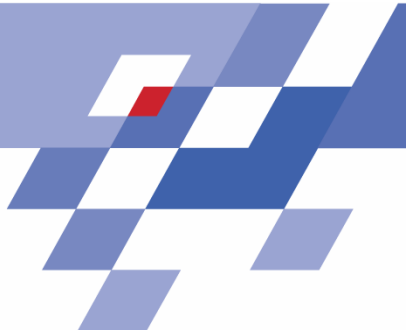
TECHNISCHE UNIVERSITÄT DORTMUND



Wintersemester 2007/08

Praktische Optimierung

(Vorlesung)

Prof. Dr. Günter Rudolph

Fakultät für Informatik

Lehrstuhl für Algorithm Engineering

Inhalt

- Multilayer-Perceptron (MLP)
- Radiale Basisfunktionsnetze (RBF)
- Kriging

} Neuronale Netze

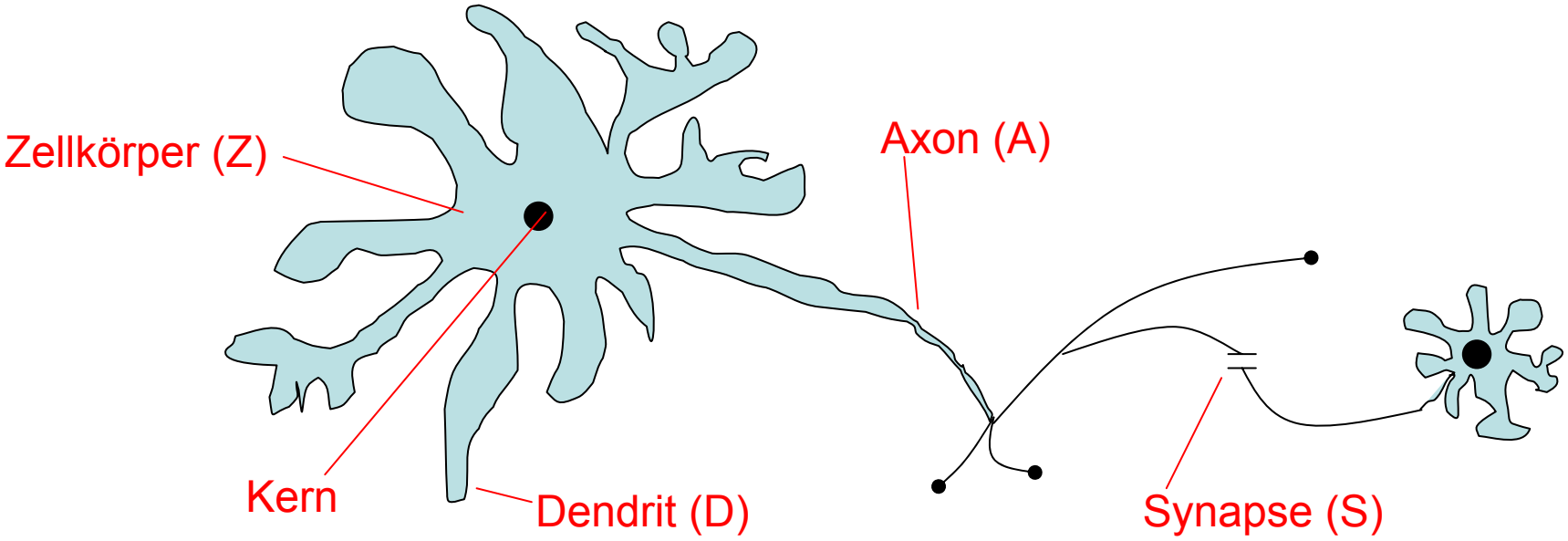
Biologisches Vorbild

- Neuronen
  - Information aufnehmen (D)
  - Information verarbeiten (Z)
  - Information weiterleiten (A / S)

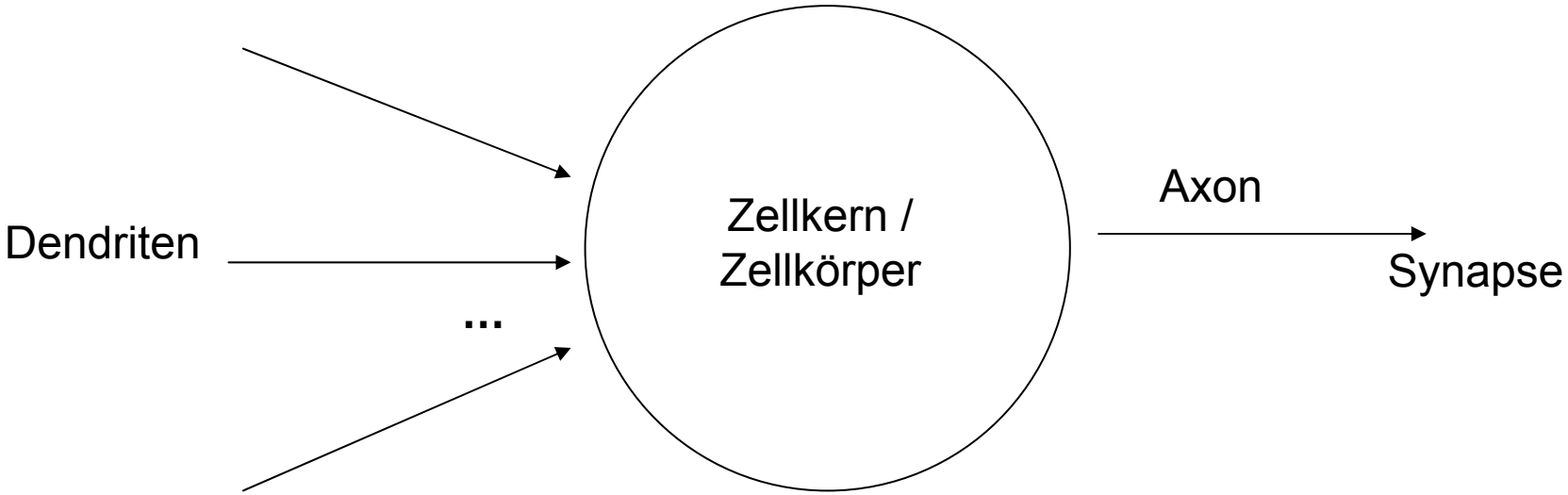
Mensch: 10^{12} Neuronen

Strom im mV-Bereich

120 m / s



Abstraktion

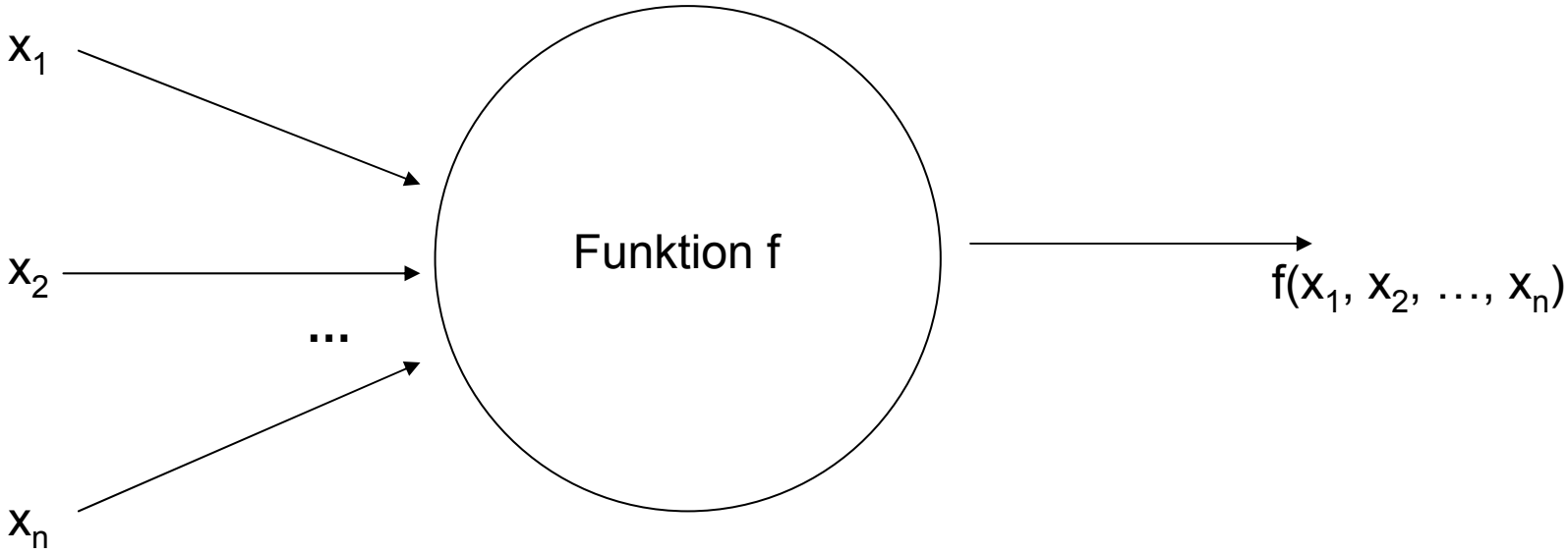


Signal-Eingabe

Signal-Verarbeitung

Signal-Ausgabe

Modell



McCulloch-Pitts-Neuron 1943:

$x_i \in \{ 0, 1 \} =: \mathbb{B}$

$f: \mathbb{B}^n \rightarrow \mathbb{B}$

Perzeptron (Rosenblatt 1958)

→ komplexes Modell → reduziert von Minsky & Papert auf das „Notwendigste“

→ Minsky-Papert-Perzeptron (MPP), 1969

Was leistet <u>ein</u> MPP?

$$w_1 x_1 + w_2 x_2 \geq \theta$$

J → 1

N → 0

umstellen nach x_2 liefert:

$$x_2 \geq \frac{\theta}{w_2} - \frac{w_1}{w_2} x_1$$

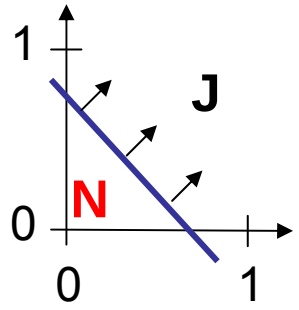
J → 1

N → 0

Bsp:

$$0,9\, x_1 + 0,8\, x_2 \geq 0,6$$

$$\Leftrightarrow \quad x_2 \geq \frac{3}{4} - \frac{9}{8} x_1$$



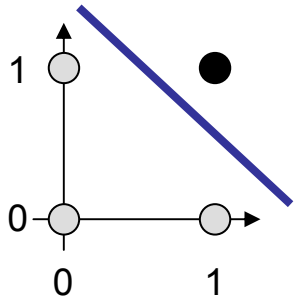
Trenngerade

<u>separiert</u> $\mathbb{R}^2$

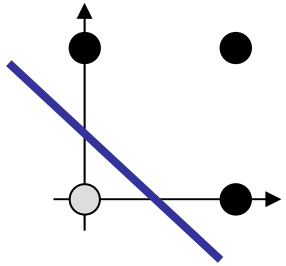
in 2 Klassen

○ = 0 ● = 1

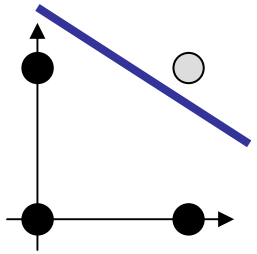
AND



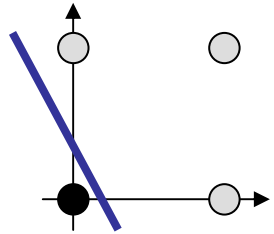
OR



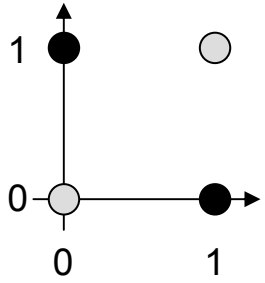
NAND



NOR



XOR



?

x_1	x_2	xor
0	0	0
0	1	1
1	0	1
1	1	0

$\Rightarrow 0 < \theta$

$\Rightarrow w_2 \geq \theta$

$\Rightarrow w_1 \geq \theta$

$\Rightarrow w_1 + w_2 < \theta$

$w_1, w_2 \geq \theta > 0$

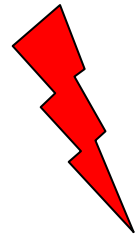
$\Rightarrow w_1 + w_2 \geq 2\theta$

Widerspruch!

$w_1 x_1 + w_2 x_2 \geq \theta$

1969: Marvin Minsky / Seymor Papert

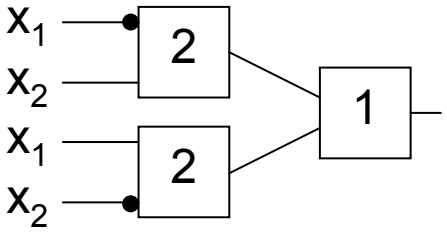
- Buch *Perceptrons* → Analyse math. Eigenschaften von Perzeptrons
- Ernüchterndes Ergebnis:
  Triviale Probleme können nicht mit Perzeptrons gelöst werden!
  - XOR-Problem
  - Parity-Problem
  - Connectivity-Problem
- „Folgerung“: Alle künstliche Neuronen haben diese Schwäche!
  ⇒ Forschung auf diesem Gebiet ist wissenschaftliche Sackgasse!



- Folge: Forschungsförderung bzgl. KNN praktisch eingestellt (~ 15 Jahre)

Wege aus der „Sackgasse“:

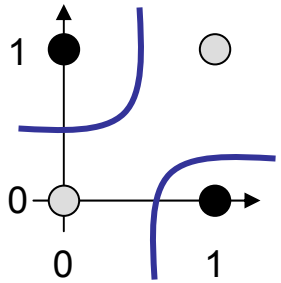
1. Mehrschichtige Perzeptrons:



$\Rightarrow$ realisiert XOR

2. Nichtlineare Trennfunktionen:

XOR $g(x_1, x_2) = 2x_1 + 2x_2 - 4x_1x_2 - 1$ mit $\theta = 0$



$g(0,0) = -1$
$g(0,1) = +1$
$g(1,0) = +1$
$g(1,1) = -1$

Wie kommt man zu den Gewichten und θ ?

bisher: durch Konstruktion

Bsp: NAND-Gatter

x_1	x_2	NAND
0	0	1
0	1	1
1	0	1
1	1	0

$\Rightarrow 0 \geq \theta$

$\Rightarrow w_2 \geq \theta$

$\Rightarrow w_1 \geq \theta$

$\Rightarrow w_1 + w_2 < \theta$

erfordert Lösung eines linearen Ungleichungssystems ($\in$ P)

(Bsp: $w_1 = w_2 = -2$, $\theta = -3$)

jetzt: durch „Lernen“ bzw. Trainieren

Perceptron-Lernen

Annahme: Testbeispiele mit richtigem Ein-/Ausgabeverhalten bekannt

Prinzip:

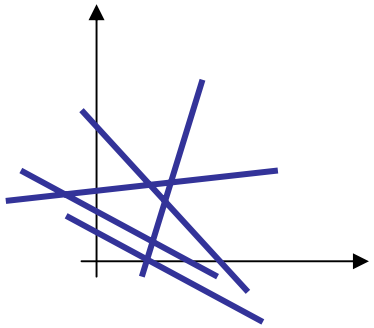
(1) wähle Gewichte irgendwie

(2) lege Testmuster an

(3) falls Perceptronausgabe falsch, dann verändere Gewichte

(4) gehe nach (2) bis richtige Perceptronausgabe für alle Testmuster

grafisch:

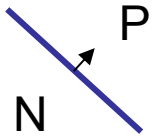


→ Verschieben und Drehen der Trenngeraden

Was kann man durch zusätzliche Schichten (Layer) erreichen?

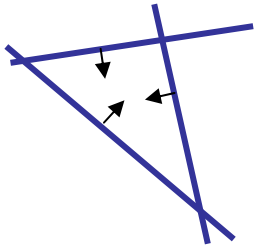
- Single-layer perceptron (SLP)

  ⇒ Hyperfläche separiert Raum in zwei Teilräume



- Two-layer perceptron

  ⇒ beliebige konvexe Mengen unterscheidbar



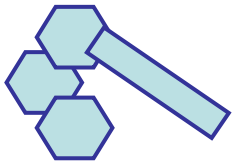
Verknüpfung mit AND in der 2. Schicht

- Three-layer perceptron

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

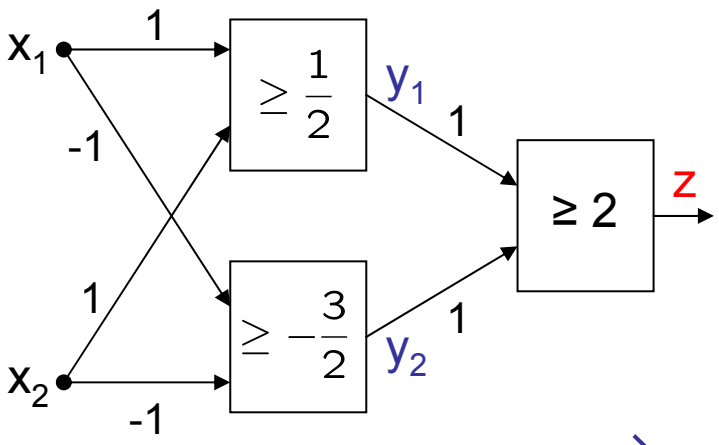
  weil mehrere konvexe Mengen bis 2. Schicht darstellbar,

  diese können in 3. Schicht kombiniert werden

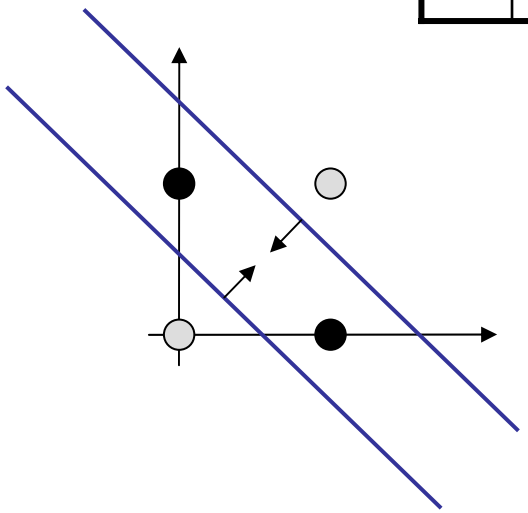


  ⇒ Mehr als 3 Schichten sind nicht nötig!

XOR mit 3 Neuronen in 2 Schichten

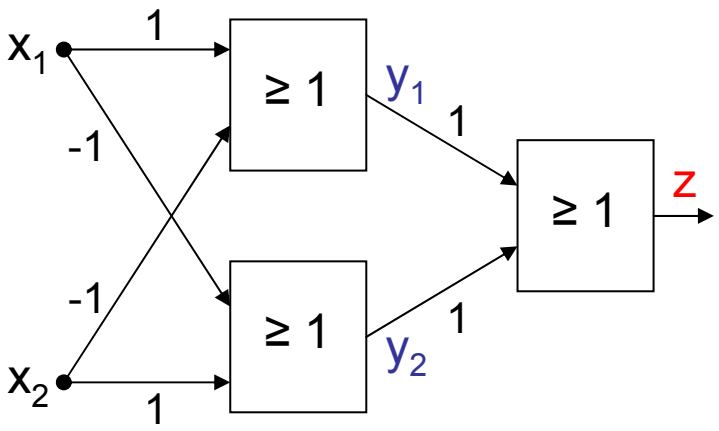


x_1	x_2	y_1	y_2	z
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0



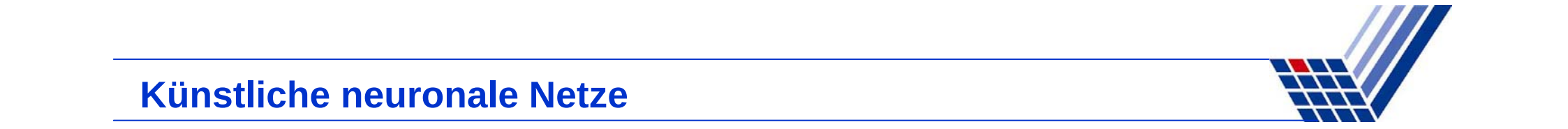
konvexe Menge

XOR mit 3 Neuronen in 2 Schichten

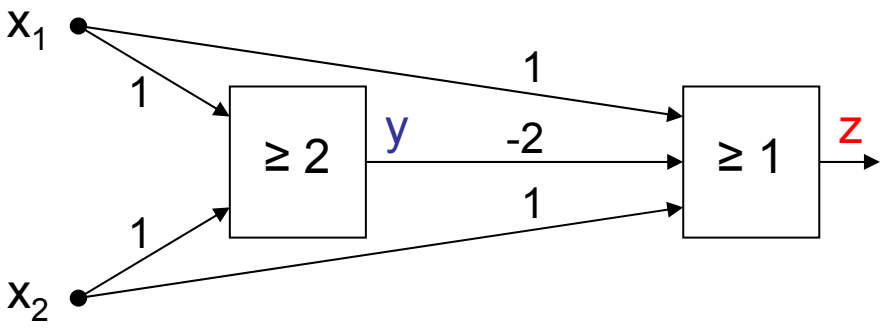


x_1	x_2	y_1	y_2	z
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

ohne AND-Verknüpfung in 2. Schicht



XOR mit 2 Neuronen möglich

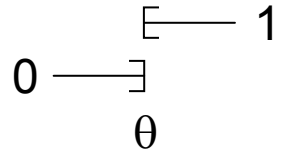


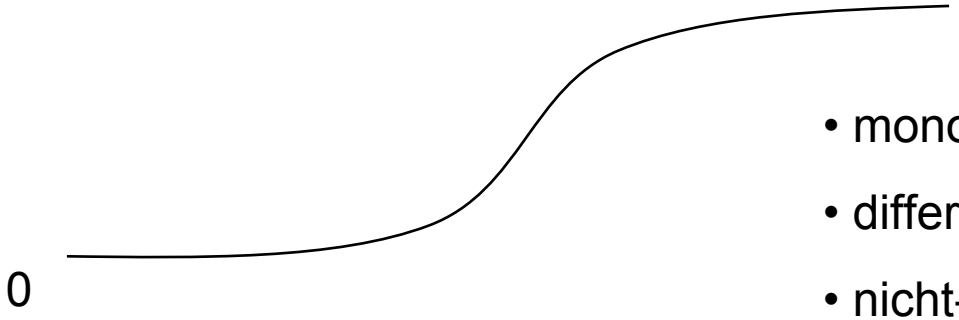
x_1	x_2	y	-2y	x_1-2y+x_2	z
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	-2	0	0

aber: keine Schichtenarchitektur

Lernalgorithmus für Multi-Layer-Perceptron

vorteilhaft: 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

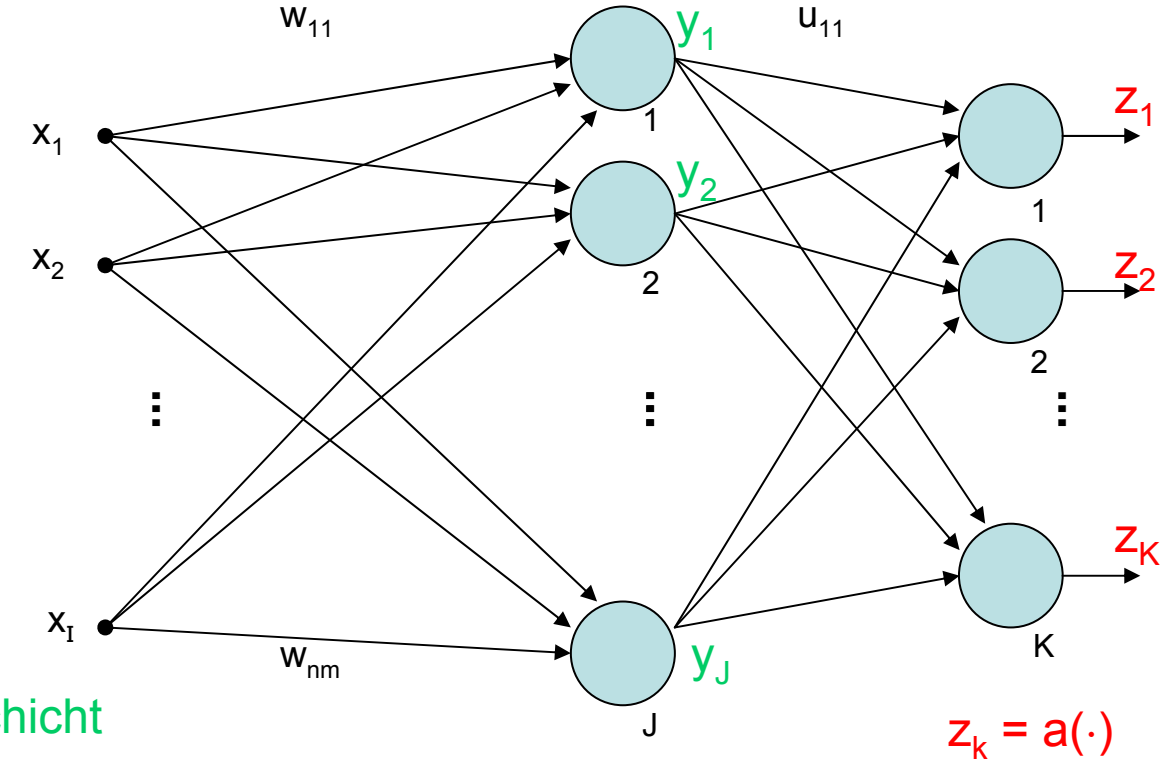
Lernalgorithmus für Multi-Layer-Perceptrons

Gradientenverfahren

$f(w_t, u_t) = TSSE$

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

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



x_i : Eingabe an Eingabeschicht

y_j : Ausgabe der verdeckten Schicht

z_k: Ausgabe der Ausgabeschicht

$z_k = a(\cdot)$

$y_j = h(\cdot)$

$$y_j = h\left(\sum_{i=1}^{I} w_{ij} \cdot x_i\right) = h(w_j' x)$$

Ausgabe von Neuron j in der verdeckten Schicht

$$z_k = a\left(\sum_{j=1}^{J} u_{jk} \cdot y_j\right) = a(u_k' y)$$

Ausgabe von Neuron k in der Ausgabeschicht

$$= a\left(\sum_{j=1}^{J} u_{jk} \cdot h\left(\sum_{i=1}^{I} w_{ij} \cdot x_i\right)\right)$$

Fehler bei Eingabe x:

$$f(w,u;x) = \sum_{k=1}^{K} (z_k(x) - z_k^*(x))^2 = \sum_{k=1}^{K} (z_k - z_k^*)^2$$

Netzausgabe — Sollausgabe bei Eingabe x

Fehler bei Eingabe x und Sollausgabe z*:

$$f(w,u;x,z^*) = \sum_{k=1}^{K} \Bigg[\underbrace{a\Bigg(\sum_{j=1}^{J} u_{jk} \cdot \underbrace{h\Bigg(\overbrace{\sum_{i=1}^{I} w_{ij} \cdot x_i}^{w_j' x} \Bigg)}_{y_j} \Bigg)}_{z_k} - z_k^*(x) \Bigg]^2$$

Gesamtfehler für alle Beispiele (x, z*) ∈ B:

$$f(w,u) = \sum_{(x,z^*)\in B} f(w,u;x,z^*) \qquad \text{(TSSE)}$$

Gradient des Gesamtfehlers:

$$\nabla f(w,u) = \sum_{(x,z^*)\in B} \nabla f(w,u;x,z^*)$$

Vektor der partiellen Ableitungen nach den Gewichten u_{jk} und w_{ij}

also:

$$\frac{\partial f(w,u)}{\partial u_{jk}} = \sum_{(x,z^*)\in B} \frac{\partial f(w,u;x,z^*)}{\partial u_{jk}}$$

bzw.

$$\frac{\partial f(w,u)}{\partial w_{ij}} = \sum_{(x,z^*)\in B} \frac{\partial f(w,u;x,z^*)}{\partial w_{ij}}$$

Annahme: $a(x) = \dfrac{1}{1+e^{-x}} \quad \Rightarrow \quad \dfrac{d\,a(x)}{dx} = a'(x) = a(x) \cdot (1 - a(x))$

und: $h(x) = a(x)$

Kettenregel der Differentialrechnung:

$$[\,p(q(x))\,]' = \underbrace{p'(q(x))}_{\text{äußere Ableitung}} \cdot \underbrace{q'(x)}_{\text{innere Ableitung}}$$

$$f(w,u;x,z^*) \;=\; \sum_{k=1}^{K} [\, a(u_k' y) - z_k^* \,]^2$$

partielle Ableitung nach u_{jk}:

$$\frac{\partial f(w,u;x,z^*)}{\partial u_{jk}} \;=\; 2\,[\, a(u_k' y) - z_k^* \,] \cdot a'(u_k' y) \cdot y_j$$

$$=\; 2\,[\, a(u_k' y) - z_k^* \,] \cdot a(u_k' y) \cdot (1 - a(u_k' y)) \cdot y_j$$

$$=\; \underbrace{2\,[\, z_k - z_k^* \,] \cdot z_k \cdot (1 - z_k)}_{\text{„Fehlersignal“ } \delta_k} \cdot y_j$$

partielle Ableitung nach w_{ij}:

$$\frac{\partial f(w,u;x,z^*)}{\partial w_{ij}} = 2\sum_{k=1}^{K}[\underbrace{a(u_k'y)}_{z_k} - z_k^*]\cdot\underbrace{a'(u_k'y)}_{z_k(1-z_k)}\cdot u_{jk}\cdot\underbrace{h'(w_j'x)}_{y_j(1-y_j)}\cdot x_i$$

$$= 2\cdot\sum_{k=1}^{K}[z_k - z_k^*]\cdot z_k\cdot(1-z_k)\cdot u_{jk}\cdot y_j(1-y_j)\cdot x_i$$

Faktoren umordnen

$$= x_i\cdot y_j\cdot(1-y_j)\cdot\sum_{k=1}^{K}\underbrace{2\cdot[z_k - z_k^*]\cdot z_k\cdot(1-z_k)}_{\text{Fehlersignal } \delta_k \text{ aus vorheriger Schicht}}\cdot u_{jk}$$

Fehlersignal δ_j aus „aktueller“ Schicht

Verallgemeinerung

Das neuronale Netz habe L Schichten (*layer*) S_1, S_2, ... S_L.

Seien Neuronen aller Schichten durchnummeriert von 1 bis N.

$j \in S_m \rightarrow$ Neuron j ist in m-ter Schicht

Alle Gewichte w_{ij} sind in Gewichtsmatrix W zusammengefasst.

Sei o_j Ausgabe (*output*) von Neuron j.

Fehlersignal:

$$\delta_j = \begin{cases} o_j \cdot (1 - o_j) \cdot (o_j - z_j^*) & \text{falls } j \in S_L \text{ (Ausgabeneuron)} \\ o_j \cdot (1 - o_j) \cdot \sum_{k \in S_{m+1}} \delta_k \cdot w_{jk} & \text{falls } j \in S_m \text{ und } m < L \end{cases}$$

Korrektur:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \gamma \cdot o_i \cdot \delta_j$$

<u>beim Online-Lernen</u>:
Korrektur nach **jedem** präsentierten Beispiel

Fehlersignal eines Neurons einer inneren Schicht bestimmt durch

- Fehlersignale aller Neuronen der nachfolgenden Schicht und
- zugehörige Verbindungsgewichte.

$\Downarrow$

- Erst Fehlersignale der Ausgabeneuronen bestimmen,
- daraus Fehlersignale der Neuronen der vorhergehenden Schicht berechnen,
- daraus Fehlersignale der Neuronen der vorhergehenden Schicht berechnen,
- usw. bis zur ersten inneren Schicht.

$\Downarrow$

Fehler wird also von Ausgabeschicht zur ersten inneren Schicht zurückgeleitet.

$\Rightarrow$ **Backpropagation** (of error)

Satz:

MLPs mit einer verdeckten Schicht sigmoidaler Einheiten sind universelle Approximatoren für stetige Funktionen.

Beweis:

Hornik, K., Stinchcombe, M., and White, H. (1989).
"Multilayer Feedforward Networks are Universal Approximators,"
Neural Networks, 2(5), 359-366.

Folgt im Grunde aus dem Satz von Weierstraß.

Netz explizit hinschreiben und ausmultiplizieren.

Sigmoidale Funktionen durch ihre Reihenentwicklung (Polynome!) ersetzen.

Ausmultiplizieren → Polynom als Ersatzzielfunktion