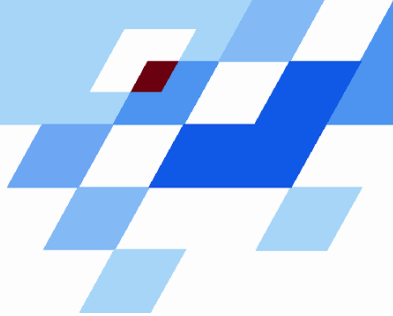


Wintersemester 2006/07

Fundamente der Computational Intelligence

(Vorlesung)

Prof. Dr. Günter Rudolph

Fachbereich Informatik

Lehrstuhl für Algorithm Engineering

Inhalt

- Single-Layer-Lernverfahren
- Multi-Layer-Perceptron

Beschleunigung des Perzeptron-Lernens

Annahme: $x \in \{ 0, 1 \}^n \quad \Rightarrow ||x|| \geq 1$ für alle $x \neq (0, ..., 0)'$

Wenn fehlerhafte Klassifikation, dann $w'x < 0$.

Also ist die Größe des Fehlers gerade $\delta = -w'x > 0$.

$\Rightarrow w_{t+1} = w_t + (\delta + \varepsilon)\, x$ für $\varepsilon > 0$ (klein) berichtigt Fehler in *einem* Schritt, weil

$$
\begin{aligned}
w'_{t+1}x &= (w_t + (\delta + \varepsilon)\, x)'\, x \\
&= \underbrace{w'_t\, x} + (\delta + \varepsilon)\, x'x \\
&= -\delta + \delta\, ||x||^2 + \varepsilon\, ||x||^2 \\
&= \underbrace{\delta\, (||x||^2 - 1)}_{\geq 0} + \underbrace{\varepsilon\, ||x||^2}_{> 0} > 0 \qquad \checkmark
\end{aligned}
$$

Verallgemeinerung

Annahme: $x \in \mathbb{R}^n \quad \Rightarrow \|x\| > 0$ für alle $x \neq (0, ..., 0)'$

wie zuvor: $w_{t+1} = w_t + (\delta + \varepsilon)\, x$ für $\varepsilon > 0$ (klein) und $\delta = - w'_t\, x > 0$

$$\Rightarrow \quad w'_{t+1} x = \underbrace{\delta\, (\|x\|^2 - 1)}_{< 0 \text{ möglich!}} + \underbrace{\varepsilon\, \|x\|^2}_{> 0}$$

Idee: Skalierung der Daten ändert Klassifikationsaufgabe nicht!

Sei $\ell = \min \{ \| x \| : x \in B \} > 0$

Setze $\hat{x} = \frac{x}{\ell} \quad \Rightarrow$ skalierte Beispielmenge $\hat{B}$

$$\Rightarrow \| \hat{x} \| \geq 1 \quad \Rightarrow \quad \| \hat{x} \|^2 - 1 \geq 0 \quad \Rightarrow \quad w'_{t+1}\, \hat{x} > 0 \quad \boxtimes$$

Es existieren zahlreiche Varianten für Perzeptron-Lernverfahren.

Satz: (Duda & Hart 1973)

Gilt für die Gewichtskorrekturregel $w_{t+1} = w_t + \gamma_t x$ (falls $w'_t x < 0$)

1. $\forall t \geq 0 : \gamma_t \geq 0$
2. $\sum_{t=0}^{\infty} \gamma_t = \infty$
3. $\lim_{m \to \infty} \frac{\sum_{t=0}^{m} \gamma_t^2}{\left(\sum_{t=0}^{m} \gamma_t \right)^2} = 0$

dann $w_t \rightarrow w^*$ für $t \rightarrow \infty$ mit $\forall x' w^* > 0$. ■

Bsp: $\gamma_t = \gamma > 0$ oder $\gamma_t = \gamma / (t+1)$ für $\gamma > 0$

Bisher: *Online-Lernen*

→ Aktualisierung der Gewichte nach jedem Trainingsmuster (falls nötig)

Jetzt: *Batch-Lernen*

→ Aktualisierung der Gewichte erst nach Test aller Trainingsmuster

→ Aktualisierungsregel:

$$w_{t+1} = w_t + \gamma \sum_{\substack{w'_t x < 0 \\ x \in B}} x \qquad (\gamma > 0)$$

vage Bewertung in Literatur:

- Vorteil : „i.A. schneller“
- Nachteil : „braucht mehr Speicher“ ← zusätzl. 1 Vektor!

Gewichte finden durch Optimierung

Sei $F(w) = \{ x \in B : w'x < 0 \}$ die Menge der durch w fehlklassifizierten Beispiele.

Zielfunktion: $f(w) = -\sum_{x \in f(w)} w'x \rightarrow \min!$

Optimum: $f(w) = 0$ genau dann, wenn F(w) leer ist

Möglicher Ansatz: *Gradientenverfahren*

$$w_{t+1} = w_t - \gamma \nabla f(w_t) \qquad (\gamma > 0)$$

konvergiert nur zu einem lokalen Minimum (abh. von w_0)

Gradientenverfahren

$w_{t+1} = w_t - \gamma \nabla f(w_t)$

Gradient zeigt in Richtung des lokal stärksten Anstieges der Funktion f(·)

Gradient $\nabla f(w) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \ldots, \frac{\partial f(w)}{\partial w_n} \right)$

$$\frac{\partial f(w)}{\partial w_i} = -\frac{\partial}{\partial w_i} \sum_{x \in F(w)} w'x = -\frac{\partial}{\partial w_i} \sum_{x \in F(w)} \sum_{j=1}^{n} w_j \cdot x_j$$

$$= -\sum_{x \in F(w)} \underbrace{\frac{\partial}{\partial w_i} \left(\sum_{j=1}^{n} w_j \cdot x_j \right)}_{x_i} = -\sum_{x \in F(w)} x_i$$

Achtung:
Indices i bei w_i bezeichnen hier Komponenten des Vektors w; sind also keine Iterationszähler!

Gradientenverfahren

also:

Gradient $\nabla f(w) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, \ldots, \frac{\partial f(w)}{\partial w_n} \right)'$

$$= \left(-\sum_{x \in F(w)} x_1, -\sum_{x \in F(w)} x_2, \ldots, -\sum_{x \in F(w)} x_n \right)'$$

$$= -\sum_{x \in F(w)} x$$

$$\Rightarrow \quad w_{t+1} = w_t + \gamma \sum_{x \in F(w_t)} x$$

Gradientenverfahren $\Leftrightarrow$ Batch-Lernen

Wie schwer ist es

(a) eine trennende Hyperebene zu finden, wenn sie existiert?

(b) zu entscheiden, dass trennende Hyperebene nicht existiert?

Sei $B = P \cup \{ -x : x \in N \}$ (nur positive Beispiele), $w_i \in \mathbb{R}$, $\theta \in \mathbb{R}$, $|B| = m$

Für jedes Beispiel $x_i \in B$ soll gelten:

$x_{i1} w_1 + x_{i2} w_2 + \ldots + x_{in} w_n \geq \theta$ $\quad \rightarrow$ triviale Lösung $w_i = \theta = 0$ ausschließen!

Deshalb zusätzlich: $\eta \in \mathbb{R}$

$x_{i1} w_1 + x_{i2} w_2 + \ldots + x_{in} w_n - \theta - \eta \geq 0$

Idee: η maximieren $\rightarrow$ falls $\eta^* > 0$, dann Lösung gefunden

Matrixschreibweise:

$$A = \begin{pmatrix} x'_1 & -1 & -1 \\ x'_2 & -1 & -1 \\ \vdots & \vdots & \vdots \\ x'_m & -1 & -1 \end{pmatrix} \qquad z = \begin{pmatrix} w \\ \theta \\ \eta \end{pmatrix}$$

Lineares Programmierungsproblem:

$f(z_1, z_2, ..., z_n, z_{n+1}, z_{n+2}) = z_{n+2} \rightarrow$ max!

unter Nebenbedingung $Az \geq 0$

} berechnet z.B. Kamarkar-Algorithmus in **polynomieller Zeit**

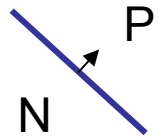
Falls $z_{n+2} = \eta > 0$, dann Gewichte und Schranke durch z gegeben.

Sonst existiert trennende Hyperebene nicht!

Was kann man durch zusätzliche Schichten (Layer) erreichen?

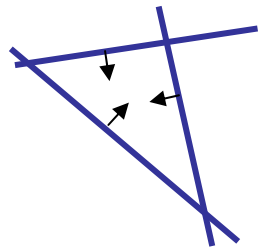
- Single-layer perceptron (SLP)

  ⇒ Hyperfläche separiert Raum in zwei Teilräume



- Two-layer perceptron

  ⇒ beliebige konvexe Mengen unterscheidbar



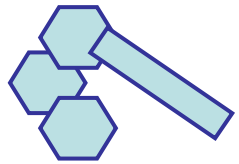
Verknüpfung mit AND in der 2. Schicht

- Three-layer perceptron

  ⇒ beliebige Mengen unterscheidbar (abh. von Anzahl der Neuronen),

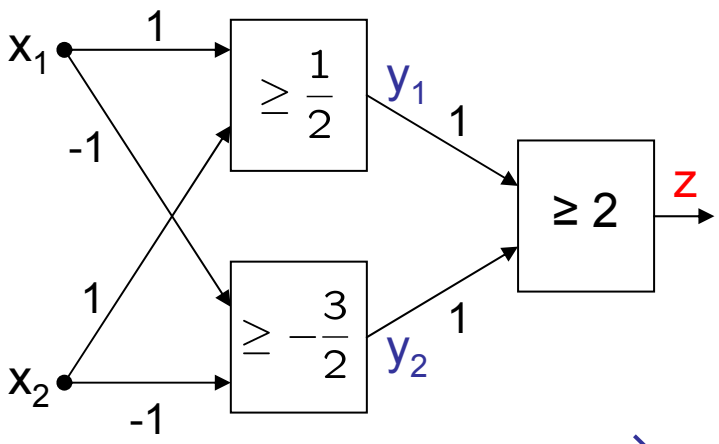
  weil mehrere konvexe Mengen bis 2. Schicht darstellbar,

  diese können in 3. Schicht kombiniert werden

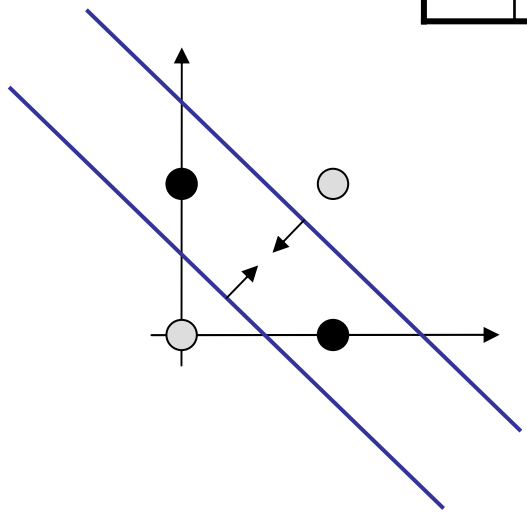


  ⇒ Mehr als 3 Schichten sind nicht nötig!

XOR mit 3 Neuronen in 2 Schichten

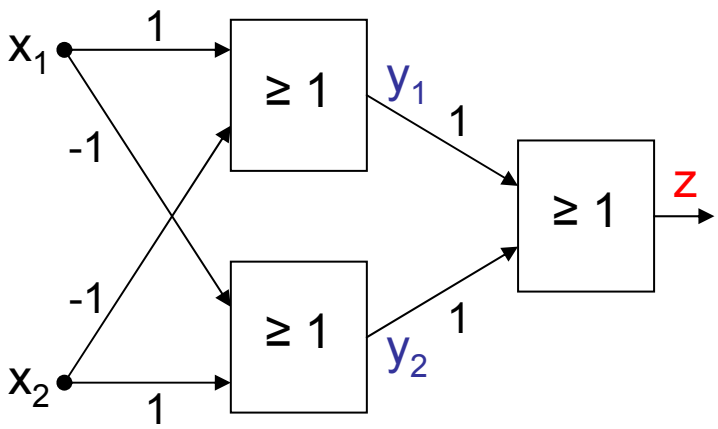


x_1	x_2	y_1	y_2	z
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0



konvexe Menge

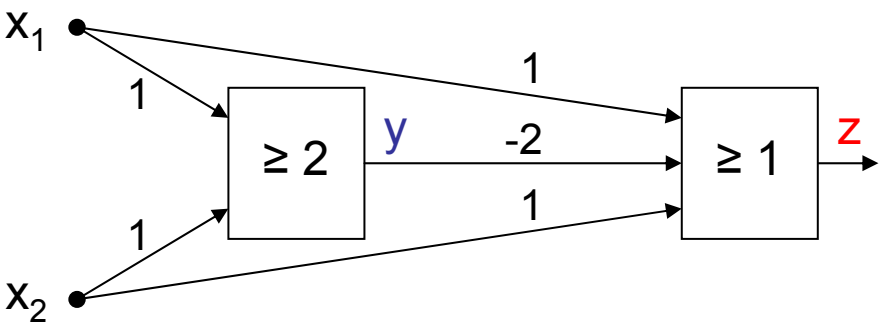
XOR mit 3 Neuronen in 2 Schichten



x_1	x_2	y_1	y_2	z
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

ohne AND-Verknüpfung in 2. Schicht

XOR mit 2 Neuronen möglich

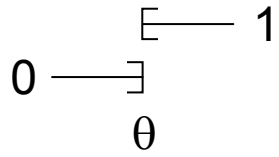


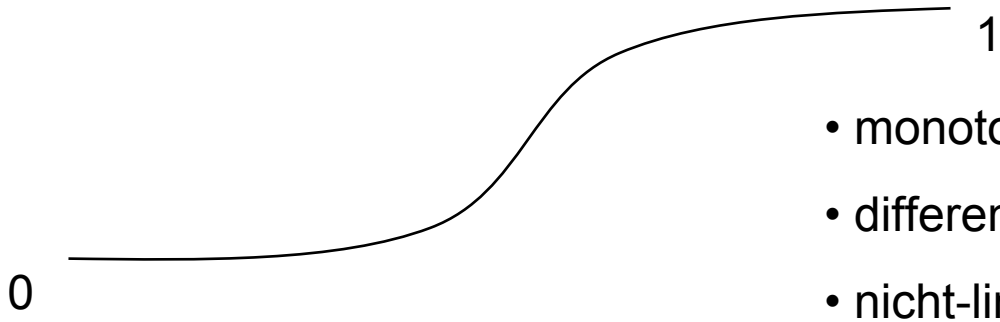
x_1	x_2	y	-2y	x_1-2y+x_2	z
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	-2	0	0

aber: keine Schichtenarchitektur

Lernalgorithmus für Multi-Layer-Perceptron

<u>vorteilhaft:</u> sigmoide Aktivierungsfunktion (statt Signum-Funktion)





- monoton wachsend
- differenzierbar
- nicht-linear
- Ausgabe $\in$ [0,1] statt $\in$ { 0, 1 }
- Schranke θ in Aktivierungs-funktion integriert

Bsp:

- $a(x) = \dfrac{1}{1+e^{-x}} \qquad a'(x) = a(x)(1-a(x))$
- $a(x) = \tanh(x) \qquad a'(x) = (1-a^2(x))$

} Werte für Ableitungen direkt aus Funktionswerten bestimmbar

Quantifizierung des Klassifikationsfehlers beim MLP

- Total Sum Squared Error (TSSE)

$$f(w) = \sum_{x \in B} \| \underbrace{g(w;x)}_{\text{Ausgabe des Netzes für Gewichte w und Eingabe x}} - \underbrace{g^*(x)}_{\text{Soll-Ausgabe des Netzes für Eingabe x}} \|^2$$

- Total Mean Squared Error (TMSE)

$$f(w) = \frac{1}{|B| \cdot \ell} \sum_{x \in B} \| g(w;x) - g^*(x) \|^2 = \underbrace{\frac{1}{|B| \cdot \ell}}_{const.} \cdot \text{TSSE}$$

Anzahl der Beispiele

Anzahl der Ausgabeneuronen

const. ⇒ führt zur gleichen Lösung wie TSSE

Lernalgorithmus für Multi-Layer-Perceptron

<u>Gradientenverfahren</u>

$f(w_t, u_t) = TSSE$

$$u_{t+1} = u_t - \gamma \nabla_u f(w_t, u_t)$$
$$w_{t+1} = w_t - \gamma \nabla_w f(w_t, u_t)$$