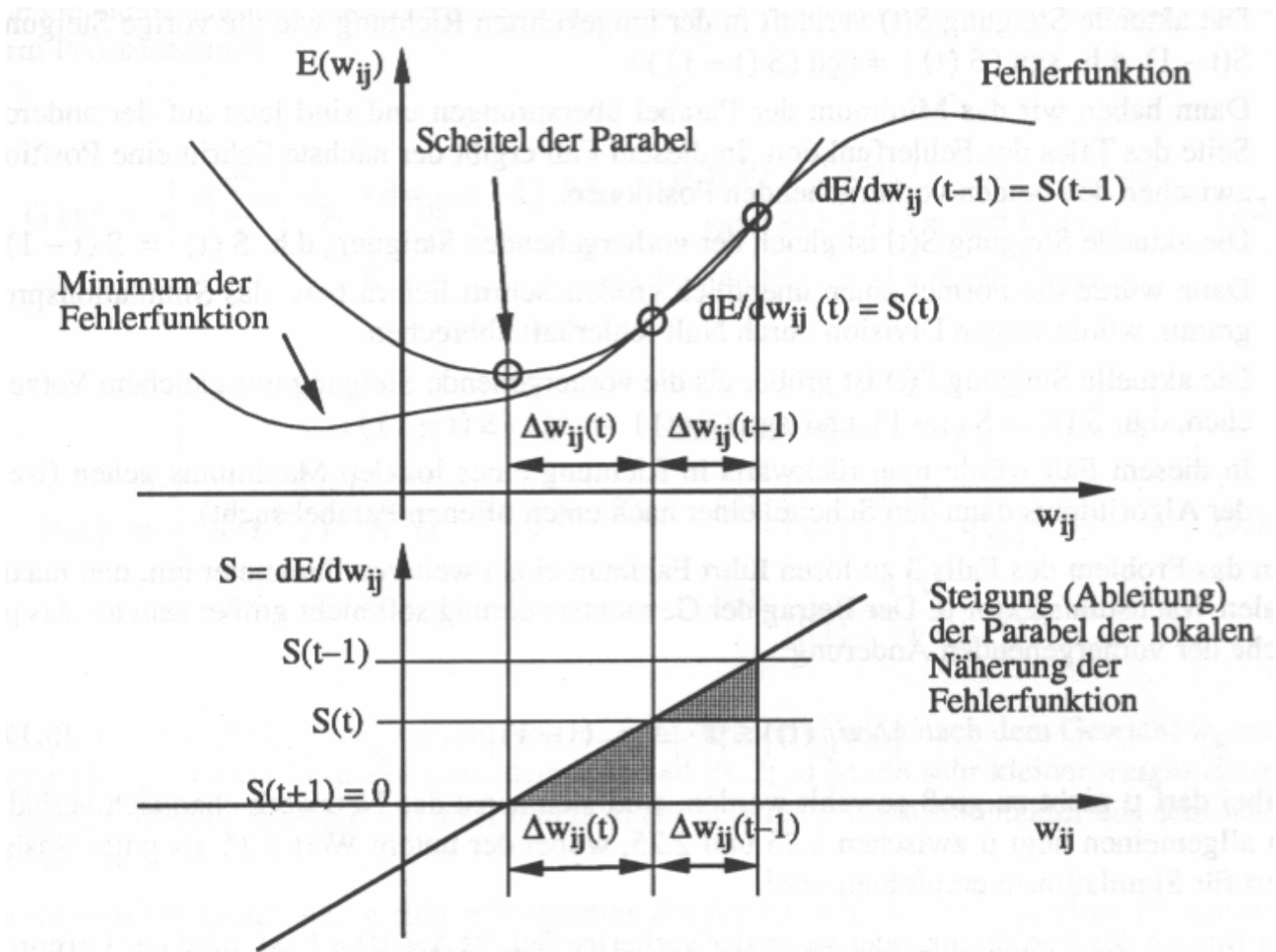
- Wirkung
  - vermeidet zu kleine Schritte auf Plateaus
  - verhindert zu große Schritte in Schluchten des Fehlergebirges

# Quick-Prop

---

- Fehlergebirge ist lokal quadratisch
- Anpassung durch nach oben geöffnete Parabel
- Scheitelpunkt der Parabel als Approximation des Fehlerminimums in einem Lernschritt
- Bestimmung der Anpassungsparabel
  - partielle Ableitung der Fehlerfunktion im vorangegangenen Lernschritt
  - partielle Ableitung der Fehlerfunktion im aktuellen Lernschritt
  - Gewichtsänderung im letzten Lernschritt

# Quick-Prop: grafische Darstellung



# Weitere Netztypen

---

- Kohonen-Netze (SOFM)
- Hopfield-Netze und Simulated Annealing

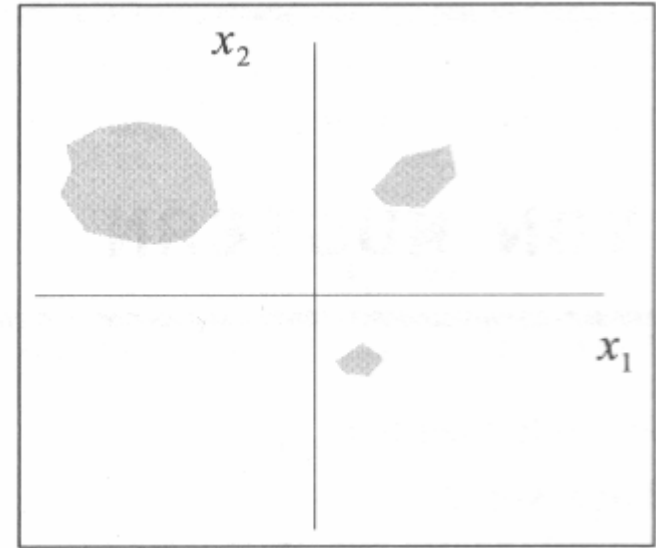
# Kohonen-Netze

---

- SOFM = Self Organizing Feature Map (selbstorganisierende Merkmalskarte)
- unüberwachtes Lernen
- 2 Schichten
  - Eingabeschicht (Merkmale)
  - Kohonenschicht/Ausgabeschicht (Prototypen)
    - räumliche Anordnung
    - Nachbarschaftsbeziehung ...
    - ... aber keine Verbindungen
- Haupteinsatzgebiet: Cluster-Aufgaben
  - Gruppierung von ähnlichen Objekten
  - Erkennen von Bündeln in einer Datenmenge

# Euklidische Distanz als Ähnlichkeitsmaß

---

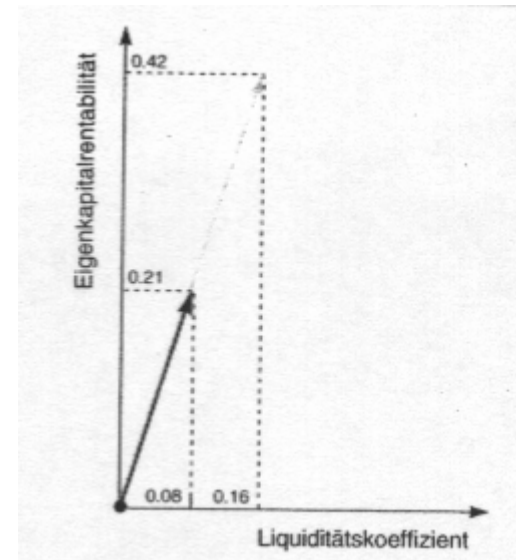
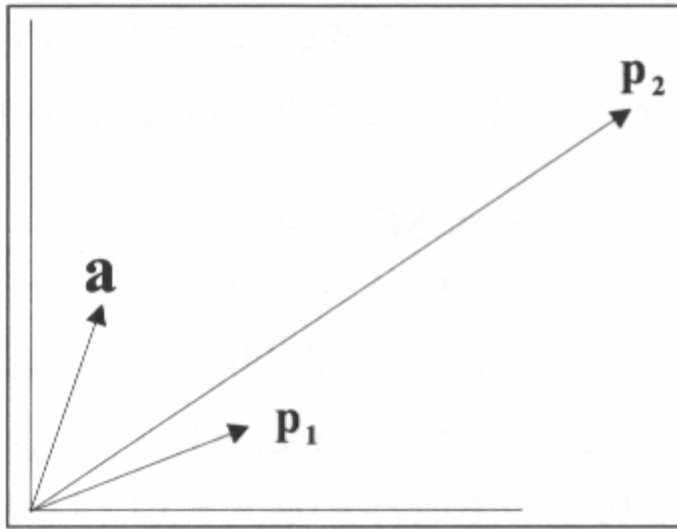


- Euklidische Distanz:

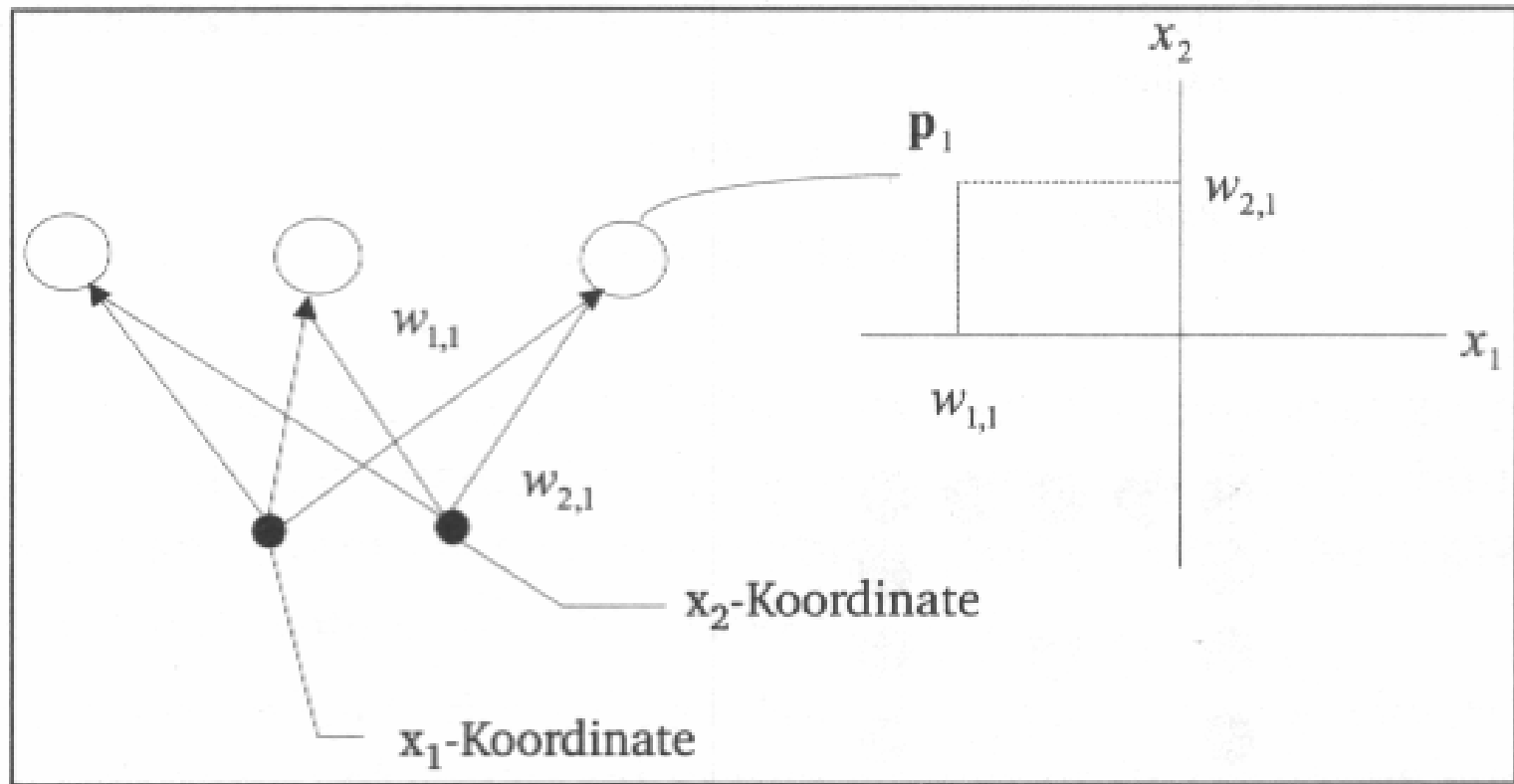
(1)

# Cosinus als Ähnlichkeitsmaß (Normierung)

(1)

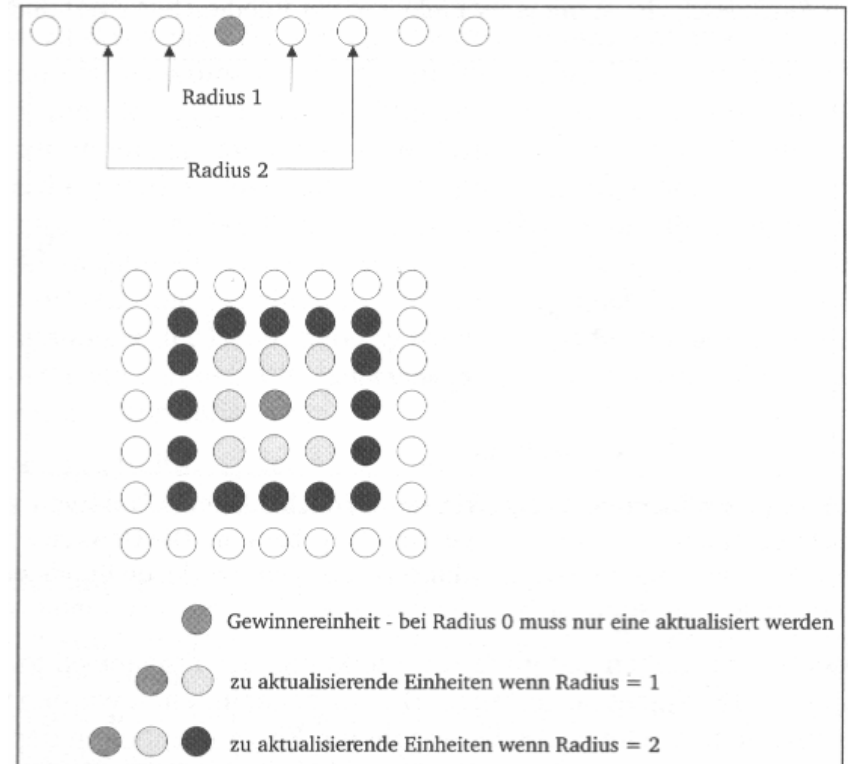


# Netzgewichte als Koordinaten der Prototypen



# Kohonenschicht und Lernverfahren

- ein- oder mehrdimensionale Anordnung
- Nachbarschaft via Radius
- Wettbewerbslernen
  - Wettbewerb um Nähe zum Trainingsmuster
  - grundsätzlich Winner-take-all ...
  - ... je nach Radius kommen auch Nachbarn zum Zuge
  - Lernrate und Radius werden im Lernprozess schrittweise verringert
- Gewichtsanzpassung: (1)



## Beispiel 1 (3.3)

---

- Topologie
  - 3 Eingabeneuronen
  - 2 Kohonnenneuronen
- 4 Trainingsvektoren

[0,8 0,7 0,4], [0,6 0,9 0,9], [0,3 0,4 0,1], [0,1 0,1 0,3]

- Startgewichte

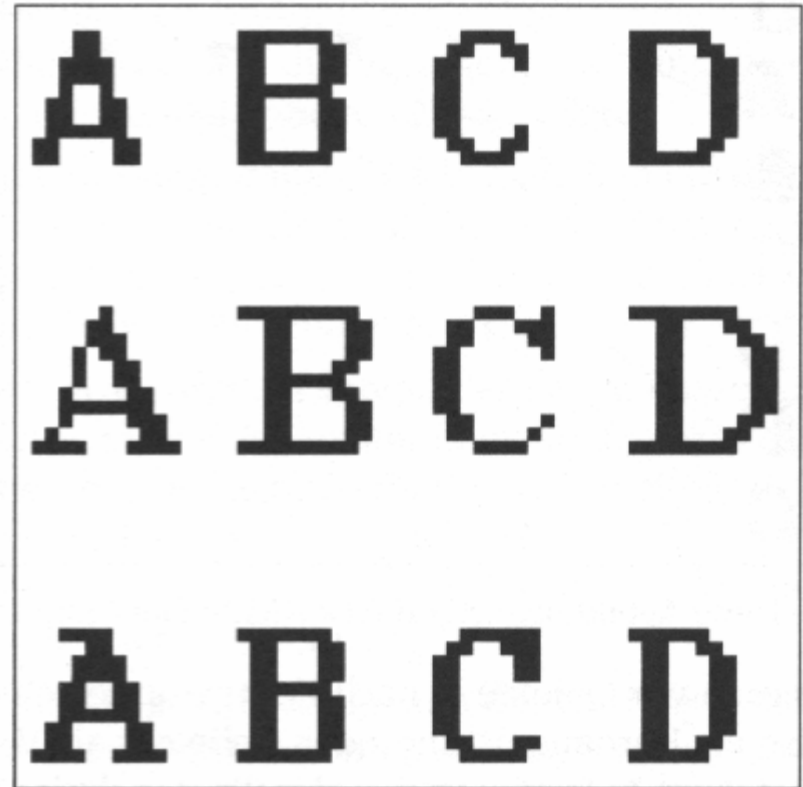
$$\begin{bmatrix} 0,5 & 0,4 \\ 0,6 & 0,2 \\ 0,8 & 0,5 \end{bmatrix}$$

- Radius = 0
- Lernrate = 0,5

## Beispiel 2: Ziffernerkennung

- 256 Eingabeneuronen (16x16-Schwarzweiß-Matrix)
- 9 Kohonenneuronen (3x3-Gitter)

A1		B1
A2		D1
A3		B3
		D3
B2		C1
D2		C2
		C3




# Gaußsches Nachbarschaftsmaß

---

- Lernregel

(1)



- Nachbarschaftsmaß

(2)



# Kritische Größen

---

- Anzahl der Kohonnenneuronen (Granularität der Clusterung)
- Lernverfahren
  - Nachbarschaftsfunktion
  - Lernrate im Zeitverlauf

# Hopfield-Netze

---

- autoassoziativ
  - Speicherfunktion
  - findet verrauschte Eingabemuster wieder
- eine Schicht (Eingabeschicht = Ausgabeschicht)
- Neuronen sind
  - vollständig und ...
  - ... symmetrisch, aber ...
  - ... nicht mit sich selbst verknüpft
- ⇒ Rückkoppelungen (Zustandsänderung eines Neurons beeinflusst potentiell alle übrigen Neuronen)
- ursprünglich: nur binäre Ein- und Ausgabewerte
- Muster in vektorieller Repräsentation

# Beispiel: Hopfield-Netz mit 4 Neuronen

---

# Neuronenfunktionen: Version 1

---

- Schwellwert = 0
- binäre Codierung: 1 und -1
- Propagierungsfunktion (netin)

(1)

A large, empty rectangular box with a thin black border, intended for a diagram or equation representing the propagation function (netin).

- Transferfunktion (out)

(2)

A large, empty rectangular box with a thin black border, intended for a diagram or equation representing the transfer function (out).

# „Lernprozess“

---

- kein typisches neuronales Training
- direkte Bestimmung der Verbindungsgewichte aus den Trainingsbeispielen
- Gewichtsmatrix zum Speichern eines Musters
  - äußeres Produkt des Mustervektors mit sich selbst
  - anschließend alle Diagonalelemente auf Null setzen

(1)



- Gewichtsmatrix zum Speichern von n Mustern

(2)



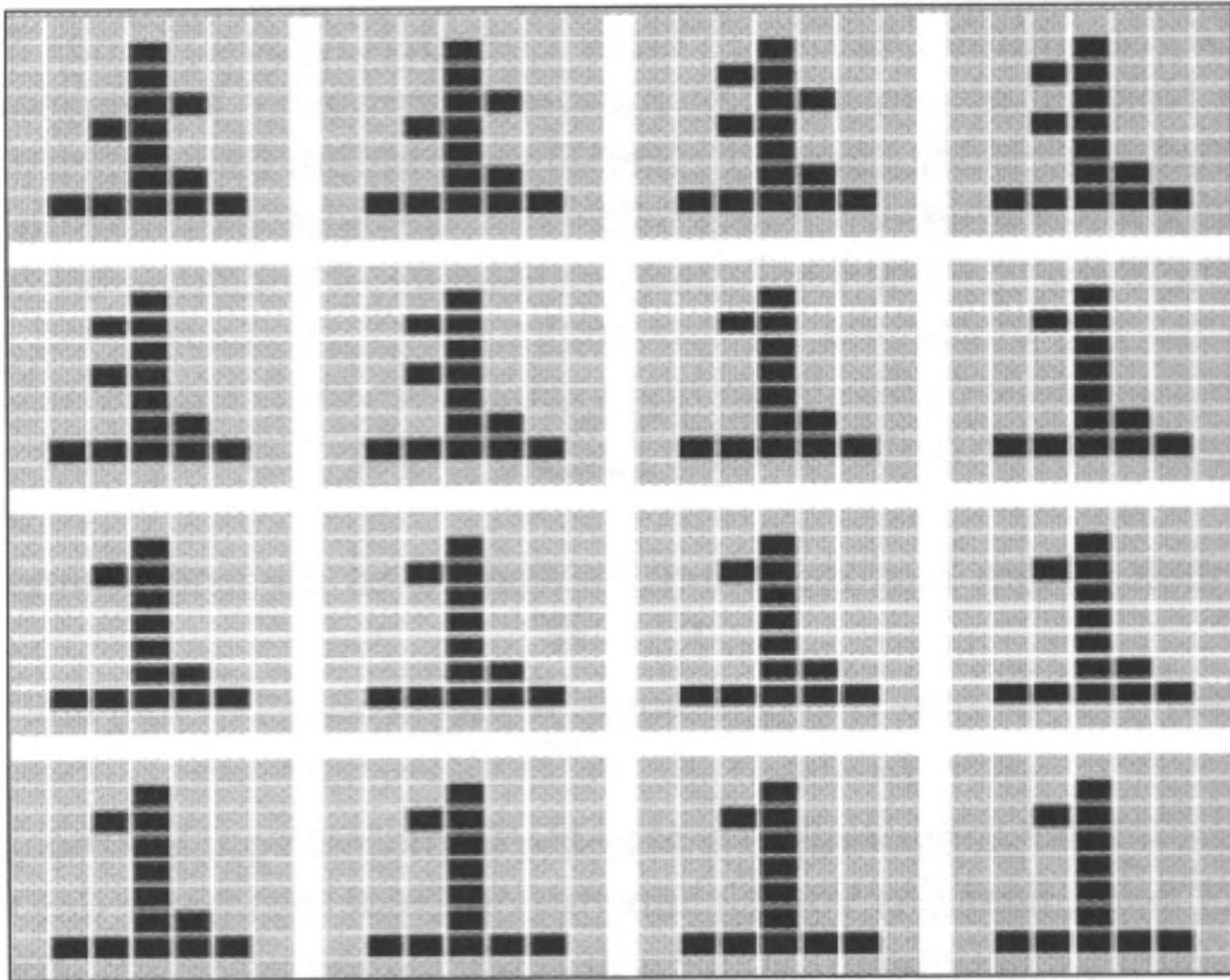
# Beispiel

---

- Trainingsmuster 1:  $[1, -1, 1, 1]$ 
  - Gewichtsmatrix?
  - stabiler Zustand des Netzes?
- Verrauschtes Muster:  $[-1, -1, 1, 1]$ 
  - Konvergenz zu stabilem Zustand?
  - Netzausgabe?
- Trainingsmuster 2:  $[1, 1, 1, -1]$ 
  - neue Gewichtsmatrix?

# Beispielanwendung: Ziffernerkennung

---

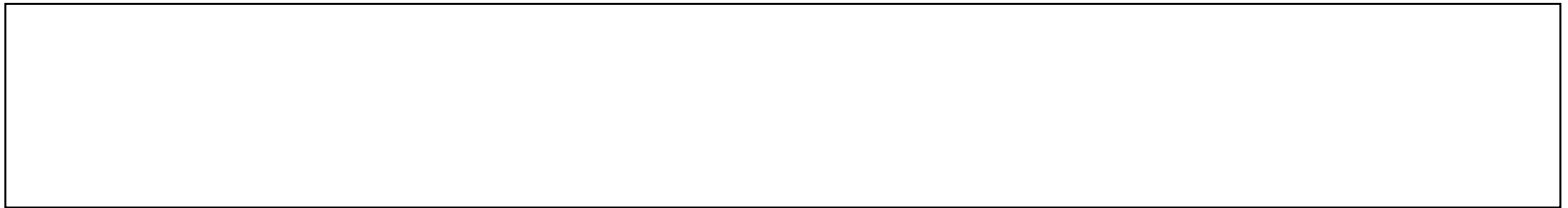


# Speicherkapazität

---

- Richtlinie von Haykin
- Maximale Speicherkapazität

(1)



# Energiefunktion

---

- Energiezustand des Netzes (→ Thermophysik)

(1)

- Einfluss der Zustandsänderung eines Neurons auf den Energiezustand des Netzes

(2)

- Energie nimmt niemals zu
- Konvergenzbeweis  
(garantiertes Erreichen eines stabilen Zustands)
- Terminierungskriterium für Mustererkennung

# Neuronenfunktionen: Version 2

---

- Schwellwert =  $\theta$
- binäre Codierung: 1 und 0
- Propagierungsfunktion (netin)

(1)

- Transferfunktion (out)

(2)

⇒ Vergleich: lokale Schwelle vs. einwirkende Inputs

# (Hamilton-) Energiefunktion

---

- Energiebeitrag eines Neurons

(1)

- Energiezustand des Netzes

(2)

# Parameterbestimmung

---

- bewiesen: jedes gespeicherte Muster korrespondiert mit Energieminimum des Netzes
- alternative Darstellung der Matrizenaddition:

(1)

A large, empty rectangular box with a thin black border, intended for a diagram illustrating the alternative representation of matrix addition.

- (Ähnlichkeit zu Hebb'scher Lernregel)
- Anpassung der Schwellenwerte

(2)

A large, empty rectangular box with a thin black border, intended for a diagram illustrating the adjustment of threshold values.

# Beispiele

---

- Trainingsmuster 1:  $[1, 0, 0]$
- Trainingsmuster 2:  $[0, 0, 1]$
- Schwellenwerte?
  
- Trainingsmuster 1:  $[1, 0, 1]$
- Trainingsmuster 2:  $[0, 1, 0]$
- Zuordnung des verrauschten Musters  $[1, 1, 1]$ ?

# Probleme von Hopfield-Netzen

---

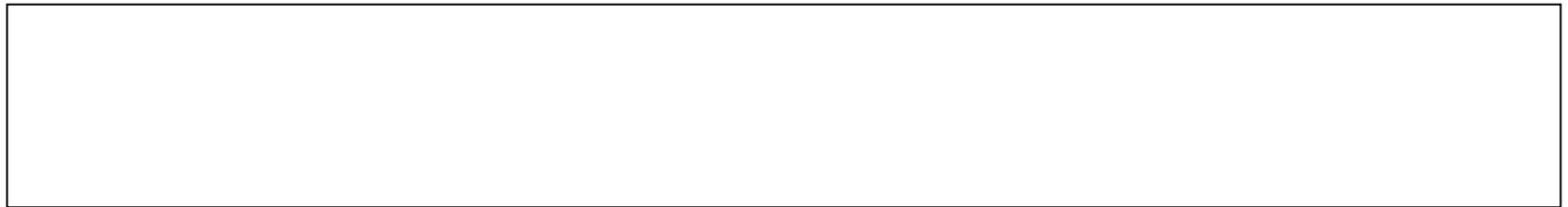
- Gewichtseinstellung nur notwendige, nicht hinreichende Bedingung
- Ausgabe „unbekannter“ Muster (Nebenminima)
  - Energieminima, deren zugehörige Muster nicht in den Trainingsbeispielen enthalten sind
  - keine Adaption der Schwellenwerte
  - Gegenmittel: Unlearning
- All-Zero-Problem
  - Schwellenwerte = 0
  - Nullvektor als stabiler Zustand
- Inversen-Problem
  - Schwellenwerte = 0
  - Inverse der Trainingsmuster als stabiler Zustand (Grund: identische Gewichtsmatrizen)

# Simulated Annealing

---

- Anstieg der Energie während der Berechnungsphase (mit im Zeitablauf abnehmender Wahrscheinlichkeit) möglich
- Ansatzpunkt:
  - Lockerung der binären Aktivierung eines Neurons
  - Wahrscheinlichkeit für Aktivierung in Abhängigkeit des Nettoinputs und der „Temperatur“  $T$
- Wahrscheinlichkeitsfunktion:

(1)



- Abkühlung = Reduktion von  $T$  im Zeitablauf

# Temperatur und Aktivierungswahrscheinlichkeit

