

# Kapitel ML: IV

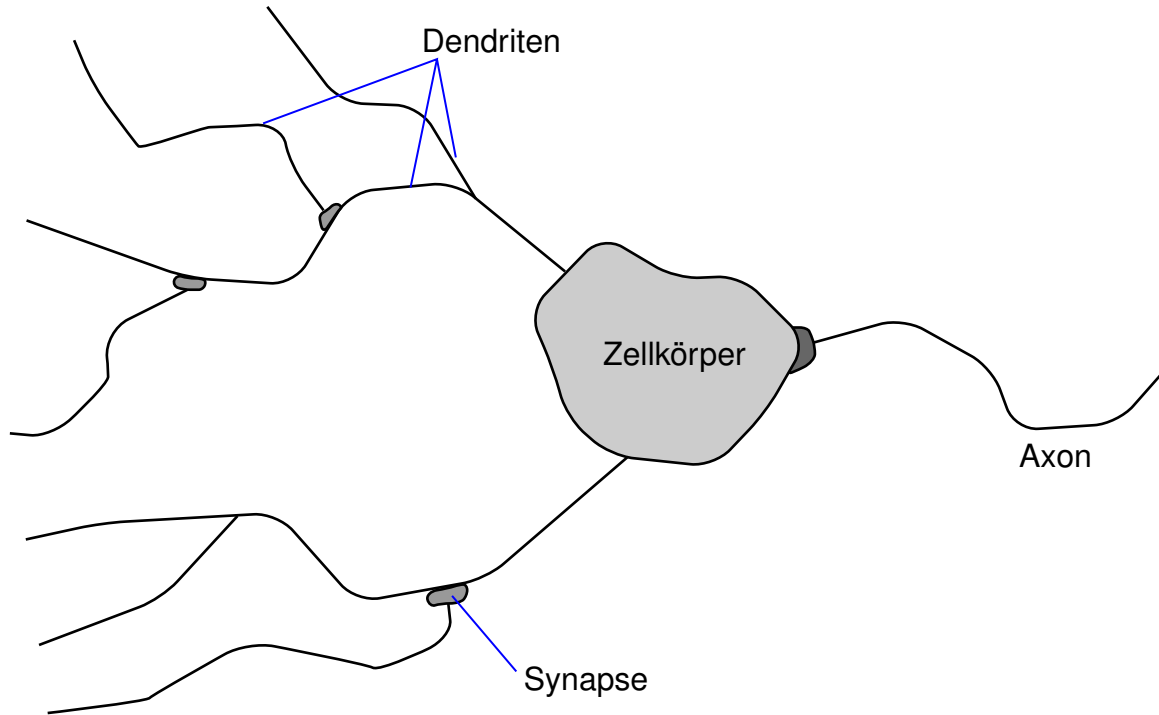
## IV. Neuronale Netze

- Perzeptron-Lernalgorithmus
- Gradientenabstiegsmethode
- Multilayer-Perzeptron
- Self-Organizing Feature Maps
- Neuronales Gas
- Radialbasisfunktionen

# Perzeptron-Lernalgorithmus

## Biologisches Vorbild

Vereinfachte Darstellung einer Nervenzelle (Neuron):



# Perzeptron-Lernalgorithmus

## Biologisches Vorbild (Fortsetzung)

### Merkmale:

- ❑ Eine Menge von Dendriten dienen der Nervenzelle als Eingangskanäle für elektrische Impulse.
- ❑ An Kontaktstellen der Dendriten (Synapsen) können elektrische Impulse ausgelöst werden.
- ❑ Synapsen können unterschiedlich starke Impulse auslösen.
- ❑ Der Zellkörper summiert die eingehenden Impulse.
- ❑ Wird eine bestimmte Reizschwelle überschritten, so entsteht im Zellkörper ein Signal, das durch das Axon weitergeleitet wird.
- ❑ Die Informationsverarbeitung ist gerichtet.

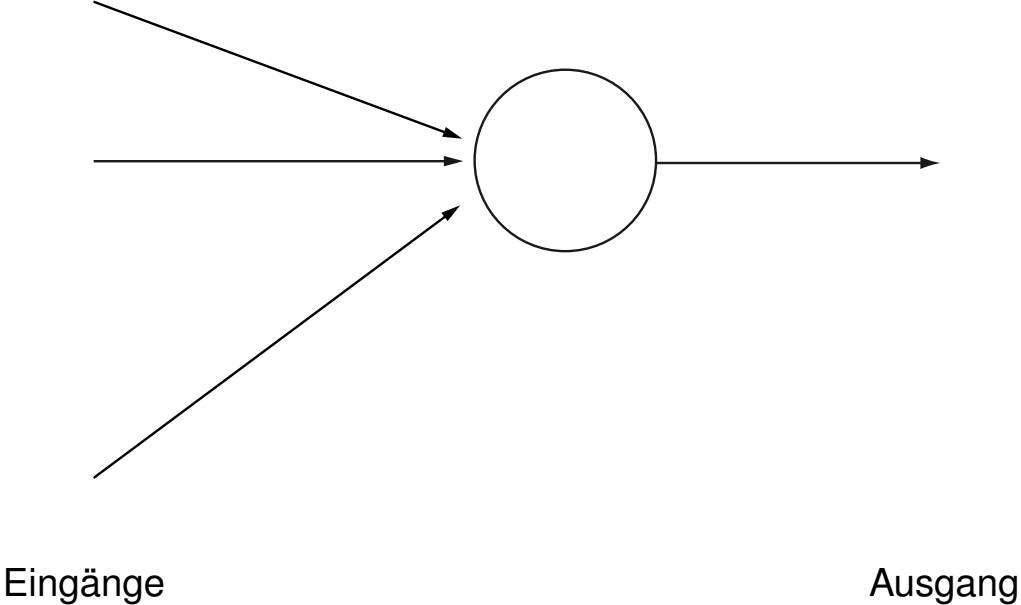
# Perzeptron-Lernalgorithmus

## Historie

- 1943 McCulloch und Pitts schlagen ein Modell für das Neuron vor
- 1949 Hebb postuliert Lern-Paradigma: Bestärkung nur für aktive Neuronen
- 1958 Rosenblatt entwickelt das Perzeptron-Modell
- 1962 Rosenblatt beweist das Perzeptron-Konvergenz-Theorem
- 1969 Minsky & Papert veröffentlichen Buch über die Grenzen des Perzeptrons
- 1970
- ⋮
- 1985
- 1986 Rumelhart & McClelland stellen das Multilayer-Perzeptron vor

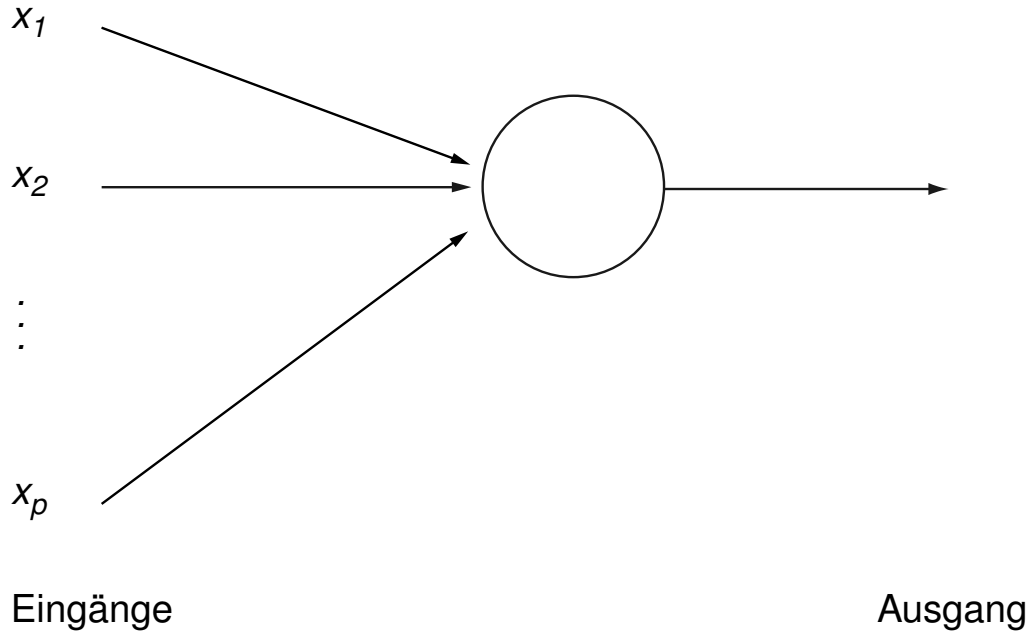
# Perzeptron-Lernalgorithmus

Rosenblatts Perzeptron [1958]



# Perzeptron-Lernalgorithmus

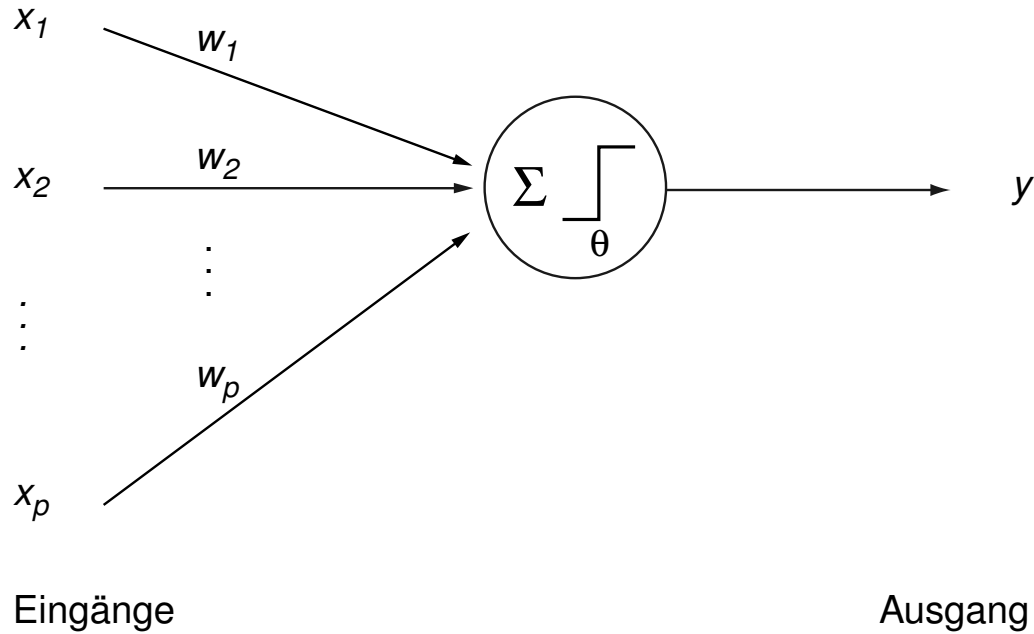
Rosenblatts Perzeptron [1958]



$$x_i, w_i \in \mathbf{R}, \quad i = 1 \dots p$$

# Perzeptron-Lernalgorithmus

Rosenblatts Perzeptron [1958]



$$x_i, w_i \in \mathbf{R}, \quad i = 1 \dots p$$

## Bemerkungen:

- ❑ Das Perzeptron von Rosenblatt basiert auf dem Neuronenmodell von McCulloch und Pitts.
- ❑ Es handelt sich um ein Feedforward-System.



# Perzeptron-Lernalgorithmus

Spezifikation des Klassifikationsproblems [vgl. ML:I-17]

- $X \subseteq \mathbf{R}^p$  sei der Instanzenraum (Merkmalsraum).
- $C = \{0, 1\}$  seien zwei Klassen.
- $c : X \rightarrow C$  sei der zu lernende Klassifikator für  $X$ .
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \dots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$  sei eine Menge von Lernbeispielen.

Wie könnte der Hypothesenraum aussehen?

# Perzeptron-Lernalgorithmus

## Berechnung im Perzeptron

Falls  $\sum_{i=1}^p w_i x_i \geq \theta$  dann  $y = 1$ , bzw.

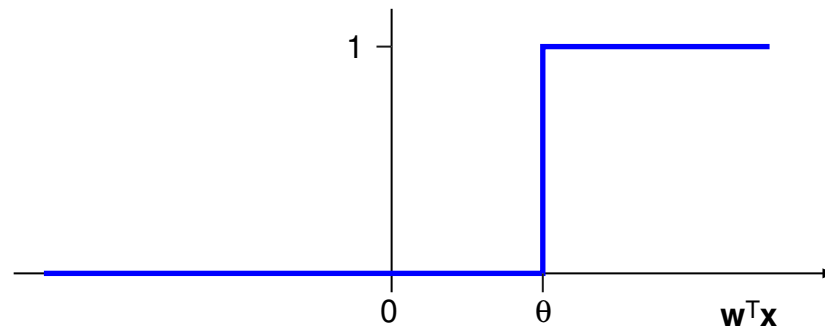
$\sum_{i=1}^p w_i x_i < \theta$  dann  $y = 0$

# Perzeptron-Lernalgorithmus

## Berechnung im Perzeptron

Falls  $\sum_{i=1}^p w_i x_i \geq \theta$  dann  $y = 1$ , bzw.

$\sum_{i=1}^p w_i x_i < \theta$  dann  $y = 0$



mit  $\sum_{i=1}^p w_i x_i = \mathbf{w}^T \mathbf{x}$  oder anderen Schreibweisen für das Skalarprodukt.

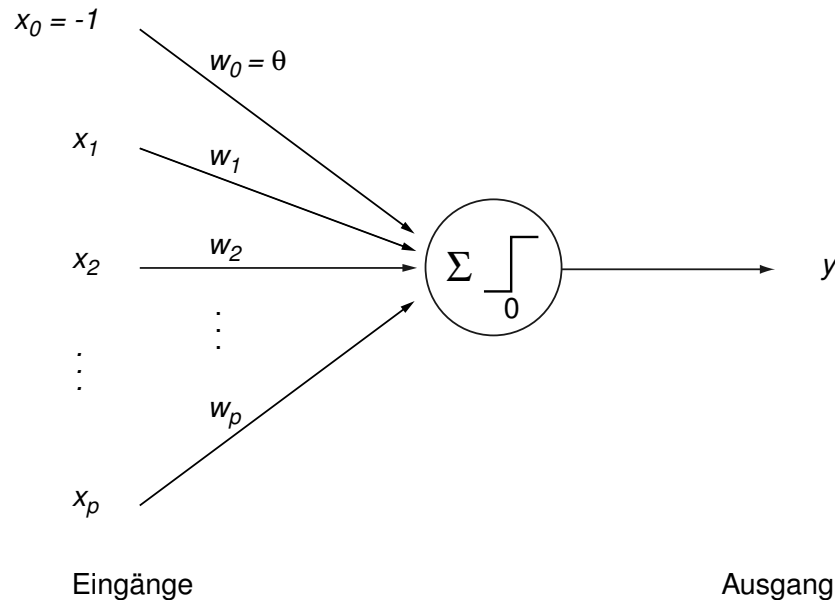
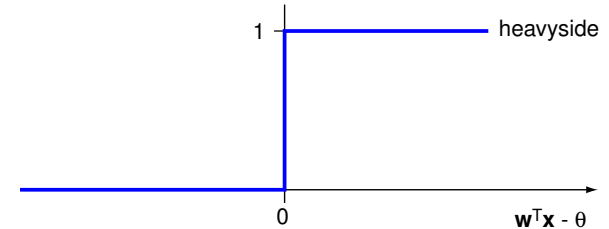
→ Hypothesen festgelegt durch  $w_1, \dots, w_p$  und  $\theta$ .

# Perzeptron-Lernalgorithmus

## Berechnung im Perzeptron

$$y(\mathbf{x}) = \text{heavyside}\left(\sum_{i=1}^p w_i x_i - \theta\right)$$

$$= \text{heavyside}\left(\sum_{i=0}^p w_i x_i\right) \quad \text{mit } w_0 = \theta, \quad x_0 = -1$$



→ Hypothesen festgelegt durch  $w_0, w_1, \dots, w_p$ .

## Bemerkungen:

- Wenn man die Lernbeispiele so anpasst, dass jeder Merkmalsvektor um ein weiteres Merkmal  $x_0$  mit  $x_0 = -1$  erweitert wird, erhält der Lernalgorithmus eine kanonische Form. Implementationen von neuronalen Netzen nehmen diese Erweiterung der Lernbeispiele meist implizit vor.
- Der Output des Perzeptrons hat den Wert 1, wenn die gewichtete Summe der Merkmalswerte einen Schwellwert überschreitet.

# Perzeptron-Lernalgorithmus

## Algorithmus zur Gewichts Anpassung

Algorithm: *PT* Perceptron Training

Input:  $D$  Lernbeispiele der Form  $(\mathbf{x}, c(\mathbf{x}))$  mit  $|\mathbf{x}| = 1 + p$ ,  $c(\mathbf{x}) \in \{0, 1\}$ .  
 $\eta$  positive kleine Konstante (Lernrate).

Output:  $\mathbf{w}$  Gewichtsvektor.

$PT(D, \eta)$

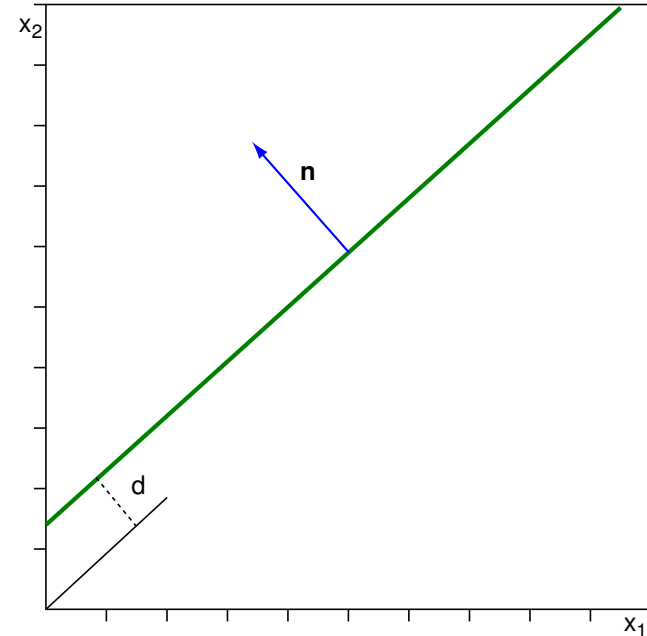
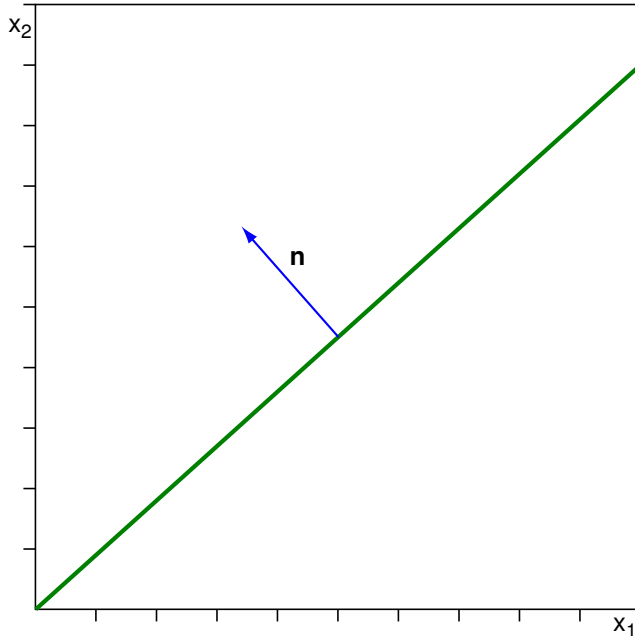
1. *initialize\_random\_weights*( $\mathbf{w}$ ),  $t = 0$
2. **REPEAT**
3.  $t = t + 1$
4.  $(\mathbf{x}, c(\mathbf{x})) = \text{random\_select}(D)$ ;
5.  $\text{error} = c(\mathbf{x}) - \text{heavyside}(\mathbf{w}^T \mathbf{x})$
6. **FOR**  $i = 0$  **TO**  $p$  **DO**
7.  $\Delta w_i = \eta \cdot \text{error} \cdot x_i$
8.  $w_i = w_i + \Delta w_i$
9. **ENDDO**
10. **UNTIL**(*convergence*( $D, Y$ )  $\vee t > t_{\max}$ )

## Bemerkungen:

- $t$  stellt eine Variable für die Zeit dar. Zu aufeinanderfolgenden Zeitpunkten wird dem Lernalgorithmus jeweils ein Trainingsbeispiel präsentiert, für dessen Klassifikation er den Gewichtsvektor im Perzeptron gegebenenfalls anpasst.
- Die Vorschrift zur Gewichts Anpassung verwendet die Abweichung der vorgegebenen Klasse  $c(\mathbf{x})$  von der durch das Perzeptron berechneten Klasse für das fehlklassifizierte  $\mathbf{x}$ . Diese Abweichung ist entweder  $-1$  oder  $+1$ , unabhängig davon, wie dicht  $\mathbf{x}$  an der aktuell berechneten Hyperebene  $\mathbf{w}$  liegt.

# Perzeptron-Lernalgorithmus

## Algorithmus zur Gewichts Anpassung



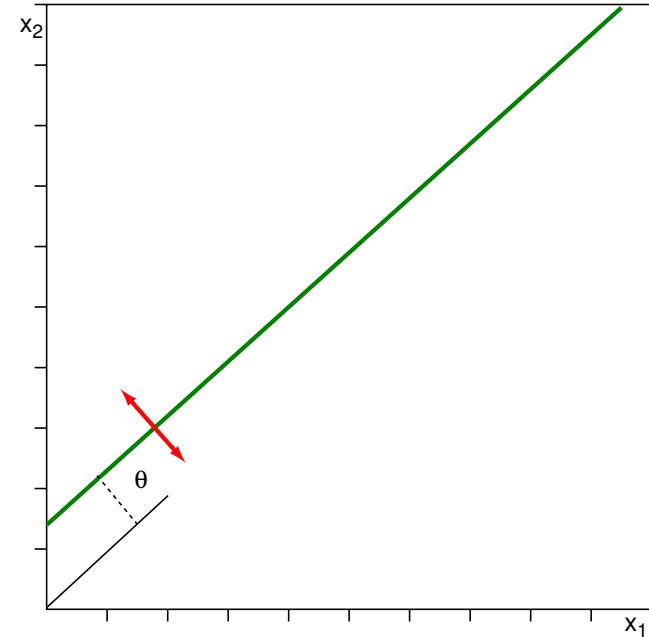
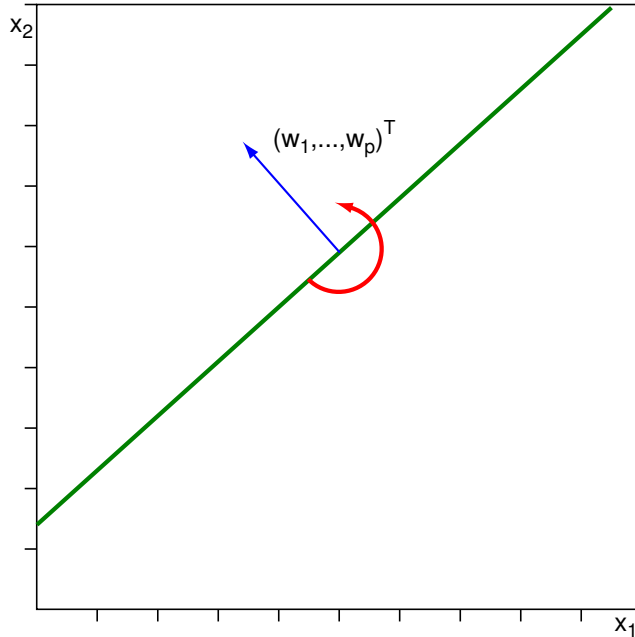
Definition einer (affinen) Hyperebene:  $\mathbf{n}^T \mathbf{x} = d$ .

- $\mathbf{n}$  bezeichnet einen Normalenvektor, senkrecht zur Hyperebene.
- Für  $\|\mathbf{n}\| = 1$  und  $d \geq 0$  bezeichnet  $d$  den Abstand des Ursprungs von der Ebene.
- $\mathbf{n}^T \mathbf{x} < d$  heißt,  $\mathbf{x}$  liegt auf derselben Seite der Ebene wie der Ursprung (für  $d > 0$ ).



# Perzeptron-Lernalgorithmus

## Algorithmus zur Gewichts Anpassung



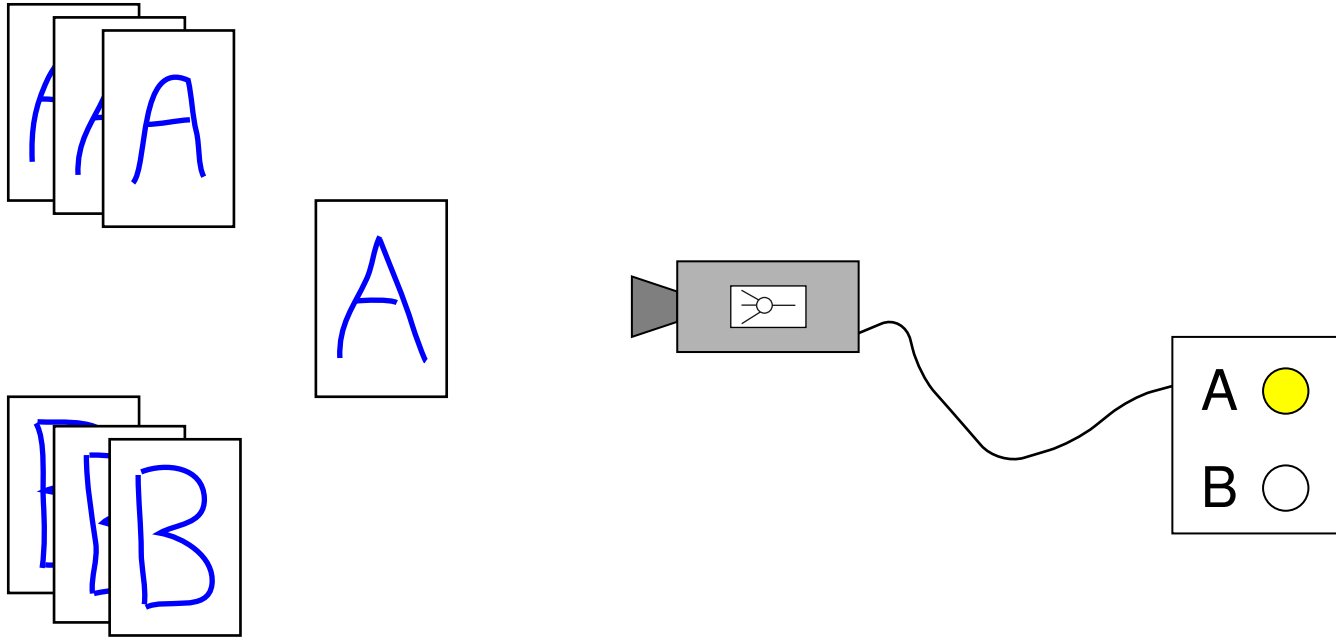
Definition einer (affinen) Hyperebene:  $\mathbf{w}^T \mathbf{x} = \theta \Leftrightarrow \sum_{i=1}^p w_i x_i = \theta$ .

## Bemerkungen:

- Die Menge aller Gewichtsvektoren  $\mathbf{w} = (w_0, w_1, \dots, w_p)^T$  bilden den Hypothesenraum  $H$ .
- Ein Perzeptron definiert eine Gerade (Hyperebene) senkrecht (normal) zu  $(w_1, \dots, w_p)^T$ .
- $\theta$  beschreibt die Verschiebung der Geraden (Hyperebene) entlang  $(w_1, \dots, w_p)^T$ .
- $\theta$  heißt Bias (= Vorurteil, systematische Verzerrung, Tendenz) des Perzeptrons.
- Das Lernverfahren ist überwacht.
- Durch den Faktor  $x_i$  in der Lernregel werden nur am Fehler beteiligte Gewichte verändert.  
Stichwort: Hebb'sches Lernen [1949]

# Perzeptron-Lernalgorithmus

## Illustration



- Die Beispiele werden dem Perzeptron präsentiert.
- Das Perzeptron berechnet einen Wert, der als Klassenzugehörigkeit interpretiert wird.

# Perzeptron-Lernalgorithmus

## Illustration (Fortsetzung)

Codierung:

- Beispiele werden anhand von Merkmalen charakterisiert: Anzahl der Kreuzungspunkte, spitzester Winkel, längste Kante, etc.
- Die Klassenzugehörigkeit,  $c(\mathbf{x})$ , wird als Zahl codiert. Beispiele der Menge  $A$  ( $B$ ) erhalten die 1 (0).

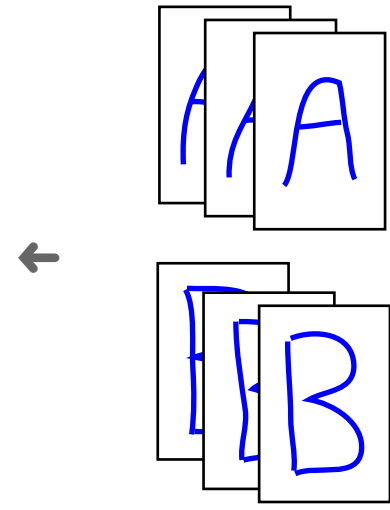
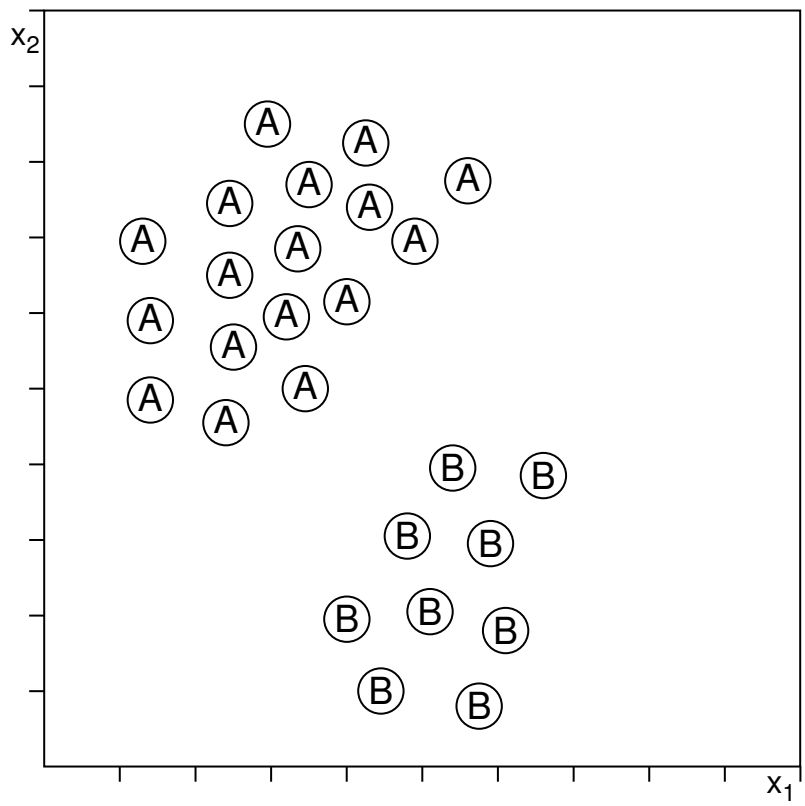
$$\underbrace{\begin{pmatrix} x_{1_1} \\ x_{1_2} \\ \vdots \\ x_{1_p} \end{pmatrix} \quad \dots \quad \begin{pmatrix} x_{k_1} \\ x_{k_2} \\ \vdots \\ x_{k_p} \end{pmatrix}}_{\text{Klasse } A \simeq c(\mathbf{x}) = 1}$$

$$\underbrace{\begin{pmatrix} x_{l_1} \\ x_{l_2} \\ \vdots \\ x_{l_p} \end{pmatrix} \quad \dots \quad \begin{pmatrix} x_{m_1} \\ x_{m_2} \\ \vdots \\ x_{m_p} \end{pmatrix}}_{\text{Klasse } B \simeq c(\mathbf{x}) = 0}$$

# Perzeptron-Lernalgorithmus

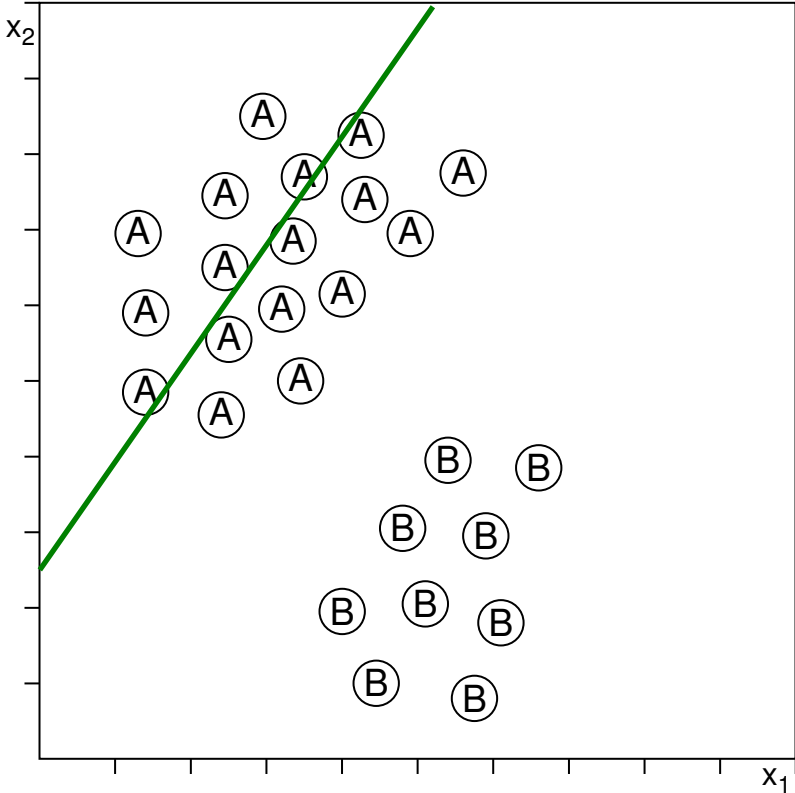
Illustration (Fortsetzung)

Darstellung der Objekte im Merkmalsraum:



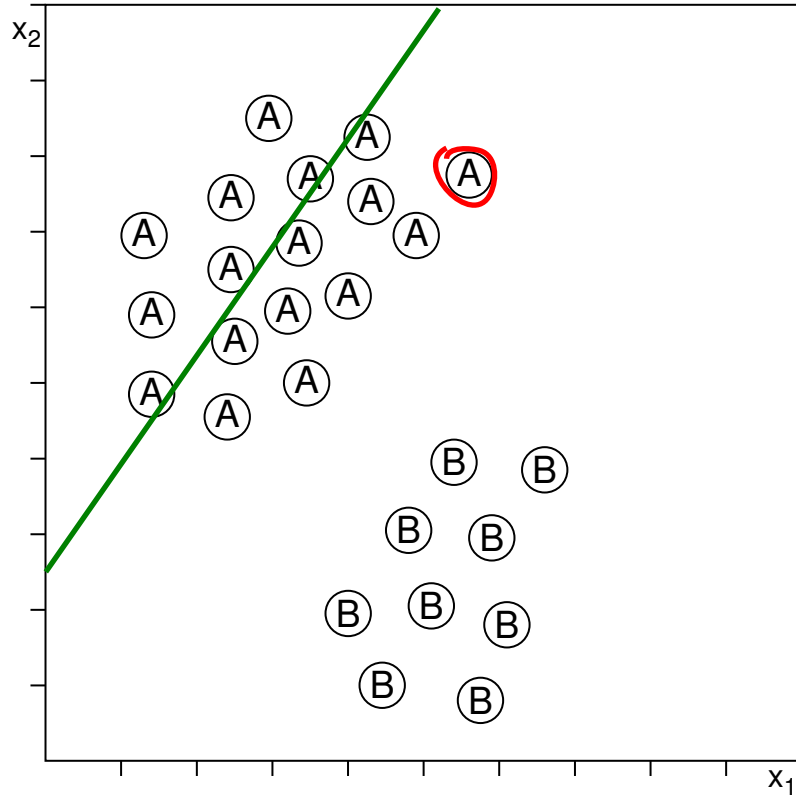
# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



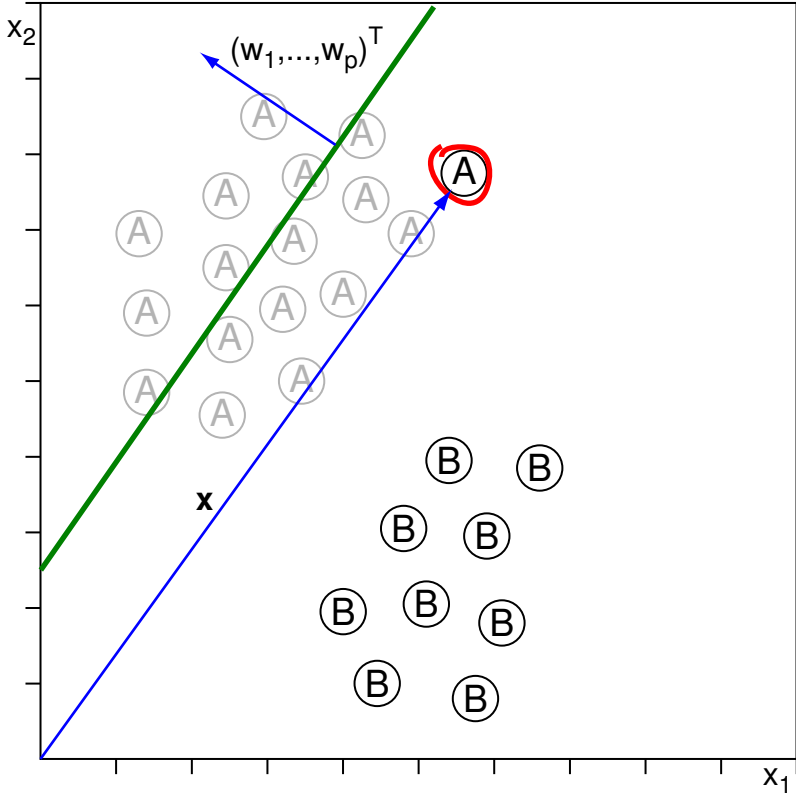
# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



# Perzeptron-Lernalgorithmus

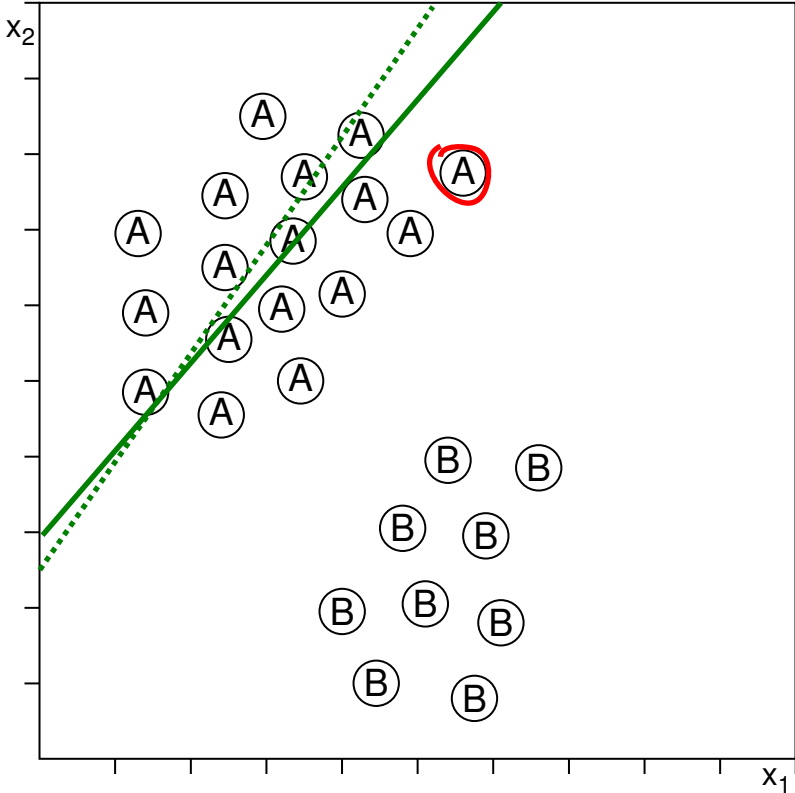
Illustration (Fortsetzung)





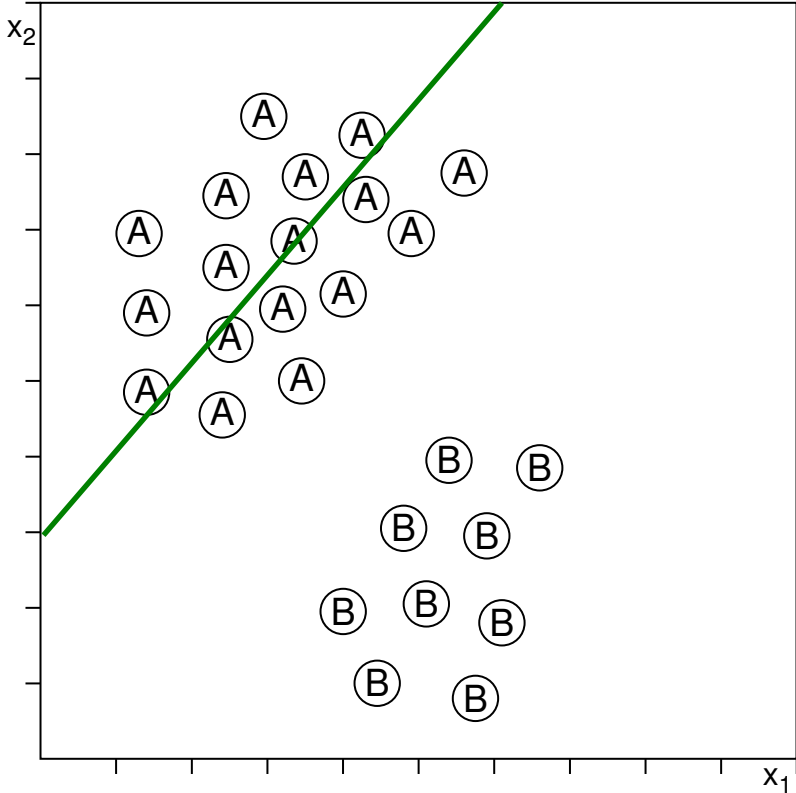
# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



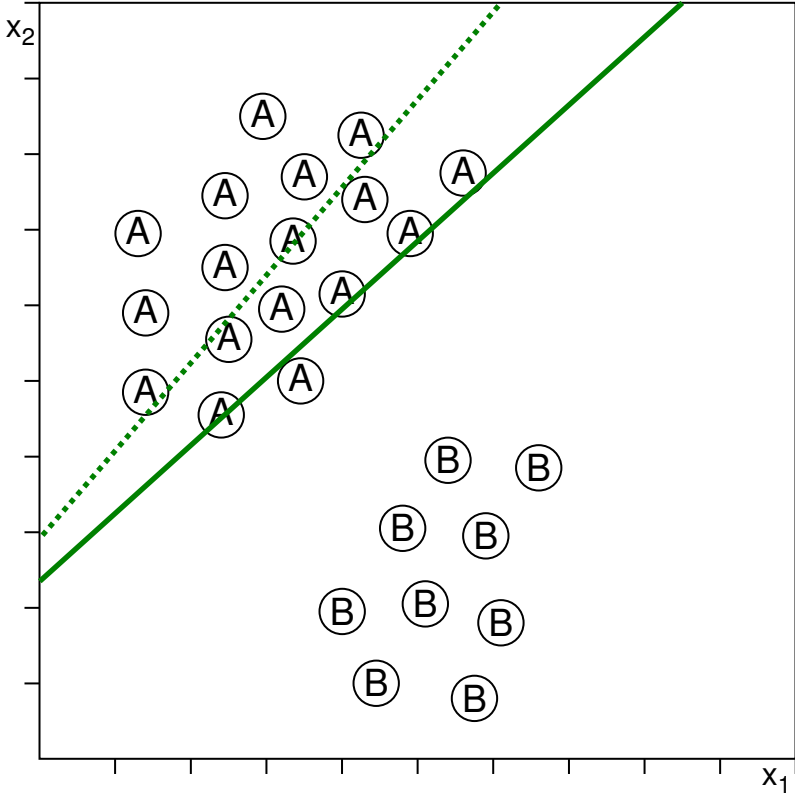
# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



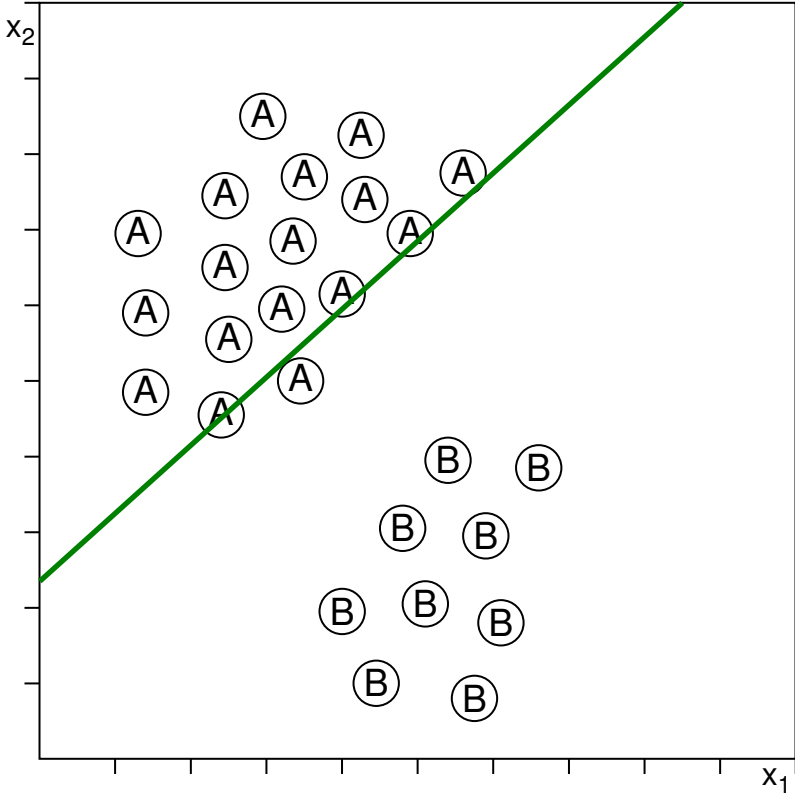
# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



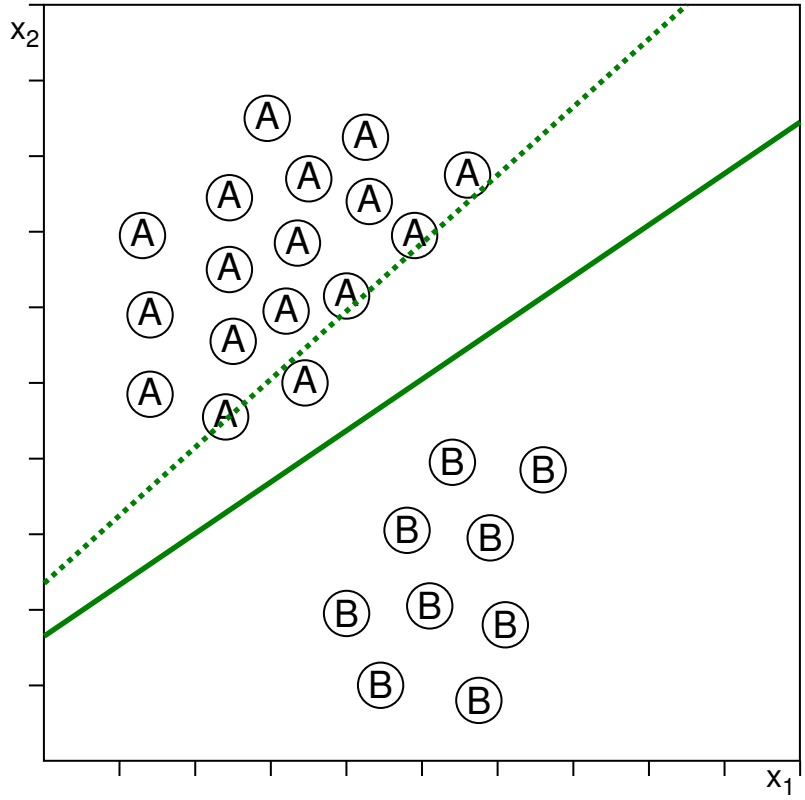
# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



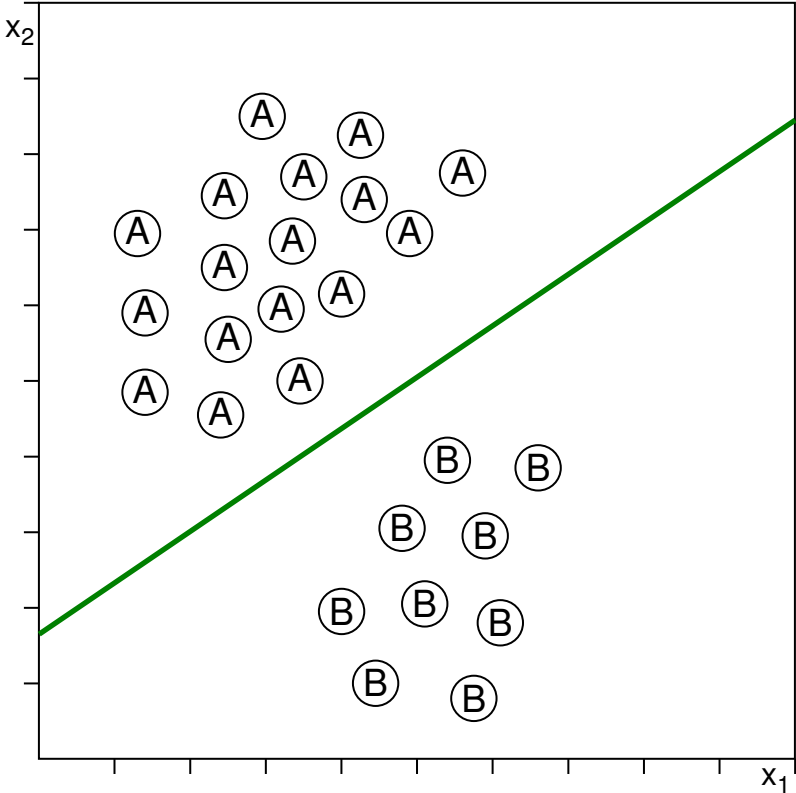
# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



# Perzeptron-Lernalgorithmus

Illustration (Fortsetzung)



# Perzeptron-Lernalgorithmus

## Perzeptron-Konvergenztheorem

Fragen:

- Welche Lernaufgaben können mit den Funktionen des Hypothesenraumes gelöst werden?
- Kann der Perzeptron-Lernalgorithmus eine solche Funktion bestimmen?

### **Satz 2 (Perzeptron-Konvergenztheorem [Rosenblatt 1962])**

Seien  $D_0$  und  $D_1$  zwei nicht-leere, endliche Mengen von Vektoren  $\mathbf{x} = (-1, x_1, \dots, x_p)^T$  mit  $D_1 \cap D_0 = \emptyset$ , und sei durch  $\hat{\mathbf{w}}$  eine Hyperebene definiert, welche die Mengen  $D_1$  und  $D_0$  trennt. Sei weiter  $D$  die Menge der Beispiele der Form  $(\mathbf{x}, 0)$  mit  $\mathbf{x} \in D_0$  oder  $(\mathbf{x}, 1)$  mit  $\mathbf{x} \in D_1$ . Dann gilt:

Werden die Vektoren aus  $D$  dem Perzeptron-Lernalgorithmus präsentiert, so konvergiert der Gewichtsvektor  $\mathbf{w}$  des Perzeptrons innerhalb endlich vieler Iterationen.

# Perzeptron-Lernalgorithmus

## Perzeptron-Konvergenztheorem: Beweis

### Vorüberlegungen:

- Die Mengen  $D_1$  und  $D_0$  werden durch eine Hyperebene  $\hat{\mathbf{w}}$  getrennt. Im weiteren Beweis benötigt man die Eigenschaft, dass für alle  $\mathbf{x} \in D_1$  gilt  $\hat{\mathbf{w}}^T \mathbf{x} > 0$ . Sei ein  $\mathbf{x}' \in D_1$  und gelte  $\hat{\mathbf{w}}^T \mathbf{x}' = 0$ .

Da  $D_0$  endlich ist, haben die Elemente  $\mathbf{x} \in D_0$  einen positiven Mindestabstand  $\delta$  von der Hyperebene  $\hat{\mathbf{w}}$ . Die Hyperebene kann also um  $\frac{\delta}{2}$  in Richtung  $D_0$  parallelverschoben werden. Für diese neue Hyperebene  $\hat{\mathbf{w}}'$  gilt weiterhin für alle  $\mathbf{x} \in D_0$  die Eigenschaft  $(\hat{\mathbf{w}}')^T \mathbf{x} < 0$ , aber nun gilt für alle  $\mathbf{x} \in D_1$  auch  $(\hat{\mathbf{w}}')^T \mathbf{x} > 0$ . Ohne Einschränkung der Allgemeinheit hat  $\hat{\mathbf{w}}$  diese gewünschte Eigenschaft.

- Der gesuchte Gewichtsvektor  $\mathbf{w}$  für das Perzeptron soll ebenfalls für die Vektoren  $\mathbf{x} \in D_0$  das Ergebnis  $\mathbf{w}^T \mathbf{x} < 0$  liefern und für  $\mathbf{x} \in D_1$  das Ergebnis  $\mathbf{w}^T \mathbf{x} > 0$ .
- Betrachte  $D' = D_1 \cup \{-\mathbf{x} \mid \mathbf{x} \in D_0\}$ ; gesucht wird also ein Gewichtsvektor  $\mathbf{w}$ , der für alle  $\mathbf{x} \in D'$  liefert  $\mathbf{w}^T \mathbf{x} > 0$ .
- Der Perzeptron-Lernalgorithmus arbeitet rundenorientiert.  $\mathbf{w}(t)$  bezeichne den Gewichtsvektor für Runde  $t$ , aus dem der Gewichtsvektor für Runde  $t + 1$  berechnet wird.  $\mathbf{x}(t)$  bezeichne den in Runde  $t$  ausgewählten Vektor aus  $D'$  und die zugehörige Klasse mit  $c(\mathbf{x}(t))$ . Der zufällig gewählte erste Gewichtsvektor sei  $\mathbf{w}(0)$ .



# Perzeptron-Lernalgorithmus

## Perzeptron-Konvergenztheorem: Beweis (Fortsetzung)

1. Das Perzeptron berechnet in Runde  $t$  das Skalarprodukt  $\mathbf{w}(t)^T \mathbf{x}(t)$ :  
Bei korrekter Klassifikation, d. h.  $\mathbf{w}(t)^T \mathbf{x}(t) > 0$ , bleibt  $\mathbf{w}$  unverändert.  
Bei Fehlklassifikation erfolgt eine Gewichtsanzpassung:  $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$
2. Betrachtung einer Folge  $(\mathbf{x}(t)), t \in \mathbb{N}$ , fehllklassifizierter Vektoren.  
Bei jeder Gewichtsanzpassung geschieht:  $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$  d. h.:  
$$\mathbf{w}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0)$$
$$\mathbf{w}(2) = \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1)$$
$$\dots$$
$$\mathbf{w}(n) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \dots + \eta \cdot \mathbf{x}(n-1)$$
3. Die durch  $\hat{\mathbf{w}}$  definierte Hyperebene trennt  $D_1$  und  $D_0$ :  $\forall \mathbf{x} \in D' : \hat{\mathbf{w}}^T \mathbf{x} > 0$   
Sei  $\delta := \min_{\mathbf{x} \in D'} \hat{\mathbf{w}}^T \mathbf{x}$ , also gilt  $\delta > 0$ .
4. Das Skalarprodukt mit  $\hat{\mathbf{w}}$  ergibt:  
$$\hat{\mathbf{w}}^T \mathbf{w}(n) = \hat{\mathbf{w}}^T \mathbf{w}(0) + \eta \cdot \hat{\mathbf{w}}^T \mathbf{x}(0) + \dots + \eta \cdot \hat{\mathbf{w}}^T \mathbf{x}(n-1)$$
$$\Rightarrow \hat{\mathbf{w}}^T \mathbf{w}(n) \geq \hat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta \geq 0 \quad \text{für } n \geq n_0 \text{ mit genügend großem } n_0 \in \mathbb{N}$$
$$\Rightarrow (\hat{\mathbf{w}}^T \mathbf{w}(n))^2 \geq (\hat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta)^2 \quad \text{für } n \geq n_0$$
5. Cauchy-Schwarz-Ungleichung:  $\|\mathbf{a}\|^2 \|\mathbf{b}\|^2 \geq (\mathbf{a}^T \mathbf{b})^2$  mit  $\|\mathbf{x}\| := \sqrt{\mathbf{x}^T \mathbf{x}}$  Euklidische Norm.  
$$\|\hat{\mathbf{w}}\|^2 \|\mathbf{w}(n)\|^2 \geq (\hat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta)^2 \quad \Rightarrow \quad \|\mathbf{w}(n)\|^2 \geq \frac{(\hat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta)^2}{\|\hat{\mathbf{w}}\|^2}$$

# Perzeptron-Lernalgorithmus

## Perzeptron-Konvergenztheorem: Beweis (Fortsetzung)

6. Noch einmal zur Gewichts Anpassung:  $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$

$$\begin{aligned}\Rightarrow \|\mathbf{w}(t+1)\|^2 &= \|\mathbf{w}(t) + \eta \cdot \mathbf{x}(t)\|^2 \\ &= (\mathbf{w}(t) + \eta \cdot \mathbf{x}(t))^T (\mathbf{w}(t) + \eta \cdot \mathbf{x}(t)) \\ &= \mathbf{w}(t)^T \mathbf{w}(t) + \eta^2 \cdot \mathbf{x}(t)^T \mathbf{x}(t) + 2\eta \cdot \mathbf{w}(t)^T \mathbf{x}(t) \\ &\leq \|\mathbf{w}(t)\|^2 + \|\eta \cdot \mathbf{x}(t)\|^2 \quad \text{wegen } \mathbf{w}(t)^T \mathbf{x}(t) < 0\end{aligned}$$

7. Somit gilt für eine Folge von  $n$  Gewichts Anpassungen:

$$\begin{aligned}\|\mathbf{w}(n)\|^2 &\leq \|\mathbf{w}(n-1)\|^2 + \|\eta \cdot \mathbf{x}(n-1)\|^2 \\ &\leq \|\mathbf{w}(n-2)\|^2 + \|\eta \cdot \mathbf{x}(n-2)\|^2 + \|\eta \cdot \mathbf{x}(n-1)\|^2 \\ &\leq \|\mathbf{w}(0)\|^2 + \|\eta \cdot \mathbf{x}(0)\|^2 + \dots + \|\eta \cdot \mathbf{x}(n-1)\|^2 \\ &= \|\mathbf{w}(0)\|^2 + \sum_{i=0}^{n-1} \|\eta \cdot \mathbf{x}(i)\|^2\end{aligned}$$

8. Mit  $\varepsilon := \max_{\mathbf{x} \in D'} \|\mathbf{x}\|^2$  folgt  $\|\mathbf{w}(n)\|^2 \leq \|\mathbf{w}(0)\|^2 + n\eta^2\varepsilon$

# Perzeptron-Lernalgorithmus

## Perzeptron-Konvergenztheorem: Beweis (Fortsetzung)

9. Beide Ungleichungen müssen erfüllt sein:

$$\|\mathbf{w}(n)\|^2 \geq \frac{(\widehat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta)^2}{\|\widehat{\mathbf{w}}\|^2} \quad \text{und} \quad \|\mathbf{w}(n)\|^2 \leq \|\mathbf{w}(0)\|^2 + n\eta^2\varepsilon, \text{ also}$$

$$\frac{(\widehat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta)^2}{\|\widehat{\mathbf{w}}\|^2} \leq \|\mathbf{w}(n)\|^2 \leq \|\mathbf{w}(0)\|^2 + n\eta^2\varepsilon$$

10. Es gilt aber:

$$\|\mathbf{w}(0)\|^2 + n\eta^2\varepsilon \in \Theta(n) \quad \text{und} \quad \frac{(\widehat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta)^2}{\|\widehat{\mathbf{w}}\|^2} \in \Theta(n^2)$$

→ Der Perzeptron-Lernalgorithmus zum Anpassen von  $\mathbf{w}$  terminiert in endlich vielen Schritten, da keine fehlklassifizierten Beispiele mehr existieren können.

11. Überlegung, wie groß  $n$  werden kann:

$$\frac{(\widehat{\mathbf{w}}^T \mathbf{w}(0) + n\eta\delta)^2}{\|\widehat{\mathbf{w}}\|^2} \leq \|\mathbf{w}(0)\|^2 + n\eta^2\varepsilon$$

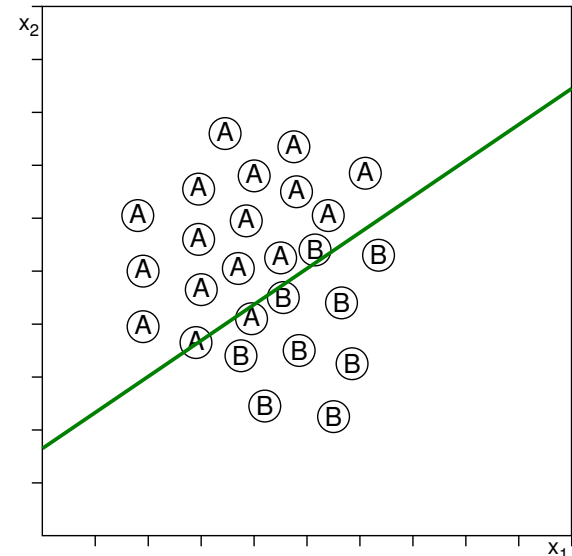
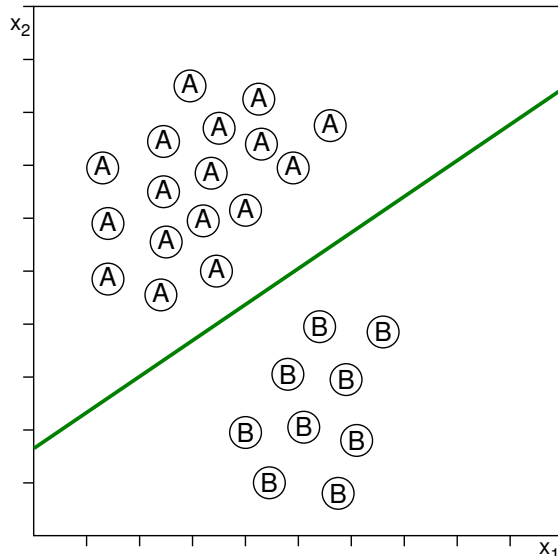
Für  $\mathbf{w}(0) = \mathbf{0}$  (alle Anfangsgewichte = 0) erhält man:

$$0 < n \leq \frac{\varepsilon}{\delta^2} \|\widehat{\mathbf{w}}\|^2$$

# Perzeptron-Lernalgorithmus

## Zusammenfassung

- Existiert eine trennende Hyperebene zwischen  $D_0$  und  $D_1$  konvergiert der Perzeptron-Lernalgorithmus; existiert keine trennende Hyperebene, kann keine Konvergenz garantiert werden.
- Mittels linearer Programmierung kann eine trennende Hyperebene in polynomieller Zeit bestimmt werden; der Perzeptron-Lernalgorithmus kann exponentiell viele Schritte benötigen.
- Reale Klassifikationsprobleme (rechts) sind oft problematisch:



# Gradientenabstiegsmethode

## Lernfehler

Ein Gradientenabstieg berücksichtigt den tatsächlichen Fehler und konvergiert auch dann, wenn  $D_1$  und  $D_0$  nicht durch eine Hyperebene trennbar sind. Diese Konvergenz erfolgt jedoch asymptotisch und ist nicht durch endlich viele Schritte abschätzbar.

Die Gradientenabstiegsmethode basiert auf der sogenannten Delta-Regel, die wiederum die Basis des Back-Propagation-Algorithmus darstellt.

# Gradientenabstiegsmethode

## Lernfehler

Ein Gradientenabstieg berücksichtigt den tatsächlichen Fehler und konvergiert auch dann, wenn  $D_1$  und  $D_0$  nicht durch eine Hyperebene trennbar sind. Diese Konvergenz erfolgt jedoch asymptotisch und ist nicht durch endlich viele Schritte abschätzbar.

Die Gradientenabstiegsmethode basiert auf der sogenannten Delta-Regel, die wiederum die Basis des Back-Propagation-Algorithmus darstellt.

Betrachtung eines linearen Perzeptrons *ohne* Schwellwertfunktion:

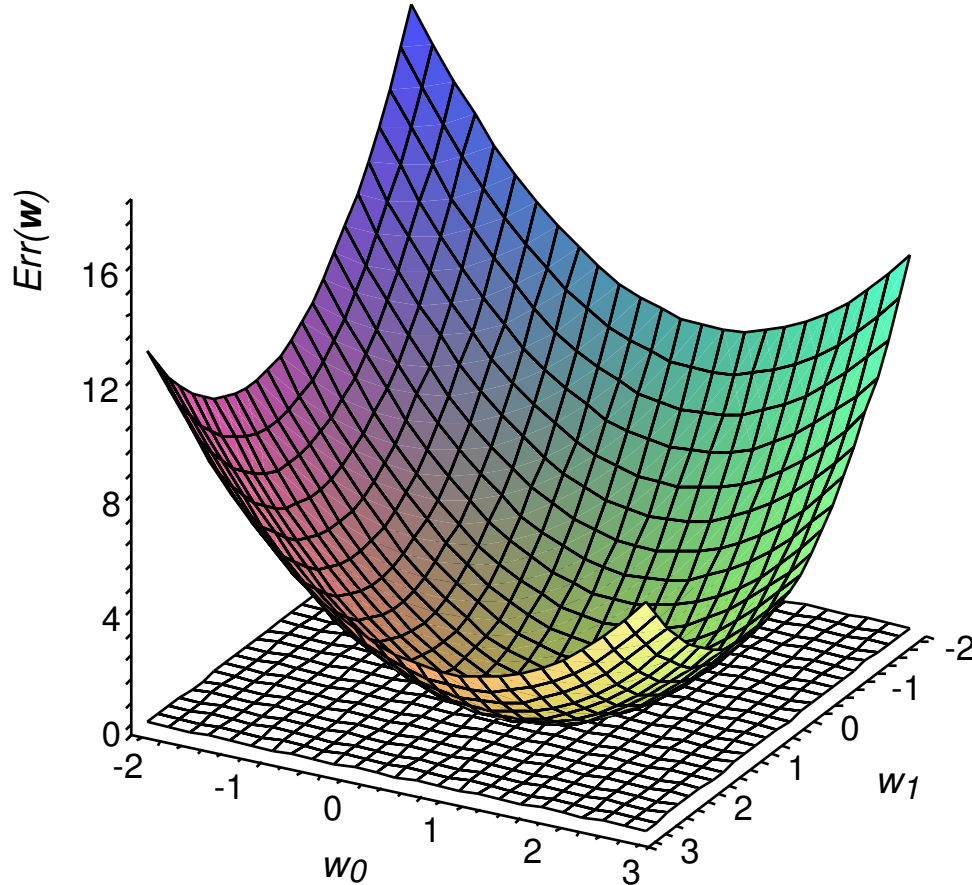
$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = \sum_{i=0}^p w_i x_i$$

Der Lernfehler  $Err(\mathbf{w})$  eines Gewichtsvektors  $\mathbf{w}$  (= Hypothese) sei wie folgt definiert:

$$Err(\mathbf{w}) = \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y(\mathbf{x}))^2$$

# Gradientenabstiegsmethode

## Lernfehler



Der Gradient  $\nabla Err(\mathbf{w})$  von  $Err(\mathbf{w})$  definiert den steilsten Abstieg:

$$\nabla Err(\mathbf{w}) = \left( \frac{\partial Err(\mathbf{w})}{\partial w_0}, \frac{\partial Err(\mathbf{w})}{\partial w_1}, \dots, \frac{\partial Err(\mathbf{w})}{\partial w_p} \right)$$

# Gradientenabstiegsmethode

Gewichtsanpassung [vgl. Algorithmus PT]

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w} \quad \text{mit} \quad \Delta \mathbf{w} = -\eta \nabla \text{Err}(\mathbf{w})$$

Komponentenweise Gewichtsanpassung:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = -\eta \frac{\partial \text{Err}(\mathbf{w})}{\partial w_i}$$



# Gradientenabstiegsmethode

Gewichtsanpassung [[vgl. Algorithmus PT](#)]

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w} \quad \text{mit} \quad \Delta \mathbf{w} = -\eta \nabla \text{Err}(\mathbf{w})$$

Komponentenweise Gewichts-anpassung:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = -\eta \frac{\partial \text{Err}(\mathbf{w})}{\partial w_i}$$

$$\begin{aligned} \frac{\partial \text{Err}(\mathbf{w})}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y(\mathbf{x}))^2 = \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} \frac{\partial}{\partial w_i} (c(\mathbf{x}) - y(\mathbf{x}))^2 \\ &= \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} 2(c(\mathbf{x}) - y(\mathbf{x})) \frac{\partial}{\partial w_i} (c(\mathbf{x}) - y(\mathbf{x})) \\ &= \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \frac{\partial}{\partial w_i} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \\ &= \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) (-x_i) \end{aligned}$$

# Gradientenabstiegsmethode

Gewichtsanpassung [[vgl. Algorithmus PT](#)]

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w} \quad \text{mit} \quad \Delta \mathbf{w} = -\eta \nabla \text{Err}(\mathbf{w})$$

Komponentenweise Gewichtsanzpassung:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = -\eta \frac{\partial \text{Err}(\mathbf{w})}{\partial w_i} = \eta \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot x_i$$

$$\begin{aligned} \frac{\partial \text{Err}(\mathbf{w})}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y(\mathbf{x}))^2 = \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} \frac{\partial}{\partial w_i} (c(\mathbf{x}) - y(\mathbf{x}))^2 \\ &= \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} 2(c(\mathbf{x}) - y(\mathbf{x})) \frac{\partial}{\partial w_i} (c(\mathbf{x}) - y(\mathbf{x})) \\ &= \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \frac{\partial}{\partial w_i} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \\ &= \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) (-x_i) \end{aligned}$$

# Gradientenabstiegsmethode

## Gewichtsanpassung: Batch Gradient Descent

Algorithm: *BGD* Batch Gradient Descent

Input:  $D$  Lernbeispiele der Form  $(\mathbf{x}, c(\mathbf{x}))$  mit  $|\mathbf{x}| = 1 + p$ ,  $c(\mathbf{x}) \in \{0, 1\}$ .  
 $\eta$  positive kleine Konstante (Lernrate).

Output:  $\mathbf{w}$  Gewichtsvektor.

*BGD*( $D, \eta$ )

1. *initialize\_random\_weights*( $\mathbf{w}$ ),  $t = 0$
2. **REPEAT**
3.  $t = t + 1$
4. **FOR**  $i = 0$  **TO**  $p$  **DO**  $\Delta w_i = 0$
5. **FOREACH**  $(\mathbf{x}, c(\mathbf{x})) \in D$  **DO**
6.  $\text{error} = c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}$
7. **FOR**  $i = 0$  **TO**  $p$  **DO**  $\Delta w_i = \Delta w_i + \eta \cdot \text{error} \cdot x_i$
8. **ENDDO**
9. **FOR**  $i = 0$  **TO**  $p$  **DO**  $w_i = w_i + \Delta w_i$
10. **UNTIL**(*convergence*( $D, Y$ )  $\vee t > t_{\max}$ )

# Gradientenabstiegsmethode

## Gewichtsanpassung: Delta-Regel

Die Formel für die Gewichtsanpassung ist mengenbasiert (Stichwort: Batch). Vor einer Gewichtsanpassung wird der Fehler *aller* Beispiele aufsummiert.

Gewichtsanpassung bezogen auf *ein* Beispiel  $(\mathbf{x}, c(\mathbf{x})) \in D$ :

$$\Delta w_i = \eta \cdot (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot x_i$$

Diese Anpassungsregel hat verschiedene Namen:

- Delta-Regel
- Widrow-Hoff-Regel
- Adaline-Regel

Als Lernfehler  $Err_d(\mathbf{w})$  eines Gewichtsvektors  $\mathbf{w}$  (= Hypothese) bezogen auf *ein* Beispiel  $d \in D$ ,  $d = (\mathbf{x}, c(\mathbf{x}))$  ergibt sich:

$$Err_d(\mathbf{w}) = \frac{1}{2}(c(\mathbf{x}) - \mathbf{w}^T \mathbf{x})^2$$

# Gradientenabstiegsmethode

## Gewichtsanpassung: Incremental Gradient Descent

Algorithm: *IGD* Incremental Gradient Descent

Input:  $D$  Lernbeispiele der Form  $(\mathbf{x}, c(\mathbf{x}))$  mit  $|\mathbf{x}| = 1 + p$ ,  $c(\mathbf{x}) \in \{0, 1\}$ .  
 $\eta$  positive kleine Konstante (Lernrate).

Output:  $\mathbf{w}$  Gewichtsvektor.

*IGD*( $D, \eta$ )

1. *initialize\_random\_weights*( $\mathbf{w}$ ),  $t = 0$
2. **REPEAT**
3.  $t = t + 1$
4. **FOREACH**  $(\mathbf{x}, c(\mathbf{x})) \in D$  **DO**
5.  $\mathit{error} = c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}$
6. **FOR**  $i = 0$  **TO**  $p$  **DO**
7.  $\Delta w_i = \eta \cdot \mathit{error} \cdot x_i$ ,  $w_i = w_i + \Delta w_i$
8. **ENDDO**
9. **ENDDO**
10. **UNTIL**(*convergence*( $D, Y$ )  $\vee t > t_{\max}$ )

## Bemerkungen:

- ❑ Die inkrementelle Gradientenabstiegsmethode wird auch stochastische Gradientenabstiegsmethode genannt.
- ❑ Die Lernfehler  $Err$  der inkrementellen Gradientenabstiegsmethode ist spezifisch für jedes Trainingsbeispiel  $d \in D$ ,  $d = (\mathbf{x}, c(\mathbf{x}))$ :  $Err_d(\mathbf{w}) = \frac{1}{2}(c(\mathbf{x}) - \mathbf{w}^T \mathbf{x})^2$
- ❑ Die Abfolge der inkrementellen Gewichtsadjustierungen ist eine Annäherung an den Gradientenabstieg der mengenbasierten Methode. Durch entsprechend kleine Werte für  $\eta$  kann diese Annäherung beliebig genau erfolgen.
- ❑ Beachte, dass die genauere Berechnung des Fehlergradienten in der mengenbasierten Methode größere Schritte bei Gewichtsadjustierungen ermöglicht.
- ❑ Mit der inkrementellen Gradientenabstiegsmethode kann bei Vorhandensein mehrerer lokaler Minima ein „Hängenbleiben“ eher vermieden werden als mit der mengenbasierten Methode.

## IV. Neuronale Netze

- Perzeptron-Lernalgorithmus
- Gradientenabstiegsmethode
- Multilayer-Perzeptron
- Self-Organizing Feature Maps
- Neuronales Gas
- Radialbasisfunktionen

# Multilayer-Perzeptron

## Definition 6 (lineare Separierbarkeit)

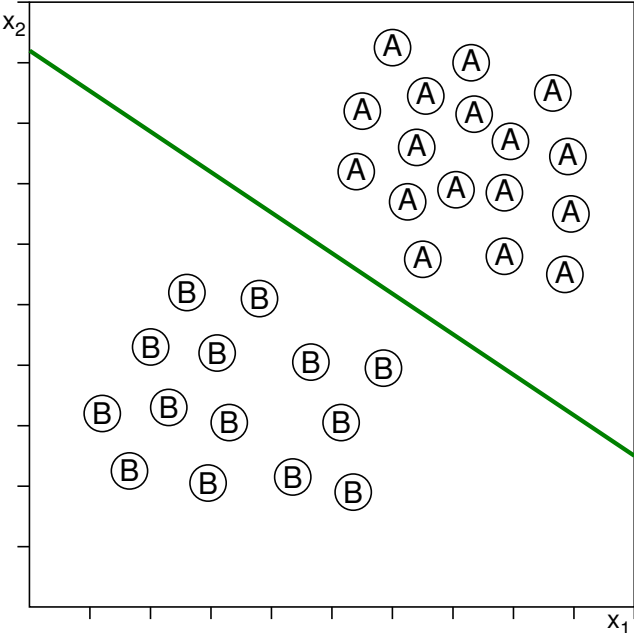
Zwei Mengen  $D_1$  und  $D_0$  von Vektoren eines  $p$ -dimensionalen Raumes heißen linear separierbar, falls  $1 + p$  reelle Zahlen  $\theta, w_1, \dots, w_p$  existieren, so dass  $\sum_{i=1}^p w_i x_i \geq \theta$  für jeden Vektor in  $D_1$  und  $\sum_{i=1}^p w_i x_i < \theta$  für jeden Vektor in  $D_0$  gilt.



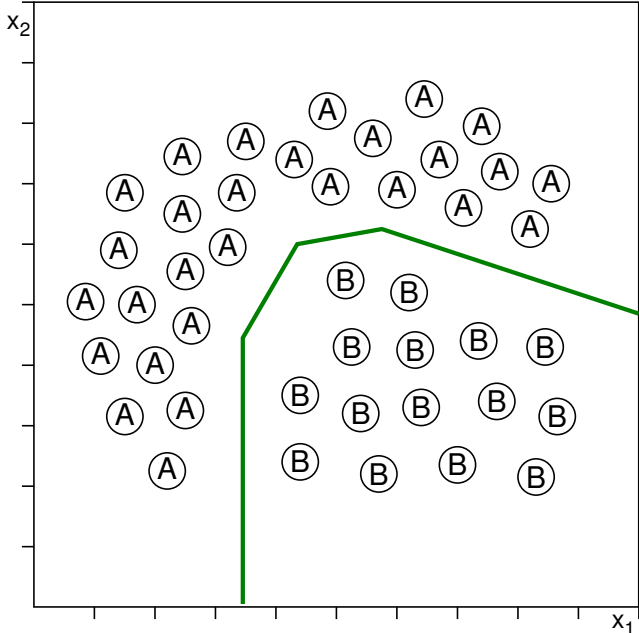
# Multilayer-Perzeptron

## Definition 6 (lineare Separierbarkeit)

Zwei Mengen  $D_1$  und  $D_0$  von Vektoren eines  $p$ -dimensionalen Raumes heißen linear separierbar, falls  $1 + p$  reelle Zahlen  $\theta, w_1, \dots, w_p$  existieren, so dass  $\sum_{i=1}^p w_i x_i \geq \theta$  für jeden Vektor in  $D_1$  und  $\sum_{i=1}^p w_i x_i < \theta$  für jeden Vektor in  $D_0$  gilt.



linear separierbar



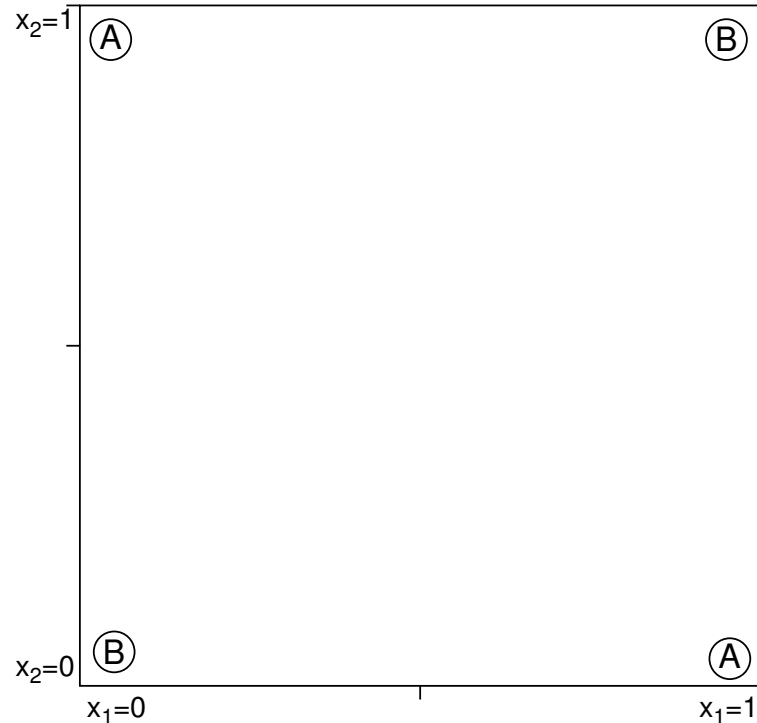
nicht linear separierbar

# Multilayer-Perzeptron

## Separierbarkeit

Die *XOR*-Funktion definiert ein kleinstes Beispiel für nicht linear separierbare Mengen:

$x_1$	$x_2$	XOR (Klasse)
0	0	0 (B)
1	0	1 (A)
0	1	1 (A)
1	1	0 (B)

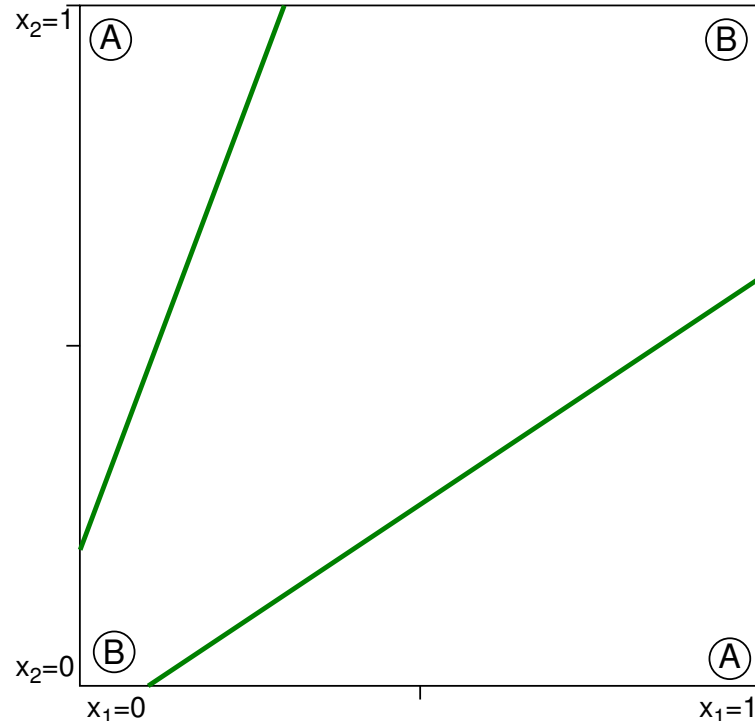


# Multilayer-Perzeptron

## Separierbarkeit

Die *XOR*-Funktion definiert ein kleinstes Beispiel für nicht linear separierbare Mengen:

$x_1$	$x_2$	XOR (Klasse)
0	0	0 (B)
1	0	1 (A)
0	1	1 (A)
1	1	0 (B)



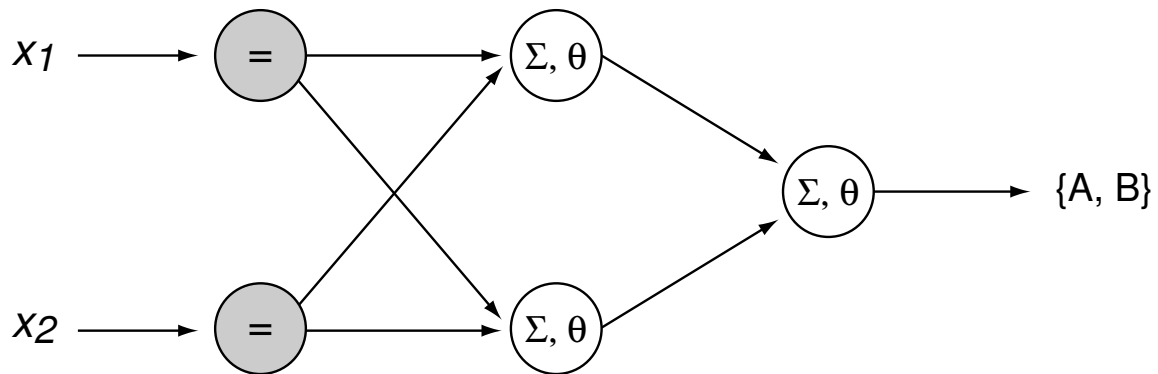
- Definition mehrerer Geraden (Hyperebenen)
- Verwendung mehrerer Perzeptrons

# Multilayer-Perzeptron

## Separierbarkeit (Fortsetzung)

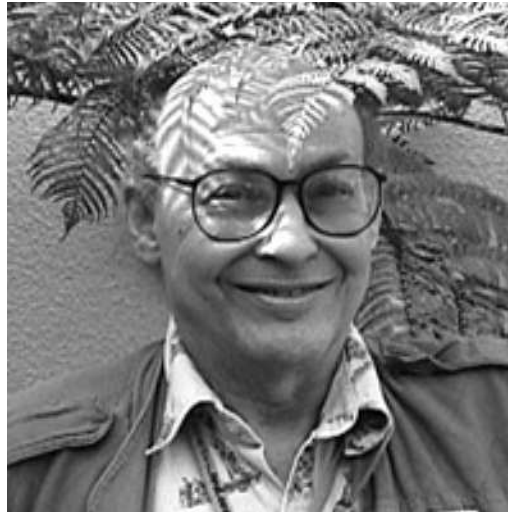
Verschaltung von Perzeptrons in mehreren Schichten: Multilayer-Perzeptron.

Das einfachste Multilayer-Perzeptron für das *XOR*-Problem:



## Bemerkungen:

- ❑ Das Konzept des Multilayer-Perzeptrons wurde 1986 von Rumelhart & McClelland vorgeschlagen. Früher – aber unbeachtet geblieben – waren die Arbeiten Werbos und Parker [1974, 1982].
- ❑ Bei einem Multilayer-Perzeptron liegt eine komplexere Lernsituation vor, für die kontinuierliche Schwellwertfunktionen und stärkere Lernverfahren benötigt werden.
- ❑ Marvin Minsky und Seymour Papert zeigten 1969 am Beispiel des *XOR*-Problems die Grenzen des einzelnen Perzeptrons. Insbesondere vermuteten sie, dass eine Erweiterung der Perzeptron-Architektur ähnlichen Beschränkungen unterläge wie ein einzelnes Perzeptron. Ein Irrtum. Sie brachten damit die Forschung in diesem Bereich für 17 Jahre zum Stillstand.

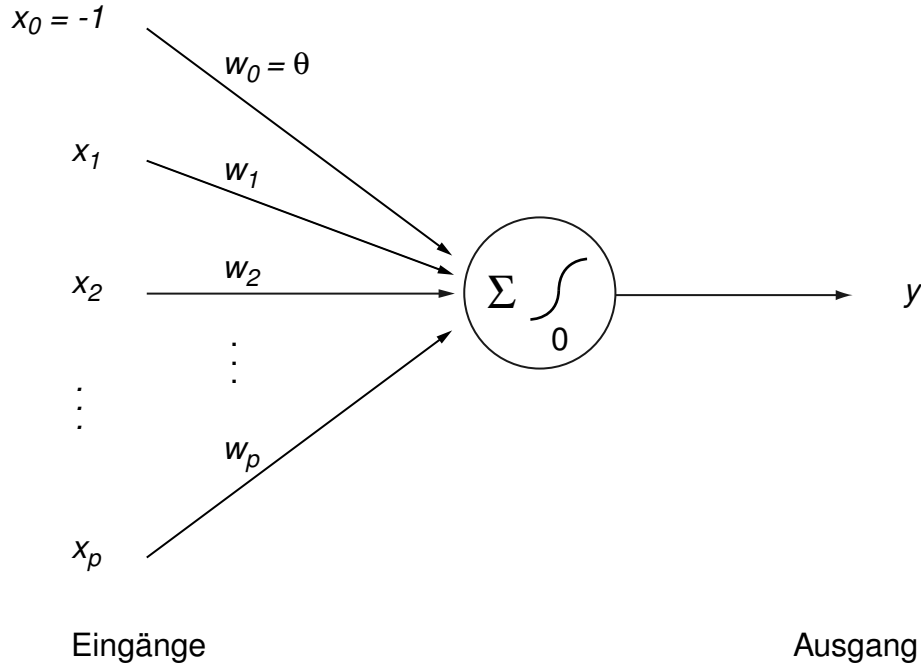


[Marvin Minsky]

# Multilayer-Perzeptron

## Berechnung im Netzwerk

Perzeptron mit kontinuierlicher Schwellwertfunktionen:



Sigmoid-Funktion  $\sigma(z)$  als Schwellwertfunktion:

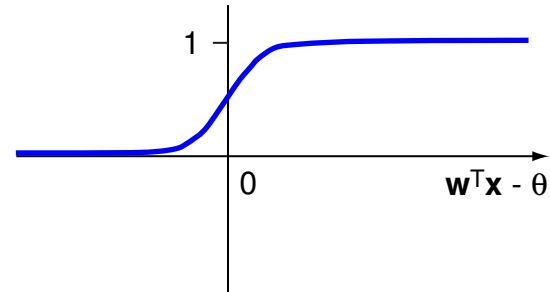
$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \text{mit der Eigenschaft:} \quad \frac{d\sigma(z)}{dz} = \sigma(z) \cdot (1 - \sigma(z))$$

# Multilayer-Perzeptron

## Berechnung im Netzwerk (Fortsetzung)

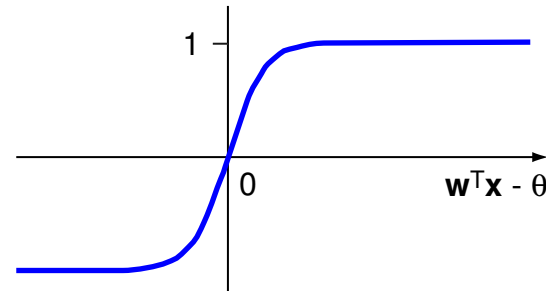
Berechnung des Perzeptronausgangs  $y(\mathbf{x})$  mittels der Sigmoid-Funktion  $\sigma$ :

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$



Anstelle der Sigmoid-Funktion wird auch die tanh-Funktion verwendet:

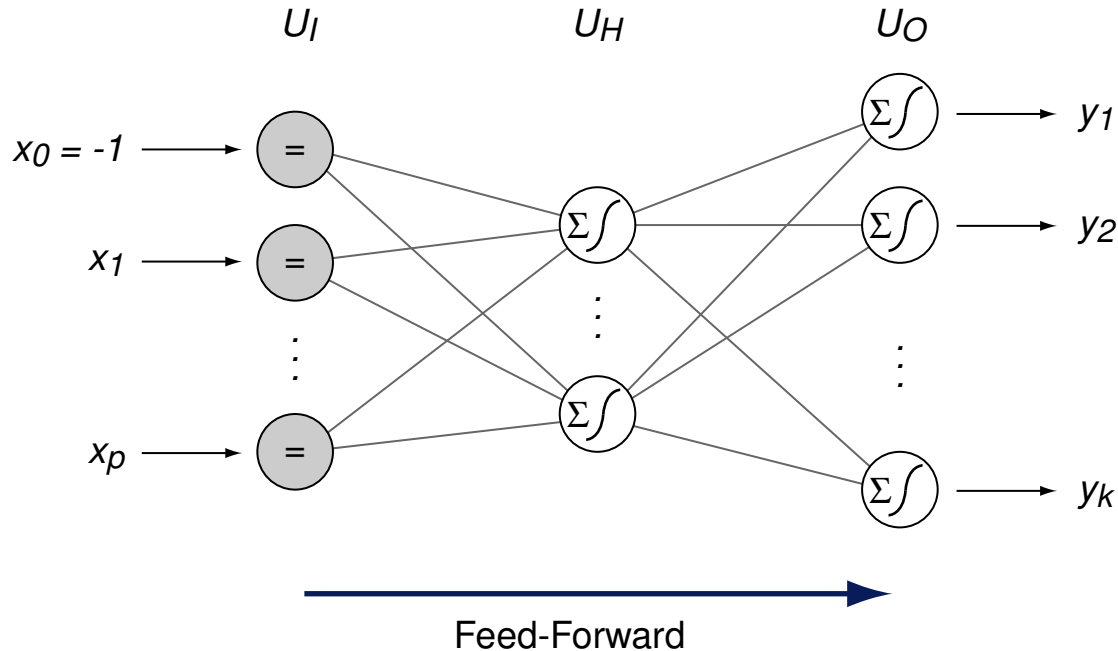
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



# Multilayer-Perzeptron

## Berechnung im Netzwerk (Fortsetzung)

Unterscheidung von Input-, Hidden- und Output-Units (-Perzeptrons):



$U_I, U_H, U_O$

Mengen von Input-, Hidden- und Output-Units

$x_{u \rightarrow v}$

Eingangswert für Unit  $v$ , bereitgestellt am Ausgang von Unit  $u$

$w_{uv}, \Delta w_{uv}$

Gewicht und Gewichtsanzpassung für Kante zwischen Unit  $u$  und Unit  $v$

$\delta_u$

Lernfehler von Unit  $u$

$y_u$

Ausgangswert von Unit  $u$



## Bemerkungen:

- ❑ Die Units der Input-Schicht führen keine Berechnungen aus; sie dienen lediglich zur Verteilung der Eingangsdaten an die nächste Schicht.
- ❑ Die Nichtlinearität der Sigmoid-Funktion ermöglicht Netzwerke, die jede beliebige Funktion approximieren können. Hierzu sind lediglich drei aktive Schichten notwendig – genauer: zwei Schichten mit Hidden-Units und eine Schicht mit Output-Units.  
Stichwort: Kolmogorov-Theorem
- ❑ Auf Basis der verallgemeinerten Delta-Regel wird eine Rückwärtspropagierung des Lernfehlers und somit ein Trainieren mehrschichtiger Netzwerke möglich.  
Stichwort: Backpropagation-Algorithmus
- ❑ Der Gradientenabstieg bzgl. der Fehlerfunktion berücksichtigt den gesamten Netzwerkgewichtsvektor.
- ❑ Multilayer-Perzeptrons werden auch Multilayer-Netzwerke oder (künstliche) neuronale Netze genannt. Eine gebräuchliche Abkürzung in diesem Zusammenhang ist ANN, für „Artificial Neural Network“.

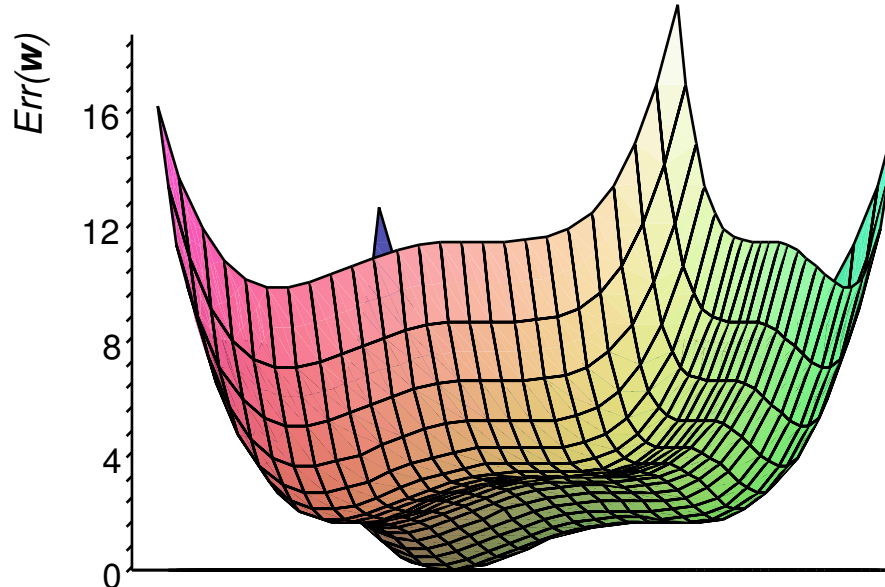
# Multilayer-Perzeptron

## Lernfehler

Der Lernfehler  $Err(\mathbf{w})$  summiert sich über die  $|U_O| = k$  Ausgänge des Netzes:

$$Err(\mathbf{w}) = \frac{1}{2} \sum_{(\mathbf{x}, \mathbf{c}(\mathbf{x})) \in D} \sum_{v \in U_O} (c_v(\mathbf{x}) - y_v(\mathbf{x}))^2$$

Durch die komplexe Form von  $Err(\mathbf{w})$  gibt es i. d. R. viele lokale Minima:



# Multilayer-Perzeptron

## Gewichtsanpassung: Incremental Gradient Descent

Input:  $D$  Lernbeispiele der Form  $(\mathbf{x}, c(\mathbf{x}))$  mit  $|\mathbf{x}| = 1 + p$ ,  $c(\mathbf{x}) \in \{0, 1\}$ .  
 $\eta$  positive kleine Konstante (Lernrate).

Output:  $w$  Gewichte für Mengen  $U_I, U_H, U_O$  von Input-, Hidden- und Output-Units.

1. *initialize\_random\_weights*( $U_I, U_H, U_O$ ),  $t = 0$
2. **REPEAT**
3.    $t = t + 1$
4.   **FOREACH**  $(\mathbf{x}, c(\mathbf{x})) \in D$  **DO**
5.     **FOREACH**  $u \in U_H$  **DO**  $y_u = \text{propagate}(\mathbf{x}, u)$  // layer 1
6.     **FOREACH**  $v \in U_O$  **DO**  $y_v = \text{propagate}(\mathbf{y}_H, v)$  // layer 2
- 7.
- 8.
- 9.
- 10.
- 11.
- 12.
13.   **ENDDO**
14. **UNTIL**(*convergence*( $D, Y_O$ )  $\vee t > t_{\max}$ )

# Multilayer-Perzeptron

## Gewichtsanpassung: Incremental Gradient Descent

Input:  $D$  Lernbeispiele der Form  $(\mathbf{x}, c(\mathbf{x}))$  mit  $|\mathbf{x}| = 1 + p$ ,  $c(\mathbf{x}) \in \{0, 1\}$ .  
 $\eta$  positive kleine Konstante (Lernrate).

Output:  $w$  Gewichte für Mengen  $U_I, U_H, U_O$  von Input-, Hidden- und Output-Units.

1. *initialize\_random\_weights*( $U_I, U_H, U_O$ ),  $t = 0$
2. **REPEAT**
3.  $t = t + 1$
4. **FOREACH**  $(\mathbf{x}, c(\mathbf{x})) \in D$  **DO**
5.     **FOREACH**  $u \in U_H$  **DO**  $y_u = \text{propagate}(\mathbf{x}, u)$  // layer 1
6.     **FOREACH**  $v \in U_O$  **DO**  $y_v = \text{propagate}(\mathbf{y}_H, v)$  // layer 2
7.     **FOREACH**  $v \in U_O$  **DO**  $\delta_v = y_v \cdot (1 - y_v) \cdot (c_v(\mathbf{x}) - y_v)$  // backpropagate layer 2
8.     **FOREACH**  $u \in U_H$  **DO**  $\delta_u = y_u \cdot (1 - y_u) \sum_{v \in U_o} w_{uv} \cdot \delta_v$  // backpropagate layer 1
- 9.
- 10.
- 11.
- 12.
13.     **ENDDO**
14. **UNTIL**(*convergence*( $D, Y_O$ )  $\vee t > t_{\max}$ )

# Multilayer-Perzeptron

## Gewichtsanpassung: Incremental Gradient Descent

Input:  $D$  Lernbeispiele der Form  $(\mathbf{x}, c(\mathbf{x}))$  mit  $|\mathbf{x}| = 1 + p$ ,  $c(\mathbf{x}) \in \{0, 1\}$ .  
 $\eta$  positive kleine Konstante (Lernrate).

Output:  $w$  Gewichte für Mengen  $U_I, U_H, U_O$  von Input-, Hidden- und Output-Units.

1. *initialize\_random\_weights*( $U_I, U_H, U_O$ ),  $t = 0$
2. **REPEAT**
3.  $t = t + 1$
4. **FOREACH**  $(\mathbf{x}, c(\mathbf{x})) \in D$  **DO**
5.     **FOREACH**  $u \in U_H$  **DO**  $y_u = \text{propagate}(\mathbf{x}, u)$  // layer 1
6.     **FOREACH**  $v \in U_O$  **DO**  $y_v = \text{propagate}(\mathbf{y}_H, v)$  // layer 2
7.     **FOREACH**  $v \in U_O$  **DO**  $\delta_v = y_v \cdot (1 - y_v) \cdot (c_v(\mathbf{x}) - y_v)$  // backpropagate layer 2
8.     **FOREACH**  $u \in U_H$  **DO**  $\delta_u = y_u \cdot (1 - y_u) \sum_{v \in U_O} w_{uv} \cdot \delta_v$  // backpropagate layer 1
9.     **FOREACH**  $w_{uv}$ ,  $(u, v) \in (U_I \times U_H) \cup (U_H \times U_O)$  **DO**
10.          $\Delta w_{uv} = \eta \cdot \delta_v \cdot x_{u \rightarrow v}$
11.          $w_{uv} = w_{uv} + \Delta w_{uv}$
12.     **ENDDO**
13. **ENDDO**
14. **UNTIL**(*convergence*( $D, Y_O$ )  $\vee t > t_{\max}$ )

# Multilayer-Perzeptron

## Erweiterung der Gewichts Anpassung durch Momentum-Term

Die Gewichts Anpassung in Iteration  $t$  wird abhängig von der Anpassung in Iteration  $t - 1$  gemacht:

$$\Delta w_{uv}(t) = \eta \cdot \delta_v \cdot x_{u \rightarrow v} + \alpha \cdot \Delta w_{uv}(t - 1)$$

mit „Momentum“  $\alpha$ ,  $0 \leq \alpha < 1$

# Multilayer-Perzeptron

## Erweiterung der Gewichtsanzpassung durch Momentum-Term

Die Gewichtsanzpassung in Iteration  $t$  wird abhängig von der Anzpassung in Iteration  $t - 1$  gemacht:

$$\Delta w_{uv}(t) = \eta \cdot \delta_v \cdot x_{u \rightarrow v} + \alpha \cdot \Delta w_{uv}(t - 1)$$

mit „Momentum“  $\alpha$ ,  $0 \leq \alpha < 1$

Effekte:

- bedingt durch die „Trägheit“ können lokale Minima überwunden werden
- bei gleichbleibender Abstiegsrichtung erhöht sich die Schrittweite und damit die Konvergenzgeschwindigkeit

# Neuronale Netze

## Quellen zum Nachlernen und Nachschlagen im Web

### Anwendung und Implementierung:

- JNNS. Java Neural Network Simulator.

<http://www-ra.informatik.uni-tuebingen.de/software/JavaNNS>

- SNNS. Stuttgart Neural Network Simulator.

<http://www-ra.informatik.uni-tuebingen.de/software/snns>

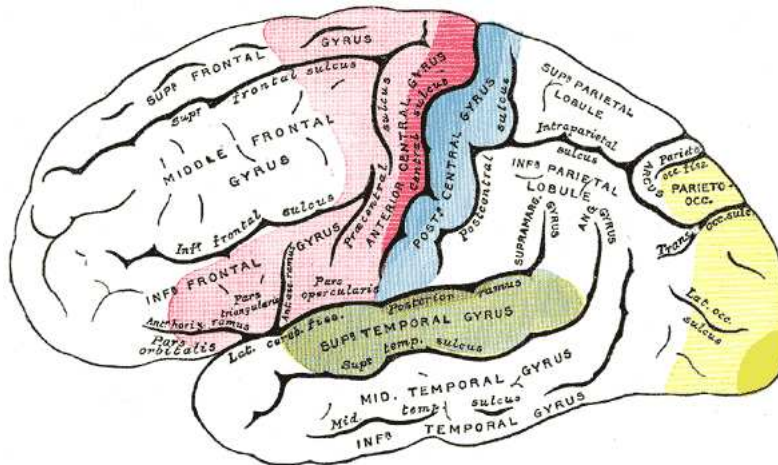


## IV. Neuronale Netze

- Perzeptron-Lernalgorithmus
- Gradientenabstiegsmethode
- Multilayer-Perzeptron
- Self-Organizing Feature Maps
- Neuronales Gas
- Radialbasisfunktionen

# Self-Organizing Feature Maps (SOMs)

Motivation: Räumliche Organisation von Aktivitäten im Gehirn



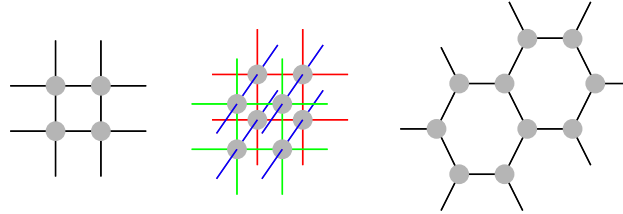
- primär-motorisches Areal
- prä/supplementär-motorisches Areale
- primär-sensible Areale
- sensible Assoziationsareale
- Hörfelder
- Sehfelder

- Strukturen im Gehirn oft linear oder planar, Signale multidimensional.
- Ähnliche Reize werden durch räumlich nahe Nervenzellen verarbeitet.
- Stärker genutzte Bereiche sind besser ausgebildet.

# Self-Organizing Feature Maps (SOMs)

Idee [Kohonen, 1982]

- Neuronen bilden die Knoten einer Gitterstruktur (typisch 2D, 3D), Kanten beschreiben Nachbarschaften.



- Abbildung von Eingangssignalen auf Erregungszustände von Neuronen. (Alle Neuronen im Gitter sind mit allen Inputknoten verbunden.)
  - Ähnliche Eingangssignale erregen „räumlich benachbarte“ Neuronen.
  - Unüberwachte Anpassung an Eingangssignale nach dem Prinzip der lateralen Inhibition (seitliche Hemmung):  
Erregungszustände von stark erregten Neuronen und deren unmittelbarer Nachbarschaft werden verstärkt, Erregungszustände entfernterer Neuronen werden gedämpft.
- Kartierung des Merkmalsraumes.

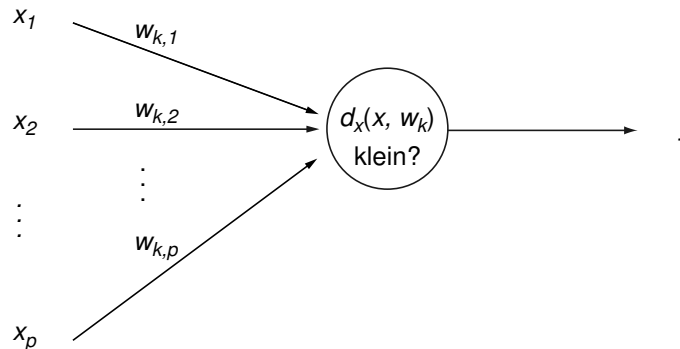
# Self-Organizing Feature Maps (SOMs)

## Formales Modell

- $D := \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq X = \mathbf{R}^p$  endliche Menge von Eingabestimuli (Merkmalsvektoren).
- $d_X : \mathbf{R}^p \times \mathbf{R}^p \rightarrow \mathbf{R}^+$  Metrik auf  $\mathbf{R}^p$ .

Beispiel: Euklidischer Abstand  $d(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^p (x_i - y_i)^2}$

- $N = \{N_1, \dots, N_K\}$  endliche Menge von Neuronen,  $N_k$  definiert durch Gewichtsvektoren  $\mathbf{w}_k \in \mathbf{R}^p$ .



- $d_N : N \times N \rightarrow \mathbf{R}^+$  Metrik auf  $N$  (Nachbarschaft).

Beispiel: Neuronen aus  $N$  liegen auf zweidimensionalen Gitter,  $d_N$  bestimmt den Euklidischen Abstand der Gitterpositionen oder die Manhattan-Distanz.

## Bemerkungen:

- Die Beispiele können auch zufällig aus dem Merkmalsraum gezogen werden nach einer festen Wahrscheinlichkeitsverteilung.

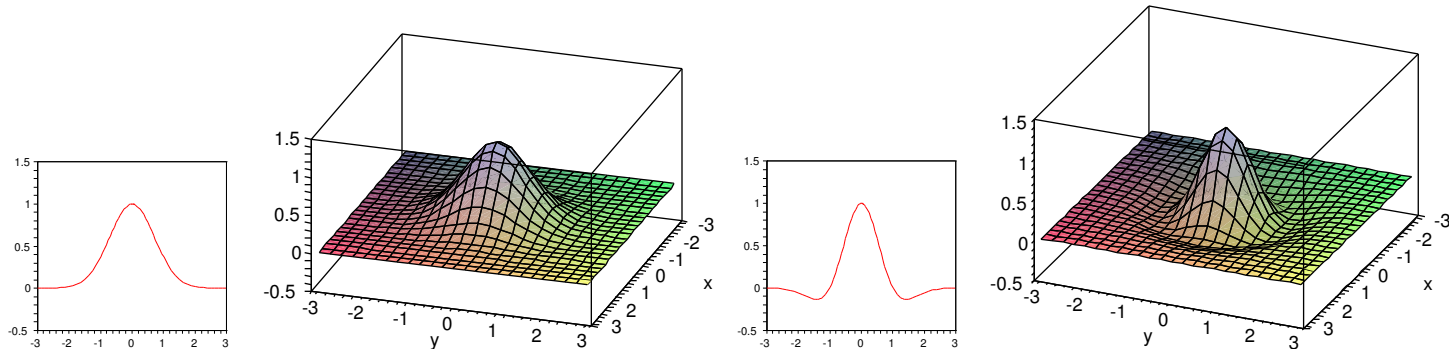
# Self-Organizing Feature Maps (SOMs)

## Formales Modell (Fortsetzung)

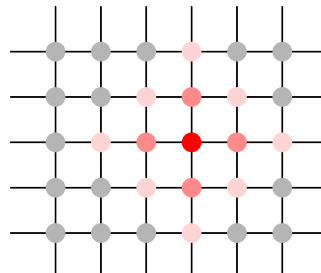
- $h(d, \sigma)$  Nachbarschaftsfunktion zur Realisierung der lateralen Inhibition: maximal für  $d = 0$ , monoton fallend.

Beispiel:  $h(d, \sigma) = \exp(-\frac{d^2}{2\sigma^2})$  (Gaußglocke mit Radius  $\sigma$  um 0) oder

$$h(d, \sigma) = \frac{1}{\sqrt{2\pi}\sigma^3} \left(1 - \frac{d^2}{\sigma^2}\right) \exp(-\frac{d^2}{2\sigma^2}) \text{ (Mexican Hat Function).}$$



Anwendung auf Gitter:



# Self-Organizing Feature Maps (SOMs)

## Formales Modell (Fortsetzung)

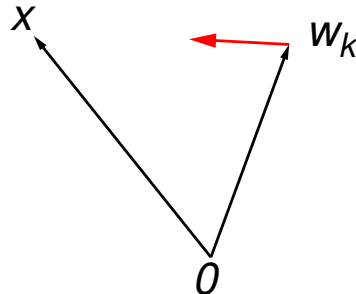
- $\sigma > 0$  bestimmt die „enge“ Nachbarschaft.

Beispiel:  $\sigma$  kann abhängig von der Runde  $t$  gewählt werden als  $\sigma(t) = \sigma_a \left( \frac{\sigma_e}{\sigma_a} \right)^{\frac{t}{t_{max}}}$  mit Anfangswert  $\sigma_a > 0$  und Endwert  $\sigma_e$  mit  $0 < \sigma_e \leq \sigma_a$ .

- $\eta > 0$  bestimmt die Lernrate (typisch  $\eta \in [0, 1]$ ).

Beispiel:  $\eta$  kann abhängig von der Runde  $t$  gewählt werden als  $\eta(t) = \eta_a \left( \frac{\eta_e}{\eta_a} \right)^{\frac{t}{t_{max}}}$  mit Anfangswert  $\eta_a > 0$  und Endwert  $\eta_e$  mit  $0 < \eta_e \leq \eta_a$ .

- $k_0(t) = \operatorname{argmin}_{k=1, \dots, K} d_X(\mathbf{x}, \mathbf{w}_k(t))$  bestimmt Index des Neuron mit dem geringsten „Abstand“ zum Eingangsstimulus  $\mathbf{x}$ .
- $\Delta \mathbf{w}_k = \eta(t) \cdot h(d_N(N_k, N_{k_0}), \sigma(t)) \cdot (\mathbf{x} - \mathbf{w}_k)$  Anpassung der Gewichte aller Neuronen.



## Bemerkungen:

- Gebräuchlich ist auch die sogenannte Bubble-Neighborhood, die Neuronen innerhalb eines gegebenen Radius  $\sigma$  mit 1 bewertet und alle anderen mit 0.
- Eine Anpassung der Nachbarschaftsfunktion  $h$  an die Rundenzahl  $t$  erfolgt indirekt durch den Parameter  $\sigma(t)$  in  $h(d, \sigma(t))$ .



# Self-Organizing Feature Maps (SOMs)

## Algorithmus zur Gewichts Anpassung

Sei  $D$  eine Menge von Trainingsbeispielen,  $\eta$  eine positive kleine Konstante, die Lernrate,  $\sigma$  eine positive Konstante, der Nachbarschaftsradius, und  $p$  die Dimension der Merkmalsvektoren.

*som\_training*( $D, \eta, \sigma$ )

1. *initialize\_random\_weights*( $\mathbf{w}_1, \dots, \mathbf{w}_K$ ),  $t = 0$ ;
2. **REPEAT**
3.  $t = t + 1$ ,  $\mathbf{x} = \textit{random\_select}(D)$ ;
4.  $k_0 = \textit{argmin}_{k=1, \dots, K} d_X(\mathbf{x}, \mathbf{w}_k)$ ;
5. **FOR**  $k = 1$  **TO**  $K$  **DO**
6.     **FOR**  $i = 0$  **TO**  $p$  **DO**
7.          $\Delta w_{k,i} = \eta \cdot h(d_N(N_k, N_{k_0}), \sigma) \cdot (x_i - w_{k,i})$ ;
8.          $w_{k,i} = w_{k,i} + \Delta w_{k,i}$ ;
9.     **ENDDO**
10. **ENDDO**
11. **UNTIL**( $t > t_{\max}$ );

## Bemerkungen:

- ❑ Die Initialisierung der Gewichte der Neuronen kann mit kleinen Zufallswerten erfolgen.
- ❑ Die Initialisierung der Gewichte der Neuronen kann mit zufällig gezogenen Elementen der Trainingsmenge erfolgen.
- ❑ Die Initialisierung der Gewichte der Neuronen kann durch Belegung mit linear geordneten Werten des durch die beiden größten Eigenwerte der Matrix der Trainingsmenge aufgespannten Teilraumes erfolgen.

# Self-Organizing Feature Maps (SOMs)

## Algorithmus zur Gewichtsanzpassung

Natürlichsprachliche Formulierung:

1. Initialisiere die Gewichte der Neuronen in der SOM.
2. Wähle zufällig einen Eingabestimulus  $\mathbf{x}$  aus  $D$ .
3. Bestimme das aktuelle Erregungszentrum  $N_{k_0}$  (Neuron mit ähnlichstem Gewichtsvektor).
4. Passe den Gewichtsvektor des Erregungszentrums und seiner Nachbarschaft mit nach außen abnehmender Stärke an den Eingabestimulus an.

$$\mathbf{w}_k = \mathbf{w}_k + \eta \cdot h(d_N(N_k, N_{k_0}), \sigma) \cdot (\mathbf{x} - \mathbf{w}_k)$$

5. ( Reduziere Lernrate  $\eta$  und Nachbarschaftsgröße  $\sigma$ . )
6. Falls Trainingsphase nicht zu Ende, gehe zu Schritt 2.

# Self-Organizing Feature Maps (SOMs)

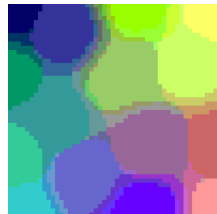
## Beispiel: Dimensionsreduktion mit SOMs

- 3-dimensionaler Integer-Merkmalraum:  $[0, 5] \times [0, 5] \times [0, 5]$  dargestellt durch Farben ( $6^3 = 216$  Elemente).
- 15 Eingabevektoren werden gleichverteilt zufällig gezogen.
- SOM mit zweidimensionalem Gitter mit  $50 \times 50$  Neuronen.
- Initialisierung der Neuronen mit
  - Zufallszahlen oder
  - Farbverlauf mit Farben rot, gelb, grün, schwarz in Ecken oder
  - Farbverlauf mit drei Zentren rot, gelb grün.
- Abbildung Gewichte auf Farben durch Rundung (Farben entsprechen Größenordnung der Gewichte).

# Self-Organizing Feature Maps (SOMs)

## Beispiel: Dimensionsreduktion mit SOMs

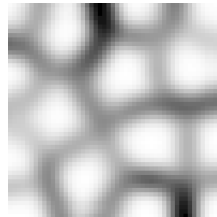
- ❑ 3-dimensionaler Integer-Merkmalraum:  $[0, 5] \times [0, 5] \times [0, 5]$  dargestellt durch Farben ( $6^3 = 216$  Elemente).
  - ❑ 15 Eingabevektoren werden gleichverteilt zufällig gezogen.
  - ❑ SOM mit zweidimensionalem Gitter mit  $50 \times 50$  Neuronen.
  - ❑ Initialisierung der Neuronen mit
    - Zufallszahlen oder
    - Farbverlauf mit Farben rot, gelb, grün, schwarz in Ecken oder
    - Farbverlauf mit drei Zentren rot, gelb grün.
  - ❑ Abbildung Gewichte auf Farben durch Rundung (Farben entsprechen Größenordnung der Gewichte).
- Ausbildung von Regionen in der SOM für die einzelnen Beispiele.



# Self-Organizing Feature Maps (SOMs)

## Beispiel: Dimensionsreduktion mit SOMs (Fortsetzung)

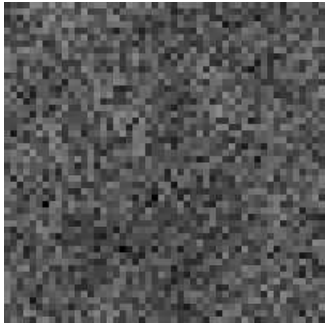
- ❑ Cluster werden gebildet von Vektoren, die nahe beieinander liegen im Vergleich zu allen übrigen Vektoren.
  - ❑ Jedes Neuron im Gitter der SOM repräsentiert einen Bereich des Eingaberaumes.  
Das Gewicht des Neuron kann als Repräsentant dieses Bereiches aufgefasst werden.
  - ❑ Ähnlichkeit von Gewichten zwischen benachbarten Neuronen kann visualisiert werden:  
Grauwert steht für durchschnittliche Distanz zu unmittelbaren Nachbarn.
- Visualisierung zeigt Schärfe der Trennung von benachbarten Clustern: Uniform Distance Matrix (U-Matrix).



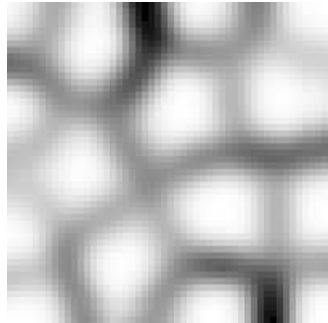
# Self-Organizing Feature Maps (SOMs)

Beispiel: Dimensionsreduktion mit SOMs (Fortsetzung)

Applet-Ansicht: soms.cl...  
Applet  
Percentage Complete = 0%  
Number of Iterations = 1000  
Random Points  
Reset  
 Colored SOM  Similarity SOM  
Applet gestartet



Applet-Ansicht: soms.cl...  
Applet  
Percentage Complete = 100%  
Number of Iterations = 1000  
Random Points  
Reset  
 Colored SOM  Similarity SOM  
Applet gestartet



[Matthew Ward, Computer Science Department, Worcester Polytechnic Institute (WPI), Worcester, MA]

<http://davis.wpi.edu/~matt/courses/soms//applet.html>

# Self-Organizing Feature Maps (SOMs)

## Beispiel: Spezialfall 2D-SOMs

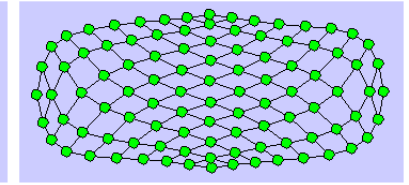
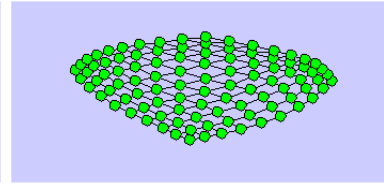
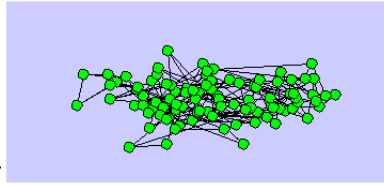
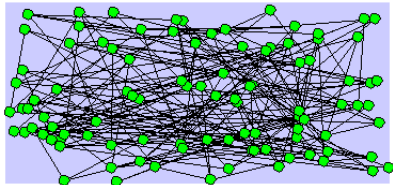
- 2-dimensionale Merkmalsvektoren, also  $D \subset \mathbb{R}^2$ :  
Eingabestimuli bezeichnen Positionen in der Ebene.
- 2-dimensionale Gewichtsvektoren können ebenfalls als Positionen in der Ebene aufgefasst werden:  
durch einzelne Neuronen repräsentierter Eingaberaum unmittelbar erkennbar.
- Unmittelbare Nachbarschaftsbeziehungen zwischen Neuronen werden durch Linien gekennzeichnet:  
SOM liegt als Gitternetz auf der Ebene des Eingaberaumes.



# Self-Organizing Feature Maps (SOMs)

## Beispiel: Spezialfall 2D-SOMs

- 2-dimensionale Merkmalsvektoren, also  $D \subset \mathbf{R}^2$ :  
Eingabestimuli bezeichnen Positionen in der Ebene.
  - 2-dimensionale Gewichtsvektoren können ebenfalls als Positionen in der Ebene aufgefasst werden:  
durch einzelne Neuronen repräsentierter Eingaberaum unmittelbar erkennbar.
  - Unmittelbare Nachbarschaftsbeziehungen zwischen Neuronen werden durch Linien gekennzeichnet:  
SOM liegt als Gitternetz auf der Ebene des Eingaberaumes.
- Erregungszentrum der SOM leicht erkennbar. Gewichtsanzpassung erscheint als Ziehen am Netzknoten in Richtung Eingabestimulus.
- Netz entfaltet sich und passt sich an Eingaberaum an. (vgl. Spring Embedder)



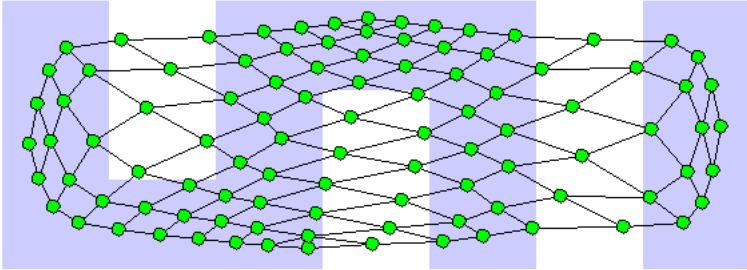
# Self-Organizing Feature Maps (SOMs)

## Beispiel: Spezialfall 2D-SOMs (Fortsetzung)

*DemoGNG*, a Java applet, implements several methods related to competitive learning. It is possible to experiment with the methods using various data distributions and observe the learning process. A common terminology is used to make it easy to compare one method to the other.

79300 v1.5

Network Model: **Self-Organizing Map**



100 Nodes

**Start** **Stop** **Reset**  Teach  Signals  Voronoi  Delaunay

Error Graph  Nodes  Edges  Random Init  White  Sound

prob. Distrib.: **UNI** max. Nodes: **100** Display: **100** Speed: **Lightning**

Grid size	epsilon_i	epsilon_f	sigma_i	sigma_f	t_max
10x10	0.1	0.0050	5.0	0.2	40000.0

**Authors: Hartmut S. Loos Bernd Fritzke**

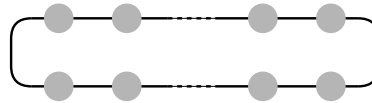
[ Bernd Fritzke, Institut für Neuroinformatik (INI), Ruhr-Universität Bochum ]

<http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html>

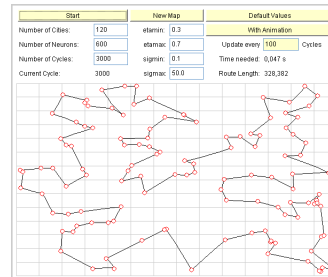
# Self-Organizing Feature Maps (SOMs)

## Anwendungsbeispiel: Traveling Salesman Problem (TSP)

- Aufgabe: Verbinde  $n$  Städte mit einer kürzestmöglichen Rundtour.  
Positionen der Städte auf Landkarte fest, alle Luftlinien zwischen zwei Städten als Wege möglich.
- SOM-Lösungsansatz:  
Verwendung einer geschlossenen Kette von  $n * k$  Neuronen als SOM.



- Positionen der Städte werden gleichverteilt zufällig als Eingabestimuli gezogen.



[Sven Börner, Fakultät Informatik, HTW Dresden]

<http://www.htw-dresden.de/~iwe/Belege/Boerner/>

# Self-Organizing Feature Maps (SOMs)

## Beispiel: Spezialfall Vektorquantisierung

- Leere Nachbarschaftsrelation: Einzelne Neuronen ohne Verbindungen.

$$d_N(N_i, N_j) = \begin{cases} 1 & \text{für } i = j \\ 0 & \text{sonst} \end{cases}$$

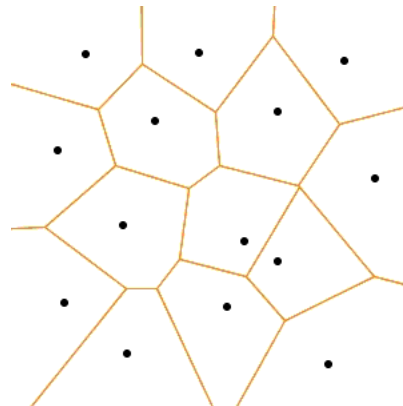
- Anziehung für  $N_{k_0}$ , Abstoßung für alle andere Neuronen.

$$h(1, \sigma) = 1 \quad \text{und} \quad h(0, \sigma) = -1$$

- Die Neuronen repräsentieren Bereiche des Eingaberaumes, die *Voronoi-Regionen*:

$N_k$  repräsentiert  $\{\mathbf{x} | d_X(\mathbf{x}, \mathbf{w}_k) < d_X(\mathbf{x}, \mathbf{w}_{k'}), k' \neq k\}$  (Winner Takes It All).

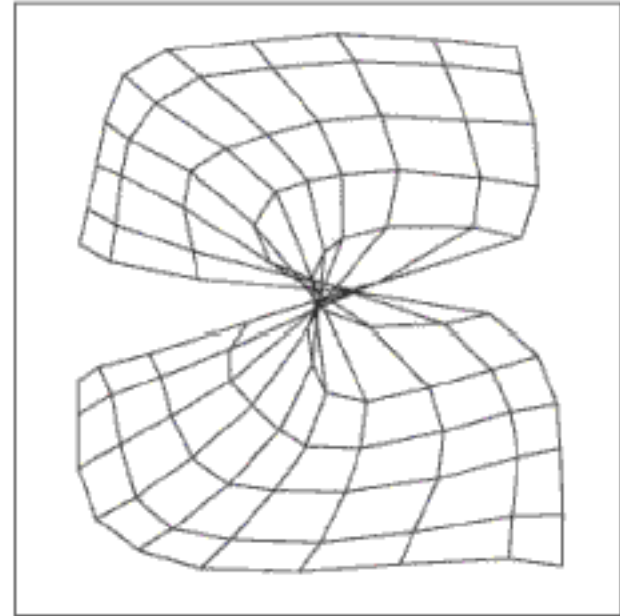
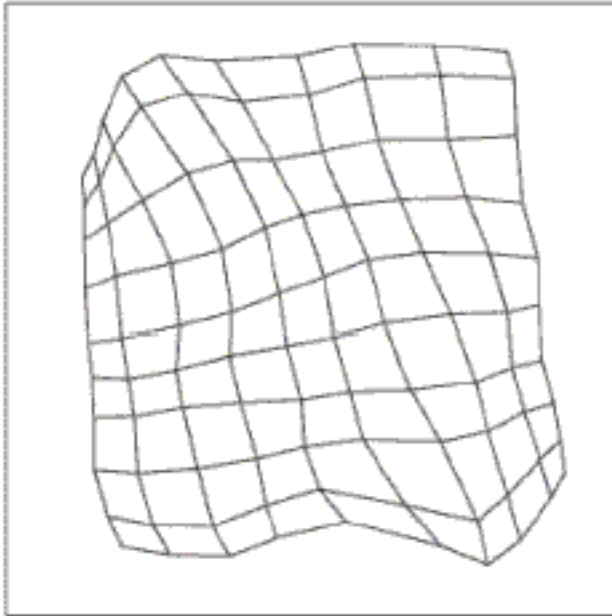
Vektoren des Eingaberaumes, die von  $N_k$  einen geringeren Abstand haben als von allen anderen Neuronen.



# Neuronales Gas (Neural Gas)

## Motivation

- ❑ SOMs bilden realisieren eine räumliche Organisation von hochdimensionales Eingangsdaten.
- ❑ Topologische Defekte



- Topologie SOM passt nicht zur Topologie des Eingaberaumes.
- Nachbarschaft wird nicht adäquat angepasst über  $\sigma(t)$ .
- ❑ Gehirn kann Strukturen selbständig ausbilden und anpassen.

# Neuronales Gas (Neural Gas)

Idee [Martinetz und Schulten, 1991]

- ❑ Abbildung von Eingangssignalen auf Erregungszustände von Neuronen wie beim SOM.
- ❑ Aber: Neuronen weisen keine vorgegebene Nachbarschaft auf.
- ❑ Durch ein Eingangssignal stark erregte Neuronen verbinden sich zu Nachbarschaften.
- ❑ Festigkeit einer Verbindung zwischen Neuronen nimmt mit der Zeit ab, Verbindung müssen immer wieder erneuert werden.  
(Siehe auch Hebbsches Lernen.)
- Struktur des Netzes bildet sich während des Lernvorgangs aus.  
Bessere Anpassung an die Topologie des Merkmalsraumes.
- ❑ Erweiterung des Ansatzes: Growing Neural Gas  
Nicht nur Kanten, sondern auch Knoten können erzeugt und gelöscht werden.

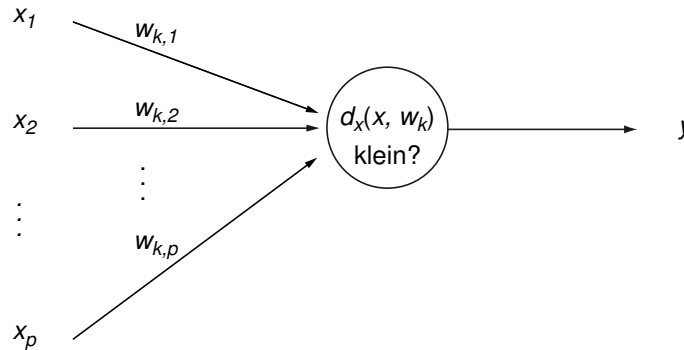
# Neuronales Gas (Neural Gas)

## Formales Modell

- $D := \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq X = \mathbb{R}^p$  endliche Menge von Eingabestimuli (Merkmalsvektoren).
- $d_X : \mathbb{R}^p \times \mathbb{R}^p \rightarrow \mathbb{R}^+$  Metrik auf  $\mathbb{R}^p$ .

Beispiel: Euklidischer Abstand  $d(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^p (x_i - y_i)^2}$

- $N = \{N_1, \dots, N_K\}$  endliche Menge von Neuronen,  $N_k$  definiert durch Gewichtsvektoren  $\mathbf{w}_k \in \mathbb{R}^p$ .



# Neuronales Gas (Neural Gas)

## Formales Modell (Fortsetzung)

$$\square \text{ sort-index} : \{1, \dots, K\} \times \mathbf{R}^K \rightarrow \{1, \dots, K\}$$

Position in der sortierten Reihenfolge der  $K$  Werte für die angegebene Komponente im Ausgangstupel.

Beispiel: Für  $K = 7$  liefert  $\text{sort-index}(2, (7.3, 6.1, 5.34, 1.02, 3.8, 4.21, 2.6)) = 6$ .

6.1 ist der 6.te Wert in dem aufsteigend sortierten Eingabetupel.

$$\square s_k := \text{sort-index}(k, (d_X(\mathbf{x}, \mathbf{w}_1), \dots, d_X(\mathbf{x}, \mathbf{w}_K)))$$

Reihenfolge der Indizes der Neuronen bezüglich des Abstands des Gewichtsvektors zu  $\mathbf{x}$ , beginnend beim kleinsten Abstand, d.h.  $s_k$  ist Position von  $N_i$  in Sortierung von  $N$  bzgl. Abstand von  $\mathbf{x}$ .

(Ersatz für die Nachbarschaftsrelation der Neuronen)



## Bemerkungen:

- Alternative Definition der  $s_k$ :

$$s_k := |\{j \in \{1, \dots, K\} : d_X(\mathbf{x}, \mathbf{w}_j) < d_X(\mathbf{x}, \mathbf{w}_k)\}|$$

(verschiedene Abstände von  $\mathbf{x}$  für alle Neuronen vorausgesetzt).

# Neuronales Gas (Neural Gas)

## Formales Modell (Fortsetzung)

- $h(d, \sigma)$  Nachbarschaftsfunktion zur Realisierung der lateralen Inhibition: maximal für  $d = 0$ , monoton fallend.

Beispiel: Übliche Wahl  $h(d, \sigma) = \exp(-\frac{d}{\sigma})$ .

- $\sigma > 0$  bestimmt die „enge“ Nachbarschaft.

Beispiel:  $\sigma$  kann abhängig von der Runde  $t$  gewählt werden als  $\sigma(t) = \sigma_a \left(\frac{\sigma_e}{\sigma_a}\right)^{\frac{t}{t_{max}}}$  mit Anfangswert  $\sigma_a > 0$  und Endwert  $\sigma_e$  mit  $0 < \sigma_e \leq \sigma_a$ .

- $\eta > 0$  bestimmt die Lernrate (typisch  $\eta \in [0, 1]$ ).

Beispiel:  $\eta$  kann abhängig von der Runde  $t$  gewählt werden als  $\eta(t) = \eta_a \left(\frac{\eta_e}{\eta_a}\right)^{\frac{t}{t_{max}}}$  mit Anfangswert  $\eta_a > 0$  und Endwert  $\eta_e$  mit  $0 < \eta_e \leq \eta_a$ .

- $\Delta \mathbf{w}_k = \eta(t) \cdot h(s_k, \sigma(t)) \cdot (\mathbf{x} - \mathbf{w}_k)$  Anpassung der Gewichte aller Neuronen.

# Neuronales Gas (Neural Gas)

## Formales Modell (Fortsetzung)

- *Connect* Verbindungsmatrix für  $N$ : quadratische  $K \times K$  Matrix.  
 $Connect(k_1, k_2) > 0$  gibt an, dass eine Verbindung zwischen  $N_{k_1}$  und  $N_{k_2}$  besteht und seit wieviel Runden sie besteht.  
 $Connect(k_1, k_2) = 0$  gibt an, dass keine Verbindung zwischen  $N_{k_1}$  und  $N_{k_2}$  existiert.
- $\tau > 0$  bestimmt das Alter, bei dem eine Verbindung in *Connect* gelöscht wird.  
Beispiel:  $\tau$  kann abhängig von der Runde  $t$  gewählt werden als  $\tau(t) = \tau_a \left( \frac{\tau_e}{\tau_a} \right)^{\frac{t}{t_{max}}}$  mit Anfangswert  $\tau_a > 0$  und Endwert  $\tau_e$  mit  $0 < \tau_e \leq \tau_a$ .
- Neue Verbindungen werden zwischen den beiden Neuronen erzeugt, die dem Eingabewert am nächsten liegen.
- Verbindungen, die älter als  $\tau$  sind, werden gelöscht.

# Neuronales Gas (Neural Gas)

## Algorithmus zur Gewichts- und Strukturanpassung

Sei  $D$  eine Menge von Trainingsbeispielen,  $\eta$  eine positive kleine Konstante, die Lernrate,  $\sigma$  eine positive Konstante, der Nachbarschaftsradius, und  $p$  die Dimension der Merkmalsvektoren.

*neural\_gas\_training*( $D, \eta, \sigma$ )

1. *initialize\_random\_weights*( $\mathbf{w}_1, \dots, \mathbf{w}_k$ ),  $t = 0$ ;
2. **REPEAT**
3.  $t = t + 1$ ,  $\mathbf{x} = \text{random\_select}(D)$ ;
4. **FOR**  $k = 1$  **TO**  $K$  **DO**
5.     **FOR**  $i = 0$  **TO**  $p$  **DO**
6.          $\Delta w_{k,i} = \eta \cdot h(s_k, \sigma) \cdot (x_i - w_{k,i})$ ;
7.          $w_{k,i} = w_{k,i} + \Delta w_{k,i}$ ;
8.     **ENDDO**
9. **ENDDO**
10. *Connect*( $s_1, s_2$ ) = 1;
11. **FOREACH**  $(i, j) \in \{1, \dots, K\}^2, i \neq j$  **DO**
12.     **IF** *Connect*( $i, j$ ) > 0 **THEN** *Connect*( $i, j$ ) = *Connect*( $i, j$ ) + 1;
13.     **IF** *Connect*( $i, j$ ) >  $\tau$  **THEN** *Connect*( $i, j$ ) = 0;
14. **ENDDO**
15. **UNTIL** ( $t > t_{\max}$ );

# Neuronales Gas (Neural Gas)

## Algorithmus zur Gewichtsanzpassung

Natürlichsprachliche Formulierung:

1. Initialisiere die Gewichte der Neuronen in der SOM.
2. Wähle zufällig einen Eingabestimulus  $\mathbf{x}$  aus  $D$ .
3. Bestimme die Reihenfolge  $s_1, \dots, s_K$  der Neuronen nach den Abständen von  $\mathbf{x}$ .
4. Passe den Gewichtsvektor des Erregungszentrums und seiner Nachbarschaft mit nach außen abnehmender Stärke an den Eingabestimulus an.

$$\mathbf{w}_k = \mathbf{w}_k + \eta \cdot h(s_k, \sigma) \cdot (\mathbf{x} - \mathbf{w}_k)$$

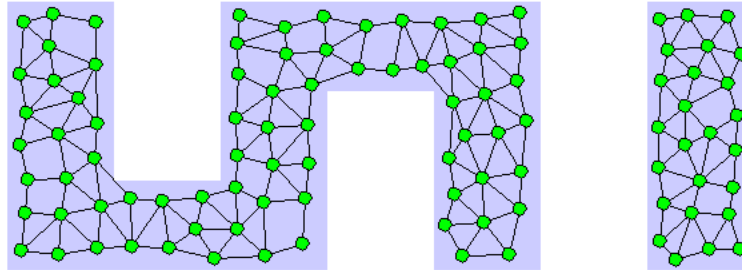
5. Bilde neue Nachbarschaft zwischen den beiden  $\mathbf{x}$  am nächsten liegenden Neuronen.
6. Inkrementiere Alter der Kanten und lösche zu alte Kanten.
7. ( Reduziere Lernrate  $\eta$  und Nachbarschaftsgröße  $\sigma$ , erhöhe  $\tau$ . )
8. Falls Trainingsphase nicht zu Ende, gehe zu Schritt 2.

# Neuronales Gas (Neural Gas)

## Beispiel: Spezialfall Neuronales Gas in 2D

*DemoGNG*, a Java applet, implements several methods related to competitive learning. It is possible to experiment with the methods using various data distributions and observe the learning process. A common terminology is used to make it easy to compare one method to the other.

48100 Network Model: **Neural Gas with CHL** v1.5



100 Nodes

Teach  Signals  Voronoi  Delaunay  
 Error Graph  Nodes  Edges  Random Init  White  Sound

prob. Distrib.: **UNI** Nodes: **100** Display: **50** Speed: **Lightning**

lambda_i	lambda_f	epsilon_i	epsilon_f	t_max	edge_i	edge_f
30.0	0.01	0.3	0.05	40000.0	20	200

Authors: **Hartmut S. Loos Bernd Fritzke**

[Bernd Fritzke, Institut für Neuroinformatik (INI), Ruhr-Universität Bochum]

<http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html>